

Nature Inspired Metaheuristics for Optimization

Helio J.C. Barbosa

Laboratório Nacional de Computação Científica - LNCC/MCTIC

Universidade Federal de Juiz de Fora

Summary

- Introduction
- Optimization Problems
- Metaheuristics

Optimization

Imagine the task of finding the highest peak in a mountain range...



under a very thick fog...

Optimization

- Your only tool is a battery operated device that can give you the *height* of your current position.
- Every time you want to know your current height the battery level drops a bit.
- Mathematicians would call this a zero-th order, or derivative-free technique.
- This technique can handle situations where derivatives are not defined/available/reliable.
- Also known as “black-box” optimization.
- Common when a computer simulation is required to evaluate the objective function and/or constraints of the problem.

Optimization

- Your only tool is a battery operated device that can give you the *height* of your current position.
- Every time you want to know your current height the battery level drops a bit.
- Mathematicians would call this a zero-th order, or derivative-free technique.
- This technique can handle situations where derivatives are not defined/available/reliable.
- Also known as “black-box” optimization.
- Common when a computer simulation is required to evaluate the objective function and/or constraints of the problem.

Optimization

- Your only tool is a battery operated device that can give you the *height* of your current position.
- Every time you want to know your current height the battery level drops a bit.
- Mathematicians would call this a zero-th order, or derivative-free technique.
- This technique can handle situations where derivatives are not defined/available/reliable.
- Also known as “black-box” optimization.
- Common when a computer simulation is required to evaluate the objective function and/or constraints of the problem.

Optimization

- Your only tool is a battery operated device that can give you the *height* of your current position.
- Every time you want to know your current height the battery level drops a bit.
- Mathematicians would call this a zero-th order, or derivative-free technique.
- This technique can handle situations where derivatives are not defined/available/reliable.
- Also known as “black-box” optimization.
- Common when a computer simulation is required to evaluate the objective function and/or constraints of the problem.

Optimization

- Your only tool is a battery operated device that can give you the *height* of your current position.
- Every time you want to know your current height the battery level drops a bit.
- Mathematicians would call this a zero-th order, or derivative-free technique.
- This technique can handle situations where derivatives are not defined/available/reliable.
- Also known as “black-box” optimization.
- Common when a computer simulation is required to evaluate the objective function and/or constraints of the problem.

Optimization

- Your only tool is a battery operated device that can give you the *height* of your current position.
- Every time you want to know your current height the battery level drops a bit.
- Mathematicians would call this a zero-th order, or derivative-free technique.
- This technique can handle situations where derivatives are not defined/available/reliable.
- Also known as “black-box” optimization.
- Common when a computer simulation is required to evaluate the objective function and/or constraints of the problem.

Optimization

- More sophisticated climbers use a battery operated device that also provides the direction of steepest ascent at your current position.
- Every time you request that information the battery level drops a bit more.
- Mathematicians would call this a first order, or gradient based technique.
- Requires differentiability of the functions involved.

Optimization

- More sophisticated climbers use a battery operated device that also provides the direction of steepest ascent at your current position.
- Every time you request that information the battery level drops a bit more.
- Mathematicians would call this a first order, or gradient based technique.
- Requires differentiability of the functions involved.

Optimization

- More sophisticated climbers use a battery operated device that also provides the direction of steepest ascent at your current position.
- Every time you request that information the battery level drops a bit more.
- Mathematicians would call this a first order, or gradient based technique.
- Requires differentiability of the functions involved.

Optimization

- More sophisticated climbers use a battery operated device that also provides the direction of steepest ascent at your current position.
- Every time you request that information the battery level drops a bit more.
- Mathematicians would call this a first order, or gradient based technique.
- Requires differentiability of the functions involved.

Optimization

- Even more sophisticated climbers are available.
- Namely, those that use second-order (derivative) information.
- Every time you request that information the battery level drops even more.
- Applicable to smoother functions.

Optimization

- Even more sophisticated climbers are available.
- Namely, those that use second-order (derivative) information.
- Every time you request that information the battery level drops even more.
- Applicable to smoother functions.

Optimization

- Even more sophisticated climbers are available.
- Namely, those that use second-order (derivative) information.
- Every time you request that information the battery level drops even more.
- Applicable to smoother functions.

Optimization

- Even more sophisticated climbers are available.
- Namely, those that use second-order (derivative) information.
- Every time you request that information the battery level drops even more.
- Applicable to smoother functions.

Optimization

- The best climber would be the one that –spending a pre-fixed level of battery charge– reaches the highest point.

A simple formulation

Find $x^* \in S$ such that

$$f(x^*) > f(x) \quad \forall x \in S \subset R^n$$

In words, x^* *maximizes* f in the admissible search set S .

- f is the *objective function*.
- x is the *design/decision variable*.

A simple formulation

Find $x^* \in S$ such that

$$f(x^*) > f(x) \quad \forall x \in S \subset R^n$$

In words, x^* *maximizes* f in the admissible search set S .

- f is the *objective function*.
- x is the *design/decision variable*.

A simple formulation

Find $x^* \in S$ such that

$$f(x^*) > f(x) \quad \forall x \in S \subset R^n$$

In words, x^* *maximizes* f in the admissible search set S .

- f is the *objective function*.
- x is the *design/decision variable*.

Local and global optima

Let $x^* \in S$ with norm $\|x^*\|$.

- x^* is a local minimizer of f , if and only if

$$\exists \varepsilon > 0 : f(x^*) \leq f(x) \quad \forall x \in S : \|x^* - x\| < \varepsilon$$

- x^* is a global minimizer of f , if and only if

$$f(x^*) \leq f(x) \quad \forall x \in S$$

Local and global optima

Let $x^* \in S$ with norm $\|x^*\|$.

