Metaheuristic Optimization

11. Difficulties

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

website

Why is optimization difficult?

- So far, we have discussed several different optimization methods.

Why is optimization difficult?

- So far, we have discussed several different optimization methods.
- You now have some experience in solving optimization problems.

Why is optimization difficult?

- So far, we have discussed several different optimization methods.
- You now have some experience in solving optimization problems.
- You may have seen that for some problems, we can find the best solution rather easily.

Why is optimization difficult?

- So far, we have discussed several different optimization methods.
- You now have some experience in solving optimization problems.
- You may have seen that for some problems, we can find the best solution rather easily.
- For other problems, the solutions that we get seem to be rather bad.

Why is optimization difficult?

- So far, we have discussed several different optimization methods.
- You now have some experience in solving optimization problems.
- You may have seen that for some problems, we can find the best solution rather easily.
- For other problems, the solutions that we get seem to be rather bad.
- Why?

Why is optimization difficult?

- So far, we have discussed several different optimization methods.
- You now have some experience in solving optimization problems.
- You may have seen that for some problems, we can find the best solution rather easily.
- For other problems, the solutions that we get seem to be rather bad.
- Why?
- What makes optimization difficult?

Why is optimization difficult?

- Some things are pretty clear

Why is optimization difficult?

- Some things are pretty clear:
  - For some problems (e.g., the finding shortest paths in a graph) there are deterministic algorithms that can give the optimal solutions quickly.

- Some things are pretty clear:
  - For some problems (e.g., the finding shortest paths in a graph) there are deterministic algorithms that can give the optimal solutions quickly.
  - For other problems, either no such method exists or it is not feasible (too slow).

- Some things are pretty clear:
  - For some problems (e.g., the finding shortest paths in a graph) there are deterministic algorithms that can give the optimal solutions quickly.
  - For other problems, either no such method exists or it is not feasible (too slow).
  - We have looked on some of these problems (Partitioning, TSP, ...)

Why is optimization difficult?

- Some things are pretty clear:
  - For some problems (e.g., the finding shortest paths in a graph) there are deterministic algorithms that can give the optimal solutions quickly.
  - For other problems, either no such method exists or it is not feasible (too slow).
  - We have looked on some of these problems (Partitioning, TSP, ...)
  - And used different algorithms to solve them

Why is optimization difficult?

- Some things are pretty clear:
  - For some problems (e.g., the finding shortest paths in a graph) there are deterministic algorithms that can give the optimal solutions quickly.
  - For other problems, either no such method exists or it is not feasible (too slow).
  - We have looked on some of these problems (Partitioning, TSP, ...)
  - And used different algorithms to solve them
  - Some algorithms worked well, some worked not so well.

Why is optimization difficult?

- Some things are pretty clear:
  - For some problems (e.g., the finding shortest paths in a graph) there are deterministic algorithms that can give the optimal solutions quickly.
  - For other problems, either no such method exists or it is not feasible (too slow).
  - We have looked on some of these problems (Partitioning, TSP, ...)
  - And used different algorithms to solve them
  - Some algorithms worked well, some worked not so well.
  - The solution quality we get depends on the algorithm we use.

Why is optimization difficult?

- Some things are pretty clear:
  - For some problems (e.g., the finding shortest paths in a graph) there are deterministic algorithms that can give the optimal solutions quickly.
  - For other problems, either no such method exists or it is not feasible (too slow).
  - We have looked on some of these problems (Partitioning, TSP, . . .)
  - And used different algorithms to solve them
  - Some algorithms worked well, some worked not so well.
  - The solution quality we get depends on the algorithm we use.
  - Different algorithms deal with the difficulties of optimization tasks differently.

Why is optimization difficult?

- Some things are pretty clear:
  - For some problems (e.g., the finding shortest paths in a graph) there are deterministic algorithms that can give the optimal solutions quickly.
  - For other problems, either no such method exists or it is not feasible (too slow).
  - We have looked on some of these problems (Partitioning, TSP, ...)
  - And used different algorithms to solve them
  - Some algorithms worked well, some worked not so well.
  - The solution quality we get depends on the algorithm we use.
  - Different algorithms deal with the difficulties of optimization tasks differently.
  - There may be different aspects that render a problem difficult.

Why is optimization difficult?

- So let us discuss these topics step-by-step

- So let us discuss these topics step-by-step
- We first look into what makes a problem *hard* in the traditional sense, for deterministic algorithms.

- So let us discuss these topics step-by-step
- We first look into what makes a problem *hard* in the traditional sense, for deterministic algorithms.
- Then we discuss what good and bad solutions are.

- So let us discuss these topics step-by-step
- We first look into what makes a problem *hard* in the traditional sense, for deterministic algorithms.
- Then we discuss what good and bad solutions are.
- Problems where we can only get bad solutions are difficult.

Why is optimization difficult?

- So let us discuss these topics step-by-step
- We first look into what makes a problem *hard* in the traditional sense, for deterministic algorithms.
- Then we discuss what good and bad solutions are.
- Problems where we can only get bad solutions are difficult.
- So we discuss different features that make them difficult.

Why is optimization difficult?

- So let us discuss these topics step-by-step
- We first look into what makes a problem *hard* in the traditional sense, for deterministic algorithms.
- Then we discuss what good and bad solutions are.
- Problems where we can only get bad solutions are difficult.
- So we discuss different features that make them difficult.
- And countermeasures!

Why is optimization difficult?

- So let us discuss these topics step-by-step
- We first look into what makes a problem *hard* in the traditional sense, for deterministic algorithms.
- Then we discuss what good and bad solutions are.
- Problems where we can only get bad solutions are difficult.
- So we discuss different features that make them difficult.
- And countermeasures!
- A good summary on this topic given in our recent article *"Evolutionary Optimization: Pitfalls and Booby Traps"* [1].

Section Outline

Algorithmic Complexity

- If we have a deterministic algorithm that finds the exact solutions for a family problems. . .

Algorithmic Complexity

- If we have a deterministic algorithm that finds the exact solutions for a family problems...
- we want to know

Algorithmic Complexity

- If we have a deterministic algorithm that finds the exact solutions for a family problems...
- we want to know:
    - How much time will it need?

Algorithmic Complexity

- If we have a deterministic algorithm that finds the exact solutions for a family problems. . .
- we want to know:
  - How much time will it need?
  - How much memory will it consume?

Algorithmic Complexity

- If we have a deterministic algorithm that finds the exact solutions for a family problems...
- we want to know:
  - How much time will it need?
  - How much memory will it consume?
- This usually depends (amongst other things) on the input size

Algorithmic Complexity

- If we have a deterministic algorithm that finds the exact solutions for a family problems. . .
- we want to know:
  - How much time will it need?
  - How much memory will it consume?
- This usually depends (amongst other things) on the input size:
  - How long it takes to solve a TSP depends on number of cities

Algorithmic Complexity

- If we have a deterministic algorithm that finds the exact solutions for a family problems...
- we want to know:
  - How much time will it need?
  - How much memory will it consume?
- This usually depends (amongst other things) on the input size:
  - How long it takes to solve a TSP depends on number of cities
  - Determine whether a number is prime or not – space/time depends on scale of number

Algorithmic Complexity

- If we have a deterministic algorithm that finds the exact solutions for a family problems...
- we want to know:
    - How much time will it need?
    - How much memory will it consume?
- This usually depends (amongst other things) on the input size:
    - How long it takes to solve a TSP depends on number of cities
    - Determine whether a number is prime or not – space/time depends on scale of number
- Of course, the time/space needed does not just depend on the input size

- If we have a deterministic algorithm that finds the exact solutions for a family problems. . .
- we want to know:
  - How much time will it need?
  - How much memory will it consume?
- This usually depends (amongst other things) on the input size:
  - How long it takes to solve a TSP depends on number of cities
  - Determine whether a number is prime or not – space/time depends on scale of number
- Of course, the time/space needed does not just depend on the input size: It is easy to see that there may be hard and easy TSPs with the same number of cities. . .
- For a given input size, there might be a best, worst, and average case scenarios.

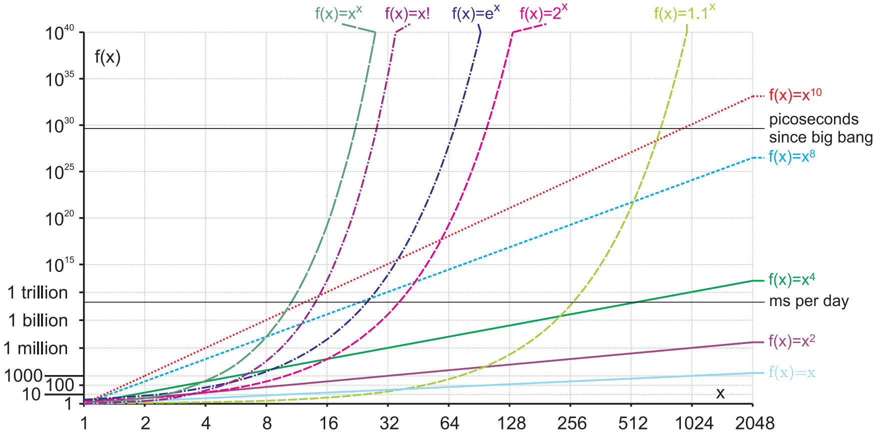
- If we have a deterministic algorithm that finds the exact solutions for a family problems. . .
- we want to know:
  - How much time will it need?
  - How much memory will it consume?
- This usually depends (amongst other things) on the input size:
  - How long it takes to solve a TSP depends on number of cities
  - Determine whether a number is prime or not – space/time depends on scale of number
- Of course, the time/space needed does not just depend on the input size: It is easy to see that there may be hard and easy TSPs with the same number of cities. . .
- For a given input size, there might be a best, worst, and average case scenarios.
- Under algorithmic complexity we can thus understand the average, minimum, or maximum time/space an algorithm needs to finish, as function on the input size.

- There are different mathematical functions

Types of Functions

- There are different mathematical functions
- Some grow fast, some grow slow with the input size x

- There are different mathematical functions
- Some grow fast, some grow slow with the input size x



- Classify the growth speed of functions with an asymptotic notation

Types of Functions

- Classify the growth speed of functions with an asymptotic notation
- Big-$\mathcal{O}$ Notation [2–4]

Types of Functions

- Classify the growth speed of functions with an asymptotic notation
- Big-$\mathcal{O}$ Notation [2–4]

$$f(x) \in \mathcal{O}(g(x)) \Leftrightarrow \exists x_0 \in \mathbb{R}, m \in \mathbb{R}^+ : |f(x)| \leq m|g(x)| \; \forall x > x_0 \tag{1}$$

- We have discussed how long an algorithm needs to exactly solve a problem

Problem Hardness

- We have discussed how long an algorithm needs to exactly solve a problem
- Let's make a statement about the problem in general...

- We have discussed how long an algorithm needs to exactly solve a problem
- Let's make a statement about the problem in general...
- Problem Hardness

Problem Hardness

- We have discussed how long an algorithm needs to exactly solve a problem
- Let's make a statement about the problem in general...
- Problem Hardness:
  - time/resources required at least to solve a problem

Problem Hardness

- We have discussed how long an algorithm needs to exactly solve a problem
- Let's make a statement about the problem in general...
- Problem Hardness:
  - time/resources required at least to solve a problem
  - based on best exact algorithm known for the problem or theoretical bounds

Problem Hardness

- We have discussed how long an algorithm needs to exactly solve a problem
- Let's make a statement about the problem in general...
- Problem Hardness:
    - time/resources required at least to solve a problem
    - based on best exact algorithm known for the problem or theoretical bounds
    - depends on the machine solving it

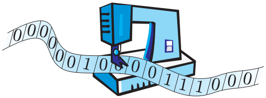
Problem Hardness

- We have discussed how long an algorithm needs to exactly solve a problem
- Let's make a statement about the problem in general…
- Problem Hardness:
  - time/resources required at least to solve a problem
  - based on best exact algorithm known for the problem or theoretical bounds
  - depends on the machine solving it
- If we are conservative, we consider the worst case scenarios

Problem Hardness

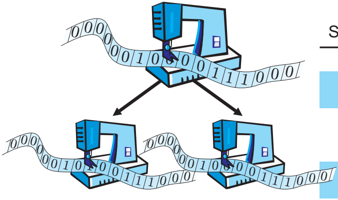
- We have discussed how long an algorithm needs to exactly solve a problem
- Let's make a statement about the problem in general...
- Problem Hardness:
  - time/resources required at least to solve a problem
  - based on best exact algorithm known for the problem or theoretical bounds
  - depends on the machine solving it
- If we are conservative, we consider the worst case scenarios, since we cannot really know what problem we will exactly get in practice.
- Can we make this even more general?

- Deterministic Turing Machine (DTM) [5]



State	Tape Symbol	⇒	Print	Motion	Next State
A	0	⇒	1	Right	A
A	1	⇒	0	Right	B
B	0	⇒	1	Right	B
B	1	⇒	1	Left	C
C	1	⇒	0	Left	C
C	0	⇒	1	Right	A

- Non-Deterministic Turing Machine (NTM)



State	Tape Symbol	⇒	Print	Motion	Next State
A	0	⇒	1	Right	A
A	1	⇒	0	Right	B
A	1	⇒	0	Right	A
B	0	⇒	1	Right	B
B	1	⇒	1	Left	C
C	1	⇒	0	Left	C
C	0	⇒	1	Right	A
C	0	⇒	1	Left	A

1. NTMs can be simulated by DTMs, but required steps grow exponentially with the length of the shortest accepting path

$\mathcal{NP}$-Hardness

1. NTMs can be simulated by DTMs, but required steps grow exponentially with the length of the shortest accepting path
   - "normal" computers can be simulated with DTMs and vice versa [6]

1. NTMs can be simulated by DTMs, but required steps grow exponentially with the length of the shortest accepting path
   - "normal" computers can be simulated with DTMs and vice versa [6]
2. Problems which can be solved by a DTM (or "normal" computer) in polynomial time are in class $\mathcal{P}$ (algorithm in $\mathcal{O}(x^p)$ with $p \in \mathbb{N}_1$)

$\mathcal{NP}$-Hardness

1. NTMs can be simulated by DTMs, but required steps grow exponentially with the length of the shortest accepting path
2. Problems which can be solved by a DTM (or "normal" computer) in polynomial time are in class $\mathcal{P}$ (algorithm in $\mathcal{O}(x^p)$ with $p \in \mathbb{N}_1$)
3. Problems which can be solved by a NTM in polynomial time are in the class $\mathcal{NP}$ [7]

1. NTMs can be simulated by DTMs, but required steps grow exponentially with the length of the shortest accepting path
2. Problems which can be solved by a DTM (or "normal" computer) in polynomial time are in class $\mathcal{P}$ (algorithm in $\mathcal{O}(x^p)$ with $p \in \mathbb{N}_1$)
3. Problems which can be solved by a NTM in polynomial time are in the class $\mathcal{NP}$ [7]
   - $\mathcal{P} \subseteq \mathcal{NP}$ (all problems in $\mathcal{P}$ are also in $\mathcal{NP}$)

$\mathcal{NP}$-Hardness

1. NTMs can be simulated by DTMs, but required steps grow exponentially with the length of the shortest accepting path
2. Problems which can be solved by a DTM (or "normal" computer) in polynomial time are in class $\mathcal{P}$ (algorithm in $\mathcal{O}(x^p)$ with $p \in \mathbb{N}_1$)
3. Problems which can be solved by a NTM in polynomial time are in the class $\mathcal{NP}$ [7]
   - $\mathcal{P} \subseteq \mathcal{NP}$ (all problems in $\mathcal{P}$ are also in $\mathcal{NP}$)
   - $\mathcal{NP} \subseteq \mathcal{P}$, i.e., $\mathcal{P} = \mathcal{NP}$: research question, probably does not hold

1. NTMs can be simulated by DTMs, but required steps grow exponentially with the length of the shortest accepting path
2. Problems which can be solved by a DTM (or "normal" computer) in polynomial time are in class $\mathcal{P}$ (algorithm in $\mathcal{O}(x^p)$ with $p \in \mathbb{N}_1$)
3. Problems which can be solved by a NTM in polynomial time are in the class $\mathcal{NP}$ [7]
   - $\mathcal{P} \subseteq \mathcal{NP}$ (all problems in $\mathcal{P}$ are also in $\mathcal{NP}$)
   - $\mathcal{NP} \subseteq \mathcal{P}$, i.e., $\mathcal{P} = \mathcal{NP}$: research question, probably does not hold
   - Algorithm for problem A can also solve problem B: B is not harder than A, B reduces to A

$\mathcal{NP}$-Hardness

1. NTMs can be simulated by DTMs, but required steps grow exponentially with the length of the shortest accepting path
2. Problems which can be solved by a DTM (or "normal" computer) in polynomial time are in class $\mathcal{P}$ (algorithm in $\mathcal{O}(x^p)$ with $p \in \mathbb{N}_1$)
3. Problems which can be solved by a NTM in polynomial time are in the class $\mathcal{NP}$ [7]
   - $\mathcal{P} \subseteq \mathcal{NP}$ (all problems in $\mathcal{P}$ are also in $\mathcal{NP}$)
   - $\mathcal{NP} \subseteq \mathcal{P}$, i.e., $\mathcal{P} = \mathcal{NP}$: research question, probably does not hold
   - Algorithm for problem A can also solve problem B: B is not harder than A, B reduces to A
   - All other problems in $\mathcal{NP}$ can be reduced to $\mathcal{NP}$-hard problems

1. NTMs can be simulated by DTMs, but required steps grow exponentially with the length of the shortest accepting path
2. Problems which can be solved by a DTM (or "normal" computer) in polynomial time are in class $\mathcal{P}$ (algorithm in $\mathcal{O}(x^p)$ with $p \in \mathbb{N}_1$)
3. Problems which can be solved by a NTM in polynomial time are in the class $\mathcal{NP}$ [7]
   - $\mathcal{P} \subseteq \mathcal{NP}$ (all problems in $\mathcal{P}$ are also in $\mathcal{NP}$)
   - $\mathcal{NP} \subseteq \mathcal{P}$, i.e., $\mathcal{P} = \mathcal{NP}$: research question, probably does not hold
   - Algorithm for problem A can also solve problem B: B is not harder than A, B reduces to A
   - All other problems in $\mathcal{NP}$ can be reduced to $\mathcal{NP}$-hard problems
   - No (worst-case) polynomial-time and -space algorithm known for $\mathcal{NP}$-hard problems

Solving $\mathcal{NP}$-hard Problems

- No algorithm with polynomial worst-case space and time complexity is known for $\mathcal{NP}$-hard problems.

