KNN-Averaging for Noisy Multi-Objective Optimisation

Stefan Klikovits*, Paolo Arcaini

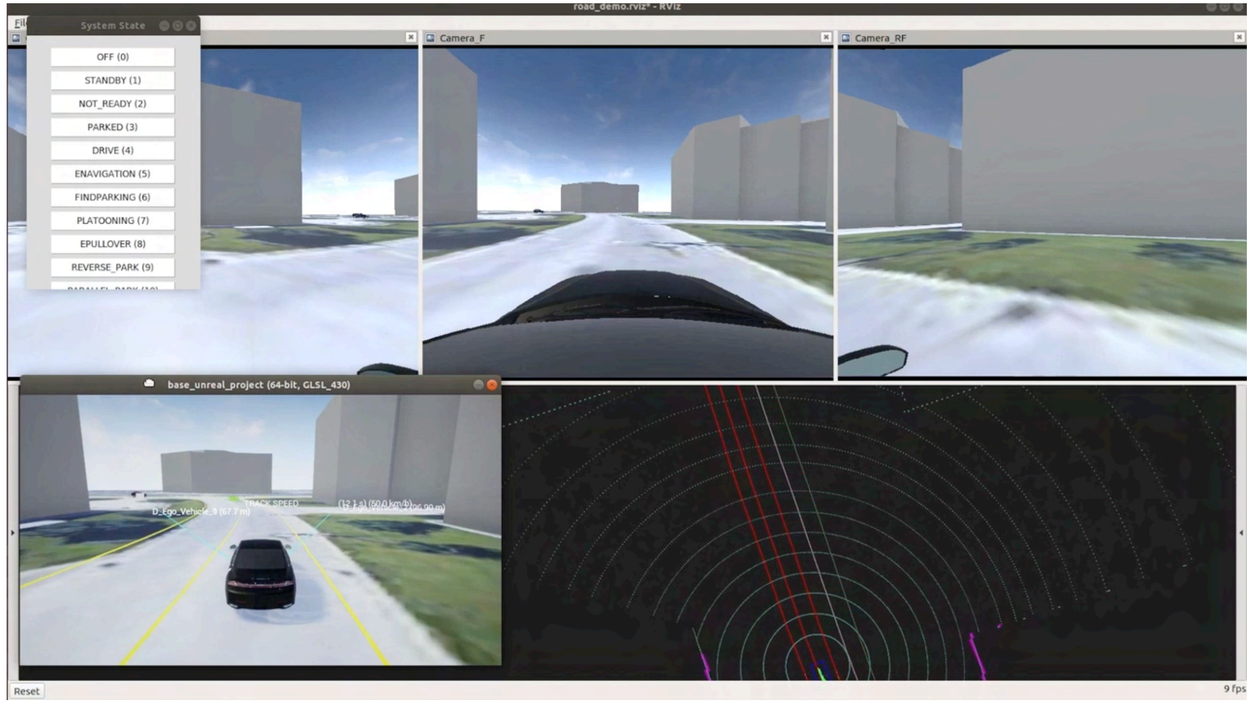
ERATO MMSD Hasuo Project
National Institute of Informatics, Tokyo

