

# **Interval methods for solving quantified nonlinear problems for control engineering and machine learning**

**Bartłomiej Jacek Kubica**

Institute of Information Technology  
Warsaw University of Life Sciences – SGGW  
Poland

Interval Online Seminar on  
Interval Methods in Control Engineering

19<sup>th</sup> of November 2021

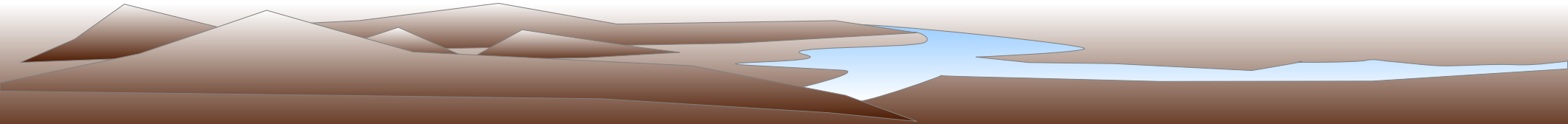


# Problem under solution

Find **all**  $x \in \mathbb{R}^n$ , satisfying the condition  $P(x)$ , i.e.,  
find the set  $\{x \in \mathbb{R}^n \mid P(x)\}$ .

where  $P(x)$  is a predicate formula with a free variable  $x$ ,  
i.e., free variables:  $x_1, \dots, x_n$ .

It can contain bound variables, also (we shall call them  
'parameters').



# Example problems

$$\{x \in X \mid h(x) = 0\}$$

$$\{x \in X \mid f(x) \in [\underline{y}, \bar{y}]\}$$

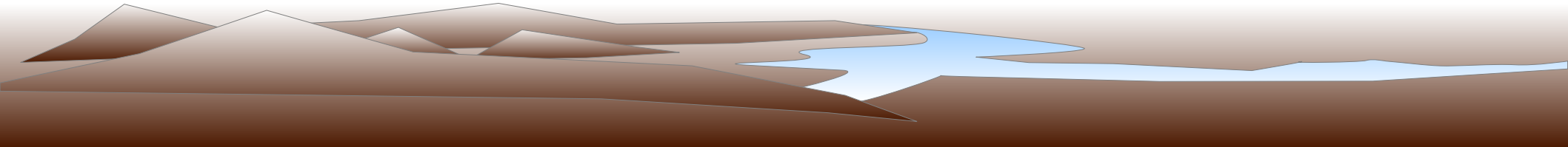
$$\{x \in X \mid (\forall t \in X) (f(x) \leq f(t))\}$$

$$\{x \in X \mid (\forall t \in X) (\forall i=1, \dots, N f_i(x) \leq f_i(t)) \vee (\exists i f_i(x) < f_i(t))\}$$

$$\{x \in X \mid (\forall i=1, \dots, n) (\forall x'_i \in x_i \subseteq \mathbb{R}^{k_i}) (f_i(x_{\setminus i}, x'_i) \geq f_i(x))\}$$

where:

$$X \subseteq \mathbb{R}^n, \quad h: \mathbb{R}^n \rightarrow \mathbb{R}^m, \quad f, f_1, \dots, f_N: \mathbb{R}^n \rightarrow \mathbb{R}$$



# Proposed algorithm

- V. Kreinovich, B. J. Kubica, *From computing sets of optima, Pareto sets and sets of Nash equilibria to general decision-related set computations*, Journal of Universal Computer Science, Vol. 16, pp. 2657 – 2685 (2010).
- B. J. Kubica, *A class of problems that can be solved using interval algorithms*, SCAN 2010, Computing, Vol. 94 (2-4), pp. 271 – 280 (2012).
- B. J. Kubica, *Interval Methods for Solving Nonlinear Constraint Satisfaction, Optimization and Similar Problems*, monograph, ISBN 978-3-030-13795-3, Springer, 2019.
- B. J. Kubica, *Interval methods for solving various kinds of quantified nonlinear problems*, in: *Beyond Traditional Probabilistic Data Processing Techniques: Interval, Fuzzy etc. Methods and Their Applications*, Springer, 2020.