Solving $\mathcal{NP}$-hard Problems

- No algorithm with polynomial worst-case space and time complexity is known for $\mathcal{NP}$-hard problems.
- It cannot be *guaranteed* to *always* find the globally optimal solution for such problems in reasonable time.

Solving $\mathcal{NP}$-hard Problems

- No algorithm with polynomial worst-case space and time complexity is known for $\mathcal{NP}$-hard problems.
- It cannot be *guaranteed* to *always* find the globally optimal solution for such problems in reasonable time.
- Many real-world problems are $\mathcal{NP}$-hard: Traveling Salesman Problems, Constraint Satisfaction Problems, Bin Packing, Vehicle Routing, ...

Solving $\mathcal{NP}$-hard Problems

- No algorithm with polynomial worst-case space and time complexity is known for $\mathcal{NP}$-hard problems.
- It cannot be *guaranteed* to *always* find the globally optimal solution for such problems in reasonable time.
- Many real-world problems are $\mathcal{NP}$-hard: Traveling Salesman Problems, Constraint Satisfaction Problems, Bin Packing, Vehicle Routing, ...
- What does this mean?

- It cannot be *guaranteed* to *always* find the globally optimal solution for such problems in reasonable time.
- Many real-world problems are $\mathcal{NP}$-hard: Traveling Salesman Problems, Constraint Satisfaction Problems, Bin Packing, Vehicle Routing, . . .
- What does this mean?
  1. If we try to find the exact globally optimal solution of these problems, it *may* take very long in the worst case. . .

Solving $\mathcal{NP}$-hard Problems

- It cannot be *guaranteed* to *always* find the globally optimal solution for such problems in reasonable time.
  1. If we try to find the exact globally optimal solution of these problems, it *may* take very long in the worst case...
  2. ...but it *may* also be possible in acceptable time (in the best or even average case).

Solving $\mathcal{NP}$-hard Problems

- It cannot be *guaranteed* to *always* find the globally optimal solution for such problems in reasonable time.
  1. If we try to find the exact globally optimal solution of these problems, it *may* take very long in the worst case...
  2. ...but it *may* also be possible in acceptable time (in the best or even average case).
  3. Example: For several pure, classical problems like the TSP, exact solutions can be found for problems with large scale in reasonable time (TSP: instances with $\geq 75\,000$ cities have been solved!).

Solving $\mathcal{NP}$-hard Problems

- It cannot be *guaranteed* to *always* find the globally optimal solution for such problems in reasonable time.
  1. If we try to find the exact globally optimal solution of these problems, it *may* take very long in the worst case. . .
  2. . . . but it *may* also be possible in acceptable time (in the best or even average case).
  3. Example: For several pure, classical problems like the TSP, exact solutions can be found for problems with large scale in reasonable time (TSP: instances with $\geq 75\,000$ cities have been solved!).
  4. (Meta-)Heuristic algorithms may provide very good approximate solutions very quickly and often even find the global optimum in a short time . . . but often cannot make good guarantees neither about runtime nor solution quality.

- It cannot be *guaranteed* to *always* find the globally optimal solution for such problems in reasonable time.
  1. If we try to find the exact globally optimal solution of these problems, it *may* take very long in the worst case...
  2. ...but it *may* also be possible in acceptable time (in the best or even average case).
  3. Example: For several pure, classical problems like the TSP, exact solutions can be found for problems with large scale in reasonable time (TSP: instances with $\geq 75\,000$ cities have been solved!).
  4. (Meta-)Heuristic algorithms may provide very good approximate solutions very quickly and often even find the global optimum in a short time ... but often cannot make good guarantees neither about runtime nor solution quality.
  5. We need to trade-off runtime vs. solution quality, especially if the problem is not well-researched.

Section Outline

Preface

- OK, so let us use a *randomized algorithm* that gives an *approximate* solution instead of a deterministic exact algorithm!

- OK, so let us use a *randomized algorithm* that gives an *approximate* solution instead of a deterministic exact algorithm!
- We cannot expect to get the global optimum

- OK, so let us use a *randomized algorithm* that gives an *approximate* solution instead of a deterministic exact algorithm!
- We cannot expect to get the global optimum
- We are interested in the quality of the solutions that we can get

Preface

- OK, so let us use a *randomized algorithm* that gives an *approximate* solution instead of a deterministic exact algorithm!
- We cannot expect to get the global optimum
- We are interested in the quality of the solutions that we can get
- What features of problems or algorithms allow us to get good solutions?

Preface

- OK, so let us use a *randomized algorithm* that gives an *approximate* solution instead of a deterministic exact algorithm!
- We cannot expect to get the global optimum
- We are interested in the quality of the solutions that we can get
- What features of problems or algorithms allow us to get good solutions?
- When/why can we not get good solutions?

Convergence

- Let us start at the end!

- Let us start at the end!
- Sooner or later, every optimization process ends (converges)

Convergence

- Let us start at the end!
- Sooner or later, every optimization process ends (converges)

Definition (Convergence)

An optimization algorithm has converged if it cannot reach new candidate solutions anymore or if it keeps on producing candidate solutions from a small subset of the solution space $\mathbb{X}$. [8, 9]

- Let us start at the end!
- Sooner or later, every optimization process ends (converges)

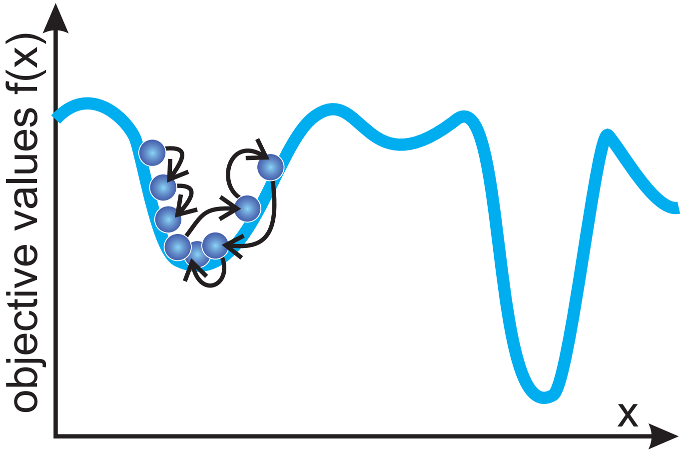
Definition (Convergence)

An optimization algorithm has converged if it cannot reach new candidate solutions anymore or if it keeps on producing candidate solutions from a small subset of the solution space $\mathbb{X}$. [8, 9]

Definition (multi-modality)

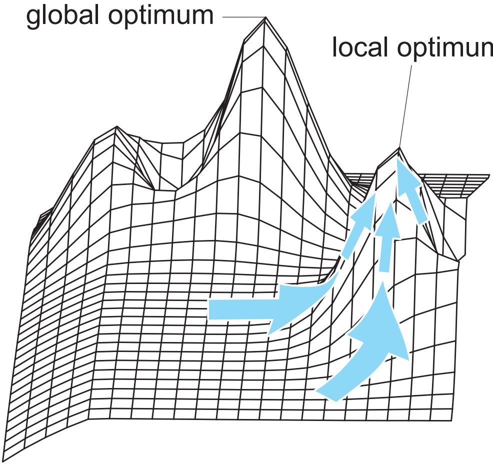
A function/optimization problem is multi-modal if it has more than one minimum / maximum /optimum. [10–14]

- Premature Convergence = convergence to local optimum [8, 9]

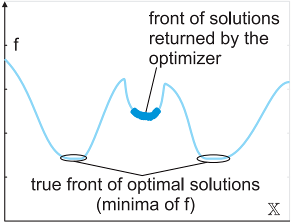


Premature Convergence

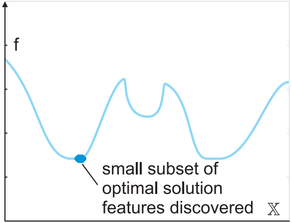
- Premature Convergence = convergence to local optimum [8, 9]



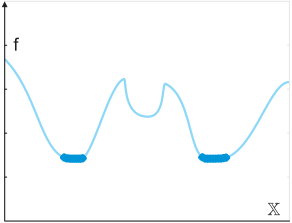
- Uniformity of convergence: We want a good scan of the potentially optimal features



Bad convergence, good spread (uniformity)

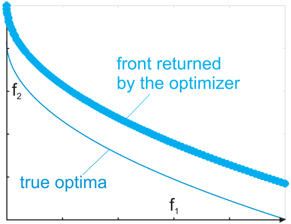


Good convergence, bad spread (non-uniformity)

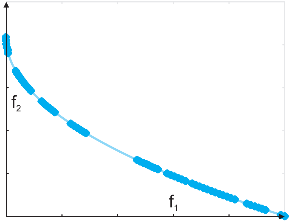


Good convergence, good spread (uniformity)

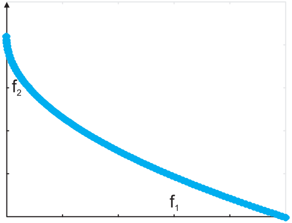
- Uniformity of convergence



Bad convergence, good spread (uniformity) [1, 15]



Good convergence, bad spread (non-uniformity) [1, 15]



Good convergence, good spread (uniformity) [1, 15]

- So what is the basic reason for unsatisfying convergence?

Exploration versus Exploitation

- So what is the basic reason for unsatisfying convergence?
- Exploration: search in distant areas of the search space, strong randomization: slow improvement

Exploration versus Exploitation

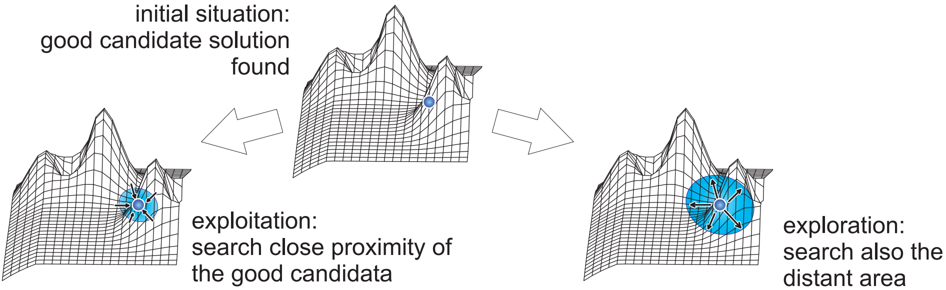
- So what is the basic reason for unsatisfying convergence?
- Exploration: search in distant areas of the search space, strong randomization: slow improvement
- Exploitation: analyze neighborhood of current best solutions: fast improvement/convergence

Exploration versus Exploitation

- So what is the basic reason for unsatisfying convergence?
- Exploration: search in distant areas of the search space, strong randomization: slow improvement
- Exploitation: analyze neighborhood of current best solutions: fast improvement/convergence
- Which should we use more?

Exploration versus Exploitation

- Which should we use more?
- This is called the Exploration versus Exploitation Dilemma [16–23]



Countermeasures

- Countermeasures against Premature Convergence

- Countermeasures against Premature Convergence
  1. Delay convergence by balancing exploration and exploitation and remembering diverse candidate solutions [16, 17, 21]

Countermeasures

- Countermeasures against Premature Convergence
  1. Delay convergence by balancing exploration and exploitation and remembering diverse candidate solutions [16, 17, 21]
  2. Design complete search operators [24–26]

Countermeasures

- Countermeasures against Premature Convergence
  1. Delay convergence by balancing exploration and exploitation and remembering diverse candidate solutions [16, 17, 21]
  2. Design complete search operators [24–26]:
     - it should be possible to reach all other points in the search space from the current one

Countermeasures

- Countermeasures against Premature Convergence
  1. Delay convergence by balancing exploration and exploitation and remembering diverse candidate solutions [16, 17, 21]
  2. Design complete search operators [24–26]:
     - it should be possible to reach all other points in the search space from the current one
     - ideally within one search step

Countermeasures

- Countermeasures against Premature Convergence
  1. Delay convergence by balancing exploration and exploitation and remembering diverse candidate solutions [16, 17, 21]
  2. Design complete search operators [24–26]
  3. Restarting [27, 28]

Countermeasures

- Countermeasures against Premature Convergence
  1. Delay convergence by balancing exploration and exploitation and remembering diverse candidate solutions [16, 17, 21]
  2. Design complete search operators [24–26]
  3. Restarting [27, 28]:
     - if no improvement for some time, restart algorithm

Countermeasures

- Countermeasures against Premature Convergence
  1. Delay convergence by balancing exploration and exploitation and remembering diverse candidate solutions [16, 17, 21]
  2. Design complete search operators [24–26]
  3. Restarting [27, 28]
  4. Low selection pressure and/or larger population size [29–31]

Countermeasures

- Countermeasures against Premature Convergence
  1. Delay convergence by balancing exploration and exploitation and remembering diverse candidate solutions [16, 17, 21]
  2. Design complete search operators [24–26]
  3. Restarting [27, 28]
  4. Low selection pressure and/or larger population size [29–31]:
     - allows for more exploration by putting less pressure to move to better solutions

Countermeasures

- Countermeasures against Premature Convergence
    1. Delay convergence by balancing exploration and exploitation and remembering diverse candidate solutions [16, 17, 21]
    2. Design complete search operators [24–26]
    3. Restarting [27, 28]
    4. Low selection pressure and/or larger population size [29–31]:
        - allows for more exploration by putting less pressure to move to better solutions
        - slows down search

Countermeasures

- Countermeasures against Premature Convergence
  1. Delay convergence by balancing exploration and exploitation and remembering diverse candidate solutions [16, 17, 21]
  2. Design complete search operators [24–26]
  3. Restarting [27, 28]
  4. Low selection pressure and/or larger population size [29–31]:
     - allows for more exploration by putting less pressure to move to better solutions
     - slows down search
     - only sometimes [31, 68]

- Countermeasures against Premature Convergence
  1. Delay convergence by balancing exploration and exploitation and remembering diverse candidate solutions [16, 17, 21]
  2. Design complete search operators [24–26]
  3. Restarting [27, 28]
  4. Low selection pressure and/or larger population size [29–31]
  5. Sharing, Niching, and Clearing [32–45]

- Countermeasures against Premature Convergence
  1. Delay convergence by balancing exploration and exploitation and remembering diverse candidate solutions [16, 17, 21]
  2. Design complete search operators [24–26]
  3. Restarting [27, 28]
  4. Low selection pressure and/or larger population size [29–31]
  5. Sharing, Niching, and Clearing [32–45]:
     - Give solutions that are very similar to each other a worse fitness

Countermeasures

- Countermeasures against Premature Convergence
  1. Delay convergence by balancing exploration and exploitation and remembering diverse candidate solutions [16, 17, 21]
  2. Design complete search operators [24–26]
  3. Restarting [27, 28]
  4. Low selection pressure and/or larger population size [29–31]
  5. Sharing, Niching, and Clearing [32–45]
  6. Clustering of candidate solutions [46–57]

