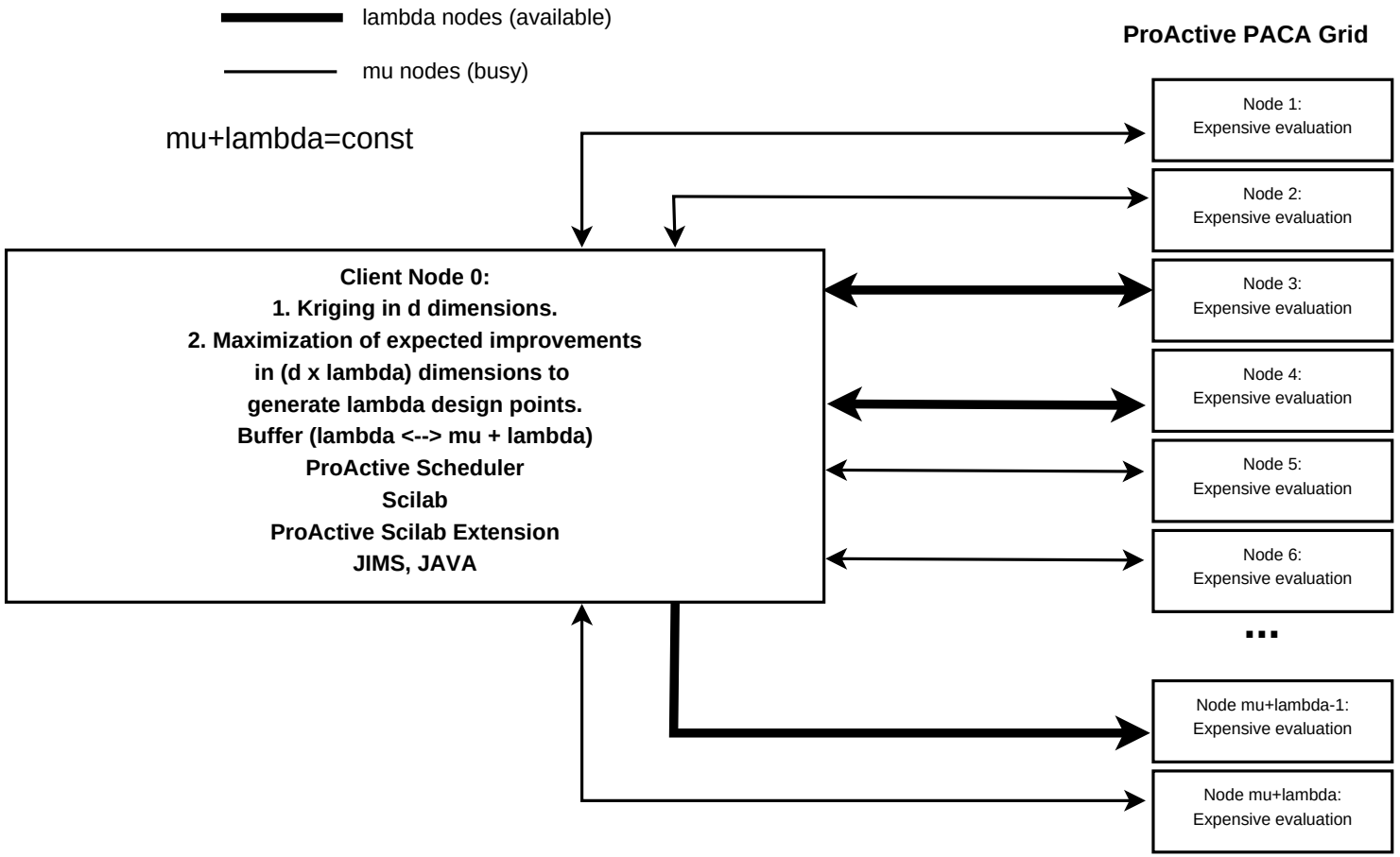
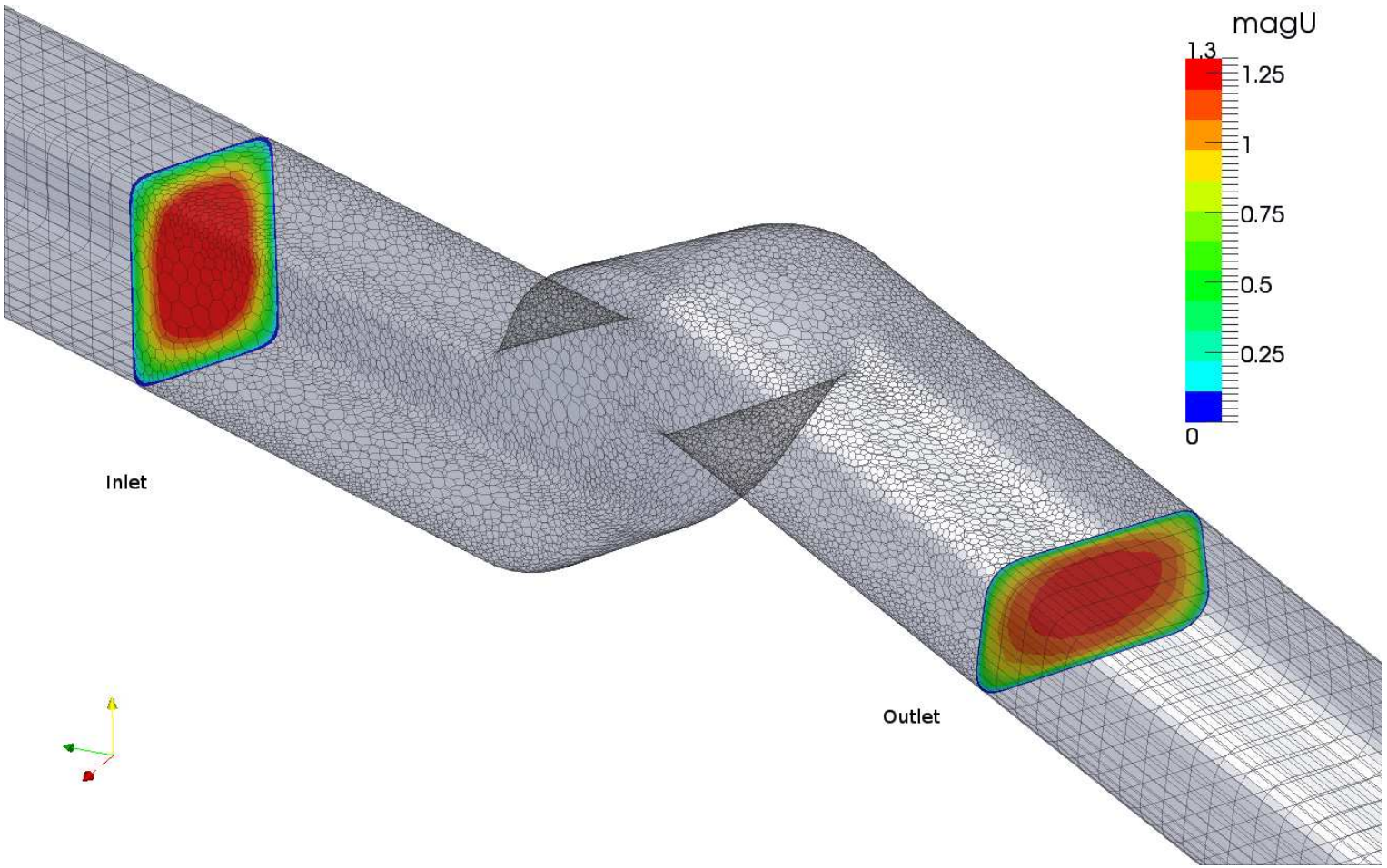
Parallel Optimization with Kriging and Multi-Point Improvements: Industrial Application

Ramūnas Girdziušas (EMSE), Rodolphe Le Riche (EMSE),
David Ginsbourger (IMSV), and Fabien Viale (INRIA)

June 25, 2012

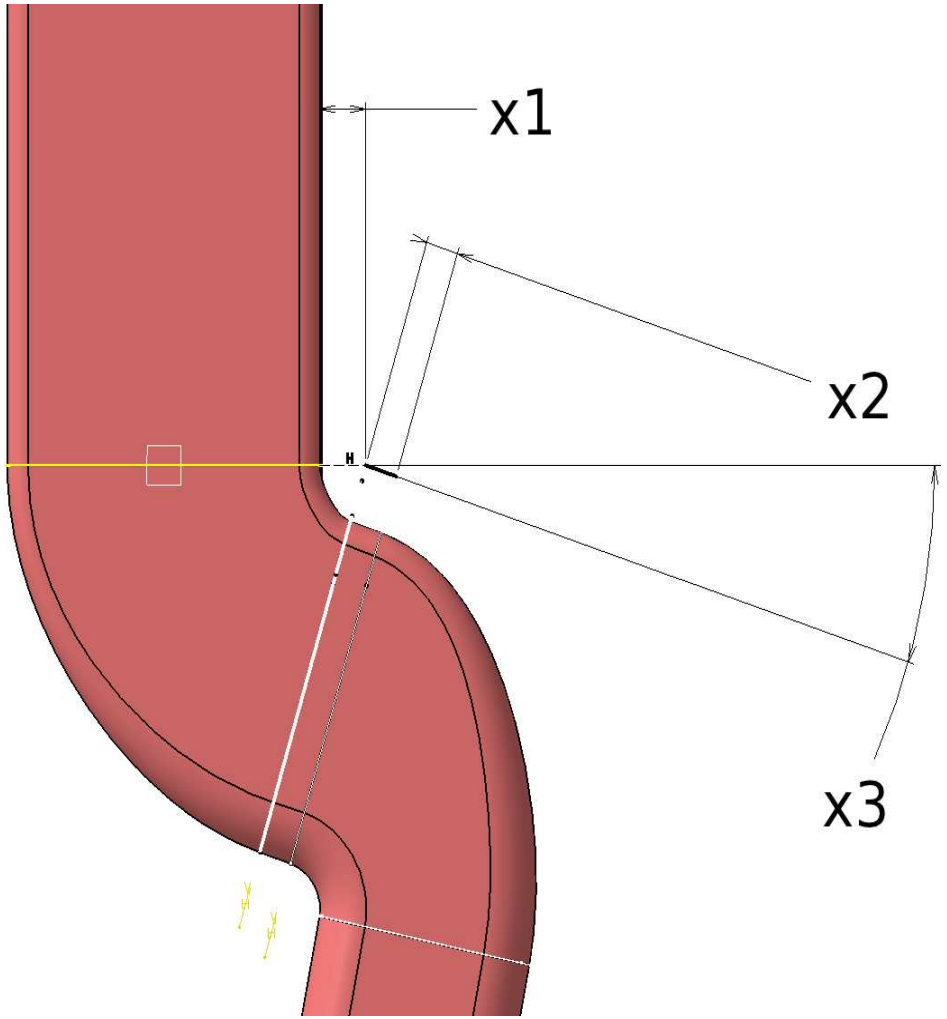
Platform

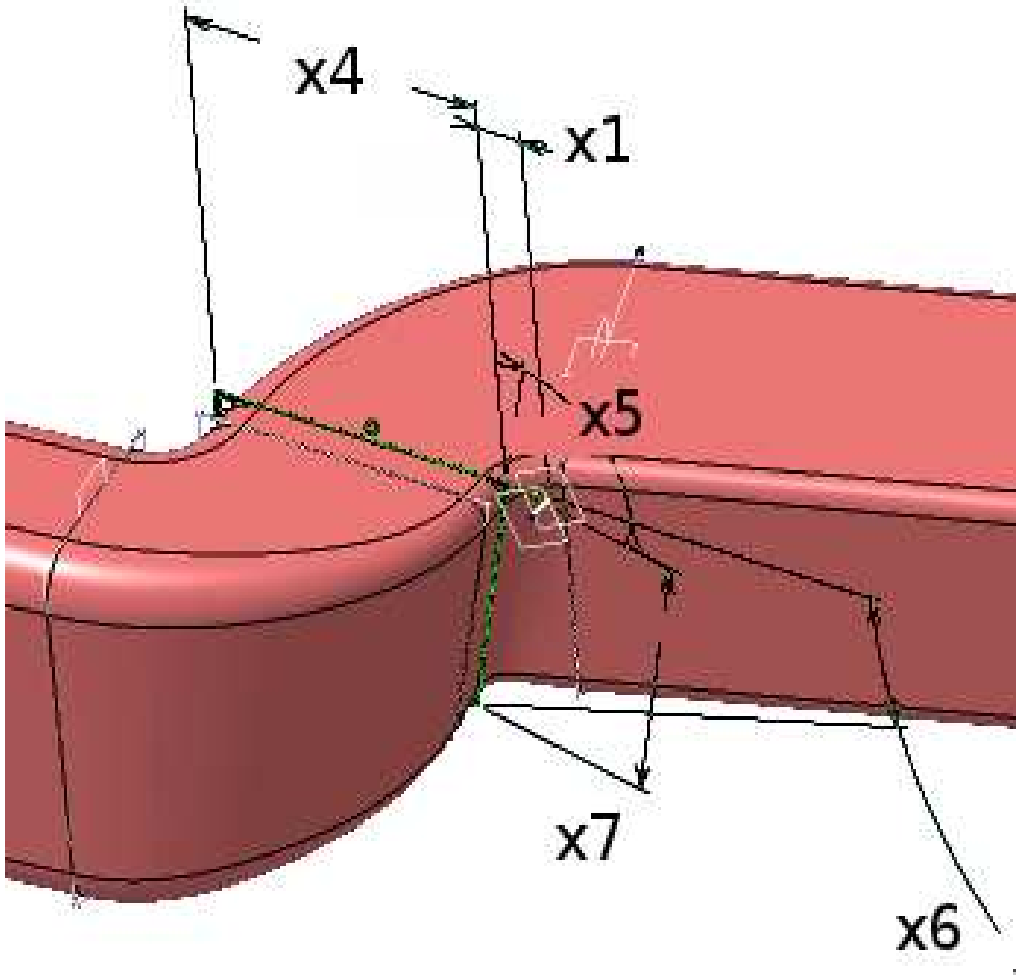

lambda nodes (available)
mu nodes (busy)
mu+lambda=const
ProActive PACA Grid
Client Node 0:
1. Kriging in d dimensions.
2. Maximization of expected improvements
in (d x lambda) dimensions to
generate lambda design points.
Buffer (lambda <--> mu + lambda)
ProActive Scheduler
Scilab
ProActive Scilab Extension
JIMS, JAVA
Node 1:
Expensive evaluation
Node 2:
Expensive evaluation
Node 3:
Expensive evaluation
Node 4:
Expensive evaluation
Node 5:
Expensive evaluation
Node 6:
Expensive evaluation
...
Node mu+lambda-1:
Expensive evaluation
Node mu+lambda:
Expensive evaluation

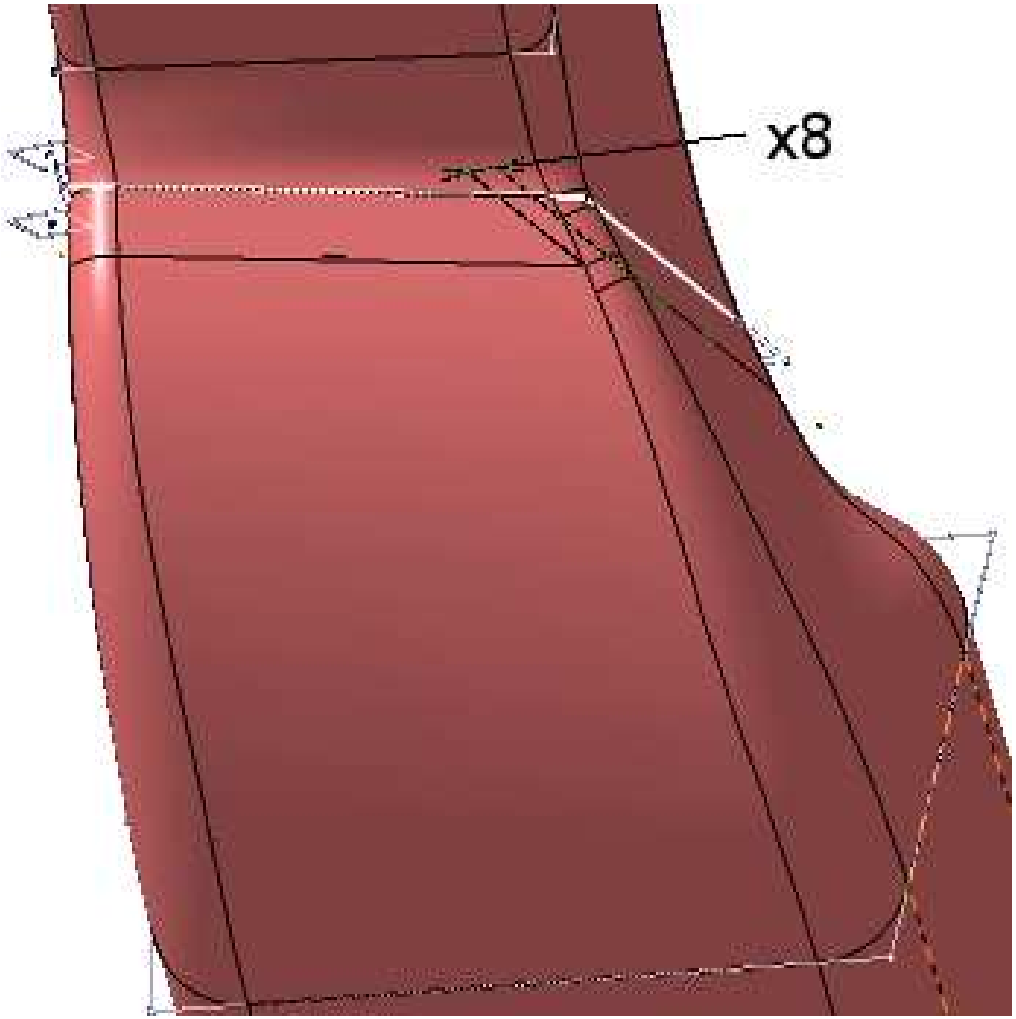

magU
1.3
1.25
1
0.75
0.5
0.25
0
Inlet
Outlet

Test Case II

- Minimization of pressure diff. (PD) at the intlet and outlet w.r.t. the geometry of an air-duct.
- The fluid is:
  - linear,
  - viscous ($\nu = 1.6 \cdot 10^{-4}\ m^2/s.$),
  - and turbulent ($Re = 4000$).
- OpenFOAM implementation of the k-ϵ model.
- Describes time averages of the velocity and pressure fields.
- Four NS-like equations coupled to three nonlinear-algebraic PDEs via the scalar fields k, ϵ, and ν_T.
- About 60K elements, 10–30min. to evaluate PD.


x1
x2
x3


x4
x1
x5
x7
x6


x8

1 point $\approx$ 25 min., 300 points in the initial DOE, $EI^{0,4}$ *algorithm*

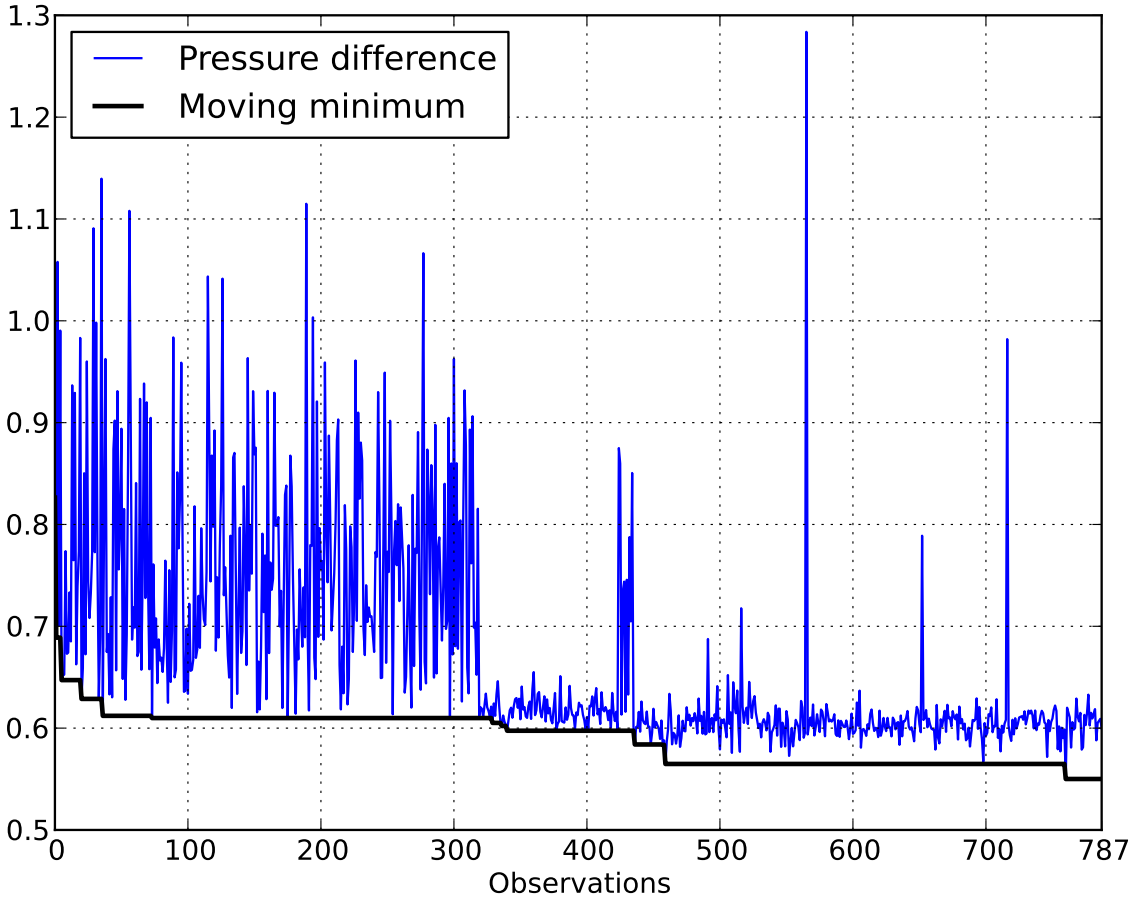


Table: Main Results

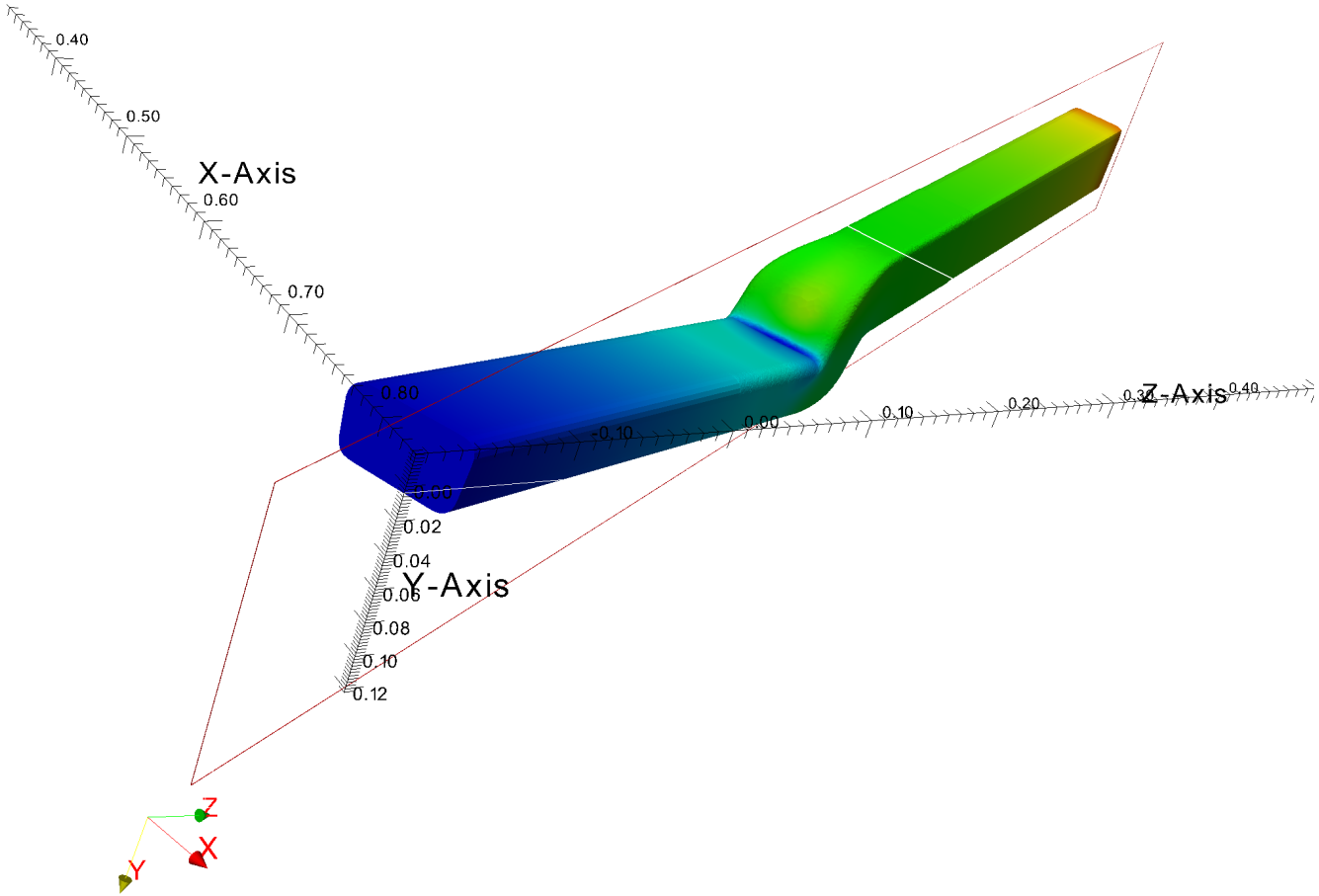
	LOBS	UPBS	Worst	[Raghavan et al. 2012]	Our result
x_1	0.0036	0.0166	0.0036	0.0149	0.0132
x_2	0.3	0.8	0.3760	0.4202	0.4756
x_3	0.0027	0.0207	0.0207	0.0102	0.0207
x_4	0.0405	0.0595	0.0595	0.0479	0.0450
x_5	1.25	1.5707	1.2525	1.5582	1.5707
x_6	0.21	0.42	0.2254	0.3849	0.3914
x_7	0.047	0.055	0.055	0.0541	0.0547
x_8	0.0008	0.0088	0.00081	0.0014	0.0016
pd	nan	nan	1.28	0.59 ± 0.01	0.56 ± 0.01