- x^* is a local minimizer of f , if and only if

$$\exists \varepsilon > 0 : f(x^*) \leq f(x) \quad \forall x \in S : \|x^* - x\| < \varepsilon$$

- x^* is a global minimizer of f , if and only if

$$f(x^*) \leq f(x) \quad \forall x \in S$$

Local and global optima

Let $x^* \in S$ with norm $\|x^*\|$.

- x^* is a local minimizer of f , if and only if

$$\exists \varepsilon > 0 : f(x^*) \leq f(x) \quad \forall x \in S : \|x^* - x\| < \varepsilon$$

- x^* is a global minimizer of f , if and only if

$$f(x^*) \leq f(x) \quad \forall x \in S$$

Local and global optima

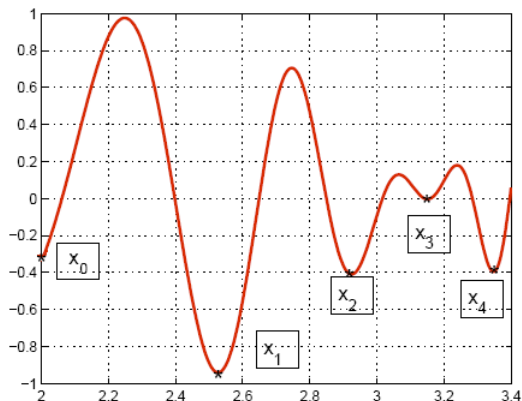


Figure: x_0, x_1, x_2, x_3 e x_4 are local optima while x_1 is the global minimum.

Standard formulation in R^n

The standard formulation of a **constrained** optimization problem in R^n is:

$$\min f(x)$$

subject to

$$g_p(x) \geq 0, \quad p = 1, 2, \dots, \bar{p}$$

$$h_q(x) = 0, \quad q = 1, 2, \dots, \bar{q}$$

$$x_i^L \leq x_i \leq x_i^U, \quad i = 1, 2, \dots, n$$

where

$$x = [x_1, x_2, \dots, x_n]^T \in R^n$$

is the vector of decision/design variables.

Standard formulation in R^n

The standard formulation of a **constrained** optimization problem in R^n is:

$$\min f(x)$$

subject to

$$g_p(x) \geq 0, \quad p = 1, 2, \dots, \bar{p}$$

$$h_q(x) = 0, \quad q = 1, 2, \dots, \bar{q}$$

$$x_i^L \leq x_i \leq x_i^U, \quad i = 1, 2, \dots, n$$

where

$$x = [x_1, x_2, \dots, x_n]^T \in R^n$$

is the vector of decision/design variables.

Standard formulation in R^n

The standard formulation of a **constrained** optimization problem in R^n is:

$$\min f(x)$$

subject to

$$g_p(x) \geq 0, \quad p = 1, 2, \dots, \bar{p}$$

$$h_q(x) = 0, \quad q = 1, 2, \dots, \bar{q}$$

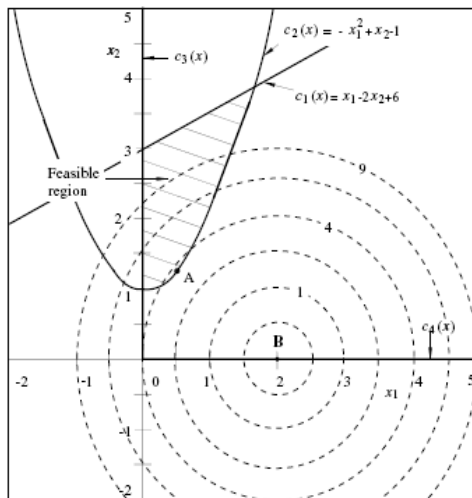
$$x_i^L \leq x_i \leq x_i^U, \quad i = 1, 2, \dots, n$$

where

$$x = [x_1, x_2, \dots, x_n]^T \in R^n$$

is the vector of decision/design variables.

Feasible Region



Constraint handling

- Metaheuristics are usually designed for unconstrained problems.
- As a result, they must be equipped with constraint handling techniques.
- Many options available in the literature; among them:
 - penalty techniques
 - selection techniques
 - special decoders
 - repair operators

Constraint handling

- Metaheuristics are usually designed for unconstrained problems.
- As a result, they must be equipped with constraint handling techniques.
- Many options available in the literature; among them:
 - penalty techniques
 - selection techniques
 - special decoders
 - repair operators

Constraint handling

- Metaheuristics are usually designed for unconstrained problems.
- As a result, they must be equipped with constraint handling techniques.
- Many options available in the literature; among them:
 - penalty techniques
 - selection techniques
 - special decoders
 - repair operators

Decision variables

Decision variables can be:

- continuous: $x_i \in R, \quad i = 1, \dots, n$
- integer: $x_i \in \{0, 1, 2, \dots\}, \quad i = 1, \dots, n$
- discrete: $x_i \in C_i = \{c_0, c_1, c_2, \dots\}, \quad i = 1, \dots, n$
- boolean: $x_i \in \{0, 1\}, \quad i = 1, \dots, n$
- functions: $x_i = x_i(t) \in C[a, b], t \in [a, b], \quad i = 1, \dots, n$

Decision variables

Decision variables can be:

- continuous: $x_i \in R, \quad i = 1, \dots, n$
- integer: $x_i \in \{0, 1, 2, \dots\}, \quad i = 1, \dots, n$
- discrete: $x_i \in C_i = \{c_0, c_1, c_2, \dots\}, \quad i = 1, \dots, n$
- boolean: $x_i \in \{0, 1\}, \quad i = 1, \dots, n$
- functions: $x_i = x_i(t) \in C[a, b], t \in [a, b], \quad i = 1, \dots, n$

Decision variables

Decision variables can be:

- continuous: $x_i \in R, \quad i = 1, \dots, n$
- integer: $x_i \in \{0, 1, 2, \dots\}, \quad i = 1, \dots, n$
- discrete: $x_i \in C_i = \{c_0, c_1, c_2, \dots\}, \quad i = 1, \dots, n$
- boolean: $x_i \in \{0, 1\}, \quad i = 1, \dots, n$
- functions: $x_i = x_i(t) \in C[a, b], t \in [a, b], \quad i = 1, \dots, n$