*supported by JSPS Kakenhi Grant-in-Aid JP20K23334

Autonomoose ADS & Simulator

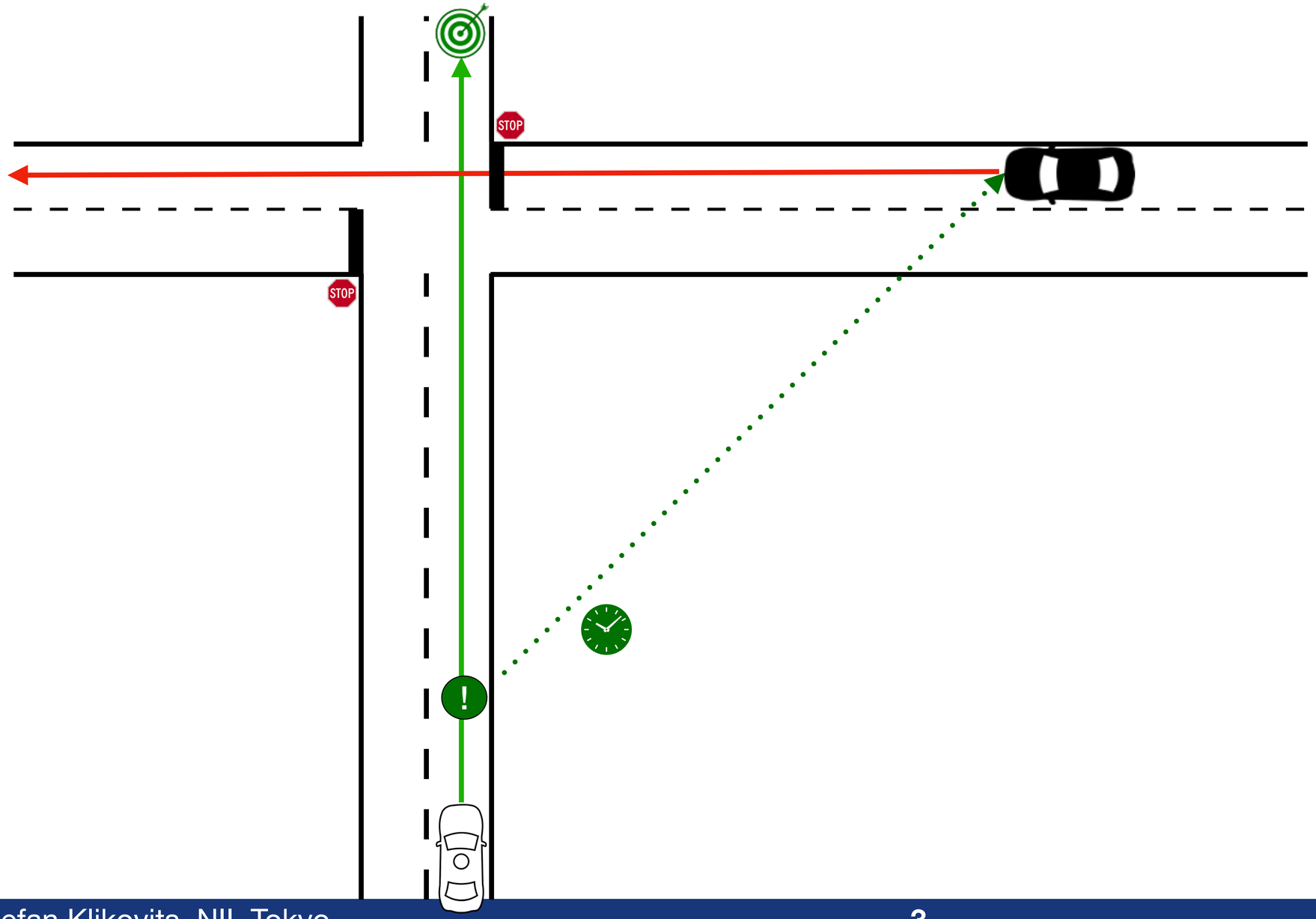
Self-Driving Vehicle

Simulation Platform

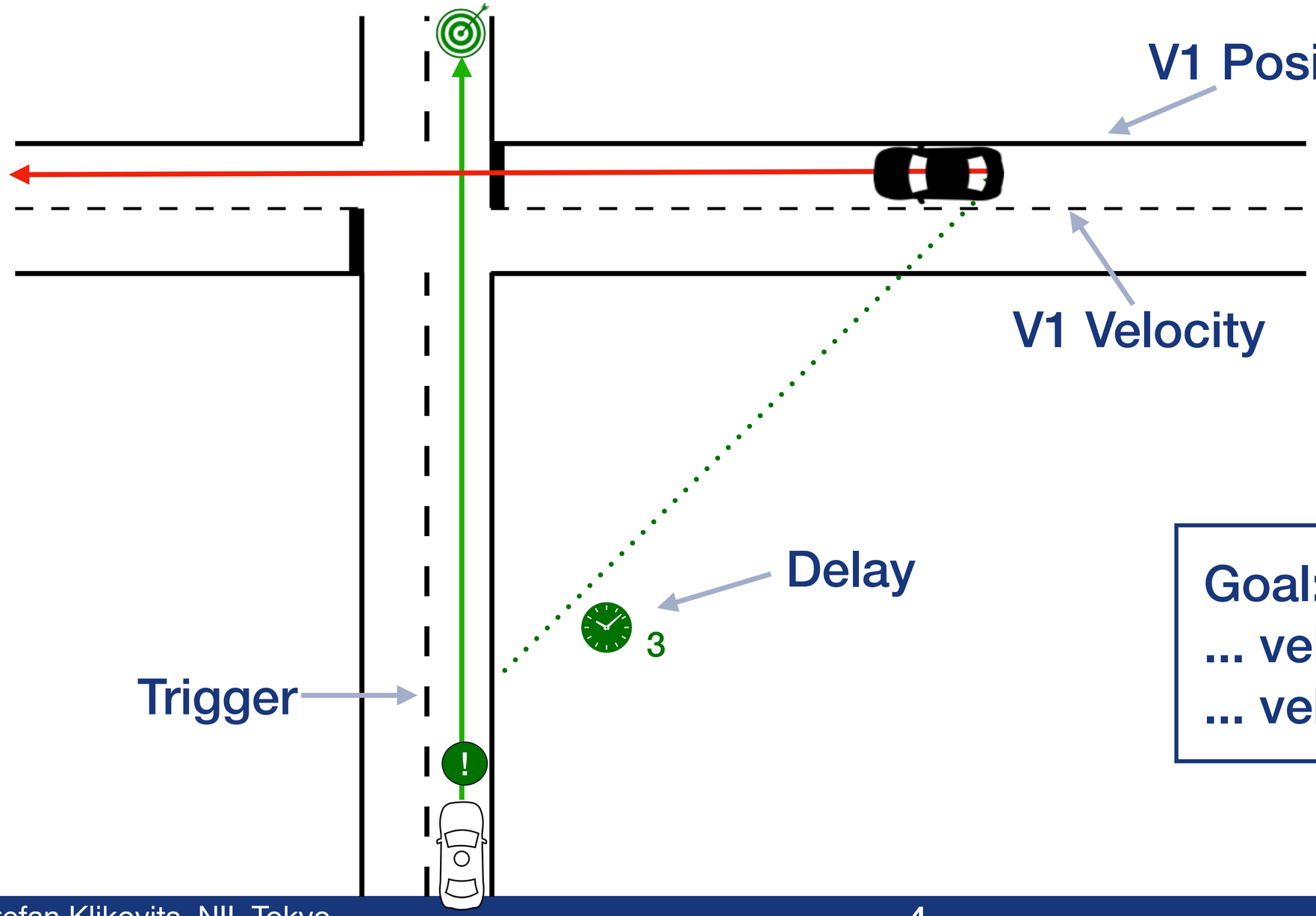


https://www.youtube.com/watch?v=KjQosq-DlMg

Scenario Search-Based Testing

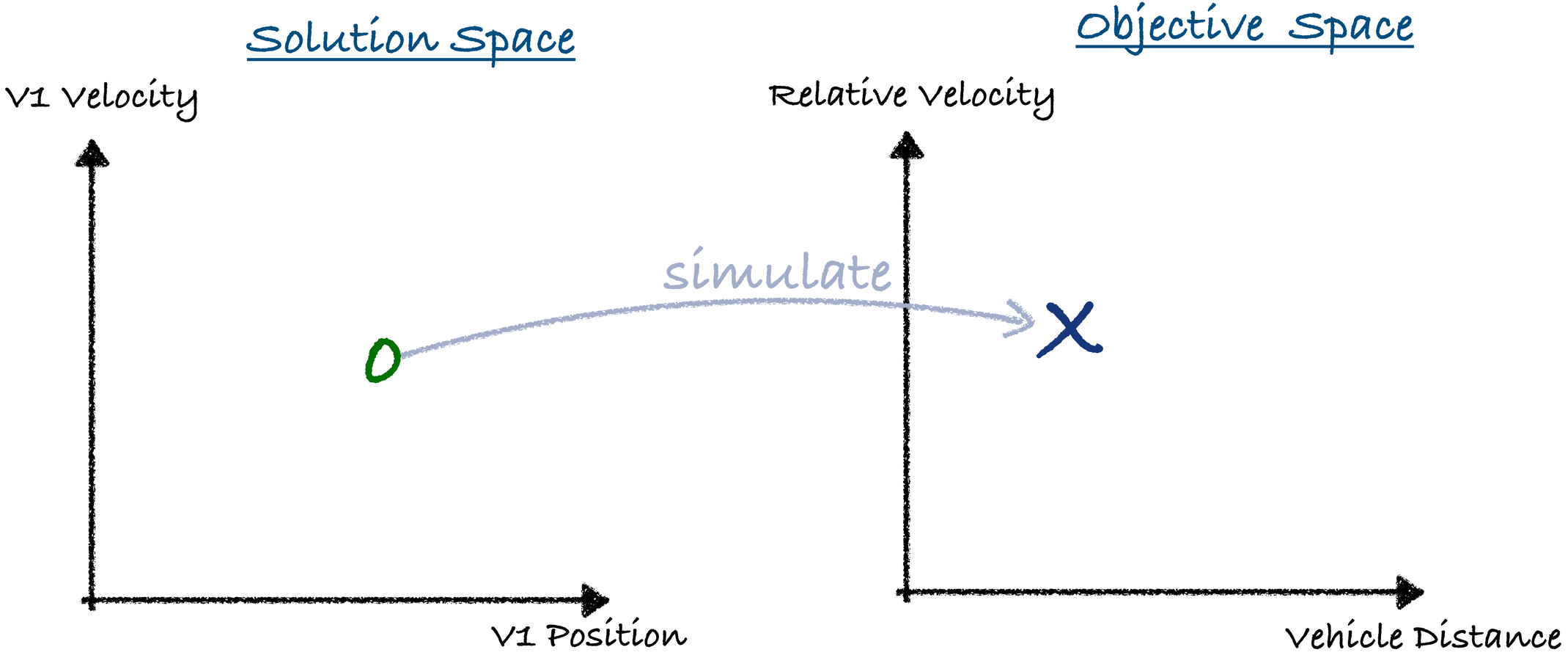


Scenario Search-Based Testing

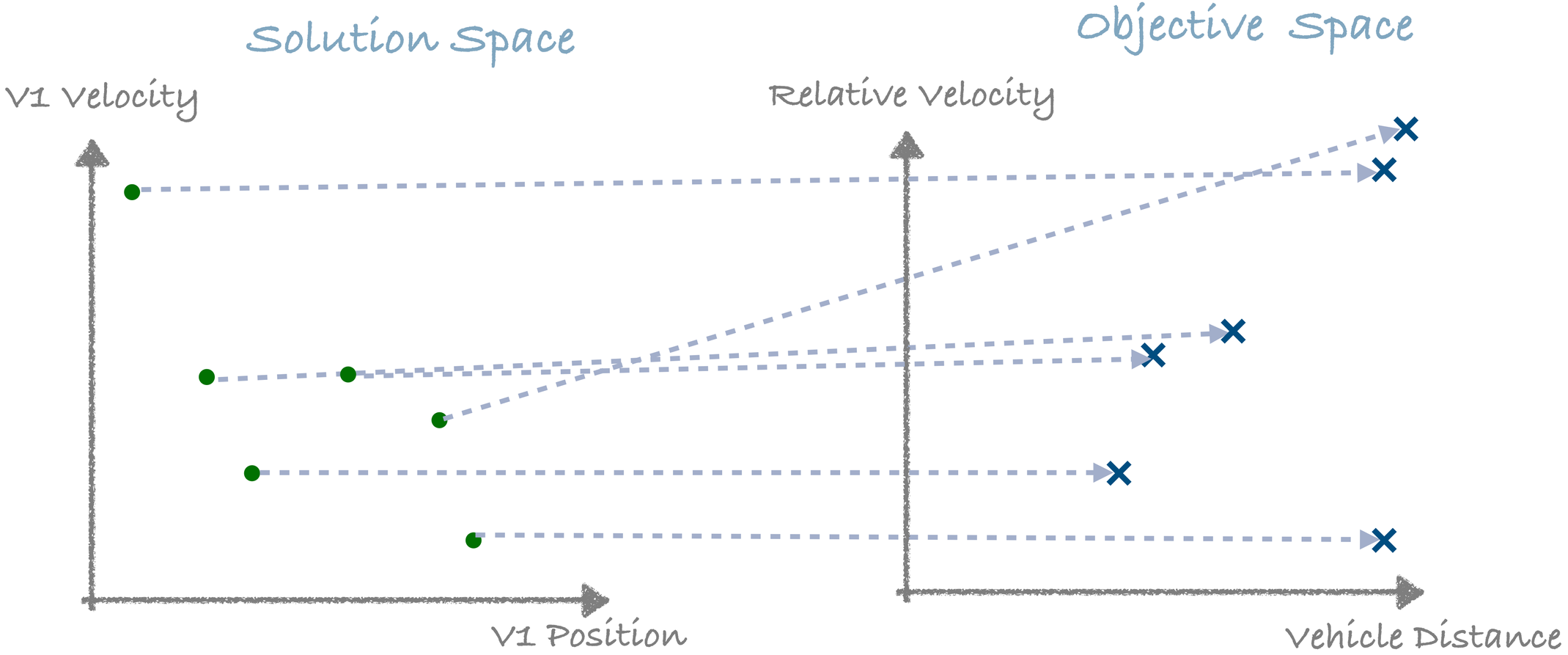


Goal: minimize ...
... vehicle distance
... velocity of V1

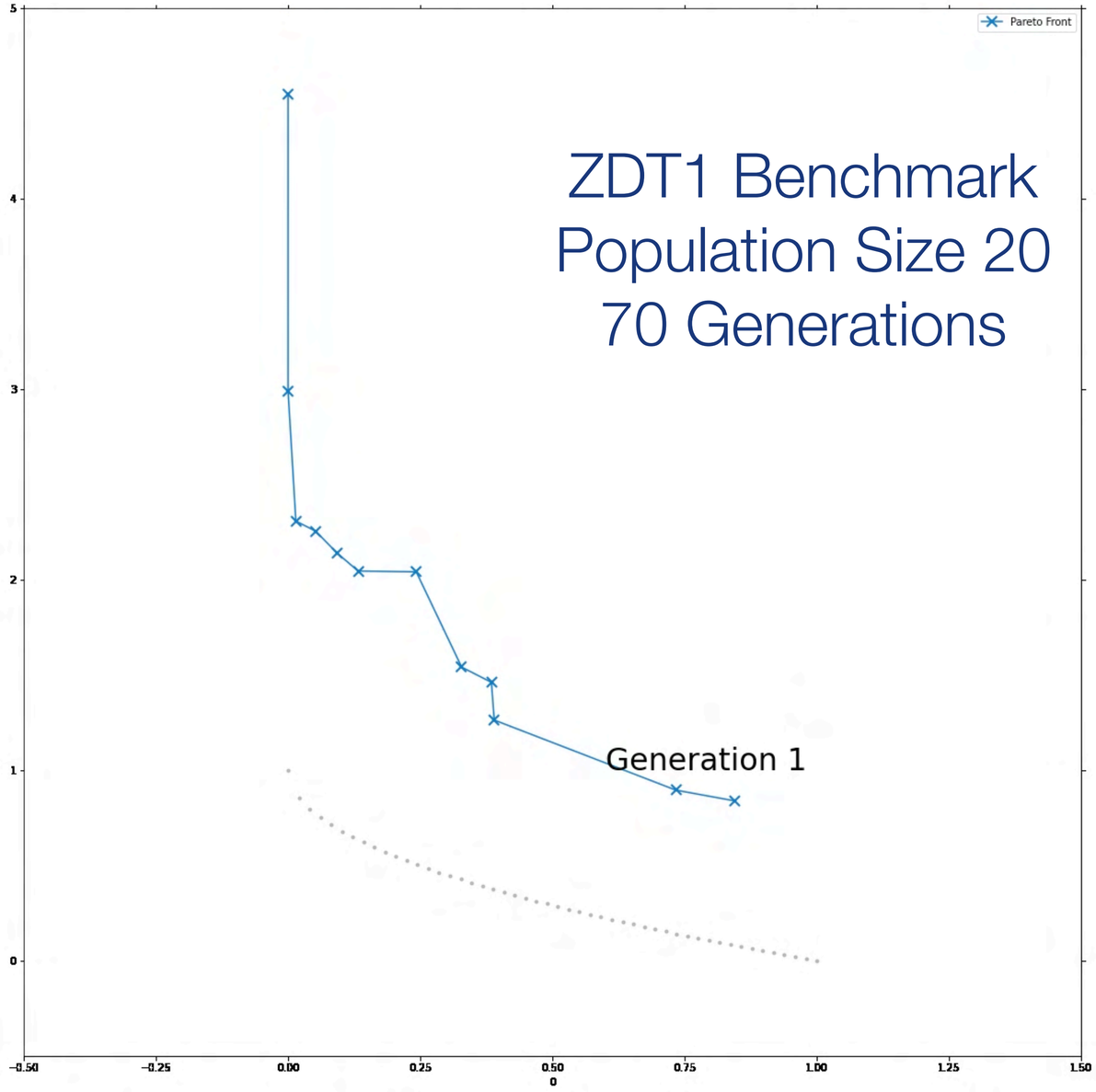
Search-Based Testing (SBT)



Search-Based Testing (SBT)

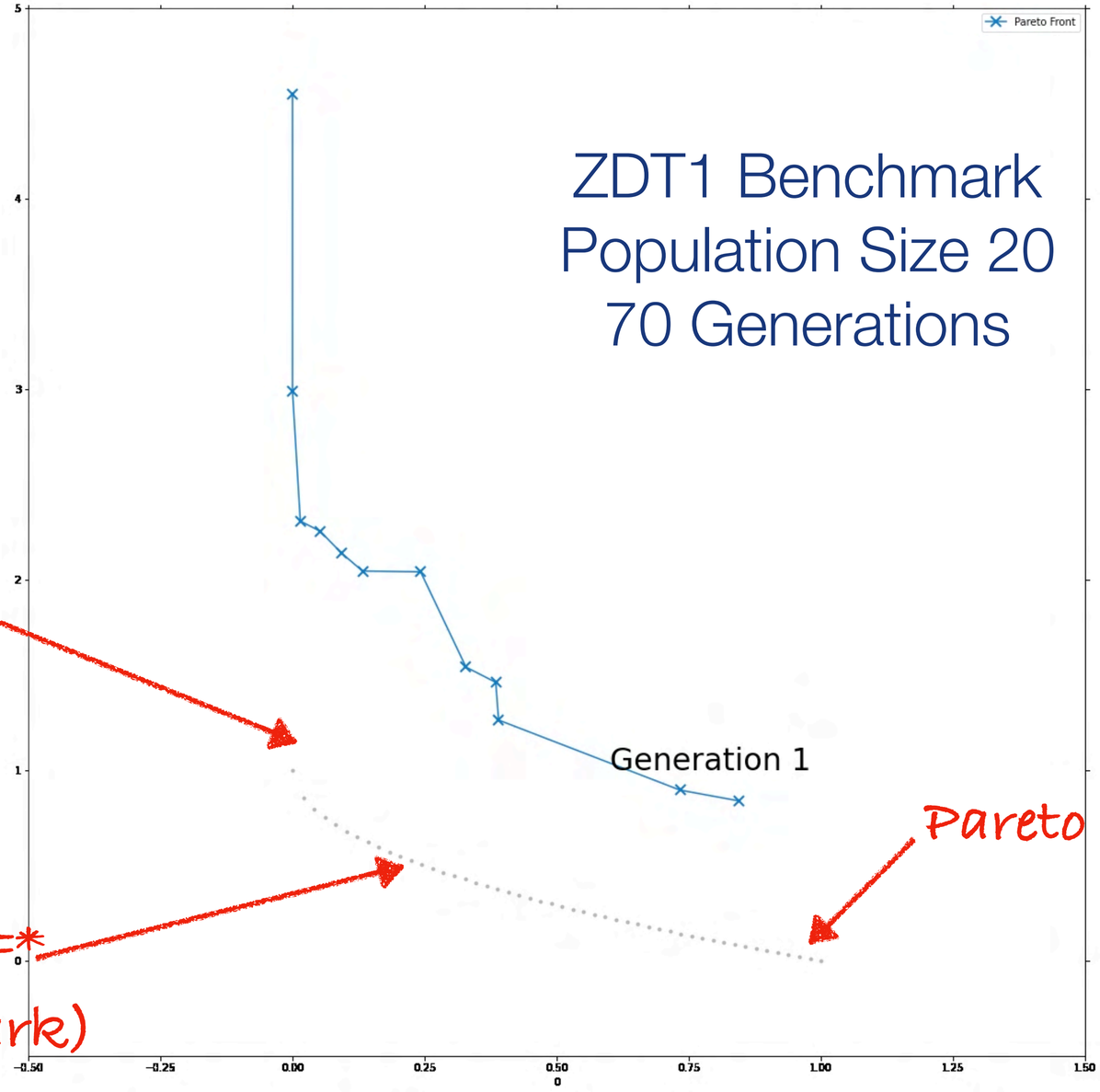


Search-Based Testing (SBT)



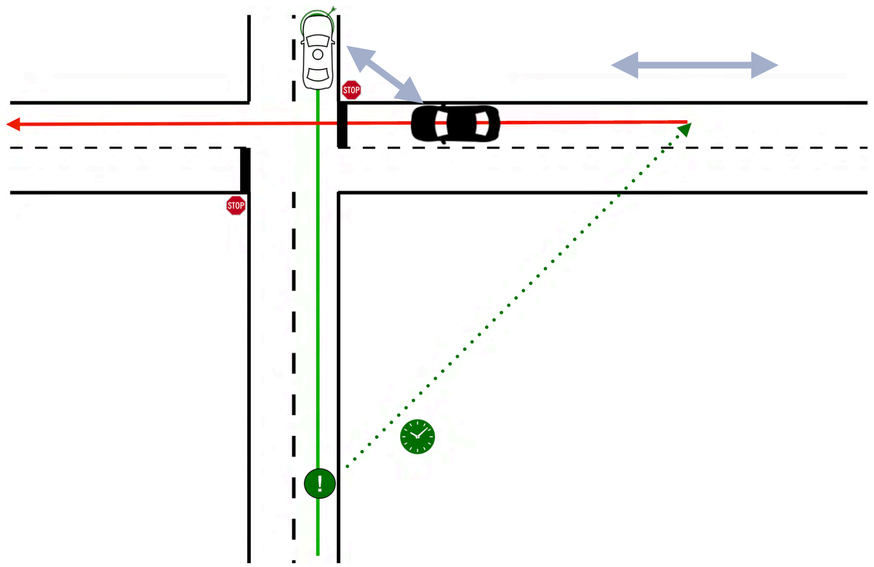
Objective Space

Search-Based Testing (SBT)