Streamlines (worst case)

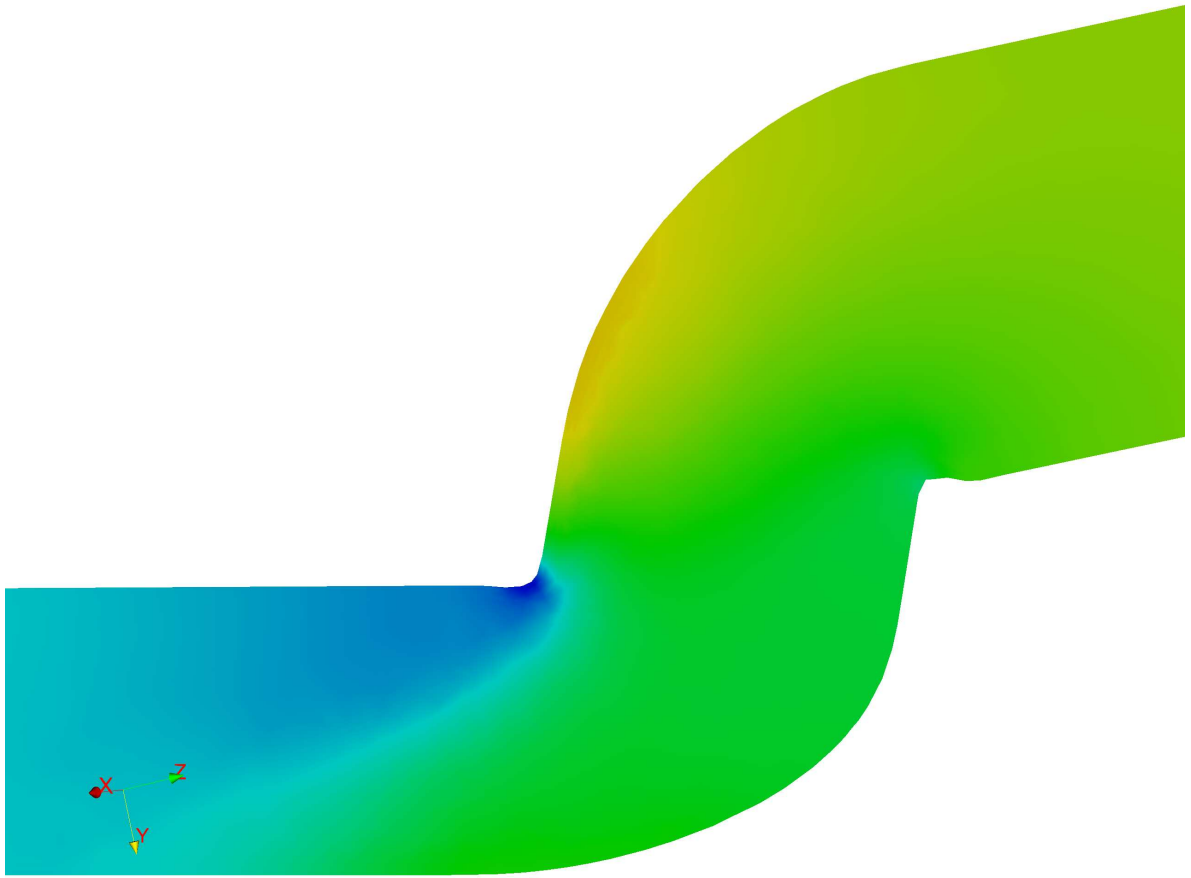
Streamlines [Raghavan et al., 2012]

Streamlines (our result)

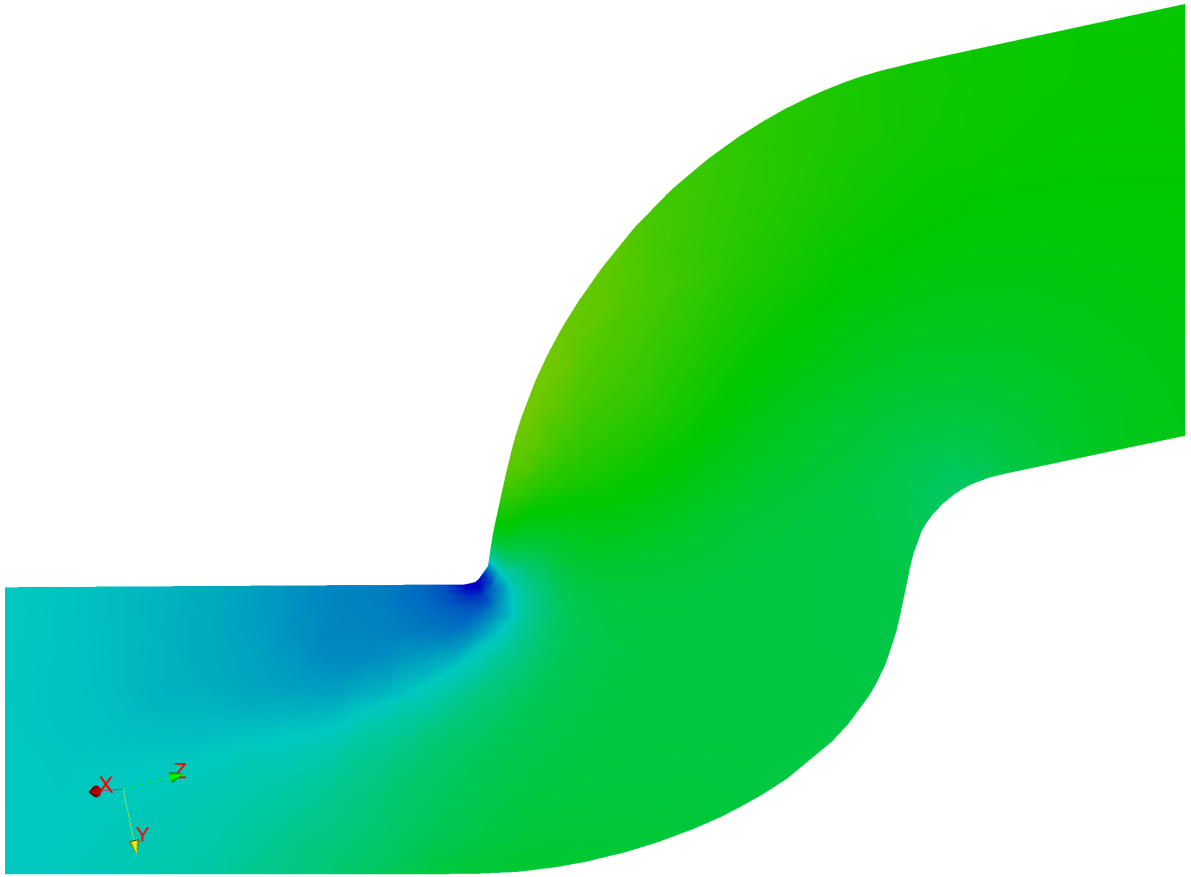
Observation slice (best design)



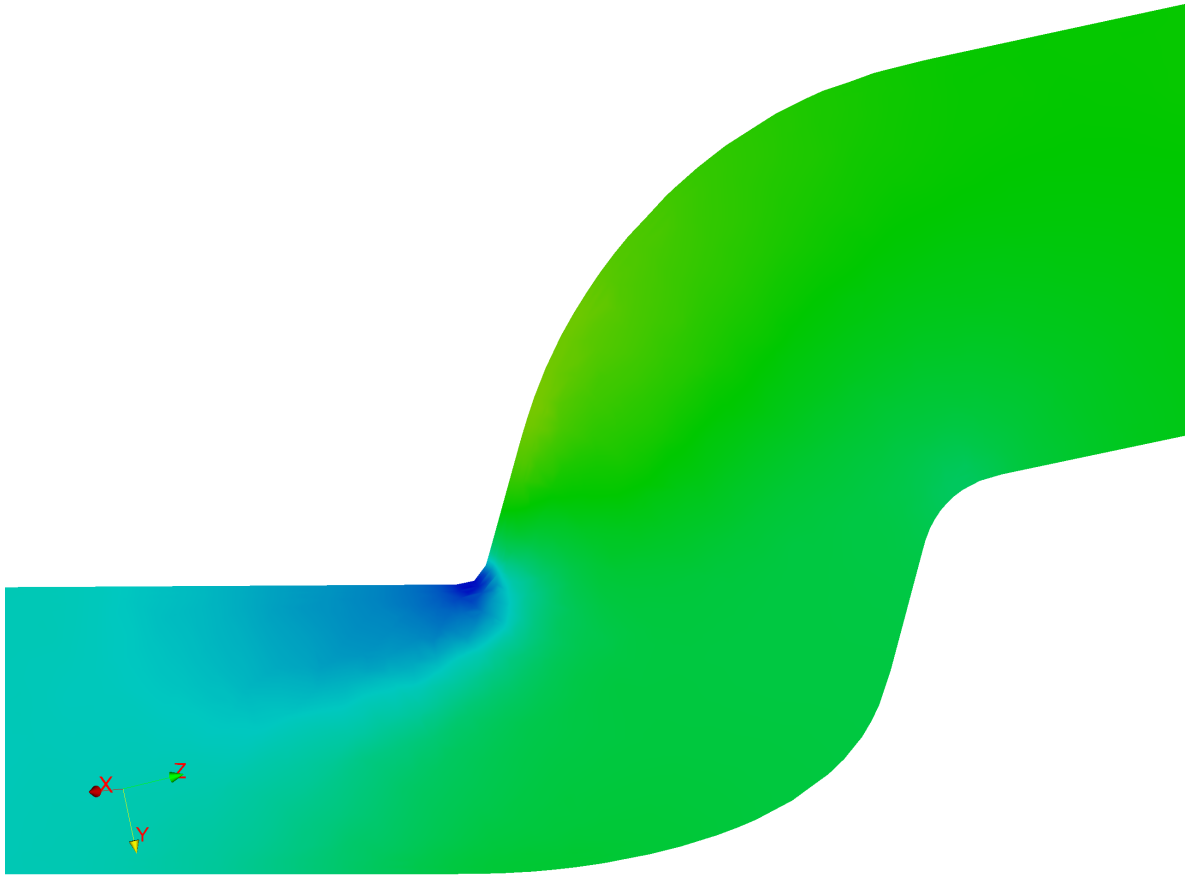
Pressure field (worst case)



Pressure field [Raghavan et al., 2012]

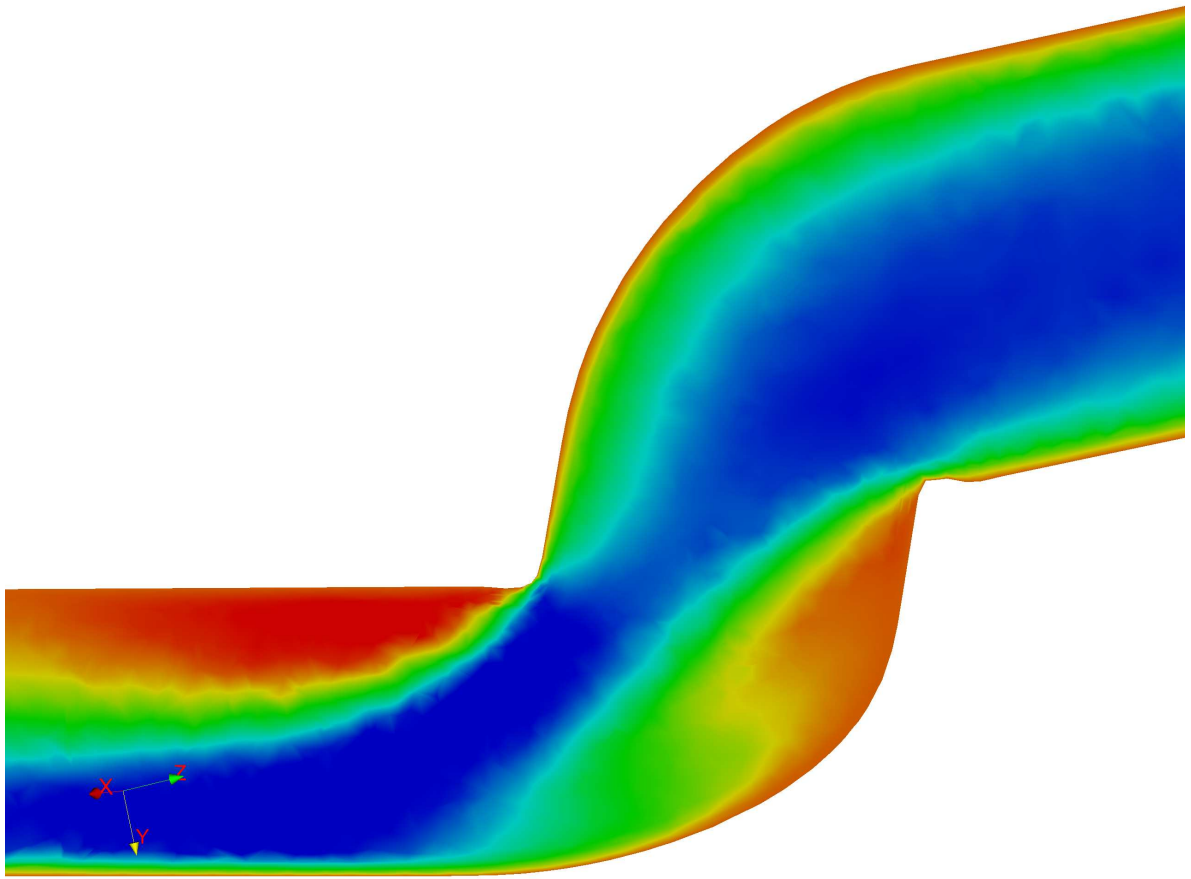


Pressure field (our result)



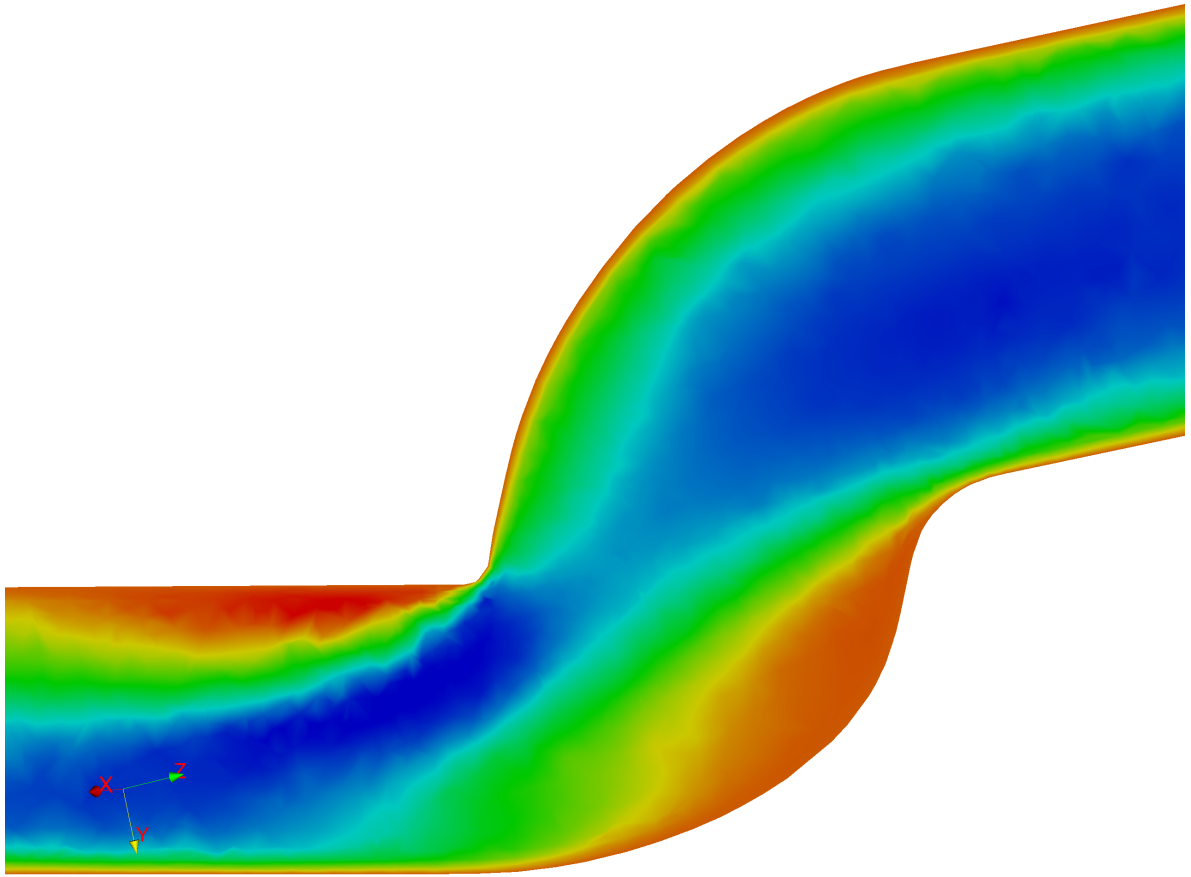
Velocity field (worst case)

z-component, blue=-1.6m/s, red=0.2m/s



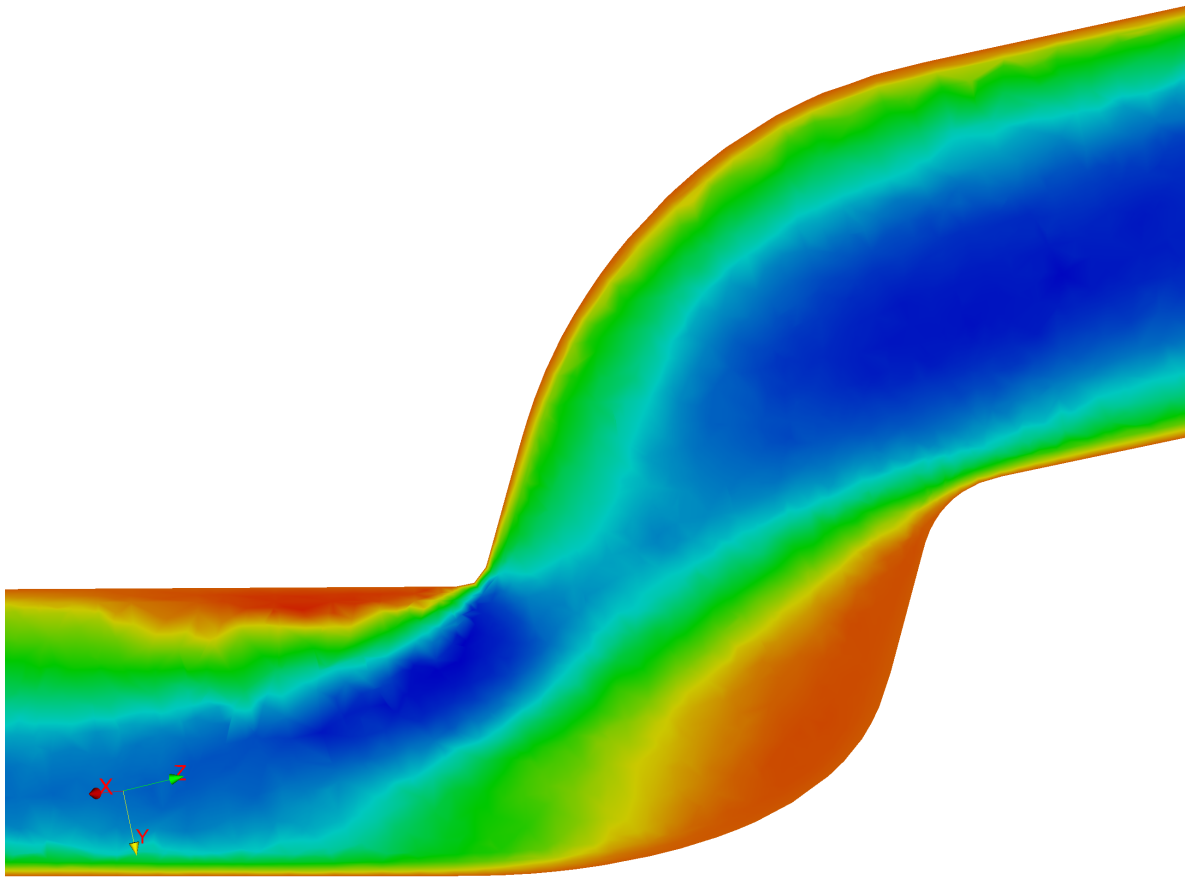
Velocity field, z-component [Raghavan et al., 2012]

z-component, blue=-1.6m/s, red=0.2m/s

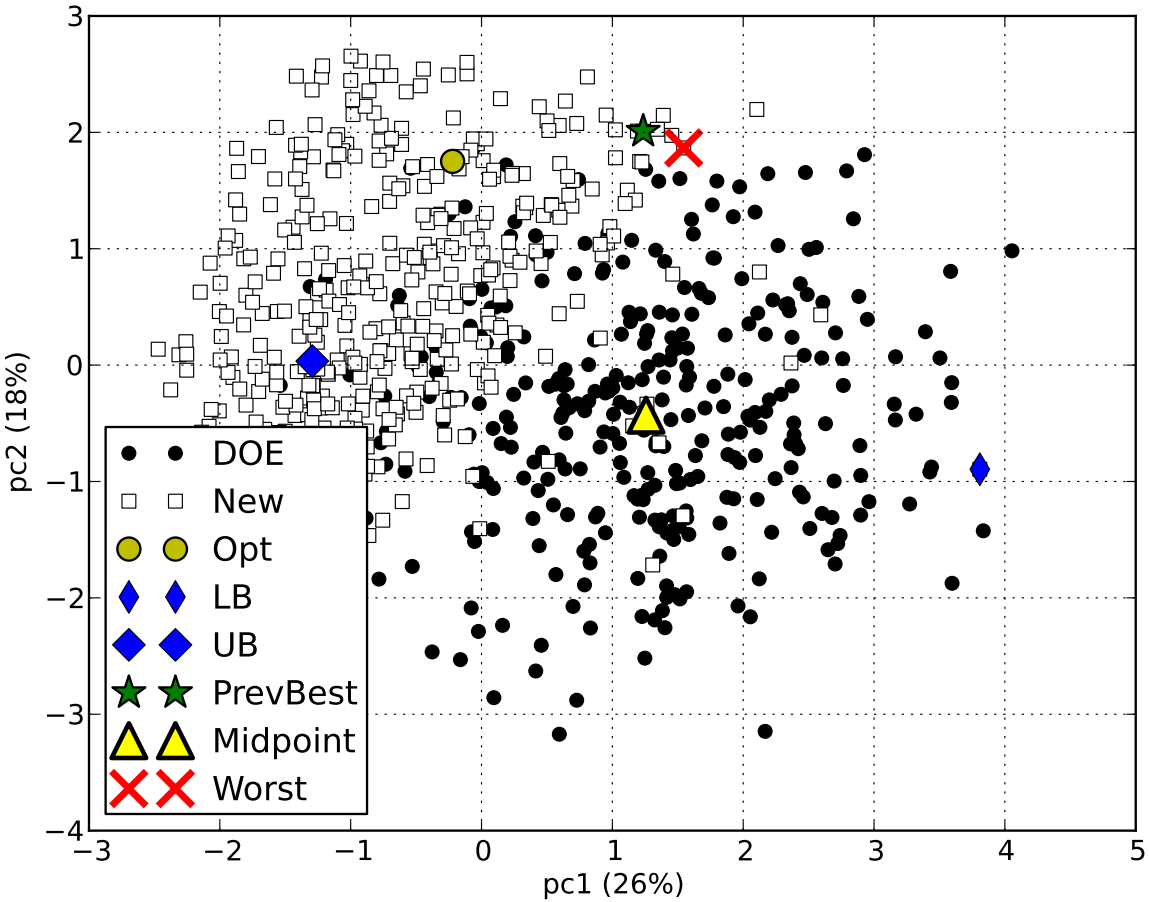


Velocity field, z-component (our result)

