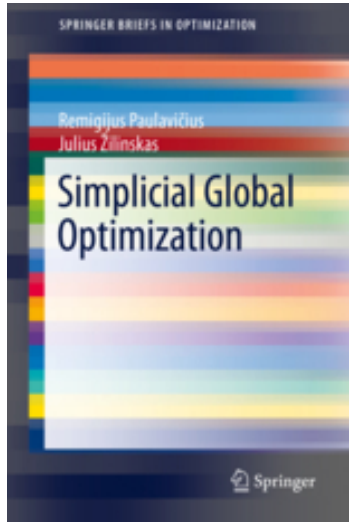
Simplicial Global Optimization

Julius Žilinskas

Vilnius University, Lithuania

September 10, 2017



http://web.vu.lt/mii/j.zilinskas

Global optimization

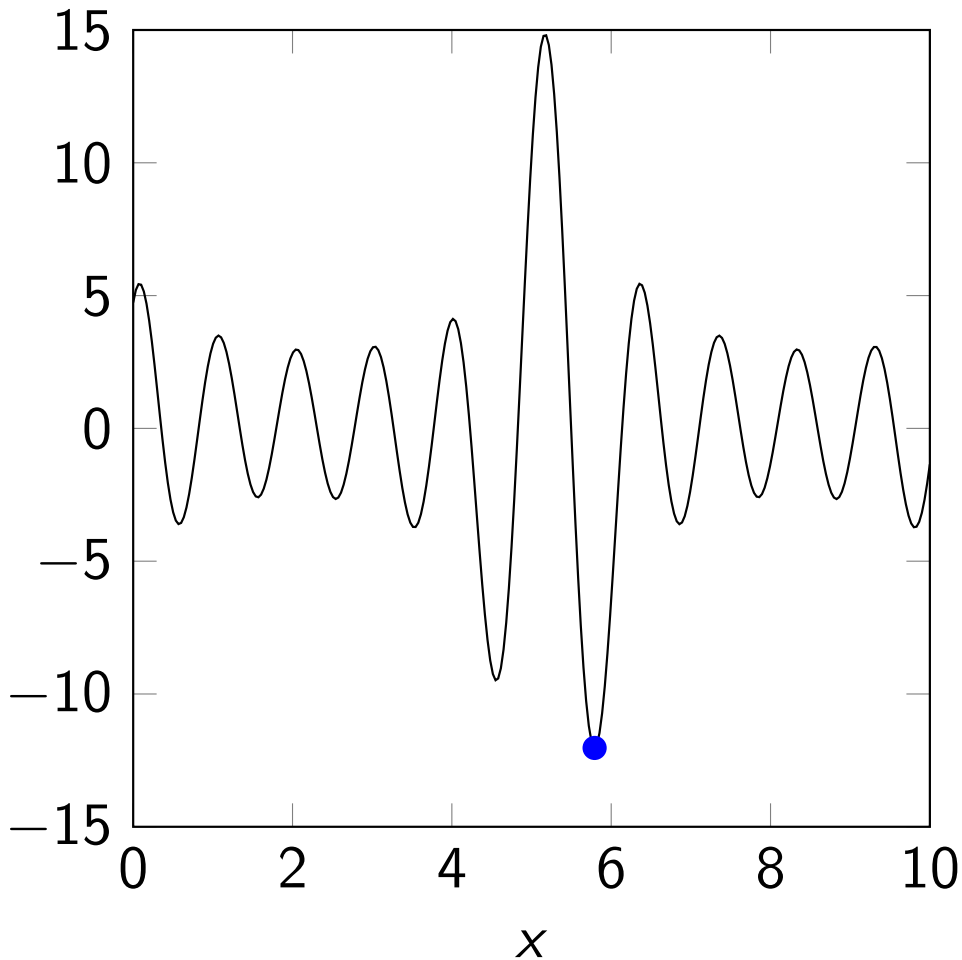
$$\text{Find } f^* = \min_{\mathbf{x}\in\mathbb{A}} f(\mathbf{x}) \text{ and } \mathbf{x}^* \in \mathbb{A},\ f(\mathbf{x}^*) = f^*, \text{ where } \mathbb{A} \subseteq \mathbb{R}^n.$$

Example:

- $n = 1$;
- $\mathbb{A} = [0, 10]$;
- Objective function

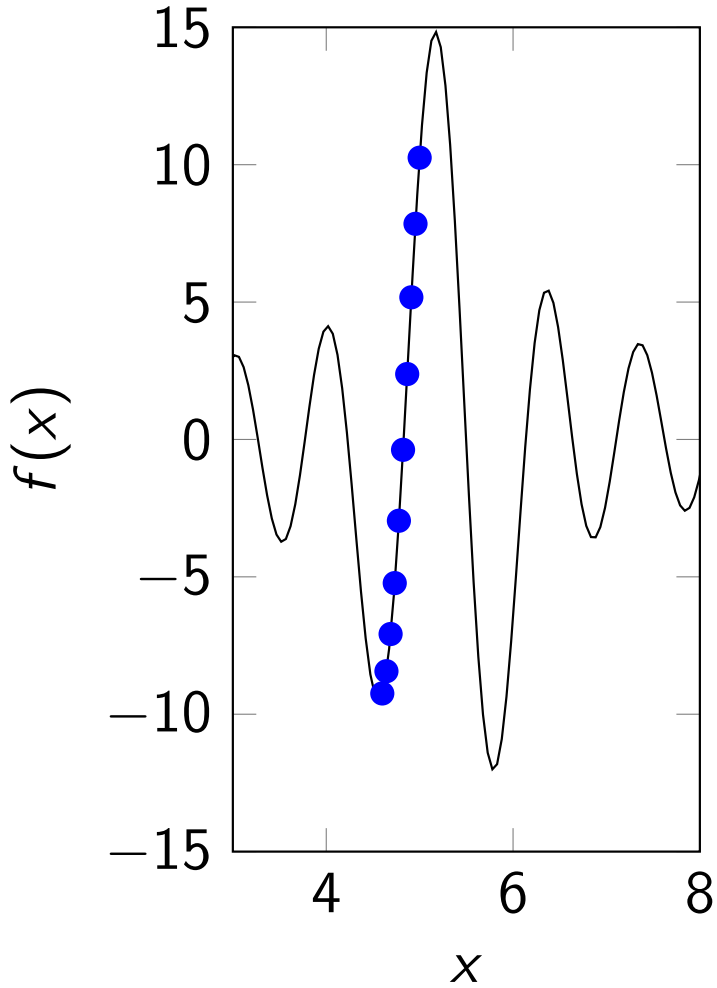
$$f(x) = -\sum_{j=1}^{5} j \sin((j+1)x+j);$$

- $f^* = -12.0312$;
- $x^* = 5.7918$.



Local optimization

- A point $\mathbf{x}^*$ is a local minimum point if $f(\mathbf{x}^*) \leq f(\mathbf{x})$ for $\mathbf{x} \in N$, where N is a neighborhood of $\mathbf{x}^*$.
- A local minimum point can be found stepping in the direction of steepest descent.
- Without additional information one cannot say if the local minimum is global.
- How do we know if we are in the deepest hole?



Algorithms for global optimization

- With guaranteed accuracy:
  - Lipschitz optimization,
  - Branch and bound algorithms.
- Randomized algorithms and heuristics:
  - Random search,
  - Multi-start,
  - Clustering methods,
  - Generalized descent methods,
  - Simulated annealing,
  - Evolutionary algorithms (genetic algorithms, evolution strategies),
  - Swarm-based optimization (particle swarm, ant colony).

Criteria of performance of global optimization algorithms

- Speed:
  - Time of optimization,
  - Number of objective function (and sometimes gradient, bounding, and other functions) evaluations,
  - Both criteria are equivalent when objective function is "expensive" – its evaluation takes much more time than auxiliary computations of an algorithm.
- Best function value found.
- Reliability – how often problems are solved with prescribed accuracy.

Parallel global optimization

- Global optimization problems are classified difficult in the sense of the algorithmic complexity theory. Global optimization algorithms are computationally intensive.
- When computing power of usual computers is not sufficient to solve a practical global optimization problem, high performance parallel computers and computational grids may be helpful.
- An algorithm is more applicable in case its parallel implementation is available, because larger practical problems may be solved by means of parallel computations.

Criteria of performance of parallel algorithms

- Efficiency of the parallelization can be evaluated using several criteria taken into account the optimization time and the number of processors.
- A commonly used criterion of parallel algorithms is the speedup:

$$s_p = \frac{t_1}{t_p},$$

where t_p is the time used by the algorithm implemented on p processors.

- The speedup divided by the number of processors is called the efficiency:

$$e_p = \frac{s_p}{p}.$$

Covering methods

- Detect and discard non-promising sub-regions using
  - interval arithmetic $\left\{f\left(X\right)|X \in \overline{X}, \overline{X} \in [\mathbb{R}, \mathbb{R}]^n\right\} \subseteq \overline{f}\left(\overline{X}\right)$,
  - Lipschitz condition $|f(x) - f(y)| \leq L||x - y||$,
  - convex envelopes,
  - statistical estimates,
  - heuristic estimates,
  - ad hoc algorithms.
- May be implemented using branch and bound algorithms.
- May guarantee accuracy for some classes of problems: $f^* \leq \min_{x \in D} f(x) + \epsilon$.

Branch and bound

- An iteration of a classical branch and bound processes a node in the search tree representing a not yet explored subspace.
- Iteration has three main components: selection of the node to process, branching and bound calculation.
- The bound for the objective function over a subset of feasible solutions should be evaluated and compared with the best objective function value found.
- If the evaluated bound is worse than the known function value, the subset cannot contain optimal solutions and the branch describing the subset can be pruned.
- The fundamental aspect of branch and bound is that the earlier the branch is pruned, the smaller the number of complete solutions that will need to be explicitly evaluated.

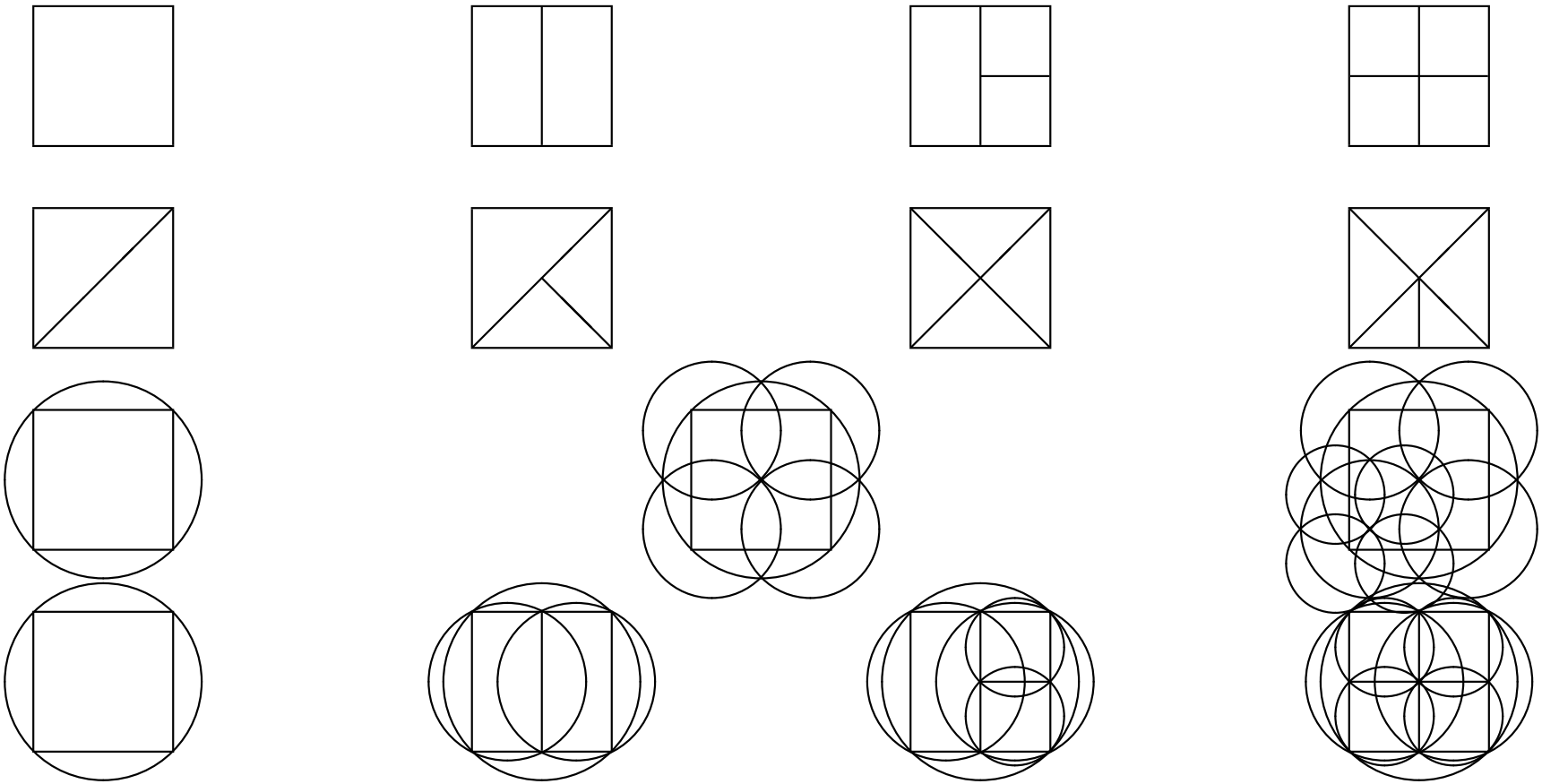
General algorithm for partition and branch-and-bound

Cover D by $L = \{L_j | D \subseteq \bigcup L_j, j = \overline{1,m}\}$ using **covering rule**.
while $L \neq \emptyset$ **do**
  Choose $l \in L$ using **selection rule**, $L \leftarrow L \backslash \{l\}$.
  if bounding rule is satisfied for l **then**
    Partition l into p subsets l_j using **subdivision rule**.
    for $j = 1, \ldots, p$ **do**
      if bounding rule is satisfied for l_j **then**
        $L \leftarrow L \bigcup \{l_j\}$.
      end if
    end for
  end if
end while

Rules of branch and bound algorithms for global optimization

- The **rules** of **covering** and **subdivision** depend on type of partitions used: hyper-rectangular, simplicial, etc.
- **Selection rules**:
  - Best first, statistical – the best candidate: priority queue.
  - Depth first – the youngest candidate: stack or without storing.
  - Breadth first – the oldest candidate: queue.
- **Bounding rule** describes how the bounds for minimum are found. For the upper bound the best currently found function value may be accepted. The lower bound may be estimated using
  - Convex envelopes of the objective function.
  - Lipschitz condition $|f(x) - f(y)| \leq L||x - y||$.
  - Interval arithmetic $\left\{ f\left(X\right) | X \in \underline{X}, \overline{X} \in [\mathbb{R}, \mathbb{R}]^n \right\} \subseteq \underline{f}\left(\overline{X}\right)$.