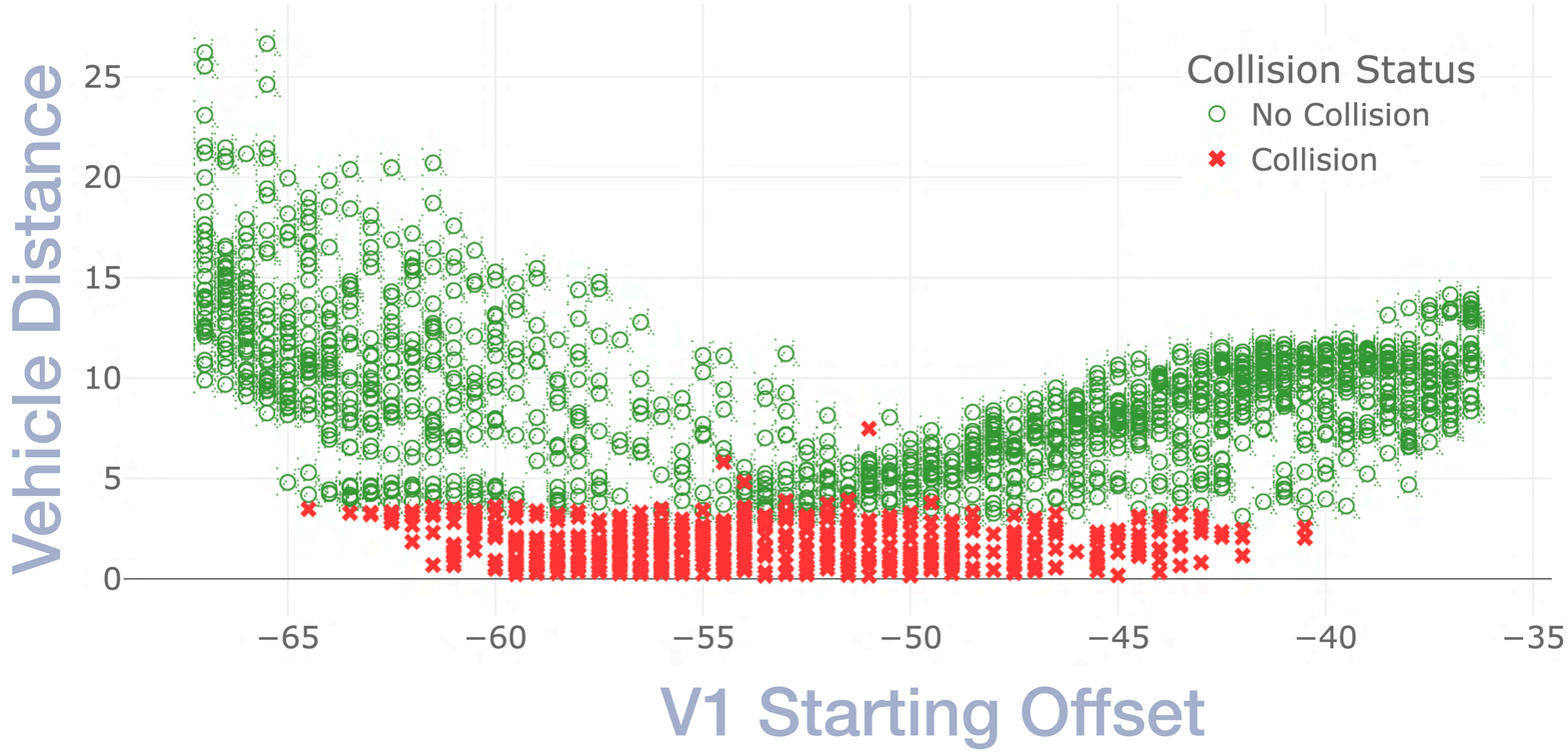


The Problem

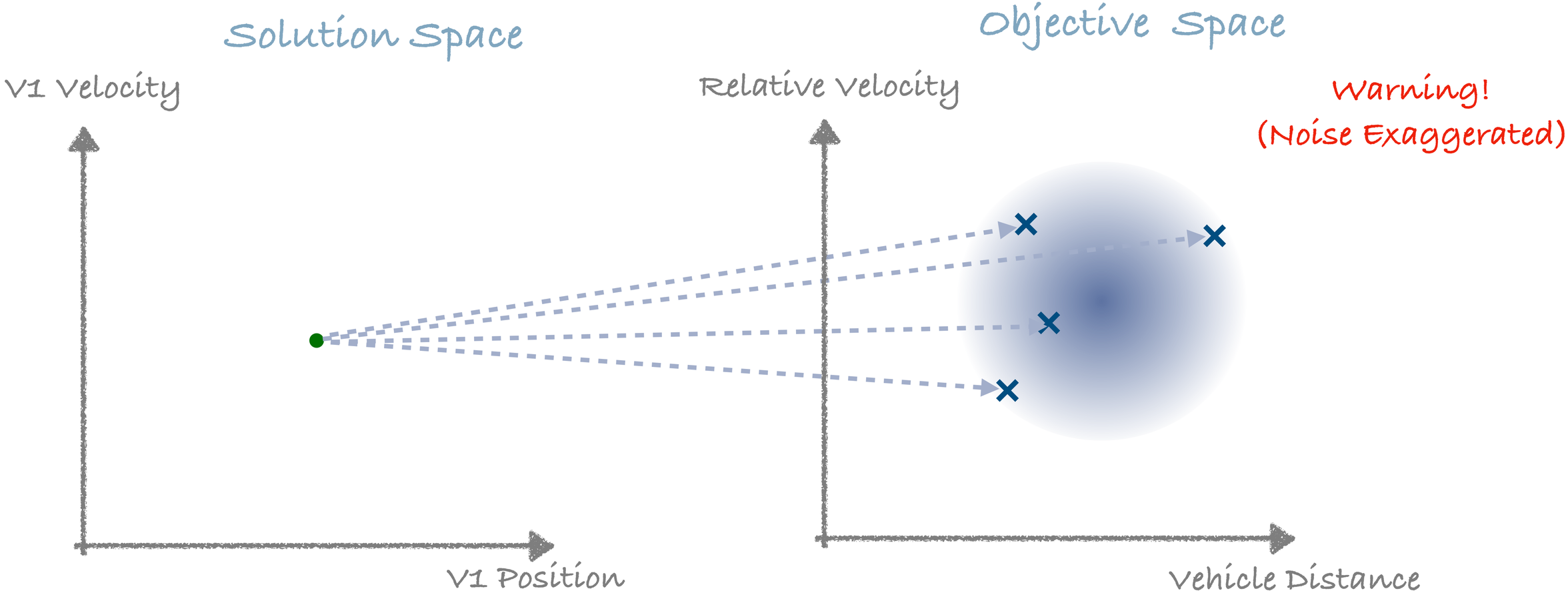
Autonomoose is noisy (non-deterministic)



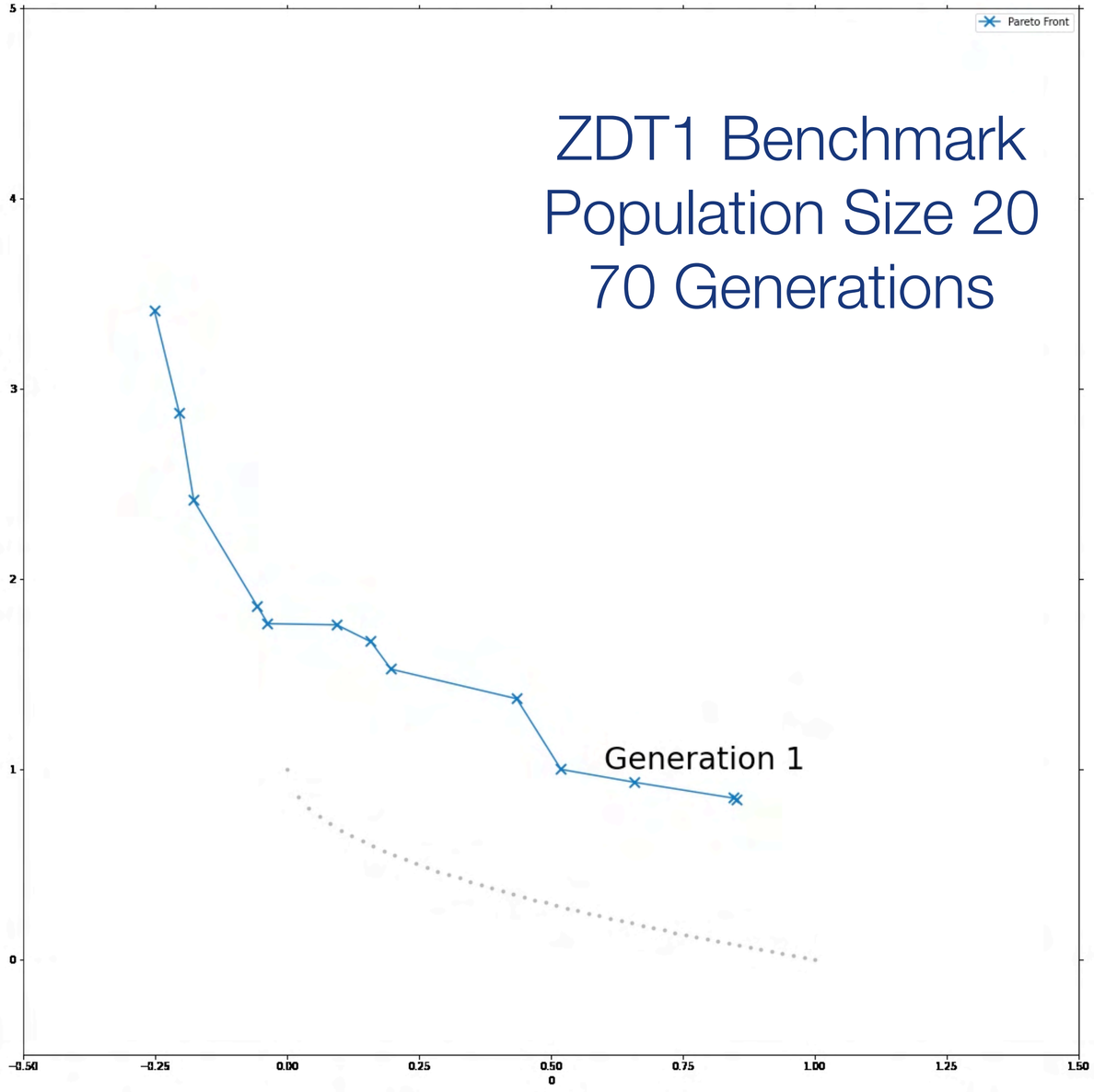
Noise Evaluation



Noisy Search-Based Testing (SBT)

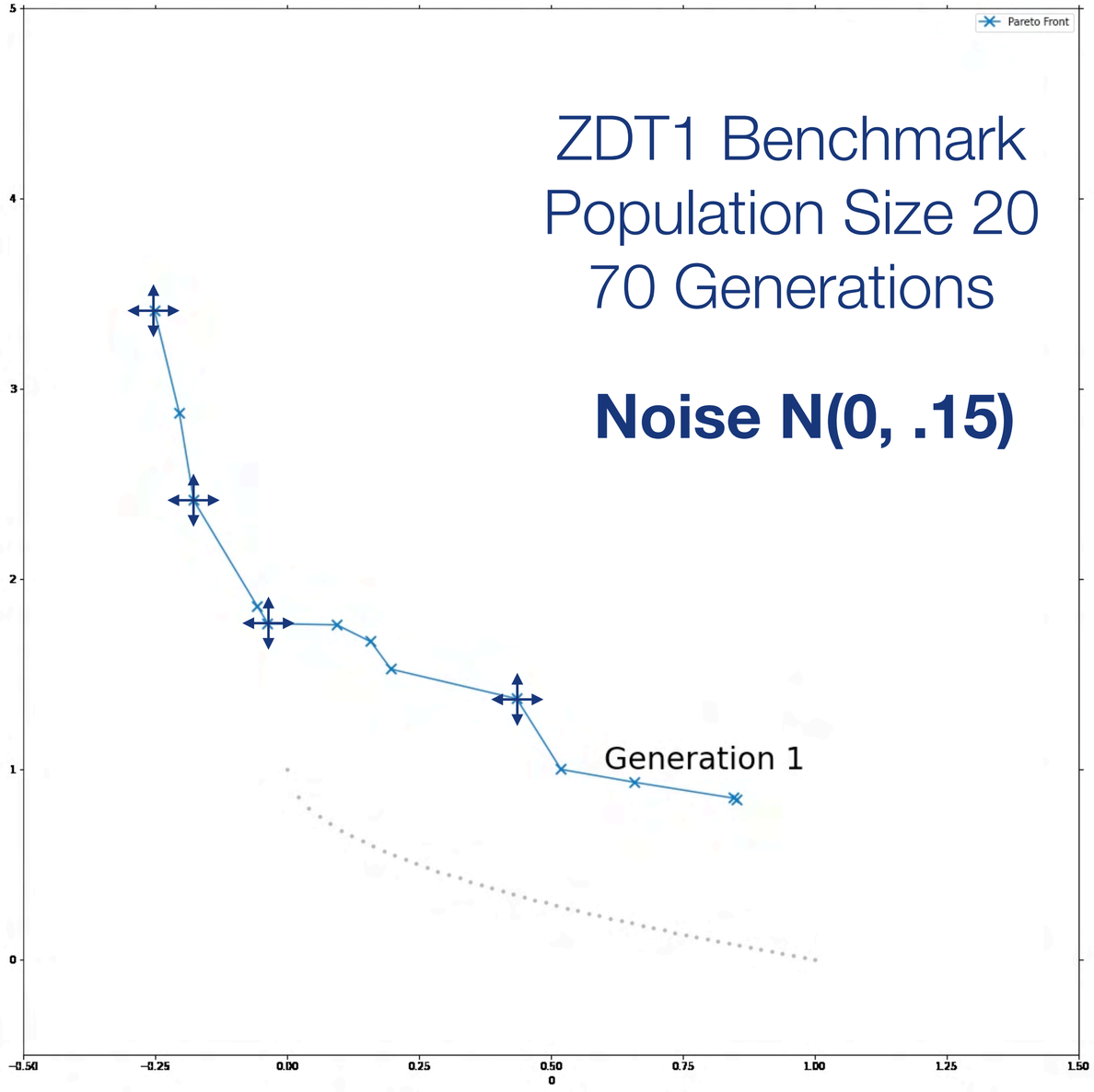


Noisy Search-Based Testing (SBT)



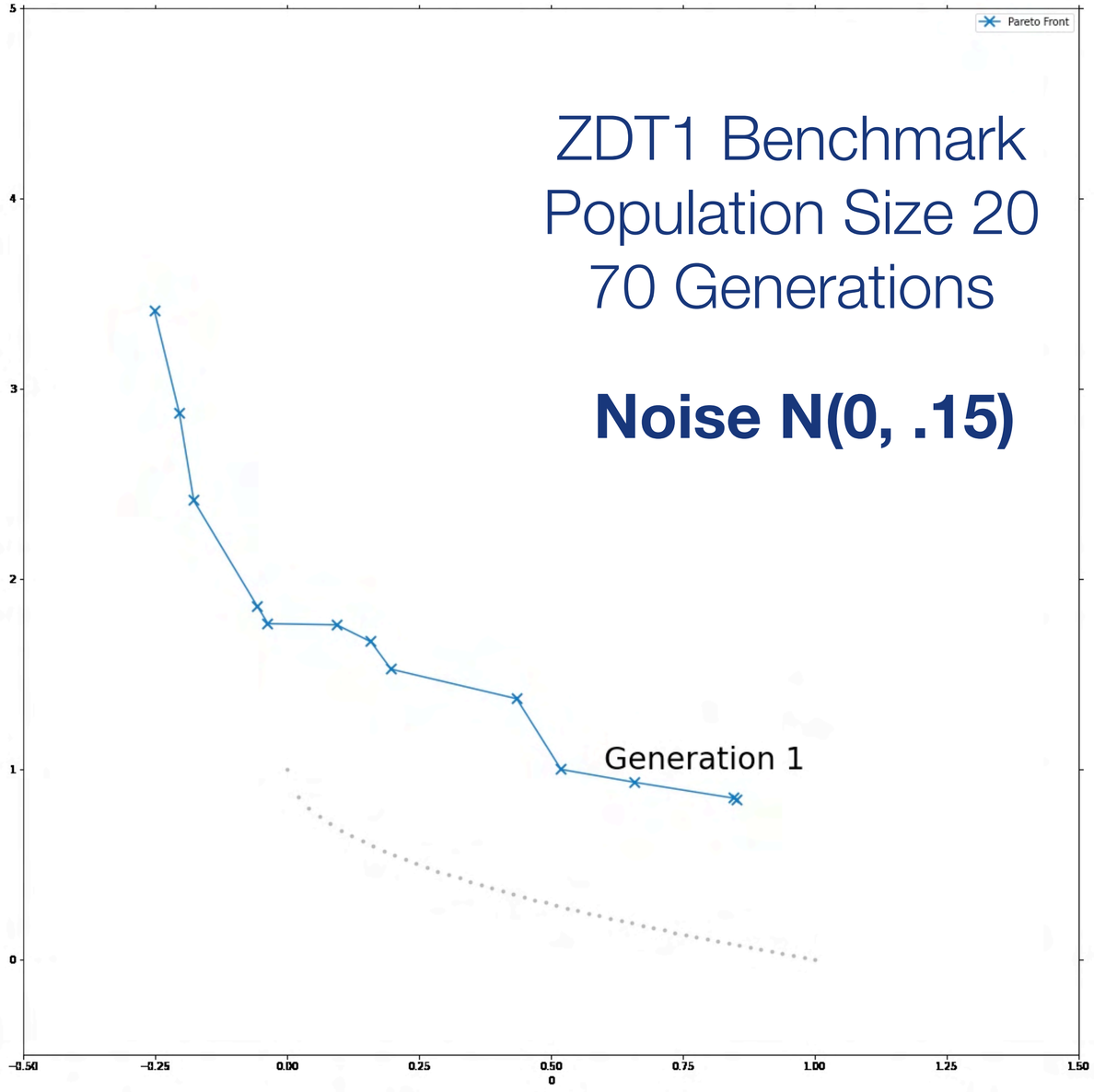
Objective Space

Noisy Search-Based Testing (SBT)