Decision variables

Decision variables can be:

- continuous: $x_i \in R, \quad i = 1, \dots, n$
- integer: $x_i \in \{0, 1, 2, \dots\}, \quad i = 1, \dots, n$
- discrete: $x_i \in C_i = \{c_0, c_1, c_2, \dots\}, \quad i = 1, \dots, n$
- boolean: $x_i \in \{0, 1\}, \quad i = 1, \dots, n$
- functions: $x_i = x_i(t) \in C[a, b], t \in [a, b], \quad i = 1, \dots, n$

Decision variables

Decision variables can be:

- continuous: $x_i \in R, \quad i = 1, \dots, n$
- integer: $x_i \in \{0, 1, 2, \dots\}, \quad i = 1, \dots, n$
- discrete: $x_i \in C_i = \{c_0, c_1, c_2, \dots\}, \quad i = 1, \dots, n$
- boolean: $x_i \in \{0, 1\}, \quad i = 1, \dots, n$
- functions: $x_i = x_i(t) \in C[a, b], t \in [a, b], \quad i = 1, \dots, n$

Decision variables

Decision variables can be:

- continuous: $x_i \in R, \quad i = 1, \dots, n$
- integer: $x_i \in \{0, 1, 2, \dots\}, \quad i = 1, \dots, n$
- discrete: $x_i \in C_i = \{c_0, c_1, c_2, \dots\}, \quad i = 1, \dots, n$
- boolean: $x_i \in \{0, 1\}, \quad i = 1, \dots, n$
- functions: $x_i = x_i(t) \in C[a, b], t \in [a, b], \quad i = 1, \dots, n$

Decision variables

But note that other parameters can be used in order to search for more complex structures that optimize a given performance index:

- networks
- shapes
- algebraic expressions
- system of differential equations
- electrical circuits
- chemical structures
- “programs”

With the conveniently defined corresponding search “spaces”.

Not much math available in this case.

Multi-objective optimization

Imagine that in a given manufacture process one wants to simultaneously:

- 1 minimize material waste
- 2 minimize production time

The objectives above are:

- conflicting
- non-commensurable

Multi-objective optimization

Imagine that in a given manufacture process one wants to simultaneously:

- 1 minimize material waste
- 2 minimize production time

The objectives above are:

- conflicting
- non-commensurable

Multi-objective optimization

Imagine that in a given manufacture process one wants to simultaneously:

- 1 minimize material waste
- 2 minimize production time

The objectives above are:

- conflicting
- non-commensurable

Multi-objective optimization

Imagine that in a given manufacture process one wants to simultaneously:

- 1 minimize material waste
- 2 minimize production time

The objectives above are:

- conflicting
- non-commensurable

Multi-objective optimization

Imagine that in a given manufacture process one wants to simultaneously:

- 1 minimize material waste
- 2 minimize production time

The objectives above are:

- conflicting
- non-commensurable

Multi-objective optimization

Imagine that in a given manufacture process one wants to simultaneously:

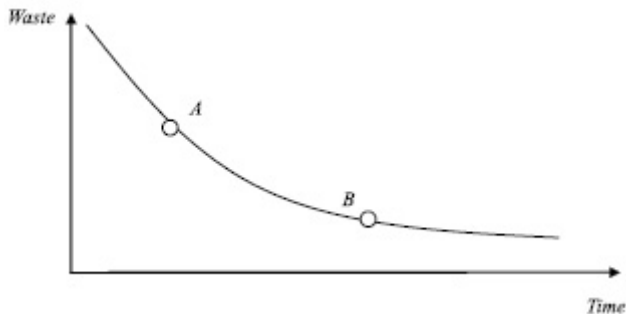
- 1 minimize material waste
- 2 minimize production time

The objectives above are:

- conflicting
- non-commensurable

Multi-objective optimization

Graphically:



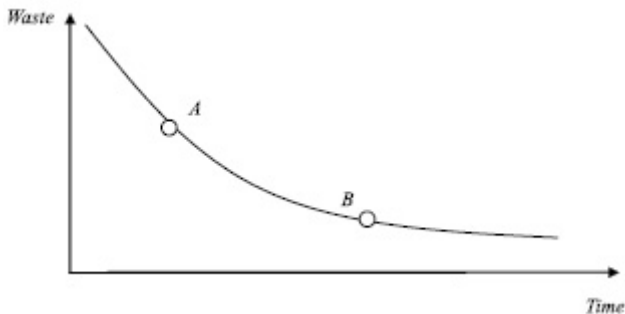
Solution A is better w.r.t. objective 1 (faster).

Solution B is better w.r.t. objective 2 (less waste).

Which one is the best ?

Multi-objective optimization

Graphically:



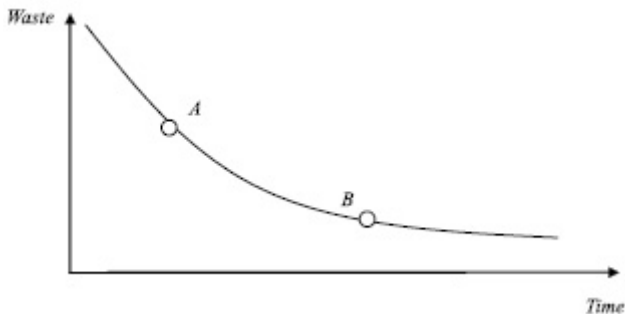
Solution A is better w.r.t. objective 1 (faster).

Solution B is better w.r.t. objective 2 (less waste).

Which one is the best ?