Rules of covering rectangular feasible region and branching



Lipschitz optimization

- Lipschitz optimization is one of the most deeply investigated subjects of global optimization. It is based on the assumption that the slope of an objective function is bounded.
- The multivariate function $f : D \to \mathbb{R}, D \subset \mathbb{R}^n$ is said to be Lipschitz if it satisfies the condition

$$|f(x) - f(y)| \leq L\|x - y\|, \quad \forall x, y \in D,$$

  where $L > 0$ is a constant called Lipschitz constant, the domain D is compact and $\|\cdot\|$ denotes a norm.
- Branch and bound algorithm with Lipschitz bounds may be built: if the evaluated bound is worse than the known function value, the sub-region cannot contain optimal solutions and the branch describing it can be pruned.

Lipschitz bounds

The efficiency of the branch and bound technique depends on the bound calculation.

- The upper bound for the minimum is the smallest value of the function at the vertices x_v: $LB(D) = \min_{x_v} f(x_v)$.
- The lower bound for the minimum is evaluated exploiting Lipschitz condition:

$$f(x) \leq f(y) + L\|x - y\|.$$

- The lower bound can be derived as

$$\mu_1 = \max_{x_v \in V(I)} \left\{ f(x_v) - L \max_{x \in I} \|x - x_v\| \right\}.$$

Comparison of selection strategies in Lipschitz optimization

Paulavičius, Žilinskas, Grothey, Optimization Letters, 4(2), 173-183, 2010, doi:10.1007/s11590-009-0156-3

Average number of function evaluations

Dimension	Best First	Statistical	Depth First	Breadth First
$n = 2$	9064	9100	9716	9068
$n = 3$	1217072	1217206	1222060	1217509
$n = 4$	879656	879276	880478	879851
$n = 5, 6$	3939678	3923552	3925984	3960629

Average total amount of simplices

Dimension	Best First	Statistical	Depth First	Breadth First
$n = 2$	18126	18199	19430	18133
$n = 3$	2434138	2434406	244114	2435013
$n = 4$	1759290	1758531	1760934	1759680
$n = 5, 6$	7878996	7846744	7851609	7920898

Average maximal number of simplices in the list

Dimension	Best First	Statistical	Depth First	Breadth First
$n = 2$	2709	640	12	3135
$n = 3$	274960	102734	23	423552
$n = 4$	168927	105814	36	249651
$n = 5, 6$	704862	136589	374	903555

Average execution time

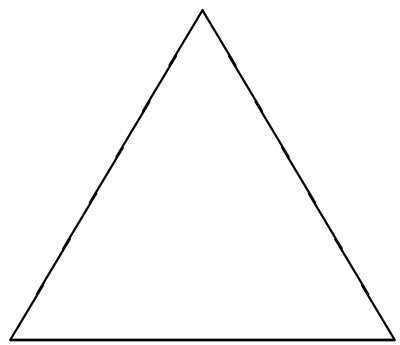
Dimension	Best First	Statistical	Depth First	Breadth First
$n = 2$	0.05	0.05	0.05	0.05
$n = 3$	17.99	16.00	12.41	13.64
$n = 4$	31.65	31.68	26.86	27.53
$n = 5, 6$	1592.19	1568.13	1577.68	1555.55

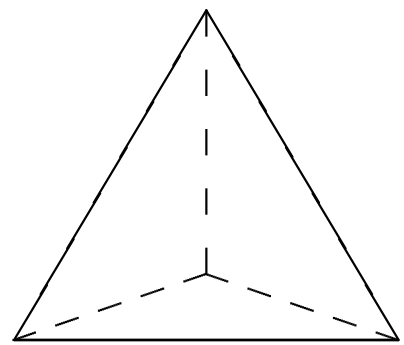
Average ratio $t_{BestFound}/t_{AllTime}$

Dimension	Best First	Statistical	Depth First	Breadth First
$n = 2$	0.234	0.236	0.365	0.408
$n = 3$	0.140	0.008	0.209	0.306
$n = 4$	0.142	0.035	0.240	0.472
$n = 5, 6$	0.074	0.000	0.005	0.077

Simplex

- An n-simplex is the convex hull of a set of $(n+1)$ affinely independent points in n-dimensional Euclidean space.
- An one-simplex is a segment of line, a two-simplex is a triangle, a three-simplex is a tetrahedron.





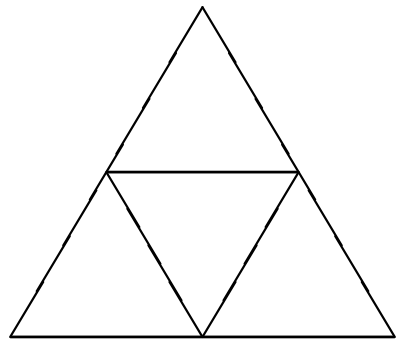
Simplicial vs rectangular partitions

- A simplex is a polyhedron in n-dimensional space, which has the minimal number of vertices.
- Simplicial partitions are preferable when values of the objective function at vertices of partitions are used to evaluate sub-regions.
- Numbers of function evaluations in Lipschitz global optimization with rectangular and simplicial partitions:

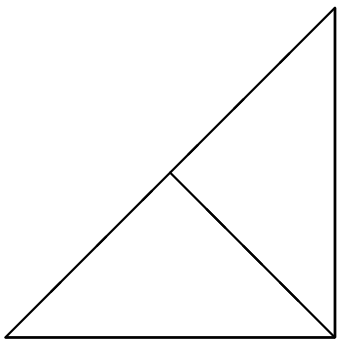
test function	1	2	3	4	5	6	7
rectangular	643	167	3531	45	73	969	7969
simplicial	611	132	2185	70	80	838	3117
test function	8	9	10	11	12	13	
rectangular	301	13953	1123	2677	12643	15695	
simplicial	244	3773	848	1566	4001	4084	

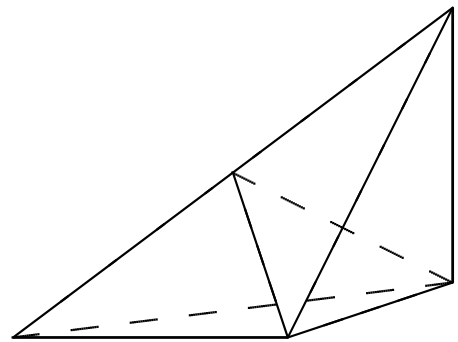
Subdivision of simplices

- Into similar simplices.



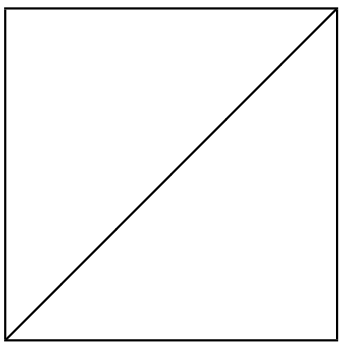
- Into two through the middle of the longest edge.

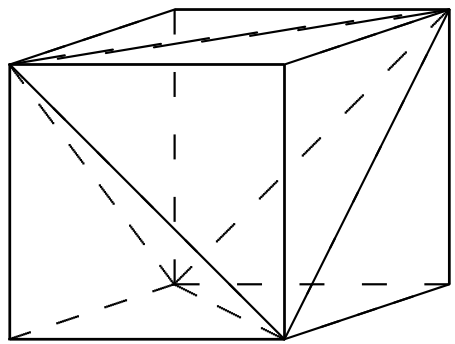


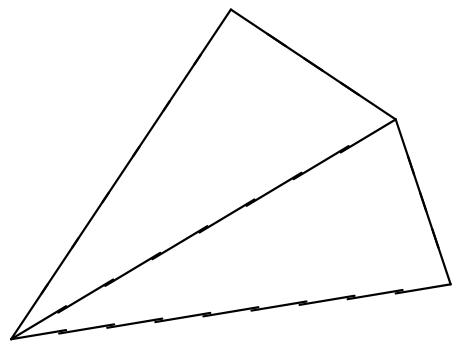


Covering of feasible region

- Very often a feasible region in global optimization is a hyper-rectangle defined by intervals of variables.
- A feasible region is face-to-face vertex triangulated: it is partitioned into n-simplices, where the vertices of simplices are also the vertices of the feasible region.
- A feasible region defined by linear inequality constraints may be vertex triangulated. In this way constraints are managed by initial covering.



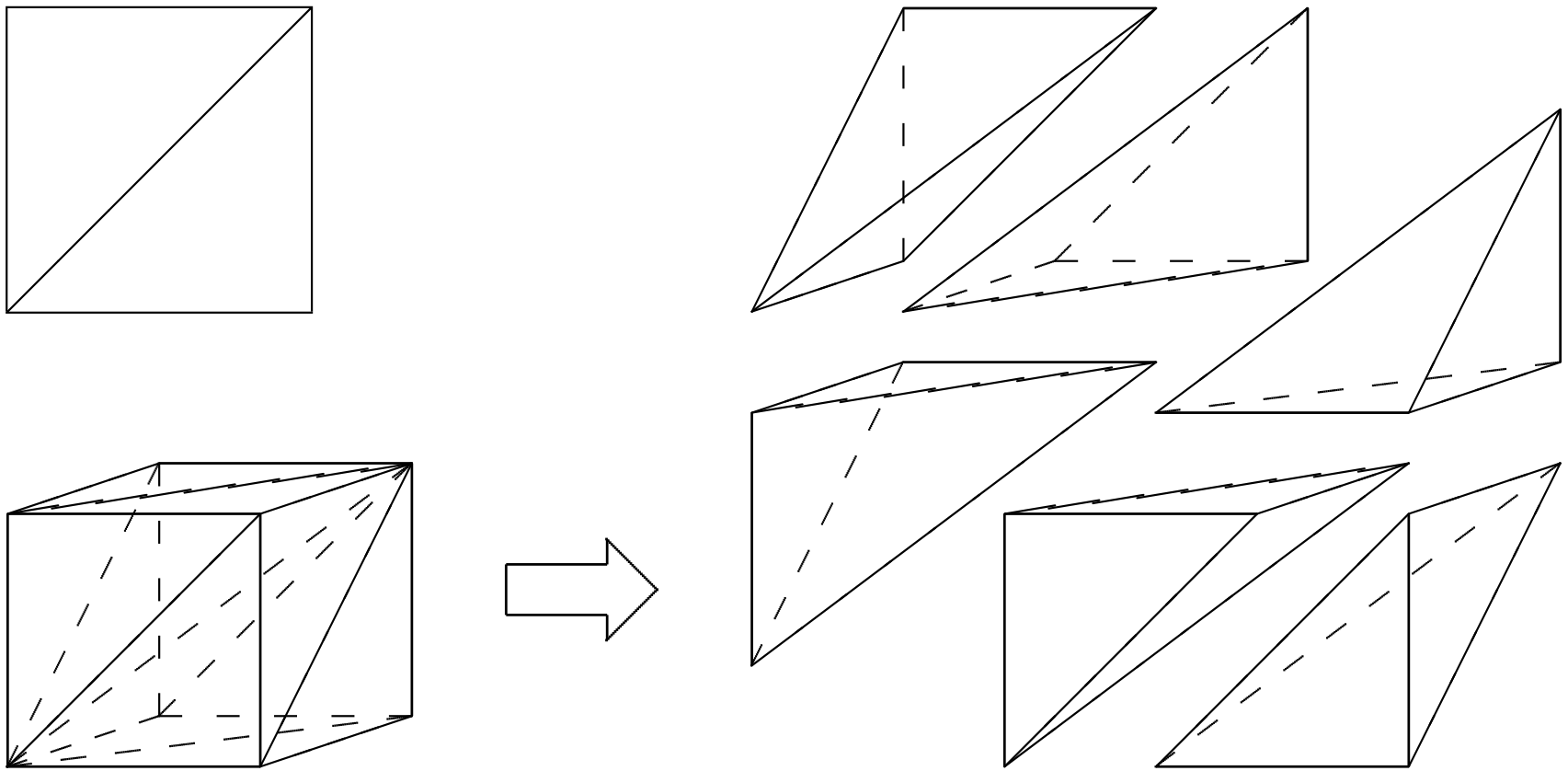




Combinatorial approach for vertex triangulation of a hyper-rectangle

- General – for any n.
- Deterministic, the number of simplices is known in advance – $n!$.
- All simplices are of equal hyper-volume – $1/n!$ of the hyper-volume of the hyper-rectangle.
- By adding just one point at the middle of diagonal of the hyper-rectangle each simplex may be subdivided into two.
- May be easily parallelized.