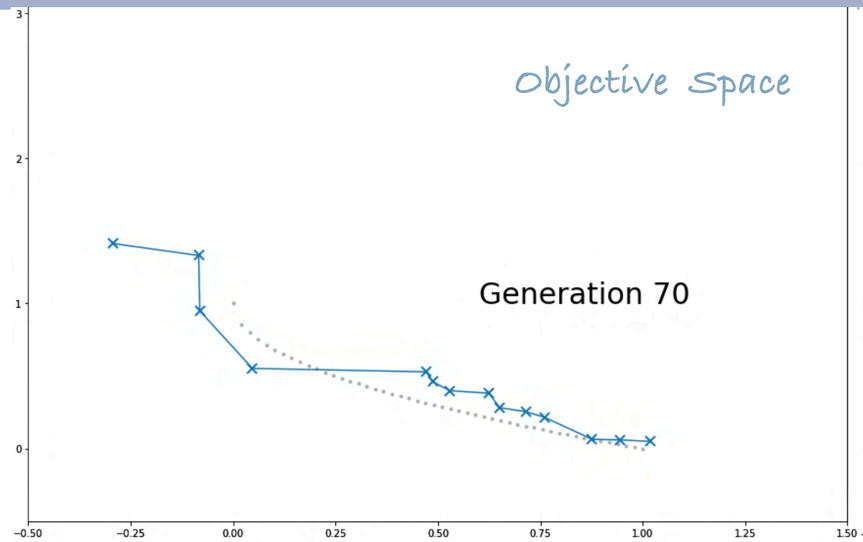
Objective Space

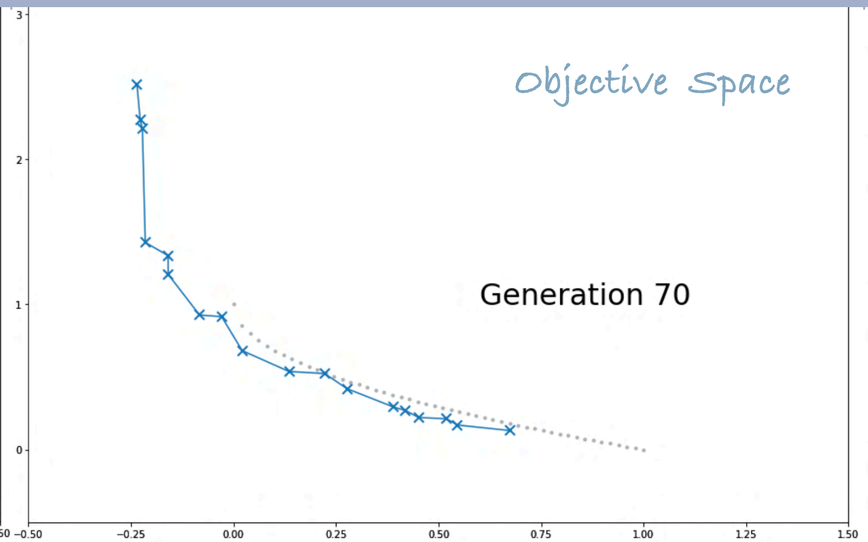
Noisy Search-Based Testing (SBT)

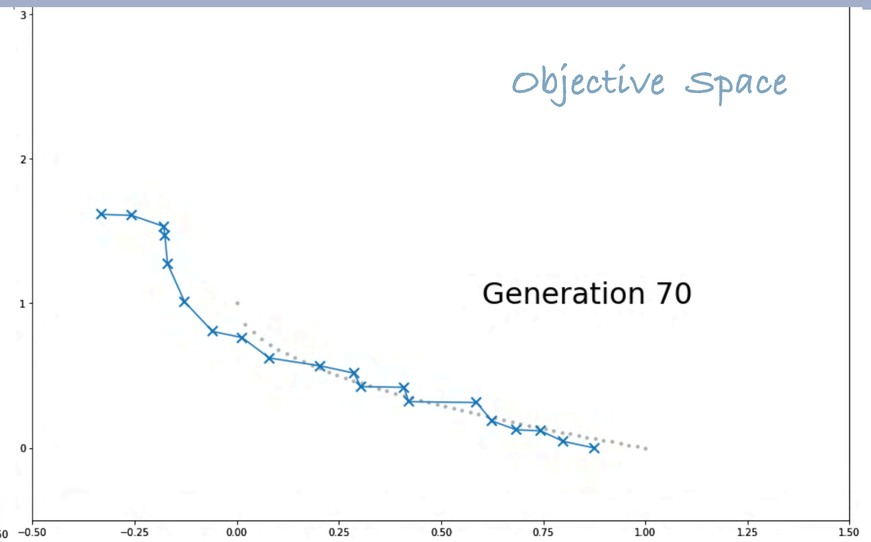


Objective Space

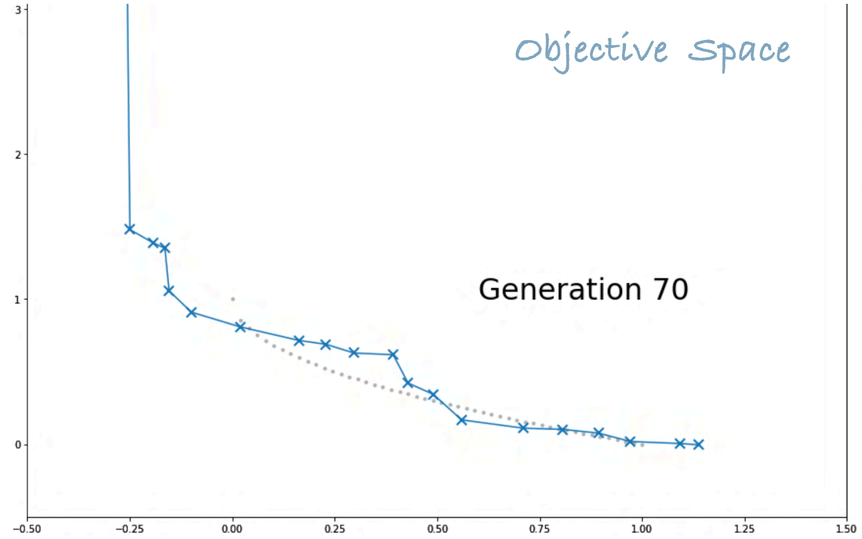
Noisy Search-Based Testing (SBT)