# Proposed algorithm

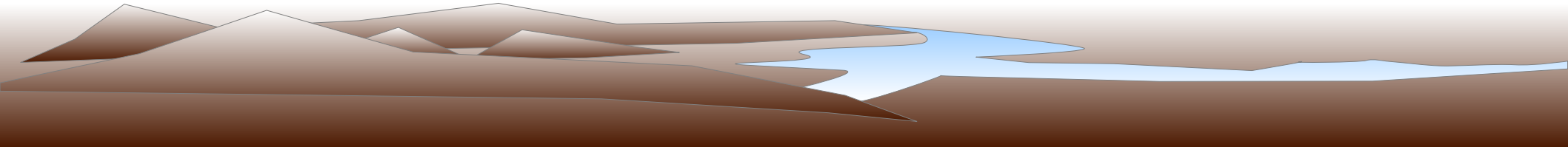
- We shall use the name **generalized branch-and-bound method** or **branch-and-bound type method**.
- Several algorithms, described in the literature, are its specific instances:
  - Branch-and-bound method.
  - Branch-and-prune method.
  - 'Nested' b&b or b&p (for parameters).
  - SIVIA – Set Inversion Via Interval Analysis (Jaulin, 1993).
  - PPS – Partitioning Parameter Set (дроблене параметров; Калмыков, 1982).

# Problem

- It is easy to understand that interval methods can be used to verify inequalities and their systems.
- Using proper theorems, we can also verify equations and their systems (interval Newton, Kantorovich, Miranda, Borsuk...).
- We can extend it to problems of the form:  
$$\{x \in X \mid (\forall t \in [t_0, t_f]) (f(x, t) \leq 0)\}$$
- But how can it be used to solve quantified problems, like global optimization:  
$$\{x \in X \mid (\forall t \in X) (f(x) \leq f(t))\}?$$

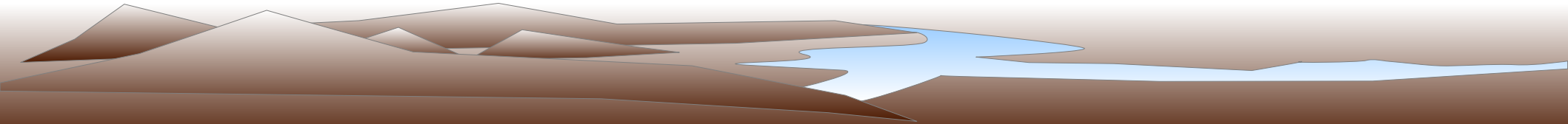
# Problem

- What are you talking about? Interval methods have been used for GO since forever!



# Problem

- What are you talking about? Interval methods have been used for GO since forever!
- Actually, not quite.
- The problem we solve in these algorithms is **not**:  
$$\{x \in X \mid (\forall t \in X) (f(x) \leq f(t))\}$$





# Problem

- What are you talking about? Interval methods have been used for GO since forever!
- Actually, not quite.

- The problem we solve in these algorithms is **not**:

$$\{x \in X \mid (\forall t \in X) (f(x) \leq f(t))\}$$

- No, really!

- It is, actually:

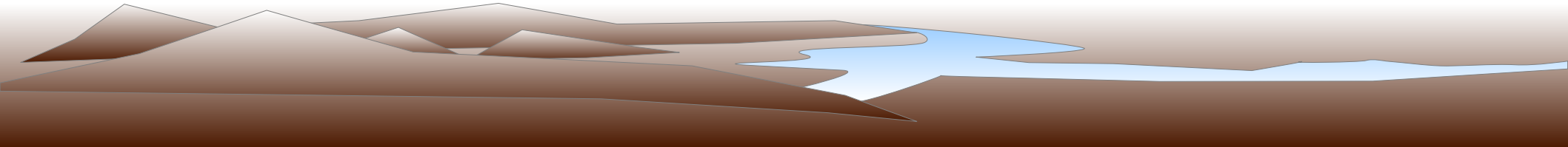
$$\{x \in X \mid (f(x) \leq y_0)\},$$

where  $y_0$  is a parameter, that we need to estimate

first:  $y_0 = f(x_0)$  for some  $x_0 \in X$  and  $\neg(\exists x' f(x') < f(x_0) - \epsilon)$

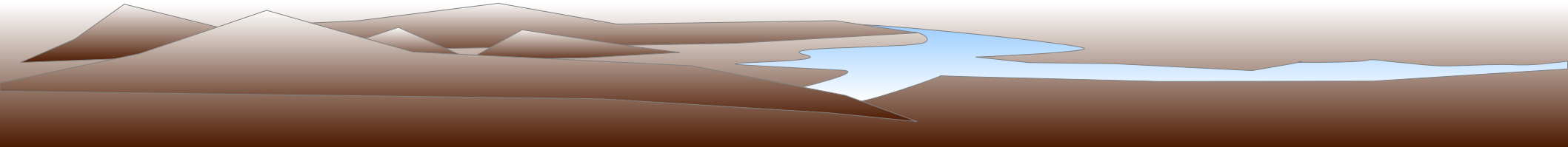
# What is the relationship between these two problems?

- They are not equivalent!
- $\{x \in X \mid (f(x) \leq y_o)\}$  is quantifier-free.
- $\{x \in X \mid (f(x) \leq y_o)\}$  is implied by  $\{x \in X \mid (\forall t \in X) (f(x) \leq f(t))\}$
- $\{x \in X \mid (f(x) \leq y_o)\}$  is weaker.