Multi-objective optimization

Graphically:



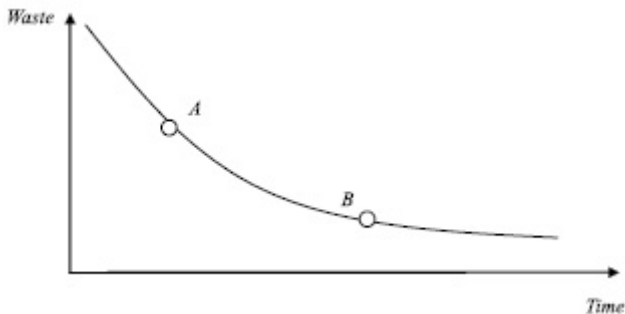
Solution A is better w.r.t. objective 1 (faster).

Solution B is better w.r.t. objective 2 (less waste).

Which one is the best ?

Multi-objective optimization

Graphically:



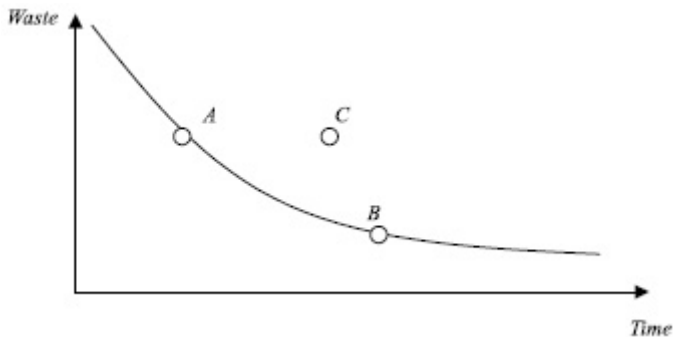
Solution A is better w.r.t. objective 1 (faster).

Solution B is better w.r.t. objective 2 (less waste).

Which one is the best ?

Multi-objective optimization

What about solution C ?

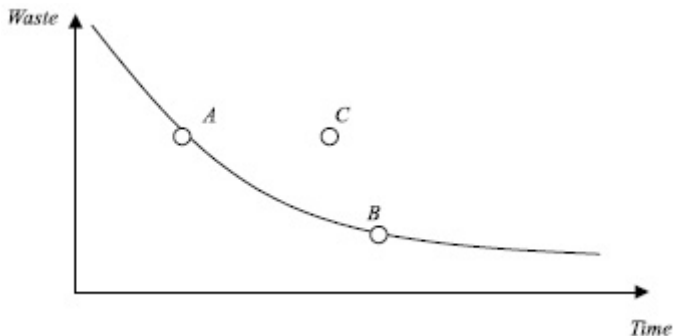


Solution C is not interesting; it is *dominated* by solution A.

Solution A is faster than C and has the same level of material waste.

Multi-objective optimization

What about solution C ?

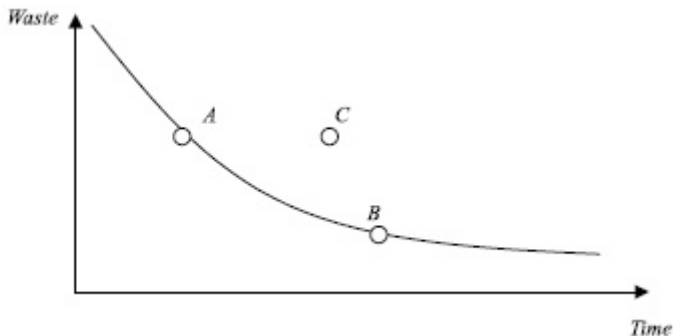


Solution C is not interesting; it is *dominated* by solution A.

Solution A is faster than C and has the same level of material waste.

Multi-objective optimization

What about solution C ?

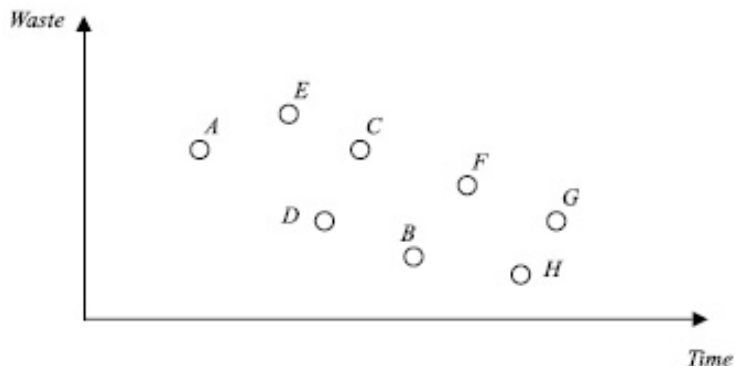


Solution C is not interesting; it is *dominated* by solution A.

Solution A is faster than C and has the same level of material waste.

Multi-objective optimization

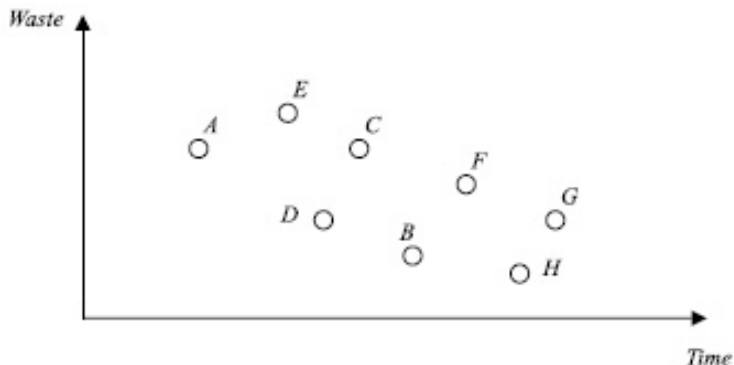
Which would be the interesting solutions now ?



The *Non-dominated* solutions: A, D, B & H.

Multi-objective optimization

Which would be the interesting solutions now ?



The *Non-dominated* solutions: A, D, B & H.

