The mesh adaptive direct search algorithm for granular and discrete variables

Sébastien Le Digabel, Charles Audet, Christophe Tribes

ISMP, 2018–07–02

Presentation outline

Blackbox optimization

Motivating example

The MADS algorithm

Computational experiments

Discussion

Blackbox optimization (BBO) problems

- Optimization problem:

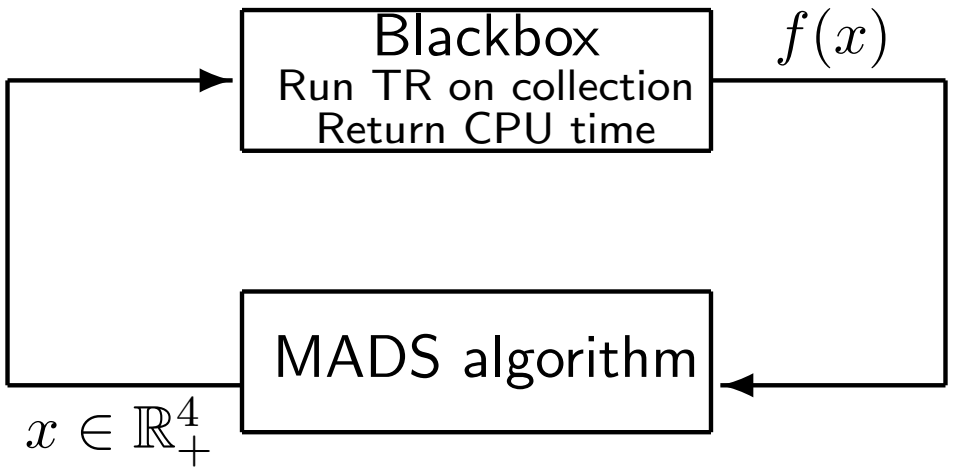
$$\min_{x \in \Omega} \quad f(x)$$

- Evaluations of f (the objective function) and of the functions defining Ω are usually the result of a computer code (a blackbox).
- Variables are typically continuous, but in this work, some of them are discrete – integers or *granular* variables.

Example: Trust-region parameter tuning (1/2)

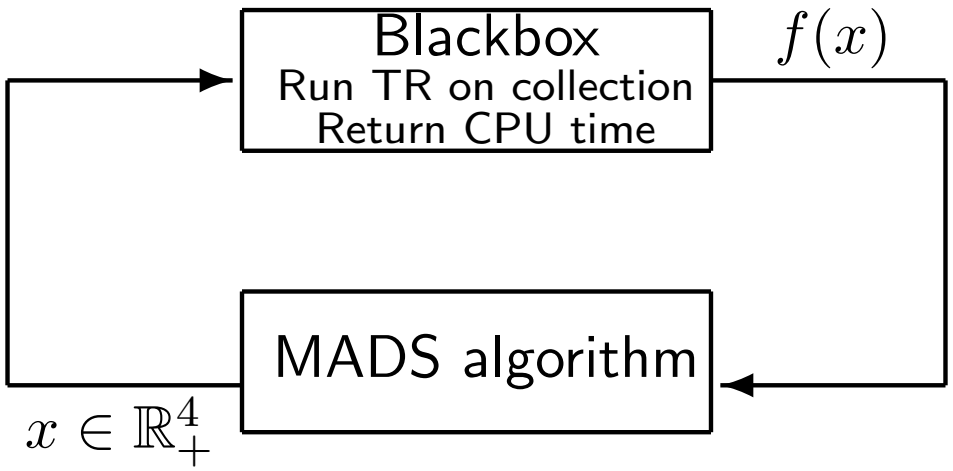
- [Audet and Orban, 2006].
- The classical trust-region algorithm depends on four parameters $x = (\eta_1, \eta_2, \alpha_1, \alpha_2) \in \mathbb{R}^4_+$.
- Consider a collection of 55 test problems from `CUTEr`.
- Let $f(x)$ be the CPU time required to solve the collection of problems by a trust-region algorithm with parameters x.
- $f(x_0) \simeq 3\text{h}45$ with the textbook values $x_0 = (1/4,\ 3/4,\ 1/2,\ 2)$.

Example: Trust-region parameter tuning (2/2)



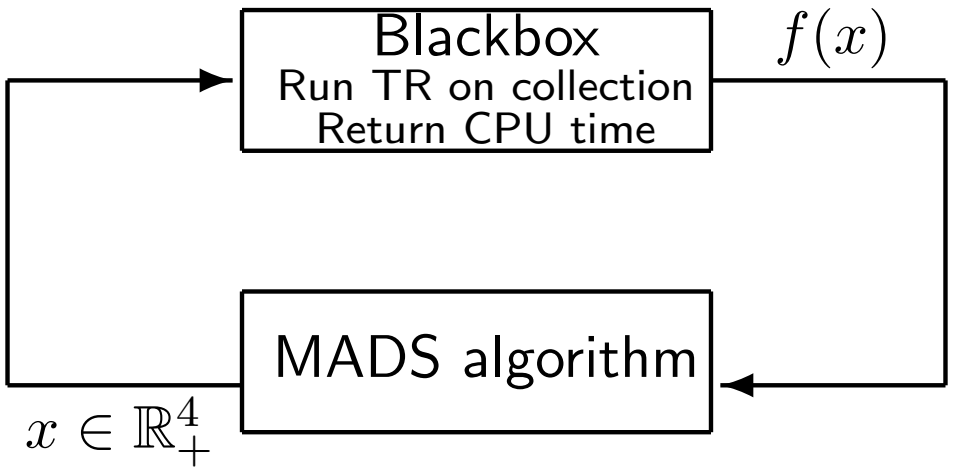
- This optimization produced $\hat{x}$ with $f(\hat{x}) \simeq 2\text{h}50$ (-30%).