Examples of combinatorial vertex triangulation of two- and three-dimensional hyper-rectangles



Algorithm for combinatorial vertex triangulation of hyper-rectangle

for $\tau =$ equals one of all permutations of $1, \ldots, n$ **do**
  for $j = 1, \ldots, n$ **do**
    $v_{1j} \leftarrow D_{j1}$
  end for
  for $i = 1, \ldots, n$ **do**
    for $j = 1, \ldots, n$ **do**
      $v_{(i+1)j} \leftarrow v_{ij}$
    end for
    $v_{(i+1)\tau_i} \leftarrow D_{\tau_i 2}$
  end for
end for

Combinatorial vertex triangulation of a unit cube

$$\tau = \{1,2,3\} \qquad v = \left\{ \begin{array}{ccc} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{array} \right\}$$

$$\tau = \{1,3,2\} \qquad v = \left\{ \begin{array}{ccc} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{array} \right\}$$

$$\tau = \{2,1,3\} \qquad v = \left\{ \begin{array}{ccc} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{array} \right\}$$

$$\tau = \{2,3,1\} \qquad v = \left\{ \begin{array}{ccc} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{array} \right\}$$

$$\tau = \{3,1,2\} \qquad v = \left\{ \begin{array}{ccc} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{array} \right\}$$

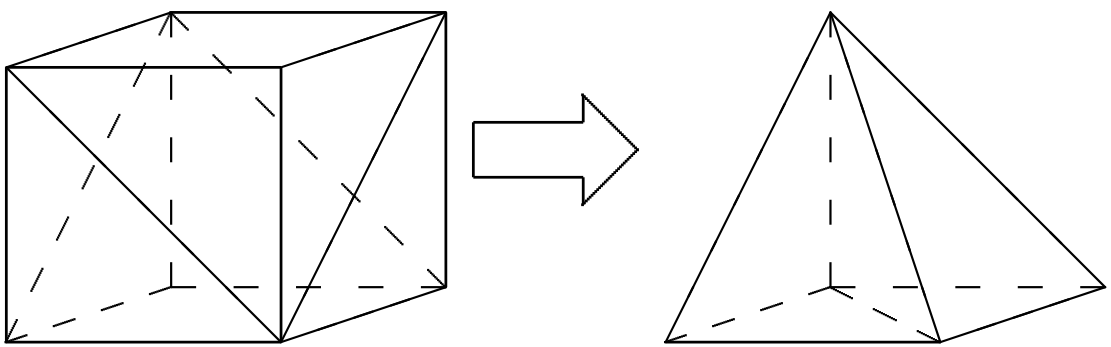
$$\tau = \{3,2,1\} \qquad v = \left\{ \begin{array}{ccc} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{array} \right\}$$

Parallel algorithm for combinatorial vertex triangulation

for $k = \lfloor n!\,rank/size \rfloor$ **to** $\lfloor n!(rank+1)/size \rfloor - 1$ **do**
  for $j = 1, \ldots, n$ **do** $\tau_j \leftarrow j$ **end for**
  $c \leftarrow 1$
  for $j = 2, \ldots, n$ **do**
    $c \leftarrow c(j-1)$
    swap $\tau_{j-\lfloor k/c \rfloor \% j}$ with τ_j
  end for
  for $j = 1, \ldots, n$ **do** $v_{0j} \leftarrow D_{j1}$ **end for**
  for $i = 1, \ldots, n$ **do**
    for $j = 1, \ldots, n$ **do** $v_{(i+1)j} \leftarrow v_{ij}$ **end for**
    $v_{(i+1)\tau_i} \leftarrow D_{\tau_i 2}$
  end for
end for

Vertex triangulation of feasible region defined by linear inequality constraints I

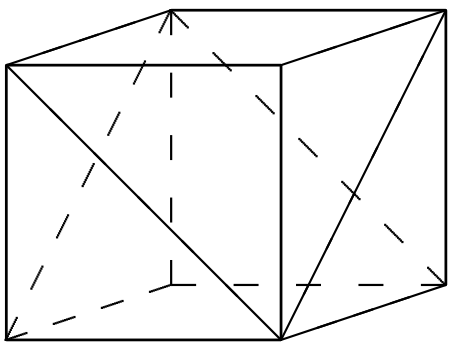
$$\begin{aligned} &\min f(x), \\ &\text{s.t.} \quad 0 \le x_i \le 1, \\ &\qquad\quad x_1 + x_2 \le 1, \\ &\qquad\quad x_2 - x_3 \le 0. \end{aligned}$$

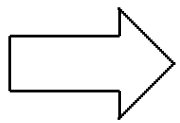


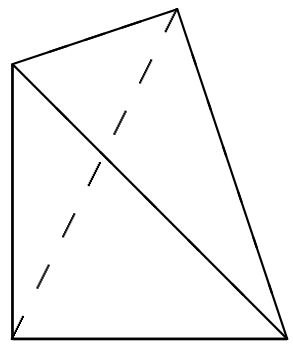
$$\left\{ \begin{array}{l} x_1 = 0, x_2 = 0, x_3 = 0 \\ x_1 = 0, x_2 = 0, x_3 = 1 \\ \cancel{x_1 = 0, x_2 = 1, x_3 = 0} \\ x_1 = 0, x_2 = 1, x_3 = 1 \\ x_1 = 1, x_2 = 0, x_3 = 0 \\ x_1 = 1, x_2 = 0, x_3 = 1 \\ \cancel{x_1 = 1, x_2 = 1, x_3 = 0} \\ \cancel{x_1 = 1, x_2 = 1, x_3 = 1} \end{array} \right\}$$

Vertex triangulation of feasible region defined by linear inequality constraints II

$$\begin{aligned} &\min f(x), \\ &\text{s.t.} \quad 0 \le x_i \le 1, \\ &\qquad x_1 + x_2 \le 1, \\ &\qquad -x_2 + x_3 \le 0. \end{aligned}$$

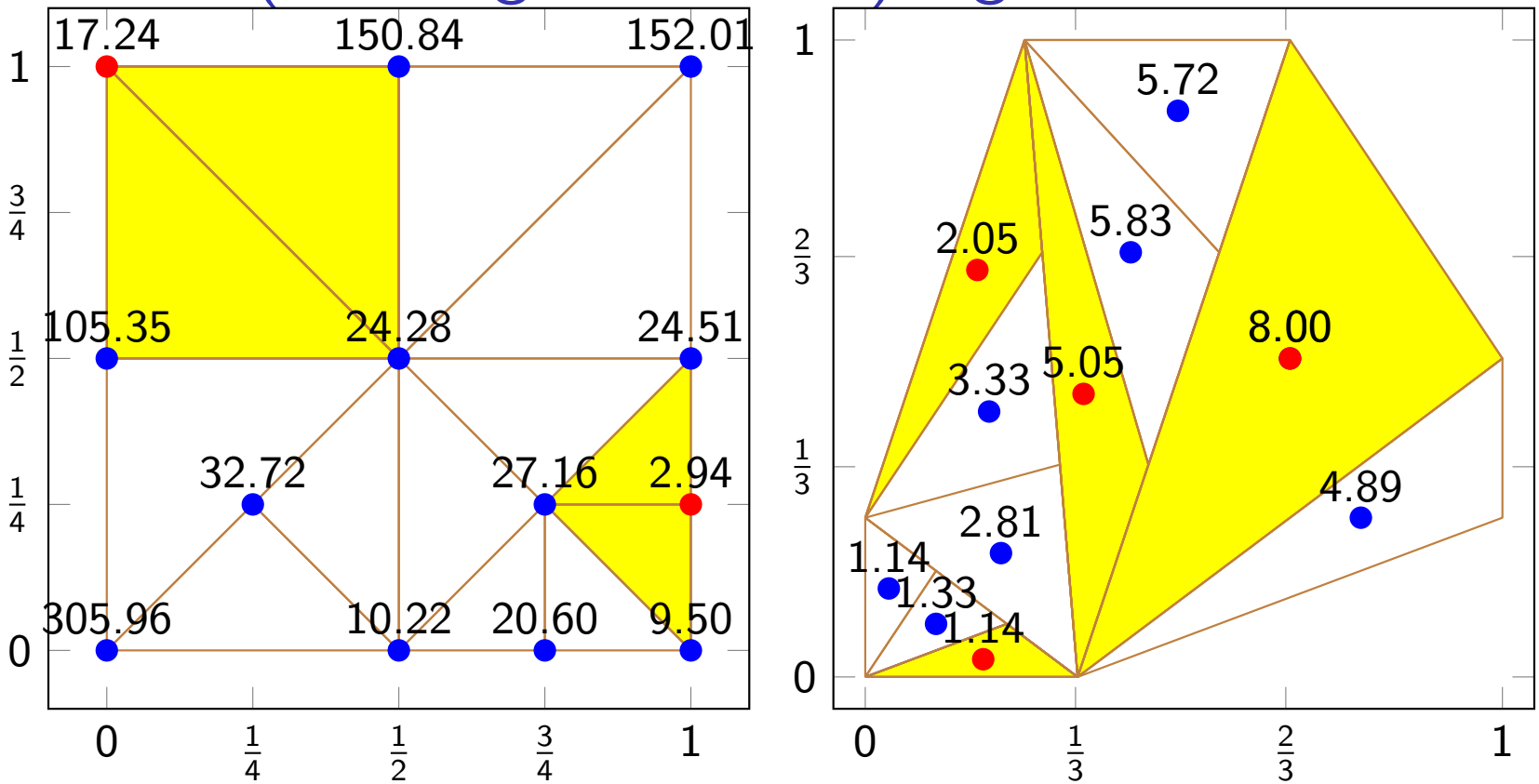






$$\left\{ \begin{array}{l} x_1 = 0, x_2 = 0, x_3 = 0 \\ \cancel{x_1 = 0, x_2 = 0, x_3 = 1} \\ x_1 = 0, x_2 = 1, x_3 = 0 \\ x_1 = 0, x_2 = 1, x_3 = 1 \\ x_1 = 1, x_2 = 0, x_3 = 0 \\ \cancel{x_1 = 1, x_2 = 0, x_3 = 1} \\ \cancel{x_1 = 1, x_2 = 1, x_3 = 0} \\ \cancel{x_1 = 1, x_2 = 1, x_3 = 1} \end{array} \right\}$$

DISIMPL (DIviding SIMPLices) algorithm

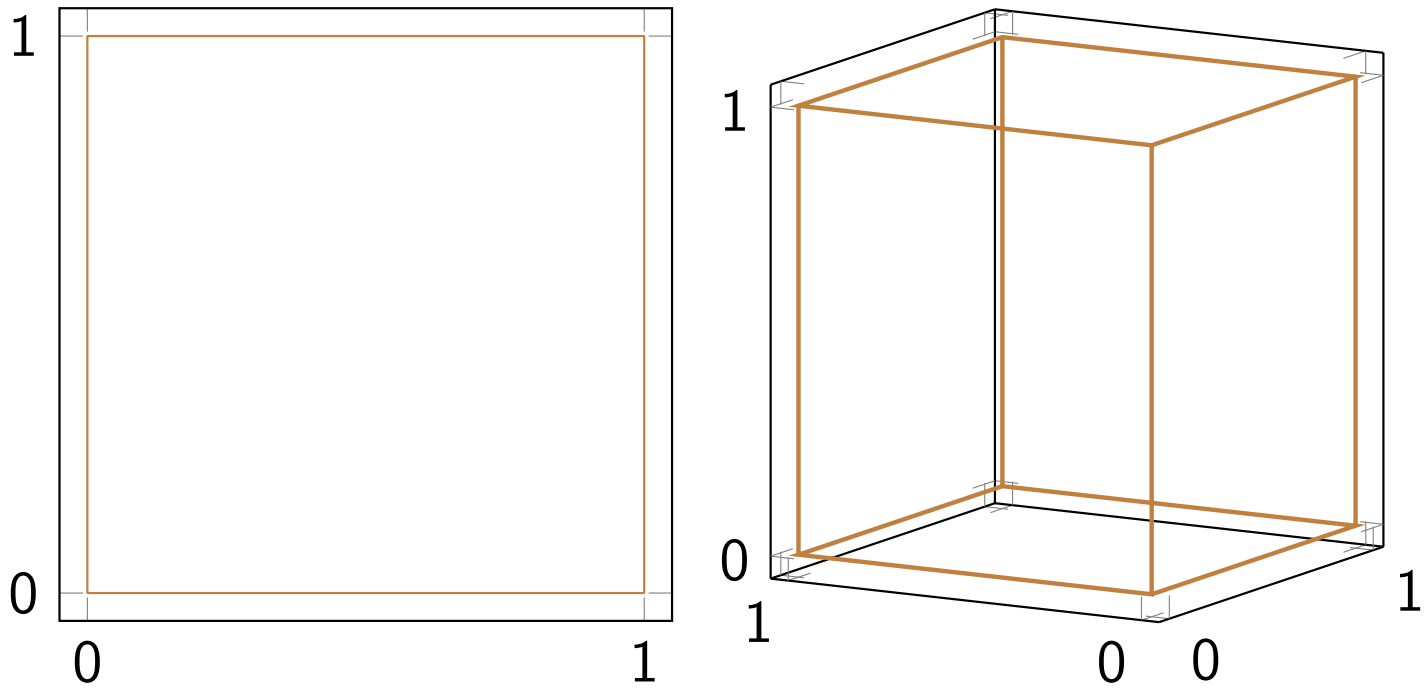


- The feasible region defined by linear constraints may be covered by simplices.
- Symmetries of objective functions may be exploited using linear inequality constraints.
- Paulavičius, Žilinskas. Optimization Letters, 10(2), 237-246, 2016, doi:10.1007/s11590-014-0772-4

DISIMPL-V and DISIMPL-C algorithms

DISIMPL - stands for DIviding SIMPLices.

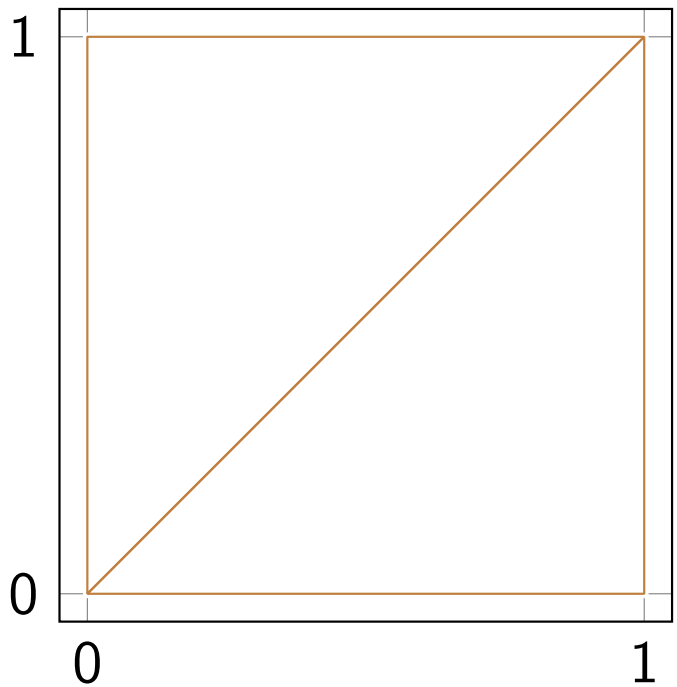
- **Initial step:** *Scale* $\mathbb{D} \to \overline{\mathbb{D}} = \{\mathbf{x} \in \mathbb{R}^n : 0 \leq x_i \leq 1,\ i = 1, \ldots, n\}$

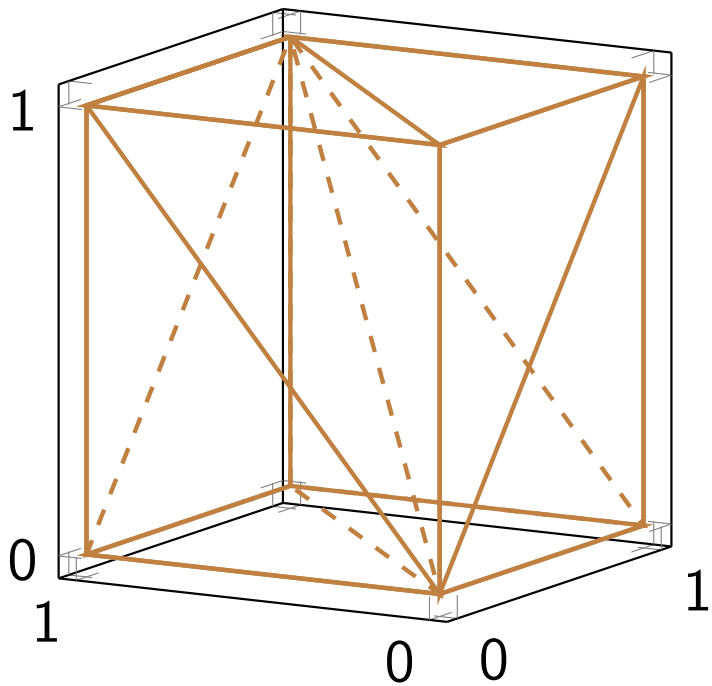


DISIMPL-V and DISIMPL-C algorithms

DISIMPL - stands for DIviding SIMPLices.