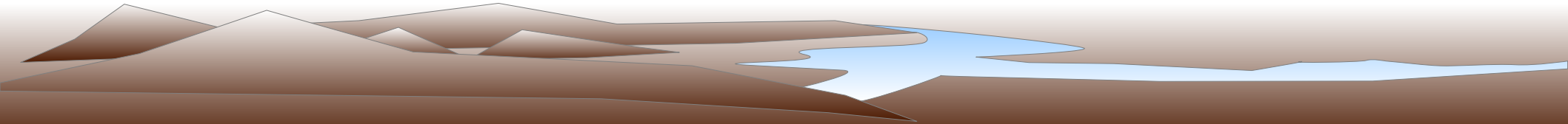
# What is the relationship between these two problems?

- They are not equivalent!
- $\{x \in X \mid (f(x) \leq y_o)\}$  is quantifier-free.
- $\{x \in X \mid (f(x) \leq y_o)\}$  is implied by  $\{x \in X \mid (\forall t \in X) (f(x) \leq f(t))\}$
- $\{x \in X \mid (f(x) \leq y_o)\}$  is weaker.
- $\{x \in X \mid (f(x) \leq y_o)\}$  is a result of **approximate quantifier elimination**.



# What is the relationship between these two problems?

- They are not equivalent!
- $\{x \in X \mid (f(x) \leq y_o)\}$  is quantifier-free.
- $\{x \in X \mid (f(x) \leq y_o)\}$  is implied by  $\{x \in X \mid (\forall t \in X) (f(x) \leq f(t))\}$
- $\{x \in X \mid (f(x) \leq y_o)\}$  is weaker.
- $\{x \in X \mid (f(x) \leq y_o)\}$  is a result of **approximate quantifier elimination**.
- It is the **Herbrand form** of the original problem:  
 $\{x \in X \mid (\forall t \in X) (f(x) \leq f(t))\}$



# Herbrand form

- Conjunction (for  $P(x) \equiv (\forall t) R(x, t)$ ) or disjunction (for  $P(x) \equiv (\exists t) R(x, t)$ ).
- The Herbrand form of a formula is commonly used in logic, to prove that this formula is a tautology.
  - A formula is a tautology when its Herbrand form is a tautology.
- But it can also be used for the approximation of a formula:

$$\begin{aligned}(\forall t) R(x, t) &\Rightarrow R(x, t_1) \wedge R(x, t_2) \wedge \dots \wedge R(x, t_N), \\(\exists t) R(x, t) &\Leftarrow R(x, t_1) \vee R(x, t_2) \vee \dots \vee R(x, t_N).\end{aligned}$$

# Herbrand form

- Most of the aforementioned problems had a universal quantifier:

$$P(x) \equiv (\forall t) R(x, t)$$

- Later in the talk, we shall meet a problem with existential quantifier, as well:

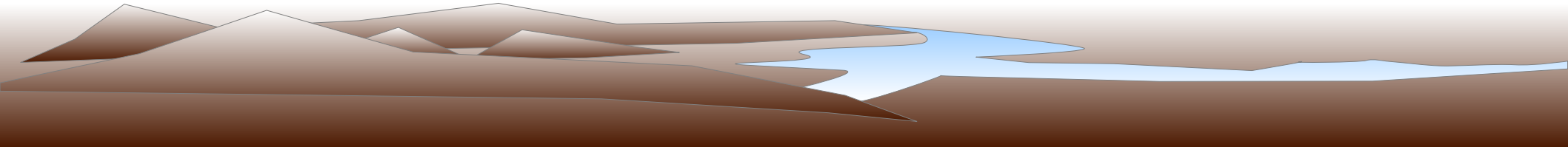
$$P(x) \equiv (\exists t) R(x, t)$$

- Please mind, the relationships are different:

$$\begin{aligned} (\forall t) R(x, t) &\Rightarrow R(x, t_1) \wedge R(x, t_2) \wedge \dots \wedge R(x, t_N), \\ (\exists t) R(x, t) &\Leftarrow R(x, t_1) \vee R(x, t_2) \vee \dots \vee R(x, t_N). \end{aligned}$$

# Comments

- Obviously, we have an analogous situation for other aforementioned problems: when approximating Pareto sets of a multicriteria problem, or seeking game solutions, we also use Herbrand expansions.
  - The Herbrand formula for these problems is more complex than for the simple, unicriterion global optimization.
  - We have to determine values of a higher number of parameters (also called 'shared quantities' in my other publications).