Example: Trust-region parameter tuning (2/2)



- This optimization produced $\hat{x}$ with $f(\hat{x}) \simeq 2\text{h}50$ (-30%).
  Victory ?

Example: Trust-region parameter tuning (2/2)



- This optimization produced $\hat{x}$ with $f(\hat{x}) \simeq 2\text{h}50$ (-30%).
  Victory ? No, because
  $\hat{x} = (0.22125,\ 0.94457031,\ 0.37933594,\ 2.3042969)$.

Granular variables

The initial point $x_0 = (0.25,\ 0.75,\ 0.50,\ 2.00)$ is frequently used because each entry is a multiple of 0.25.

Its granularity is $\mathcal{G} = 0.25$

Granular variables

The initial point $x_0 = (0.25,\ 0.75,\ 0.50,\ 2.00)$ is frequently used because each entry is a multiple of 0.25.

Its granularity is $\mathcal{G} = 0.25$

- How can we devise a direct search algorithm so that it stops on a prescribed granularity?
  With a granularity of $\mathcal{G} = 0.05$, the code might produce
  $$\hat{x} = (0.20, 0.95, 0.40, 2.30).$$
  Which is much nicer (for a human) than
  $$\hat{x} = (0.22125,\ 0.94457031,\ 0.37933594,\ 2.3042969).$$

Granular variables

The initial point $x_0 = (0.25,\ 0.75,\ 0.50,\ 2.00)$ is frequently used because each entry is a multiple of 0.25.

Its granularity is $\mathcal{G} = 0.25$

- How can we devise a direct search algorithm so that it stops on a prescribed granularity?
  With a granularity of $\mathcal{G} = 0.05$, the code might produce
  $$\hat{x} = (0.20, 0.95, 0.40, 2.30).$$
  Which is much nicer (for a human) than
  $$\hat{x} = (0.22125,\ 0.94457031,\ 0.37933594,\ 2.3042969).$$
- This may be achieved using integer variables, together with a relative scaling, but there is a simpler way for mesh-based methods.

Blackbox optimization

Motivating example

The MADS algorithm

Computational experiments

Discussion

Mesh Adaptive Direct Search (MADS) in $\mathbb{R}^n$

- [Audet and Dennis, Jr., 2006].
- Iterative algorithm that evaluates the blackbox at some trial points on a spatial discretization called the mesh.
- One iteration = search and poll.
- The search allows trial points generated anywhere on the mesh.
- The poll consists in generating a list of trial points constructed from poll directions. These directions grow dense.
- At the end of the iteration, the mesh size is reduced if no new success point is found.

[0] Initializations (x_0, Δ_0: initial poll size)

[1] Iteration k

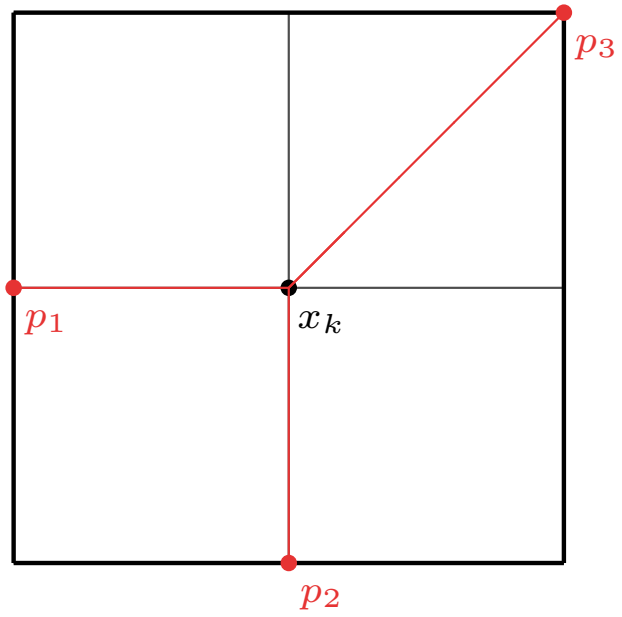
- let $\delta^k \leq \Delta^k$ be the mesh size parameter
- **Search**
  - test a finite number of mesh points
- **Poll** (if the Search failed)
  - construct set of directions D_k
  - test poll set $P_k = \{x_k + \delta^k d : d \in D_k\}$ with $\|\delta^k d\| \simeq \Delta_k$

[2] Updates

- if success
  - $x_{k+1} \leftarrow$ success point
  - increase Δ^k
- else
  - $x_{k+1} \leftarrow x_k$
  - decrease Δ^k
- $k \leftarrow k+1$, stop if $\Delta^k \leq \Delta_{\min}$ or go to **[1]**

Poll illustration (successive fails and mesh shrinks)