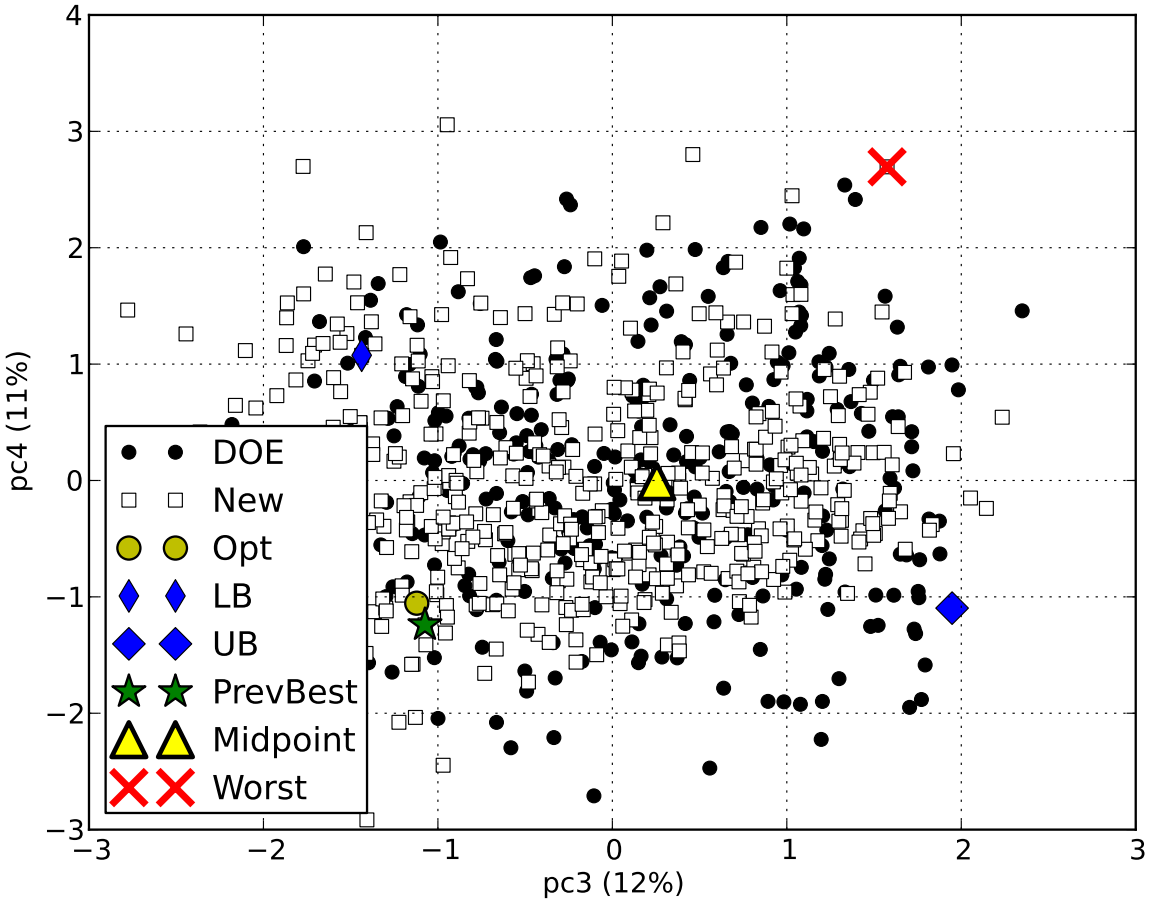
z-component, blue=-1.6m/s, red=0.2m/s



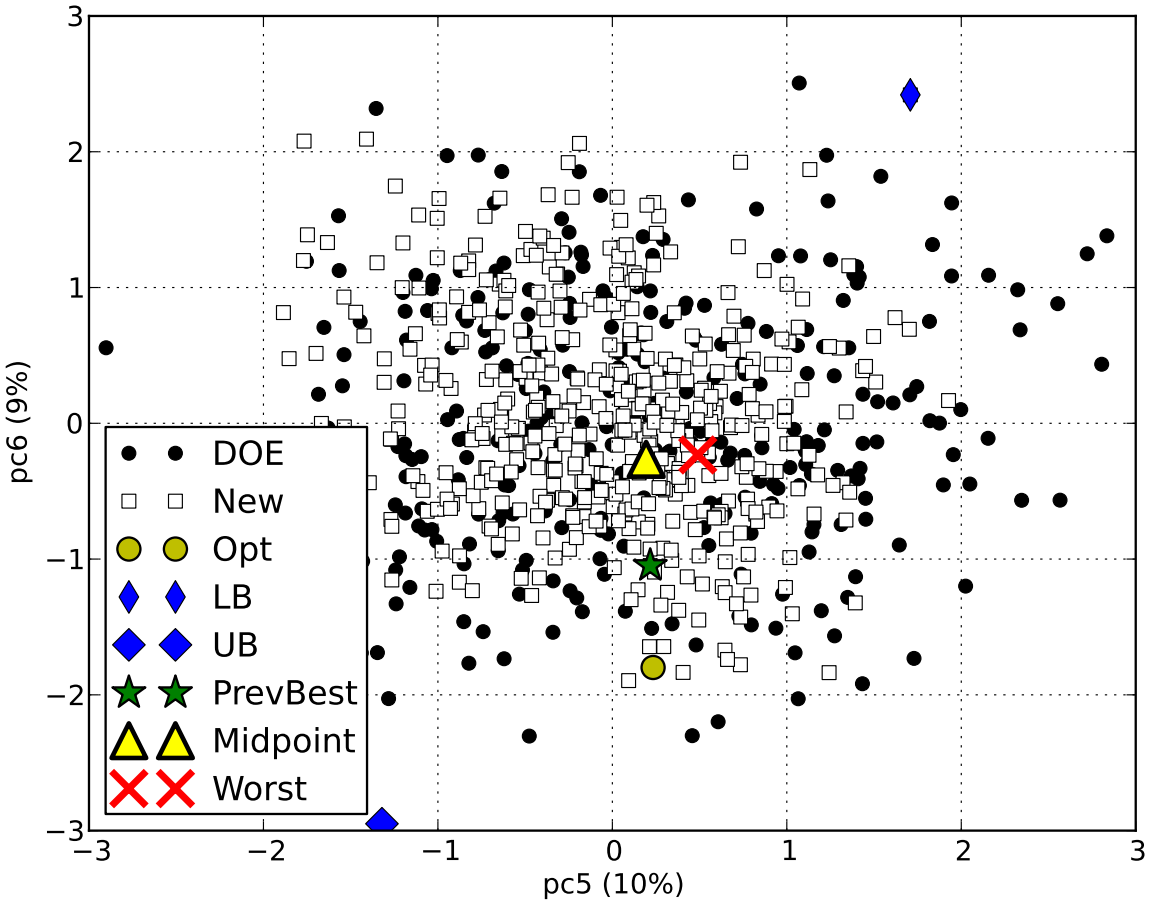
1st and 2nd principal components of generated design points



3rd and 4th principal components of generated design points



5^{th} and 6^{th} principal components of generated design points



7^{th} and 8^{th} principal components of generated design points

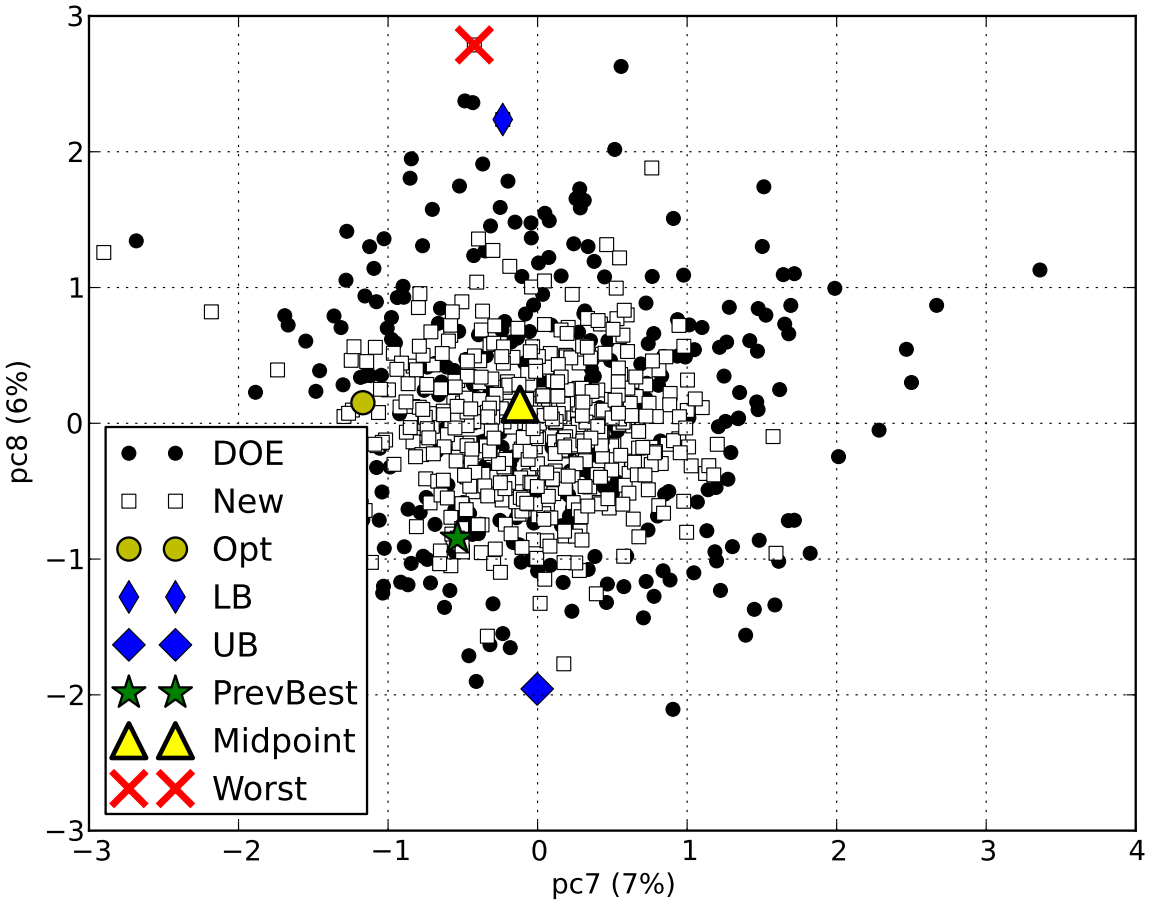


Table: The eigenvectors $\mathbf{v}_1$–$\mathbf{v}_4$ of the data correlation matrix.

Coord.	$\mathbf{v}_1$	$\mathbf{v}_2$	$\mathbf{v}_3$	$\mathbf{v}_4$
1	0.47061	−0.01957	0.00218	−0.34455
2	−0.24119	−0.39696	0.20201	−0.27114
3	−0.50275	−0.03497	0.19678	0.10206
4	0.04089	0.33272	0.87867	0.14932
5	−0.49912	0.22259	−0.30076	−0.06057
6	−0.33944	0.20969	0.11621	−0.74648
7	−0.16022	0.56533	−0.18227	0.33491
8	−0.27554	−0.56302	0.10589	0.31937

Table: The eigenvectors $\mathbf{v}_5$–$\mathbf{v}_8$ of the data correlation matrix.

Coord.	$\mathbf{v}_5$	$\mathbf{v}_6$	$\mathbf{v}_7$	$\mathbf{v}_8$
1	−0.09434	−0.65004	−0.26550	−0.39683
2	−0.80995	0.01527	0.04344	0.10855
3	0.19948	−0.48401	−0.50937	0.40419
4	−0.04204	0.13262	−0.04856	−0.26749
5	−0.11561	0.22181	−0.39944	−0.62056
6	0.29979	−0.07762	0.41716	0.01265
7	−0.36915	−0.44528	0.41711	0.02842
8	0.23239	−0.26807	0.39781	−0.45799

Conclusions (End of Part I)

- Consider our solution as a fine-tuning of the one in [Raghavan et al., 2012].
- Both results identical in the eigen-directions 2, 3, 4, and 5.
- Previous result is closer to the worst case, esp. along the first eigenvector (26% of variance).
- Parameterization of the geometry is not redundant (low variance concentration).
- PD could be minimized in a linear subspace $\mathbb{R}^5 \subset \mathbb{R}^8$ (e.g. using the eigen-directions 1, 3, 4, 6, and 8).

Part II. Optimization Algorithms: Synchronous Node Access

- Two parallelization methods "lie on the surface":
  - Domain decomposition.
  - Multi-point improvements [Ginsbourger et al., 2010].
- Consider EGO ($\mu = 0$, $\lambda = 1$). Let $T \sim U(t_{\min}, t_{\max})$, be the time to evaluate a cost func., $t_{\min} = 10\text{min.}$, $t_{\max} = 30\text{min.}$ If the blocking time $t_b = 2\text{min.}$, then the expected node update time will be

$$EUT \equiv E(T_u) = t_b + E(T) = 22\text{min.} \quad (1)$$

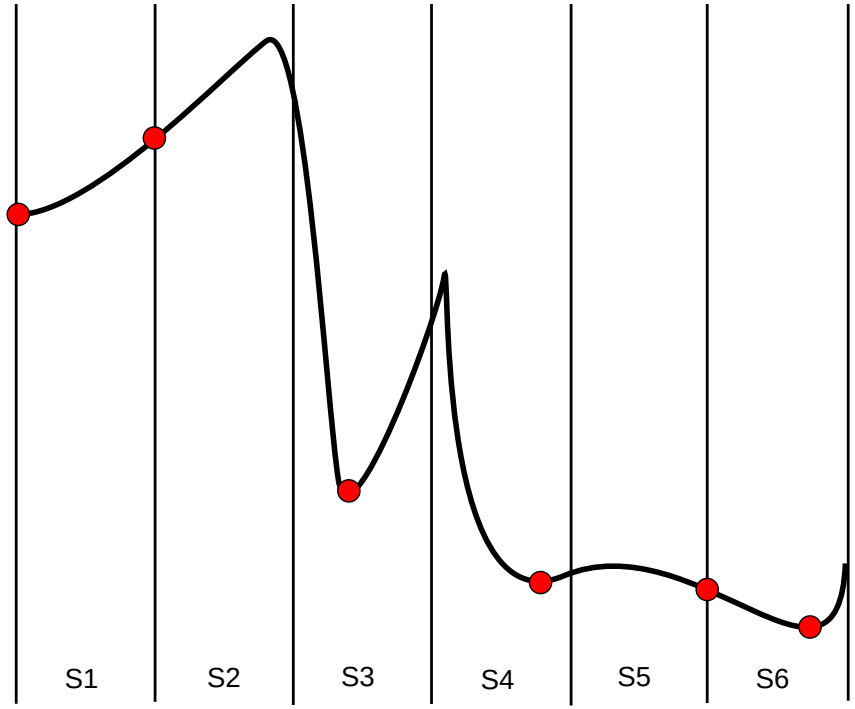
- For $\lambda = 4$ nodes with $T_i \sim U(t_{\min}, t_{\max})$:

$$EUT = t_b + E(\max\{T_1, T_2, T_3, T_4\}) \approx 28\text{min.} \quad (2)$$

- Thus, the 4-point algorithm needs to be at least $28/22 \approx 1.27$ times more efficient to save time.

Domain decomposition

Independent optimizations in each subdomain. Expl. in 1D:



Kriging-Based Optimization with Multi-Point Improvements

- The $(\mu + \lambda)$-point generalization of EGO-1998 [Ginsbourger et al., 2010, Janusevskis et al., 2012].
- Given μ busy points $\mathbf{x}_{1:\mu}$ and λ free nodes, find λ new points:

$$\max_{\mathbf{x} \in \mathbb{R}^{d\lambda}} \mathbb{E}\Big(\max\left(0, \min\left(f_{\min}, Y(\mathbf{x}_{1:\mu})\right) - \min Y(\mathbf{x})\right) | A \Big),$$

- $f_{\min}$ - current minimum,
- $Y(\mathbf{x}_{1:\mu}) = (Y(\mathbf{x}_1), \ldots, Y(\mathbf{x}_\mu))$ - random surrogates,
- $Y(\mathbf{x}) = (Y(\mathbf{x}_{\mu+1}), \ldots, Y(\mathbf{x}_{\mu+\lambda}))$ - random surrogates,
- A is the event when Y values equal to all known responses.

Implementation Details

- No ML estimation of the Gaussian kernel variances. Instead,

$$\text{kernel variance}_i = \left(\frac{|\text{upb}_i - \text{lob}_i|}{2^{1+\frac{8}{d}}} \right)^2, i = 1, \ldots, d. \quad (3)$$

- MC seed number is always initialized to the generation number.
- Cholesky decomposition is placed inside "try" and "catch".
- CMA-ES is applied to maximize the improvements.
- Box constraints are handled with bound-projection and additive penalty.
- Initial DOE is generated with Latin HyperCube Sampling.

- The quality will be assessed by using the normalized real improvement (NRI) defined as

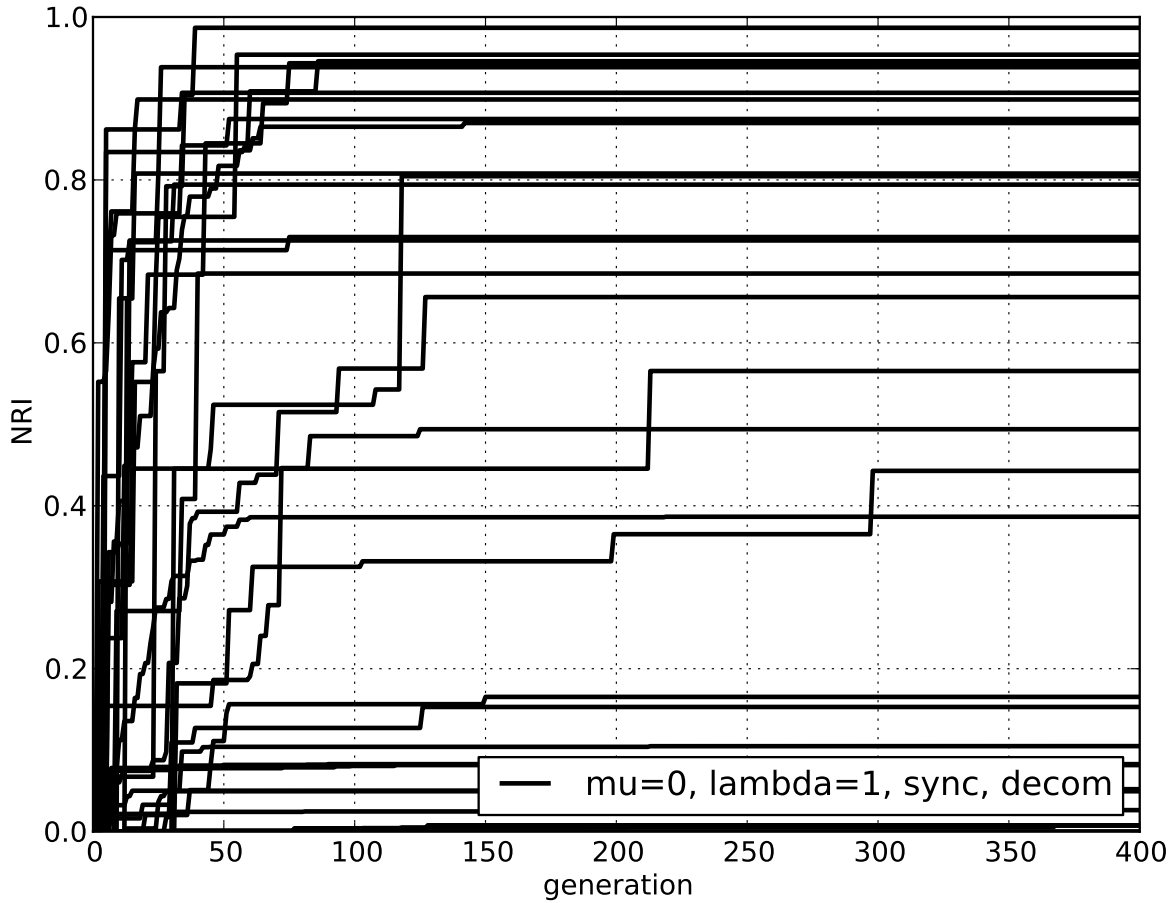
$$\mathrm{NRI}(\mathrm{generation}) = \frac{f_0 - f_{\min}(\mathrm{generation})}{f_0 - f_{\mathrm{true}}}. \qquad (4)$$

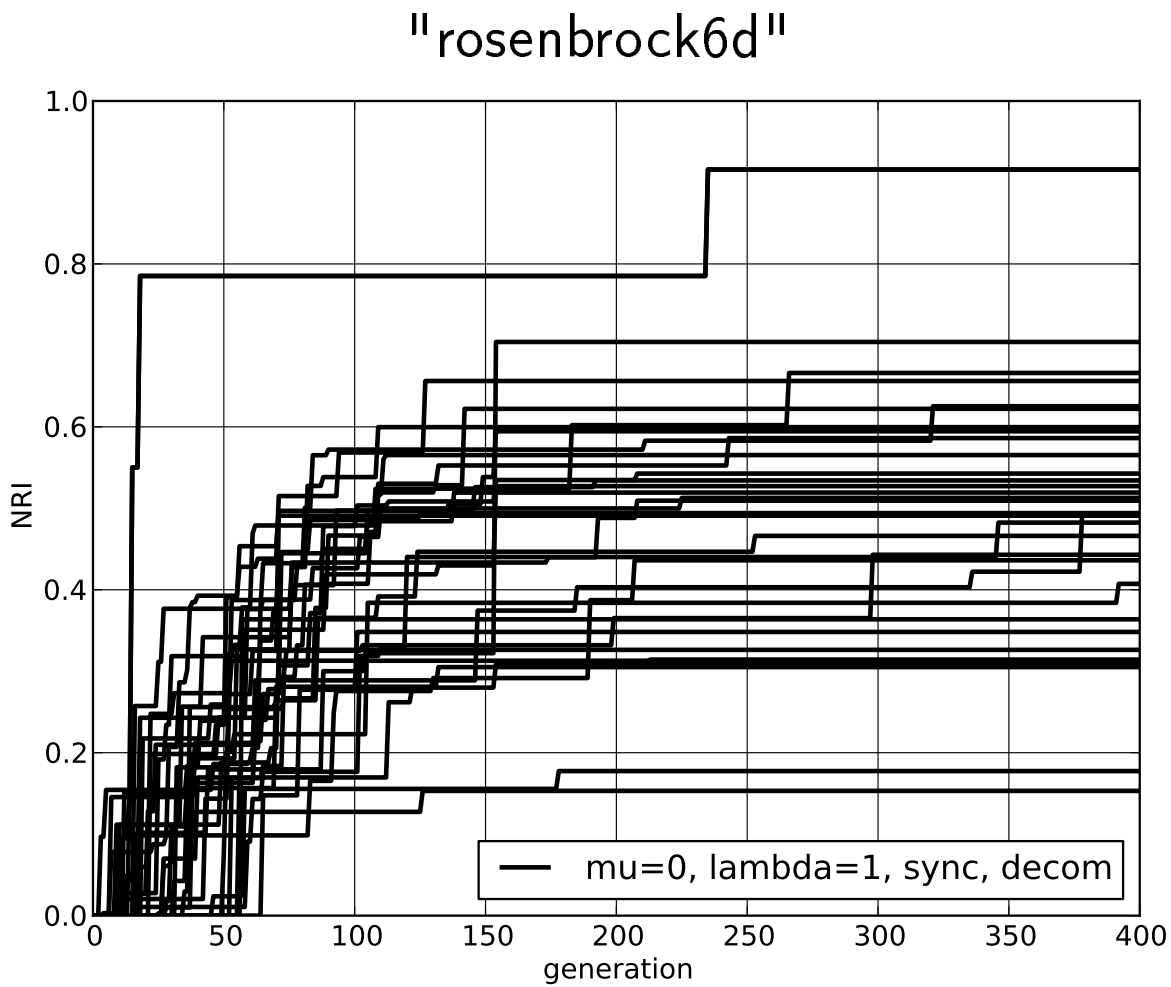
- 100 independent optimization runs averaged (over different initial DOEs).
- f_0 is the smallest value of the cost function achieved on the initial DOE.
- $f_{\min}$ denotes the value achieved after a particular generation of points is evaluated.
- f_{true} is the true ideal minimal value.