Countermeasures

- Countermeasures against Premature Convergence
  1. Delay convergence by balancing exploration and exploitation and remembering diverse candidate solutions [16, 17, 21]
  2. Design complete search operators [24–26]
  3. Restarting [27, 28]
  4. Low selection pressure and/or larger population size [29–31]
  5. Sharing, Niching, and Clearing [32–45]
  6. Clustering of candidate solutions [46–57]:
     - Clustering: unsupervised machine learning – divide a set of elements into groups of similar elements

Countermeasures

- Countermeasures against Premature Convergence
  1. Delay convergence by balancing exploration and exploitation and remembering diverse candidate solutions [16, 17, 21]
  2. Design complete search operators [24–26]
  3. Restarting [27, 28]
  4. Low selection pressure and/or larger population size [29–31]
  5. Sharing, Niching, and Clearing [32–45]
  6. Clustering of candidate solutions [46–57]:
     - Clustering: unsupervised machine learning – divide a set of elements into groups of similar elements
     - Here: cluster population, treat every cluster separately

Countermeasures

- Countermeasures against Premature Convergence
  1. Delay convergence by balancing exploration and exploitation and remembering diverse candidate solutions [16, 17, 21]
  2. Design complete search operators [24–26]
  3. Restarting [27, 28]
  4. Low selection pressure and/or larger population size [29–31]
  5. Sharing, Niching, and Clearing [32–45]
  6. Clustering of candidate solutions [46–57]:
     - Clustering: unsupervised machine learning – divide a set of elements into groups of similar elements
     - Here: cluster population, treat every cluster separately
     - Allows the population to trace different optima at once

Countermeasures

- Countermeasures against Premature Convergence
  1. Delay convergence by balancing exploration and exploitation and remembering diverse candidate solutions [16, 17, 21]
  2. Design complete search operators [24–26]
  3. Restarting [27, 28]
  4. Low selection pressure and/or larger population size [29–31]
  5. Sharing, Niching, and Clearing [32–45]
  6. Clustering of candidate solutions [46–57]
  7. Self-Adaptation [58, 59]

Countermeasures

- Countermeasures against Premature Convergence
  1. Delay convergence by balancing exploration and exploitation and remembering diverse candidate solutions [16, 17, 21]
  2. Design complete search operators [24–26]
  3. Restarting [27, 28]
  4. Low selection pressure and/or larger population size [29–31]
  5. Sharing, Niching, and Clearing [32–45]
  6. Clustering of candidate solutions [46–57]
  7. Self-Adaptation [58, 59]:
     - change parameters of optimization algorithm in order to prevent (or speed-up) convergence

Countermeasures

- Countermeasures against Premature Convergence
  1. Delay convergence by balancing exploration and exploitation and remembering diverse candidate solutions [16, 17, 21]
  2. Design complete search operators [24–26]
  3. Restarting [27, 28]
  4. Low selection pressure and/or larger population size [29–31]
  5. Sharing, Niching, and Clearing [32–45]
  6. Clustering of candidate solutions [46–57]
  7. Self-Adaptation [58, 59]
  8. Multi-Objectivization [60–67]

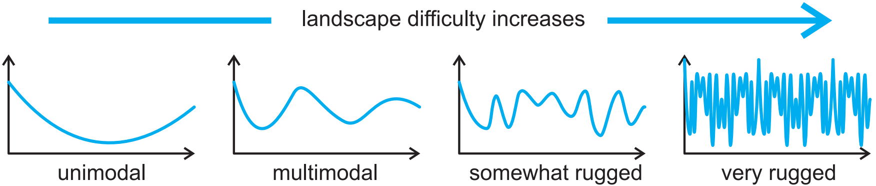
Countermeasures

- Countermeasures against Premature Convergence
  1. Delay convergence by balancing exploration and exploitation and remembering diverse candidate solutions [16, 17, 21]
  2. Design complete search operators [24–26]
  3. Restarting [27, 28]
  4. Low selection pressure and/or larger population size [29–31]
  5. Sharing, Niching, and Clearing [32–45]
  6. Clustering of candidate solutions [46–57]
  7. Self-Adaptation [58, 59]
  8. Multi-Objectivization [60–67]:
     - turn a single-objective problem into an multi-objective one by creating an artificial objective function targeting one specific aspect of the solutions

Countermeasures

- Countermeasures against Premature Convergence
  1. Delay convergence by balancing exploration and exploitation and remembering diverse candidate solutions [16, 17, 21]
  2. Design complete search operators [24–26]
  3. Restarting [27, 28]
  4. Low selection pressure and/or larger population size [29–31]
  5. Sharing, Niching, and Clearing [32–45]
  6. Clustering of candidate solutions [46–57]
  7. Self-Adaptation [58, 59]
  8. Multi-Objectivization [60–67]:
     - turn a single-objective problem into an multi-objective one by creating an artificial objective function targeting one specific aspect of the solutions
     - Pareto-based optimization (see Lesson 15: *Multi-Objective Optimization*) then increases diversity

Section Outline


landscape difficulty increases
unimodal
multimodal
somewhat rugged
very rugged

- Why??

Causality

- Basic assumption behind metaheuristic optimization:

- Basic assumption behind metaheuristic optimization:

Definition (Strong Causality)

Small changes to an object should lead to small changes in its behavior / objective values. [69–71]

- What happens if the causality in a problem is weak?

Countermeasures

1. Hybridization of EAs with local search:

Countermeasures

1. Hybridization of EAs with local search:
   - Lamarckian evolution [72, 73]

Countermeasures

1. Hybridization of EAs with local search:
   - Lamarckian evolution [72, 73]:
     - EA + local search on the genotype level

Countermeasures

1. Hybridization of EAs with local search:
   - Lamarckian evolution [72, 73]:
     - EA + local search on the genotype level
     - Each genotype is generated by the normal search operations and then refined with a local search

1. Hybridization of EAs with local search:
   - Lamarckian evolution [72, 73]:
     - EA + local search on the genotype level
     - Each genotype is generated by the normal search operations and then refined with a local search
     - The fitness landscape appears smooth to the EA, as it only sees local optima and not the rugged spikes in between

Countermeasures

1. Hybridization of EAs with local search:
   - Lamarckian evolution [72, 73]
   - Baldwin effect [73–76]

Countermeasures

1. Hybridization of EAs with local search:
   - Lamarckian evolution [72, 73]
   - Baldwin effect [73–76]:
     - EA + local search on the phenotype level

Countermeasures

1. Hybridization of EAs with local search:
   - Lamarckian evolution [72, 73]
   - Baldwin effect [73–76]:
     - EA + local search on the phenotype level
     - Each phenotype is the result of a GPM applied to a genotype

Countermeasures

1. Hybridization of EAs with local search:
   - Lamarckian evolution [72, 73]
   - Baldwin effect [73–76]:
     - EA + local search on the phenotype level
     - Each phenotype is the result of a GPM applied to a genotype
     - The phenotype is then refined with a local search

Countermeasures

1. Hybridization of EAs with local search:
   - Lamarckian evolution [72, 73]
   - Baldwin effect [73–76]:
     - EA + local search on the phenotype level
     - Each phenotype is the result of a GPM applied to a genotype
     - The phenotype is then refined with a local search
     - The fitness landscape appears smooth to the EA, as it only sees local optima and not the rugged spikes in between

Countermeasures

1. Hybridization of EAs with local search:
   - Lamarckian evolution [72, 73]
   - Baldwin effect [73–76]
   - Memetic Algorithms [77–86]

Countermeasures

1. Hybridization of EAs with local search:
   - Lamarckian evolution [72, 73]
   - Baldwin effect [73–76]
   - Memetic Algorithms [77–86]:
     - Similar to Baldwin effect and Lamarckian evolution

Countermeasures

1. Hybridization of EAs with local search:
    - Lamarckian evolution [72, 73]
    - Baldwin effect [73–76]
    - Memetic Algorithms [77–86]:
        - Similar to Baldwin effect and Lamarckian evolution
        - The search operations themselves perform some local optimization

1. Hybridization of EAs with local search:
   - Lamarckian evolution [72, 73]
   - Baldwin effect [73–76]
   - Memetic Algorithms [77–86]:
     - Similar to Baldwin effect and Lamarckian evolution
     - The search operations themselves perform some local optimization
     - The fitness landscape appears smooth to the EA, as it only sees local optima and not the rugged spikes in between

Countermeasures

1. Hybridization of EAs with local search:
   - Lamarckian evolution [72, 73]
   - Baldwin effect [73–76]
   - Memetic Algorithms [77–86]
   - other hybrid approaches [87–94]

Countermeasures

1. Hybridization of EAs with local search:
   - Lamarckian evolution [72, 73]
   - Baldwin effect [73–76]
   - Memetic Algorithms [77–86]
   - other hybrid approaches [87–94]:
     - Combinations of Evolutionary Algorithms with local search, or combinations of Evolutionary Algorithms with other concepts from Machine Learning

Countermeasures

1. Hybridization of EAs with local search:
   - Lamarckian evolution [72, 73]
   - Baldwin effect [73–76]
   - Memetic Algorithms [77–86]
   - other hybrid approaches [87–94]
2. Landscape approximation [95]

Countermeasures

1. Hybridization of EAs with local search:
   - Lamarckian evolution [72, 73]
   - Baldwin effect [73–76]
   - Memetic Algorithms [77–86]
   - other hybrid approaches [87–94]
2. Landscape approximation [95]:
   - Try to adjust the parameters of a (simple) model M or function so that it behaves similar to the (points so-far seen from the) objective function f

Countermeasures

1. Hybridization of EAs with local search:
   - Lamarckian evolution [72, 73]
   - Baldwin effect [73–76]
   - Memetic Algorithms [77–86]
   - other hybrid approaches [87–94]
2. Landscape approximation [95]:
   - Try to adjust the parameters of a (simple) model M or function so that it behaves similar to the (points so-far seen from the) objective function f
   - Optimize on this simple model (which has stronger causality)

Countermeasures

1. Hybridization of EAs with local search:
   - Lamarckian evolution [72, 73]
   - Baldwin effect [73–76]
   - Memetic Algorithms [77–86]
   - other hybrid approaches [87–94]
2. Landscape approximation [95]:
   - Try to adjust the parameters of a (simple) model M or function so that it behaves similar to the (points so-far seen from the) objective function f
   - Optimize on this simple model (which has stronger causality)
   - After a few steps, go back to original f and test solutions

Countermeasures

1. Hybridization of EAs with local search:
   - Lamarckian evolution [72, 73]
   - Baldwin effect [73–76]
   - Memetic Algorithms [77–86]
   - other hybrid approaches [87–94]
2. Landscape approximation [95]:
   - Try to adjust the parameters of a (simple) model M or function so that it behaves similar to the (points so-far seen from the) objective function f
   - Optimize on this simple model (which has stronger causality)
   - After a few steps, go back to original f and test solutions
   - Update M, then repeat

Countermeasures

1. Hybridization of EAs with local search:
   - Lamarckian evolution [72, 73]
   - Baldwin effect [73–76]
   - Memetic Algorithms [77–86]
   - other hybrid approaches [87–94]
2. Landscape approximation [95]
3. (2-, n-) Staged optimization [96]

Countermeasures

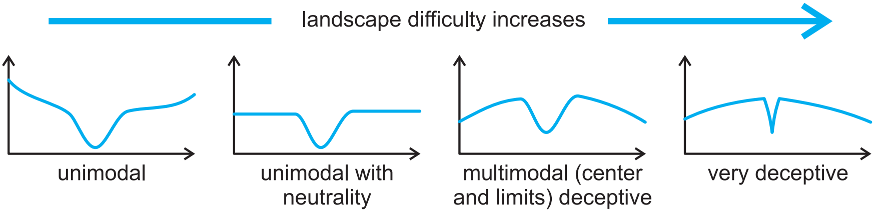
1. Hybridization of EAs with local search:
   - Lamarckian evolution [72, 73]
   - Baldwin effect [73–76]
   - Memetic Algorithms [77–86]
   - other hybrid approaches [87–94]
2. Landscape approximation [95]
3. (2-, n-) Staged optimization [96]:
   - First, apply an optimization algorithm with slow convergence, which is good in exploring the search space and finding the region where the optimum may reside

Countermeasures

1. Hybridization of EAs with local search:
   - Lamarckian evolution [72, 73]
   - Baldwin effect [73–76]
   - Memetic Algorithms [77–86]
   - other hybrid approaches [87–94]
2. Landscape approximation [95]
3. (2-, n-) Staged optimization [96]:
   - First, apply an optimization algorithm with slow convergence, which is good in exploring the search space and finding the region where the optimum may reside
   - Then, apply an optimization algorithm which is very good at explotation in that region only

Section Outline

- Gradient and information lead optimizer away from optimum [88, 97, 98]



- Why??

Countermeasures

- Countermeasures

- Countermeasures
  1. Choose appropriate representation, maybe combine representations [99]

Countermeasures

- Countermeasures
  1. Choose appropriate representation, maybe combine representations [99]
  2. Preventing convergence:

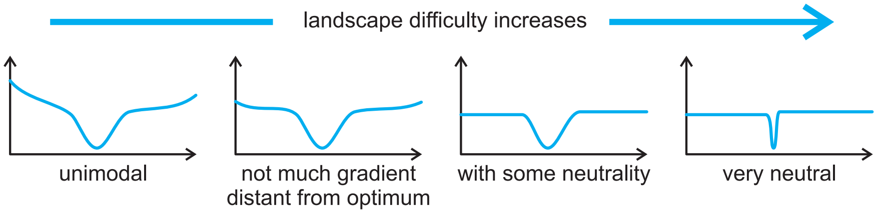
Countermeasures

- Countermeasures
  1. Choose appropriate representation, maybe combine representations [99]
  2. Preventing convergence:
     - Fitness Uniform Selection Scheme [100–103]

- Countermeasures
  1. Choose appropriate representation, maybe combine representations [99]
  2. Preventing convergence:
     - Fitness Uniform Selection Scheme [100–103]
     - Novelty Search [104–106]

Section Outline

- Neutrality: Many candidate solutions have same objective values
- Little or no information gained from sampling the solution space



- Why??

Good and Bad

Definition (Evolvability)

The evolvability of an optimization process in its current state defines how likely the search operations will lead to candidate solutions with new (and eventually, better) objectives values. [107–111]

Good and Bad

Definition (Evolvability)

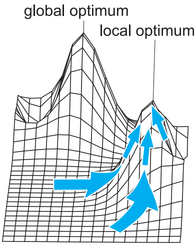
The evolvability of an optimization process in its current state defines how likely the search operations will lead to candidate solutions with new (and eventually, better) objectives values. [107–111]

- Neutral networks can connect different places in the search space

Definition (Evolvability)

The evolvability of an optimization process in its current state defines how likely the search operations will lead to candidate solutions with new (and eventually, better) objectives values. [107–111]

- Neutral networks can connect different places in the search space

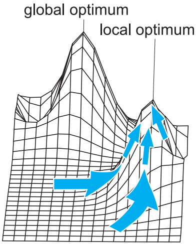


premature convergence

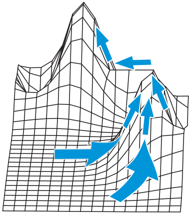
Definition (Evolvability)

The evolvability of an optimization process in its current state defines how likely the search operations will lead to candidate solutions with new (and eventually, better) objectives values. [107–111]

- Neutral networks can connect different places in the search space



premature convergence

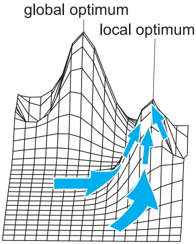


small neutral bridge

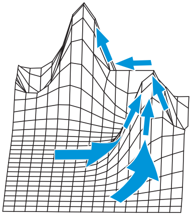
Definition (Evolvability)

The evolvability of an optimization process in its current state defines how likely the search operations will lead to candidate solutions with new (and eventually, better) objectives values. [107–111]

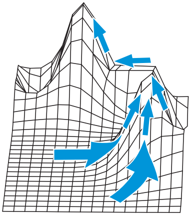
- Neutral networks can connect different places in the search space



premature convergence



small neutral bridge



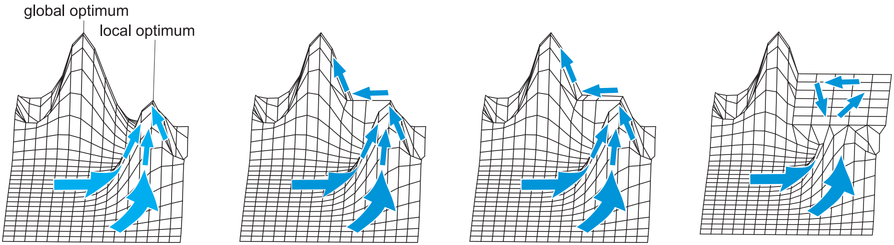
wide neutral bridge

Good and Bad

Definition (Evolvability)

The evolvability of an optimization process in its current state defines how likely the search operations will lead to candidate solutions with new (and eventually, better) objectives values. [107–111]

- Neutral networks can connect different places in the search space



premature convergence | small neutral bridge | wide neutral bridge | neutral area too large

Section Outline

Epistasis: The other root of all evil...