$$\delta^k = 1$$
$$\Delta^k = 1$$

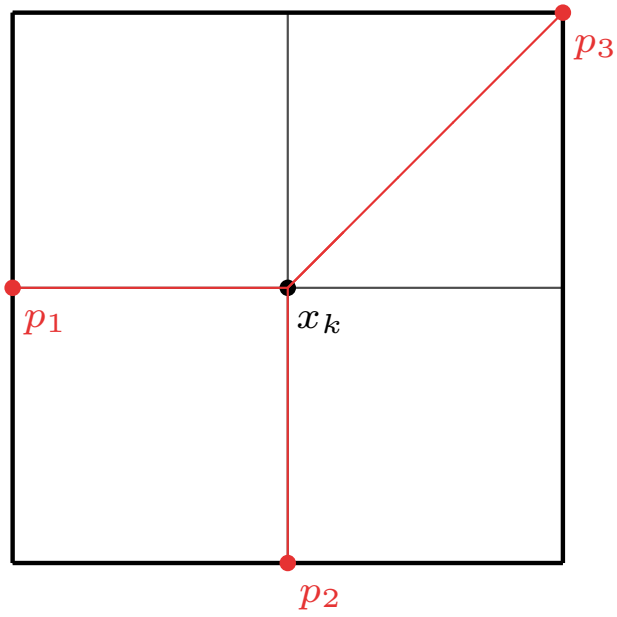


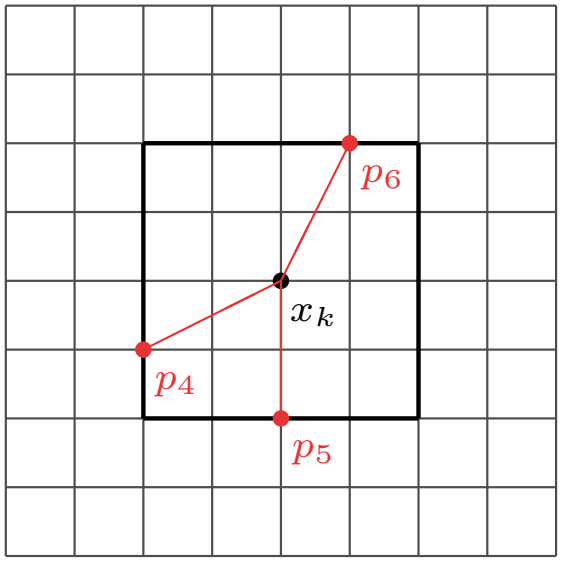
trial points$=\{p_1, p_2, p_3\}$

Poll illustration (successive fails and mesh shrinks)

$$\delta^k = 1 \qquad \Delta^k = 1$$

$$\delta^{k+1} = 1/4 \qquad \Delta^{k+1} = 1/2$$

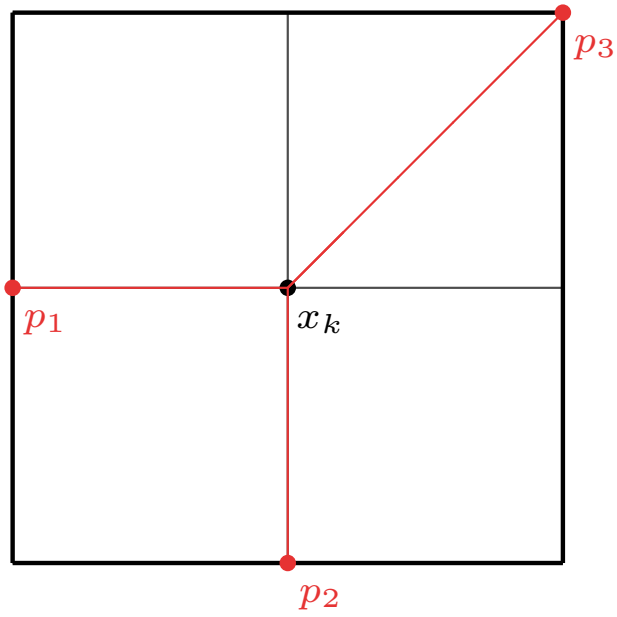




trial points$=\{p_1, p_2, p_3\}$ $\qquad = \{p_4, p_5, p_6\}$

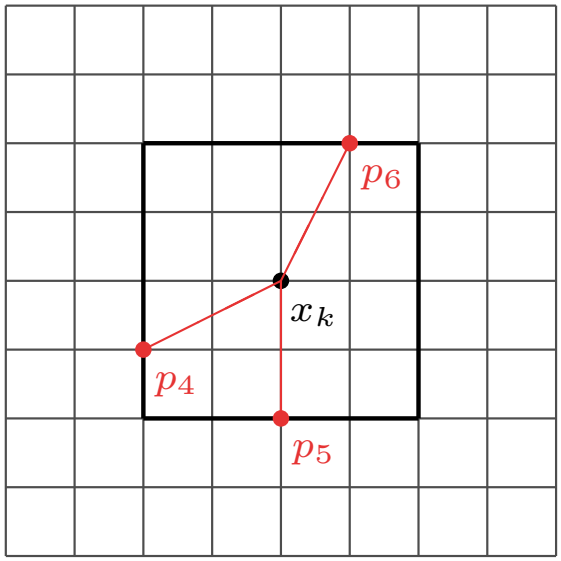
Poll illustration (successive fails and mesh shrinks)

$\delta^k = 1$
$\Delta^k = 1$



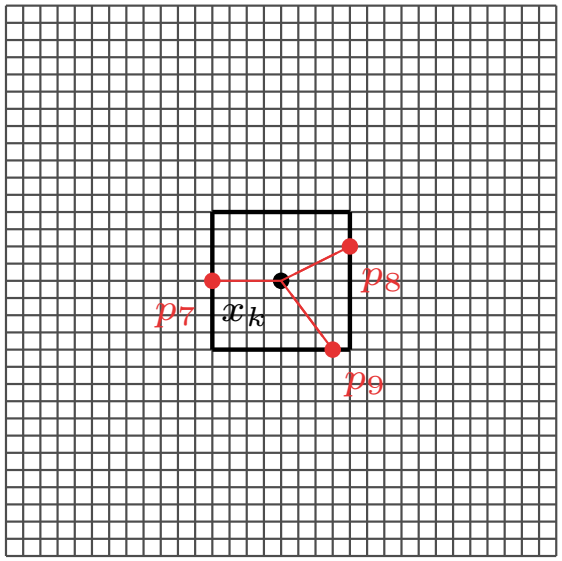
trial points$=\{p_1, p_2, p_3\}$

$\delta^{k+1} = 1/4$
$\Delta^{k+1} = 1/2$



$= \{p_4, p_5, p_6\}$

$\delta^{k+2} = 1/16$
$\Delta^{k+2} = 1/4$



$= \{p_7, p_8, p_9\}$

Discrete variables in MADS – so far

- MADS has been designed for continuous variables.
- Some theory exists for categorical variables [Abramson, 2004].
- So far: Only a patch allows to handle integer variables: Rounding + minimal mesh size of one.