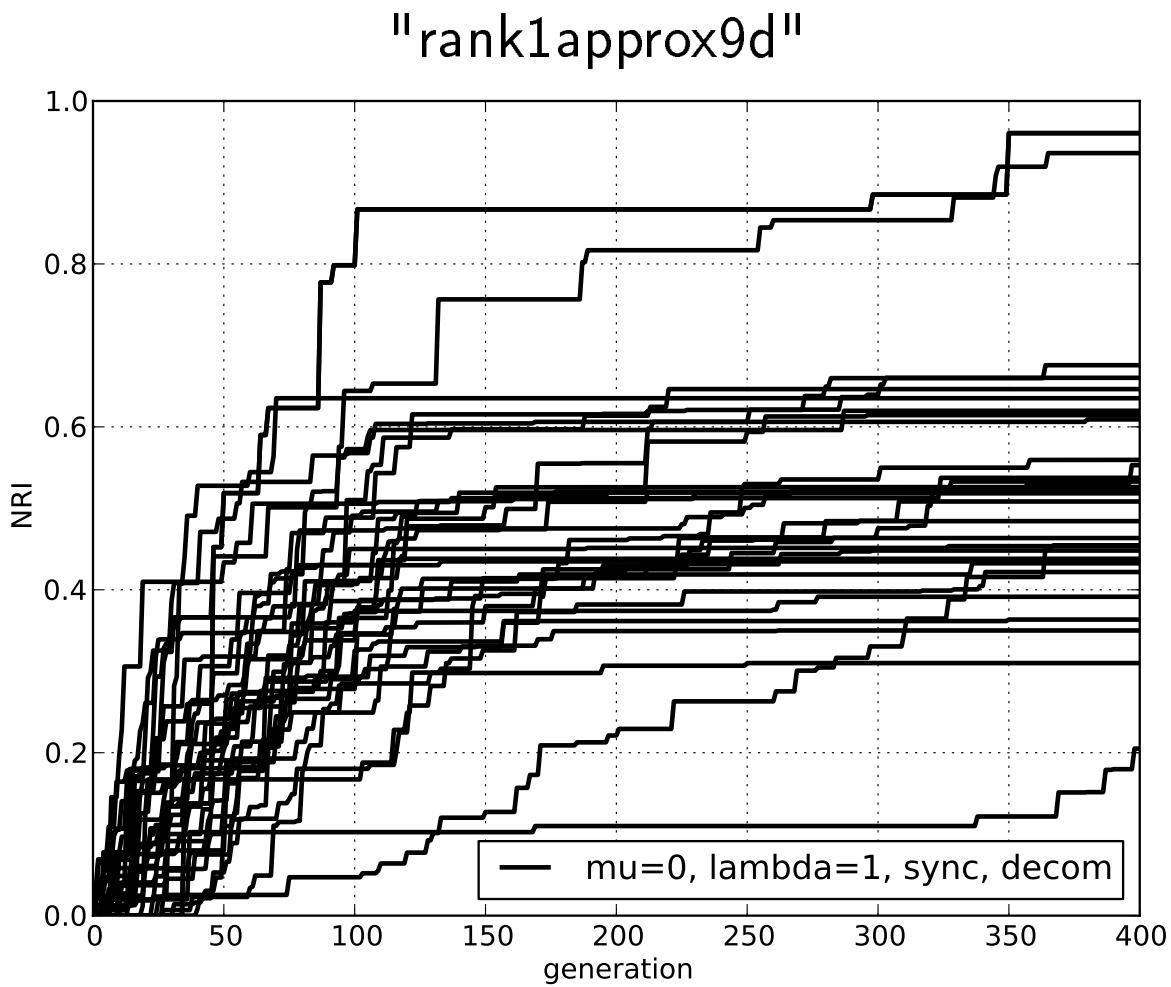
Table: Optimization Criteria

Label	Cost function	Domain
"michalewicz2d"	$\sum_{i=1}^{2} \sin(x_i) \sin^2(ix_i^2/\pi)$	$[0,5]^2$
"rosenbrock6d"	$\sum_{i=1}^{5} 100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2$	$[0,5]^6$
"rank1approx9d"	$\|\mathbf{A}_{4\times5} - \mathbf{x}_{4\times1}\mathbf{y}_{1\times5}\|_2$, $a_{ij} \sim U(0,1)$	$[-1,1]^9$

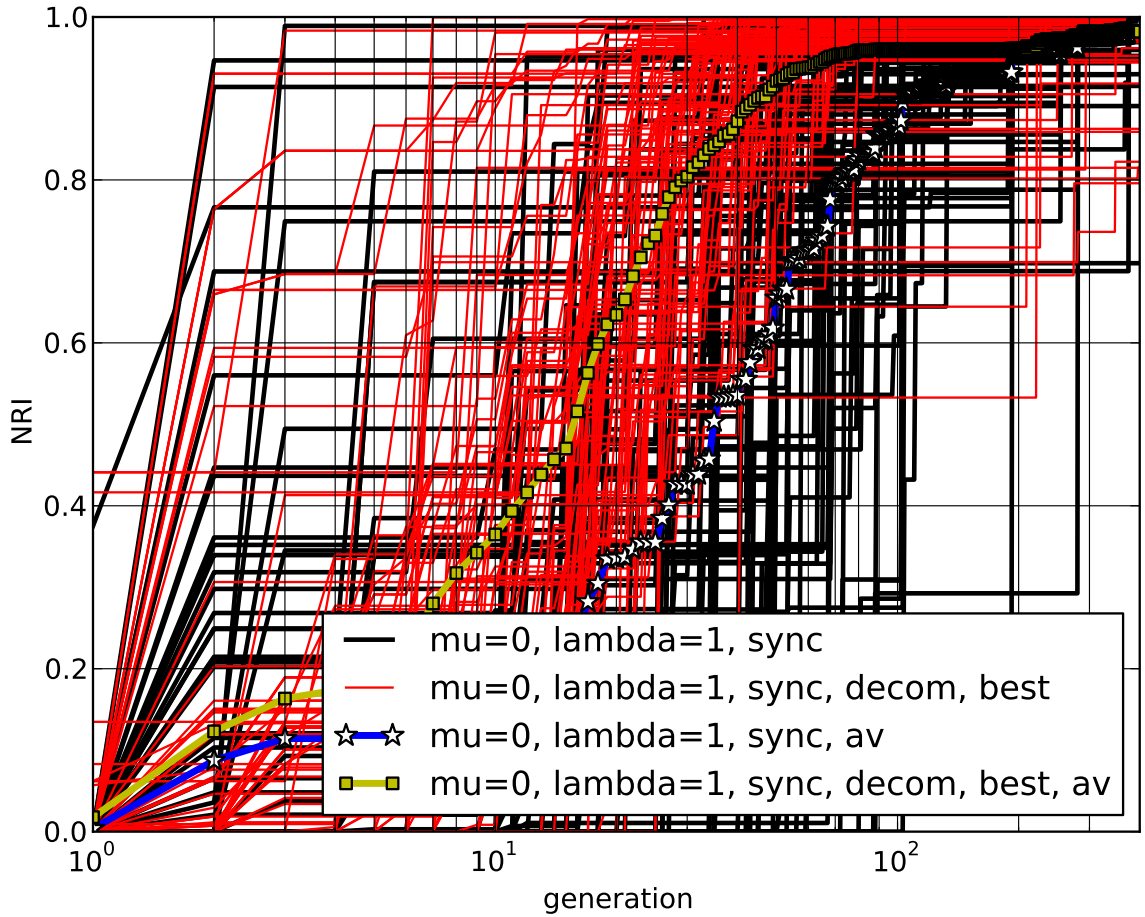
Domain decomposition, 1 curve = 1 subdomain, 32 subdomains "michalewicz2d"

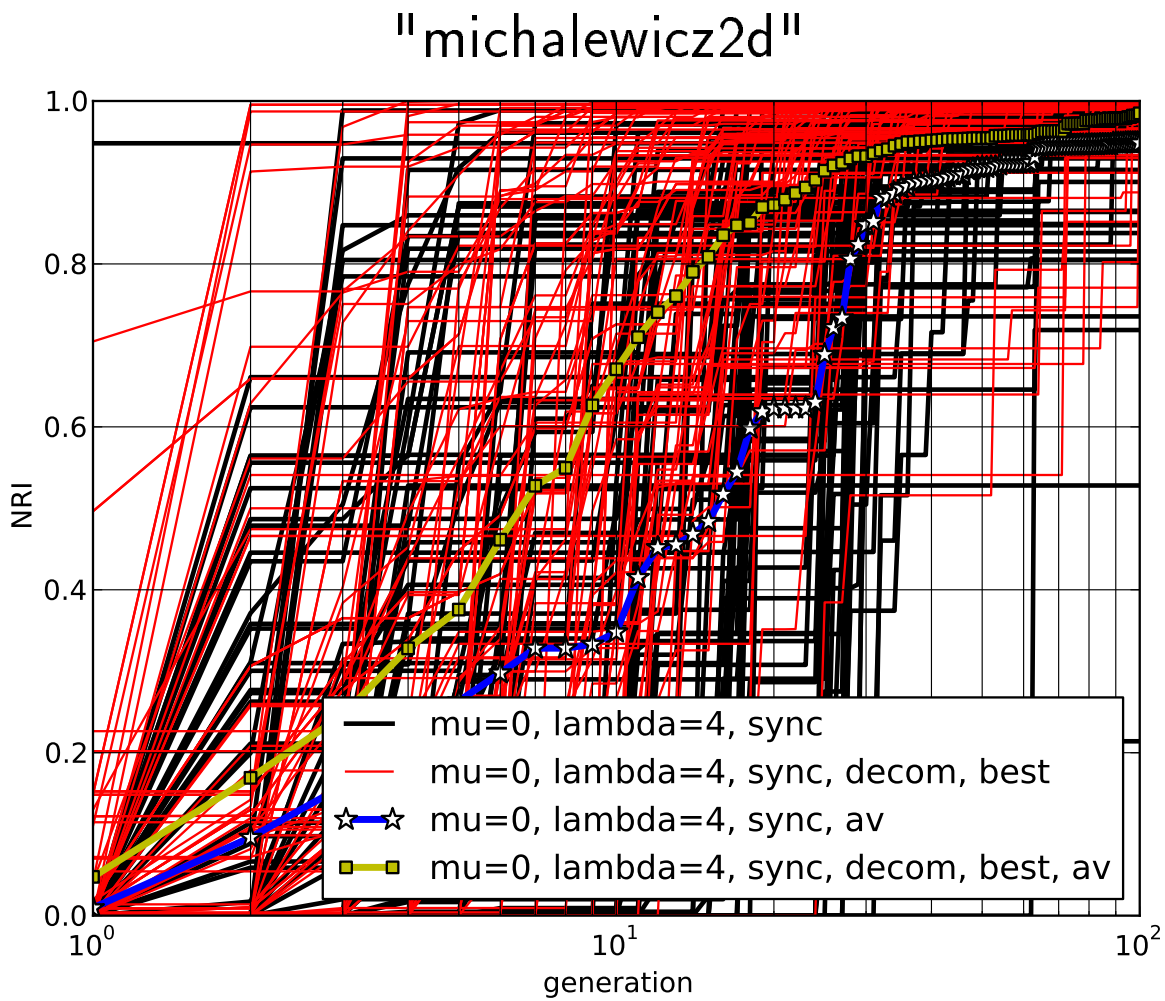


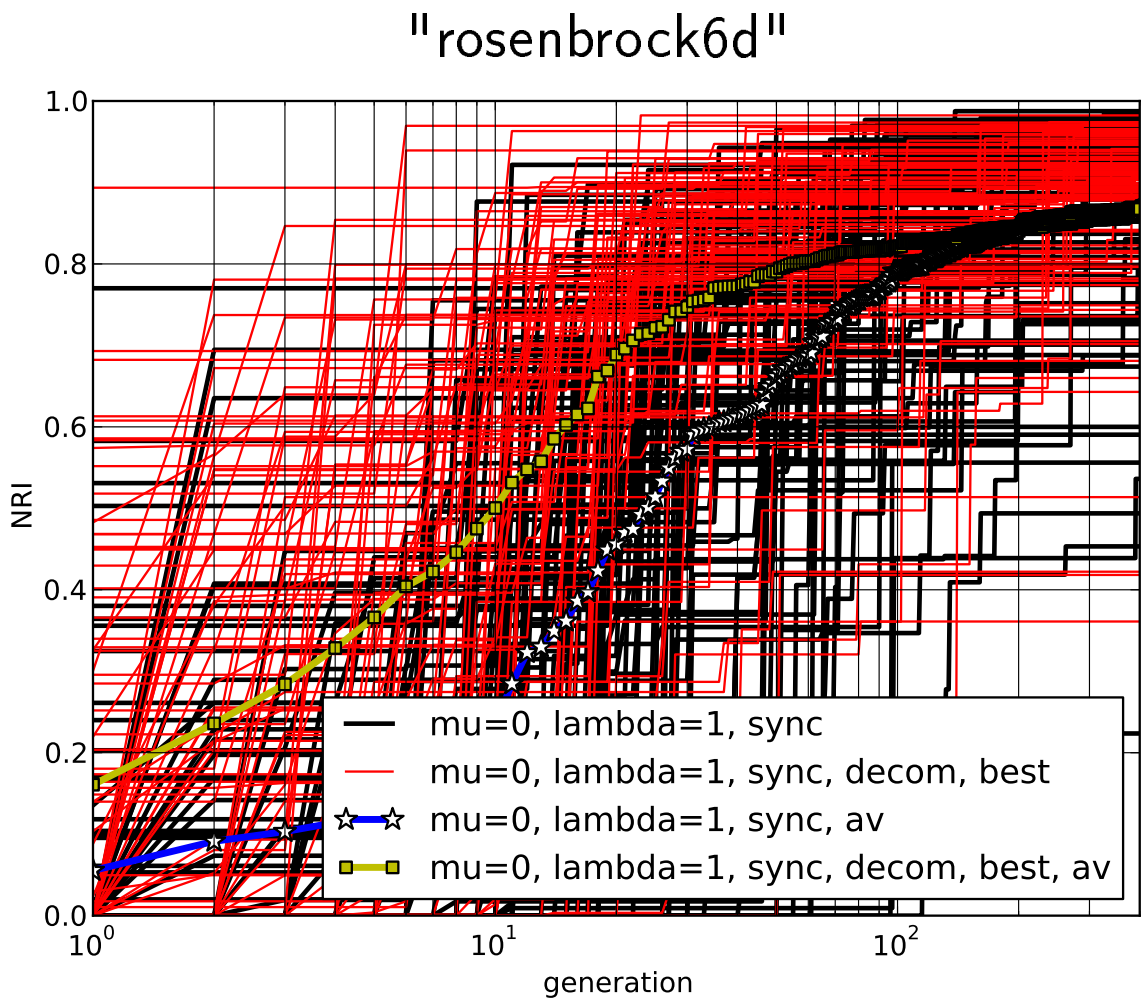

"rosenbrock6d"
NRI
generation
mu=0, lambda=1, sync, decom
0.0
0.2
0.4
0.6
0.8
1.0
0
50
100
150
200
250
300
350
400

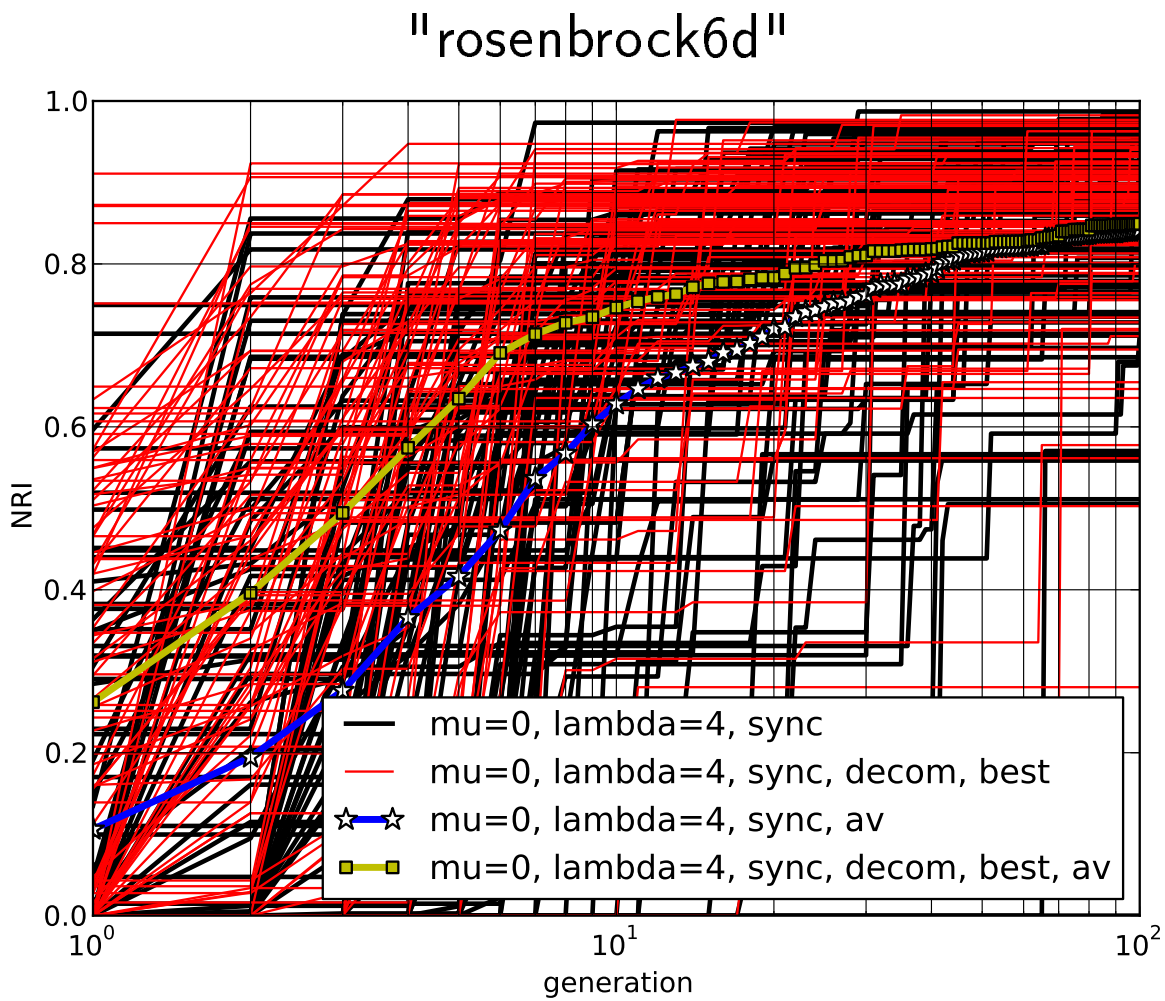

"rank1approx9d"
NRI
generation
mu=0, lambda=1, sync, decom
0.0
0.2
0.4
0.6
0.8
1.0
0
50
100
150
200
250
300
350
400

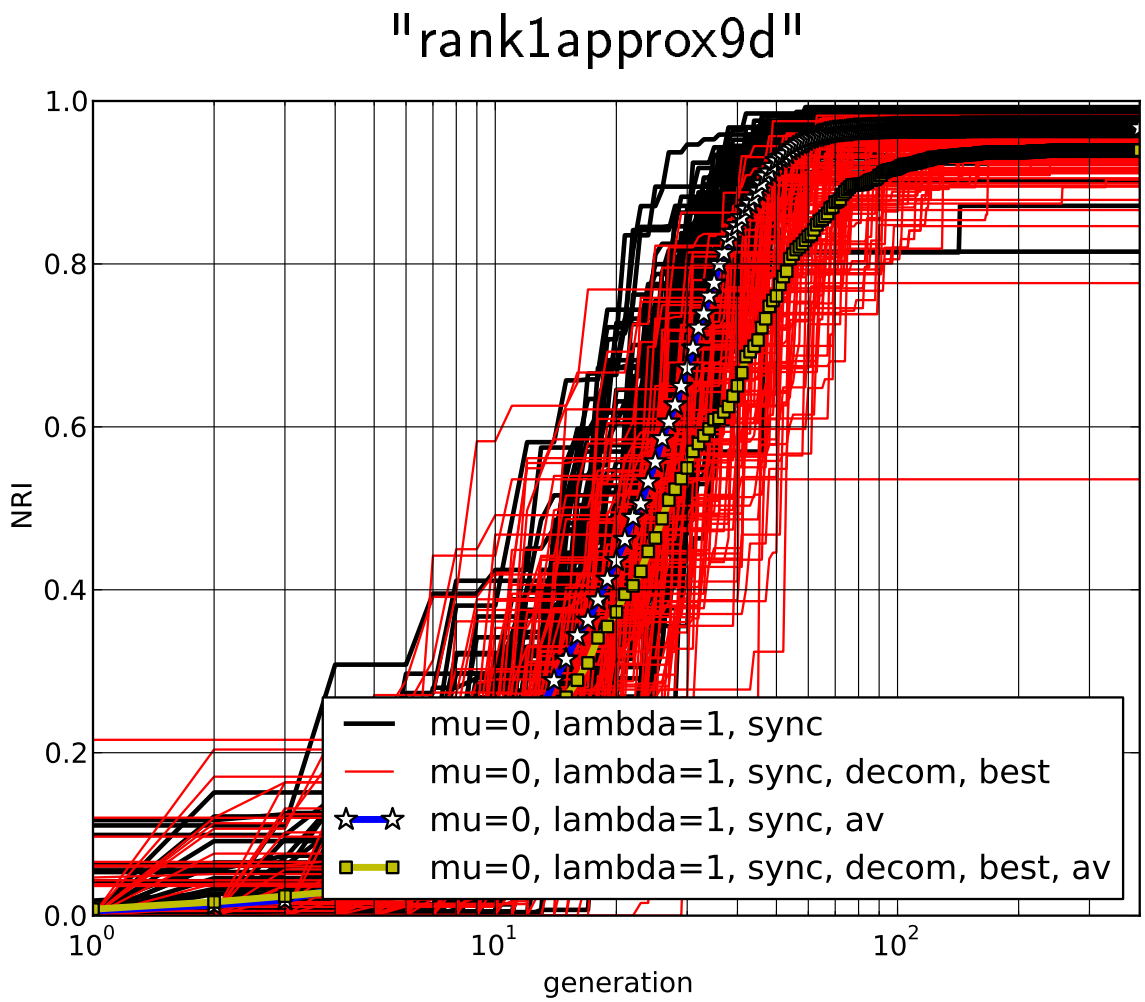
1 curve = 1 initial DOE (for decom – in the best sub-domain) "michalewicz2d"

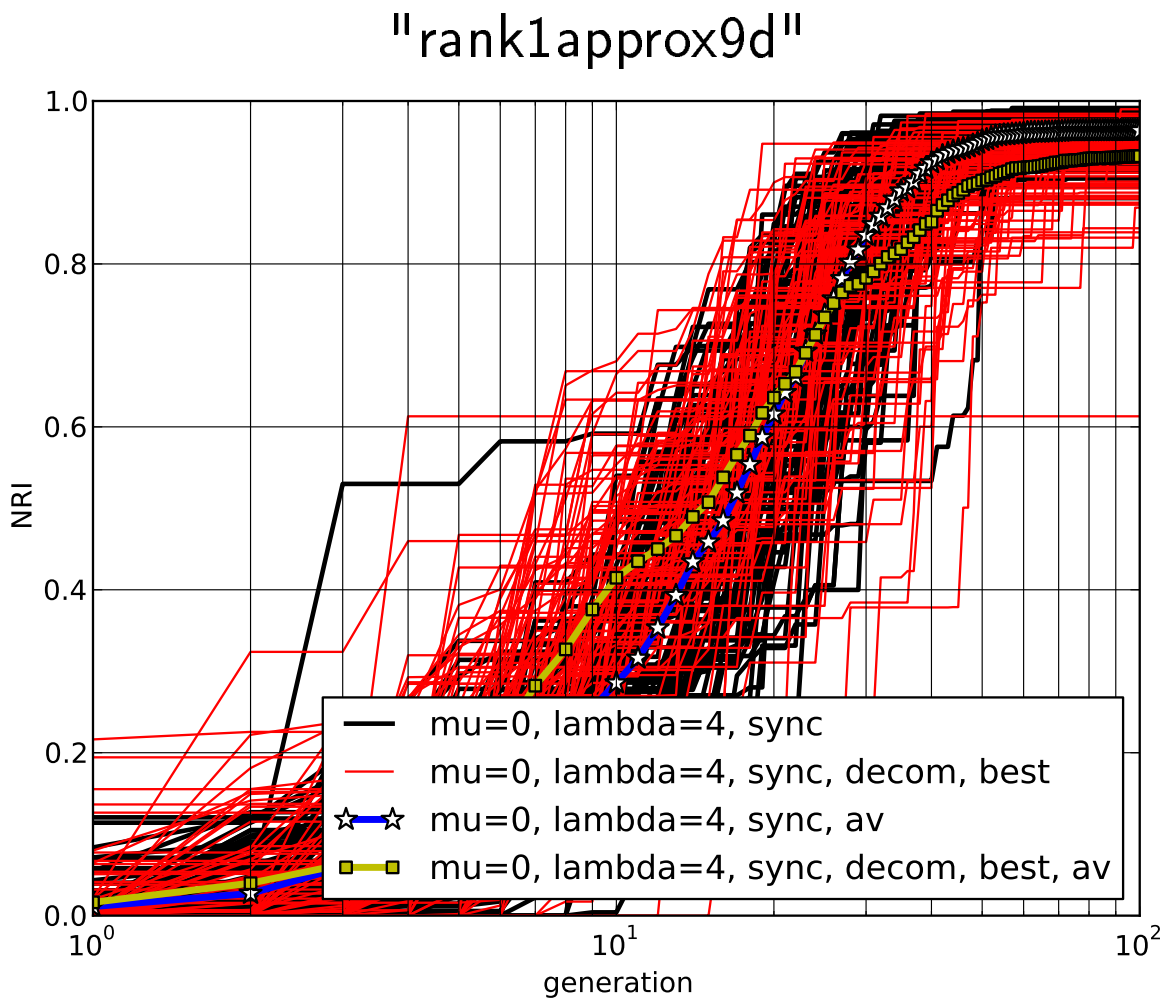


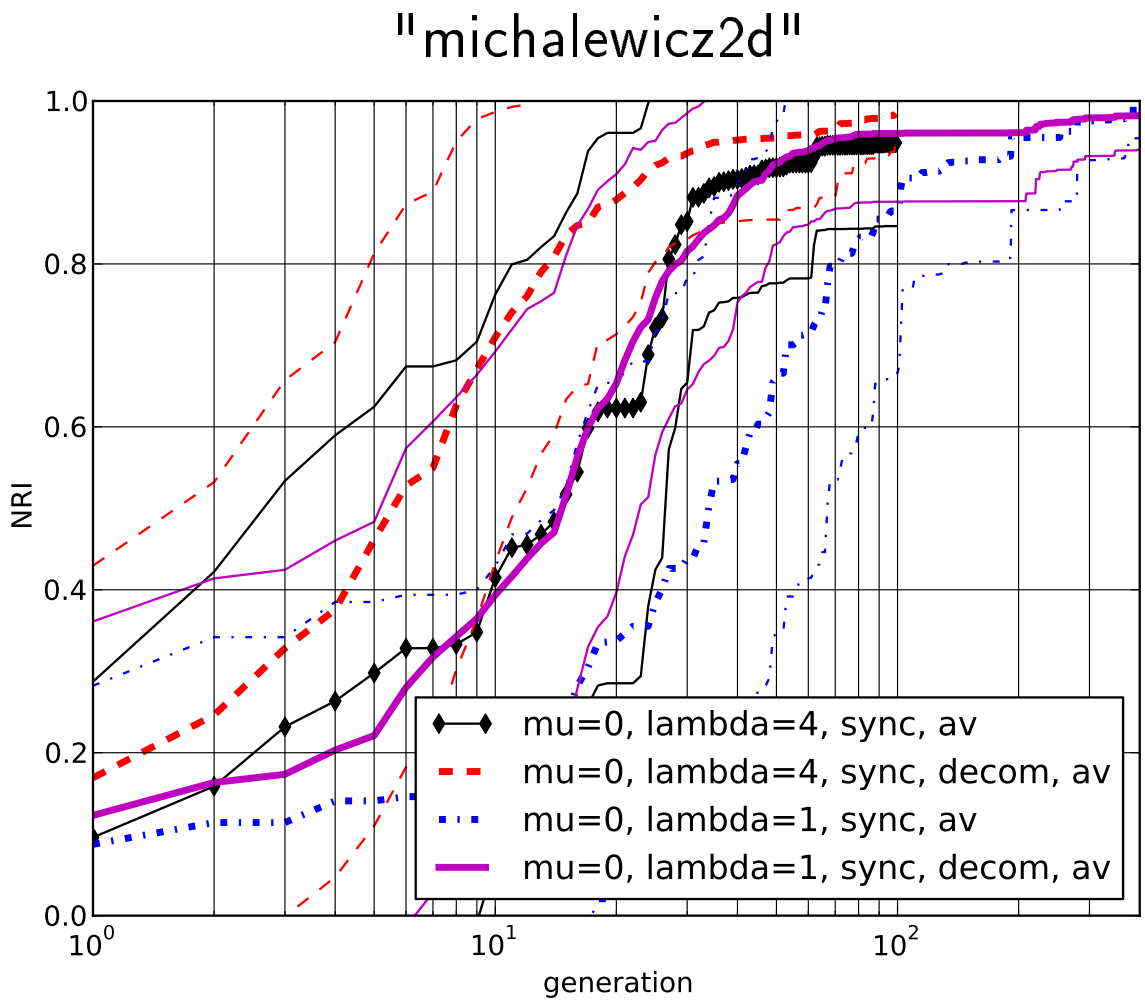

"michalewicz2d"
NRI
generation
1.0
0.8
0.6
0.4
0.2
0.0
10^0
10^1
10^2
mu=0, lambda=4, sync
mu=0, lambda=4, sync, decom, best
mu=0, lambda=4, sync, av
mu=0, lambda=4, sync, decom, best, av