# Generic algorithm

$Lpos = \{\}; Lverif = \{\}; Lcheck = \{\};$

// Phase I

**while** (there are boxes to consider) **do**

    pop ( $\mathbf{x}$ );

    process ( $\mathbf{x}$ ); // using **interval tools**

**if** ( $\mathbf{x}$  was verified to contain a solution/a point satisfying some necessary conditions) **then** push ( $Lverif, \mathbf{x}$ );

**else if** ( $\mathbf{x}$  is verified not to contain solutions) **then**

**if** ( $\mathbf{x}$  may be necessary in phase II) **then** push ( $Lcheck, \mathbf{x}$ );

**else** discard  $\mathbf{x}$ ;

**end if**

**if** ( $\mathbf{x}$  was discarded or stored) **then** pop ( $\mathbf{x}$ );

**else if** ( $\text{diam}(\mathbf{x}) < \epsilon$ ) **then** push ( $Lpos, \mathbf{x}$ );

**else**

        bisect ( $\mathbf{x}, \mathbf{x}1, \mathbf{x}2$ ); push ( $\mathbf{x}2$ );  $\mathbf{x} = \mathbf{x}1$ ;

**end if**

**end while**

// Phase II – verification of  $P(x)$  for stored solution candidates

**for each** ( $\mathbf{x}$  in  $Lverif \cup Lpos$ ) **do**

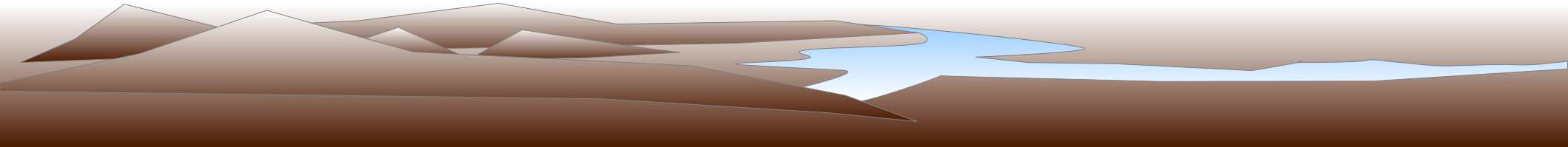
**if** ( $\mathbf{x}$  does not contain a solution) **then** discard  $\mathbf{x}$ ;

**end for each**



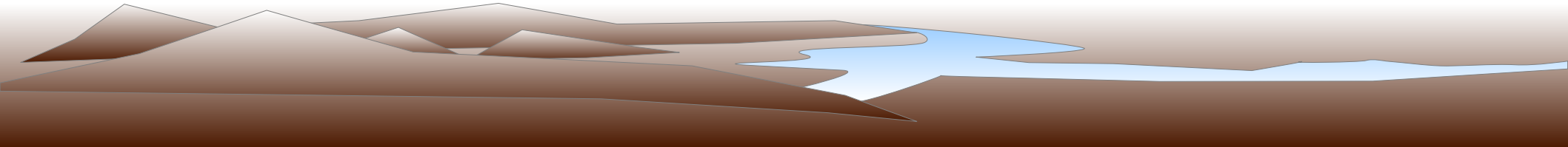
# What does the algorithm result in?

- Two lists:
  - **Lverif** – the list of boxes verified (certified) to contain a solution,
  - **Lpos** – the list of boxes possibly containing a solution.
- What conditions have to be verified so that the solution was `verified`?
- There can be more than two lists, in general:
  - Some conditions are verified, some are not.
  - We classify points of the domain into more than two classes.



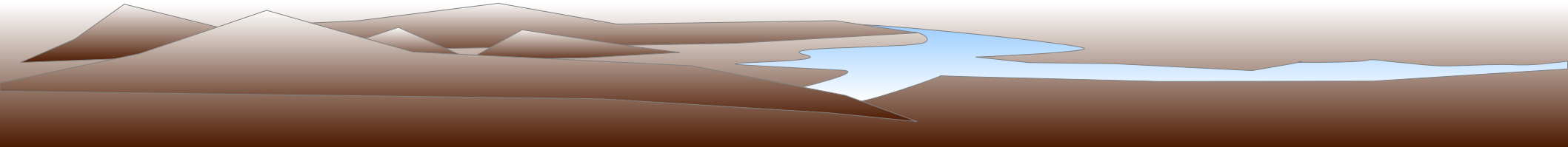
# Algorithm's other details

- In what order do we process boxes? Does the order matter?
- How do we store boxes?
- What tools do we use to process a box?
- What information is needed to process a box in phase I?
- What information is needed to verify a box in phase II?
- In particular, what boxes do we store in *Lcheck* – if any?