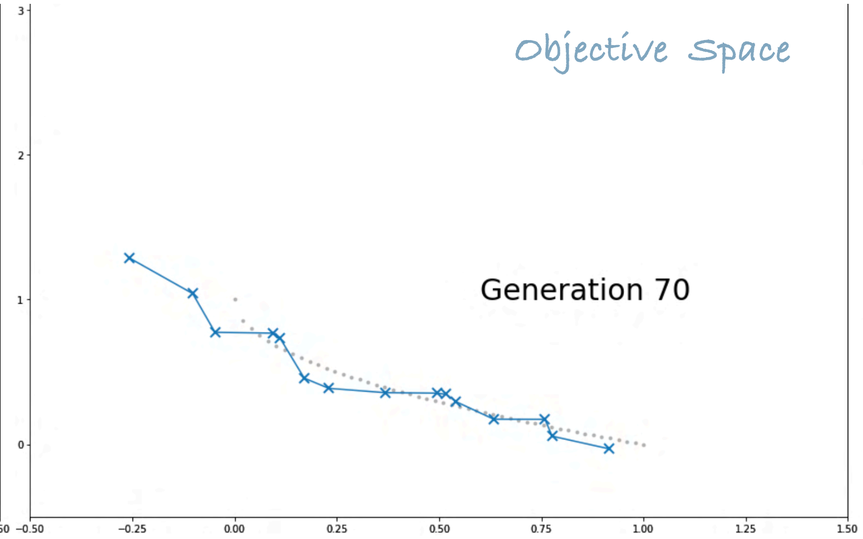


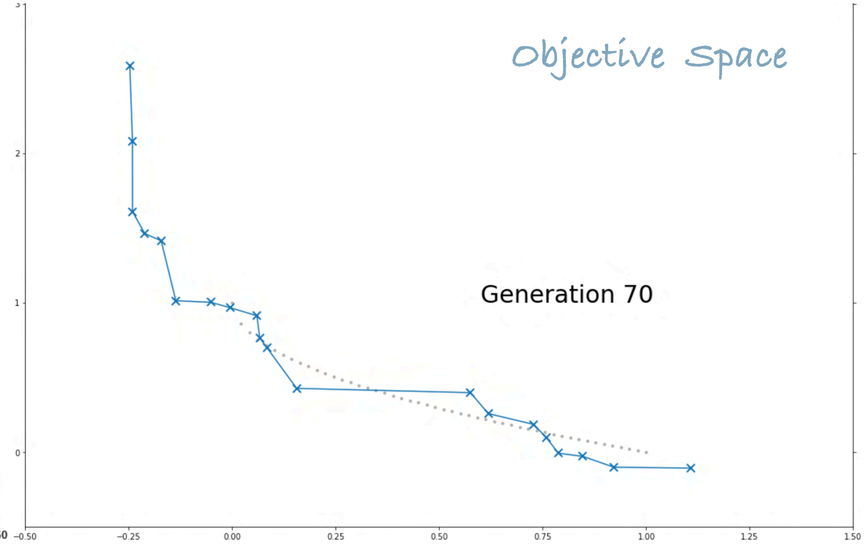




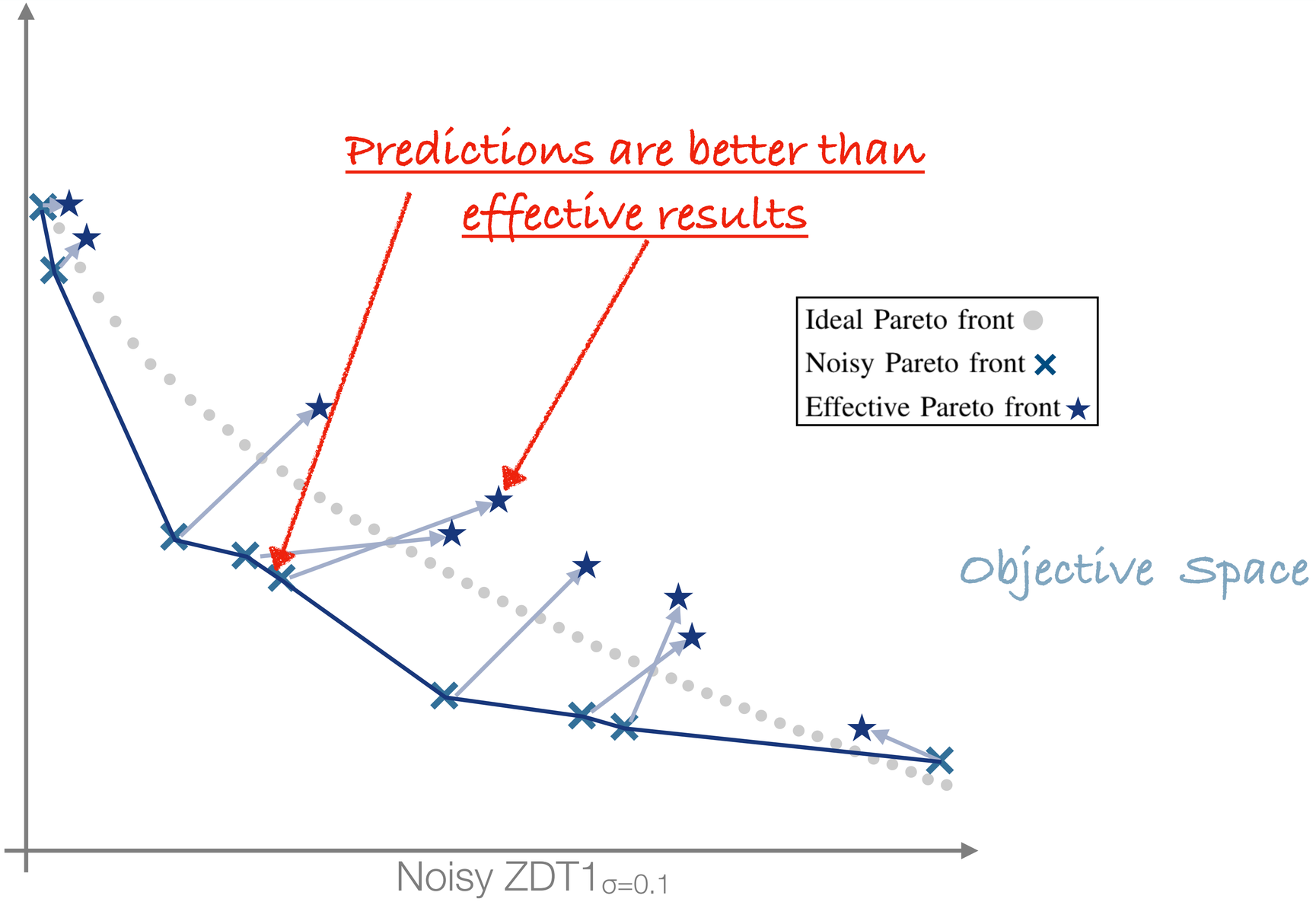
Repeat Experiments



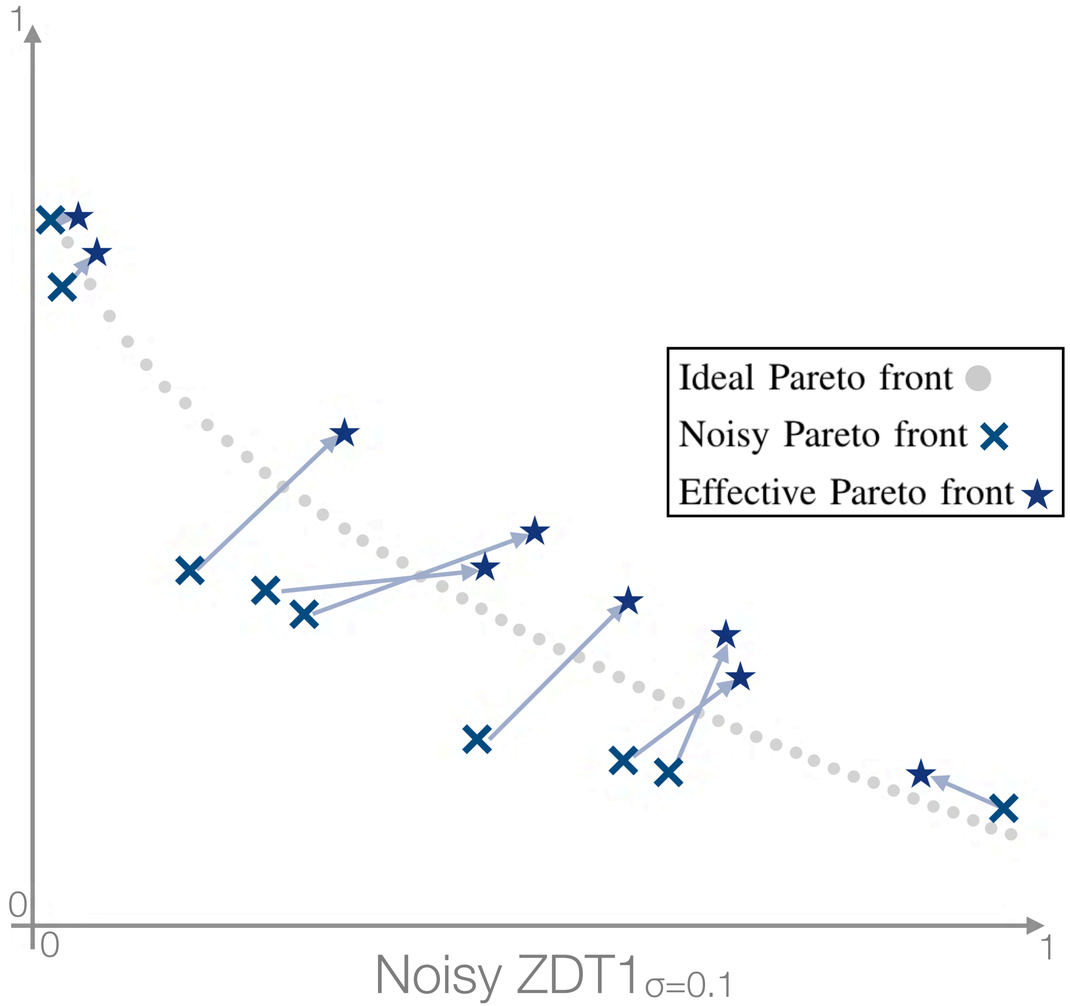




Noisy Search-Based Testing



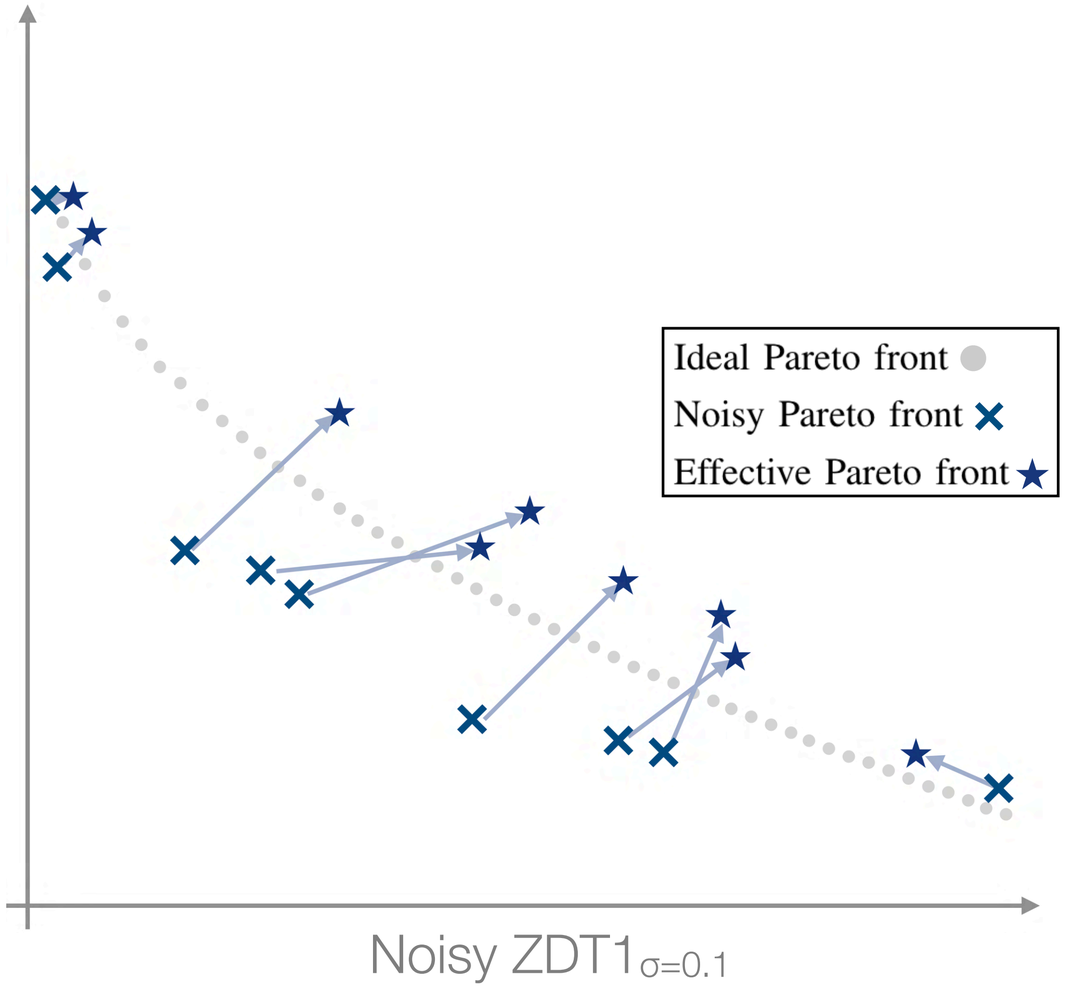
Noisy Search-Based Testing



Observations

1. (Noisy) PF surpasses Ideal PF
2. Repetition yields Effective PF
3. Some Effective PF values are dominated
4. PF (mostly) positively impacted by noise
5. PF values are extreme outliers (exceeding σ)
6. Noise hinders further improvement

Noisy Search-Based Testing



Problems

- **Selection of** non-representative **outliers**
- Yielded **objectives not reproducible** (not reliable/trustworthy)