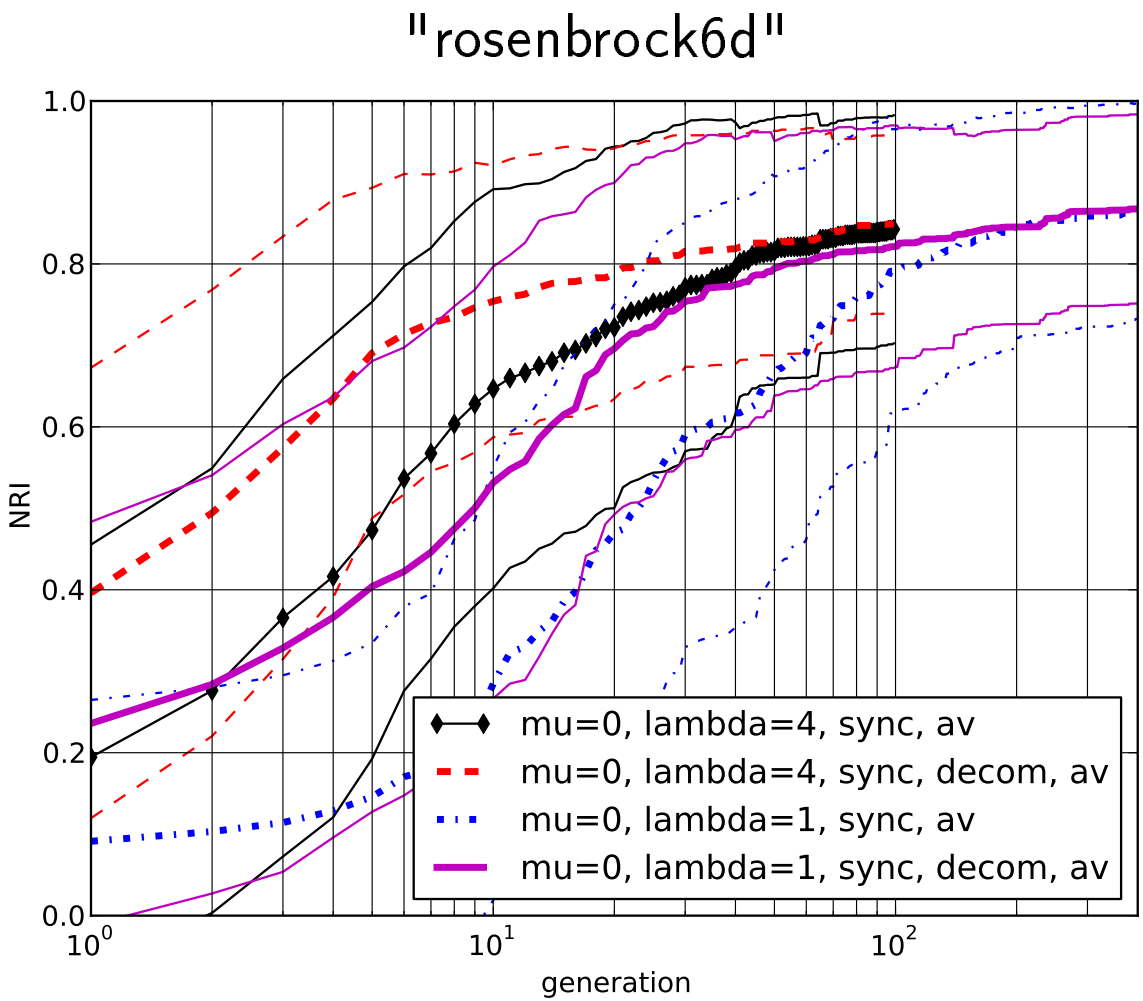

"rosenbrock6d"
NRI
generation
1.0
0.8
0.6
0.4
0.2
0.0
10^0
10^1
10^2
mu=0, lambda=1, sync
mu=0, lambda=1, sync, decom, best
mu=0, lambda=1, sync, av
mu=0, lambda=1, sync, decom, best, av

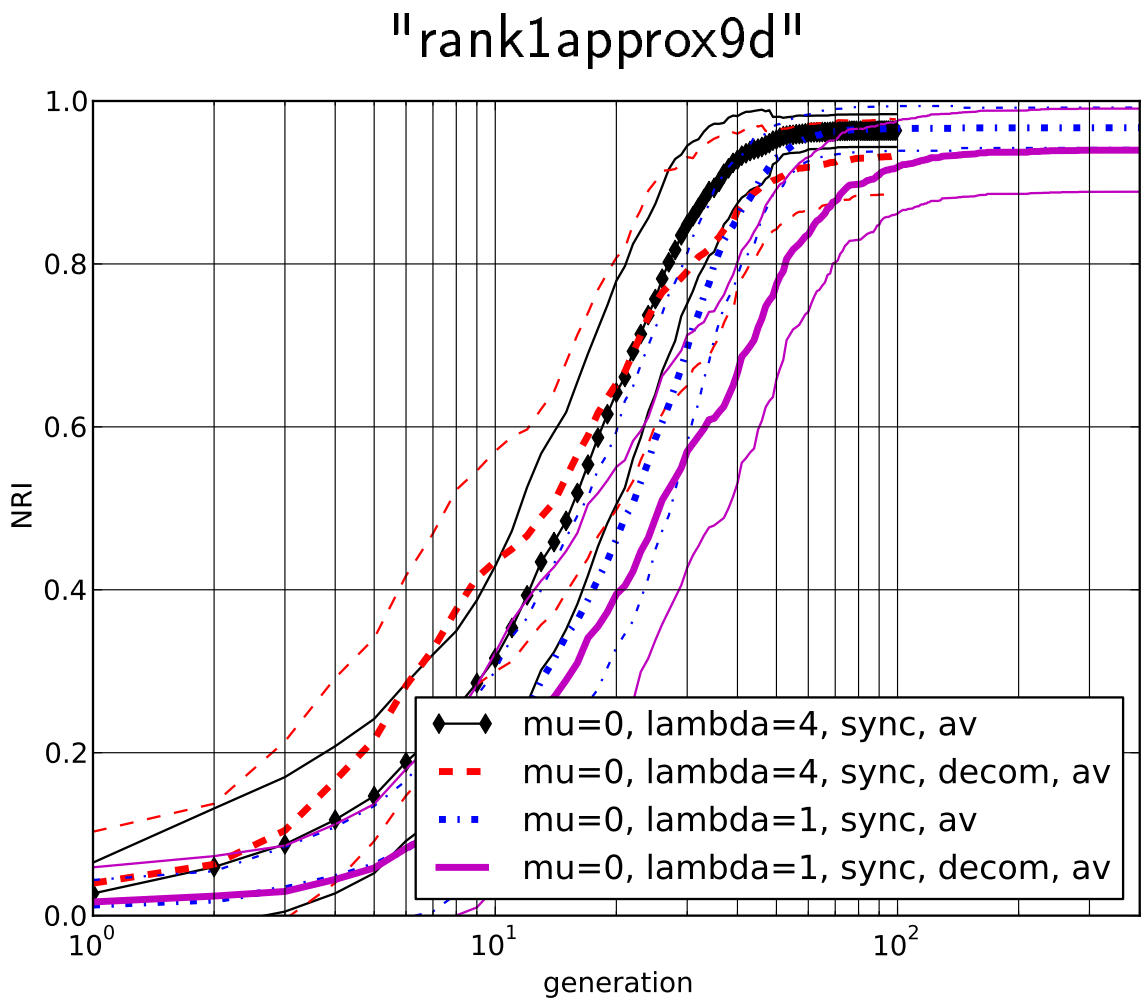

"rosenbrock6d"
NRI
generation
1.0
0.8
0.6
0.4
0.2
0.0
10^0
10^1
10^2
mu=0, lambda=4, sync
mu=0, lambda=4, sync, decom, best
mu=0, lambda=4, sync, av
mu=0, lambda=4, sync, decom, best, av


"rank1approx9d"
NRI
generation
1.0
0.8
0.6
0.4
0.2
0.0
10^0
10^1
10^2
mu=0, lambda=1, sync
mu=0, lambda=1, sync, decom, best
mu=0, lambda=1, sync, av
mu=0, lambda=1, sync, decom, best, av


"rank1approx9d"
NRI
generation
1.0
0.8
0.6
0.4
0.2
0.0
10^0
10^1
10^2
mu=0, lambda=4, sync
mu=0, lambda=4, sync, decom, best
mu=0, lambda=4, sync, av
mu=0, lambda=4, sync, decom, best, av


"michalewicz2d"
NRI
generation
1.0
0.8
0.6
0.4
0.2
0.0
10^0
10^1
10^2
mu=0, lambda=4, sync, av
mu=0, lambda=4, sync, decom, av
mu=0, lambda=1, sync, av
mu=0, lambda=1, sync, decom, av


"rosenbrock6d"
NRI
generation
1.0
0.8
0.6
0.4
0.2
0.0
10^0
10^1
10^2
mu=0, lambda=4, sync, av
mu=0, lambda=4, sync, decom, av
mu=0, lambda=1, sync, av
mu=0, lambda=1, sync, decom, av


"rank1approx9d"
NRI
generation
1.0
0.8
0.6
0.4
0.2
0.0
10^0
10^1
10^2
mu=0, lambda=4, sync, av
mu=0, lambda=4, sync, decom, av
mu=0, lambda=1, sync, av
mu=0, lambda=1, sync, decom, av

Conclusion (End of Part II)

- Domain decomposition:
  - can be better in low-dimensional problems,
  - but when $d = 9$, it becomes myopic.
  - Suitable for an initial assessment of multimodality.
- Define *speed-up* as

$$S(NRI) \equiv \frac{\text{time to reach NRI by EI}^{0,1}}{\text{time to reach NRI by EI}^{0,\lambda}}. \tag{5}$$

- Multi-point improvements ($\lambda = 4$):
  - $S_{\max} = 2.5$ when $d = 2, 6$, and $S_{\max} = 1.4$ when $d = 9$.
  - When NRI>0.9, it is not clear which λ value is better.

Part III. Optimization Algorithms: Asynchronous Node Access

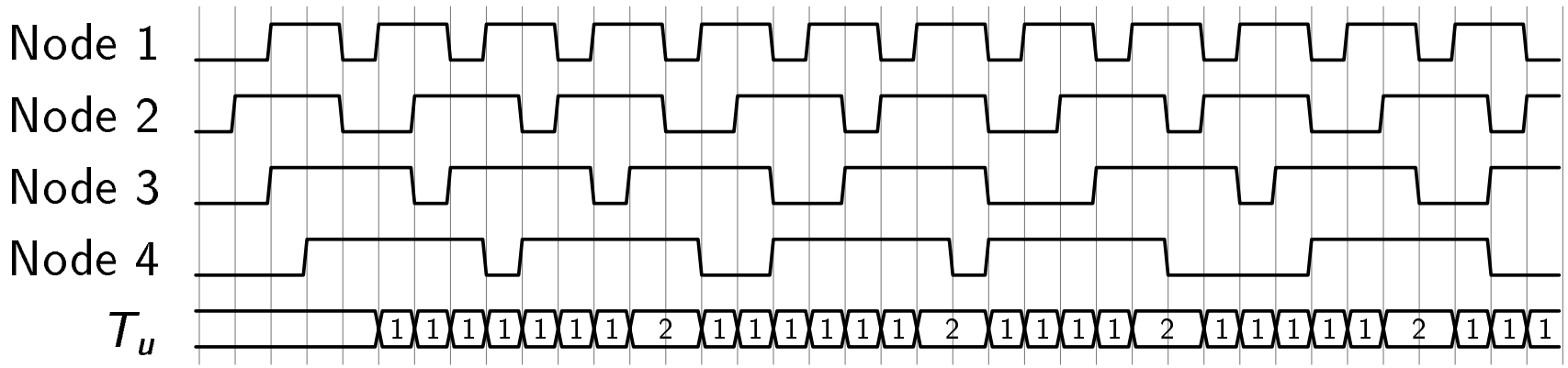
- There are more nodes available ($\mu + \lambda \approx \mathcal{O}(100)$) than the highest dimension in which we can maximize EI ($d \times \lambda$).
- Main idea:
  - Send $m = \mu + \lambda$ points for the evaluation,
  - and keep updating λ points as they become available.

Simple Demonstration

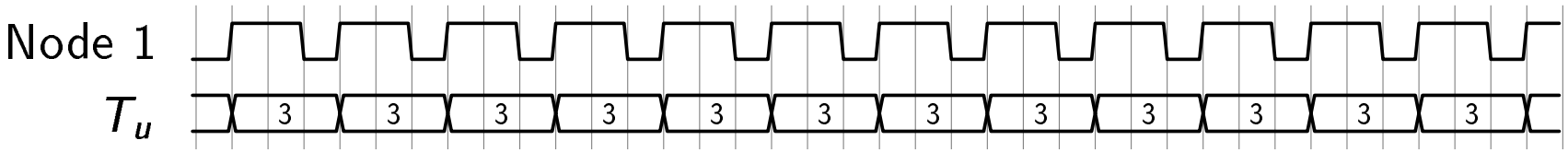
An example to gain an intuition about $EUT = E(T_u)$:

1. The falling front indicates that the node becomes available.
2. It takes one time unit to update the node.
3. When more than one node is available at the access time, the faster node is preferred.

Asynchronous mode, $\mu = 3, \lambda = 1$



Synchronous mode, $\mu = 0, \lambda = 1$



Synchronous mode, $\mu = 0, \lambda = 4$

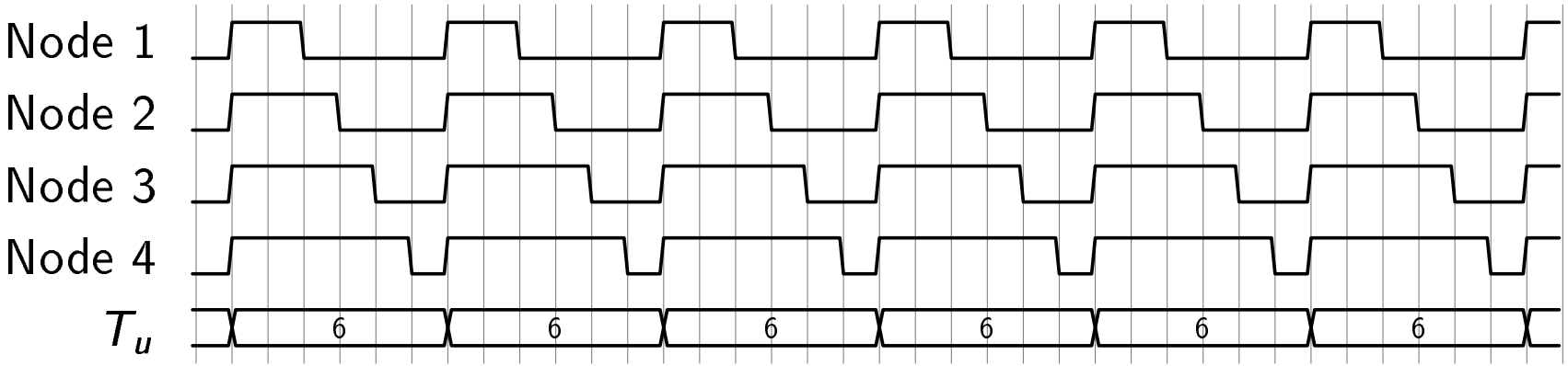
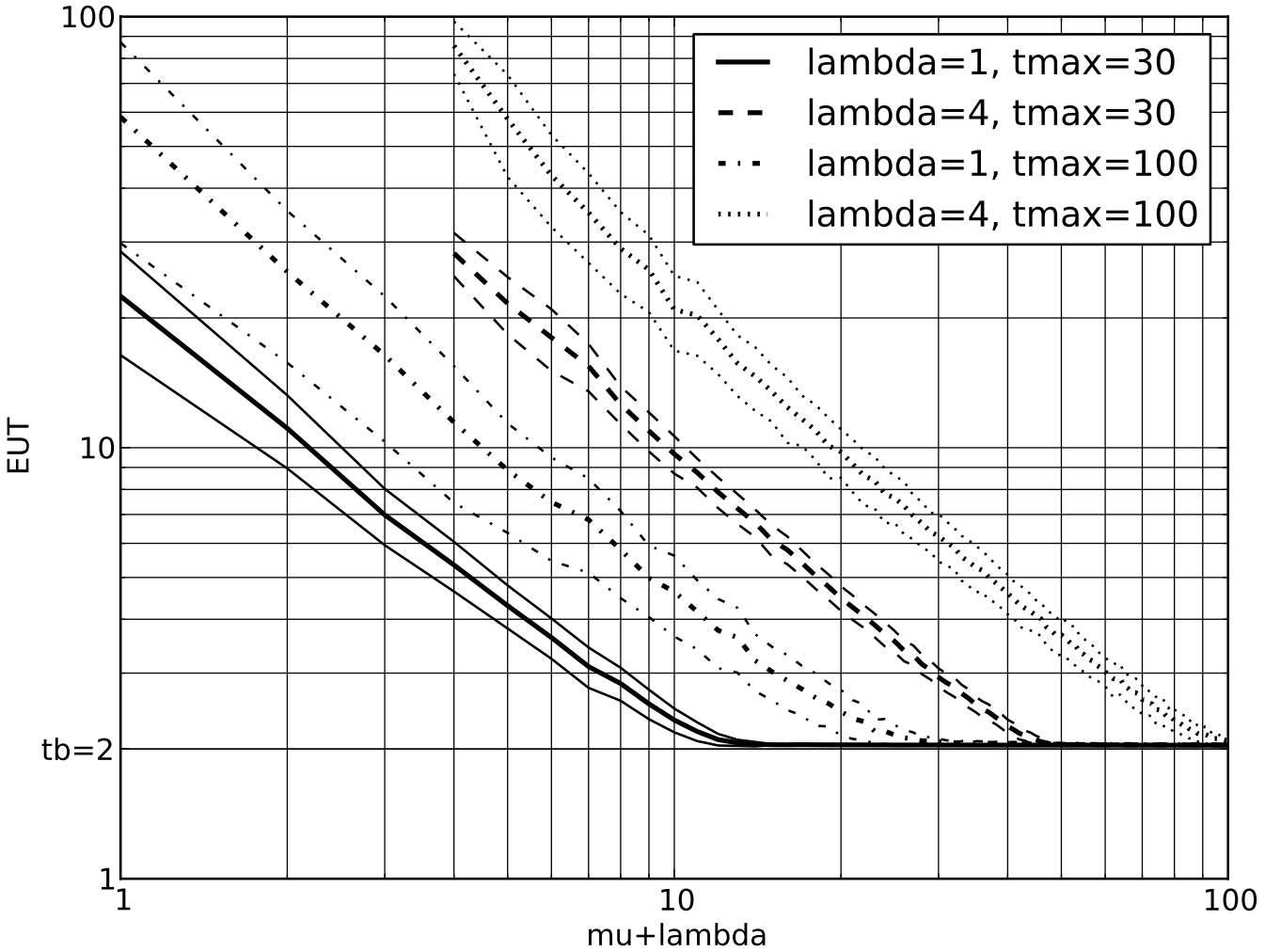


Table: Average node update time for different algorithms. Parameters: $t_{\min} = 10$, $t_{\max} = 30$, $t_b = 2$. Averaging is performed with $25 \cdot 10^4$ points.