# Algorithm's other details

- All depends on the problem under consideration.
- Does the presence of solution in one area have influence on its presence elsewhere?
  - For equations systems – not really.
  - For optimization problems – it does (the optimum occurs to be local only, if we have found a better point elsewhere).
  - For seeking Pareto sets – also.
  - ...
- Obviously, rejection/reduction tests rely on the problem under consideration, also.

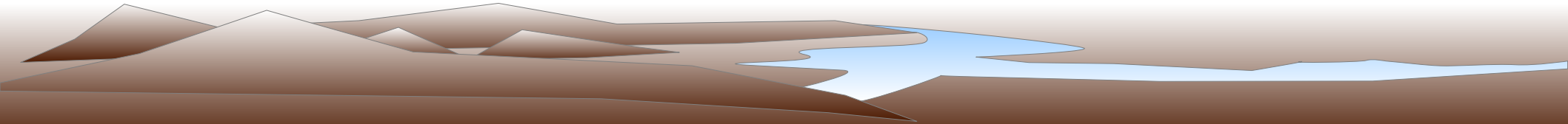


# Tools to process a single box

- Order of function approximation:
  - 0<sup>th</sup> order tools – comparing function values.
  - 1<sup>st</sup> order tools – use of gradients.
  - 2<sup>nd</sup> order tools – use of Hesse matrices.
  - Higher order tools?
- Operations:
  - Simply, comparing function values.
  - Several versions of the interval Newton operator (*componentwise*, GS) – on various levels.
  - Various constraint satisfaction methods (consistency enforcing, SIVIA, etc.).
  - Tests based on algebraic topology.

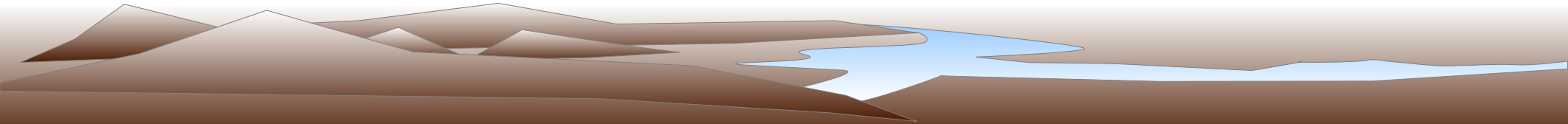
‣ ...

# How to make the branch-and-bound-type method efficient?



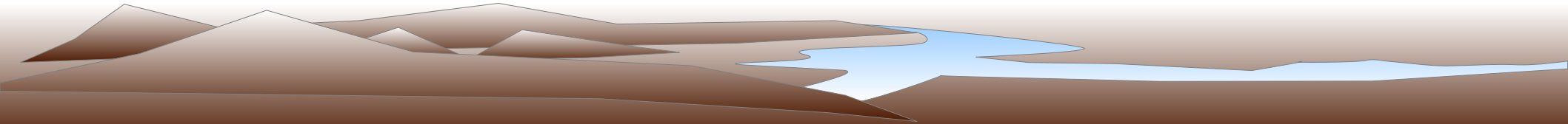
# How to make the branch-and-bound-type method efficient?

- There is a great deal of interval tools.
- **All** of them give guaranteed (verified) results.
- **None** of them are intelligent *per se*!



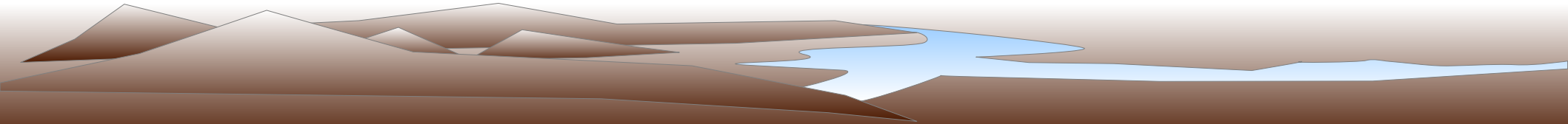
# How to make the branch-and-bound-type method efficient?

- There is a great deal of interval tools.
- **All** of them give guaranteed (verified) results.
- **None** of them are intelligent *per se*!
- It is crucial to develop an adequate **heuristic** to:
  - **choose** the interval tools adequate for a specific box,
  - **arrange** them,
  - **parameterize** them.



# How to make the branch-and-bound-type method efficient?

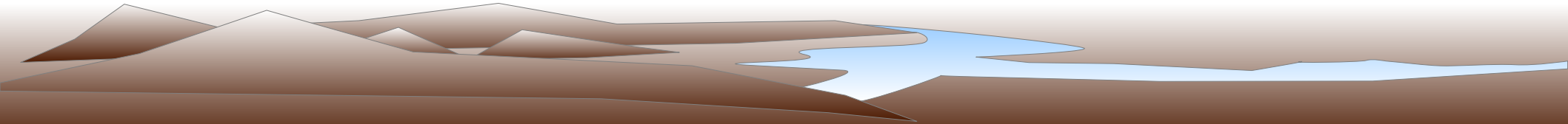
- The author devoted several papers to design heuristics for two problems:
  - **Nonlinear equations systems** – especially seeking all solutions of **underdetermined systems**.
  - **Seeking Pareto sets** of a multicriteria problem.
- Many tools & versions; several papers.
- The topic is often misunderstood...