Multi-objective optimization

$$\begin{array}{ll}\text{minimize} & \mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_k(\mathbf{x})] \\ \text{subject to} & \mathbf{x} = (x_1, x_2, \dots, x_n) \in S\end{array}$$

where

- $k(\geq 2)$ is the number of objectives,
- S is the admissible set,
- \mathbf{f} is the vector of objectives,
- $f_i : R^n \rightarrow R$,
- \mathbf{x} is the decision vector,

Search algorithms

Depend strongly on the set where the search is carried out, and additional constraints imposed on the candidate solutions.

Any search algorithm requires

- a *representation* for the candidate solutions
- suitable *move operators* to generate new candidates

and essentially involve the repetition of the steps

- generate candidate solution
- test candidate solution

until a stopping criterion is satisfied.

Search algorithms

Depend strongly on the set where the search is carried out, and additional constraints imposed on the candidate solutions.

Any search algorithm requires

- a *representation* for the candidate solutions
- suitable *move operators* to generate new candidates

and essentially involve the repetition of the steps

- generate candidate solution
- test candidate solution

until a stopping criterion is satisfied.

Search algorithms

Depend strongly on the set where the search is carried out, and additional constraints imposed on the candidate solutions.

Any search algorithm requires

- a *representation* for the candidate solutions
- suitable *move operators* to generate new candidates

and essentially involve the repetition of the steps

- generate candidate solution
- test candidate solution

until a stopping criterion is satisfied.

Search algorithms

Depend strongly on the set where the search is carried out, and additional constraints imposed on the candidate solutions.

Any search algorithm requires

- a *representation* for the candidate solutions
- suitable *move operators* to generate new candidates

and essentially involve the repetition of the steps

- generate candidate solution
- test candidate solution

until a stopping criterion is satisfied.

Search algorithms

Depend strongly on the set where the search is carried out, and additional constraints imposed on the candidate solutions.

Any search algorithm requires

- a *representation* for the candidate solutions
- suitable *move operators* to generate new candidates

and essentially involve the repetition of the steps

- generate candidate solution
- test candidate solution

until a stopping criterion is satisfied.

Search algorithms

Depend strongly on the set where the search is carried out, and additional constraints imposed on the candidate solutions.

Any search algorithm requires

- a *representation* for the candidate solutions
- suitable *move operators* to generate new candidates

and essentially involve the repetition of the steps

- generate candidate solution
- test candidate solution

until a stopping criterion is satisfied.

Search algorithms

Depend strongly on the set where the search is carried out, and additional constraints imposed on the candidate solutions.

Any search algorithm requires

- a *representation* for the candidate solutions
- suitable *move operators* to generate new candidates

and essentially involve the repetition of the steps

- generate candidate solution
- test candidate solution

until a stopping criterion is satisfied.

Search algorithms

Depend strongly on the set where the search is carried out, and additional constraints imposed on the candidate solutions.

Any search algorithm requires

- a *representation* for the candidate solutions
- suitable *move operators* to generate new candidates

and essentially involve the repetition of the steps

- generate candidate solution
- test candidate solution

until a stopping criterion is satisfied.

Search algorithms

According to the amount of domain knowledge used, they can be

- **weak**
- **strong**

Also:

- deterministic
- stochastic

Search algorithms

They all face the dilemma:

- Exploration/Diversification.
versus
- Exploitation/Intensification.

Search algorithms

They all face the dilemma:

- Exploration/Diversification.
versus
- Exploitation/Intensification.

Search algorithms

They all face the dilemma:

- **Exploration/Diversification.**
versus
- **Exploitation/Intensification.**

Search algorithms

They all face the dilemma:

- **Exploration/Diversification.**
versus
- **Exploitation/Intensification.**

Search Algorithms

Poli & Logan identify possible components of a search algorithm:

- a *representation* for the candidate solutions
- an *initialization* procedure
- a *stopping criterion*
- a *memory structure* holding the solutions to be operated upon
- a procedure for *memory management* to control inclusion/removal of candidate solutions in/from the memory
- a procedure for the *selection* of candidate(s) to be operated upon
- a set of *move operators* to be applied to one or more candidate solutions in order to generate new ones

Search Algorithms

Poli & Logan identify possible components of a search algorithm:

- a *representation* for the candidate solutions
- an *initialization* procedure
- a *stopping criterion*
- a *memory structure* holding the solutions to be operated upon
- a procedure for *memory management* to control inclusion/removal of candidate solutions in/from the memory
- a procedure for the *selection* of candidate(s) to be operated upon
- a set of *move operators* to be applied to one or more candidate solutions in order to generate new ones

Search Algorithms

Poli & Logan identify possible components of a search algorithm:

- a *representation* for the candidate solutions
- an *initialization* procedure
- a *stopping criterion*
- a *memory structure* holding the solutions to be operated upon
- a procedure for *memory management* to control inclusion/removal of candidate solutions in/from the memory
- a procedure for the *selection* of candidate(s) to be operated upon
- a set of *move operators* to be applied to one or more candidate solutions in order to generate new ones

Search Algorithms

Poli & Logan identify possible components of a search algorithm:

- a *representation* for the candidate solutions
- an *initialization* procedure
- a *stopping criterion*
- a *memory structure* holding the solutions to be operated upon
- a procedure for *memory management* to control inclusion/removal of candidate solutions in/from the memory
- a procedure for the *selection* of candidate(s) to be operated upon
- a set of *move operators* to be applied to one or more candidate solutions in order to generate new ones

Search Algorithms

Poli & Logan identify possible components of a search algorithm:

- a *representation* for the candidate solutions
- an *initialization* procedure
- a *stopping criterion*
- a *memory structure* holding the solutions to be operated upon
- a procedure for *memory management* to control inclusion/removal of candidate solutions in/from the memory
- a procedure for the *selection* of candidate(s) to be operated upon
- a set of *move operators* to be applied to one or more candidate solutions in order to generate new ones

Search Algorithms

Poli & Logan identify possible components of a search algorithm:

- a *representation* for the candidate solutions
- an *initialization* procedure
- a *stopping criterion*
- a *memory structure* holding the solutions to be operated upon
- a procedure for *memory management* to control inclusion/removal of candidate solutions in/from the memory
- a procedure for the *selection* of candidate(s) to be operated upon
- a set of *move operators* to be applied to one or more candidate solutions in order to generate new ones

Search Algorithms

Poli & Logan identify possible components of a search algorithm:

- a *representation* for the candidate solutions
- an *initialization* procedure
- a *stopping criterion*
- a *memory structure* holding the solutions to be operated upon
- a procedure for *memory management* to control inclusion/removal of candidate solutions in/from the memory
- a procedure for the *selection* of candidate(s) to be operated upon
- a set of *move operators* to be applied to one or more candidate solutions in order to generate new ones

Search Algorithms

Poli & Logan identify possible components of a search algorithm:

- a *representation* for the candidate solutions
- an *initialization* procedure
- a *stopping criterion*
- a *memory structure* holding the solutions to be operated upon
- a procedure for *memory management* to control inclusion/removal of candidate solutions in/from the memory
- a procedure for the *selection* of candidate(s) to be operated upon
- a set of *move operators* to be applied to one or more candidate solutions in order to generate new ones

Search Algorithms

more ingredients...

- a *control strategy* for applying the move operators
- a *repair ou filter* operator to deal with infeasible solutions
- a procedure to define a *quality gradient* to improve a candidate solution
- a procedure to *evaluate the quality* of a candidate solution
- a procedure to *compare the quality* of 2 candidates
- a *history* containing a list of solutions already tested.

Search Algorithms

more ingredients...

- a *control strategy* for applying the move operators
- a *repair ou filter* operator to deal with infeasible solutions
- a procedure to define a *quality gradient* to improve a candidate solution
- a procedure to *evaluate the quality* of a candidate solution
- a procedure to *compare the quality* of 2 candidates
- a *history* containing a list of solutions already tested.

Search Algorithms

more ingredients...

- a *control strategy* for applying the move operators
- a *repair ou filter* operator to deal with infeasible solutions
- a procedure to define a *quality gradient* to improve a candidate solution
- a procedure to *evaluate the quality* of a candidate solution
- a procedure to *compare the quality* of 2 candidates
- a *history* containing a list of solutions already tested.

Search Algorithms

more ingredients...

- a *control strategy* for applying the move operators
- a *repair ou filter* operator to deal with infeasible solutions
- a procedure to define a *quality gradient* to improve a candidate solution
- a procedure to *evaluate the quality* of a candidate solution
- a procedure to *compare the quality* of 2 candidates
- a *history* containing a list of solutions already tested.

Search Algorithms

more ingredients...

- a *control strategy* for applying the move operators
- a *repair ou filter* operator to deal with infeasible solutions
- a procedure to define a *quality gradient* to improve a candidate solution
- a procedure to *evaluate the quality* of a candidate solution
- a procedure to *compare the quality* of 2 candidates
- a *history* containing a list of solutions already tested.

Search Algorithms

more ingredients...

- a *control strategy* for applying the move operators
- a *repair ou filter* operator to deal with infeasible solutions
- a procedure to define a *quality gradient* to improve a candidate solution
- a procedure to *evaluate the quality* of a candidate solution
- a procedure to *compare the quality* of 2 candidates
- a *history* containing a list of solutions already tested.

Search Algorithms

more ingredients...

- a *control strategy* for applying the move operators
- a *repair ou filter* operator to deal with infeasible solutions
- a procedure to define a *quality gradient* to improve a candidate solution
- a procedure to *evaluate the quality* of a candidate solution
- a procedure to *compare the quality* of 2 candidates
- a *history* containing a list of solutions already tested.

Heurística, uma definição bem geral:

"A heuristic is *anything* that is:

- (i) helpful, useful, based on experience, and
- (ii) unjustified, unjustifiable, and potentially fallible."

....your best bet !

Billy V. Koen, *Professor*,
The University of Texas/Austin

Características de uma heurística:

- A heuristic does not guarantee a solution
- It may contradict other heuristics
- It reduces the search time for solving a problem
- Its acceptance depends on the immediate context, instead of on an absolute standard

Billy V. Koen, *Professor*,
The University of Texas/Austin

Meta-heuristic

A master strategy that guides and modifies other heuristics to produce solutions beyond those that are normally generated in a quest for local optimality.

Manuel Laguna

Some well-known metaheuristics:

- The genetic algorithm
- Simulated annealing
- GRASP (greedy randomized adaptive search procedure)
- Tabu search

In some quarters...

“Algorithms are conceived in analytic purity in the high citadels of academic research, heuristics are midwived by expediency in the dark corners of the practitioner’s lair ... and are accorded lower status.”

Fred Glover

Nature as source of inspiration

- for hardware
- for software

Two ways:

- bio-inspiration
- bio-mimetism

Nature as source of inspiration

- for hardware
- for software

Two ways:

- bio-inspiration
- bio-mimetism

Nature as source of inspiration

- for hardware
- for software

Two ways:

- bio-inspiration
- bio-mimetism

Nature as source of inspiration

- for hardware
- for software

Two ways:

- bio-inspiration
- bio-mimetism

Nature as source of inspiration

- for hardware
- for software

Two ways:

- bio-inspiration
- bio-mimetism

Nature as source of inspiration

- for hardware
- for software

Two ways:

- bio-inspiration
- bio-mimetism

Sources of inspiration

- Biology: artificial neural networks
- Physics: simulated annealing technique

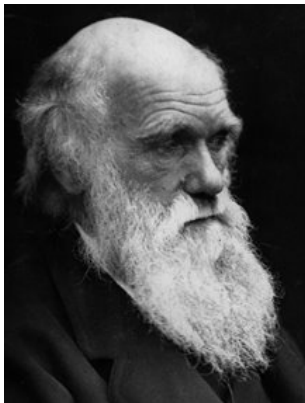
Sources of inspiration

- **Biology: artificial neural networks**
- Physics: simulated annealing technique

Sources of inspiration

- **Biology: artificial neural networks**
- **Physics: simulated annealing technique**

Sources of inspiration



Darwinian evolution...