Asynchronous	m	λ	EUT	Deviation
True	32	1	2.04	0.0024
True	32	4	2.77	0.13
False	1	1	22.0	5.77
False	4	4	28.0	3.27

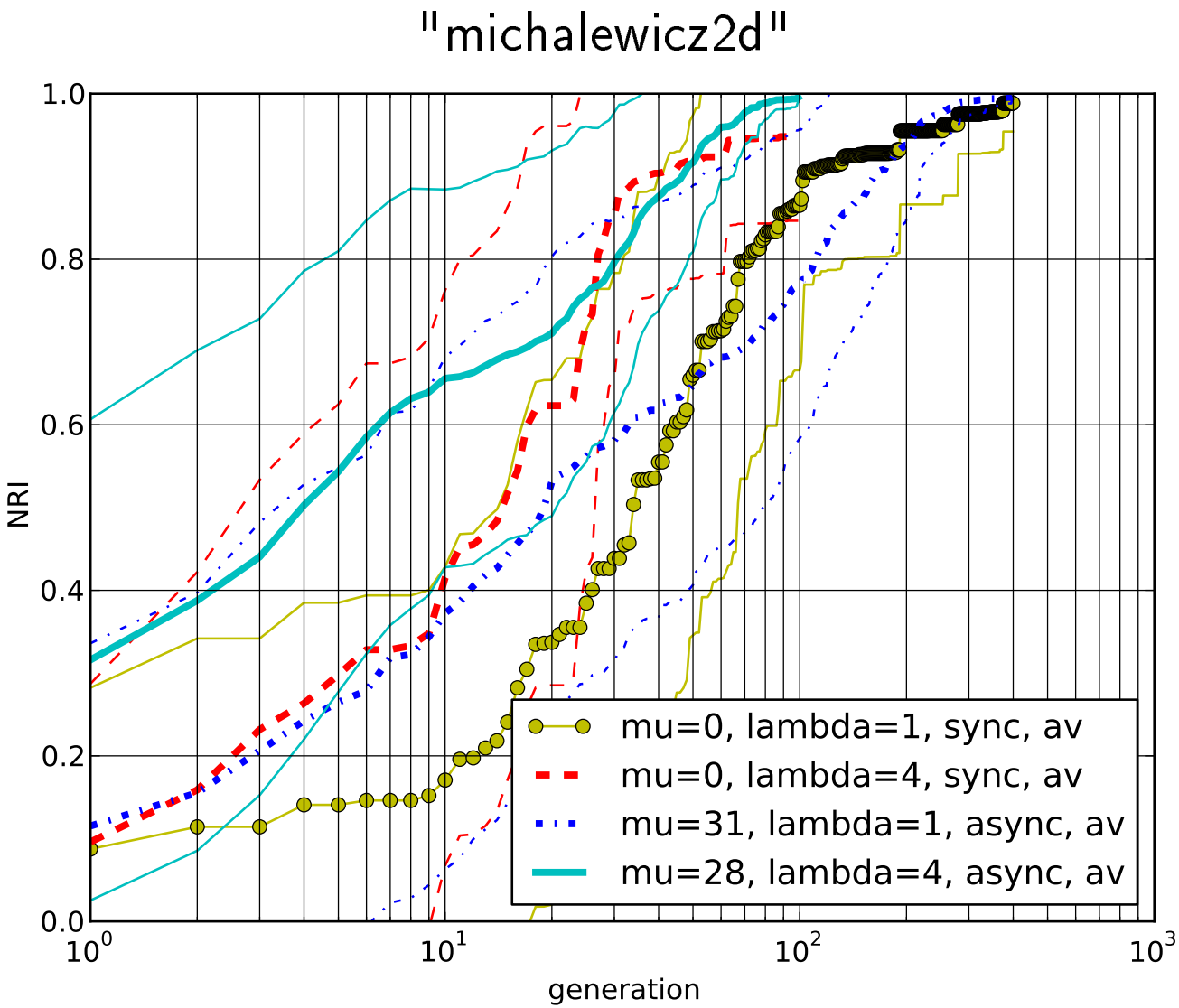
$t_{\min} = 10$

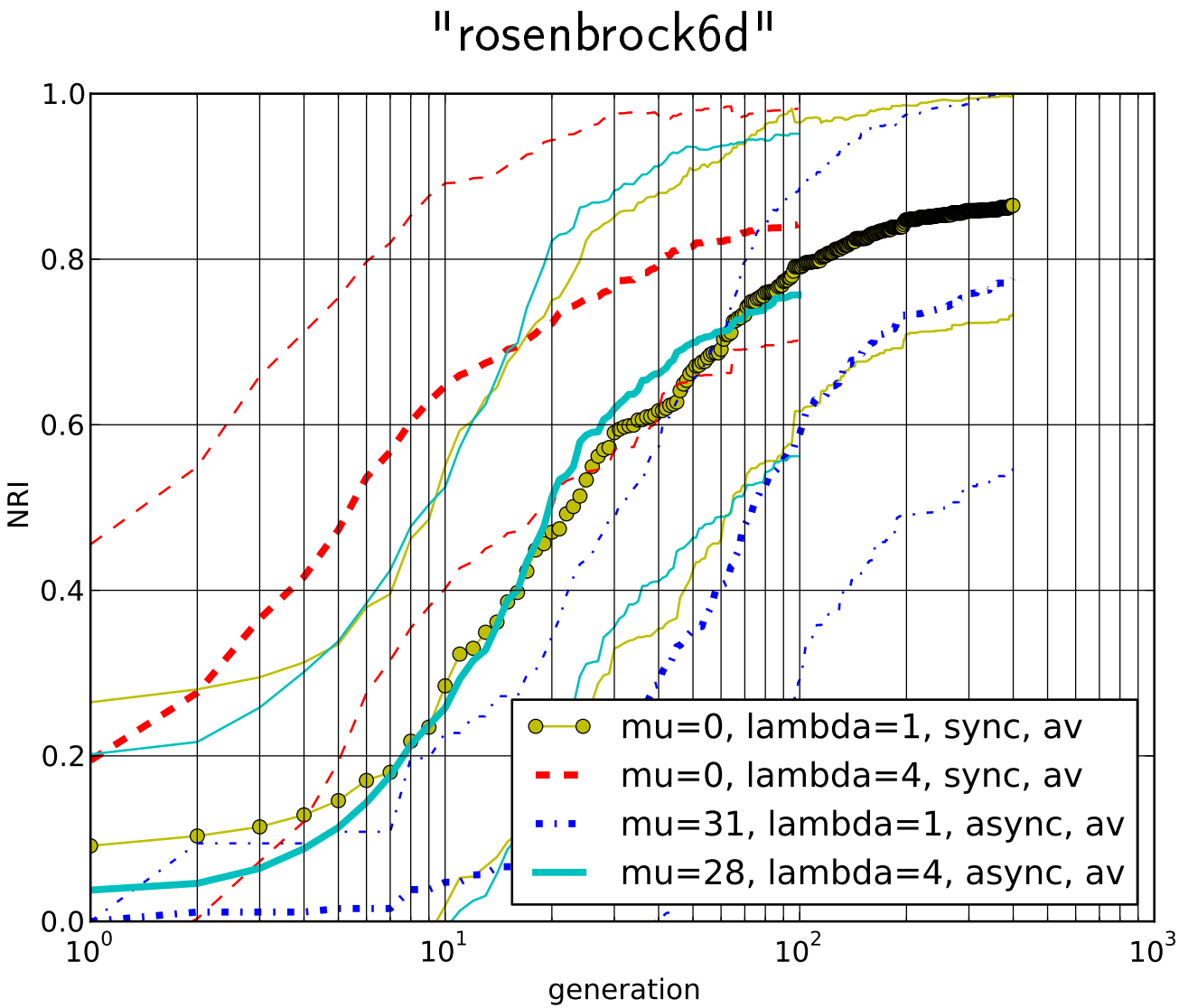

lambda=1, tmax=30
lambda=4, tmax=30
lambda=1, tmax=100
lambda=4, tmax=100
EUT
100
10
tb=2
1
1
10
100
mu+lambda

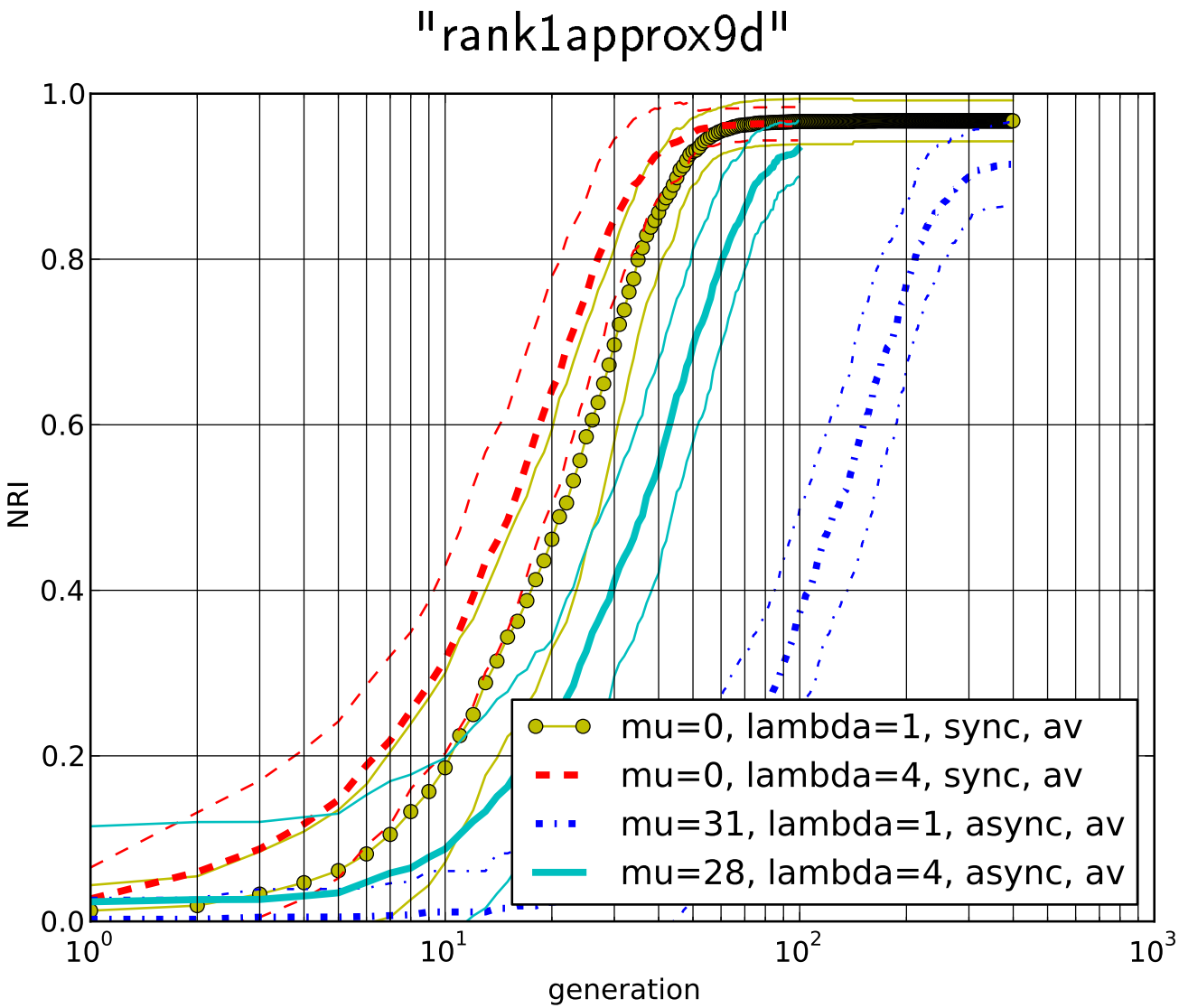
- The EUT values decrease roughly as $\mathcal{O}(m^{-1})$.
- A more precise rule that fits our data is $\mathcal{O}(m^{-1-\alpha})$, where

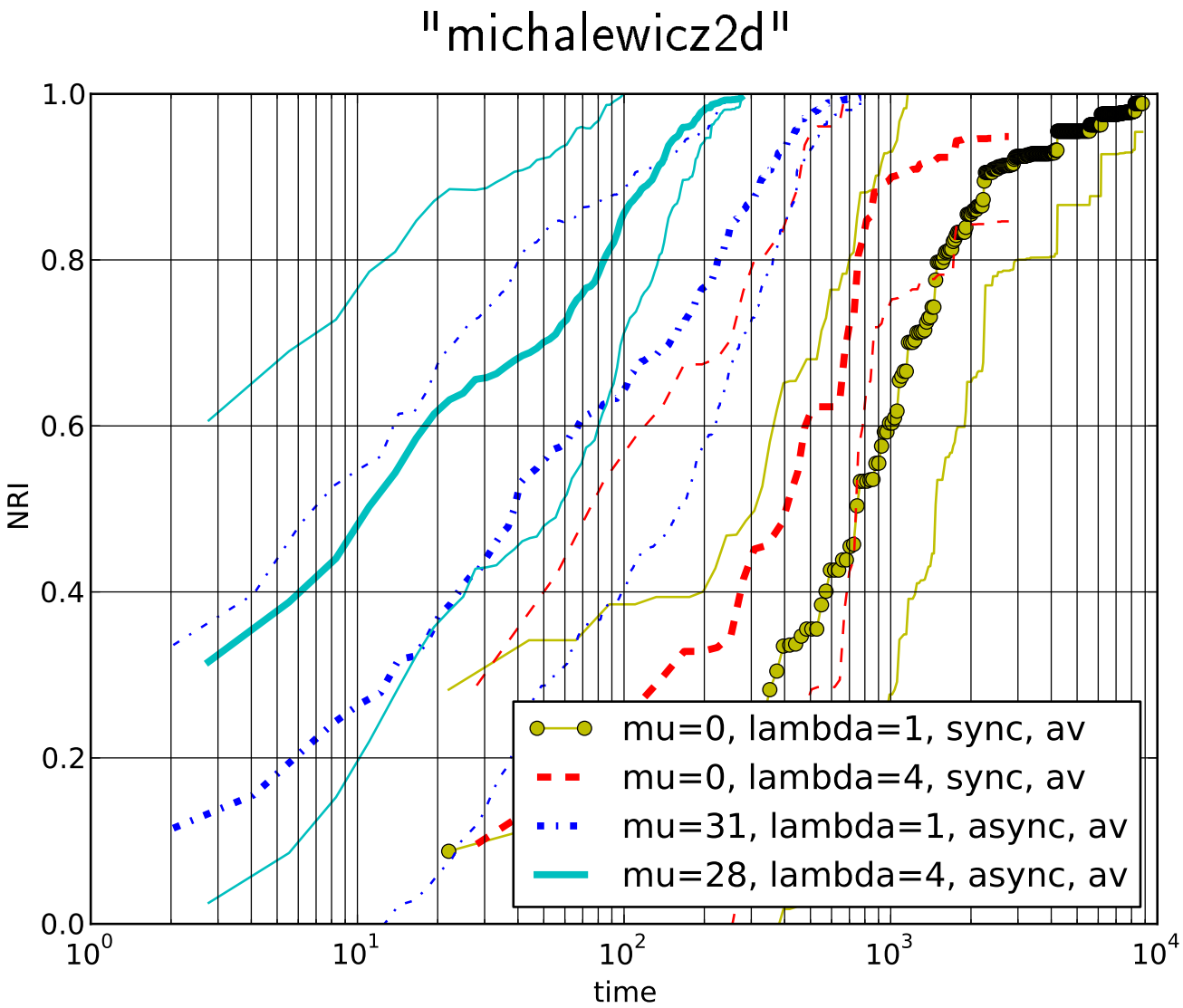
$$\alpha \approx \frac{t_b}{3t_{\min}}(\lambda - 1). \tag{6}$$

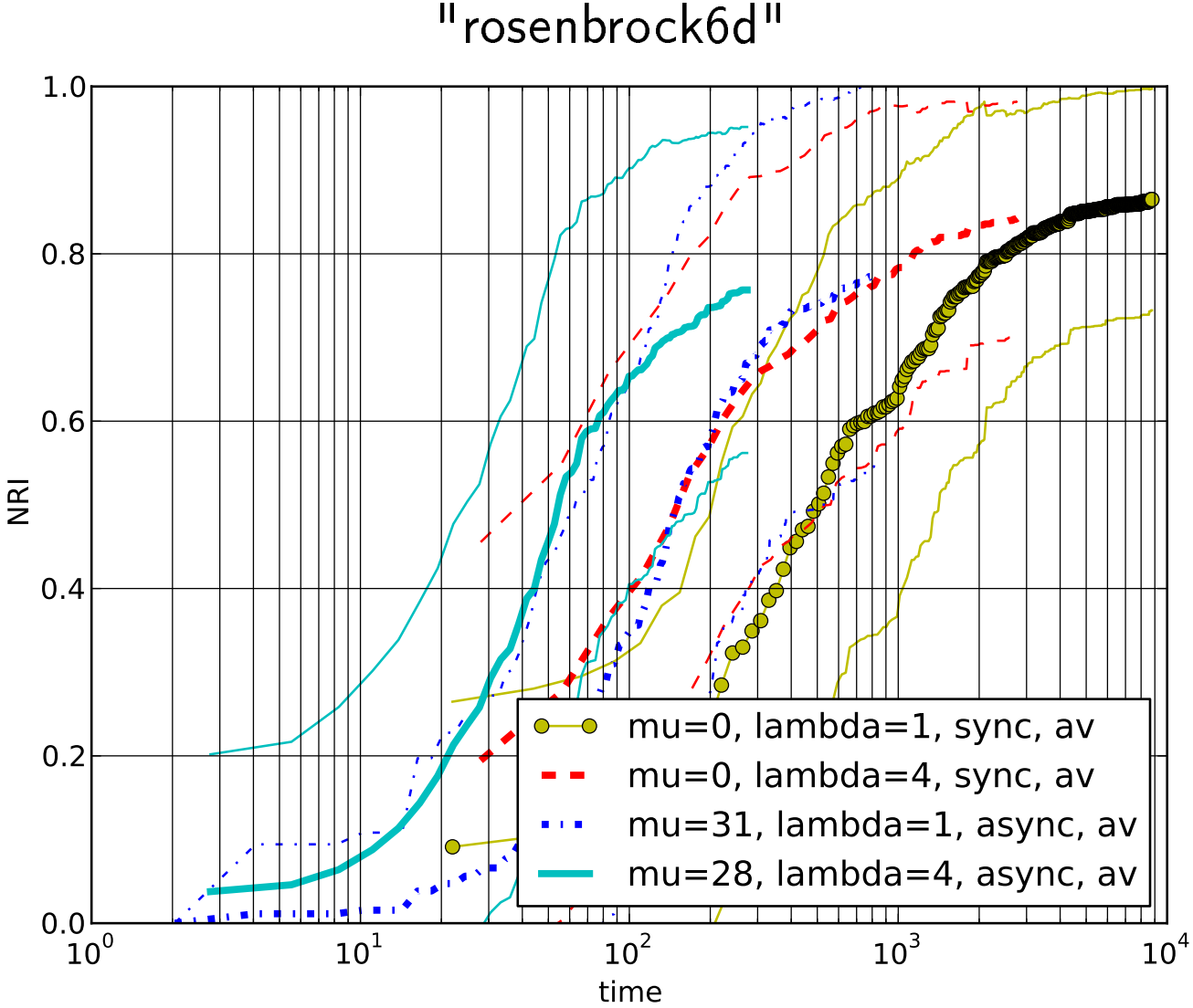
  Notice that $t_{\max}$ is not present in the equation.
- If m is large enough, EUT reduces to the blocking time it takes to generate and send new points. So what exactly is optimization of an expensive-to-evaluate function? The practical function evaluation time is a function of t_{min}, m, and t_b.

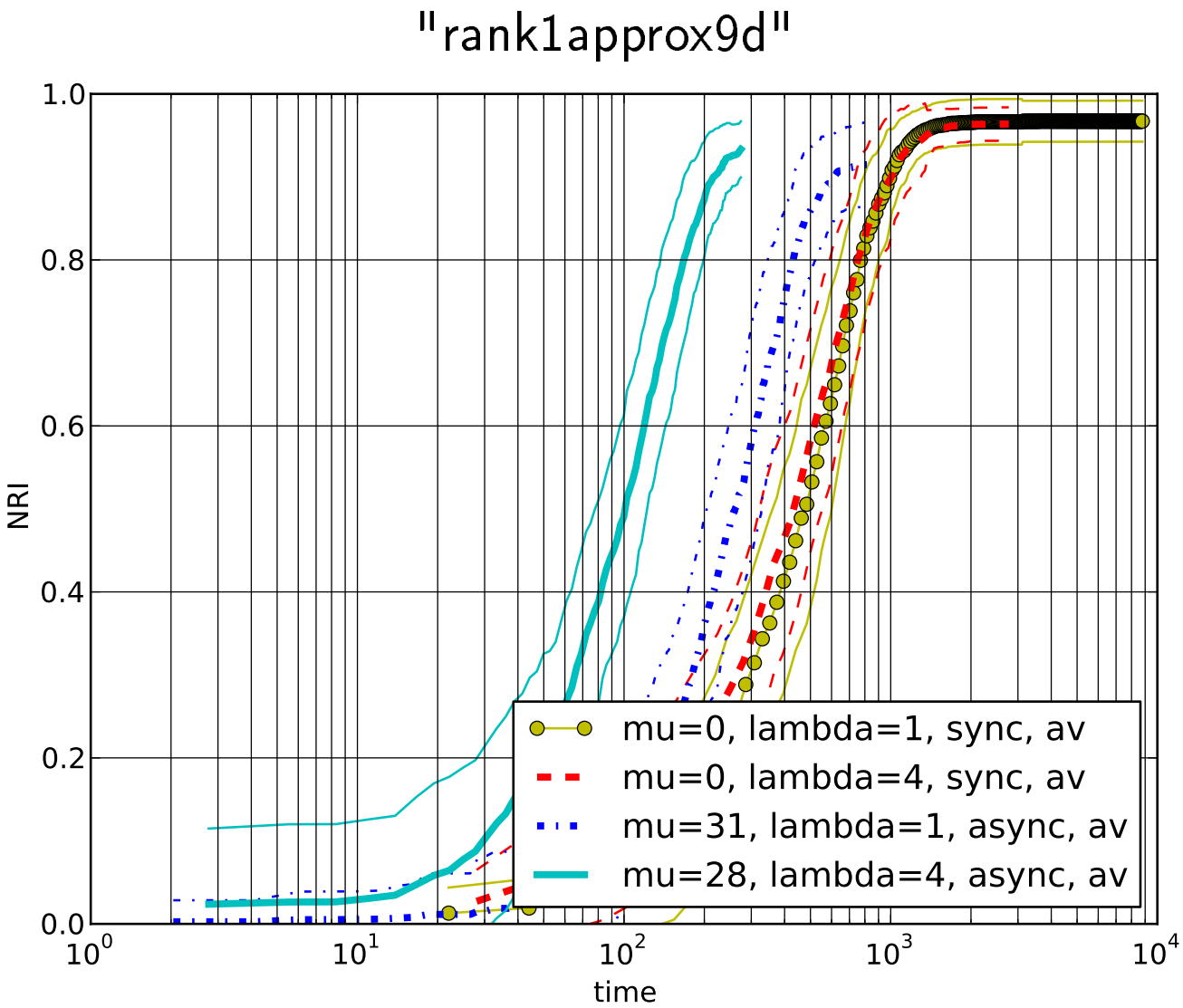

"michalewicz2d"
NRI
generation
1.0
0.8
0.6
0.4
0.2
0.0
10^0
10^1
10^2
10^3
mu=0, lambda=1, sync, av
mu=0, lambda=4, sync, av
mu=31, lambda=1, async, av
mu=28, lambda=4, async, av


"rosenbrock6d"
NRI
generation
1.0
0.8
0.6
0.4
0.2
0.0
10^0
10^1
10^2
10^3
mu=0, lambda=1, sync, av
mu=0, lambda=4, sync, av
mu=31, lambda=1, async, av
mu=28, lambda=4, async, av


"rank1approx9d"
NRI
generation
1.0
0.8
0.6
0.4
0.2
0.0
10^0
10^1
10^2
10^3
mu=0, lambda=1, sync, av
mu=0, lambda=4, sync, av
mu=31, lambda=1, async, av
mu=28, lambda=4, async, av


"michalewicz2d"
NRI
time
1.0
0.8
0.6
0.4
0.2
0.0
10^0
10^1
10^2
10^3
10^4
mu=0, lambda=1, sync, av
mu=0, lambda=4, sync, av
mu=31, lambda=1, async, av
mu=28, lambda=4, async, av


"rosenbrock6d"
NRI
time
1.0
0.8
0.6
0.4
0.2
0.0
10^0
10^1
10^2
10^3
10^4
mu=0, lambda=1, sync, av
mu=0, lambda=4, sync, av
mu=31, lambda=1, async, av
mu=28, lambda=4, async, av


"rank1approx9d"
NRI
time
1.0
0.8
0.6
0.4
0.2
0.0
10^0
10^1
10^2
10^3
10^4
mu=0, lambda=1, sync, av
mu=0, lambda=4, sync, av
mu=31, lambda=1, async, av
mu=28, lambda=4, async, av

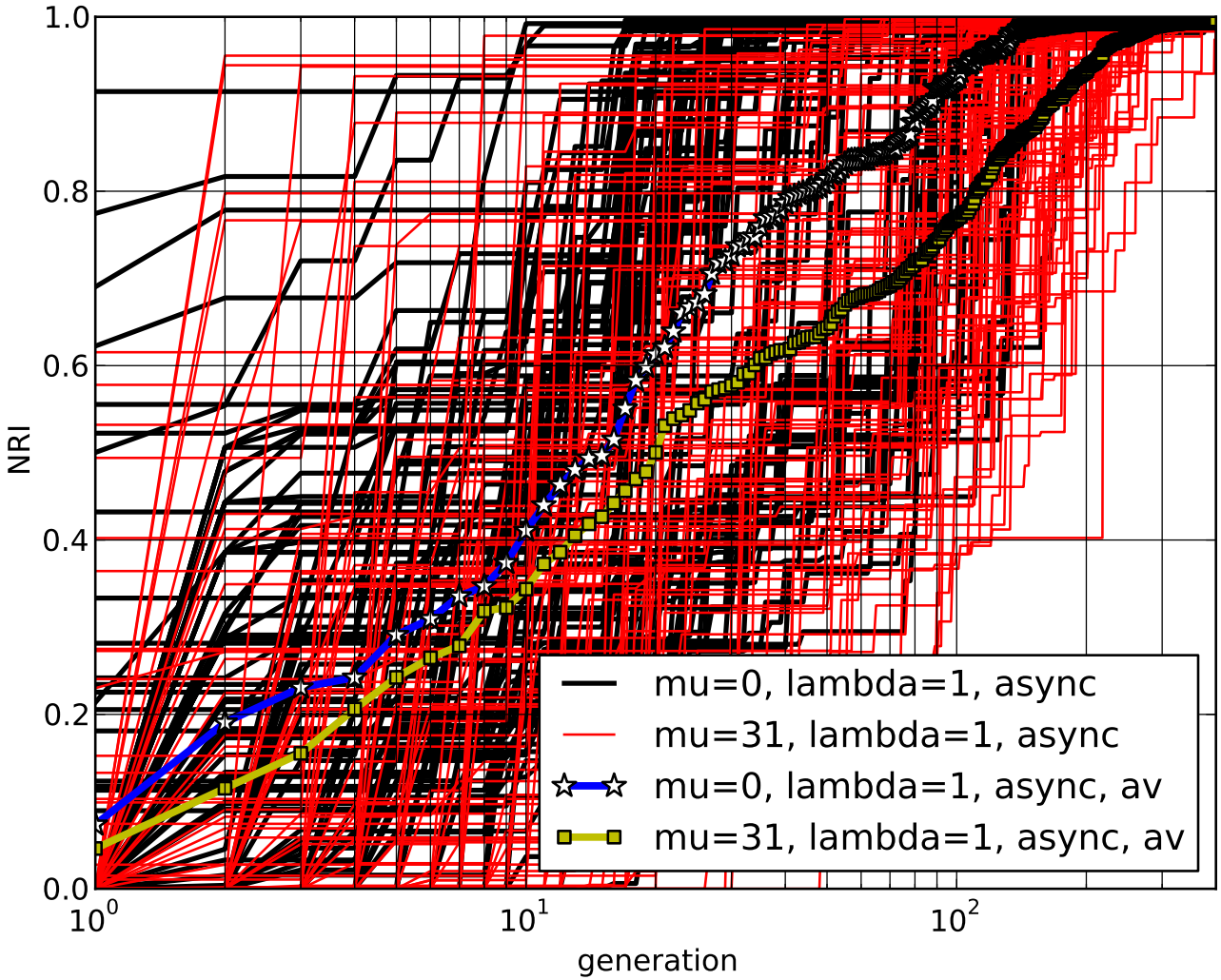
Conclusion (End of Part III)

- In the synchronous access, EUT $= \mathcal{O}(t_{\text{max}})$.
- In the asynchronous access, EUT $= \mathcal{O}(t_{\text{b}})$, and this is possible with realistic buffer sizes $\mathcal{O}(100)$.
- The algorithm with $\mu = 28, \lambda = 4$ is consistently 3x faster than $\mu = 31$, $\lambda = 1$.
- In real time, the speed-up may reach 4x ("rank1approx9d").
- The above comments apply to the grid with heterogeneous nodes.