In this work, we start from scratch and present direct search methods with a natural way of handling discrete variables.

Mesh refinement on $\min(x - 1/3)^2$

Δ^k	x^k
1	0
0.5	0.5
0.25	0.25
0.125	0.375
0.0625	0.3125
0.03125	0.34375
0.015625	0.328125
0.0078125	0.3359375
0.00390625	0.33203125
0.001953125	0.333984375

Mesh refinement on $\min(x - 1/3)^2$

Δ^k	x^k
1	0
0.5	0.5
0.25	0.25
0.125	0.375
0.0625	0.3125
0.03125	0.34375
0.015625	0.328125
0.0078125	0.3359375
0.00390625	0.33203125
0.001953125	0.333984375

alternately

Δ^k	x^k
1	0
0.5	0.5
0.2	0.4
0.1	0.3
0.05	0.35
0.02	0.34
0.01	0.33
0.005	0.335
0.002	0.332
0.001	0.333

Idea: Instead of dividing Δ^k by 2, change it so that

10×10^b refines to 5×10^b

5×10^b refines to 2×10^b

2×10^b refines to 1×10^b

Mesh refinement on $\min(x - 1/3)^2$

Δ^k	x^k
1	0
0.5	0.5
0.25	0.25
0.125	0.375
0.0625	0.3125
0.03125	0.34375
0.015625	0.328125
0.0078125	0.3359375
0.00390625	0.33203125
0.001953125	0.333984375

alternately

Δ^k	x^k
1	0
0.5	0.5
0.2	0.4
0.1	0.3
0.05	0.35
0.02	0.34
0.01	0.33
0.005	0.335
0.002	0.332
0.001	0.333

Idea: Instead of dividing Δ^k by 2, change it so that

10×10^b refines to 5×10^b

5×10^b refines to 2×10^b

2×10^b refines to 1×10^b

To get three decimals, one simply sets the granularity to 0.001.
Integer variables are treated by setting the granularity to $\mathcal{G} = 1$.

Poll and mesh size parameter update

- The poll size parameter Δ^k is updated as
  $10 \times 10^b \longleftrightarrow 5 \times 10^b \longleftrightarrow 2 \times 10^b \longleftrightarrow 1 \times 10^b$
- The fine underlying mesh is defined with the mesh size parameter
  $$\delta^k = \begin{cases} 1 & \text{if } \Delta^k \geq 1, \\ \max\{10^{2b}, \mathcal{G}\} & \text{otherwise, i.e. } \Delta^k \in \{1, 2, 5\} \times 10^b. \end{cases}$$
- Example: Granularity of $\mathcal{G} = 0.005$:

δ^k	Δ^k
1	5
1	2
1	1
0.01	0.5
0.01	0.2
0.01	0.1
0.005	0.05
0.005	0.02
0.005	0.01
0.005	0.005 $\leftarrow$ `stop`

[0] Initializations (x_0, $\Delta_0 \in \{1, 2, 5\} \times 10^b$, $\mathcal{G}$ granularity)

[1] Iteration k

- let $\delta^k \leq \Delta^k$ be the mesh size parameter
- **Search**
  - test a finite number of mesh points
- **Poll** (if the Search failed)
  - construct set of directions D_k
  - test poll set $P_k = \{x_k + \delta^k d : d \in D_k\}$ with $\|\delta^k d\| \simeq \Delta_k$

[2] Updates

- if success
  - $x_{k+1} \leftarrow$ success point
  - increase Δ^k
- else
  - $x_{k+1} \leftarrow x_k$
  - decrease Δ^k
- $k \leftarrow k + 1$, stop if $\delta^k = \mathcal{G}$ or go to **[1]**

Theory

- MADS analysis relies on *"the sequence of trial points are located on some discretization of the space of variables called the mesh"*.
- By multiplying or dividing Δ^k by a rational number τ, [Torczon, 1997] showed that all trial points from iteration 0 to ℓ were located on a fine underlying mesh. The proof is not trivial and uses the fact that $\tau \in \mathbb{Q}$, and does not work for $\tau \in \mathbb{R}$ (paper won SIAM Outstanding Paper Prize).

Theory

- MADS analysis relies on *"the sequence of trial points are located on some discretization of the space of variables called the mesh"*.
- By multiplying or dividing Δ^k by a rational number τ, [Torczon, 1997] showed that all trial points from iteration 0 to ℓ were located on a fine underlying mesh.
- With the new mesh, that technical part of the proof becomes:

Consider any trial point t considered from iteration $k = 0$ to ℓ.
If a granularity of $\mathcal{G}_i$ is requested on variable i,

$$t_i \text{ lies on the mesh of granularity } \mathcal{G}_i$$

if no granularity is requested on variable i,

$$t_i \text{ lies on the mesh of granularity } 10^{b_i}$$

with $b_i = \min\{b_i^k : k = 0, 1, \ldots, \ell\}$.