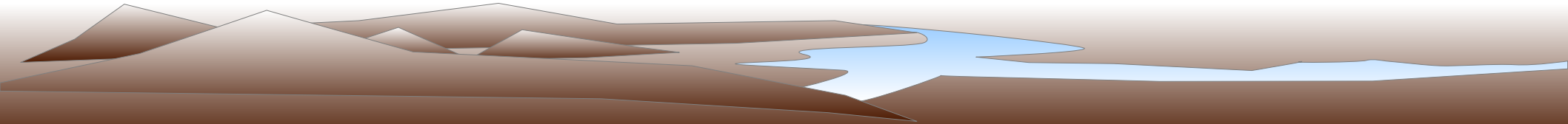
# Issues in designing heuristics

- Seemingly similar problems might require quite different heuristics.
- The Euclid space of higher dimension has a significantly different geometry than  $\mathbb{R}$  or  $\mathbb{R}^2$ .
  - The interior is small – most results are located near boundaries
  - Even inside a box with small diameter, the distances can be relatively large and the function can change significantly.
  - Bisection hardly reduces distances in the box.



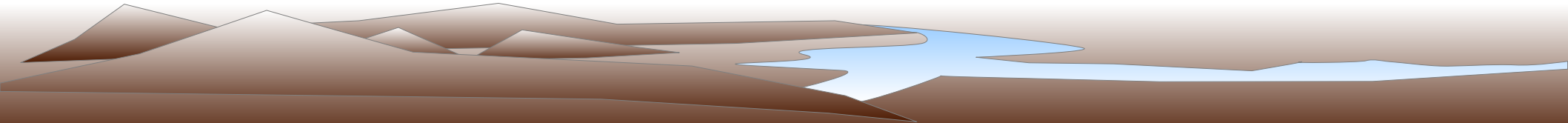
# Bisection

- Often, it is assumed that bisection should minimize the diameter of the objective function on resulting boxes.



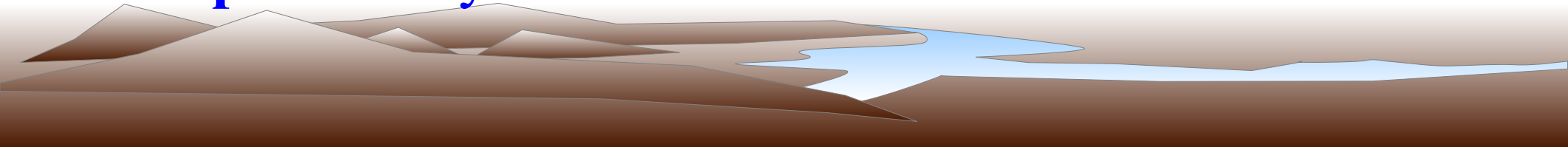
# Bisection

- Often, it is assumed that bisection should minimize the diameter of the objective function on resulting boxes.
- An example of such heuristic is MaxSmear (Shary, 1992; Ratz, 1992; Ratz & Csendes, 1995).
  - Works very well for optimization problems.
  - Works reasonably well for well-determined equations systems.
  - Fails miserably for underdetermined systems.



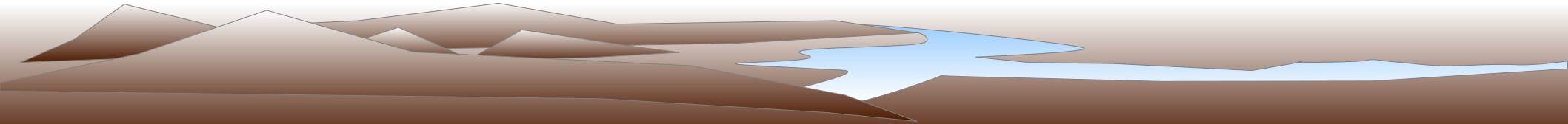
# Bisection

- Often, it is assumed that bisection should minimize the diameter of the objective function on resulting boxes.
- An example of such heuristic is MaxSmear (Shary, 1992; Ratz, 1992; Ratz & Csendes, 1995).
  - Works very well for optimization problems.
  - Works reasonably well for well-determined equations systems.
  - Fails miserably for underdetermined systems.
- In my opinion, the objective of bisection should be defined in a different way: **give boxes that are easy to process by the used interval tools.**



# Bisection

- For equations solving, the main tool is some kind of the interval Newton operator.
- So, for a single equation in two variables, it might seem reasonable to choose the **minimal smear**.
- But the convergence...
- A proper policy should take into account several criteria.
- For several, advanced tools, such a policy cannot be too simple...