Definition (Epistasis)

One gene influences the behavior (contribution to the objective function) of other genes [39, 112–119]

Epistasis: The other root of all evil. . .

Definition (Epistasis)

One gene influences the behavior (contribution to the objective function) of other genes [39, 112–119]

Definition (Pleiotropy)

One gene is responsible for multiple phenotypical traits [108]

Epistasis: The other root of all evil...

Definition (Epistasis)

One gene influences the behavior (contribution to the objective function) of other genes [39, 112–119]

Definition (Pleiotropy)

One gene is responsible for multiple phenotypical traits [108]

Definition (Separability)

A function of n variables is separable if it can be rewritten as a sum of n functions of just one variable [120–123]

Epistasis: The other root of all evil...

Definition (Epistasis)

One gene influences the behavior (contribution to the objective function) of other genes [39, 112–119]

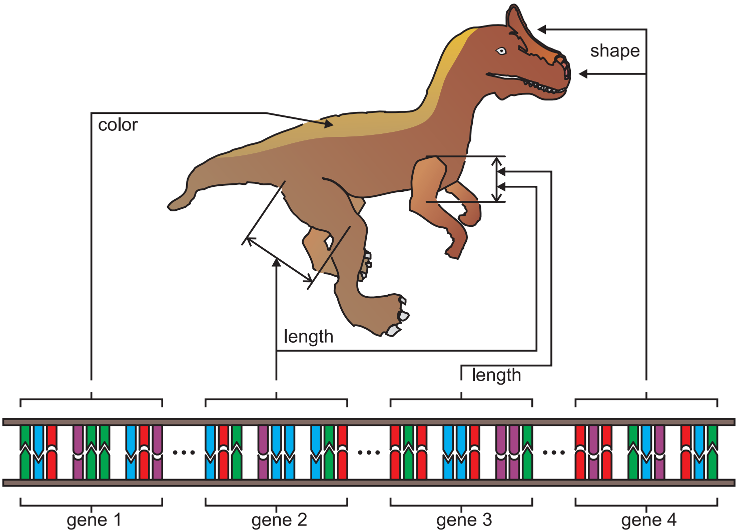
Definition (Pleiotropy)

One gene is responsible for multiple phenotypical traits [108]

Definition (Separability)

A function of n variables is separable if it can be rewritten as a sum of n functions of just one variable [120–123]

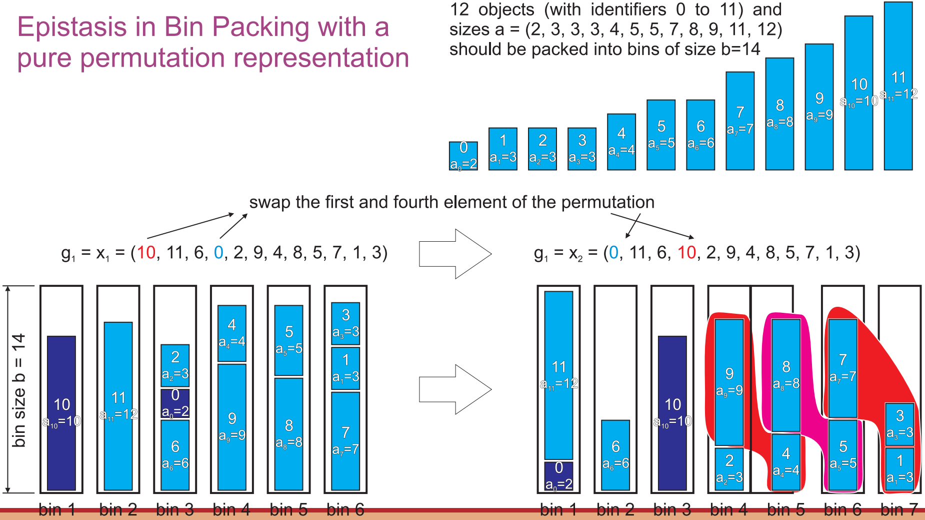
- non-epistatic (separable) problems can be solved efficiently by decomposition

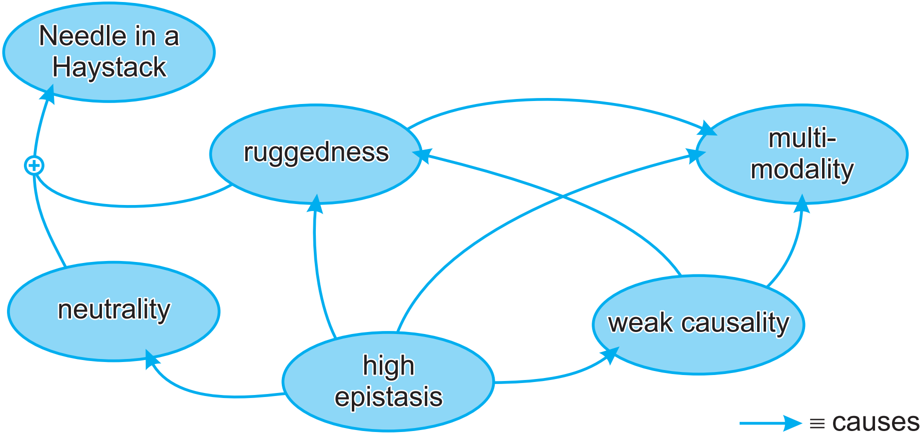


shape
color
length
length
gene 1
gene 2
gene 3
gene 4

Epistasis in Bin Packing with a pure permutation representation

12 objects (with identifiers 0 to 11) and sizes a = (2, 3, 3, 3, 4, 5, 5, 7, 8, 9, 11, 12) should be packed into bins of size b=14




Needle in a Haystack
ruggedness
multi-modality
neutrality
high epistasis
weak causality
≡ causes

Countermeasures

1. See countermeasures for ruggedness, neutrality, multi-modality...

Countermeasures

1. See countermeasures for ruggedness, neutrality, multi-modality...
2. Choose appropriate representation [124–127] and search operators [128]

Countermeasures

1. See countermeasures for ruggedness, neutrality, multi-modality...
2. Choose appropriate representation [124–127] and search operators [128]
3. Parameter Tweaking [128]

Countermeasures

1. See countermeasures for ruggedness, neutrality, multi-modality...
2. Choose appropriate representation [124–127] and search operators [128]
3. Parameter Tweaking [128]
4. Linkage learning [129–135] and Variable Interaction Learning [136]

Countermeasures

1. See countermeasures for ruggedness, neutrality, multi-modality...
2. Choose appropriate representation [124–127] and search operators [128]
3. Parameter Tweaking [128]
4. Linkage learning [129–135] and Variable Interaction Learning [136]:
   - Try to find out which genes (components of the genotype) are (epistatically) linked together

Countermeasures

1. See countermeasures for ruggedness, neutrality, multi-modality...
2. Choose appropriate representation [124–127] and search operators [128]
3. Parameter Tweaking [128]
4. Linkage learning [129–135] and Variable Interaction Learning [136]:
    - Try to find out which genes (components of the genotype) are (epistatically) linked together
    - Try to change these genes only together, consider them as a unit

Countermeasures

1. See countermeasures for ruggedness, neutrality, multi-modality...
2. Choose appropriate representation [124–127] and search operators [128]
3. Parameter Tweaking [128]
4. Linkage learning [129–135] and Variable Interaction Learning [136]:
    - Try to find out which genes (components of the genotype) are (epistatically) linked together
    - Try to change these genes only together, consider them as a unit
    - For example: In crossover, try to always pass such genes together to the offspring and to not separate them

Countermeasures

1. See countermeasures for ruggedness, neutrality, multi-modality...
2. Choose appropriate representation [124–127] and search operators [128]
3. Parameter Tweaking [128]
4. Linkage learning [129–135] and Variable Interaction Learning [136]
5. If epistasis is limited: cooperative-coevolution approach [136–138] (see later)

Section Outline

Scalability

- Time required for solving $\mathcal{NP}$-hard problems grows exponential with input size

Scalability

- Time required for solving $\mathcal{NP}$-hard problems grows exponential with input size
- Metaheuristic optimization: approximately solve $\mathcal{NP}$-hard problems in feasible time

Scalability

- Time required for solving $\mathcal{NP}$-hard problems grows exponential with input size
- Metaheuristic optimization: approximately solve $\mathcal{NP}$-hard problems in feasible time
- ... but their time requirement also grows with problem size...

Scalability

- Time required for solving $\mathcal{NP}$-hard problems grows exponential with input size
- Metaheuristic optimization: approximately solve $\mathcal{NP}$-hard problems in feasible time
- ... but their time requirement also grows with problem size...
- "Curse of Dimensionality": solution space volume increases exponentially with number of decision variables (genes) [139, 140]

Scalability

- Time required for solving $\mathcal{NP}$-hard problems grows exponential with input size
- Metaheuristic optimization: approximately solve $\mathcal{NP}$-hard problems in feasible time
- . . . but their time requirement also grows with problem size. . .
- "Curse of Dimensionality": solution space volume increases exponentially with number of decision variables (genes) [139, 140]
- Example: search in $(1 \ldots 10)^n$

- Time required for solving $\mathcal{NP}$-hard problems grows exponential with input size
- Metaheuristic optimization: approximately solve $\mathcal{NP}$-hard problems in feasible time
- …but their time requirement also grows with problem size…
- "Curse of Dimensionality": solution space volume increases exponentially with number of decision variables (genes) [139, 140]
- Example: search in $(1\dots10)^n$
- any algorithm (for non-trivial problems) takes longer for larger inputs…

Countermeasures

- Countermeasures

- Countermeasures
  1. Parallelization and distribution

- Countermeasures
  1. Parallelization and distribution:
     - sub-linear speed-up can be achieved [149]

Countermeasures

- Countermeasures
  1. Parallelization and distribution:
     - sub-linear speed-up can be achieved [149]
     - Parallelization: Use multi-core CPU + multiple threads

- Countermeasures
  1. Parallelization and distribution:
     - sub-linear speed-up can be achieved [149]
     - Parallelization: Use multi-core CPU + multiple threads or GPUs [150–155]

Countermeasures

- Countermeasures
  1. Parallelization and distribution:
     - sub-linear speed-up can be achieved [149]
     - Parallelization: Use multi-core CPU + multiple threads or GPUs [150–155]
     - Distribution: Use multiple computers in a network [156, 157], a cluster, or a grid

- Countermeasures
  1. Parallelization and distribution
  2. Indirect Representation 1: Generative [141, 142]

- Countermeasures
  1. Parallelization and distribution
  2. Indirect Representation 1: Generative [141, 142]:
     - Genotypes are small, search space is smaller, can be explored more easily

Countermeasures

- Countermeasures
  1. Parallelization and distribution
  2. Indirect Representation 1: Generative [141, 142]:
     - Genotypes are small, search space is smaller, can be explored more easily
     - They are mapped to larger, more complex phenotypes by a simple functional GPM

- Countermeasures
  1. Parallelization and distribution
  2. Indirect Representation 1: Generative [141, 142]:
     - Genotypes are small, search space is smaller, can be explored more easily
     - They are mapped to larger, more complex phenotypes by a simple functional GPM
     - Utilizes/assumes symmetries in the phenotypes

Countermeasures

- Countermeasures
  1. Parallelization and distribution
  2. Indirect Representation 1: Generative [141, 142]
  3. Indirect Representation 2: Development [143, 144]

Countermeasures

- Countermeasures
  1. Parallelization and distribution
  2. Indirect Representation 1: Generative [141, 142]
  3. Indirect Representation 2: Development [143, 144]:
     - Similar to generative mapping, the search space is smaller

Countermeasures

- Countermeasures
  1. Parallelization and distribution
  2. Indirect Representation 1: Generative [141, 142]
  3. Indirect Representation 2: Development [143, 144]:
     - Similar to generative mapping, the search space is smaller
     - But: GPM is more complex, a simulation which incorporates feedback from an environment or the objective function

- Countermeasures
  1. Parallelization and distribution
  2. Indirect Representation 1: Generative [141, 142]
  3. Indirect Representation 2: Development [143, 144]:
     - Similar to generative mapping, the search space is smaller
     - But: GPM is more complex, a simulation which incorporates feedback from an environment or the objective function
     - Better behavior than generative mappings

Countermeasures

- Countermeasures
  1. Parallelization and distribution
  2. Indirect Representation 1: Generative [141, 142]
  3. Indirect Representation 2: Development [143, 144]
  4. Exploiting Separability, e.g., with coevolution [136–138, 145, 146]

- Countermeasures
  1. Parallelization and distribution
  2. Indirect Representation 1: Generative [141, 142]
  3. Indirect Representation 2: Development [143, 144]
  4. Exploiting Separability, e.g., with coevolution [136–138, 145, 146]:
     - Try to divide the problem into (almost) unrelated problems with smaller search spaces

Countermeasures

- Countermeasures
  1. Parallelization and distribution
  2. Indirect Representation 1: Generative [141, 142]
  3. Indirect Representation 2: Development [143, 144]
  4. Exploiting Separability, e.g., with coevolution [136–138, 145, 146]:
     - Try to divide the problem into (almost) unrelated problems with smaller search spaces
     - Solve them more or less separately, combine solutions to get overall solution, and repeat

Countermeasures

- Countermeasures
  1. Parallelization and distribution
  2. Indirect Representation 1: Generative [141, 142]
  3. Indirect Representation 2: Development [143, 144]
  4. Exploiting Separability, e.g., with coevolution [136–138, 145, 146]:
     - Try to divide the problem into (almost) unrelated problems with smaller search spaces
     - Solve them more or less separately, combine solutions to get overall solution, and repeat
     - Cooperative Coevolution [136, 138]: Use an EA that can find our how to divide the problem by itself and then applies the above

- Countermeasures
  1. Parallelization and distribution
  2. Indirect Representation 1: Generative [141, 142]
  3. Indirect Representation 2: Development [143, 144]
  4. Exploiting Separability, e.g., with coevolution [136–138, 145, 146]
  5. Using multiple algorithms at once [147] or portfolios [148]

Section Outline

No Free Lunch Theorem

- Question: Can an optimization algorithm A be better than algorithm B?

No Free Lunch Theorem

- Question: Can an optimization algorithm A be better than algorithm B?
- Question: Can an optimization algorithm A be better than a Random Walk?

- Question: Can an optimization algorithm A be better than algorithm B?
- Question: Can an optimization algorithm A be better than a Random Walk?
- Wolpert and Macready [158] – No Free Lunch Theorem: Over all optimization problems ϕ over finite domains, the sum of the probabilities to reach a certain objective value y after m steps with algorithm A is the same as with algorithm B

$$\sum_{\forall \phi} P(y|\phi, m, A) = \sum_{\forall \phi} P(y|\phi, m, B) \tag{2}$$

No Free Lunch Theorem

- According to the No Free Lunch Theorem, the answers are:

No Free Lunch Theorem

- According to the No Free Lunch Theorem, the answers are:
- Question: Can an optimization algorithm A be better than algorithm B?

No Free Lunch Theorem

- According to the No Free Lunch Theorem, the answers are:
- Question: Can an optimization algorithm A be better than algorithm B? Not for all problems!

No Free Lunch Theorem

- According to the No Free Lunch Theorem, the answers are:
- Question: Can an optimization algorithm A be better than algorithm B? Not for all problems!
- Question: Can an optimization algorithm A be better than a Random Walk?

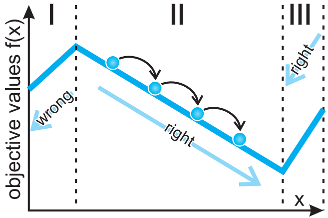
No Free Lunch Theorem

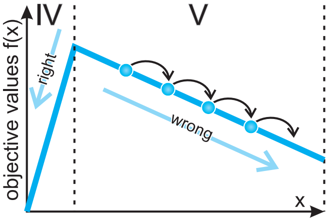
- According to the No Free Lunch Theorem, the answers are:
- Question: Can an optimization algorithm A be better than algorithm B? Not for all problems!
- Question: Can an optimization algorithm A be better than a Random Walk? Not for all problems!