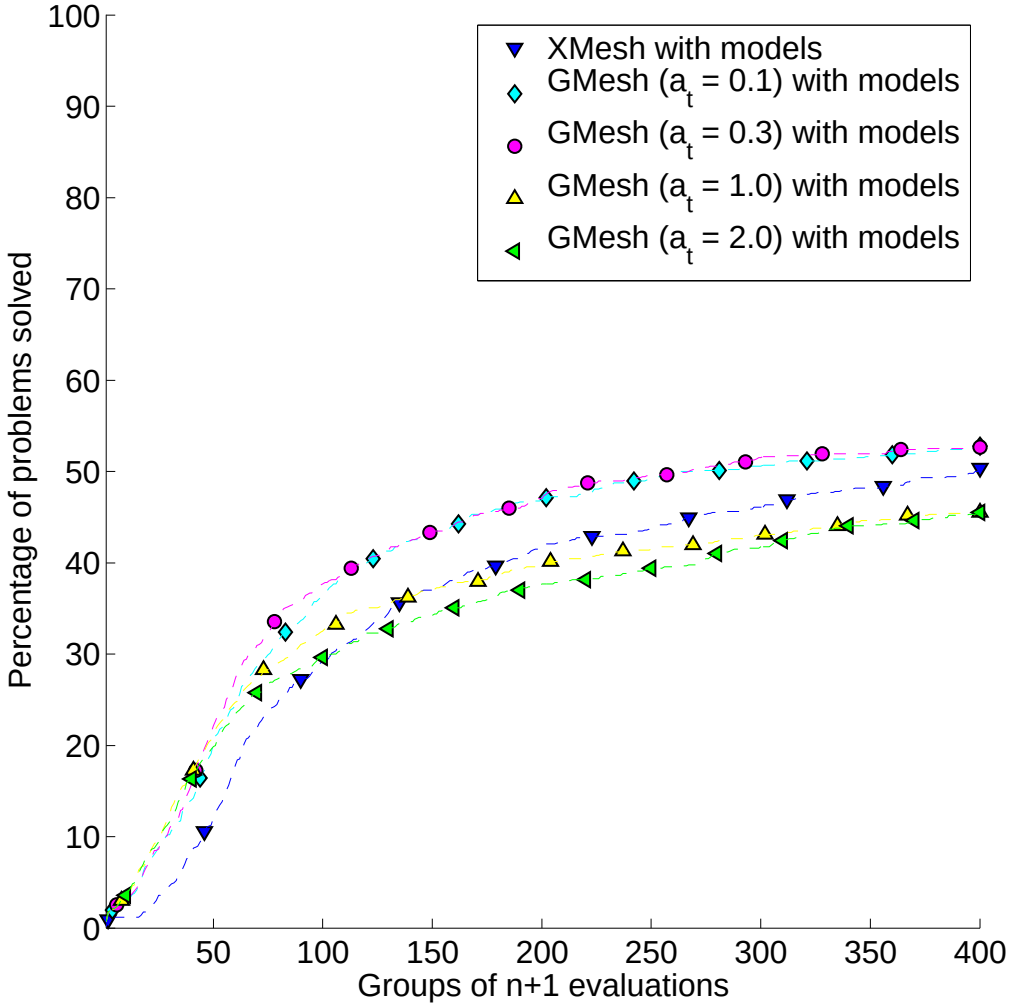
Results on continuous analytical problems

- 87 continuous analytical computational problems from the optimization literature ($n = 2$ to 20, 19 constrained).
- 10 LHS starting points are considered for each problem, for a total of 870 instances.
- `NOMAD 3.7.3` (previous version with classic mesh: XMesh)
  vs
  `NOMAD 3.8.1` (release with new mesh: GMesh).

  (current release is `NOMAD 3.9.0`).
- Data profiles [Moré and Wild, 2009].

Data profiles on continuous problems

XMesh vs GMesh for convergence tolerance $\tau = 10^{-7}$:

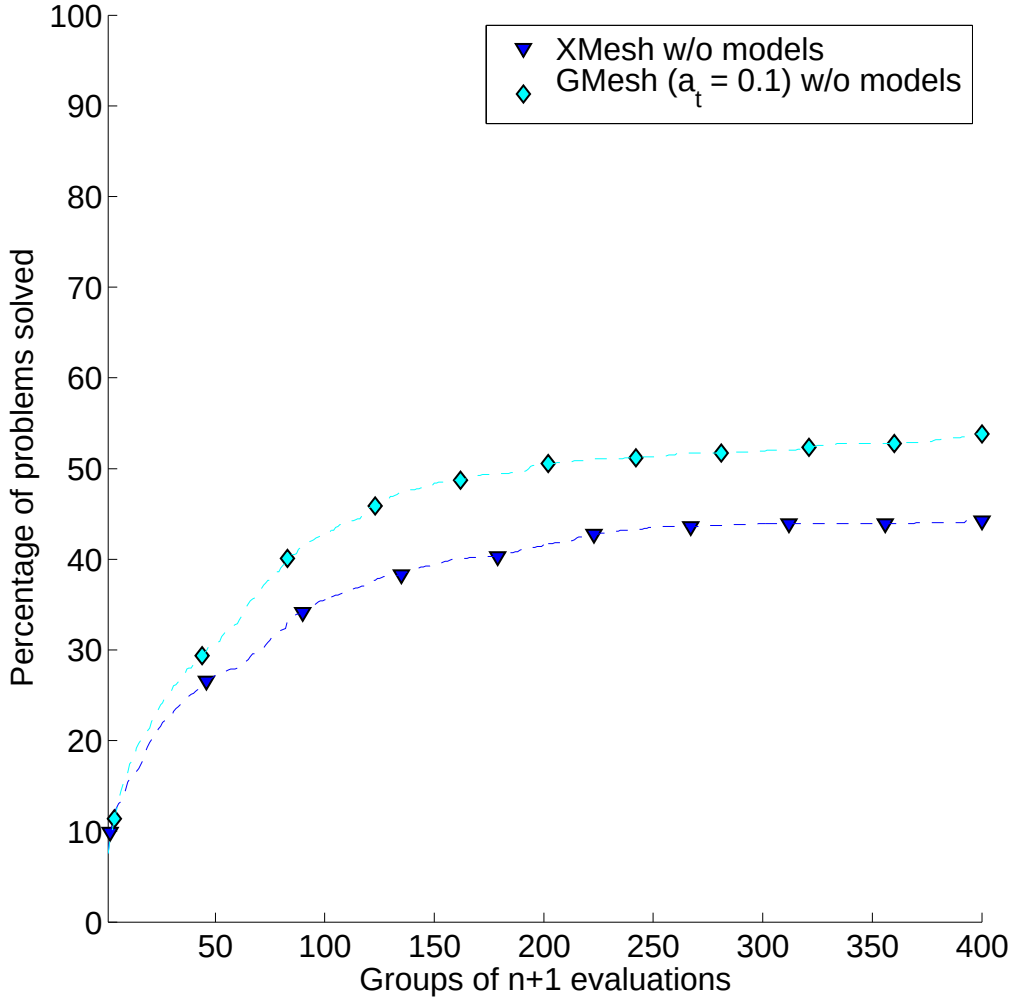


Results on problems with discrete variables

- Set of 94 analytical mixed-integer problems.
- 10 starting points for a total of 940 instances.
- GMesh vs XMesh.