- **Initial step:** *Scale* $\mathbb{D} \to \overline{\mathbb{D}} = \{\mathbf{x} \in \mathbb{R}^n : 0 \leq x_i \leq 1,\ i = 1, \ldots, n\}$
- **Covering step:** *Cover* $\overline{\mathbb{D}}$ by $n!$ simplices of equal hyper-volume

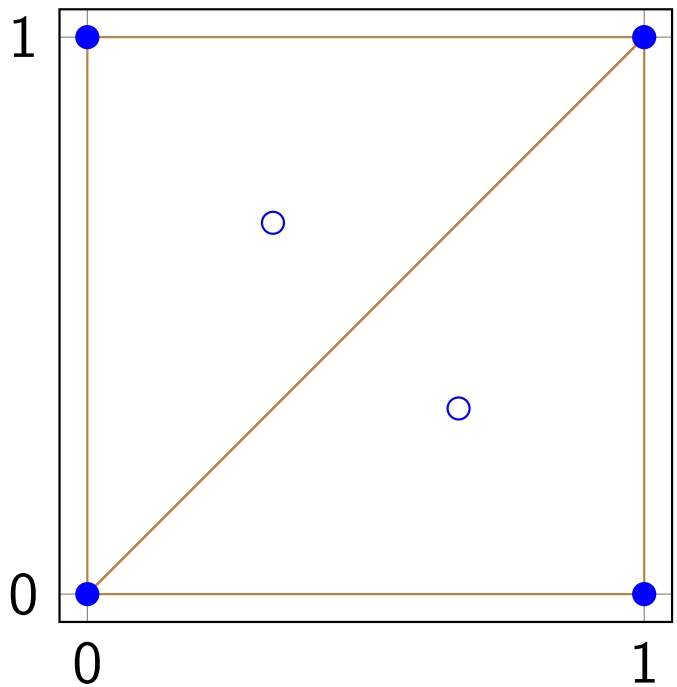


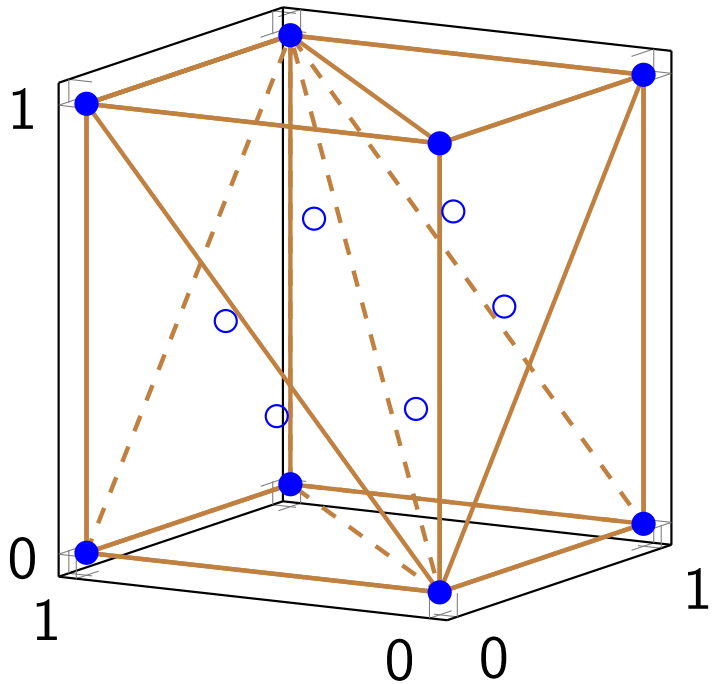


DISIMPL-V and DISIMPL-C algorithms

DISIMPL - stands for DIviding SIMPLices.

- **Initial step:** *Scale* $\mathbb{D} \to \overline{\mathbb{D}} = \{\mathbf{x} \in \mathbb{R}^n : 0 \leq x_i \leq 1,\ i = 1, \ldots, n\}$
- **Covering step:** *Cover* $\overline{\mathbb{D}}$ by $n!$ simplices and *evaluate* at all the vertices [DISIMPL-V] (barycenters [DISIMPL-C]).

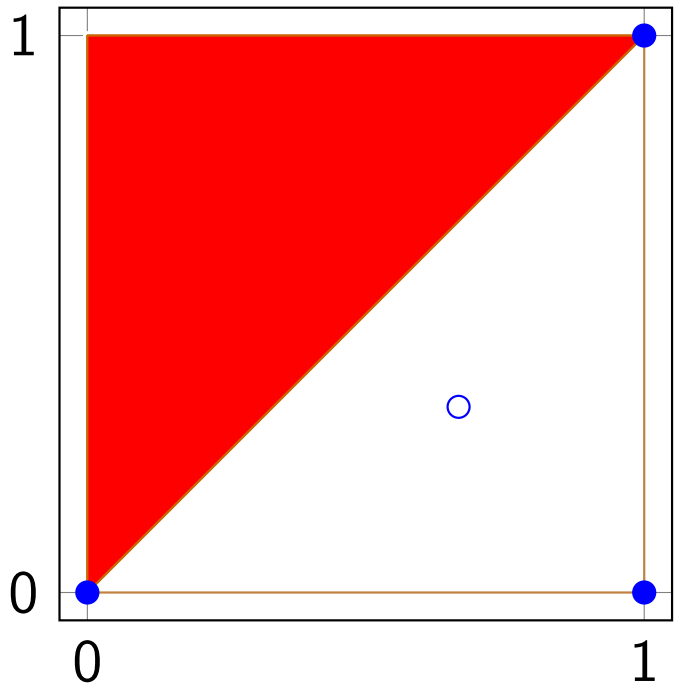


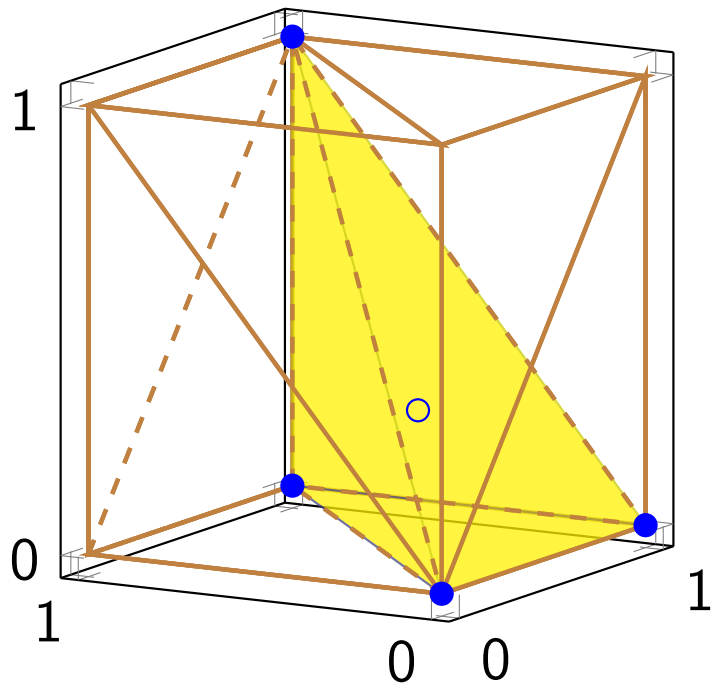


DISIMPL-V and DISIMPL-C algorithms

DISIMPL - stands for DIviding SIMPLices.

- **Initial step:** *Scale* $\mathbb{D} \to \overline{\mathbb{D}} = \{\mathbf{x} \in \mathbb{R}^n : 0 \leq x_i \leq 1,\ i = 1, \ldots, n\}$
- **Covering step:** *Cover* $\overline{\mathbb{D}}$ by $n!$ simplices and *evaluate* at all the vertices [DISIMPL-V] (barycenters [DISIMPL-C]). For *symmetric problems* there may be one initial simplex.





DISIMPL-V and DISIMPL-C algorithms

DISIMPL - stands for DIviding SIMPLices.

- **Initial step:** *Scale* $\mathbb{D} \to \overline{\mathbb{D}} = \{\mathbf{x} \in \mathbb{R}^n : 0 \leq x_i \leq 1,\ i = 1, \ldots, n\}$
- **Covering step:** *Cover* $\overline{\mathbb{D}}$ by $n!$ simplices and *evaluate* at all the vertices [DISIMPL-V] (barycenters [DISIMPL-C]). For *symmetric problems* there may be one initial simplex.
- **Step 1:** *Identify* and *select potentially optimal simplices*

Potentially optimal simplices (POS)[DISIMPL-V]

Let $\mathbb{S}$ be a set of all simplices created by DISIMPL-V after k iterations, $\varepsilon > 0$ and f_{min} current best function value. A $j \in \mathbb{S}$ is POS if $\exists \tilde{L} > 0$ such that

$$\begin{aligned} \min_{\mathbf{v} \in \mathbb{V}(j)} f(\mathbf{v}) - \tilde{L}\delta_j &\leq \min_{\mathbf{v} \in \mathbb{V}(i)} f(\mathbf{v}) - \tilde{L}\delta_i, \quad \forall i \in \mathbb{S} \\ \min_{\mathbf{v} \in \mathbb{V}(j)} f(\mathbf{v}) - \tilde{L}\delta_j &\leq f_{min} - \varepsilon |f_{min}|. \end{aligned}$$

Here δ_j is the length of its longest edge and $\mathbb{V}(j)$ is the vertex

DISIMPL-V and DISIMPL-C algorithms

DISIMPL - stands for DIviding SIMPLices.

- **Initial step:** *Scale* $\mathbb{D} \to \overline{\mathbb{D}} = \{\mathbf{x} \in \mathbb{R}^n : 0 \leq x_i \leq 1,\ i = 1, \ldots, n\}$
- **Covering step:** *Cover* $\overline{\mathbb{D}}$ by $n!$ simplices and *evaluate* at all the vertices [DISIMPL-V] (barycenters [DISIMPL-C]). For *symmetric problems* there may be one initial simplex.
- **Step 1:** *Identify* and *select potentially optimal simplices*

Potentially optimal simplices (POS) [DISIMPL-C]

Let $\mathbb{S}$ be a set of all simplices created by DISIMPL-C after k iterations, $\varepsilon > 0$ and f_{min} current best function value. A $j \in \mathbb{S}$ is POS if $\exists \tilde{L} > 0$ such that

$$\begin{aligned} f(\mathbf{c}_j) - \tilde{L}\delta_j &\leq f(\mathbf{c}_i) - \tilde{L}\delta_i, \quad \forall i \in \mathbb{S} \\ f(\mathbf{c}_j) - \tilde{L}\delta_j &\leq f_{min} - \varepsilon |f_{min}|. \end{aligned}$$

Here $\mathbf{c}_j$ is the geometric center (centroid) of simplex j and a measure δ_j is the maximal distance from $\mathbf{c}_j$ to its vertices.

DISIMPL-V and DISIMPL-C algorithms

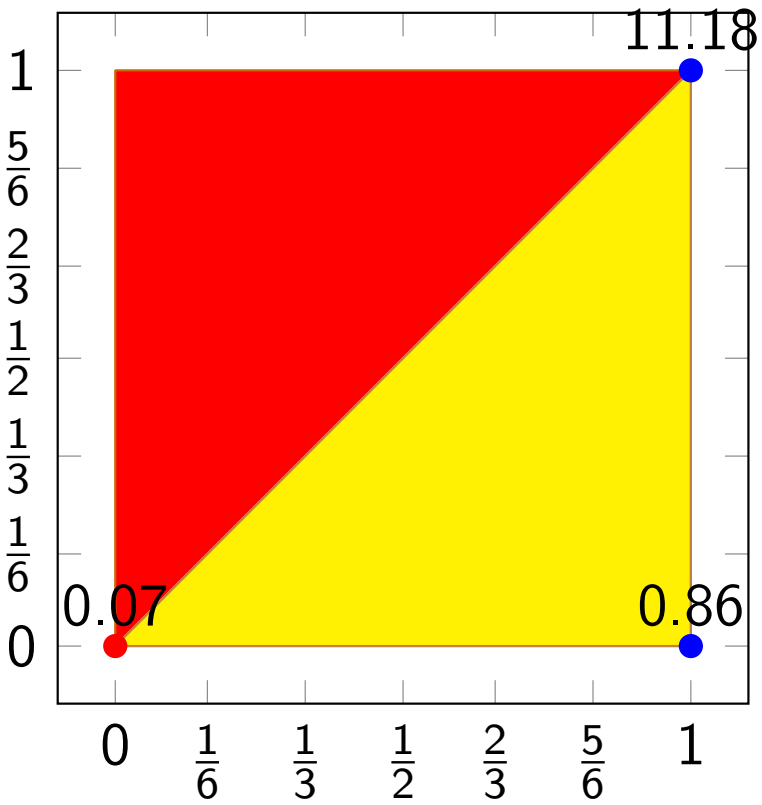
DISIMPL - stands for DIviding SIMPLices.

- **Initial step:** *Scale* $\mathbb{D} \to \overline{\mathbb{D}} = \{\mathbf{x} \in \mathbb{R}^n : 0 \leq x_i \leq 1,\ i = 1, \ldots, n\}$
- **Covering step:** *Cover* $\overline{\mathbb{D}}$ by $n!$ simplices and *evaluate* at all the vertices [DISIMPL-V] (barycenters [DISIMPL-C]). For *symmetric problems* there may be one initial simplex.
- **Step 1:** *Identify* and *select potentially optimal simplices*
- **Step 2:** *Sample* the function and *divide* into two by a hyper-plane passing through the middle point of the longest edge and the vertices which do not belong to the longest edge (DISIMPL-C) or into three simplices dividing the longest edge too.
- Repeat **Step 1** and **Step 2** until satisfied some *stopping criteria*.