No Free Lunch Theorem

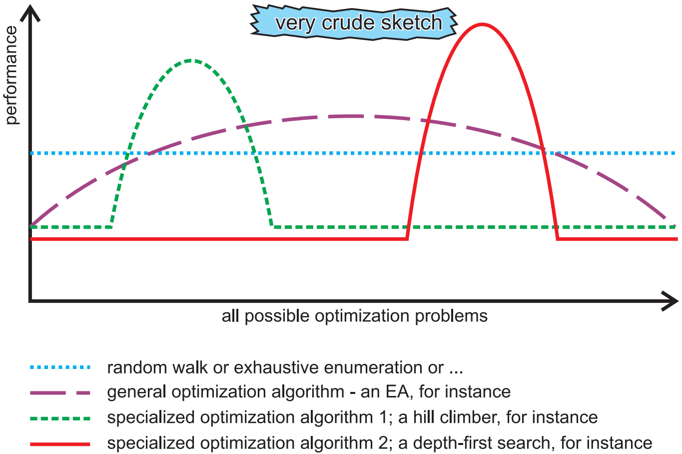
- According to the No Free Lunch Theorem, the answers are:
- Question: Can an optimization algorithm A be better than algorithm B? Not for all problems!
- Question: Can an optimization algorithm A be better than a Random Walk? Not for all problems!
- Put simply: For every problem where method A works well, we can construct a problem where the method does not work

- According to the No Free Lunch Theorem, the answers are:
- Question: Can an optimization algorithm A be better than algorithm B? Not for all problems!
- Question: Can an optimization algorithm A be better than a Random Walk? Not for all problems!
- Put simply: For every problem where method A works well, we can construct a problem where the method does not work



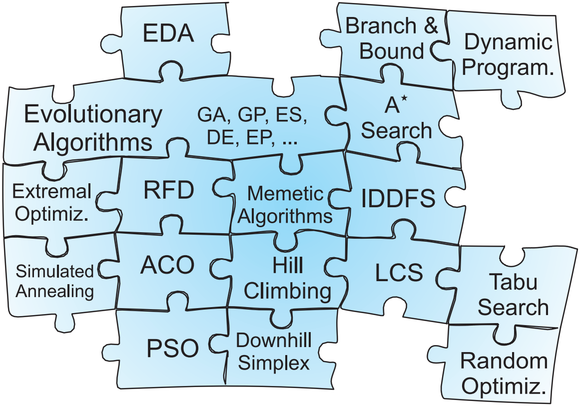


- Different algorithms are good for different problems



No Free Lunch Theorem

- Different algorithms are good for different problems and not all possible problems actually occur in practice [1, 15, 39]



Section Outline

Summary

- Optimization is difficult: This was the first batch of problems
- Many problems can only be solved exactly with algorithms of high complexity
- Metaheuristic optimizers may converge prematurely or non-uniformly
- Ruggedness is not good
- Deceptiveness is not good
- Neutrality can be good or bad
- Epistasis is always bad – and often a representation issue!
- No Free Lunch Theorem

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. Thomas Weise, Raymond Chiong, and Ke Tang. Evolutionary optimization: Pitfalls and booby traps. *Journal of Computer Science and Technology (JCST)*, 27(5):907–936, September 2012. doi: 10.1007/s11390-012-1274-4. Special Issue on Evolutionary Computation, edited by Xin Yao and Pietro S. Oliveto.
2. Donald Ervin Knuth. *Fundamental Algorithms*, volume 1 of *The Art of Computer Programming (TAOCP)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., third edition, 1997. ISBN 0-201-89683-4 and 978-0-201-89683-1. URL http://books.google.de/books?id=J_MySQAACAAJ.
3. Paul Gustav Heinrich Bachmann. *Die Analytische Zahlentheorie / Dargestellt von Paul Bachmann*, volume Zweiter Theil of *Zahlentheorie: Versuch einer Gesamtdarstellung dieser Wissenschaft in ihren Haupttheilen*. Ann Arbor, MI, USA: University of Michigan Library, Scholarly Publishing Office, 1894. ISBN 1418169633, 141818540X, 978-1418169633, and 978-1418185404. URL http://gallica.bnf.fr/ark:/12148/bpt6k994750.
4. Edmund Landau. *Handbuch der Lehre von der Verteilung der Primzahlen*. Leipzig, Germany: B. G. Teubner and Providence, RI, USA: AMS Chelsea Publishing, 1909. ISBN 0821826506 and 9780821826508. URL http://books.google.de/books?id=2ivxUiSogLgC.
5. Alan Mathison Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 1937. doi: 10.1112/plms/s2-42.1.230. URL http://www.thocp.net/biographies/turing_alan.html.
6. Michael Machtey and Paul Young. *An Introduction to the General Theory of Algorithms*, volume 2 of *Theory of Computation, The Computer Science Library*. New York, NY, USA: Elsevier Science Publishing Company, Inc. and Amsterdam, The Netherlands: North-Holland Scientific Publishers Ltd., 1978. ISBN 0-444-00226-X, 0-444-00227-8, and 978-0-444-00227-3. URL http://books.google.de/books?id=qncEAQAAIAAJ.
7. Jon Kleinberg and Christos H. Papadimitriou. Computability and complexity. In USA: Committee on the Fundamentals of Computer Science: Challenges Washington, DC, Computer Science Opportunities, and National Research Council of the National Academies Telecommunications Board, editors, *Computer Science: Reflections on the Field, Reflections from the Field*, chapter 2.1, pages 37–50. Washington, DC, USA: National Academies Press, 2004. URL http://www.cs.cornell.edu/home/kleinber/cstb-turing.pdf.
8. 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.

9. 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.
10. J. Shekel. Test functions for multimodal search techniques. In *Fifth Annual Princeton Conference on Information Science and Systems*, pages 354–359. Princeton, NJ, USA: Princeton University Press, March 1971. URL `http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO_files/Page2354.htm`.
11. Antanas Žilinskas. Algorithm as 133: Optimization of one-dimensional multimodal functions. *Journal of the Royal Statistical Society: Series C – Applied Statistics*, 27(3):367–375, 1978. doi: 10.2307/2347182.
12. 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`.
13. Jeffrey Horn and David Edward Goldberg. Genetic algorithm difficulty and the modality of the fitness landscape. In L. Darrell Whitley and Michael D. Vose, editors, *Proceedings of the Third Workshop on Foundations of Genetic Algorithms (FOGA 3)*, pages 243–269, Estes Park, CO, USA, July 31–August 2, 1994. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.31.3340`.
14. Soraya B. Rana. *Examining the Role of Local Optima and Schema Processing in Genetic Search*. PhD thesis, Fort Collins, CO, USA: Colorado State University, Department of Computer Science, GENITOR Research Group in Genetic Algorithms and Evolutionary Computation, July 1, 1999. URL `http://www.cs.colostate.edu/~genitor/dissertations/rana.ps.gz`.
15. Thomas Weise, Michael Zapf, Raymond Chiong, and Antonio Jesús Nebro Urbaneja. Why is optimization difficult? In Raymond Chiong, editor, *Nature-Inspired Algorithms for Optimisation*, volume 193/2009 of *Studies in Computational Intelligence*, chapter 1, pages 1–50. Berlin/Heidelberg: Springer-Verlag, April 30, 2009. doi: 10.1007/978-3-642-00267-0_1.
16. Larry J. Eshelman, Richard A. Caruana, and James David Schaffer. Biases in the crossover landscape. In James David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms (ICGA'89)*, pages 10–19, Fairfax, VA, USA: George Mason University (GMU), June 4–7, 1989. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.
17. John Henry Holland. Genetic algorithms – computer programs that "evolve" in ways that resemble natural selection can solve complex problems even their creators do not fully understand. *Scientific American*, 267(1):44–50, July 1992. URL `http://www2.econ.iastate.edu/tesfatsi/holland.gaintro.htm`.

18. Ágoston E. Eiben and C. A. Schippers. On evolutionary exploration and exploitation. *Fundamenta Informaticae – Annales Societatis Mathematicae Polonae, Series IV*, 35(1-2):35–50, July–August 1998. URL http://www.cs.vu.nl/~gusz/papers/FunInf98-Eiben-Schippers.ps.
19. Nitin Muttil and Shie-Yui Liong. Superior exploration–exploitation balance in shuffled complex evolution. *Journal of Hydraulic Engineering*, 130(12):1202–1205, December 2004. doi: 10.1061/(ASCE)0733-9429(2004)130:12(1202).
20. Heni Ben Amor and Achim Rettinger. Intelligent exploration for genetic algorithms: Using self-organizing maps in evolutionary computation. In Hans-Georg Beyer, Una-May O'Reilly, Dirk V. Arnold, Wolfgang Banzhaf, Christian Blum, Eric W. Bonabeau, Erick Cantú-Paz, Dipankar Dasgupta, Kalyanmoy Deb, James A. Foster, Edwin D. de Jong, Hod Lipson, Xavier Llorà, Spiros Mancoridis, Martin Pelikan, Günther R. Raidl, Terence Soule, Jean-Paul Watson, and Eckart Zitzler, editors, *Proceedings of Genetic and Evolutionary Computation Conference (GECCO'05)*, pages 1531–1538, Washington, DC, USA: Loews L'Enfant Plaza Hotel, June 25–27, 2005. New York, NY, USA: ACM Press. doi: 10.1145/1068009.1068250.
21. Kalyanmoy Deb. Genetic algorithms for optimization. KanGAL Report 2001002, Kanpur, Uttar Pradesh, India: Kanpur Genetic Algorithms Laboratory (KanGAL), Department of Mechanical Engineering, Indian Institute of Technology Kanpur (IIT), 2001. URL http://www.iitk.ac.in/kangal/papers/isna.ps.gz.
22. Fred W. Glover. Tabu search – part ii. *ORSA Journal on Computing*, 2(1):190–206, Winter 1990. doi: 10.1287/ijoc.2.1.4. URL http://leeds-faculty.colorado.edu/glover/TS%20-%20Part%20II-ORSA-aw.pdf.
23. Fred W. Glover, Éric D. Taillard, and Dominique de Werra. A user's guide to tabu search. *Annals of Operations Research*, 41(1):3–28, March 1993. doi: 10.1007/BF02078647. Special issue on Tabu search.
24. Liviu Badea and Monica Stanciu. Refinement operators can be (weakly) perfect. In Sašo Džeroski and Peter A. Flach, editors, *Proceedings of 9th International Workshop on Inductive Logic Programming (ILP-99)*, volume 1634/1999 of *Lecture Notes in Computer Science (LNCS)*, pages 21–32, Bled, Slovenia, June 24–27, 1999. Berlin, Germany: Springer-Verlag GmbH. doi: 10.1007/3-540-48751-4_4. URL http://www.ai.ici.ro/papers/ilp99.ps.gz.
25. Hendrik Skubch. Hierarchical strategy learning for flux agents. Master's thesis, Dresden, Sachsen, Germany: Technische Universität Dresden, February 18, 2007.
26. Hendrik Skubch. *Hierarchical Strategy Learning for FLUX Agents: An Applied Technique*. Saarbrücken, Saarland, Germany: VDM Verlag Dr. Müller AG und Co. KG, 2006. ISBN 3-8364-5271-5 and 978-3-8364-5271-7. URL http://books.google.de/books?id=syxkPQAACAAJ.

Bibliography IV

27. Paola Festa and Mauricio G.C. Resende. An annotated bibliography of grasp. AT&T Labs Research Technical Report TD-5WYSEW, Florham Park, NJ, USA: AT&T Labs, February 29, 2004. URL http://www.research.att.com/~mgcr/grasp/gannbib/gannbib.html.
28. Thomas A. Feo and Mauricio G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133, March 1995. doi: 10.1007/BF01096763. URL http://www.research.att.com/~mgcr/doc/gtut.ps.Z.
29. Tianshi Chen, Jun He, Guoliang Chen, and Xin Yao. Choosing selection pressure for wide-gap problems. *Theoretical Computer Science*, 411(6):926–934, February 6, 2010. doi: 10.1016/j.tcs.2009.12.014. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.158.8889.
30. Jun He and Xin Yao. From an individual to a population: An analysis of the first hitting time of population-based evolutionary algorithms. *IEEE Transactions on Evolutionary Computation (IEEE-EC)*, 6(5):495–511, October 2002. doi: 10.1109/TEVC.2002.800886. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.148.2275.
31. Tianshi Chen, Ke Tang, Guoliang Chen, and Xin Yao. A large population size can be unhelpful in evolutionary algorithms. *Theoretical Computer Science*, 436:54–70, June 8, 2012. doi: 10.1016/j.tcs.2011.02.016.
32. John Henry Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. Ann Arbor, MI, USA: University of Michigan Press, 1975. ISBN 0-472-08460-7 and 978-0-472-08460-9. URL http://books.google.de/books?id=JE5RAAAAMAAJ.
33. Kalyanmoy Deb. Genetic algorithms for multimodal function optimization. Master's thesis, Tuscaloosa: Clearinghouse for Genetic Algorithms, University of Alabama, 1989. TCGA Report No. 89002.
34. David Edward Goldberg and Jon T. Richardson. Genetic algorithms with sharing for multimodal function optimization. In John J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms and their Applications (ICGA'87)*, pages 41–49, Cambridge, MA, USA: Massachusetts Institute of Technology (MIT), July 28–31, 1987. Mahwah, NJ, USA: Lawrence Erlbaum Associates, Inc. (LEA).
35. Kalyanmoy Deb and David Edward Goldberg. An investigation of niche and species formation in genetic function optimization. In James David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms (ICGA'89)*, pages 42–50, Fairfax, VA, USA: George Mason University (GMU), June 4–7, 1989. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.
36. Bruno Sareni and Laurent Krähenbühl. Fitness sharing and niching methods revisited. *IEEE Transactions on Evolutionary Computation (IEEE-EC)*, 2(3):97–106, September 1998. doi: 10.1109/4235.735432. URL http://hal.archives-ouvertes.fr/hal-00359799/en/.

37. Brad L. Miller and Michael J. Shaw. Genetic algorithms with dynamic niche sharing for multimodal function optimization. IlliGAL Report 95010, Urbana-Champaign, IL, USA: University of Illinois at Urbana-Champaign, Department of Computer Science, Department of General Engineering, Illinois Genetic Algorithms Laboratory (IlliGAL), December 1, 1995. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.52.3568`.
38. David Edward Goldberg, Kalyanmoy Deb, and Jeffrey Horn. Massive multimodality, deception, and genetic algorithms. In Reinhard Männer and Bernard Manderick, editors, *Proceedings of Parallel Problem Solving from Nature 2 (PPSN II)*, pages 37–48, Brussels, Belgium, September 28–30, 1992. Amsterdam, The Netherlands: Elsevier Science Publishers B.V. and Amsterdam, The Netherlands: North-Holland Scientific Publishers Ltd.
39. Thomas Weise. *Global Optimization Algorithms – Theory and Application*. Germany: it-weise.de (self-published), 2009. URL `http://www.it-weise.de/projects/book.pdf`.
40. Thomas Weise, Alexander Podlich, and Christian Gorldt. Solving real-world vehicle routing problems with evolutionary algorithms. In Raymond Chiong and Sandeep Dhakal, editors, *Natural Intelligence for Scheduling, Planning and Packing Problems*, volume 250 of *Studies in Computational Intelligence*, chapter 2, pages 29–53. Berlin/Heidelberg: Springer-Verlag, October 2009. doi: 10.1007/978-3-642-04039-9_2.
41. Thomas Weise, Alexander Podlich, Kai Reinhard, Christian Gorldt, and Kurt Geihs. Evolutionary freight transportation planning. In Mario Giacobini, Penousal Machado, Anthony Brabazon, Jon McCormack, Stefano Cagnoni, Michael O'Neill, Gianni A. Di Caro, Ferrante Neri, Anikó Ekárt, Mike Preuß, Anna Isabel Esparcia-Alcázar, Franz Rothlauf, Muddassar Farooq, Ernesto Tarantino, Andreas Fink, and Shengxiang Yang, editors, *Applications of Evolutionary Computing – Proceedings of EvoWorkshops 2009: EvoCOMNET, EvoENVIRONMENT, EvoFIN, EvoGAMES, EvoHOT, EvoIASP, EvoINTERACTION, EvoMUSART, EvoNUM, EvoSTOC, EvoTRANSLOG (EvoWorkshops'09)*, volume 5484/2009 of *Lecture Notes in Computer Science (LNCS)*, pages 768–777, Tübingen, Germany: Eberhard-Karls-Universität Tübingen, Fakultät für Informations- und Kognitionswissenschaften, April 15–17, 2009. Berlin, Germany: Springer-Verlag GmbH. doi: 10.1007/978-3-642-01129-0_87.
42. Alan Pétrowski. A clearing procedure as a niching method for genetic algorithms. In Keisoku Jidō and Seigyo Gakkai, editors, *Proceedings of IEEE International Conference on Evolutionary Computation (CEC'96)*, pages 798–803, Nagoya, Aichi, Japan: Nagoya University, Symposium & Toyoda Auditorium, 1996. Los Alamitos, CA, USA: IEEE Computer Society Press. doi: 10.1109/ICEC.1996.542703. URL `http://sci2s.ugr.es/docencia/cursoMieres/clearing-96.pdf`.