Darwinian evolution

“Nothing in Biology makes sense, except in the light of evolution.”

Theodosius Dobzhansky

“If I were to give an award for the single best idea anyone has ever had, I’d give it to Darwin, ahead of Newton and Einstein and everyone else.”

Daniel C. Dennet

Darwinian evolution

“Nothing in Biology makes sense, except in the light of evolution.”

Theodosius Dobzhansky

“If I were to give an award for the single best idea anyone has ever had, I’d give it to Darwin, ahead of Newton and Einstein and everyone else.”

Daniel C. Dennet

Darwinian evolution

Biological evolution ... is change in the properties of populations of organisms that transcend the lifetime of a single individual. ... individual organisms do not evolve. ... are those that are inheritable via the genetic material from one generation to the next."

Douglas J. Futuyma

Darwinian Evolution

The elements

- variation
- inheritance
- selection

Darwinian Evolution

The elements

- variation
- inheritance
- selection

Darwinian Evolution

The elements

- variation
- inheritance
- selection

The Parallel Nature-Computation

Nature	Evolutionary Computation
Individual	Candidate Solution
Population	Set of candidate solutions
Fitness	Solution quality
Genotype	Solution representation
Gene	A part of the solution representation
Development	Decoding the representation
Crossover & Mutation	Move operators
Natural selection	Re-use of good (sub-)solutions

The origins of Evolutionary Computation

Precursors:

- Turing (1948)
- Barricelli (1953)
- Anderson(1953)
- Fraser(1957)
- Friedberg(1958)
- Bremermann(1962)

Turing's 1950 paper "Computing Machinery and Intelligence":

"We cannot expect to find a good child-machine at the first attempt. One must experiment with teaching one such machine and see how well it learns. One can then try another and see if it is better or worse. There is an obvious connection between this process and evolution, by the identifications:

Structure of the child machine = Hereditary material

Changes of the child machine = Mutations

Natural selection = Judgment of the experimenter"

The origins of Evolutionary Computation

Pioneers:



Ingo Rechenberg & Hans-Paul Schwefel

The origins of Evolutionary Computation

Pioneers:



John Holland

The Genetic Algorithm

Basic Features:

- employ a population of candidate solutions
- operates on the encoding of a candidate solution rather than on the solution itself
- use probabilistic transition rules
- minimum requirements on the objective function

The Genetic Algorithm

Basic Features:

- employ a population of candidate solutions
- operates on the encoding of a candidate solution rather than on the solution itself
- use probabilistic transition rules
- minimum requirements on the objective function

The Genetic Algorithm

Basic Features:

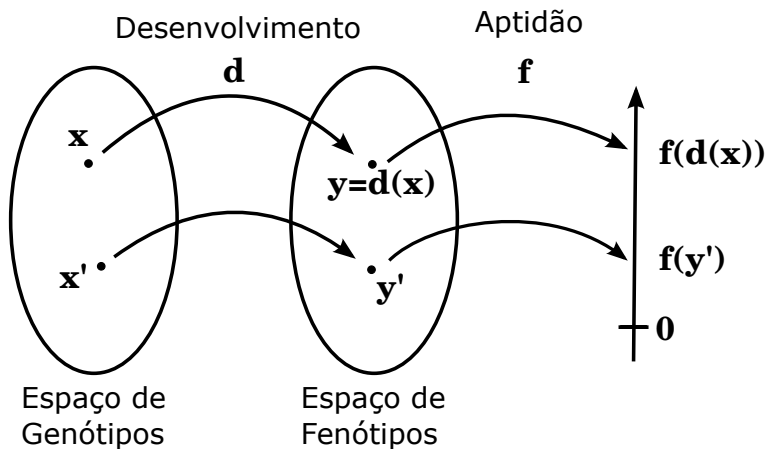
- employ a population of candidate solutions
- operates on the encoding of a candidate solution rather than on the solution itself
- use probabilistic transition rules
- minimum requirements on the objective function

The Genetic Algorithm

Basic Features:

- employ a population of candidate solutions
- operates on the encoding of a candidate solution rather than on the solution itself
- use probabilistic transition rules
- minimum requirements on the objective function

The spaces involved in a genetic algorithm



The Genetic Algorithm

Generic GA

Begin

Initialize the population

Evaluate individuals in the population

repeat

- Select individuals for reproduction

- Apply recombination & mutation operators

- Evaluate individuals in the population

- Select individuals for survival

until stop criterion satisfied

End

The Genetic Algorithm

Main components:

- candidate solution encoding
- population structure & size
- fitness function
- variation operators & their probability of application
- selection operator
- survival operator (population management)

The Genetic Algorithm

Main components:

- candidate solution encoding
- population structure & size
- fitness function
- variation operators & their probability of application
- selection operator
- survival operator (population management)

The Genetic Algorithm

Main components:

- candidate solution encoding
- population structure & size
- fitness function
- variation operators & their probability of application
- selection operator
- survival operator (population management)

The Genetic Algorithm

Main components:

- candidate solution encoding
- population structure & size
- fitness function
- variation operators & their probability of application
- selection operator
- survival operator (population management)

The Genetic Algorithm

Main components:

- candidate solution encoding
- population structure & size
- fitness function
- variation operators & their probability of application
- selection operator
- survival operator (population management)

The Genetic Algorithm

Main components:

- candidate solution encoding
- population structure & size
- fitness function
- variation operators & their probability of application
- selection operator
- survival operator (population management)

Some advantages of the GAs:

- easy interface with existent simulators
- extensibility
- easy hybridization with other techniques
- natural parallelism
- robustness

(Don't miss the next talk !!!)