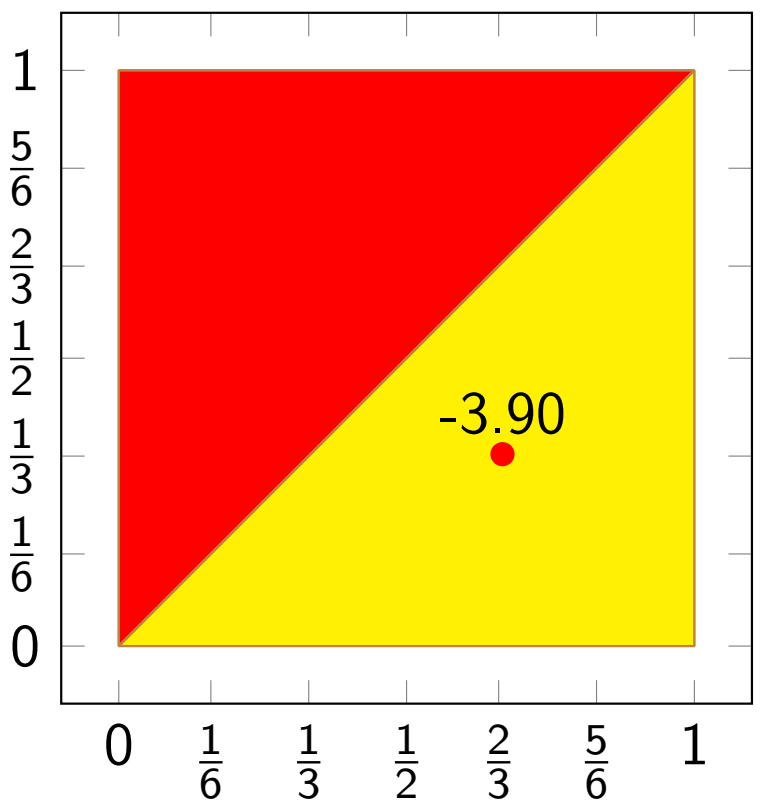
Visualization of DISIMPL-V and DISIMPL-C

- DISIMPL-V **Result:** $fmin = 0.07$, $iter = 0$, $f.eval = 3$
- DISIMPL-C **Result:** $fmin = -3.90$, $iter = 0$, $f.eval = 1$

DISIMPL-V



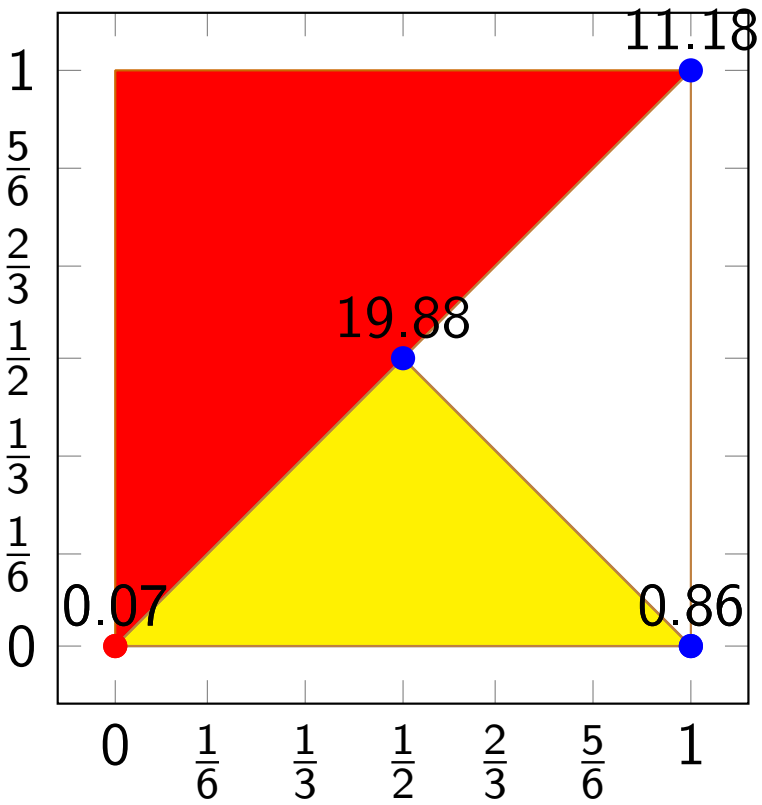
DISIMPL-C



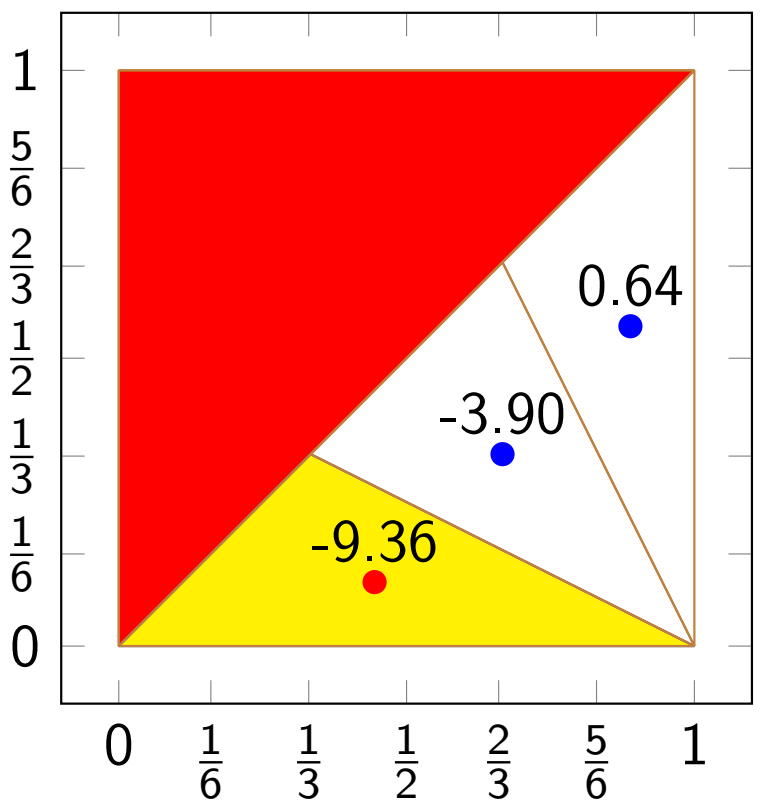
Visualization of DISIMPL-V and DISIMPL-C

- DISIMPL-V **Result:** $fmin = 0.07$, $iter = 1$, $f.eval = 4$
- DISIMPL-C **Result:** $fmin = -9.36$, $iter = 1$, $f.eval = 3$

DISIMPL-V



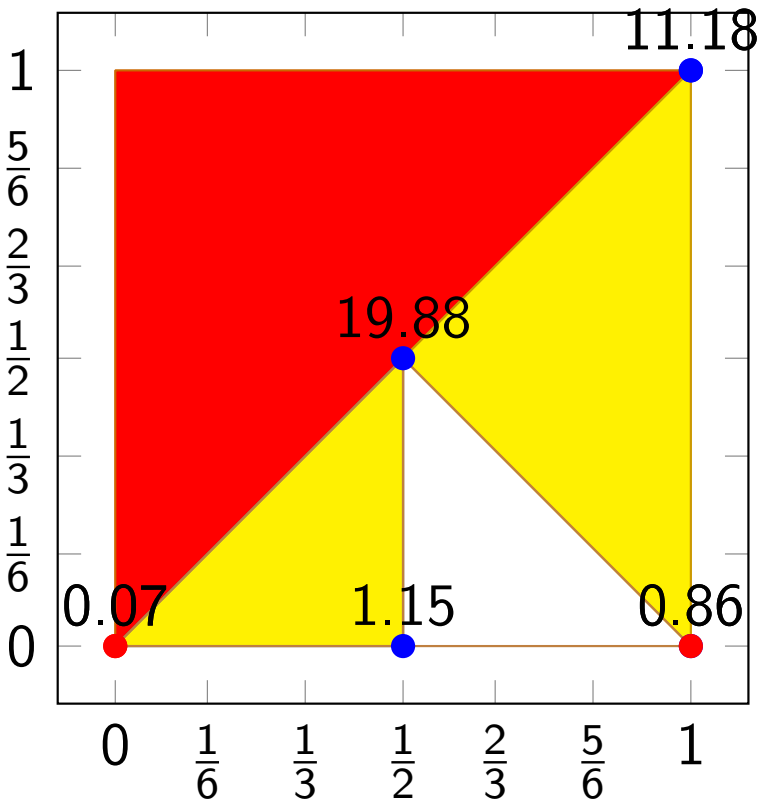
DISIMPL-C



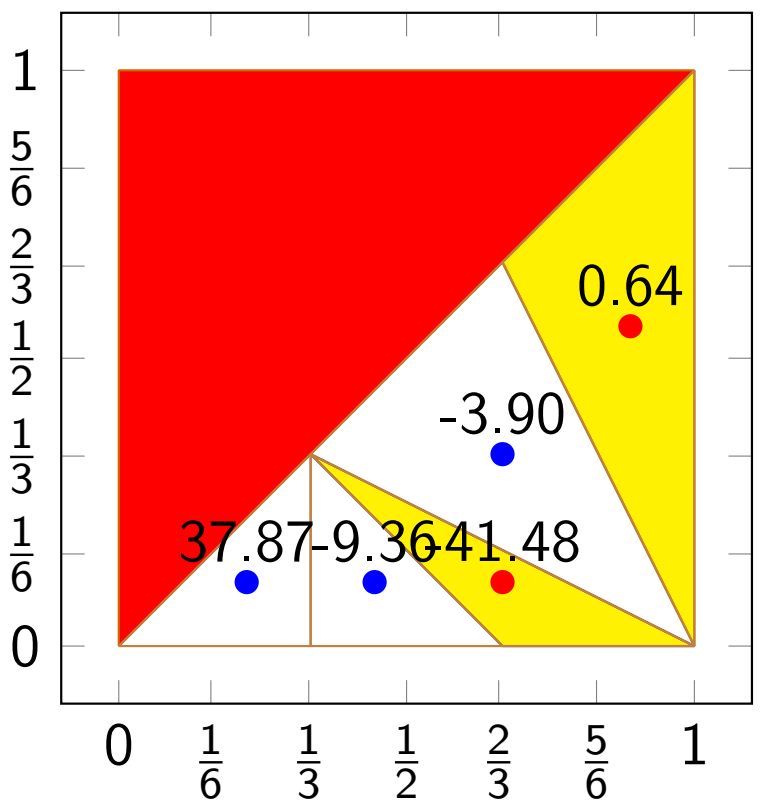
Visualization of DISIMPL-V and DISIMPL-C

- DISIMPL-V **Result:** $fmin = 0.07$, $iter = 2$, $f.eval = 5$
- DISIMPL-C **Result:** $fmin = -41.48$, $iter = 2$, $f.eval = 5$

DISIMPL-V



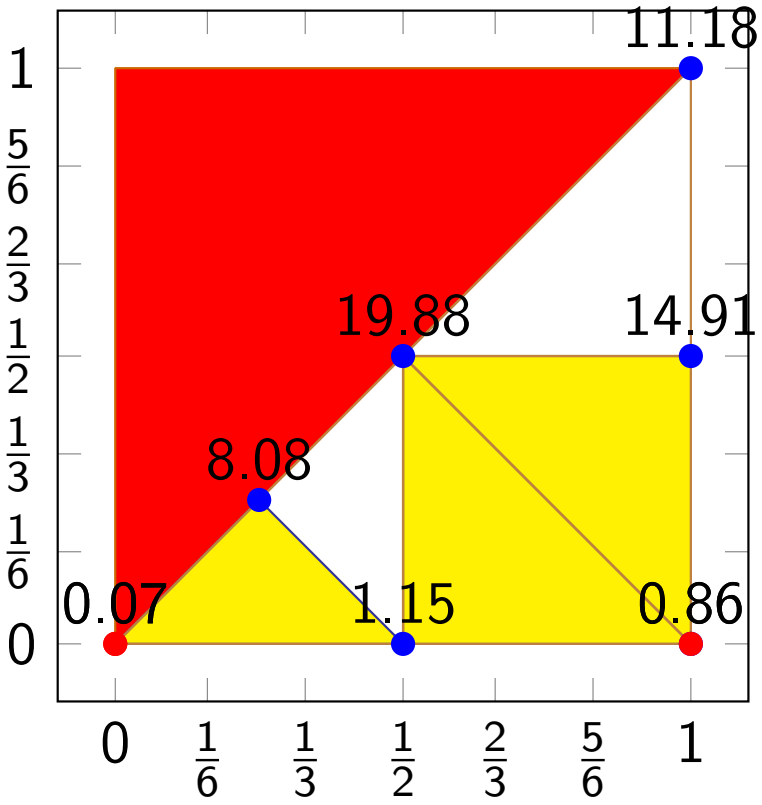
DISIMPL-C



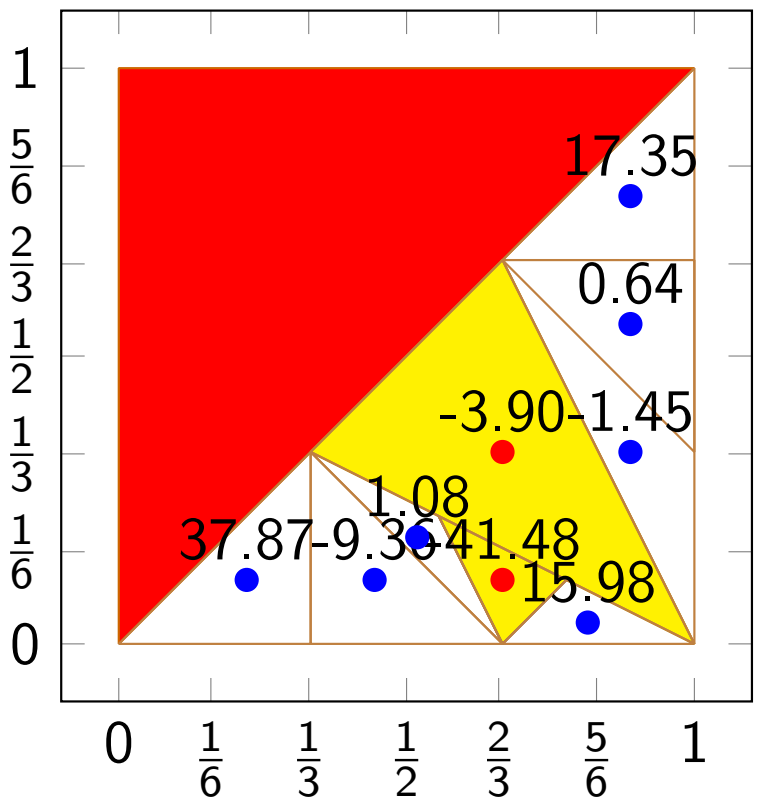
Visualization of DISIMPL-V and DISIMPL-C

- DISIMPL-V **Result:** $fmin = 0.07$, $iter = 3$, $f.eval = 7$
- DISIMPL-C **Result:** $fmin = -16.20$, $iter = 3$, $f.eval = 9$

DISIMPL-V



DISIMPL-C



Coping with symmetries of objective function

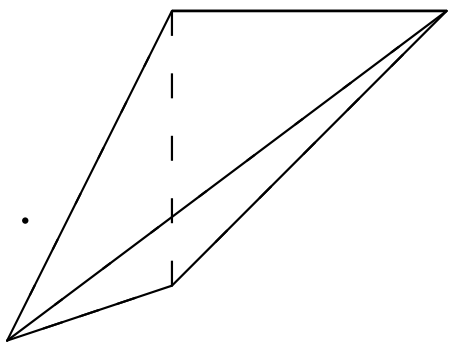
- If interchange of the variables x_i and x_j does not change the value of the objective function, it is symmetric over the hyper-plane $x_i = x_j$.
- It is possible to avoid such symmetries by setting linear constraints on such variables: $x_i \leq x_j$.
- The resulting constrained search space may be vertex triangulated.
- The search space and the numbers of local and global minimizers may be reduced by avoiding symmetries.