Bibliography VI

43. Alan Pétrowski. An efficient hierarchical clustering technique for speciation. Technical report, Evry Cedex, France: Institut National des Télécommunications, 1997. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.54.1131.
44. Simon Ronald, John Asenstorfer, and Millist Vincent. Representational redundancy in evolutionary algorithms. In *Second IEEE International Conference on Evolutionary Computation (CEC'95)*, volume 2, pages 631–637, Perth, WA, Australia: University of Western Australia, November 29–December 1, 1995. Los Alamitos, CA, USA: IEEE Computer Society Press. doi: 10.1109/ICEC.1995.487457.
45. Paul J. Darwen and Xin Yao. Every niching method has its niche: Fitness sharing and implicit sharing compared. In Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberg, and Hans-Paul Schwefel, editors, *Proceedings of the 4th International Conference on Parallel Problem Solving from Nature (PPSN IV)*, volume 1141/1996 of *Lecture Notes in Computer Science (LNCS)*, pages 398–407, Berlin, Germany, September 22–24, 1996. Berlin, Germany: Springer-Verlag GmbH. doi: 10.1007/3-540-61723-X_1004. URL sclab.yonsei.ac.kr/courses/03EC/darwen96every.pdf.
46. Thomas Weise, Stefan Niemczyk, Raymond Chiong, and Mingxu Wan. A framework for multi-model edas with model recombination. In *Proceedings of the 4th European Event on Bio-Inspired Algorithms for Continuous Parameter Optimisation (EvoNUM'11), Applications of Evolutionary Computation – Proceedings of EvoApplications 2011: EvoCOMPLEX, EvoGAMES, EvoIASP, EvoINTELLIGENCE, EvoNUM, and EvoSTOC, Part 1 (EvoAPPLICATIONS'11)*, volume 6624 of *Lecture Notes in Computer Science (LNCS)*, pages 304–313, Torino, Italy, April 27–29, 2011. Berlin, Germany: Springer-Verlag GmbH. doi: 10.1007/978-3-642-20525-5_31.
47. Stefan Niemczyk and Thomas Weise. A general framework for multi-model estimation of distribution algorithms. Technical report, Kassel, Hesse, Germany: University of Kassel, Fachbereich 16: Elektrotechnik/Informatik, Distributed Systems Group, March 10, 2010.
48. David Wallin and Conor Ryan. Maintaining diversity in edas for real-valued optimisation problems. In *Frontiers in the Convergence of Bioscience and Information Technologies (FBIT'07)*, pages 795–800, Jeju City, South Korea, October 11–13, 2007. Piscataway, NJ, USA: IEEE (Institute of Electrical and Electronics Engineers). doi: 10.1109/FBIT.2007.132.
49. Teresa Miquélez, Endika Bengoetxea, and Pedro Larrañaga. Evolutionary computation based on bayesian classifiers. *International Journal of Applied Mathematics and Computer Science (AMCS)*, 14(3):335–349, September 2004. URL http://www.amcs.uz.zgora.pl/?action=get_file&file=AMCS_2004_14_3_4.pdf.

Bibliography VII

50. Quiang Lu and Xin Yao. Clustering and learning gaussian distribution for continuous optimization. *IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applications and Reviews*, 35(2):195–204, May 2005. doi: 10.1109/TSMCC.2004.841914. URL http://www.cs.bham.ac.uk/~xin/papers/published_IEEETSMC_LuYa05.pdf.
51. Michaël Defoin Platel, Stefan Schliebs, and Nikola Kasabov. Quantum-inspired evolutionary algorithm: A multimodel eda. *IEEE Transactions on Evolutionary Computation (IEEE-EC)*, 13(6):1218–1232, December 2009. doi: 10.1109/TEVC.2008.2003010.
52. Marcus Gallagher, Marcus R. Frean, and Tom Downs. Real-valued evolutionary optimization using a flexible probability density estimator. In Wolfgang Banzhaf, Jason M. Daida, Ágoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark J. Jakiela, and Robert Elliott Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'99)*, pages 840–846, Orlando, FL, USA, July 13–17, 1999. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.35.5113.
53. Shumeet Baluja. Population-based incremental learning – a method for integrating genetic search based function optimization and competitive learning. Technical Report CMU-CS-94-163, Pittsburgh, PA, USA: Carnegy Mellon University (CMU), School of Computer Science, Computer Science Department, June 2, 1994. URL http://www.ri.cmu.edu/pub_files/pub1/baluja_shumeet_1994_2/baluja_shumeet_1994_2.pdf.
54. Chang Wook Ahn and Rudrapatna S. Ramakrishna. Clustering-based probabilistic model fitting in estimation of distribution algorithms. *IEICE Transactions on Information and Systems, Oxford Journals*, E89-D(1):381–383, January 2006. doi: 10.1093/ietisy/e89-d.1.381.
55. Aizeng Cao, Yueting Chen, Jun Wei, and Jinping Li. A hybrid evolutionary algorithm based on edas and clustering analysis. In Dàizhǎn Chéng and Mǐn Wú, editors, *Proceedings of the 26th Chinese Control Conference (CCC'07)*, pages 754–758, Zhangjiajie, Hunan, China, July 26–31, 2007. Piscataway, NJ, USA: IEEE (Institute of Electrical and Electronics Engineers). doi: 10.1109/CHICC.2006.4347236.
56. Tatsuya Okabe, Yaochu Jin, Bernhard Sendhoff, and Markus Olhofer. Voronoi-based estimation of distribution algorithm for multi-objective optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'04)*, volume 2, pages 1594–1601, Portland, OR, USA, June 20–23, 2004. Los Alamitos, CA, USA: IEEE Computer Society Press. doi: 10.1109/CEC.2004.1331086. URL http://www.soft-computing.de/cec-2004-1445-final.pdf.

57. Kumara Sastry and David Edward Goldberg. Multiobjective hboa, clustering, and scalability. In Hans-Georg Beyer, Una-May O'Reilly, Dirk V. Arnold, Wolfgang Banzhaf, Christian Blum, Eric W. Bonabeau, Erick Cantú-Paz, Dipankar Dasgupta, Kalyanmoy Deb, James A. Foster, Edwin D. de Jong, Hod Lipson, Xavier Llorà, Spiros Mancoridis, Martin Pelikan, Günther R. Raidl, Terence Soule, Jean-Paul Watson, and Eckart Zitzler, editors, *Proceedings of Genetic and Evolutionary Computation Conference (GECCO'05)*, pages 663–670, Washington, DC, USA: Loews L'Enfant Plaza Hotel, June 25–27, 2005. New York, NY, USA: ACM Press. doi: 10.1145/1068009.1068122.
58. Günter Rudolph. Self-adaptation and global convergence: A counter-example. In Peter John Angeline, Zbigniew Michalewicz, Marc Schoenauer, Xin Yao, and Ali M. S. Zalzala, editors, *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'99)*, volume 1, pages 646–651, Washington, DC, USA: Mayflower Hotel, July 6–9, 1999. Piscataway, NJ, USA: IEEE Computer Society. doi: 10.1109/CEC.1999.781994. URL `http://ls11-www.cs.uni-dortmund.de/people/rudolph/publications/papers/CEC99.pdf`.
59. Günter Rudolph. Self-adaptive mutations may lead to premature convergence. *IEEE Transactions on Evolutionary Computation (IEEE-EC)*, 5(4):410–414, 2001. doi: 10.1109/4235.942534.
60. Darrell F. Lochtefeld and Frank William Ciarallo. Multiobjectivization via helper-objectives with the tunable objectives problem. *IEEE Transactions on Evolutionary Computation (IEEE-EC)*, 16(3):373–395, June 2012. doi: 10.1109/TEVC.2011.2136345.
61. Darrell F. Lochtefeld and Frank William Ciarallo. Helper-objective optimization strategies for the job-shop scheduling problem. *Applied Soft Computing*, 11(6):4161–4174, September 2011. doi: 10.1016/j.asoc.2011.03.007.
62. Joshua D. Knowles, Richard A. Watson, and David Wolfe Corne. Reducing local optima in single-objective problems by multi-objectivization. In Eckart Zitzler, Kalyanmoy Deb, Lothar Thiele, Carlos Artemio Coello Coello, and David Wolfe Corne, editors, *Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization (EMO'01)*, volume 1993/2001 of *Lecture Notes in Computer Science (LNCS)*, pages 269–283, Zürich, Switzerland: Eidgenössische Technische Hochschule (ETH) Zürich, March 7–9, 2001. Berlin, Germany: Springer-Verlag GmbH. doi: 10.1007/3-540-44719-9_19. URL `http://www.macs.hw.ac.uk/~dwcorne/rlo.pdf`.
63. Mikkel T. Jensen. Guiding single-objective optimization using multi-objective methods. In Günther R. Raidl, Jean-Arcady Meyer, Martin Middendorf, Stefano Cagnoni, Juan Jesús Romero Cardalda, David Wolfe Corne, Jens Gottlieb, Agnès Guillot, Emma Hart, Colin G. Johnson, and Elena Marchiori, editors, *Applications of Evolutionary Computing, Proceedings of EvoWorkshop 2003: EvoBIO, EvoCOP, EvoIASP, EvoMUSART, EvoROB, and EvoSTIM (EvoWorkshop'03)*, volume 2611/2003 of *Lecture Notes in Computer Science (LNCS)*, pages 91–98, Wivenhoe Park, Colchester, Essex, UK: University of Essex, April 14–16, 2003. Berlin, Germany: Springer-Verlag GmbH. doi: 10.1007/3-540-36605-9_25.

Bibliography IX

64. Mikkel T. Jensen. Helper-objectives: Using multi-objective evolutionary algorithms for single-objective optimisation. *Journal of Mathematical Modelling and Algorithms*, 3(4):323–347, December 2004. doi: 10.1023/B:JMMA.0000049378.57591.c6.
65. Frank Neumann and Ingo Wegener. Can single-objective optimization profit from multiobjective optimization? In *Multiobjective Problem Solving from Nature – From Concepts to Applications*, Natural Computing Series, pages 115–130. New York, NY, USA: Springer New York, 2008. doi: 10.1007/978-3-540-72964-8_6.
66. Martin Jähne, Xiaodong Li, and Jürgen Branke. Evolutionary algorithms and multi-objectivization for the travelling salesman problem. In Franz Rothlauf, Günther R. Raidl, Anna Isabel Esparcia-Alcázar, Ying-Ping Chen, Gabriela Ochoa, Ender Ozcan, Marc Schoenauer, Anne Auger, Hans-Georg Beyer, Nikolaus Hansen, Steffen Finck, Raymond Ros, L. Darrell Whitley, Garnett Wilson, Simon Harding, William Benjamin Langdon, Man Leung Wong, Laurence D. Merkle, Frank W. Moore, Sevan G. Ficici, William Rand, Rick L. Riolo, Nawwaf Kharma, William R. Buckley, Julian Francis Miller, Kenneth Owen Stanley, Jaume Bacardit i Peñarroya, Will N. Browne, Jan Drugowitsch, Nicola Beume, Mike Preuß, Stephen Frederick Smith, Stefano Cagnoni, Alexandru Floares, Aaron Baughman, Steven Matt Gustafson, Maarten Keijzer, Arthur Kordon, and Clare Bates Congdon, editors, *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation (GECCO'09)*, pages 595–602, Montréal, QC, Canada: Delta Centre-Ville Hotel, July 8–12, 2009. New York, NY, USA: Association for Computing Machinery (ACM). doi: 10.1145/1569901.1569984. URL http://goanna.cs.rmit.edu.au/~xiaodong/publications/multi-objectivization-jahne-gecco09.pdf.
67. Darrell F. Lochtefeld and Frank William Ciarallo. Deterministic helper objective sequence applied to the job-shop scheduling problem. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'10)*, pages 431–438, Portland, OR, USA: Portland Marriott Downtown Waterfront Hotel, July 7–11, 2010. New York, NY, USA: ACM Press. doi: 10.1145/1830483.1830566.
68. Erik van Nimwegen and James P. Crutchfield. Optimizing epochal evolutionary search: Population-size dependent theory. *Machine Learning*, 45(1):77–114, October 2001. doi: 10.1023/A:1012497308906.
69. Ingo Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. PhD thesis, Berlin, Germany: Technische Universität Berlin, 1971. URL http://books.google.de/books?id=QcNNGQAACAAJ.
70. Ingo Rechenberg. *Evolutionsstrategie '94*, volume 1 of *Werkstatt Bionik und Evolutionstechnik*. Bad Cannstadt, Stuttgart, Baden-Württemberg, Germany: Frommann-Holzboog Verlag, 1994. ISBN 3-7728-1642-8 and 978-3-772-81642-0. URL http://books.google.de/books?id=savAAAACAAJ.

Bibliography X

71. Justinian P. Rosca and Dana H. Ballard. Causality in genetic programming. In Larry J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA'95)*, pages 256–263, Pittsburgh, PA, USA: University of Pittsburgh, July 15–19, 1995. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.26.2503.
72. Jean-Baptiste Pierre Antoine de Monet, Chevalier de Lamarck. *Philosophie zoologique – ou Exposition des considérations relatives à l'histoire naturelle des Animaux; à la diversité de leur organisation et des facultés qu'ils en obtiennent; ....* Paris, France: Dentu and Paris, France: J. B. Baillière Liberaire, 1809. ISBN 1412116465 and 9781412116466. URL http://www.lamarck.cnrs.fr/ice/ice_book_detail.php?type=text&bdd=lamarck&table=ouvrages_lamarck&bookId=29. Philosophie zoologique – ou Exposition des considérations relatives à l'histoire naturelle des Animaux; à la diversité de leur organisation et des facultés qu'ils en obtiennent; aux causes physiques qui maintiennent en eux la vie et donnent lieu aux mouvements qu'ils exécutant; enfin, à celles qui produisent les unes le sentiment, et les autres l'intelligence de ceux qui en sont doués.
73. L. Darrell Whitley, V. Scott Gordon, and Keith E. Mathias. Lamarckian evolution, the baldwin effect and function optimization. In Yuval Davidor, Hans-Paul Schwefel, and Reinhard Männer, editors, *Proceedings of the Third Conference on Parallel Problem Solving from Nature; International Conference on Evolutionary Computation (PPSN III)*, volume 866/1994 of *Lecture Notes in Computer Science (LNCS)*, pages 5–15, Jerusalem, Israel, October 9–14, 1994. Berlin, Germany: Springer-Verlag GmbH. doi: 10.1007/3-540-58484-6_245. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.18.2428.
74. James Mark Baldwin. A new factor in evolution. *The American Naturalist*, 30(354):441–451, June 1896. URL http://www.brocku.ca/MeadProject/Baldwin/Baldwin_1896_h.html.
75. Geoffrey E. Hinton and Steven J. Nowlan. How learning can guide evolution. *Complex Systems*, 1(3):495–502, 1987. URL http://htpprints.yorku.ca/archive/00000172/.
76. Geoffrey E. Hinton and Steven J. Nowlan. How learning can guide evolution. In Richard K. Belew and Melanie Mitchell, editors, *Adaptive Individuals in Evolving Populations: Models and Algorithms*, volume 26 of *Santa Fe Institue Studies in the Sciences of Complexity*, pages 447–454. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. and Boulder, CO, USA: Westview Press, January 15, 1996.
77. Pablo Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Caltech Concurrent Computation Program C3P 826, Pasadena, CA, USA: California Institute of Technology (Caltech), Caltech Concurrent Computation Program (C3P), 1989. URL http://www.each.usp.br/sarajane/SubPaginas/arquivos_aulas_IA/memetic.pdf.