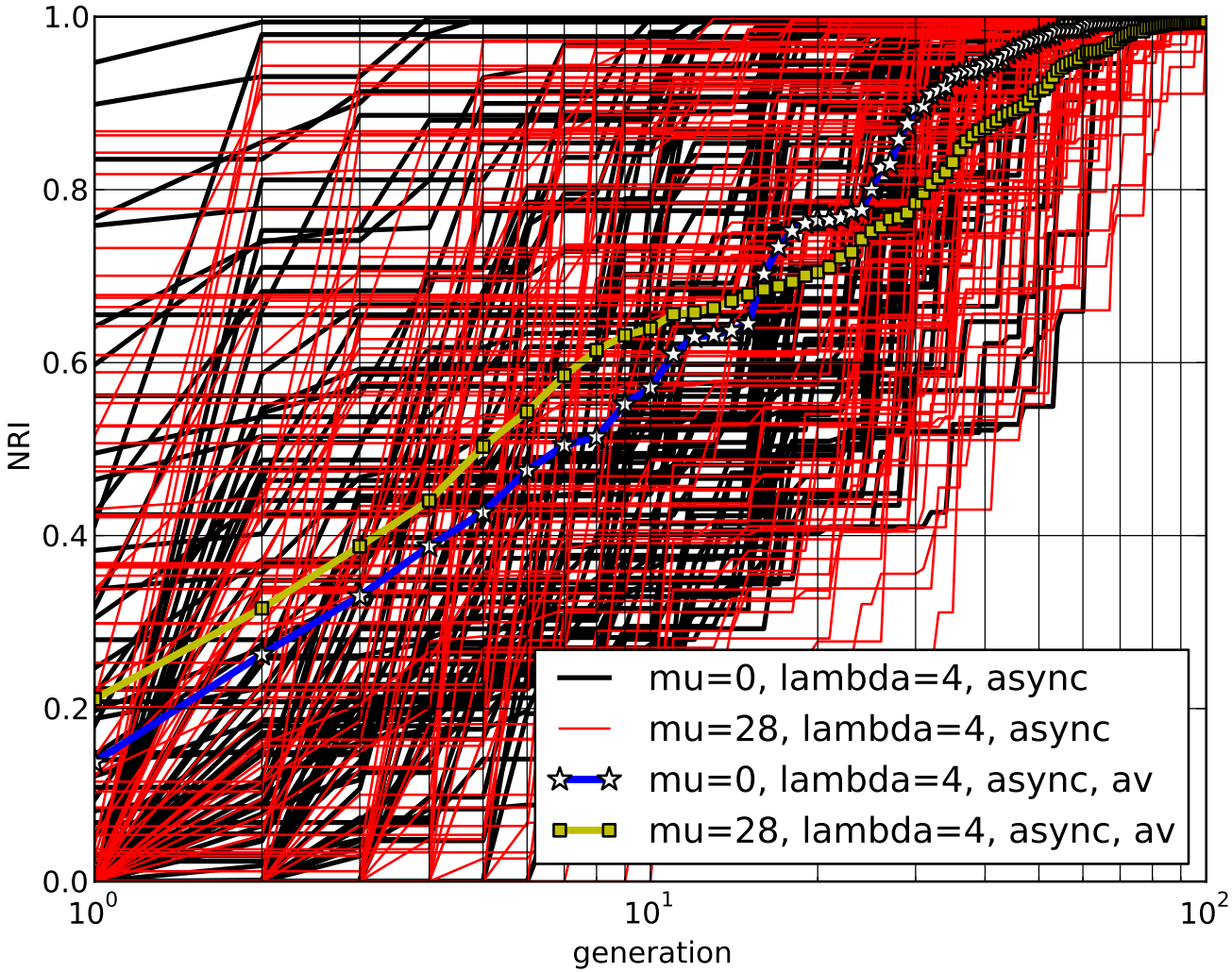
Part IV. Asynchronous Gaussian Conditioning

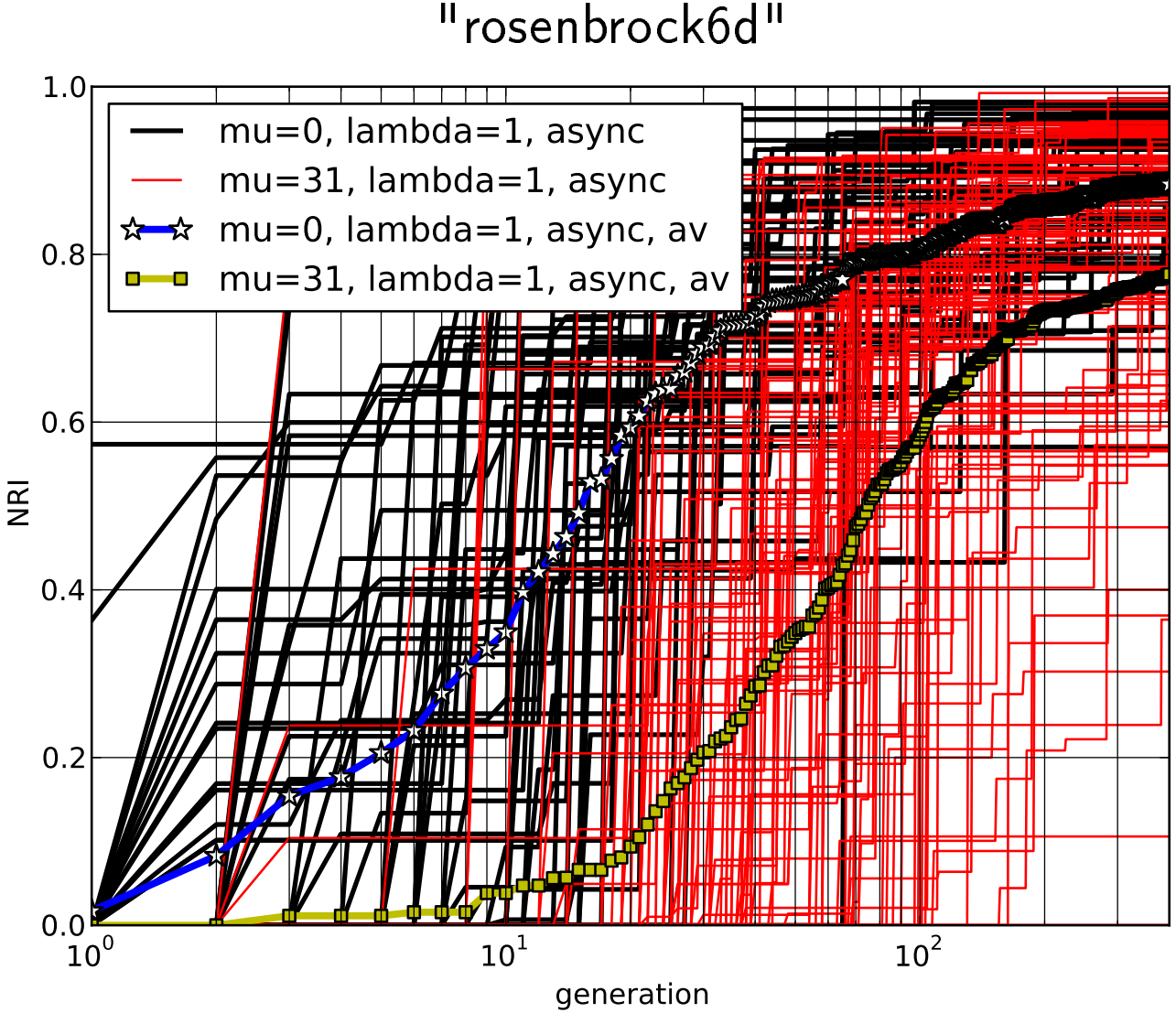
1. Are multi-point improvements useful with the asynchronous node access? Yes.
2. Does the conditioning on μ points help? Theoretically, yes.

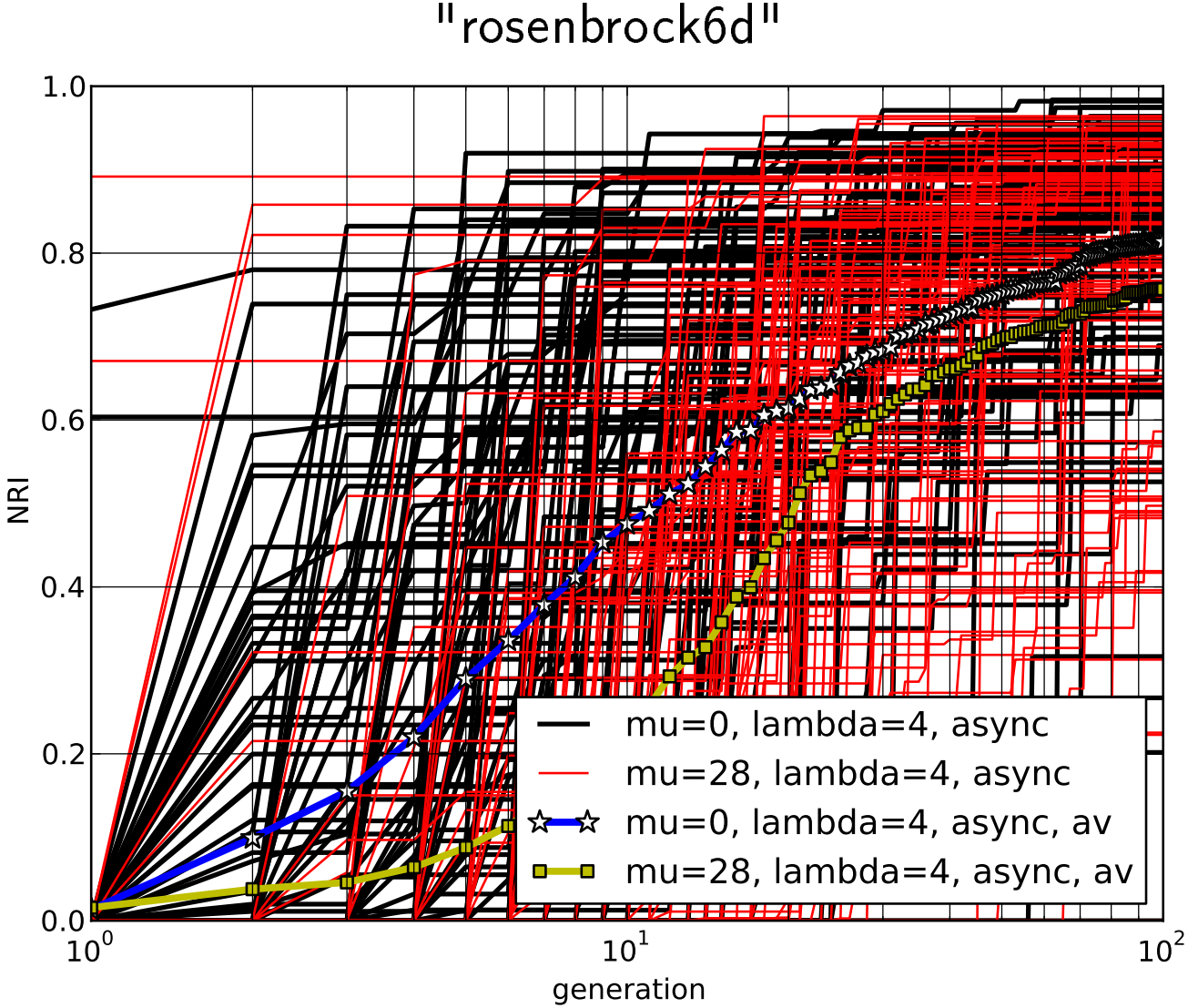
"michalewicz2d"

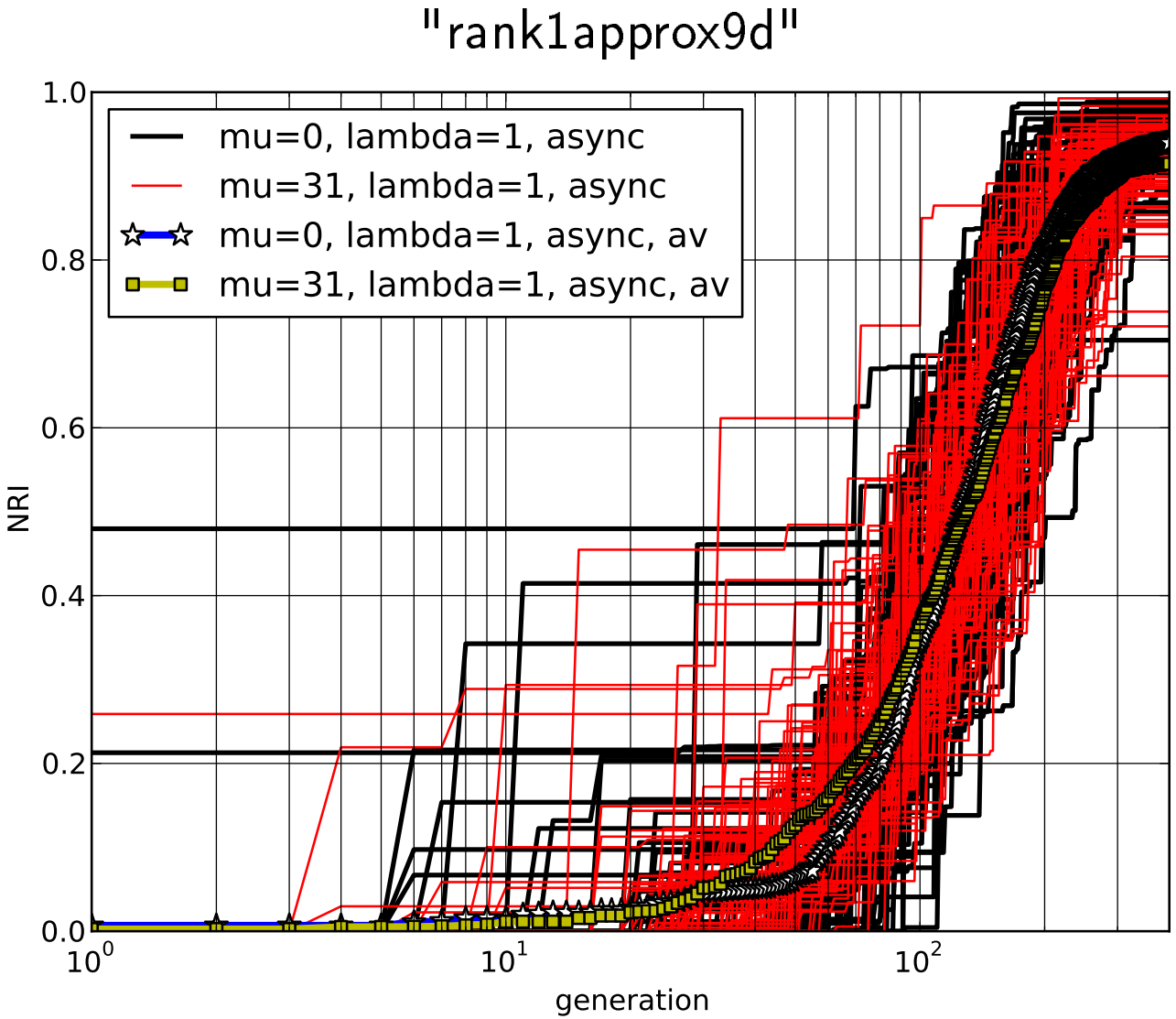


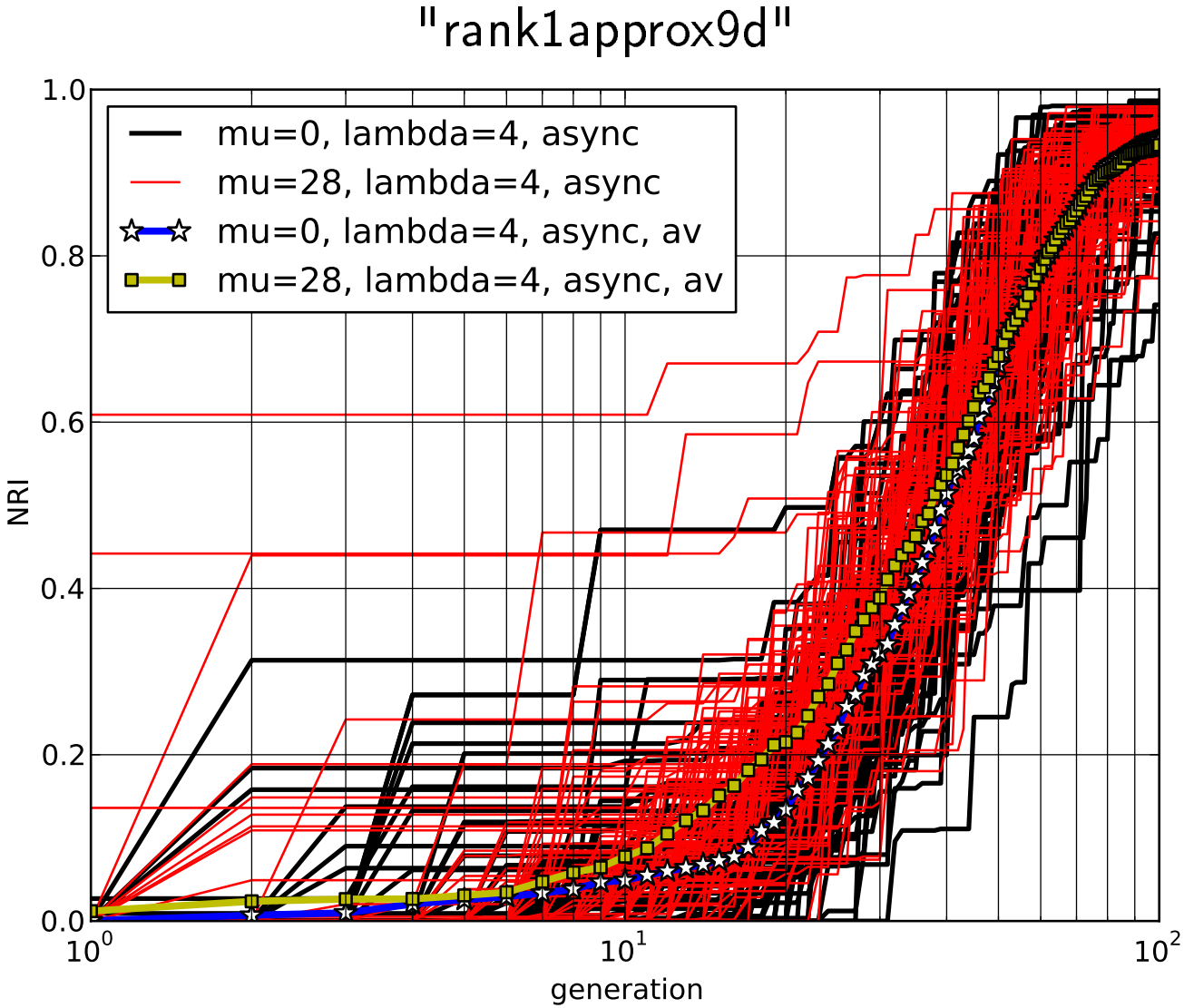
"michalewicz2d"



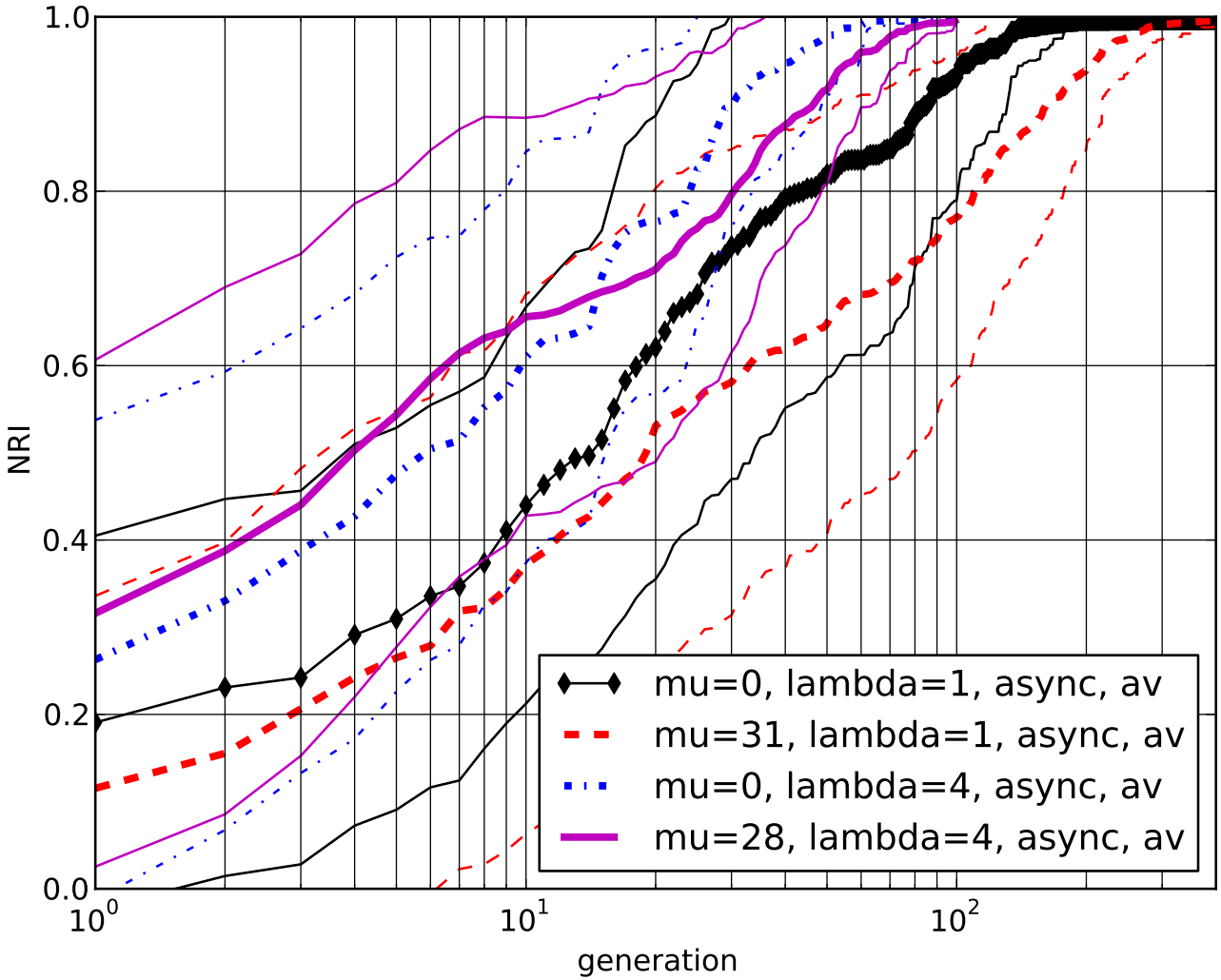

"rosenbrock6d"
mu=0, lambda=1, async
mu=31, lambda=1, async
mu=0, lambda=1, async, av
mu=31, lambda=1, async, av
NRI
generation
1.0
0.8
0.6
0.4
0.2
0.0
10^0
10^1
10^2