Example of coping with symmetries of objective function

$$f(x) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos(x_i) + 1, D = [-500, 700]^n$$

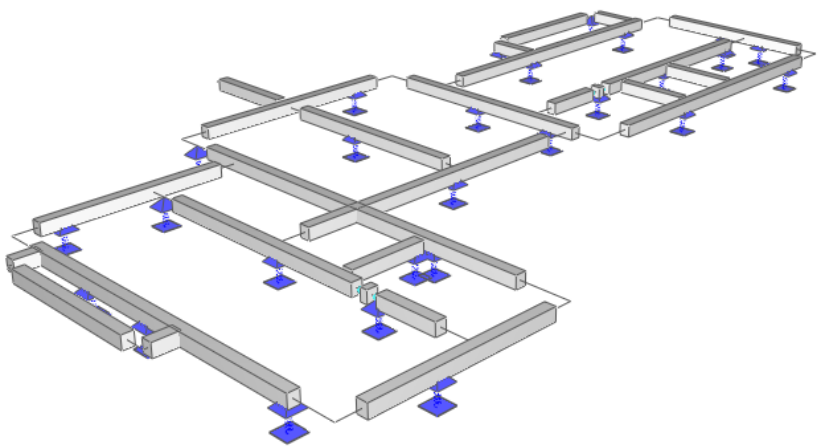
- The objective function is symmetric over hyper-planes $x_i = x_j$.
- Constraints for avoiding symmetries: $x_1 \leq x_2 \leq \ldots \leq x_n$.
- The resulting simplicial search space:

$$D = \left\{ \begin{array}{ccccc} -500 & -500 & \ldots & -500 & -500 \\ -500 & -500 & \ldots & -500 & 700 \\ \vdots & & \ddots & & \vdots \\ -500 & 700 & \ldots & 700 & 700 \\ 700 & 700 & \ldots & 700 & 700 \end{array} \right\}.$$



Example of coping with symmetries: optimization of gillage-type foundations

- Grillage foundation consists of separate beams, which are supported by piles or reside on other beams.
- The piles should be positioned minimizing the largest difference between reactive forces and limit magnitudes of reactions.



Optimization of gillage-type foundations: formulation

- Black-box problem. The values of objective function are evaluated by an independent package which models reactive forces in the grillage using finite element method.
- Gradient may be estimated using sensitivity analysis implemented in the modelling package.
- The number of piles is n.
- The position of a pile is given by a real number, which is mapped to a two-dimensional position by the modelling package. Possible values are from zero to the sum of length of all beams l.
- Feasible region is $[0, l]^n$.
- Characteristic of all piles are equal, their interchange does not change the value of objective function.

Optimization of gillage-type foundations: simplicial search space I

- Let us constrain the problem avoiding symmetries of the objective function: $x_1 \leq x_2 \leq \ldots \leq x_n$.
- Simplicial search space:

$$D = \left\{ \begin{array}{ccccc} 0 & 0 & \ldots & 0 & 0 \\ 0 & 0 & \ldots & 0 & l \\ \vdots & & \ddots & & \vdots \\ 0 & l & \ldots & l & l \\ l & l & \ldots & l & l \end{array} \right\}.$$

- Search space and the numbers of local and global minimizers are reduced $n!$ times with respect to the original feasible region.

Optimization of gillage-type foundations: simplicial search space II

- The distance between adjacent piles cannot be too small due to the specific capacities of a pile driver.
- Let us constrain the problem avoiding symmetries and distances between two piles smaller than δ: $x_1 \leq x_2 - \delta, \ldots, x_{n-1} \leq x_n - \delta$.
- Simplicial search space:

$$D = \left\{ \begin{array}{ccccc} 0 & \delta & \ldots & (n-2)\delta & (n-1)\delta \\ 0 & \delta & \ldots & (n-2)\delta & l \\ \vdots & & \ddots & & \vdots \\ 0 & l-(n-2)\delta & \ldots & l-\delta & l \\ l-(n-1)\delta & l-(n-2)\delta & \ldots & l-\delta & l \end{array} \right\}.$$

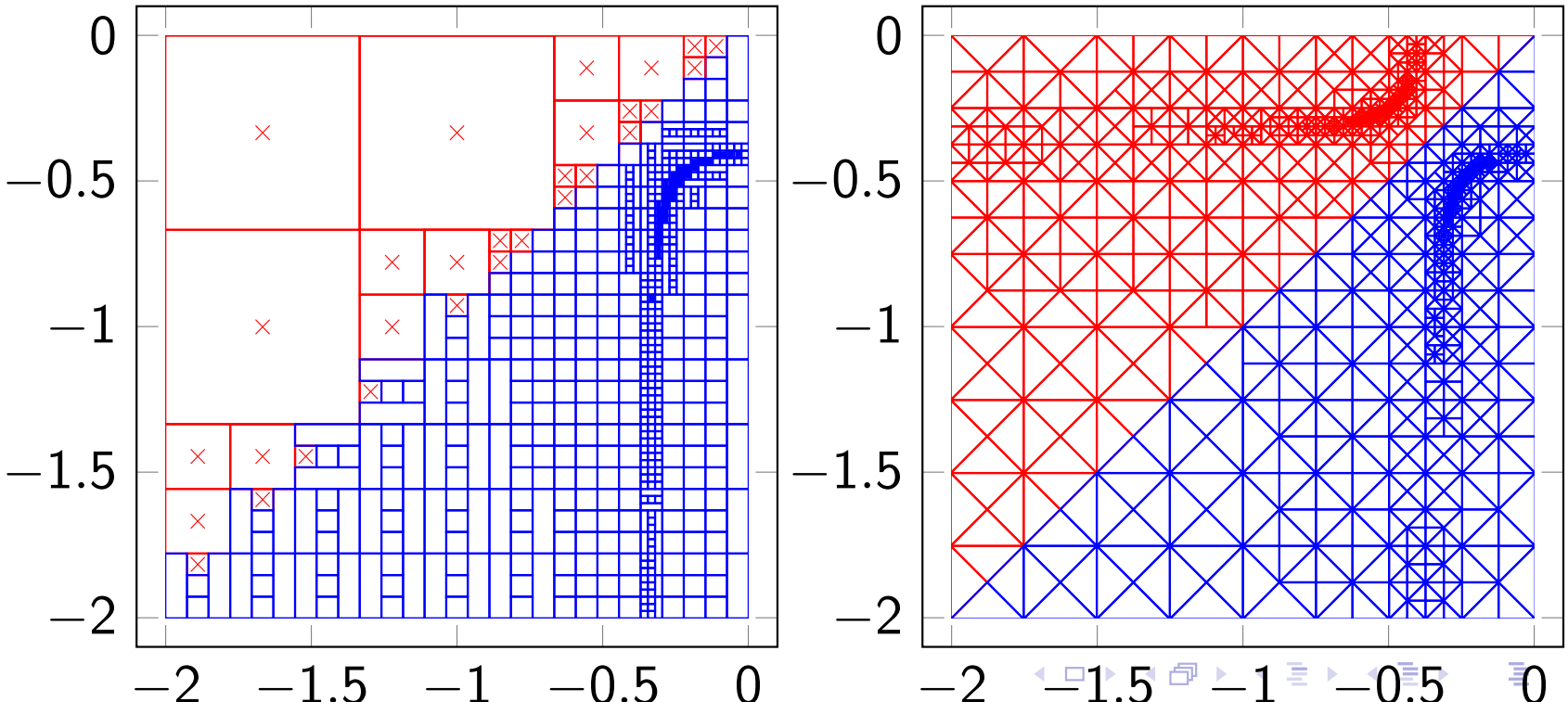
Least squares nonlinear regression

- The optimization problem in nonlinear LSR:

$$\min_{\mathbf{x}\in\mathbb{D}} \sum_{k=1}^{m} (y_i - \varphi(\mathbf{x}, \mathbf{z}_i))^2 = \min_{\mathbf{x}\in\mathbb{D}} f(\mathbf{x}),$$

where the measurements y_i at points $\mathbf{z}_i = (z_{1i}, z_{2i}, ..., z_{pi})$ should be tuned by the nonlinear function $\varphi(\mathbf{x}, \cdot)$, e.g.

$$\varphi(\mathbf{x}, z) = x_1 + x_2 \exp(-x_4 z) + x_3 \exp(-x_5 z).$$



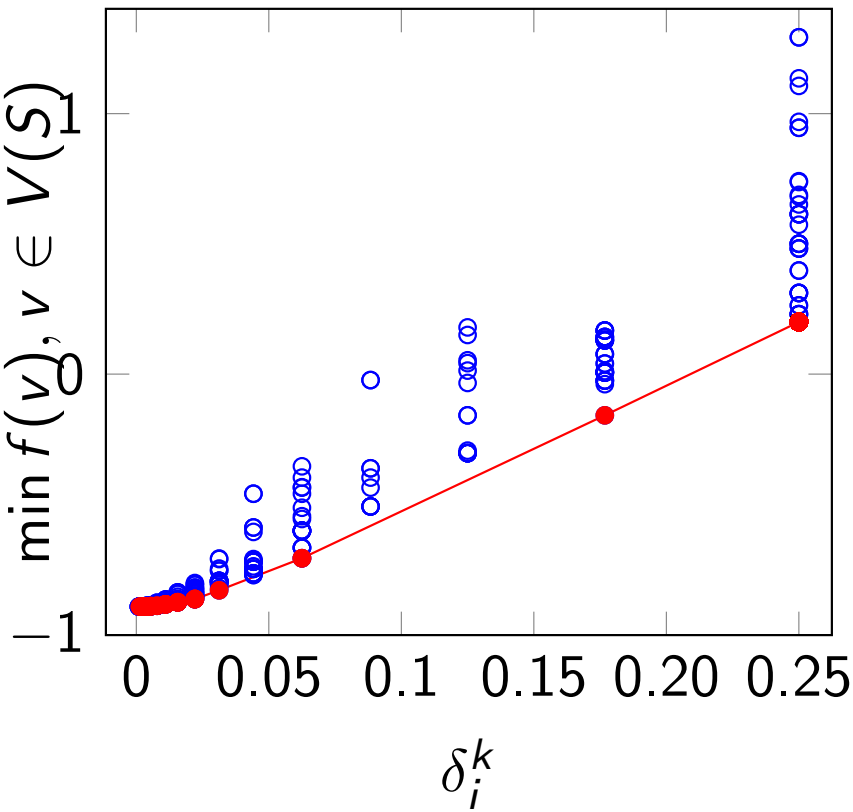
Experimental investigation

#	DISIMPL-V		DISIMPL-C		SymDIRECT	
	$pe < 1.0$	$pe < 0.01$	$pe < 1.0$	$pe < 0.01$	$pe < 1.0$	$pe < 0.01$
1.	156 (1)	**216 (2)**	172 (0)	554 (0)	**105 (2)**	409 (1)
2.	**141 (2)**	**192 (2)**	252 (0)	430 (0)	199 (1)	289 (1)
3.	935 (1)	1914 (1)	2456 (0)	3534 (0)	**705 (2)**	**955 (2)**
4.	939 (1)	1894 (1)	**686 (2)**	**1718 (2)**	1577 (0)	2763 (0)
5.	288 (1)	1171 (0)	**202 (2)**	1162 (1)	425 (0)	**1025 (2)**
6.	333 (0)	977 (1)	274 (1)	1006 (0)	**201 (2)**	**815 (2)**
7.	612 (0)	**2439 (2)**	258 (1)	5034 (0)	**157 (2)**	2701 (1)
8.	**224 (2)**	**3879 (2)**	282 (1)	4050 (1)	407 (0)	5927 (0)
9.	612 (0)	**2439 (2)**	258 (1)	4974 (0)	**157 (2)**	2701 (1)
10.	**224 (2)**	**3879 (2)**	282 (1)	4050 (1)	407 (0)	5897 (0)
11.	612 (0)	**2439 (2)**	258 (1)	5034 (0)	**157 (2)**	2839 (1)
12.	**224 (2)**	4041 (1)	282 (1)	**4026 (2)**	407 (0)	5633 (0)
TP	(12)	**(18)**	(11)	(7)	**(13)**	(11)

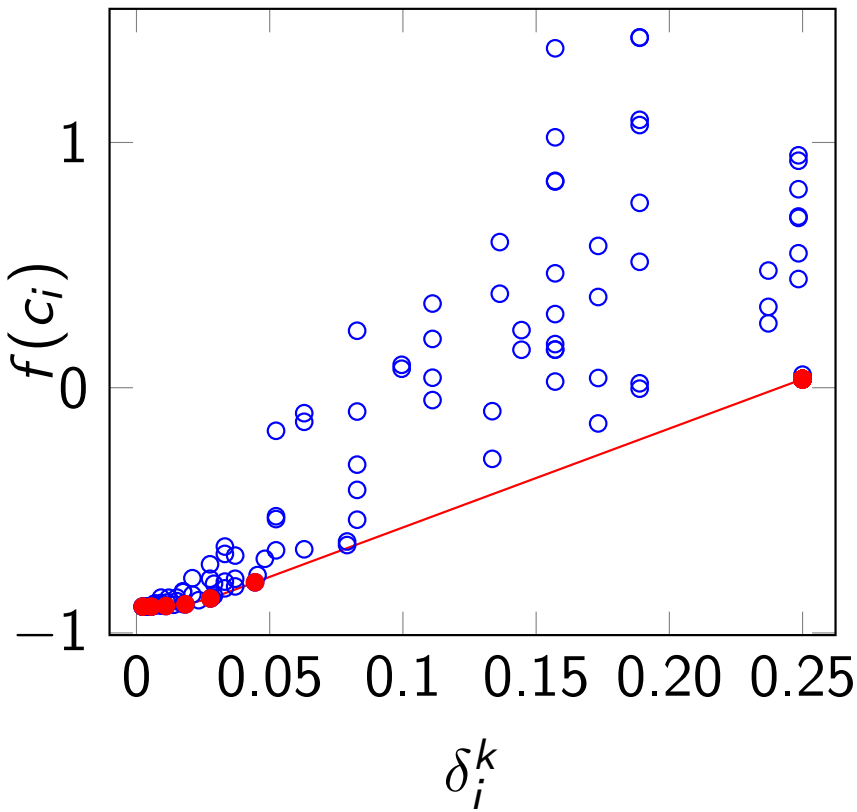
Motivation of globally-biased DIRECT-type algorithms

- It is well known that the DIRECT algorithm quickly gets close to the optimum but can take much time to achieve a high degree of accuracy.