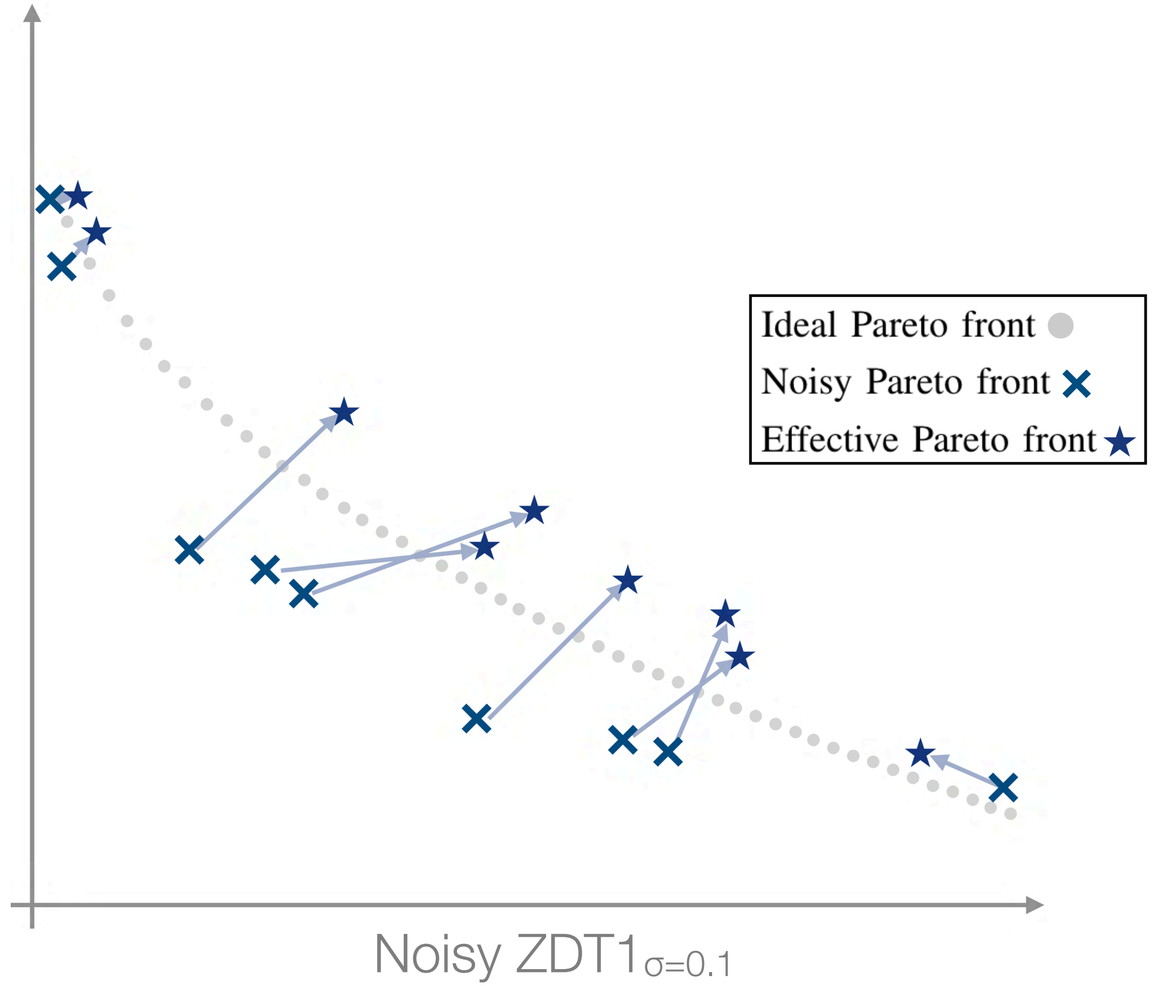
Research Goal

Our **goal** is to

*push the **Pareto Front** closer towards the **Effective Pareto Front**,*

but

*keep the **Effective Pareto Front** close to the **Ideal Pareto front**.*



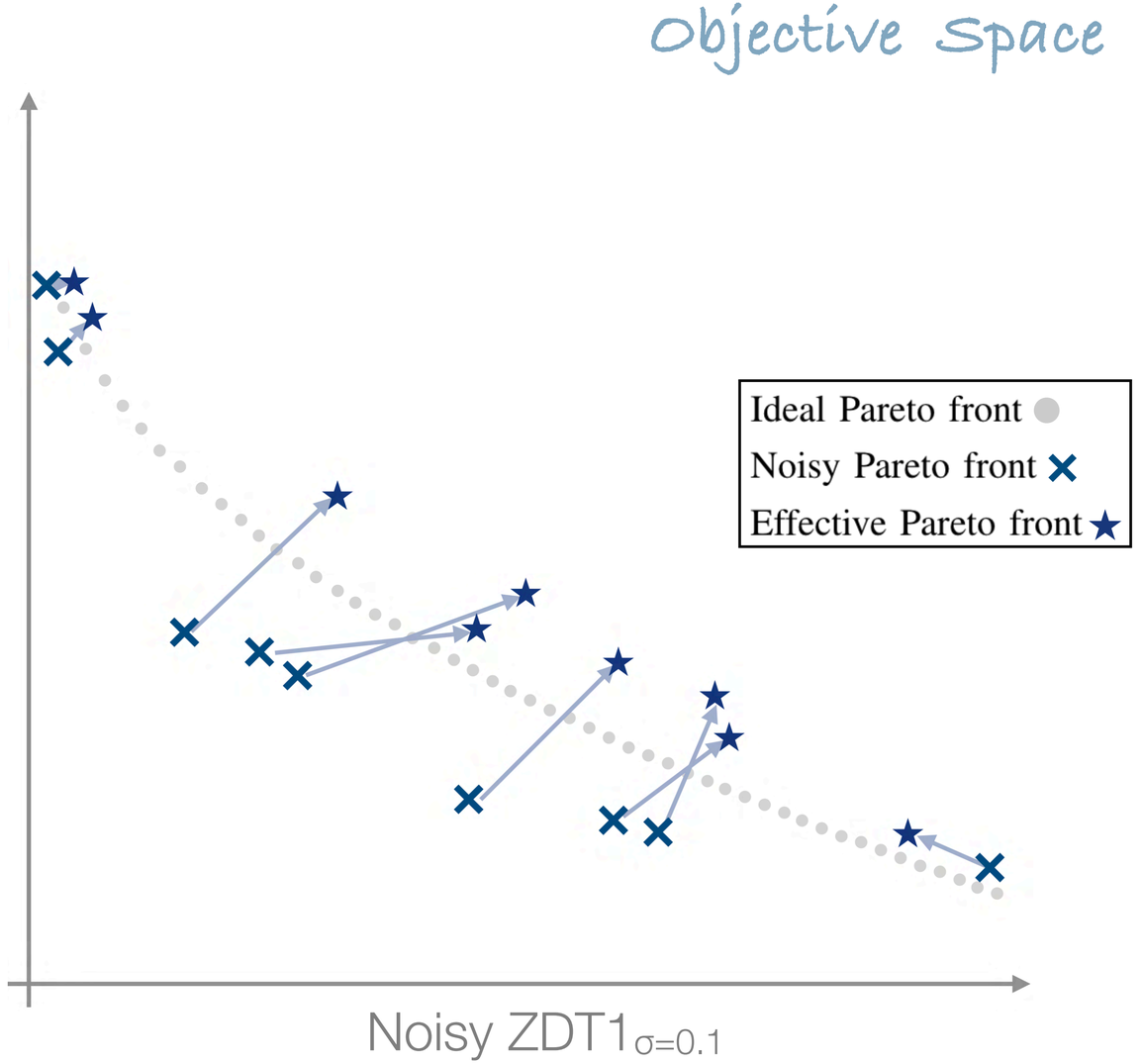
Research Goal

Our **goal** is to

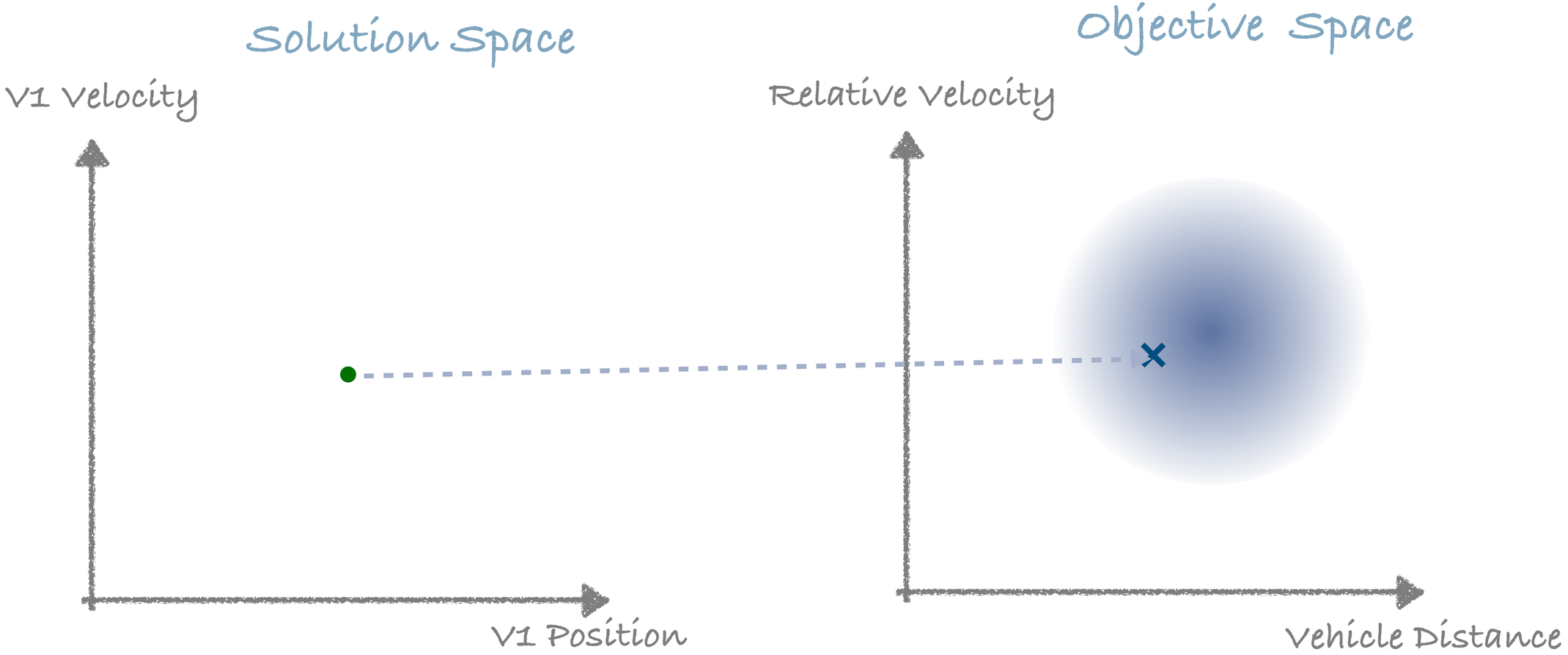
push the ✕ *closer towards the* ★,

but

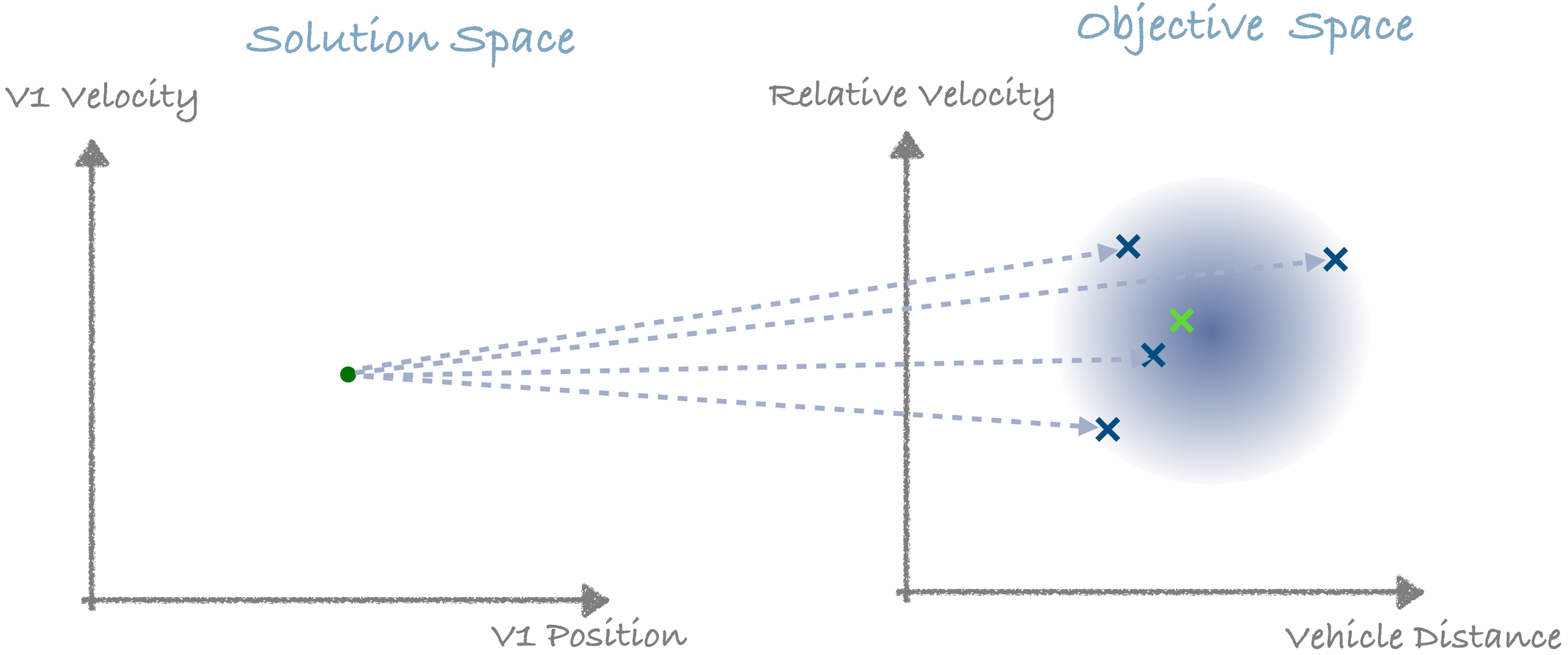
keep the ★ *close to the* ●.



Repetition to the Rescue



Repetition to the Rescue



But it is expensive...

System

(1 GPU + 60-120 sec) x

Search Settings

(10 pop size x 100 generations) =

15 - 30 hours per scenario optimisation

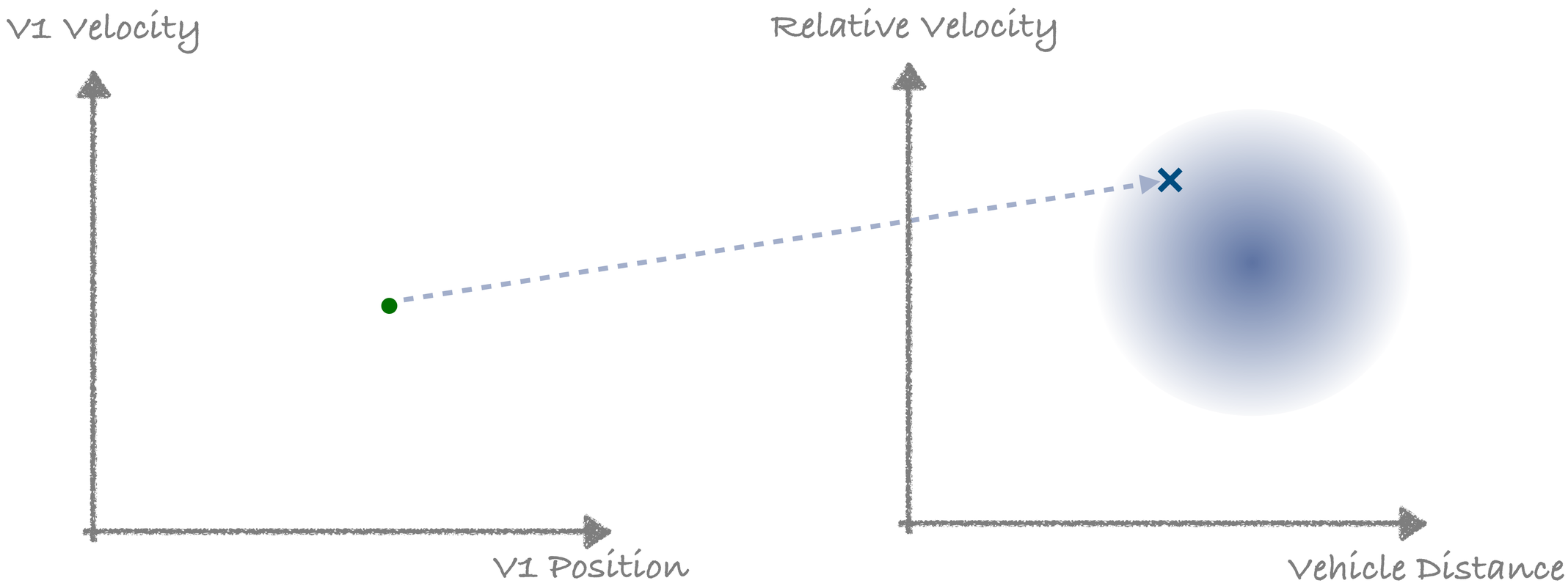
x 10 repetitions =

kNN-Averaging

a.k.a. *Let's repeat without repeating?*

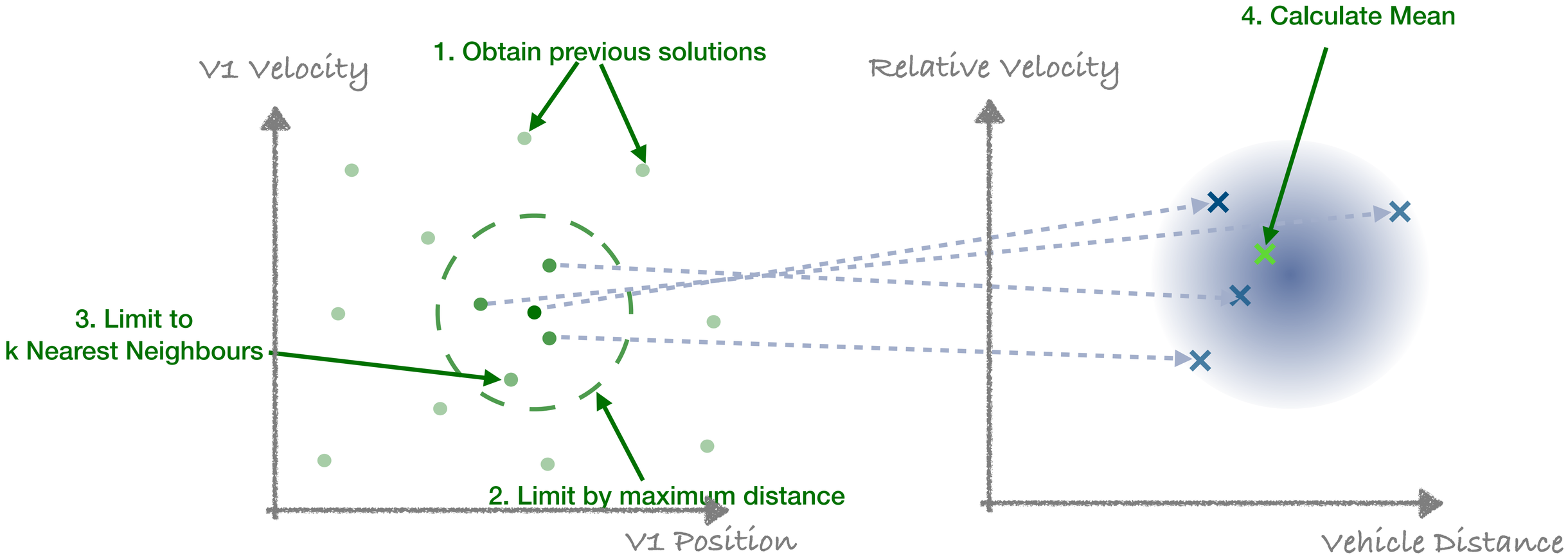
kNN-Averaging

Idea: Use Historical Values to Simulate Repetition



kNN-Averaging

Idea: Use Historical Values to Simulate Repetition



Open Questions (Hyper-Parameters)

How many neighbours do we need?

Intuition: Looking at more neighbours, we get a more representative value.

What's the size of the neighbourhood?

Intuition: In sparse spaces, do not use too distant neighbours.

Should close neighbours weigh more?

Intuition: Close neighbours are similar, similarity drops with distance.

Research Questions

RQ1: Can kNN-Avg reduce outlier-effect?

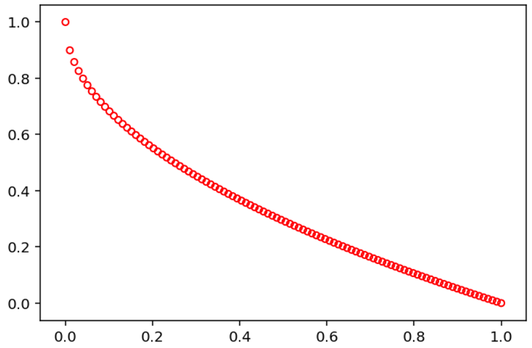
RQ2: Are solutions by kNN-Avg better/worse?

RQ3: Relation noise-level and kNN-Avg's efficiency

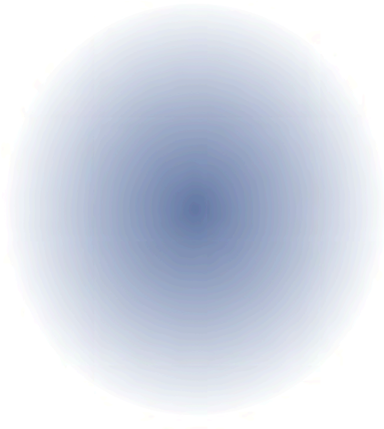
RQ4: Influence of hyper-parameters

Experimental Setup

Problem **Noise** **Evaluate** **Performance?**



+



Baseline

kNN-Avg

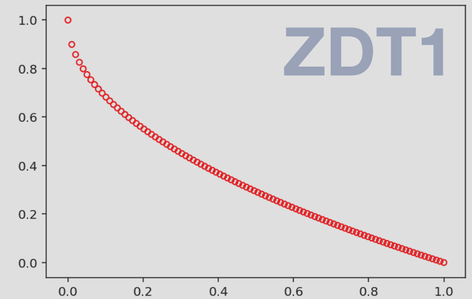
Implementation based on www.pymoo.org // NSGA-II

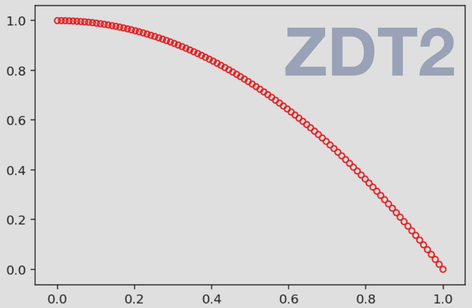
https://pymoo.org/problems/multi/zdt.html

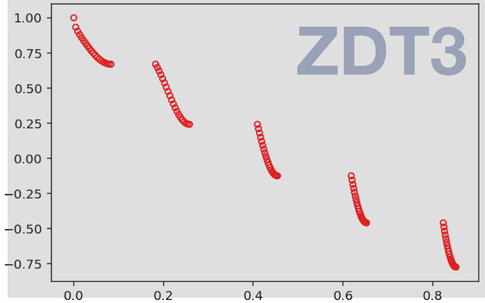
Experimental Setup

Problem

n_var ∈ [2, 4, 10]



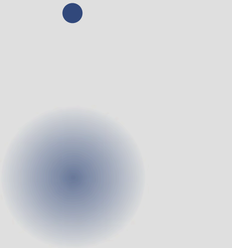




https://pymoo.org/problems/multi/zdt.html

Noise

σ ∈ [0, .05, .1, .25, .5]



Evaluate

Configurations:

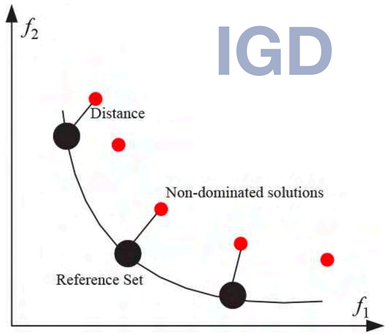
k ∈ [1,10,25,50, 100,1000]

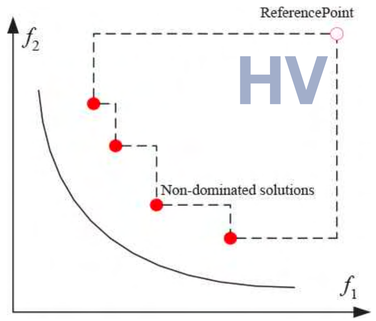
MD ∈ [.25, .5, 1, 2, 4]

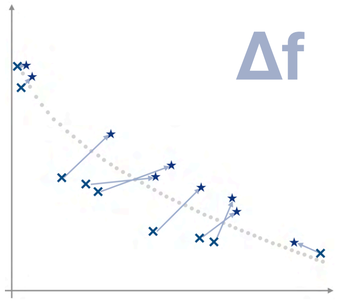
pop size ∈ [10,20]

100 generations

Performance?