Bibliography XI

78. Michael G. Norman and Pablo Moscato. A competitive and cooperative approach to complex combinatorial search. In *Proceedings of the 20th Informatics and Operations Research Meeting (20th Jornadas Argentinas e Informática e Investigación Operativa) (JAIIO'91)*, pages 3.15–3.29, Buenos Aires, Argentina: Centro Cultural General San Martín, August 20–23, 1991. Also published as Technical Report Caltech Concurrent Computation Program, Report. 790, California Institute of Technology, Pasadena, California, USA, 1989.
79. Diana Holstein and Pablo Moscato. Memetic algorithms using guided local search: A case study. In David Wolfe Corne, Marco Dorigo, Fred W. Glover, Dipankar Dasgupta, Pablo Moscato, Riccardo Poli, and Kenneth V. Price, editors, *New Ideas in Optimization*, McGraw-Hill's Advanced Topics In Computer Science Series, pages 235–244. Maidenhead, England, UK: McGraw-Hill Ltd., May 1999.
80. Luciana Buriol, Paulo M. França, and Pablo Moscato. A new memetic algorithm for the asymmetric traveling salesman problem. *Journal of Heuristics*, 10(5):483–506, September 2004. doi: 10.1023/B:HEUR.0000045321.59202.52. URL `http://www.springerlink.com/content/w617486q60mphg88/fulltext.pdf`.
81. Maria J. Blesa, Pablo Moscato, and Fatos Xhafa. A memetic algorithm for the minimum weighted k-cardinality tree subgraph problem. In Mauricio G.C. Resende, Jorge Pinho de Sousa, and Ana Viana, editors, *4th Metaheuristics International Conference – Metaheuristics: Computer Decision-Making (MIC'01)*, volume 86 of *Applied Optimization*, pages 85–90, Porto, Portugal, July 16–20, 2001. Berlin, Germany: Springer-Verlag GmbH. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.21.5376`.
82. Pablo Moscato and Carlos Cotta. A gentle introduction to memetic algorithms. In Fred W. Glover and Gary A. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, chapter 5, pages 105–144. Norwell, MA, USA: Kluwer Academic Publishers, Dordrecht, Netherlands: Springer Netherlands, and Boston, MA, USA: Springer US, 2003. doi: 10.1007/0-306-48056-5_5. URL `http://www.lcc.uma.es/~ccottap/papers/handbook03memetic.pdf`.
83. Nicholas J. Radcliffe and Patrick David Surry. Formal memetic algorithms. In Terence Claus Fogarty, editor, *Proceedings of the Workshop on Artificial Intelligence and Simulation of Behaviour, International Workshop on Evolutionary Computing, Selected Papers (AISB'94)*, volume 865/1994 of *Lecture Notes in Computer Science (LNCS)*, pages 1–16, Leeds, UK, April 11–13, 1994. Chichester, West Sussex, UK: Society for the Study of Artificial Intelligence and the Simulation of Behaviour (SSAISB), Berlin, Germany: Springer-Verlag GmbH. doi: 10.1007/3-540-58483-8_1. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.38.9885`.

84. Jason Digalakis and Konstantinos Margaritis. A parallel memetic algorithm for solving optimization problems. In Mauricio G.C. Resende, Jorge Pinho de Sousa, and Ana Viana, editors, *4th Metaheuristics International Conference – Metaheuristics: Computer Decision-Making (MIC'01)*, volume 86 of *Applied Optimization*, pages 121–125, Porto, Portugal, July 16–20, 2001. Berlin, Germany: Springer-Verlag GmbH. URL `http://citeseer.ist.psu.edu/digalakis01parallel.html`.
85. Jason Digalakis and Konstantinos Margaritis. Performance comparison of memetic algorithms. *Journal of Applied Mathematics and Computation*, 158:237–252, October 2004. doi: 10.1016/j.amc.2003.08.115. URL `http://www.complexity.org.au/ci/draft/draft/digala02/digala02s.pdf`.
86. Natalio Krasnogor and James E. Smith. A tutorial for competent memetic algorithms: Model, taxonomy, and design issues. *IEEE Transactions on Evolutionary Computation (IEEE-EC)*, 9(5):474–488, October 2005. doi: 10.1109/TEVC.2005.850260. URL `http://www.cs.nott.ac.uk/~nxk/PAPERS/IEEE-TEC-lastVersion.pdf`.
87. David H. Ackley. *A Connectionist Machine for Genetic Hillclimbing*. PhD thesis, Pittsburgh, PA, USA: Carnegy Mellon University (CMU), August 31, 1987. URL `http://books.google.de/books?id=3ttfAAAAMAAJ`.
88. David Edward Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989. ISBN 0-201-15767-5 and 978-0-201-15767-3. URL `http://books.google.de/books?id=2IIJAAAACAAJ`.
89. Martina Gorges-Schleuter. Asparagos: An asynchronous parallel genetic optimization strategy. In James David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms (ICGA'89)*, pages 422–427, Fairfax, VA, USA: George Mason University (GMU), June 4–7, 1989. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.
90. Heinz Mühlenbein. Parallel genetic algorithms, population genetics and combinatorial optimization. In Jörg D. Becker, Ignaz Eisele, and F. W. Mündemann, editors, *Parallelism, Learning, Evolution: Workshop on Evolutionary Models and Strategies (Neubiberg, Germany, 1989-03-10/11) and Workshop on Parallel Processing: Logic, Organization, and Technology (Wildbad Kreuth, Germany, 1989-07-24 to 28) (WOPPLOT'89)*, volume 565/1991 of *Lecture Notes in Computer Science (LNCS)*, pages 398–406, Germany, 1989. Berlin, Germany: Springer-Verlag GmbH. doi: 10.1007/3-540-55027-5_23.
91. Heinz Mühlenbein. Parallel genetic algorithms, population genetics and combinatorial optimization. In James David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms (ICGA'89)*, pages 416–421, Fairfax, VA, USA: George Mason University (GMU), June 4–7, 1989. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

92. Heinz Mühlenbein. How genetic algorithms really work – i. mutation and hillclimbing. In Reinhard Männer and Bernard Manderick, editors, *Proceedings of Parallel Problem Solving from Nature 2 (PPSN II)*, pages 15–26, Brussels, Belgium, September 28–30, 1992. Amsterdam, The Netherlands: Elsevier Science Publishers B.V. and Amsterdam, The Netherlands: North-Holland Scientific Publishers Ltd. URL http://muehlenbein.org/mut92.pdf.
93. Donald E. Brown, Christopher L. Huntley, and Andrew R. Spillane. A parallel genetic heuristic for the quadratic assignment problem. In James David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms (ICGA'89)*, pages 406–415, Fairfax, VA, USA: George Mason University (GMU), June 4–7, 1989. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.
94. Lawrence Davis, editor. *Handbook of Genetic Algorithms*. VNR Computer Library. Stamford, CT, USA: Thomson Publishing Group, Inc. and New York, NY, USA: Van Nostrand Reinhold Co., January 1991. ISBN 0-442-00173-8, 1850328250, 978-0-442-00173-5, and 978-1850328254. URL http://books.google.de/books?id=vTG5PAAACAAJ.
95. Ko-Hsin Liang, Xin Yao, and Charles S. Newton. Evolutionary search of approximated n-dimensional landscapes. *International Journal of Knowledge-Based and Intelligent Engineering Systems*, 4(3):172–183, June 2000. URL http://sclab.yonsei.ac.kr/courses/03EC/liang00evolutionary.pdf.
96. Yu Wang, Bin Li, and Thomas Weise. Estimation of distribution and differential evolution cooperation for large scale economic load dispatch optimization of power systems. *Information Sciences – Informatics and Computer Science Intelligent Systems Applications: An International Journal*, 180(12):2405–2420, June 2010. doi: 10.1016/j.ins.2010.02.015.
97. Albert Donally Bethke. *Genetic Algorithms as Function Optimizers*. PhD thesis, Ann Arbor, MI, USA: University of Michigan, 1980. National Science Foundation Grant No. MCS76-04297.
98. Gunar E. Liepins and Michael D. Vose. Deceptiveness and genetic algorithm dynamics. In Bruce M. Spatz and Gregory J. E. Rawlins, editors, *Proceedings of the First Workshop on Foundations of Genetic Algorithms (FOGA'90)*, pages 36–50, Bloomington, IN, USA: Indiana University, Bloomington Campus, July 15–18, 1990. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. URL http://www.osti.gov/bridge/servlets/purl/6445602-CfqU6M/.
99. Thorsten Schnier and Xin Yao. Using multiple representations in evolutionary algorithms. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'00)*, volume 1, pages 479–486, La Jolla, CA, USA: La Jolla Marriott Hotel, July 16–19, 2000. Piscataway, NJ, USA: IEEE Computer Society. doi: 10.1109/CEC.2000.870335. URL http://www.cs.bham.ac.uk/~txs/publications/2000/SchnierYaoCEC2000.pdf.

100. Shane Legg, Marcus Hutter, and Akshat Kumar. Tournament versus fitness uniform selection. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'04)*, pages 2144–2151, Portland, OR, USA, June 20–23, 2004. Los Alamitos, CA, USA: IEEE Computer Society Press. doi: 10.1109/CEC.2004.1331162. URL http://arxiv.org/abs/cs/0403038v1.
101. Marcus Hutter. Fitness uniform selection to preserve genetic diversity. In David B. Fogel, Mohamed A. El-Sharkawi, Xin Yao, Hitoshi Iba, Paul Marrow, and Mark Shackleton, editors, *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'02), 2002 IEEE World Congress on Computation Intelligence (WCCI'02)*, volume 1-2, pages 783–788, Honolulu, HI, USA: Hilton Hawaiian Village Hotel (Beach Resort & Spa), May 12–17, 2002. Piscataway, NJ, USA: IEEE Computer Society, Los Alamitos, CA, USA: IEEE Computer Society Press. doi: 10.1109/CEC.2002.1007025. URL http://arxiv.org/abs/cs/0103015. Also: Technical Report IDSIA-01-01, 17 January 2001.
102. Marcus Hutter and Shane Legg. Fitness uniform optimization. *IEEE Transactions on Evolutionary Computation (IEEE-EC)*, 10(5):568–589, October 2006. doi: 10.1109/TEVC.2005.863127. URL http://arxiv.org/abs/cs/0610126v1.
103. Shane Legg and Marcus Hutter. Fitness uniform deletion: A simple way to preserve diversity. In Hans-Georg Beyer, Una-May O'Reilly, Dirk V. Arnold, Wolfgang Banzhaf, Christian Blum, Eric W. Bonabeau, Erick Cantú-Paz, Dipankar Dasgupta, Kalyanmoy Deb, James A. Foster, Edwin D. de Jong, Hod Lipson, Xavier Llorà, Spiros Mancoridis, Martin Pelikan, Günther R. Raidl, Terence Soule, Jean-Paul Watson, and Eckart Zitzler, editors, *Proceedings of Genetic and Evolutionary Computation Conference (GECCO'05)*, pages 1271–1278, Washington, DC, USA: Loews L'Enfant Plaza Hotel, June 25–27, 2005. New York, NY, USA: ACM Press. doi: 10.1145/1068009.1068216. URL http://arxiv.org/abs/cs/0504035v1.
104. Joel Lehman and Kenneth Owen Stanley. Exploiting open-endedness to solve problems through the search for novelty. In Seth Bullock, Jason Noble, Richard A. Watson, and Mark A. Bedau, editors, *Proceedings of the Eleventh International Conference on the Simulation and Synthesis of Living Systems (Artificial Life XI)*, pages 329–336, Winchester, Hampshire, UK, August 5–8, 2008. Cambridge, MA, USA: MIT Press. URL http://eplex.cs.ucf.edu/papers/lehman_alife08.pdf.
105. Joel Lehman and Kenneth Owen Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223, Summer 2011. doi: 10.1162/EVCO_a_00025. URL http://eplex.cs.ucf.edu/papers/lehman_ecj10.pdf.

106. Joel Lehman and Kenneth Owen Stanley. Evolving a diversity of virtual creatures through novelty search and local competition. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'11)*, pages 211–218, Dublin, Ireland, July 12–16, 2011. doi: 10.1145/2001576.2001606. URL http://eplex.cs.ucf.edu/papers/lehman_gecco11.pdf.
107. Lee Altenberg. Fitness distance correlation analysis: An instructive counterexample. In Thomas Bäck, editor, *Proceedings of The Seventh International Conference on Genetic Algorithms (ICGA'97)*, pages 57–64, East Lansing, MI, USA: Michigan State University, July 19–23, 1997. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. URL http://dynamics.org/Altenberg/PAPERS/FDCAAIC/.
108. Lee Altenberg. The schema theorem and price's theorem. In L. Darrell Whitley and Michael D. Vose, editors, *Proceedings of the Third Workshop on Foundations of Genetic Algorithms (FOGA 3)*, pages 23–49, Estes Park, CO, USA, July 31–August 2, 1994. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. URL http://dynamics.org/Altenberg/FILES/LeeSTPT.pdf.
109. Gwoing Tina Yu. Program evolvability under environmental variations and neutrality. In Fernando Almeida e Costa, Luís Mateus Rocha, Ernesto Jorge Fernandes Costa, Inman Harvey, and António Coutinho, editors, *Proceedings of the 9th European Conference on Advances in Artificial Life (ECAL'07)*, volume 4648/2007 of *Lecture Notes in Computer Science (LNCS)*, pages 835–844, Lisbon, Portugal, September 10–14, 2007. Berlin, Germany: Springer-Verlag GmbH. doi: 10.1007/978-3-540-74913-4_84.
110. Andreas Wagner. Robustness, evolvability, and neutrality. *FEBS Letters*, 579(8):1772–1778, March 21, 2005. doi: 10.1016/j.febslet.2005.01.063.
111. Richard Dawkins. The evolution of evolvability. In Christopher Gale Langdon, editor, *The Proceedings of an Interdisciplinary Workshop on the Synthesis and Simulation of Living Systems (Artificial Life'87)*, volume 6 of *Santa Fe Institute Studies in the Sciences of Complexity*, pages 201–220, Los Alamaos, NM, USA: Oppenheimer Study Center, September 21, 1987. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. and Boulder, CO, USA: Westview Press.
112. Yuval Davidor. Epistasis variance: A viewpoint on ga-hardness. In Bruce M. Spatz and Gregory J. E. Rawlins, editors, *Proceedings of the First Workshop on Foundations of Genetic Algorithms (FOGA'90)*, pages 23–35, Bloomington, IN, USA: Indiana University, Bloomington Campus, July 15–18, 1990. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

113. Lee Altenberg. Nk fitness landscapes. In Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz, editors, *Handbook of Evolutionary Computation*, Computational Intelligence Library, chapter B2.7.2. New York, NY, USA: Oxford University Press, Inc., Dirac House, Temple Back, Bristol, UK: Institute of Physics Publishing Ltd. (IOP), and Boca Raton, FL, USA: CRC Press, Inc., January 1, 1997. URL `http://www.cmi.univ-mrs.fr/~pardoux/LeeNKFL.pdf`.
114. Bart Naudts and Alain Verschoren. Epistasis on finite and infinite spaces. In *8th International Conference on Systems Research, Informatics and Cybernetics (InterSymp'96)*, pages 19–23, Baden-Baden, Baden-Württemberg, Germany, August 14–18, 1996. Tecumseh, ON, Canada: International Institute for Advanced Studies in Systems Research and Cybernetic (IIAS). URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.32.6455`.
115. Colin R. Reeves and Christine C. Wright. Epistasis in genetic algorithms: An experimental design perspective. In Larry J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA'95)*, pages 217–224, Pittsburgh, PA, USA: University of Pittsburgh, July 15–19, 1995. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.39.29`.
116. William Bateson. *Mendel's Principles of Heredity*. Kessinger Publishing's® Rare Reprints. Cambridge, UK: Cambridge University Press, 1909. ISBN 1428648194 and 9781428648197. URL `http://books.google.de/books?id=WWZfDQljn8gC`.
117. Sir Ronald Aylmer Fisher. The correlations between relatives on the supposition of mendelian inheritance. *Philosophical Transactions of the Royal Society of Edinburgh*, 52:399–433, October 1, 1918. URL `http://www.library.adelaide.edu.au/digitised/fisher/9.pdf`.
118. Patrick C. Phillips. The language of gene interaction. *Genetics*, 149(3):1167–1171, July 1998. URL `http://www.genetics.org/cgi/reprint/149/3/1167.pdf`.
119. Jay L. Lush. Progeny test and individual performance as indicators of an animal's breeding value. *Journal of Dairy Science (JDS)*, 18(1):1–19, January 1935. URL `http://jds.fass.org/cgi/reprint/18/1/1`.
120. Ke Tang, Xin Yao, Ponnuthurai Nagaratnam Suganthan, Cara MacNish, Ying-Ping Chen, Chih-Ming Chen, and Zhenyu Yang. Benchmark functions for the cec'2008 special session and competition on large scale global optimization. Technical report, Hefei, Anhui, China: University of Science and Technology of China (USTC), School of Computer Science and Technology, Nature Inspired Computation and Applications Laboratory (NICAL), 2007. URL `http://sci2s.ugr.es/programacion/workshop/Tech.Report.CEC2008.LSGO.pdf`.
121. George Hadley. *Nonlinear and Dynamics Programming*. World Student. Reading, MA, USA: Addison-Wesley Professional, December 1964. ISBN 0201026643 and 978-0201026641.