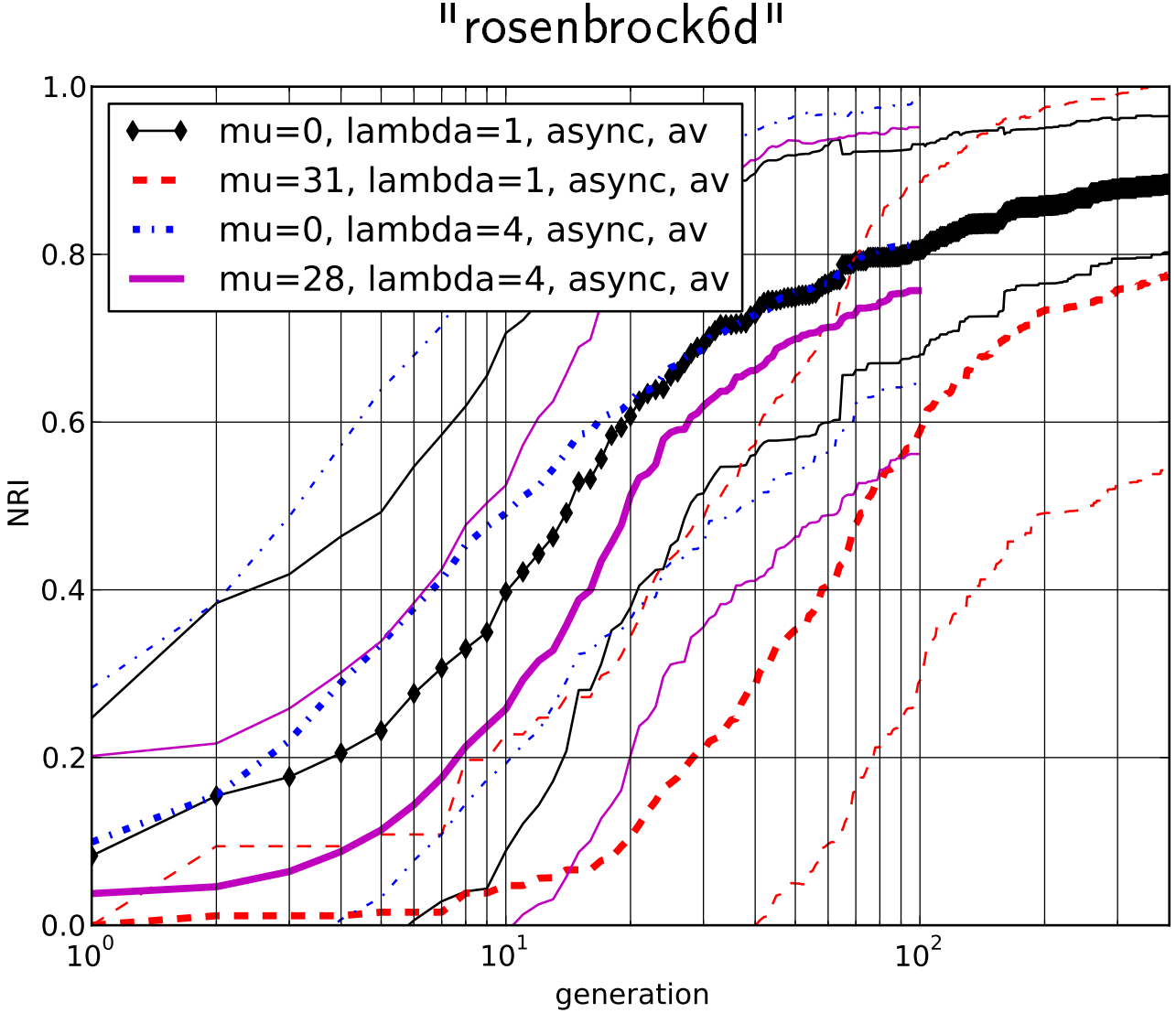

"rosenbrock6d"
NRI
generation
1.0
0.8
0.6
0.4
0.2
0.0
10^0
10^1
10^2
mu=0, lambda=4, async
mu=28, lambda=4, async
mu=0, lambda=4, async, av
mu=28, lambda=4, async, av

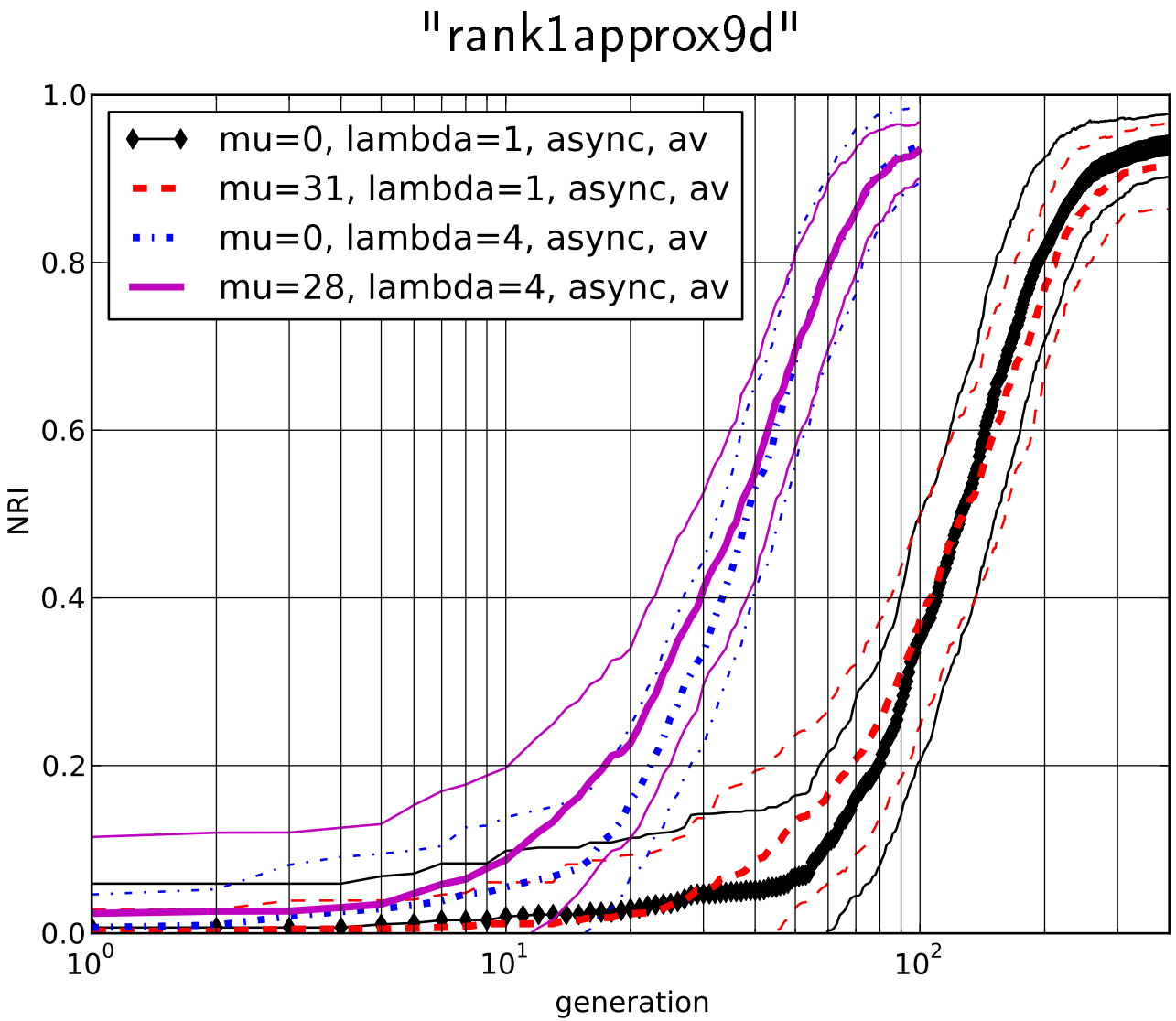

"rank1approx9d"
mu=0, lambda=1, async
mu=31, lambda=1, async
mu=0, lambda=1, async, av
mu=31, lambda=1, async, av
NRI
generation
1.0
0.8
0.6
0.4
0.2
0.0
10^0
10^1
10^2


"rank1approx9d"
mu=0, lambda=4, async
mu=28, lambda=4, async
mu=0, lambda=4, async, av
mu=28, lambda=4, async, av
NRI
generation
1.0
0.8
0.6
0.4
0.2
0.0
10^0
10^1
10^2

"michalewicz2d"



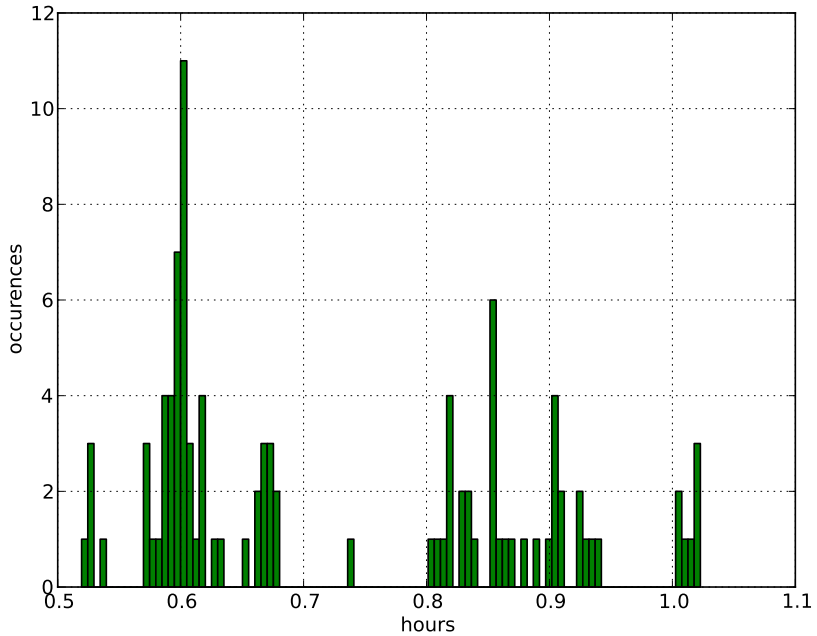

"rosenbrock6d"
mu=0, lambda=1, async, av
mu=31, lambda=1, async, av
mu=0, lambda=4, async, av
mu=28, lambda=4, async, av
NRI
generation
1.0
0.8
0.6
0.4
0.2
0.0
10^0
10^1
10^2


"rank1approx9d"
mu=0, lambda=1, async, av
mu=31, lambda=1, async, av
mu=0, lambda=4, async, av
mu=28, lambda=4, async, av
NRI
generation
1.0
0.8
0.6
0.4
0.2
0.0
10^0
10^1
10^2

Conclusion (Part IV)

- The conditioning on μ points dooes not improve the optimization.
- It considerably increases the blocking time. For "rank1approx9d", t_b increases 10x when $\mu = 0 \longrightarrow 31$.

$(\mu = 0, \lambda = 1)$



$(\mu = 31, \lambda = 1)$

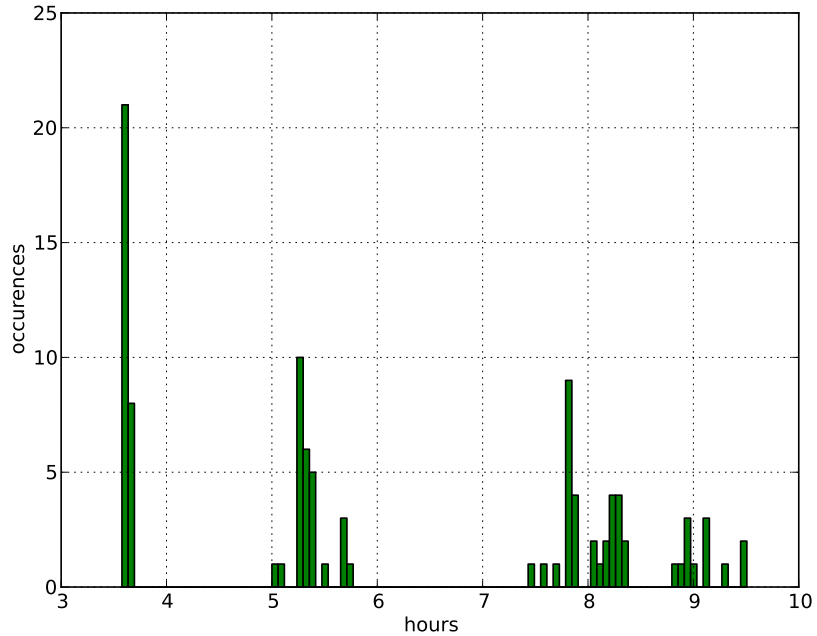


Figure: Optimization is also an "expensive-to-evaluate function".

Part V. Expectation of Multi-Point Improvement

- How to calculate the integral in [Ginsbourger et al., 2010]?

$$\alpha(\mathbf{m}, \mathbf{C}) = \int_{\mathbb{R}^d} g(\mathbf{y}) p(\mathbf{y}) d\mathbf{y}, \tag{7}$$

$$g(\mathbf{y}) = \max\left(0, f_{\text{best}} - \min \mathbf{y}\right), \tag{8}$$

$$\mathbf{y} \sim N(\mathbf{m}, \mathbf{C}). \tag{9}$$

- After "standardization" with $\mathbf{C} = \mathbf{L}\mathbf{L}^T$:

$$\alpha(\mathbf{m}, \mathbf{C}) = \int_{\mathbb{R}^d} g(\mathbf{m} + \mathbf{L}\mathbf{u}) p(\mathbf{u}) d\mathbf{u}, \tag{10}$$

$$p(\mathbf{u}) \sim N(\mathbf{0}, \mathbf{I}). \tag{11}$$

- Needs to be evaluated $10^5 \ldots 10^6$ times during a single run.

- The integral can be bounded [Ginsbourger et al., 2010]:

$$\max_{1\leq i\leq d} \alpha(m_i, \sigma_{ii}^2) \leq \alpha(\mathbf{m}, \mathbf{C}) \leq \sum_{i}^{d} \alpha(m_i, \sigma_{ii}^2), \qquad (12)$$

where

$$\alpha(m_i, \sigma_{ii}^2) = (f_{\text{best}} - m_i)\Phi\left(\frac{f_{\text{best}} - m_i}{\sigma_i}\right) + \sigma_i\,\phi\left(\frac{f_{\text{best}} - m_i}{\sigma_i}\right). \qquad (13)$$

- Φ and ϕ are the distribution and density functions of the standard normal variable, resp.
- The upper bound is often very close to the true value,
- but its maximization is useless as the point with the maximal one-point improvement gets replicated $d - 1$ times.

- Spherical rules, unscented transforms do not work well.
- Standard MC with 10^3 is a practical choice.
- Two methods have been developed that might compete with MC-10^3: ENSEMI1, and ENSEMI2.
- They are more accurate for suboptimal improvements.
- Less accurate at the points with the maximal improvement.
- Tested thoroughly, (about 100 integrals, dimensions up to $d = 16$).

- One works with a fully-symmetric rule

$$\int_{\mathbb{R}^d} h(\mathbf{u})p(\mathbf{u})d\mathbf{u} \simeq \sum_{i=1}^{d} w_i \left(h(\mathbf{P}^i\mathbf{v}) + h(-\mathbf{P}^i\mathbf{v})\right), \qquad (14)$$

where $\mathbf{P}$ is the circular shift matrix, $\mathbf{v} = [v, 0, \ldots, 0]^T \in \mathbb{R}^d$, where v is a free parameter.

- Each "slice" of $h(\mathbf{u})$ must be integrated exactly:

$$\int_{\mathbb{R}^d} \max(0, f_{\text{best}} - m_i - \tilde{\mathbf{l}}_i\mathbf{u})p(\mathbf{u})d\mathbf{u} =$$

$$= m_i'\Phi\left(\frac{m_i'}{\sigma_i'}\right) + \sigma_i'\,\phi\left(\frac{m_i'}{\sigma_i'}\right), \qquad (15)$$

where $\tilde{\mathbf{l}}_i$ is the *ith* row of $\mathbf{L}$, $m_i' = f_{\text{best}} - m_i$, and $\sigma_i' = \|\tilde{\mathbf{l}}_i\|$.

- We abbreviate this method as ENSEMI1 (ENSEMI1 is Not Symmetric Exact Monomial Integration).

- An extension, called ENSEMI2, introduces d additional weights

$$\int_{\mathbb{R}^d} h(\mathbf{u})p(\mathbf{u})d\mathbf{u} \simeq \sum_{i=1}^{d} w_i h(\mathbf{P}^i\mathbf{v}) + \sum_{i=d+1}^{2d} w_i h(-\mathbf{P}^{d+i}\mathbf{v}), \quad (16)$$

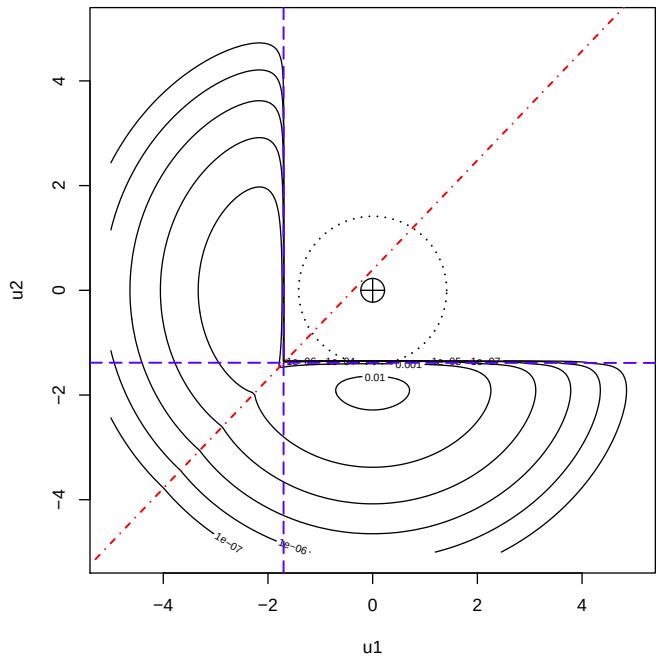
$(\mathbf{P}^{d+i} = \mathbf{P}^i)$.

- One then demands that in addition to the "mean slices", the "variances slices" must be integrated exactly:

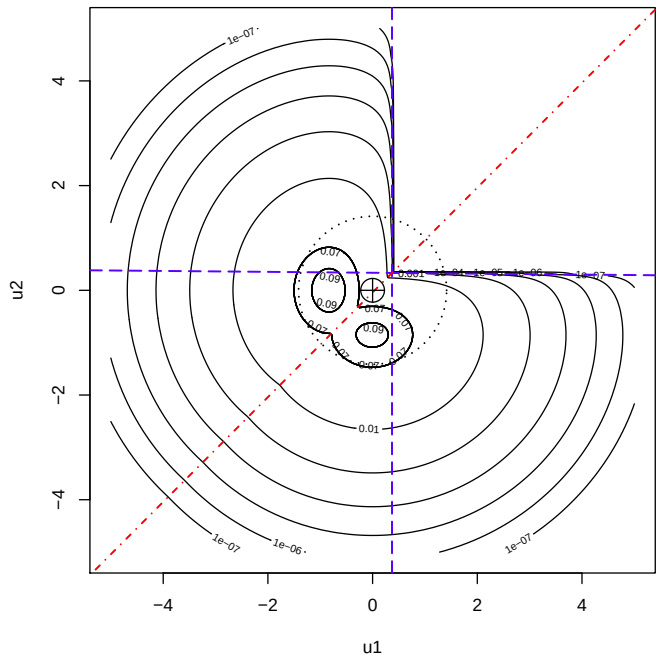
$$\int_{\mathbb{R}^d} \left(\max(0, f_{\text{best}} - m_i - \tilde{\mathbf{l}}_i\mathbf{u}) - s_i \right)^2 p(\mathbf{u})d\mathbf{u}$$
$$= \sigma_i'^2\Phi^2\left(\frac{m_i'}{\sigma_i'}\right) + \sigma_i'^2\phi^2\left(\frac{m_i'}{\sigma_i'}\right) - m_i'\sigma_i\Phi\left(\frac{m_i'}{\sigma_i'}\right)\phi\left(\frac{m_i'}{\sigma_i'}\right). \quad (17)$$

What Do the Integrands Look Like?

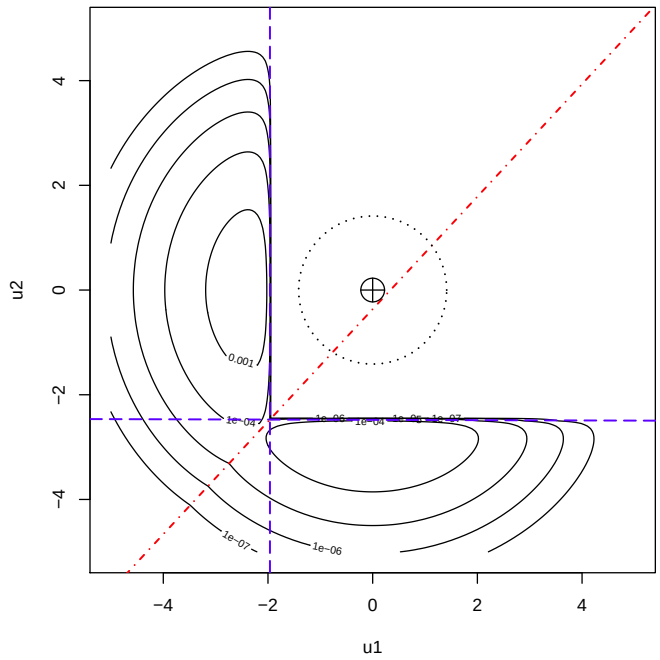
(iter: 1, loc: random)



(iter: 1, loc: optimal)



(iter: 50, loc: random)



(iter: 50, loc: optimal)

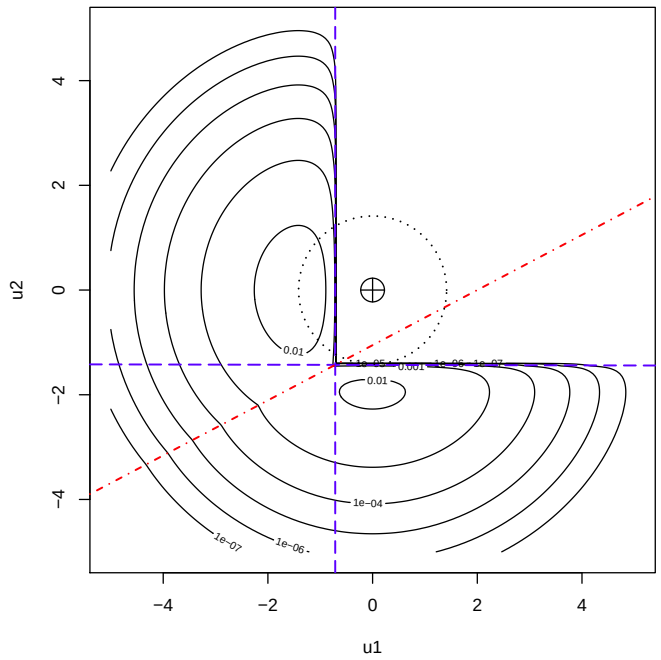


Table: Median Relative Integration Errors, %. Far from Optimum.

	Problem Set I		
	$d = 2$	$d = 8$	$d = 16$
Upper Bound [Ginsbourger et al., 2010]	0.37	0.93	1.8
ENSEMI1	0.32	1.0	2.7
MC, $n = 10^3$	21 ± 6	14 ± 3	10 ± 2
ENSEMI2	100	4.5	11
Improved Lu–Darmofal	18	34	60
Lower Bound [Ginsbourger et al., 2010]	34	68	79
Lu–Darmofal, 2004	100	170	370

Table: Median Relative Integration Errors, %. Close to Optimum.

	Problem Set II		
	$d = 2$	$d = 8$	$d = 16$
MC, $n = 10^3$	4 ± 1	3 ± 1	3 ± 1
ENSEMI2	6.9	10	5.1
ENSEMI1	3.0	6.0	10
Upper Bound [Ginsbourger et al., 2010]	3.0	14	18
Improved Lu–Darmofal	3.6	9.1	45
Lu–Darmofal, 2004	10	9.8	68
Lower Bound [Ginsbourger et al., 2010]	25	64	73

Conclusion (End of Part V)

- The ability to integrate monomials exactly is overestimated.
- The accuracy of the economical MC sampling is very high.
- When $d = 2$, the relative difference between the upper bound and the true value of the expected improvement can be as low as 0.37%. ENSEMI1 reduces it further down to 0.32%.
- ENSEMIx demand $O(d^3)$ multiplications per integration, while MC needs $O(d^2 n)$, given the number of samples n. Typically, $d \ll n$, and the loss of the accuracy is acceptable.
- Integration is likely to make a difference only when d is large, but then improvements become overestimated, etc.


Ginsbourger, D., Riche, R. L., and Carraro, L. (2010).
Kriging is well-suited to parallelize optimization.
DOI:10.1007/978-3-642-10701-6_6.


Janusevskis, J., Riche, R. L., Ginsbourger, D., and Girdziusas, R. (2012).
Expected improvements for the asynchronous parallel global optimization of expensive functions: Potentials and challenges.
http://www.emse.fr/~leriche/.


Raghavan, B., Breitkopf, P., Tourbier, Y., and Villon, P. (2012).
Towards a space reduction approach for efficient structural shape optimization.
Submitted to J. Struct. Multidiscipl. Opt.

Details for the Asynchronous Model of the Average Node Update Time

Below we present the algorithm and the Scilab code to estimate the EUT values given in Table 5 and Fig. 51.

- The model fills λ readily available nodes with new requests.
- Other models are possible, such as "replace the free λ nodes with totally new nodes".
- In what follows, $\mathcal{T}$ is a set of $\mu + \lambda$ elements, and each element is the time it takes to evaluate an expensive function by a particular node.

1. Find λ smallest elements of $\mathcal{T}$ (not necessarily distinct):

$$S = \{t_{i_1}, t_{i_2}, \ldots, t_{i_\lambda}\}. \tag{18}$$

2. Find the largest element in S:

$$t_c = \max S. \tag{19}$$

3. Compute the node update time

$$t_u = t_b + t_c. \tag{20}$$

4. Form the set $M = \mathcal{T} \setminus S$, and map every element t of M according to:

$$t \mapsto \max(0, t - t_u). \tag{21}$$

5. Update the set $\mathcal{T}$:

$$\mathcal{T} = M \cup S. \tag{22}$$

Listing 1: Scilab code which estimates the value of EUT.

```
function [EUT] = estimateEUT(lamb, buffsz)
tmin = 10;tmax = 30;tb = 2;ngenerations = 250;
grand("setsd", sum((getdate())^2));
tbuff0 = grand(1, buffsz, "unf", tmin, tmax);
tbuff = tbuff0;
tcvec = [];
for i=1:ngenerations
  [vals, inds] = gsort(tbuff, "g", "i");
  tc = max(vals(1:lamb));
  tbuff = tbuff - tc - tb;
  tbuff(inds(1:lamb)) = tbuff0(inds(1:lamb));
  tbuff(find(tbuff<=0)) = 0;
  tcvec($+1) = tc;
end
EUT = mean(tcvec)+tb;
endfunction
```