Data profiles on discrete problems

XMesh vs GMesh for convergence tolerance $\tau = 10^{-7}$:

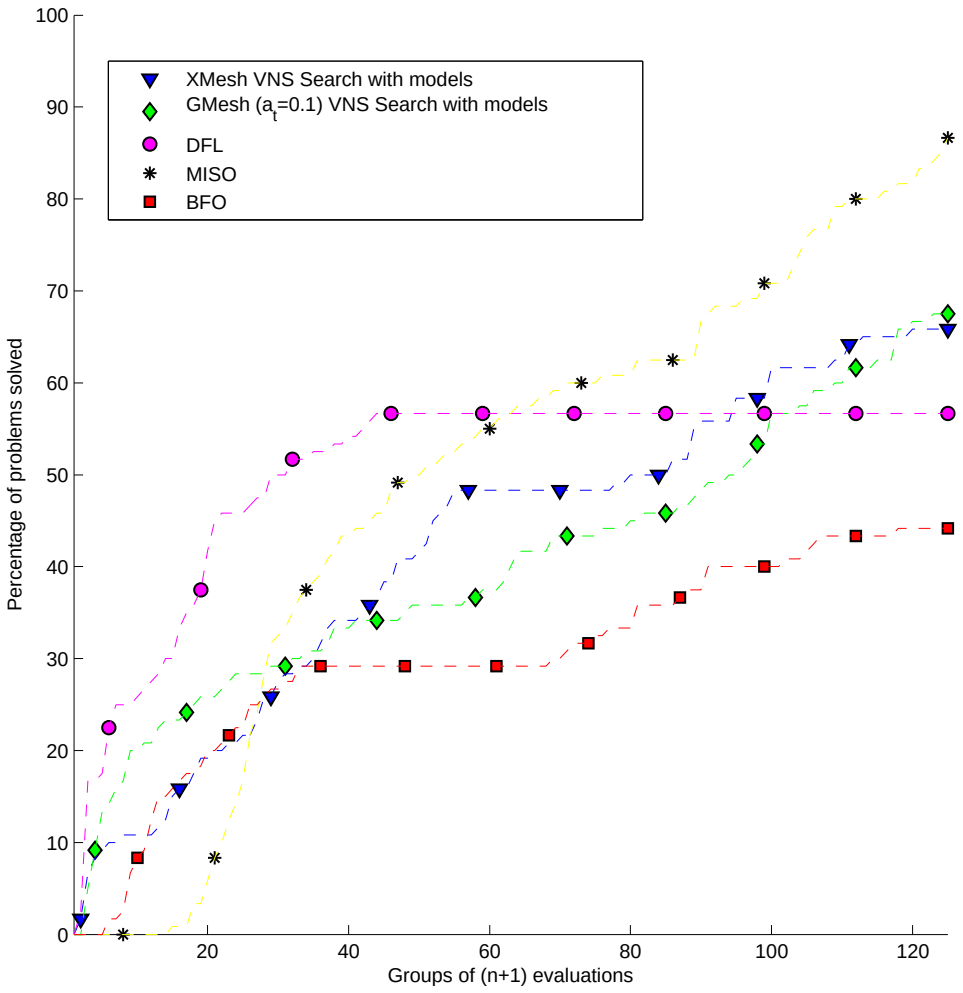


Comparison with other solvers on discrete problems

- Set of 12 unconstrained analytical integer problems.
- 10 starting points for a total of 120 instances.
- GMesh vs XMesh vs 3 other solvers:
  - DFL [Pillo et al., 2015]. *Not the 2018 version*.
  - MISO [Mueller, 2018].
  - BFO [Porcelli and Toint, 2017].

Data profiles for NOMAD vs other solvers

XMesh vs GMesh vs DFL vs MISO vs BFO for convergence tolerance $\tau = 10^{-7}$:



Computational times

Optimization times (in seconds) for 3 selected problems from a single starting point with evaluation budget of $150 \times n$ and a limit of 2,000 evaluations for MISO:

n	GMesh	XMesh	DFL	MISO	BFO
5	6	5	1*	69	0.1*
10	16	16	3*	298	0.3
15	58	57	8*	2,168	0.2

* converged before reaching the maximum evaluation budget.

Trust-region parameter tuning

Find the values of the four parameters $x = (\eta_1, \eta_2, \alpha_1, \alpha_2) \in \mathbb{R}^4_+$ that minimize the overall CPU time to solve 55 `CUTEr` problems.

- A surrogate function s is defined as the time to solve a collection of small-sized problems.
- In 2006, $f(x) \simeq$ 4h and $s(x) \simeq$ 1m. The surrogate was 200 times faster.

Trust-region parameter tuning

Find the values of the four parameters $x = (\eta_1, \eta_2, \alpha_1, \alpha_2) \in \mathbb{R}^4_+$ that minimize the overall CPU time to solve 55 `CUTEr` problems.

- A surrogate function s is defined as the time to solve a collection of small-sized problems.
- In 2006, $f(x) \simeq$ 4h and $s(x) \simeq$ 1m. The surrogate was 200 times faster.