DISIMPL-V



DISIMPL-C



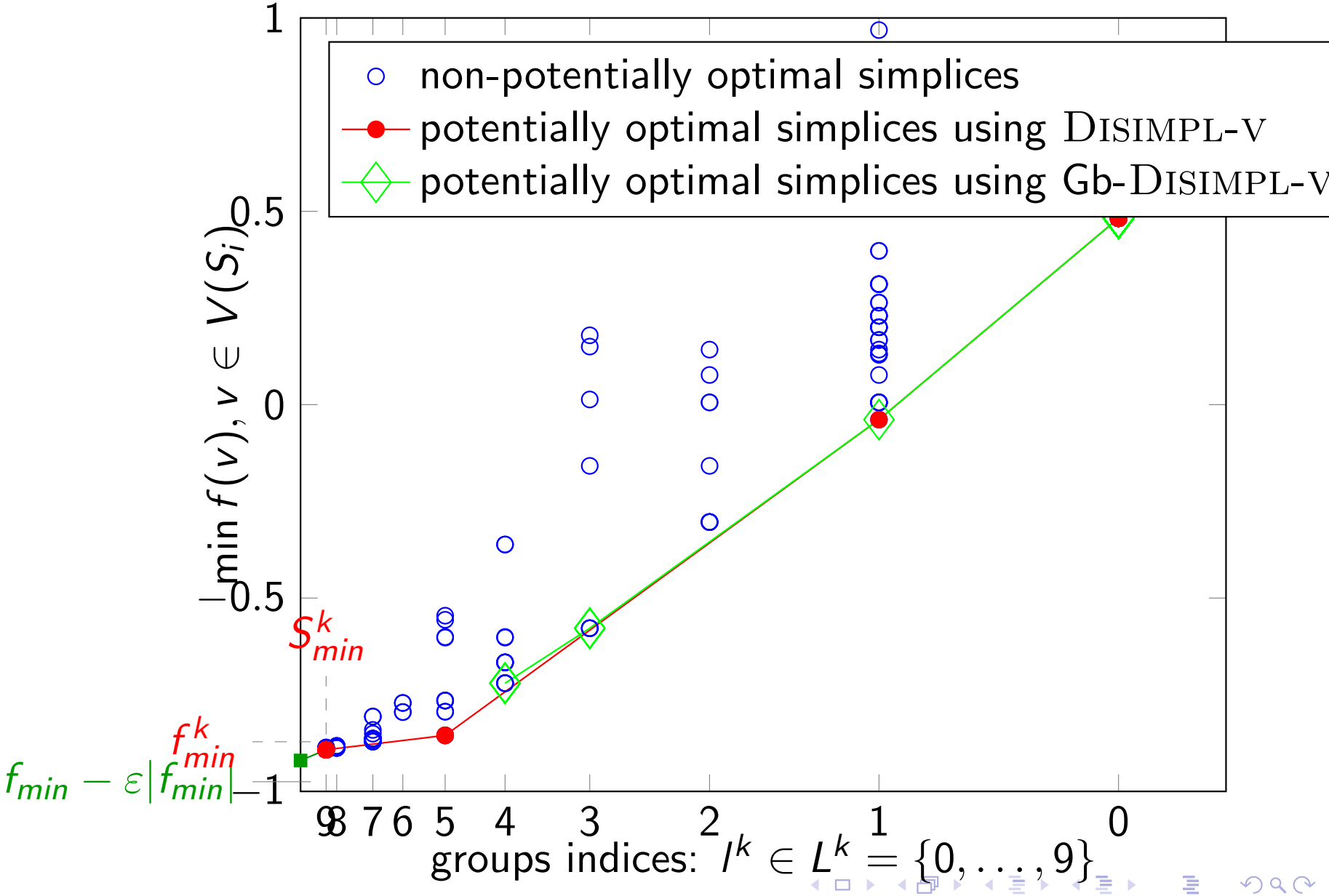
non-potentially optimal simplices — potentially optimal simplices

The globally-biased DISIMPL (Gb-DISIMPL)

- Gb-DISIMPL consists of the following two-phases: a *usual phase* (as in the original DISIMPL method) and a *global* one.
- The usual phase is performed until a sufficient number of subdivisions of simplices near the current best point is performed.
- Once many subdivisions around the current best point have been executed, its neighborhood contains only small simplices and all larger ones are located far from it.
- Thus, the two-phase approach forces the Gb-DISIMPL algorithm to explore larger simplices and to return to the usual phase only when an improved minimal function value is obtained.
- Each of these phases can consist of several iterations.
- Paulavičius, Sergeev, Kvasov, Žilinskas. JOGO, 59(2-3), 545-567, 2014, doi:10.1007/s10898-014-0180-4

Journal of Global Optimization Best Paper Award 2014.

Visualization of Gb-DISIMPL-V



Visualization of Gb-DISIMPL

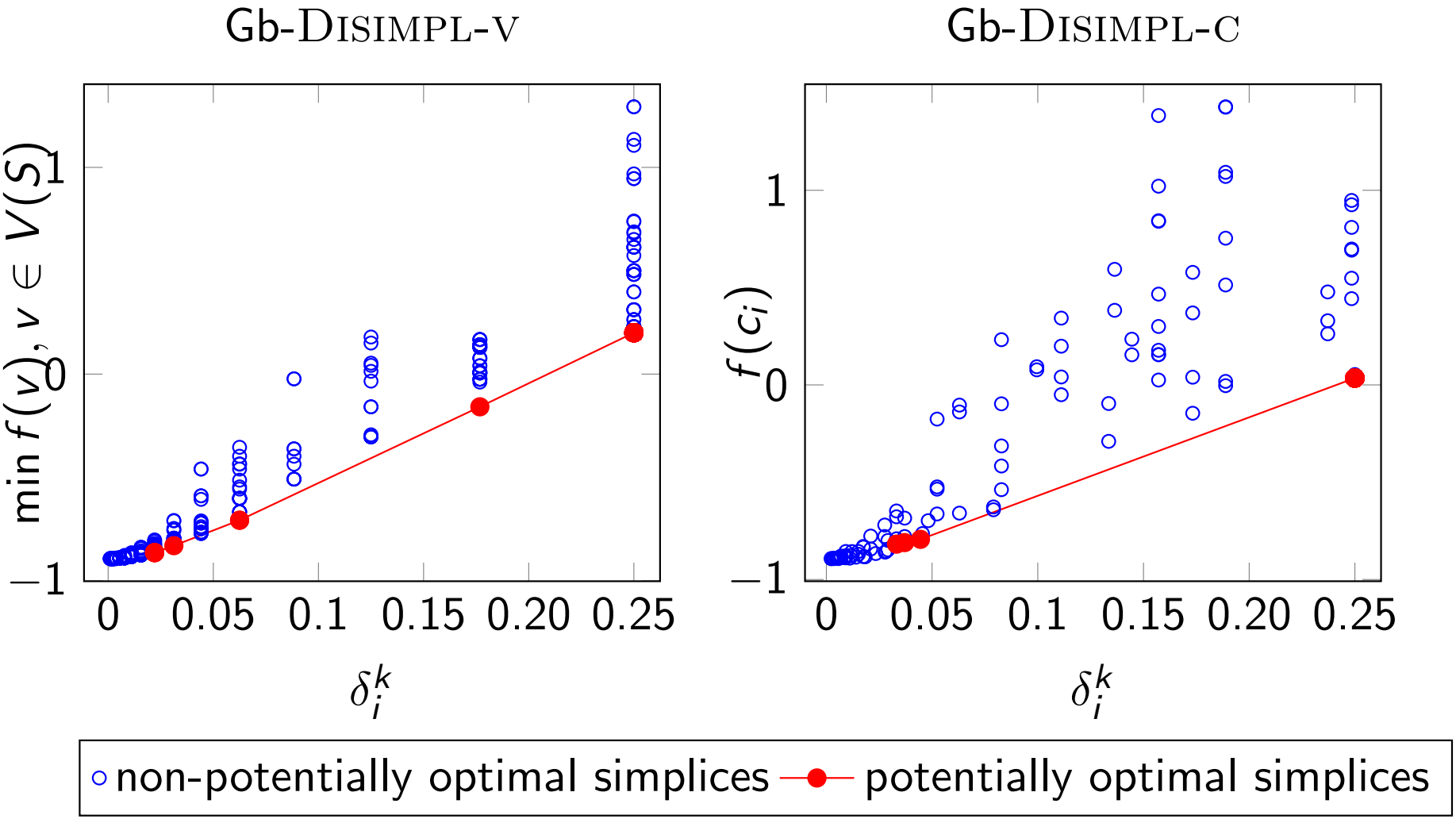


Figure: Geometric interpretation of potentially optimal simplices in $k = 23$ iteration of the Gb-DISIMPL-V and Gb-DISIMPL-C algorithms on a $n = 2$ GKLS test problem

Description of GKLS classes of test problems

The standard test problems are too simple for global optimization methods, therefore we use GKLS generator

Class (#)	Difficulty	Δ	n	m	f^*	d	r
1.	Simple	10^{-4}	2	10	-1.0	0.90	0.20
2.	Hard	10^{-4}	2	10	-1.0	0.90	0.10
3.	Simple	10^{-6}	3	10	-1.0	0.66	0.20
4.	Hard	10^{-6}	3	10	-1.0	0.90	0.20
5.	Simple	10^{-6}	4	10	-1.0	0.66	0.20
6.	Hard	10^{-6}	4	10	-1.0	0.90	0.20
7.	Simple	10^{-7}	5	10	-1.0	0.66	0.30
8.	Hard	10^{-7}	5	10	-1.0	0.66	0.20

Here: n – dimension; m – no. of local minima; f^* – global minima value; d – distance from the global minimizer to the vertex; r – radius of the attraction region.

Comparison criteria

1. *The number (maximal and average over each class) of function evaluations (f.e.)* executed by the methods until the satisfaction of the stopping rule:

Stopping Rule

The global minimizer $x^* \in D = [l, u]$ was considered to be found when an algorithm generated a trial point $x_j \in S_j$ such that

$$|x_j(i) - x^*(i)| \leq \sqrt[n]{\Delta}(u(i) - l(i)), \quad 1 \leq i \leq n, \tag{1}$$

where $0 < \Delta \leq 1$ is an accuracy coefficient. The algorithms also stopped if the maximal number of function evaluations $T_{max} = 1\,000\,000$ was reached.

2. *The number of subregions* generated until condition (1) is satisfied. This number reflects indirectly degree of qualitative examination of D during the search for a global minimizer.

Comparison on 8 classes of GKLS problems

#	DIRECT	DIRECT/	DISIMPL-C		DISIMPL-V	
			Original	Gb	Original	Gb
	Average number of function evaluations					
1	198.89	292.79	200.80	187.72	192.93	**173.09**
2	1063.78	1267.07	1189.60	701.28	1003.56	**535.95**
3	1117.70	1785.73	2614.02	2217.08	1061.83	**745.07**
4	>42322.65	4858.93	7158.76	4452.86	2598.91	**1419.11**
5	>47282.89	18983.55	>46500.90	19593.56	10617.98	**5051.86**
6	>95708.25	68754.02	>78794.37	>53716.76	33985.16	**12796.86**
7	>16057.46	16758.44	>43428.74	>25306.30	11200.44	**8579.89**
8	>217215.58	>269064.35	>255311.84	>164754.26	64750.99	**34079.21**

Comparison on 8 classes of GKLS problems

#	DIRECT	DIRECT/	DISIMPL-C		DISIMPL-V	
			Original	Gb	Original	Gb
Number of function evaluations (50%)						
1	**111**	152	132	132	151	151
2	1062	1328	705	556	1021	**521**
3	**386**	591	1403	1438	787	721
4	1749	1967	3315	2960	2594	**1333**
5	4805	7194	6211	4717	7334	**4623**
6	16114	33147	20677	14116	29807	**11305**
7	**1660**	9246	3893	3522	7252	6734
8	55092	126304	50096	25746	42680	**25106**
Number of function evaluations (100%)						
1	1159	2318	960	938	773	**418**
2	3201	3414	6696	2624	2683	**1745**
3	12507	13309	50166	20568	4740	**2636**
4	FAIL (4)	29233	111252	33346	7354	**3483**
5	FAIL (4)	118744	FAIL (4)	238964	58764	**12960**
6	FAIL (7)	287857	FAIL (10)	FAIL (2)	118482	**42246**
7	FAIL (1)	178217	FAIL (2)	288372	48590	**32119**
8	FAIL (16)	FAIL (4)	FAIL (21)	FAIL (9)	382593	**201365**

Comparison on 8 classes of GKLS problems

#	DIRECT	DIRECT/	DISIMPL-C		DISIMPL-V	
			Original	Gb	Original	Gb
			Number of subregions (50%)			
1	111	152	132	132	243	243
2	1062	1328	705	556	1862	921
3	386	591	1403	1438	2785	2566
4	1749	1967	3315	2960	10841	5188
5	4805	7194	6211	4717	63996	42044
6	16114	33147	20677	14116	301997	108520
7	1660	9246	3893	3522	140013	171215
8	55092	126304	50096	25746	1238077	728281
			Number of subregions (100%)			
1	1159	2318	960	938	1419	735
2	3201	3414	6696	2624	5085	3281
3	12507	13309	50166	20568	20973	11548
4	>1000000	29233	111252	33346	33087	15075
5	>1000000	118744	>1000000	238964	649395	128534
6	>1000000	287857	>1000000	>1000000	1335106	444923
7	>1000000	178217	>1000000	288372	1465960	1288262
8	>1000000	>1000000	>1000000	>1000000	11423212	6719537