# Bisection

- For example, the heuristic of Kubica, 2012:

find index  $j_{max}$  and diameter  $w_{max}$  of the longest component;

find index  $j_{min}$  and diameter  $w_{min}$  of the shortest component;

find index  $j_{max\_nonred}$  and diameter  $w_{max\_nonred}$  of the longest component **not reduced** by the latest use of the Newton;

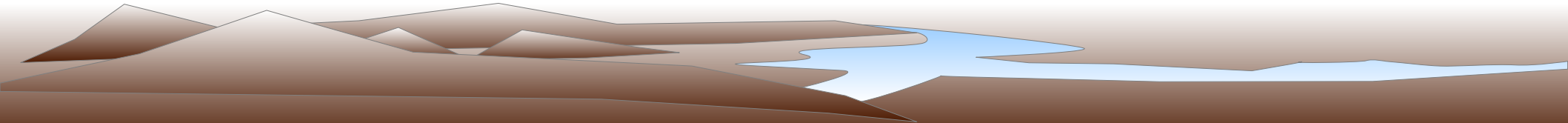
**if** ((Newton operator reduced no component) or ( $w_{max} > 1.5 * w_{max\_nonred}$ )) **then return**  $j_{max}$ ;

**else if** ( $w_{max\_nonred} > 8 * w_{min}$ ) **then return**  $j_{max\_nonred}$ ;

find index  $j$  and diameter  $w$  of the component with the smallest maximal absolute value in all rows of the Jacobi matrix;

**if** ( $w > 0.1$ ) **then return**  $j$ ;

**else return**  $j_{max\_nonred}$ ;



# Bisection

- For Pareto sets seeking, the proper heuristic is quite different:

find the index  $i$  of the criterion with maximal distance from the set in the criteria space;

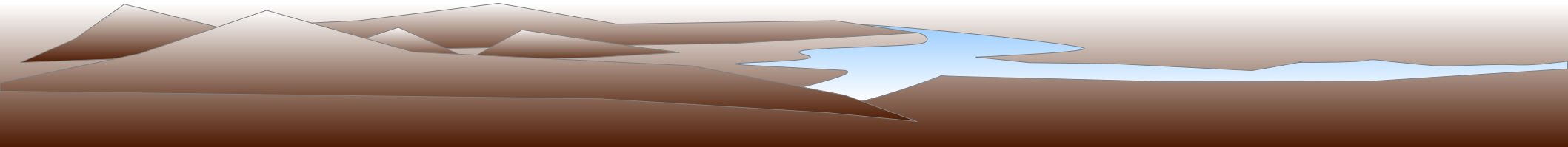
find the index  $j$  and diameter  $w$  of the component with maximal smear with respect to criterion  $i$ ;

find the index  $j\_max$  and diameter  $w\_max$  of the component with maximal diameter;

**if** ( $w\_max < 8 * w$ ) **then return**  $j$ ;

**else return**  $j\_max$ ;

- Reasons: different interval tools, used in the algorithm.



# Problems encountered in ML

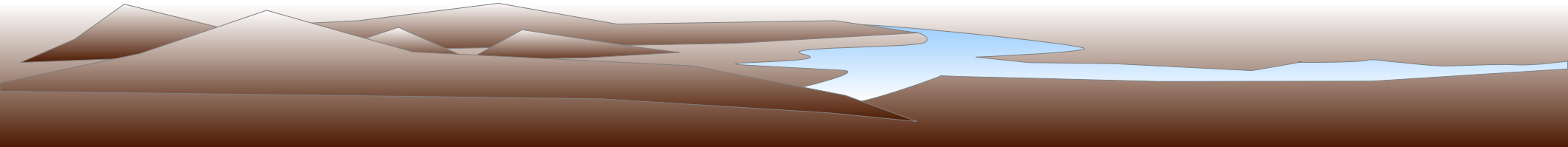
- The problem of training a classification/regression tools is often formulated as an optimization problem:  $\min_p \|f(\mathbf{x}_k) - \mathbf{y}_k\|$

$$\min_p \left( \sum_{k=1}^m (f(\mathbf{x}_k) - \mathbf{y}_k)^2 \right) \quad (\text{LSQ})$$

$$\min_p \left( - \sum_{k=1}^m f(\mathbf{x}_k) \log(\mathbf{y}_k) + (1 - f(\mathbf{x}_k)) \log(1 - \mathbf{y}_k) \right) \quad (\text{KL div})$$