Year	CPU $f(x)$	CPU $s(x)$	Ratio
2006	13,461s	69.0s	200
2018	1,008s	2.3s	440
Ratio	14	30	

Standard $\times 2 \div 2$ *versus* New $\{1, 2, 5\} \times 10^b$

0.25	0.75	0.50	2	329.64729
0.25	0.80	0.55	1.55	254.32179
0.25	0.95	0.70	1	NaN
0.35	0.95	0.60	1	NaN
0.15	0.70	0.60	1	NaN
0.30	0.75	0.40	1	NaN
0.40	0.70	0.65	1.55	299.24299
0.05	0.90	0.55	4.70	304.38179
0.25	0.90	0.55	1.55	230.25339
0.25	1.	0.55	1.55	NaN
0.15	0.85	0.60	2.45	332.73279
0.25	1.	0.50	1.10	NaN
0.20	0.95	0.75	1.10	384.09890
0.05	0.90	0.50	1.10	315.53730
0.30	0.85	0.50	1	NaN
0.45	0.70	0.50	5.15	376.13209
0.30	0.90	0.55	1.55	234.08379
0.25	0.90	0.65	1.55	267.51490
0.15	0.90	0.55	2	307.73540
0.25	0.80	0.55	2	344.48180
0.30	0.95	0.55	2.45	284.48099
0.30	0.95	0.45	1	NaN
0.1875	0.9125	0.625	1.775	324.89690
0.2125	0.925	0.5375	1.55	237.63839
0.2625	0.90	0.55	2	297.25700
0.2375	0.8875	0.60	1.55	260.38650
0.225	0.8625	0.5375	1.6625	316.95100
0.3125	0.9375	0.525	1	NaN
0.271875	0.909375	0.559375	1.6625	312.84760
0.25	0.925	0.55	1.55	231.82699
0.26875	0.90	0.546875	1.409375	252.02610
0.246875	0.90	0.571875	1.4375	319.19470
0.234375	0.90	0.5375	1.409375	255.43649
0.25	0.875	0.54375	1.94375	303.55739
0.24375	0.91875	0.54609375	1.59921875	238.37809
0.24609375	0.9109375	0.553125	1.54296875	305.85789
0.26171875	0.903125	0.55234375	1.5640625	269.32650
0.24765625	0.89921875	0.5515625	1.6625	325.77249
0.24921875	0.89609375	0.56171875	1.52890625	252.59420
0.24609375	0.890625	0.53125	1.4515625	313.61869
0.2578125	0.8998046875	0.5529296875	1.5623046875	259.61660
0.251171875	0.9060546875	0.5498046875	1.546484375	252.58099
0.2455078125	0.901171875	0.5515625	1.58515625	295.79539
0.25390625	0.899609375	0.5494140625	1.5939453125	244.50489
0.2515625	0.8998046875	0.5560546875	1.5447265625	251.01300
0.248046875	0.893359375	0.5431640625	1.4796875	265.02609
0.251806640625	0.901220703125	0.55107421875	1.55703125	268.87639
0.248876953125	0.902685546875	0.551123046875	1.553076171875	265.05930
0.252783203125	0.901123046875	0.550439453125	1.54296875	306.20870
0.250830078125	0.900341796875	0.54921875	1.575927734375	315.93150
0.250244140625	0.898876953125	0.552783203125	1.557470703125	261.50459
0.247314453125	0.89697265625	0.546435546875	1.520556640625	262.96580
0.249853515625	0.8998291015625	0.55010986328125	1.55120849609375	239.99700
0.2500732421875	0.90140380859375	0.55042724609375	1.5546142578125	268.07650
0.249755859375	0.9005126953125	0.5485595703125	1.54857177734375	238.33220
0.24979248046875	0.90042724609375	0.55035400390625	1.5370361328125	305.50679
0.25152587890625	0.9000732421875	0.54979248046875	1.547802734375	268.33620
0.24886474609375	0.8975830078125	0.55086669921875	1.56197509765625	264.80709
0.250225830078125	0.900222778320312	0.5490966796875	1.5502197265625	229.52039

0.25	0.75	0.5	2	330.1009
0.25	0.85	0.6	1	NaN
0.15	0.75	0.5	3	323.5899
0.05	0.75	0.5	4	340.3754
0.15	0.75	0.5	2	326.0686
0.05	0.75	0.4	4	330.5359
0.25	0.75	0.5	5	334.6560
0.15	0.85	0.5	3	316.2798
0.15	0.95	0.5	3	303.1888
0.15	1.	0.5	3	NaN
0.15	0.85	0.4	3	318.4759
0.05	1.	0.5	3	NaN
0.15	1.	0.6	3	NaN
0.35	1.	0.5	3	NaN
0.15	1.	0.5	5	NaN
0.05	0.05	0.4	1	NaN
0.15	0.85	0.475	3	313.9554
0.15	1.	0.476	2	NaN
0.15	1.	0.532	4	NaN
0.25	0.95	0.503	3	100286.7
0.15	0.95	0.55	2	303.1791
0.15	0.95	0.6	1	NaN
0.15	0.85	0.45	4	290.1130
0.15	0.75	0.35	6	338.5636
0.05	0.85	0.45	5	310.3020
0.15	0.75	0.45	1	NaN
0.15	1.	0.55	2	NaN
0.05	1.	0.25	3	NaN
0.05	0.55	0.55	3	400.4212
0.35	0.45	0.45	1	391.8666
0.2	0.65	0.45	4	351.8827
0.138	0.85	0.35	4	316.5801
0.151	0.85	0.45	6	291.6727
0.2	0.85	0.45	4	297.7219
0.149	1.	0.45	4	NaN
0.112	0.65	0.55	2	409.2380
0.141	0.85	0.4	5	316.5398
0.15	0.95	0.456	4	283.9472
0.15	1.	0.462	4	NaN
0.2	0.65	0.456	5	346.7783
0.147	0.95	0.556	4	311.6209
0.142	1.	0.456	3	NaN
0.174	1.	0.456	5	NaN
0.1	0.95	0.456	4	280.4177
0.05	0.95	0.456	4	283.5251
0.1	0.85	0.556	4	314.9984
0.1	0.95	0.456	5	281.0185
0.2	0.95	0.456	4	274.8974
0.3	0.95	0.456	4	282.7640
0.2	0.65	0.356	3	344.0217
0.3	0.95	0.456	5	282.3462
0.1	0.95	0.556	4	317.3498
0.2	0.75	0.456	4	329.5962
0.4	0.95	0.556	4	304.3251
0	1.	0.256	3	NaN
0.3	0.85	0.517	4.5	289.2744
0.3	0.95	0.414	3.5	237.7690
0.4	0.95	0.372	3	277.6089