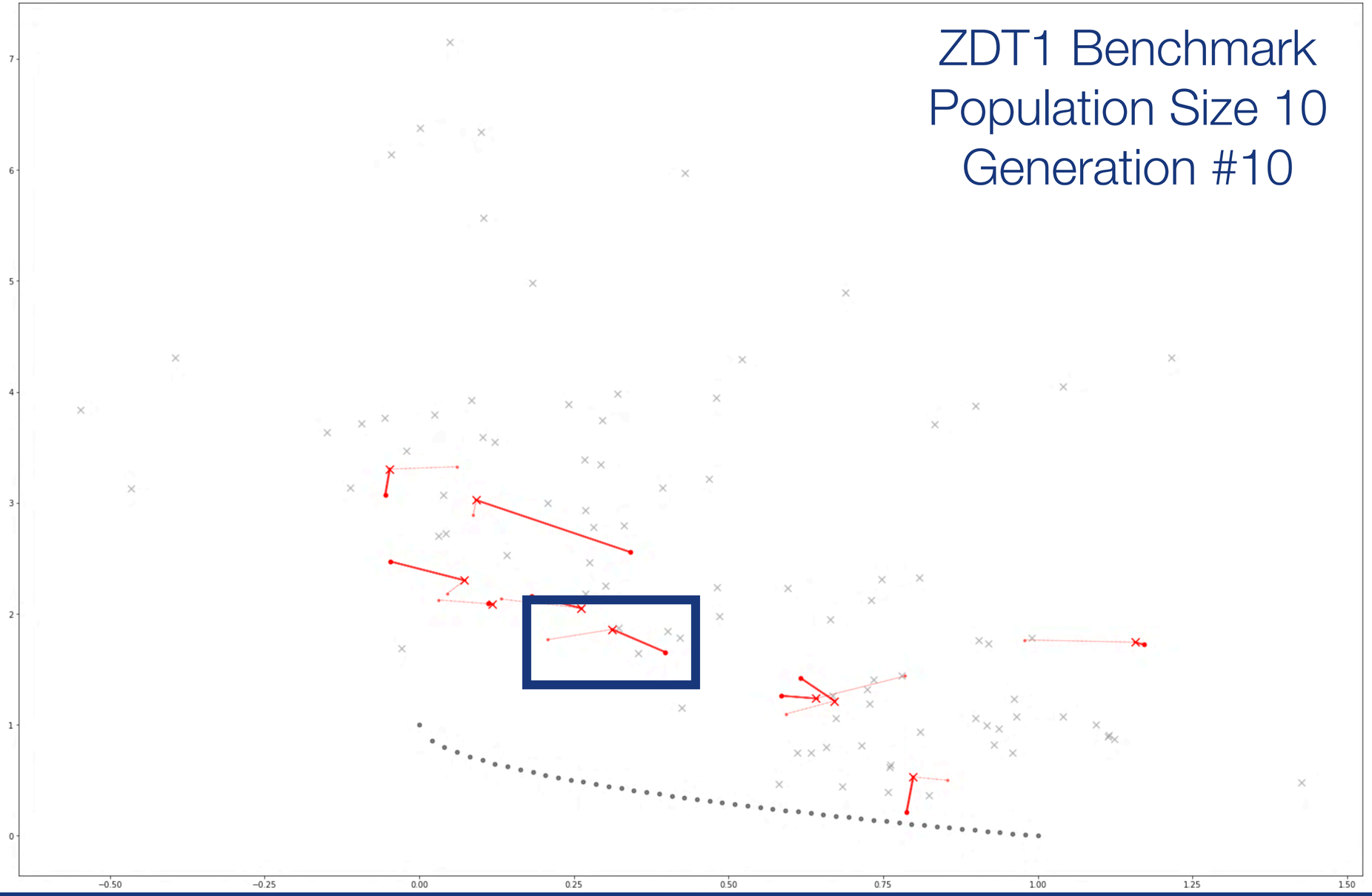


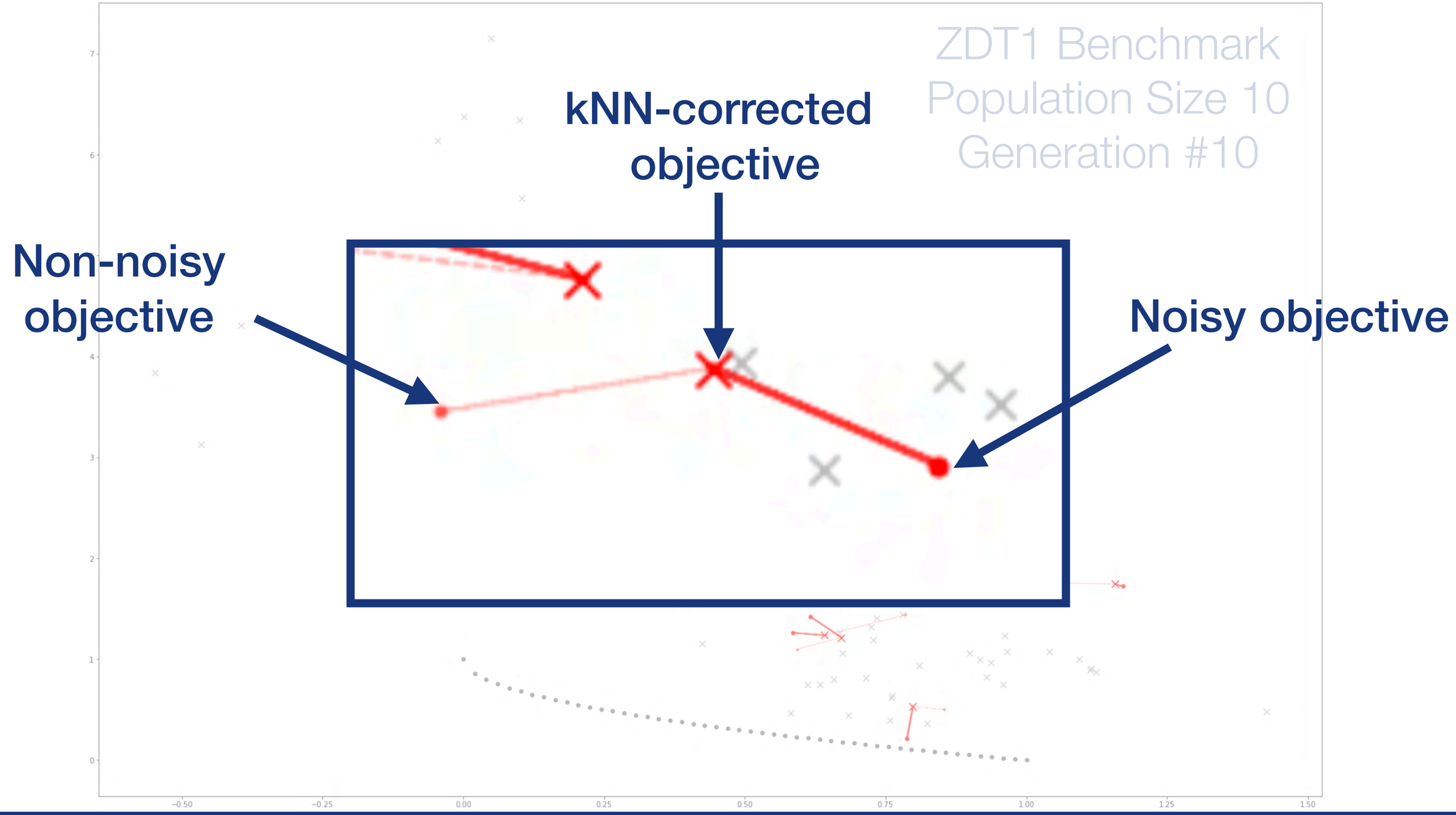




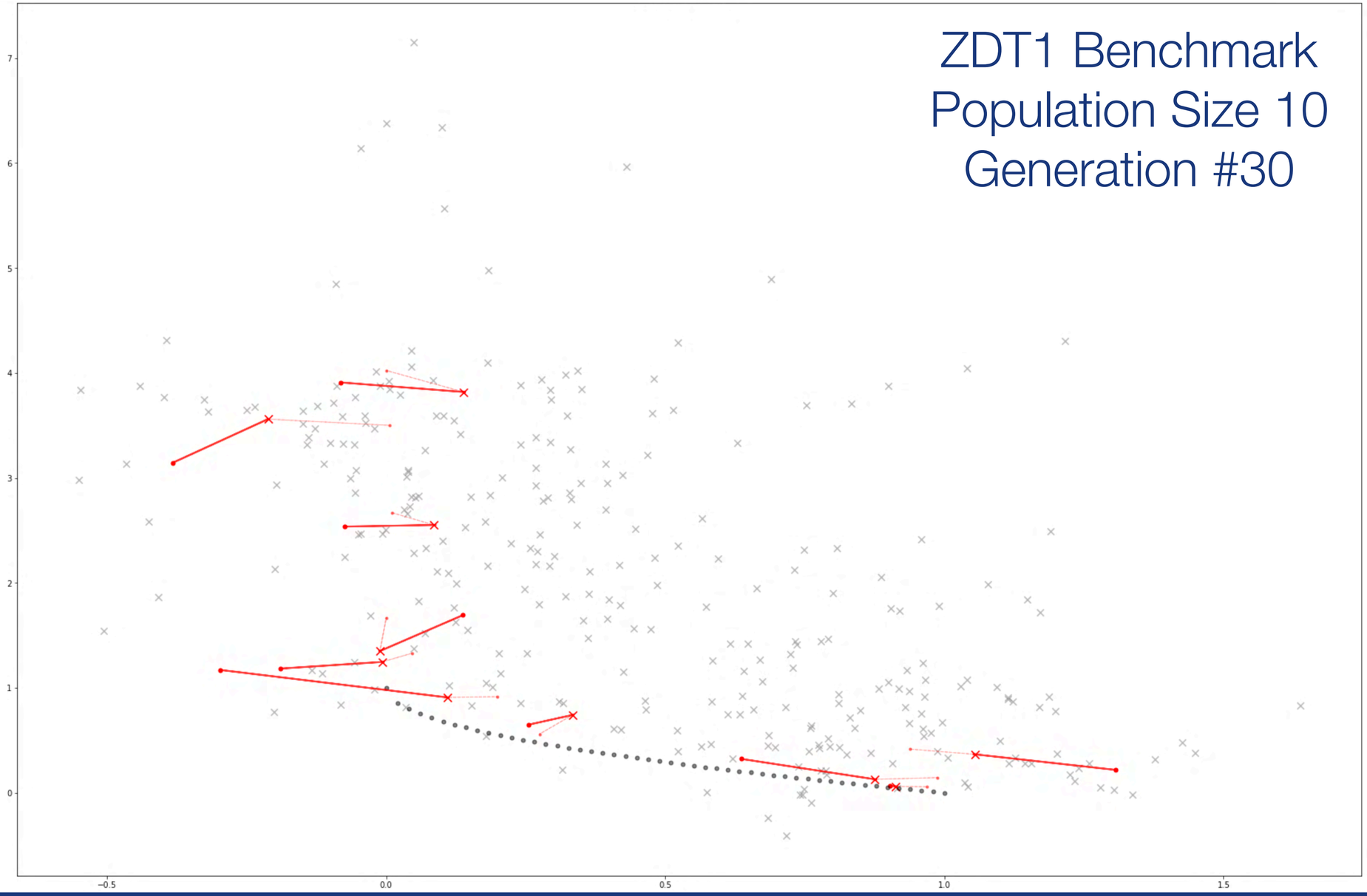
https://www.atlantis-press.com/journals/ijcis/125912233/view