122. Domingo Ortiz-Boyer, César Hervás-Martínez, and Carlos A. Reyes García. Cixl2: A crossover operator for evolutionary algorithms based on population features. *Journal of Artificial Intelligence Research (JAIR)*, 24:1–48, July 2005. URL http://www.cs.cmu.edu/afs/cs/project/jair/pub/volume24/ortizboyer05a-html/Ortiz-Boyer.html.
123. Ke Tang, Xiaodong Li, Ponnuthurai Nagaratnam Suganthan, Zhenyu Yang, and Thomas Weise. Benchmark functions for the cec'2010 special session and competition on large-scale global optimization. Technical report, Hefei, Anhui, China: University of Science and Technology of China (USTC), School of Computer Science and Technology, Nature Inspired Computation and Applications Laboratory (NICAL), January 8, 2010.
124. Thomas Weise and Ke Tang. Evolving distributed algorithms with genetic programming. *IEEE Transactions on Evolutionary Computation (IEEE-EC)*, 16(2):242–265, April 2012. doi: 10.1109/TEVC.2011.2112666.
125. Thomas Weise, Michael Zapf, and Kurt Geihs. Rule-based genetic programming. In *Proceedings of the 2nd International Conference on Bio-Inspired Models of Network, Information, and Computing Systems (BIONETICS'07)*, pages 8–15, Budapest, Hungary: Radisson SAS Beke Hotel, December 10–13, 2007. Piscataway, NJ, USA: IEEE Computer Society. doi: 10.1109/BIMNICS.2007.4610073.
126. Thomas Weise and Michael Zapf. Evolving distributed algorithms with genetic programming: Election. In Lihong Xu, Erik D. Goodman, and Yongsheng Ding, editors, *Proceedings of the First ACM/SIGEVO Summit on Genetic and Evolutionary Computation (GEC'09)*, pages 577–584, Shanghai, China: Hua-Ting Hotel & Towers, June 12–14, 2009. New York, NY, USA: ACM Press. doi: 10.1145/1543834.1543913.
127. Thomas Weise. *Evolving Distributed Algorithms with Genetic Programming*. PhD thesis, Kassel, Hesse, Germany: University of Kassel, Fachbereich 16: Elektrotechnik/Informatik, Distributed Systems Group, May 4, 2009. Won the Dissertation Award of The Association of German Engineers (Verein Deutscher Ingenieure, VDI).
128. Masaya Shinkai, Arturo Hernández Aguirre, and Kiyoshi Tanaka. Mutation strategy improves gas performance on epistatic problems. In David B. Fogel, Mohamed A. El-Sharkawi, Xin Yao, Hitoshi Iba, Paul Marrow, and Mark Shackleton, editors, *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'02), 2002 IEEE World Congress on Computation Intelligence (WCCI'02)*, volume 1, pages 968–973, Honolulu, HI, USA: Hilton Hawaiian Village Hotel (Beach Resort & Spa), May 12–17, 2002. Piscataway, NJ, USA: IEEE Computer Society, Los Alamitos, CA, USA: IEEE Computer Society Press. doi: 10.1109/CEC.2002.1007056.
129. Georges Raif Harik. *Learning Gene Linkage to Efficiently Solve Problems of Bounded Difficulty using Genetic Algorithms*. PhD thesis, Ann Arbor, MI, USA: University of Michigan, 1997. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.54.7092.

130. Masaharu Munetomo and David Edward Goldberg. Linkage identification by non-monotonicity detection for overlapping functions. *Evolutionary Computation*, 7(4):377–398, Winter 1999. doi: 10.1162/evco.1999.7.4.377.
131. Ying-Ping Chen. *Extending the Scalability of Linkage Learning Genetic Algorithms – Theory & Practice*, volume 190/2006 of *Studies in Fuzziness and Soft Computing*. Berlin, Germany: Springer-Verlag GmbH, 2004. ISBN 3-540-28459-1 and 978-3-540-28459-8. doi: 10.1007/b102053. URL http://books.google.de/books?id=kKr3rKhPU7oC.
132. Kalyanmoy Deb, Ankur Sinha, and Saku Kukkonen. Multi-objective test problems, linkages, and evolutionary methodologies. In Maarten Keijzer and Mike Cattolico, editors, *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation (GECCO'06)*, pages 1141–1148, Seattle, WA, USA: Renaissance Seattle Hotel, July 8–12, 2006. New York, NY, USA: ACM Press. doi: 10.1145/1143997.1144179.
133. David Edward Goldberg, Kalyanmoy Deb, and Bradley Korb. Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, 3(5):493–530, 1989. URL http://www.complex-systems.com/pdf/03-5-5.pdf.
134. Martin Pelikan, David Edward Goldberg, and Erick Cantú-Paz. Boa: The bayesian optimization algorithm. In Wolfgang Banzhaf, Jason M. Daida, Ágoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark J. Jakiela, and Robert Elliott Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'99)*, pages 525–532, Orlando, FL, USA, July 13–17, 1999. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. URL https://eprints.kfupm.edu.sa/28537/.
135. Erick Cantú-Paz, Martin Pelikan, and David Edward Goldberg. Linkage problem, distribution estimation, and bayesian networks. *Evolutionary Computation*, 8(3):311–340, Fall 2000. doi: 10.1162/106365600750078808. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.16.236.
136. Wenxiang Chen, Thomas Weise, Zhenyu Yang, and Ke Tang. Large-scale global optimization using cooperative coevolution with variable interaction learning. In Robert Schaefer, Carlos Cotta, Joanna Kołodziej, and Günter Rudolph, editors, *Proceedings of the 11th International Conference on Parallel Problem Solving From Nature, Part 2 (PPSN'10-2)*, volume 6239 of *Lecture Notes in Computer Science (LNCS)*, pages 300–309, Kraków, Poland: AGH University of Science and Technology, September 11–15, 2010. Berlin, Germany: Springer-Verlag GmbH. doi: 10.1007/978-3-642-15871-1_31.

137. Mitchell A. Potter and Kenneth Alan De Jong. A cooperative coevolutionary approach to function optimization. In Yuval Davidor, Hans-Paul Schwefel, and Reinhard Männer, editors, *Proceedings of the Third Conference on Parallel Problem Solving from Nature; International Conference on Evolutionary Computation (PPSN III)*, volume 866/1994 of *Lecture Notes in Computer Science (LNCS)*, pages 249–257, Jerusalem, Israel, October 9–14, 1994. Berlin, Germany: Springer-Verlag GmbH. doi: 10.1007/3-540-58484-6_269. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.54.7033.
138. Mitchell A. Potter and Kenneth Alan De Jong. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary Computation*, 8(1):1–29, Spring 2000. doi: 10.1162/106365600568086. URL http://mitpress.mit.edu/journals/pdf/evco_8_1_1_0.pdf.
139. Richard Ernest Bellman. *Dynamic Programming*. Dover Books on Mathematics. Princeton, NJ, USA: Princeton University Press, 1957. ISBN 0486428095 and 978-0486428093. URL http://books.google.de/books?id=fyVtp3EMxasC.
140. Richard Ernest Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton, NJ, USA: Princeton University Press, 1961. ISBN 0691079013 and 978-0691079011. URL http://books.google.de/books?id=8wY3PAAACAAJ.
141. Conor Ryan, John James Collins, and Michael O'Neill. Grammatical evolution: Evolving programs for an arbitrary language. In Wolfgang Banzhaf, Riccardo Poli, Marc Schoenauer, and Terence Claus Fogarty, editors, *Proceedings of the First European Workshop on Genetic Programming (EuroGP'98)*, volume 1391/1998 of *Lecture Notes in Computer Science (LNCS)*, pages 83–95, Paris, France, April 14–15, 1998. Berlin, Germany: Springer-Verlag GmbH. URL http://www.grammatical-evolution.org/papers/eurogp98.ps.
142. Michael O'Neill and Conor Ryan. *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*, volume 4 of *Genetic Programming Series*. New York, NY, USA: Springer Science+Business Media, Inc., 2003. ISBN 1402074441 and 978-1-4020-7444-8. URL http://books.google.de/books?id=yVx5dg_opRIC.
143. Alexandre Devert, Thomas Weise, and Ke Tang. A study on scalable representations for evolutionary optimization of ground structures. *Evolutionary Computation*, 20(3):453–472, Fall 2012. doi: 10.1162/EVCO_a_00054. URL http://www.marmakoide.org/download/publications/devweita-ecj-preprint.pdf.
144. Alexandre Devert. *Building Processes Optimization: Toward an Artificial Ontogeny based Approach*. PhD thesis, Paris, France: Université Paris-Sud, Ecole Doctorale d'Informatique and Orsay, France: Institut National de Recherche en Informatique et en Automatique (INRIA), Centre de Recherche Saclay – Île-de-France, May 2009.

145. Phil Husbands and Frank Mill. Simulated co-evolution as the mechanism for emergent planning and scheduling. In Richard K. Belew and Lashon Bernard Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms (ICGA'91)*, pages 264–270, San Diego, CA, USA: University of California (UCSD), July 13–16, 1991. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. URL http://www.informatics.sussex.ac.uk/users/philh/pubs/icga91Husbands.pdf.
146. 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.
147. Antonio LaTorre, José María Peña, Santiago Muelas, and Manuel Zaforas. Hybrid evolutionary algorithms for large scale continuous problems. In Franz Rothlauf, Günther R. Raidl, Anna Isabel Esparcia-Alcázar, Ying-Ping Chen, Gabriela Ochoa, Ender Ozcan, Marc Schoenauer, Anne Auger, Hans-Georg Beyer, Nikolaus Hansen, Steffen Finck, Raymond Ros, L. Darrell Whitley, Garnett Wilson, Simon Harding, William Benjamin Langdon, Man Leung Wong, Laurence D. Merkle, Frank W. Moore, Sevan G. Ficici, William Rand, Rick L. Riolo, Nawwaf Kharma, William R. Buckley, Julian Francis Miller, Kenneth Owen Stanley, Jaume Bacardit i Peñarroya, Will N. Browne, Jan Drugowitsch, Nicola Beume, Mike Preuß, Stephen Frederick Smith, Stefano Cagnoni, Alexandru Floares, Aaron Baughman, Steven Matt Gustafson, Maarten Keijzer, Arthur Kordon, and Clare Bates Congdon, editors, *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation (GECCO'09)*, pages 1863–1864, Montréal, QC, Canada: Delta Centre-Ville Hotel, July 8–12, 2009. New York, NY, USA: Association for Computing Machinery (ACM). doi: 10.1145/1569901.1570205.
148. Fei Peng, Ke Tang, Guoliang Chen, and Xin Yao. Population-based algorithm portfolios for numerical optimization. *IEEE Transactions on Evolutionary Computation (IEEE-EC)*, 14(5):782–800, March 29, 2010. doi: 10.1109/TEVC.2010.2040183.
149. Gene M. Amdahl. Validity of the single processor approach to achieving large-scale computing capabilities. In *American Federation of Information Processing Societies: Proceedings of the Spring Joint Computer Conference (AFIPS)*, pages 483–485, Atlantic City, NJ, USA, April 18–20, 1967. New York, NY, USA: Association for Computing Machinery (ACM) and London, New York: Academic Press. doi: 10.1145/1465482.1465560. URL http://www-inst.eecs.berkeley.edu/~n252/paper/Amdahl.pdf.

150. Simon Harding and Wolfgang Banzhaf. Fast genetic programming on gpus. In Marc Ebner, Michael O'Neill, Anikó Ekárt, Leonardo Vanneschi, and Anna Isabel Esparcia-Alcázar, editors, *Proceedings of the 10th European Conference on Genetic Programming (EuroGP'07)*, volume 4445/2007 of *Lecture Notes in Computer Science (LNCS)*, pages 90–101, València, Spain, April 11–13, 2007. Berlin, Germany: Springer-Verlag GmbH. doi: 10.1007/978-3-540-71605-1_9. URL http://www.cs.mun.ca/~banzhaf/papers/eurogp07.pdf.
151. William Benjamin Langdon. A simd interpreter for genetic programming on gpu graphics cards. Computer Science Technical Report CSM-470, Wivenhoe Park, Colchester, Essex, UK: University of Essex, Departments of Mathematical and Biological Sciences, July 3, 2007. URL http://cswww.essex.ac.uk/technical-reports/2007/csm_470.pdf.
152. William Benjamin Langdon and Wolfgang Banzhaf. A simd interpreter for genetic programming on gpu graphics cards. In Michael O'Neill, Leonardo Vanneschi, Steven Matt Gustafson, Anna Isabel Esparcia-Alcázar, Ivanoe de Falco, Antonio Della Cioppa, and Ernesto Tarantino, editors, *Genetic Programming – Proceedings of the 11th European Conference on Genetic Programming (EuroGP'08)*, volume 4971/2008 of *Lecture Notes in Computer Science (LNCS)*, pages 73–85, Naples, Italy, March 26–28, 2008. Berlin, Germany: Springer-Verlag GmbH. doi: 10.1007/978-3-540-78671-9_7. URL http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/langdon_2008_eurogp.html.
153. Weihang Zhu. A study of parallel evolution strategy – pattern search on a gpu computing platform. In Lihong Xu, Erik D. Goodman, and Yongsheng Ding, editors, *Proceedings of the First ACM/SIGEVO Summit on Genetic and Evolutionary Computation (GEC'09)*, pages 765–771, Shanghai, China: Hua-Ting Hotel & Towers, June 12–14, 2009. New York, NY, USA: ACM Press. doi: 10.1145/1543834.1543939.
154. Shigeyoshi Tsutsui and Yoshiji Fujimoto. Solving quadratic assignment problems by genetic algorithms with gpu computation: A case study. In Franz Rothlauf, Günther R. Raidl, Anna Isabel Esparcia-Alcázar, Ying-Ping Chen, Gabriela Ochoa, Ender Ozcan, Marc Schoenauer, Anne Auger, Hans-Georg Beyer, Nikolaus Hansen, Steffen Finck, Raymond Ros, L. Darrell Whitley, Garnett Wilson, Simon Harding, William Benjamin Langdon, Man Leung Wong, Laurence D. Merkle, Frank W. Moore, Sevan G. Ficici, William Rand, Rick L. Riolo, Nawwaf Kharma, William R. Buckley, Julian Francis Miller, Kenneth Owen Stanley, Jaume Bacardit i Peñarroya, Will N. Browne, Jan Drugowitsch, Nicola Beume, Mike Preuß, Stephen Frederick Smith, Stefano Cagnoni, Alexandru Floares, Aaron Baughman, Steven Matt Gustafson, Maarten Keijzer, Arthur Kordon, and Clare Bates Congdon, editors, *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation (GECCO'09)*, pages 2523–2530, Montréal, QC, Canada: Delta Centre-Ville Hotel, July 8–12, 2009. New York, NY, USA: Association for Computing Machinery (ACM). doi: 10.1145/1570256.1570355. URL http://www2.hannan-u.ac.jp/~tsutsui/ps/gecco/wk3006-tsutsui.pdf.

Bibliography XXII

155. Kamil Rocki and Reiji Suda. An efficient gpu implementation of a multi-start tsp solver for large problem instances. In Terence Soule and Jason H. Moore, editors, *Companion Material Proceedings Genetic and Evolutionary Computation Conference (GECCO'12)*, pages 1441–1442, Philadelphia, PA, USA: Doubletree by Hilton Hotel Philadelphia Center City, July 7–11, 2012. New York, NY, USA: Association for Computing Machinery (ACM). doi: 10.1145/2330784.2330978.
156. Thomas Weise and Kurt Geihs. Dgpf – an adaptable framework for distributed multi-objective search algorithms applied to the genetic programming of sensor networks. In Bogdan Filipič and Jurij Šilc, editors, *Proceedings of the Second International Conference on Bioinspired Optimization Methods and their Applications (BIOMA'06)*, Informacijska Družba (Information Society), pages 157–166, Ljubljana, Slovenia: Jožef Stefan International Postgraduate School, October 9–10, 2006. Ljubljana, Slovenia: Jožef Stefan Institute.
157. Thomas Weise, Kurt Geihs, and Philipp Andreas Baer. Genetic programming for proactive aggregation protocols. In Bartłomiej Beliczyński, Andrzej Dzieliński, Marcin Iwanowski, and Bernardete Ribeiro, editors, *Proceedings of the 8th International Conference on Adaptive and Natural Computing Algorithms, Part I (ICANNGA'07)*, volume 4431/2007 of *Lecture Notes in Computer Science (LNCS)*, pages 167–173, Warsaw, Poland: Warsaw University, April 11–17, 2007. Berlin, Germany: Springer-Verlag GmbH. doi: 10.1007/978-3-540-71618-1_19.
158. David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation (IEEE-EC)*, 1(1):67–82, April 1997. doi: 10.1109/4235.585893. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.39.6926.