- Alternatively, a CSP can be used:

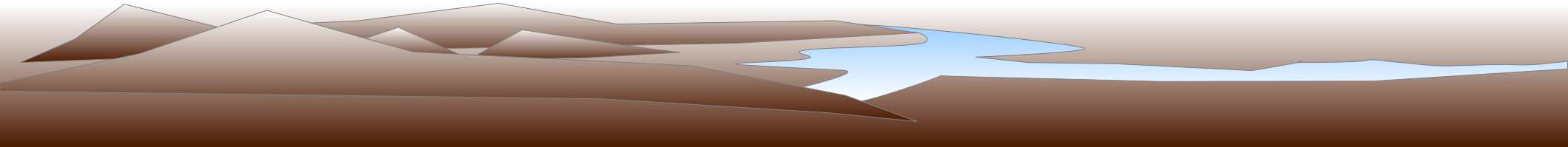
$$f(\mathbf{x}_k) \subseteq \mathbf{y}_k, \quad k = 1, \dots, m$$





# Problems encountered in ML

- Yet another possibility: find points satisfying as many constraints of the CSP, as possible.
  - Adequate in the presence of outliers.
- Another problem: find all points satisfying a fuzzy predicate.
  - More lists are needed there: for several alpha-cuts of the fuzzy solution set.
  - Other than that, the generalized branch-and-bound algorithm can be adopted with minor changes only.



# Problems encountered in ML

- A problem with existential quantifier: find points of a dynamical system, where it has periods (of an arbitrary length).

- In particular, we can consider it for a recurrent neural network (Hopfield, LSTM, Boltzmann machine...).

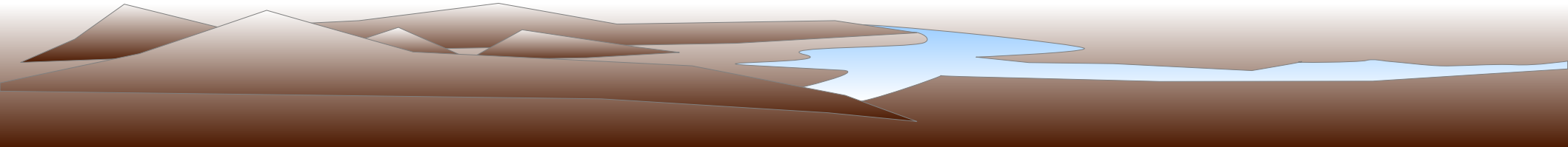
- Considering a dynamical system of the form:  $x_{k+1} = f(x_k)$ , we can formulate the problem, for instance, as follows:

$$\{x \in X \mid (\exists x_1, \dots, x_n \in X) \wedge (x_1 = f(x)) \wedge (x_2 = f(x_1)) \wedge \dots \wedge (x = f(x_n))\}$$

- We consider here cycles of length at most  $n$ .

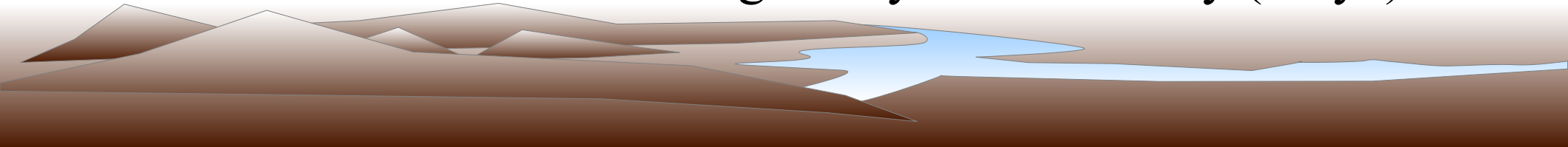
- How about longer ones?

- A proper data structure: graph showing possible transitions of values.



# Comments and summary

- Interval methods provide us with a toolset for solving a large variety of problems, difficult to **formulate** or **handle** using other methods.
- The author is not sure about the general requirements for the formal system, for which we can use interval methods, but it is a large class of systems (the notion of a H-continuous function might be related).
- We can adopt the **generalized branch-and-bound algorithm** for very sophisticated and versatile problems.
  - This process can hardly be automated, as we need proper heuristics that can be designed by a human only (only?).



# Comments and summary

- This has some impact on our understanding of what computers can solve, and what human are needed to solve.
- This is fascinating, and it (probably) has some philosophical consequences.
- Nevertheless, whenever possible to reduce the problem to either optimization or CSP, it had better be done – for **efficiency reasons**.
- Oh, and it is worth noting that branch-and-bound type algorithms parallelize well and can utilize current hardware architectures efficiently.