Some advantages of the GAs:

- easy interface with existent simulators
- extensibility
- easy hybridization with other techniques
- natural parallelism
- robustness

(Don't miss the next talk !!!)

Some advantages of the GAs:

- easy interface with existent simulators
- extensibility
- easy hybridization with other techniques
- natural parallelism
- robustness

(Don't miss the next talk !!!)

Some advantages of the GAs:

- easy interface with existent simulators
- extensibility
- easy hybridization with other techniques
- natural parallelism
- robustness

(Don't miss the next talk !!!)

Some advantages of the GAs:

- easy interface with existent simulators
- extensibility
- easy hybridization with other techniques
- natural parallelism
- robustness

(Don't miss the next talk !!!)

Other sources of inspiration

Ant Colony Algorithm	First reference
Ant System (AS)	Dorigo <i>et al</i> 1991
Elitist AS (EAS)	Dorigo <i>et al</i> 1992
Ant-Q	Gambardella & Dorigo 1995
Ant Colony System (ACS)	Dorigo & Gambardella 1996
Max-Min AS (MMAS)	Stützle & Hoos 1996
Rank-Based AS (AS_{Rank}/RBAS)	Bullnheimer <i>et al</i> 1997
ANTS	Maniezzo 1999
Best-Worst AS (BWAS)	Cordón <i>et al</i> 2000
Hyper-Cube AS (HCAS)	Blum <i>et al</i> 2001
.	.
.	.
.	.

Other sources of inspiration

Artificial Immune Systems

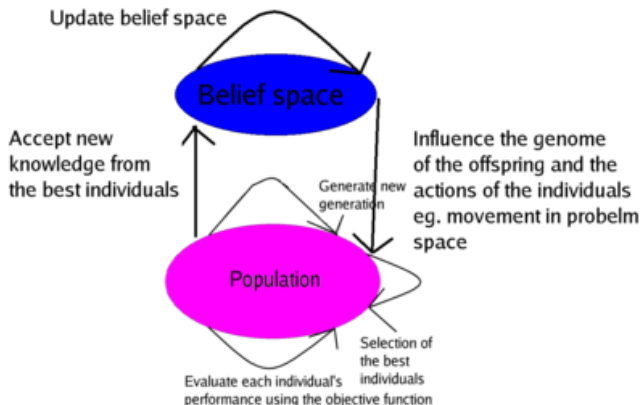
- Adaptive Clonal Selection (ACS)
- Optimization Immune Algorithm (opt-IMMALG)
- Optimized Artificial Immune Network (opt-aiNET)
- Optimization Immune Algorithm (opt-IA)
- Clonal Selection Algorithm (CLONALG, CLONALG1, CLONALG2)
- B-Cell Algorithm (BCA)
- Cloning, Information Gain, Aging (CLIGA)
- Immunological Algorithm (IA)

Other sources of inspiration

Particle Swarm Optimization



A Cultural algorithm



Pseudo-code

Cultural Algorithm

Begin

Initialize population space

Initialize belief space (domain specific knowledge, normative value-ranges...)

while termination condition not met **do**

 Perform actions on the individuals of the population

 Evaluate each individual by using the fitness function

 Select parents to produce a new generation of offspring

 Update the belief space via acceptance function

end while

End

However, Laguna noted that...

“A popular thrust of many research initiatives, and especially of publications designed to catch the public eye, is to associate various methods with processes found in nature.

The trend of using metaphors of nature embodies a wave of “New Romanticism”, reminiscent of the Romanticism of the 18th and 19th centuries...

The current fascination with natural phenomena as a foundation for problem-solving methods undoubtedly is fueled by our sense of mystery concerning the ability of such phenomena to generate outcomes that are still far beyond our comprehension.

And...

Metaphors of nature have a place. They appear chiefly to be useful for spurring ideas to launch the first phases of an investigation.

As long as care is taken to prevent such metaphors from cutting off lines of inquiry beyond their scope, they provide a means for “dressing up” the descriptions of various meta-heuristics in a way that appeals to our instinct to draw parallels between simple phenomena and abstract designs.

It is up to prudence to determine when the symbolism of the New Romanticism obscures rather than illuminates the pathway to improved understanding.

Concluding that...

Within the realm of meta-heuristic design, there is a great deal we have yet to learn.

The issue of whether the analogies that underlie some of our models may limit or enhance our access to further discovery deserves careful reflection.”

Manuel Laguna

Conclusions

Metaheuristics have a wide range of applications:

- Complex optimization problems,
 - Structural (as well as parametric) identification problems,
 - Design applications
-
- They do not require much domain knowledge.
 - But many function evaluations (computer simulations) are usually required.
-
- Surrogate models (metamodels) can be used,
 - Hybridization with available techniques is simple,
 - Natural parallelization.

Conclusions

Metaheuristics have a wide range of applications:

- Complex optimization problems,
 - Structural (as well as parametric) identification problems,
 - Design applications
-
- They do not require much domain knowledge.
 - But many function evaluations (computer simulations) are usually required.
-
- Surrogate models (metamodels) can be used,
 - Hybridization with available techniques is simple,
 - Natural parallelization.

Conclusions

Metaheuristics have a wide range of applications:

- Complex optimization problems,
 - Structural (as well as parametric) identification problems,
 - Design applications
-
- They do not require much domain knowledge.
 - But many function evaluations (computer simulations) are usually required.
-
- Surrogate models (metamodels) can be used,
 - Hybridization with available techniques is simple,
 - Natural parallelization.

Conclusions

Metaheuristics have a wide range of applications:

- Complex optimization problems,
 - Structural (as well as parametric) identification problems,
 - Design applications
-
- They do not require much domain knowledge.
 - But many function evaluations (computer simulations) are usually required.
-
- Surrogate models (metamodels) can be used,
 - Hybridization with available techniques is simple,
 - Natural parallelization.

Thank you for your attention.

`hcbm@lncc.br`

`http://www.lncc.br/~hcbm`