Operating characteristics for the algorithms

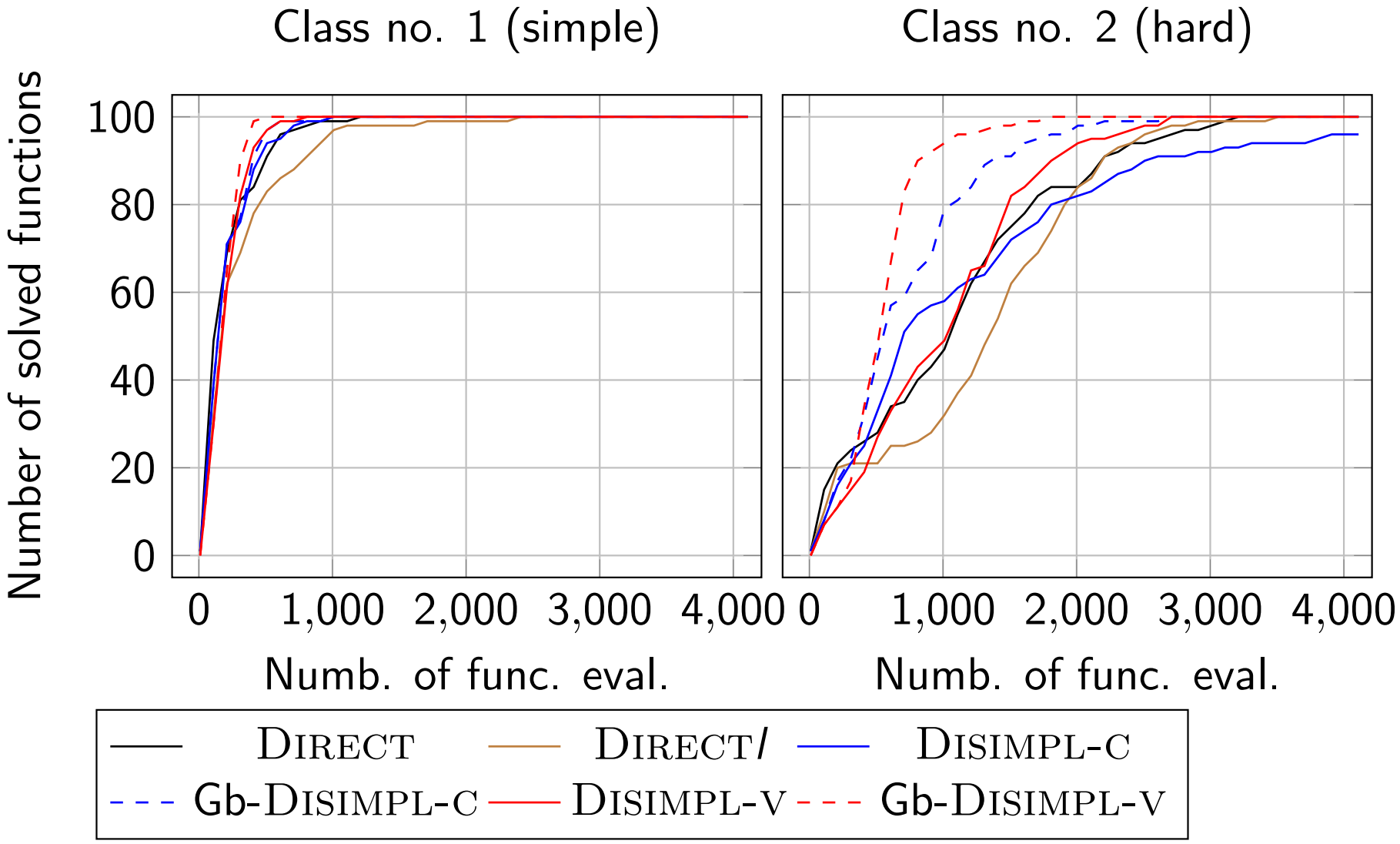


Figure: Operating characteristics for the $n = 2$ test problems

Operating characteristics for the algorithms

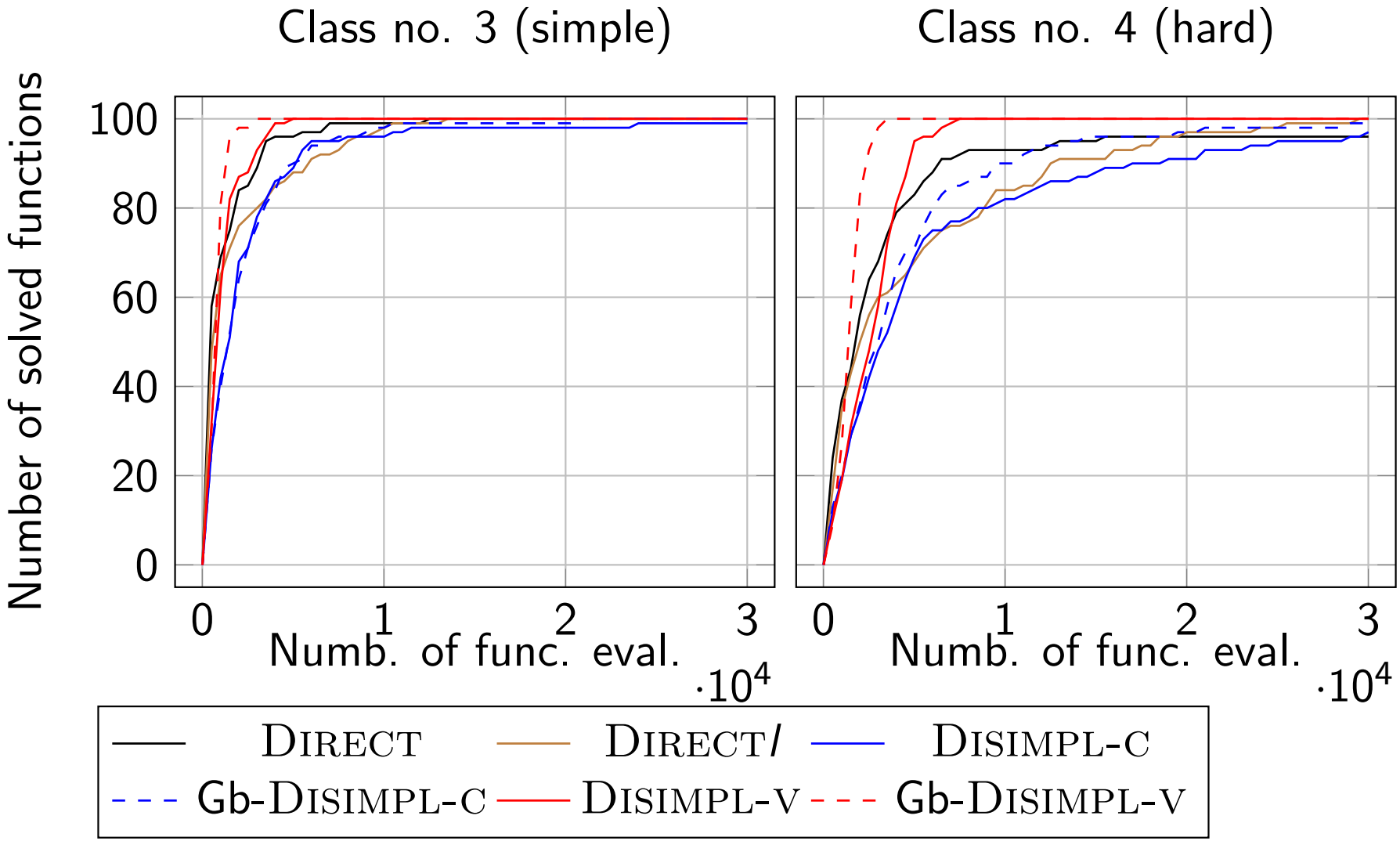


Figure: Operating characteristics for the $n = 3$ test problems

Operating characteristics for the algorithms

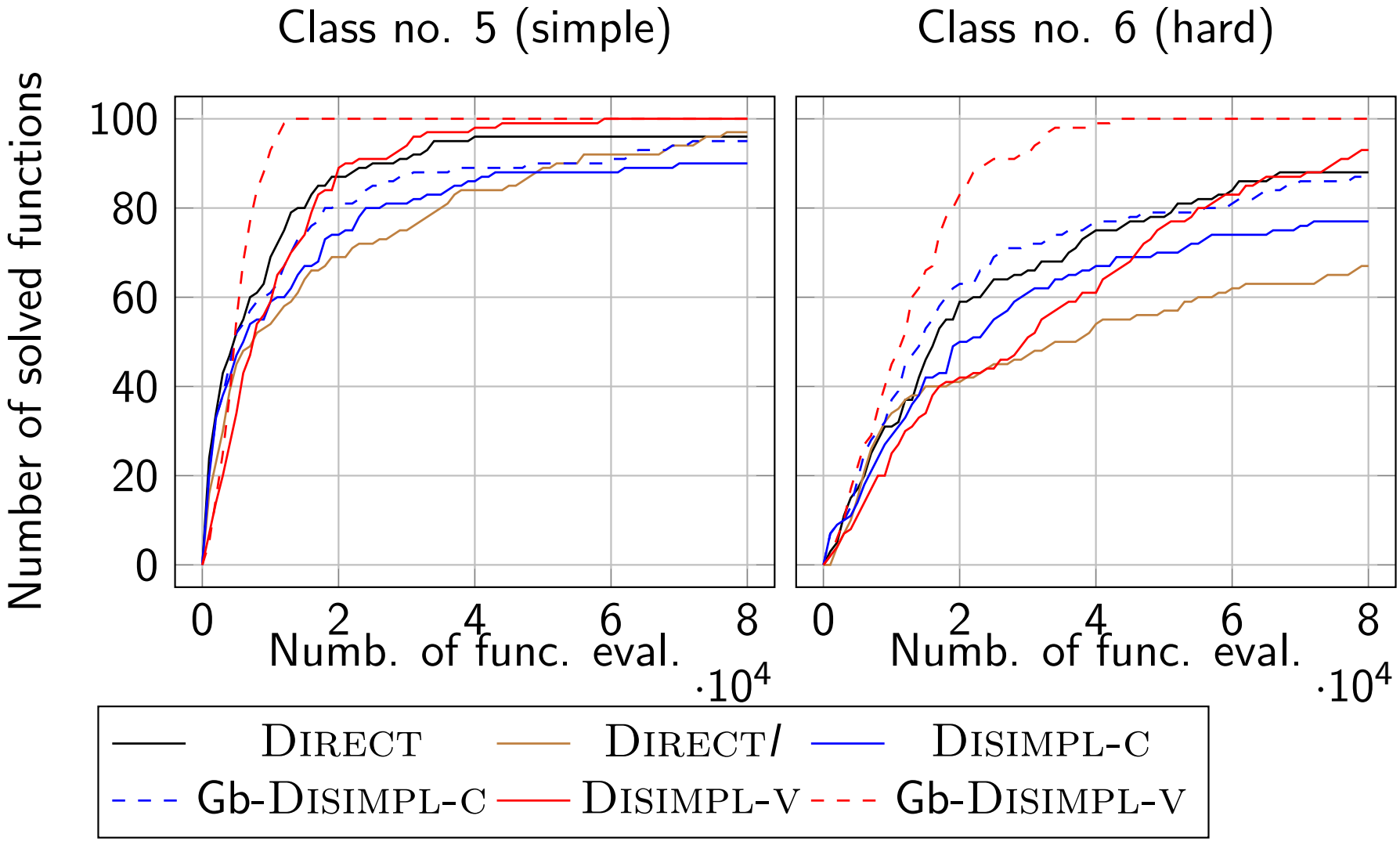


Figure: Operating characteristics for the $n = 4$ test problems

Operating characteristics for the algorithms

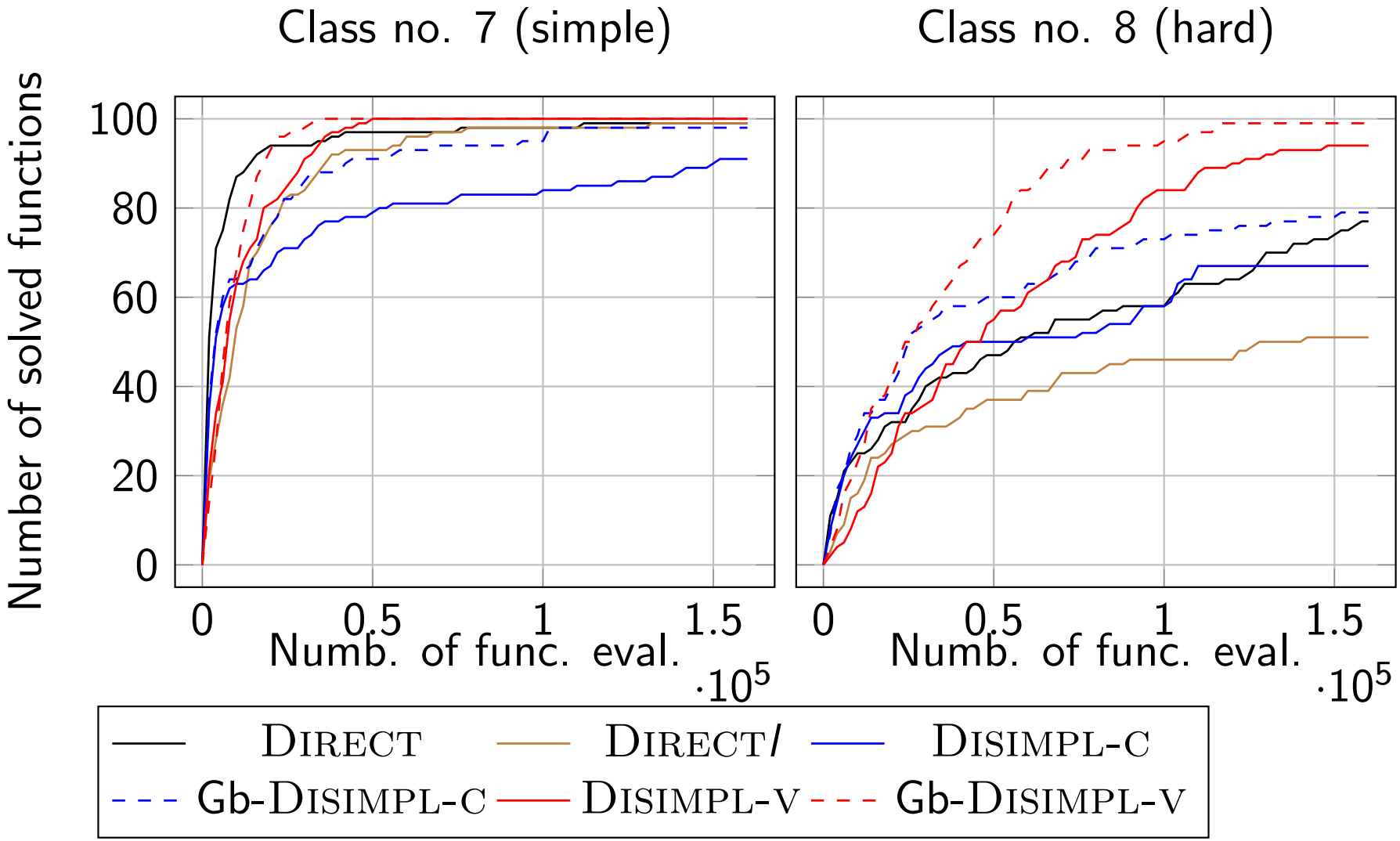


Figure: Operating characteristics for the $n = 5$ test problems

Thank you for your attention