Trust-region parameter tuning: Results

Algo.	$\mathcal{G}$	Solution	f^{2018}	Improv. (%)
Classical		(0.25, 0.75, 0.5, 2)	1,008.0	0
XMesh		(0.2939819787, 0.979406601, 0.4716387306, 1.474147761)	733.6	27
GMesh		(0.672010424, 0.685829734, 0.061485394, 1.34816385)	727.0	28
GMesh	0.005	(0.845, 0.99, 0.485, 1.575)	697.6	31
GMesh	0.01	(0.74, 0.99, 0.17, 1.34)	688.7	32
GMesh	0.05	(0.2, 0.9, 0.2, 1.3)	768.4	24

Observations:

- On this example, the new strategies seem preferable.
- Trust-region recommendation for humans:

$$(\eta_1,\ \eta_2,\ \alpha_1,\ \alpha_2) = (0.74,\ 0.99,\ 0.17,\ 1.34).$$

Discussion (1/2)

- New mesh parameter update rules to control the number of decimals $\{1, 2, 5\} \times 10^b$:
  - A native way to handle granularity of variables.
  - Integer variables are handled by setting $\mathcal{G} = 1$.

Discussion (1/2)

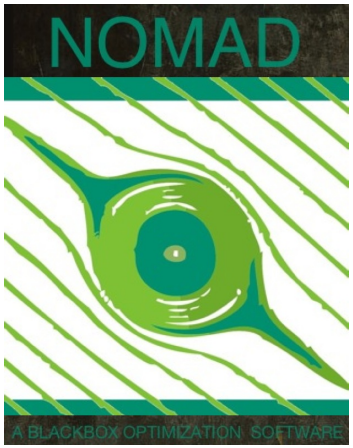
- New mesh parameter update rules to control the number of decimals $\{1, 2, 5\} \times 10^b$:
  - A native way to handle granularity of variables.
  - Integer variables are handled by setting $\mathcal{G} = 1$.
- Computational experiments on trust-region parameters:
  - New parameters reduce CPU time by $\simeq 30\%$ (versus textbook).
  - New parameters have granularity 0.01 (readable by humans).

Discussion (1/2)

- New mesh parameter update rules to control the number of decimals $\{1, 2, 5\} \times 10^b$:
  - A native way to handle granularity of variables.
  - Integer variables are handled by setting $\mathcal{G} = 1$.
- Computational experiments on trust-region parameters:
  - New parameters reduce CPU time by $\simeq 30\%$ (versus textbook).
  - New parameters have granularity 0.01 (readable by humans).
- Computational experiments on analytical problems: $\simeq$ 3% performance improvement over the previous `NOMAD` version.

Discussion (2/2)

- Associated paper submitted [Audet et al., 2018].
- This is part of our NOMAD 3.9 software (06-2018):
  - The only additional input from the user is $\mathcal{G}$.
  - ... and it is optional.
  - www.gerad.ca/nomad.



References I

Abramson, M. (2004).
Mixed variable optimization of a Load-Bearing thermal insulation system using a filter pattern search algorithm.
Optimization and Engineering, 5(2):157–177.

Audet, C. and Dennis, Jr., J. (2006).
Mesh Adaptive Direct Search Algorithms for Constrained Optimization.
SIAM Journal on Optimization, 17(1):188–217.

Audet, C., Digabel, S. L., and Tribes, C. (2018).
The mesh adaptive direct search algorithm for granular and discrete variables.
Technical Report G-2018-16, Les cahiers du GERAD.

Audet, C. and Orban, D. (2006).
Finding optimal algorithmic parameters using derivative-free optimization.
SIAM Journal on Optimization, 17(3):642–664.

Le Digabel, S. (2011).
Algorithm 909: NOMAD: Nonlinear Optimization with the MADS algorithm.
ACM Transactions on Mathematical Software, 37(4):44:1–44:15.

Moré, J. and Wild, S. (2009).
Benchmarking derivative-free optimization algorithms.
SIAM Journal on Optimization, 20(1):172–191.

References II

Mueller, J. (2018).
Homepage.
https://ccse.lbl.gov/people/julianem/.

Pillo, G., G.Fasano, G.Liuzzi, S.Lucidi, V.Piccialli, F.Rinaldi, and M.Sciandrone (2015).
DFL - Derivative-Free Library; A software library for derivative-free optimizationt.
Software available at http://www.dis.uniroma1.it/~lucidi/DFL/.

Porcelli, M. and Toint, P. (2017).
BFO, A Trainable Derivative-free Brute Force Optimizer for Nonlinear Bound-constrained Optimization and Equilibrium Computations with Continuous and Discrete Variables.
ACM Transactions on Mathematical Software, 44(1):6:1–6:25.

Torczon, V. (1997).
On the convergence of pattern search algorithms.
SIAM Journal on Optimization, 7(1):1–25.