kNN-Avg Effect

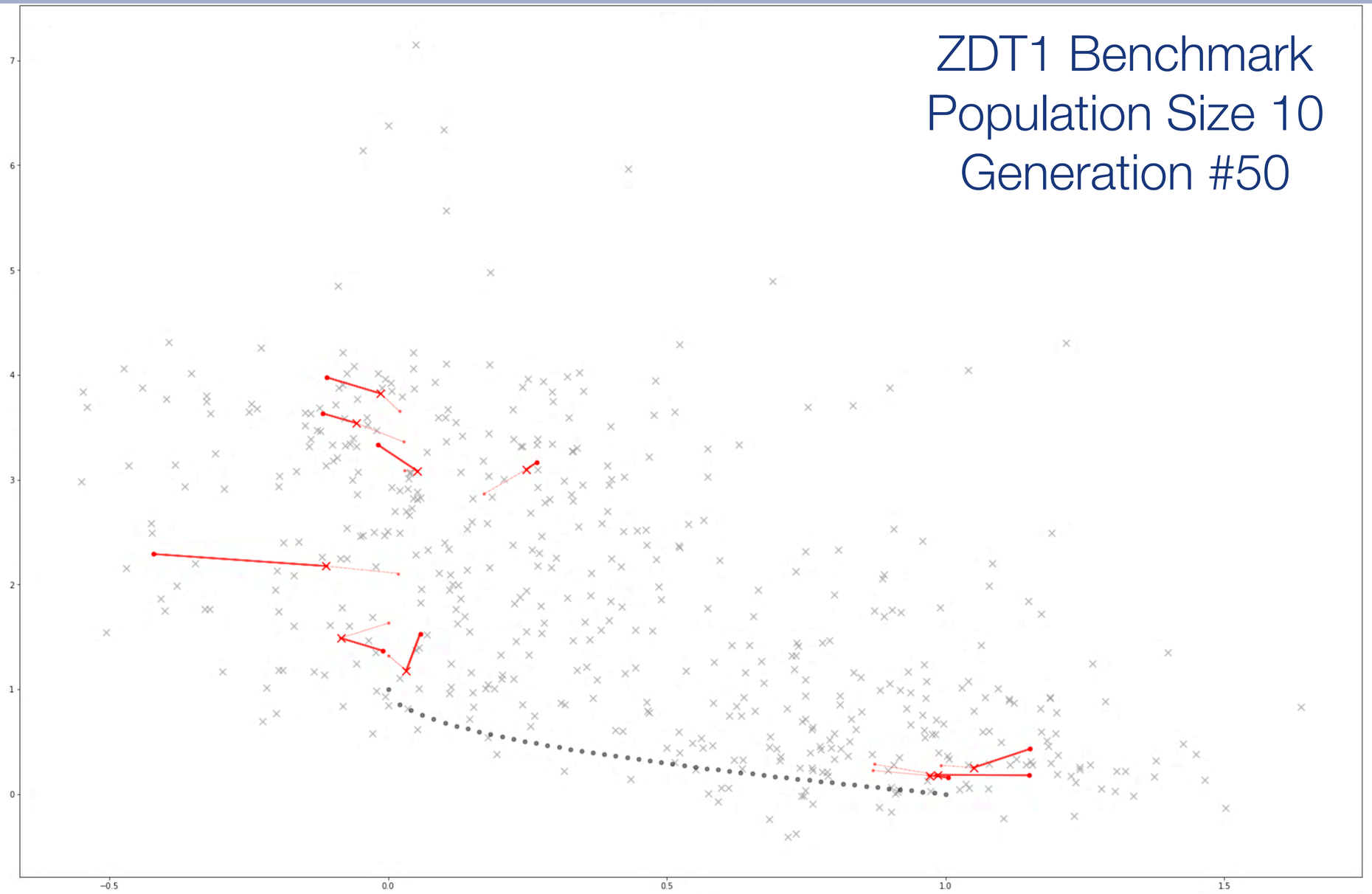



ZDT1 Benchmark
Population Size 10
Generation #10
kNN-corrected objective
Non-noisy objective
Noisy objective

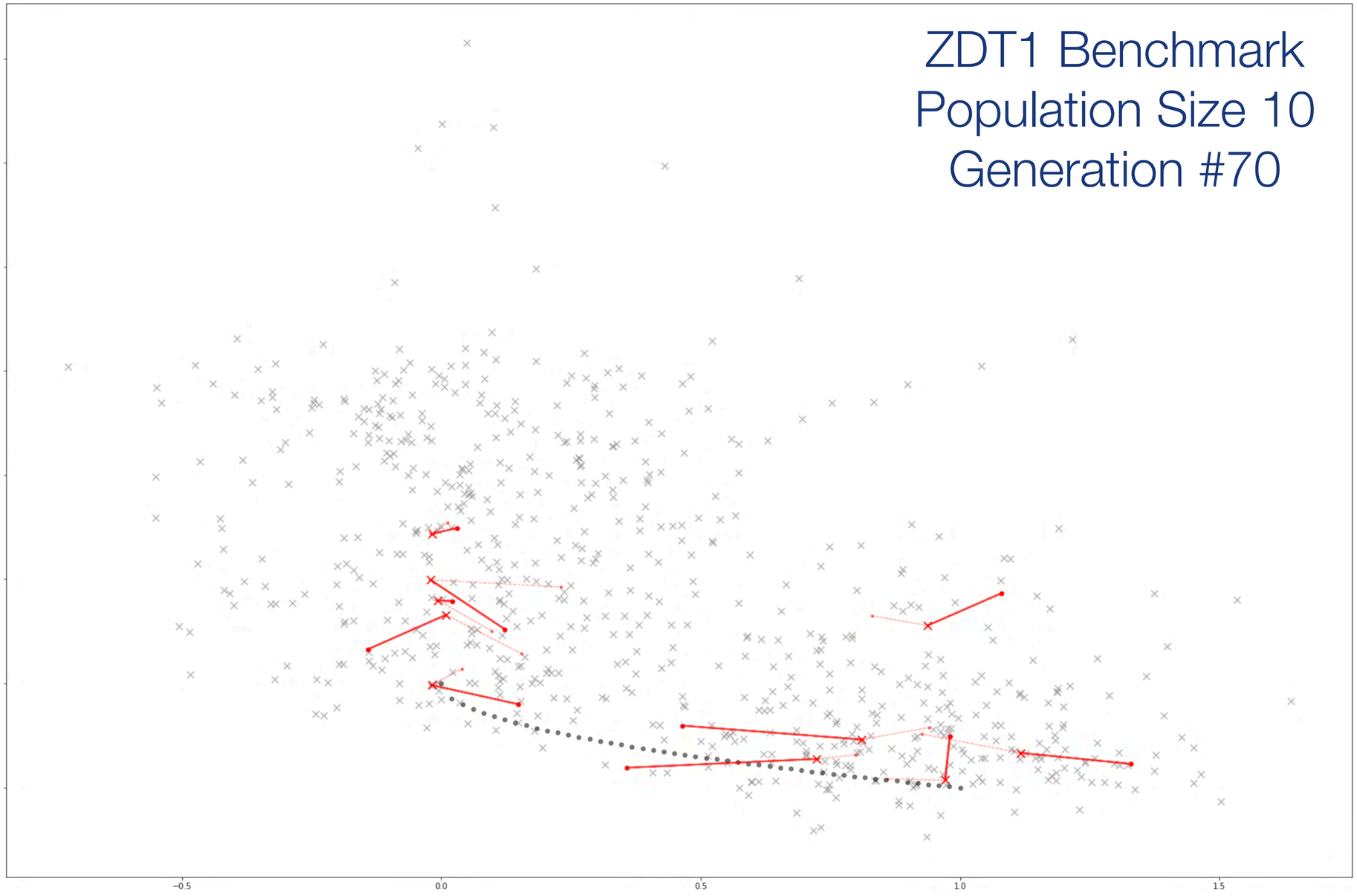
kNN-Avg Effect



kNN-Avg Effect

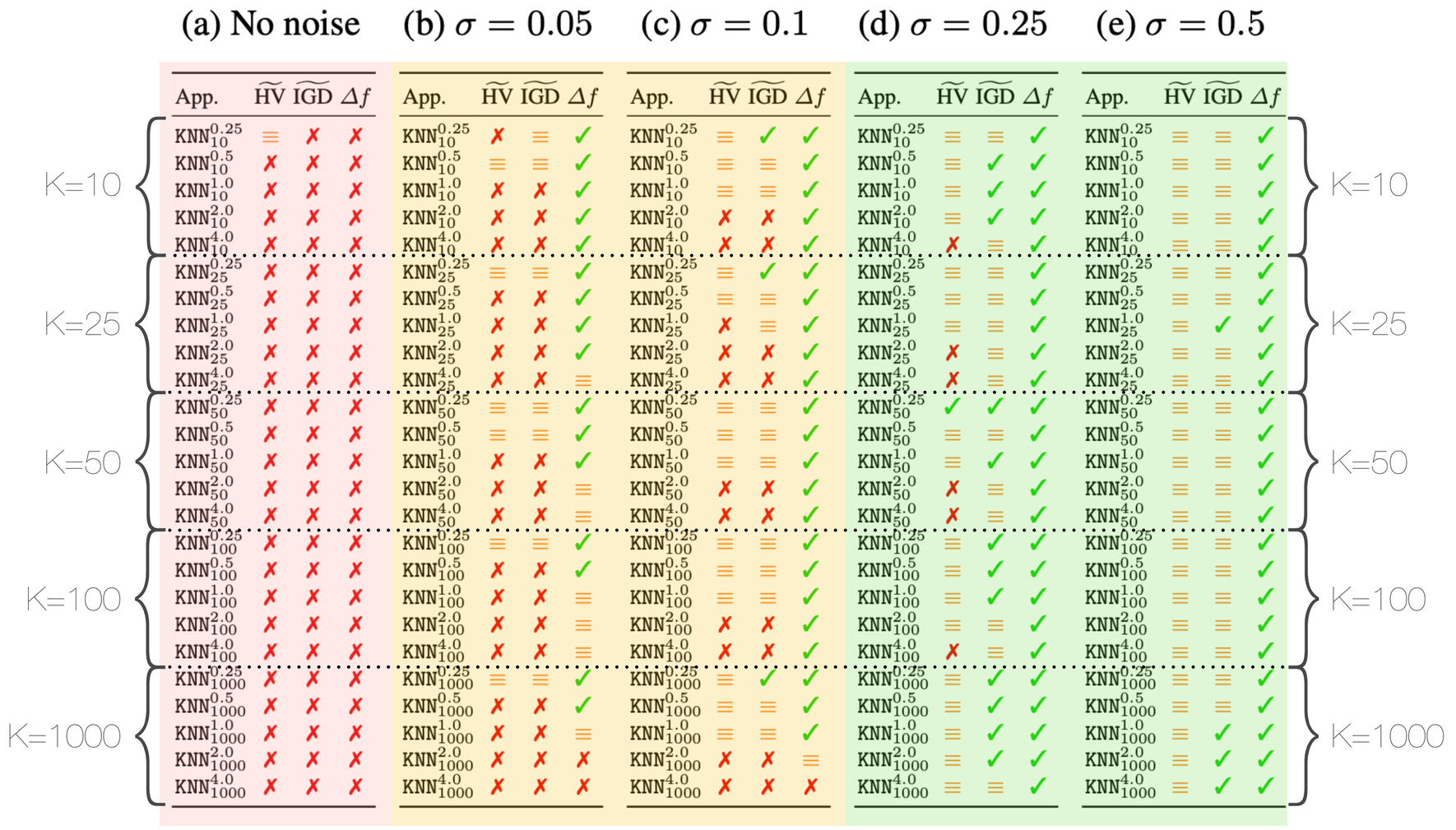


kNN-Avg Effect



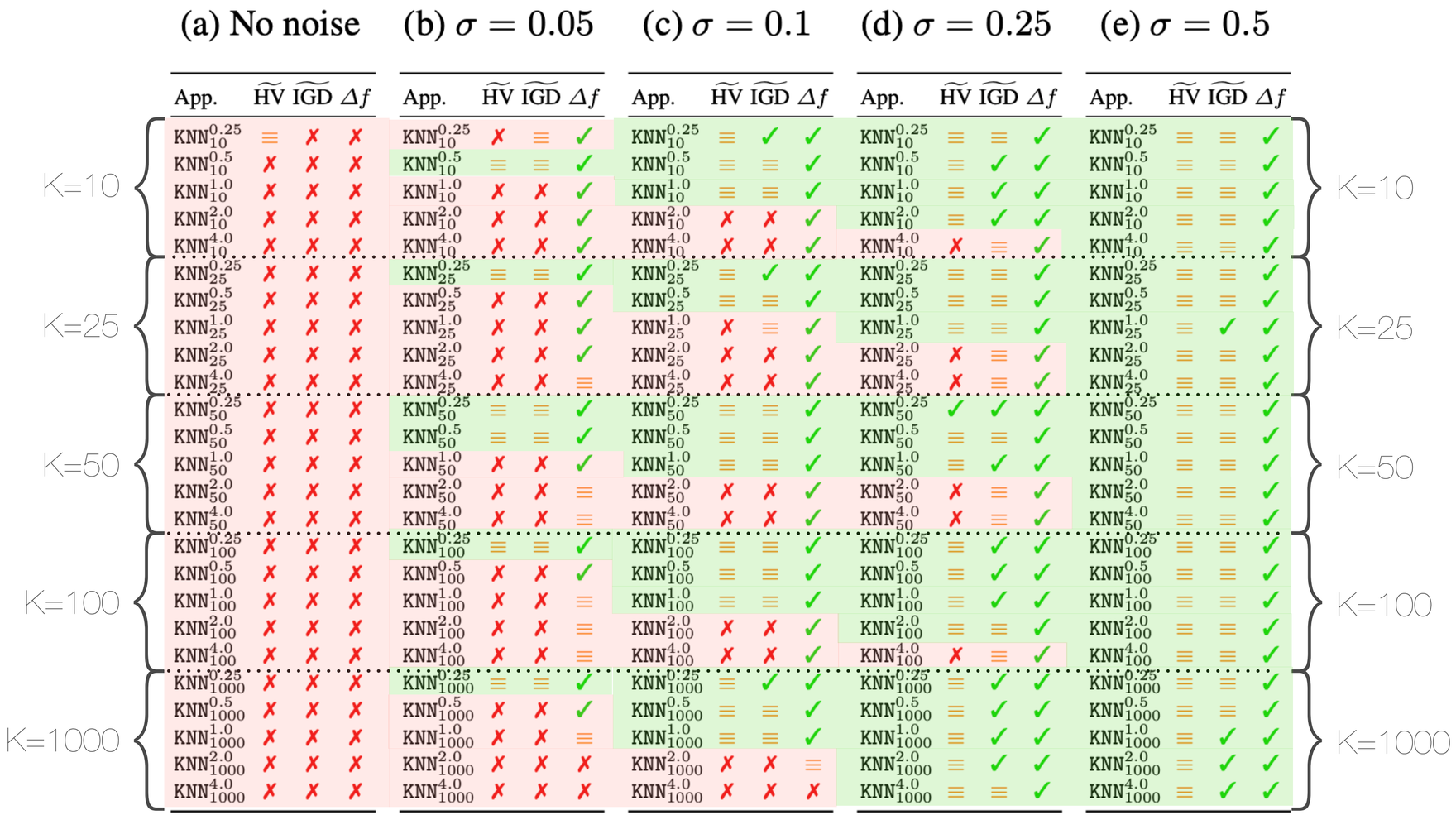
Results

KNN_k^{MD}



Hyper-Parameters

KNN_k^{MD}



	(a) No noise			(b) $\sigma = 0.05$			(c) $\sigma = 0.1$			(d) $\sigma = 0.25$			(e) $\sigma = 0.5$			
	App.	$\widetilde{HV}$	$\widetilde{IGD}$	Δf	App.	$\widetilde{HV}$	$\widetilde{IGD}$	Δf	App.	$\widetilde{HV}$	$\widetilde{IGD}$	Δf	App.	$\widetilde{HV}$	$\widetilde{IGD}$	Δf
K=10	$KNN_{10}^{0.25}$	≡	✗	✗	$KNN_{10}^{0.25}$	✗	≡	✓	$KNN_{10}^{0.25}$	≡	✓	✓	$KNN_{10}^{0.25}$	≡	≡	✓	$KNN_{10}^{0.25}$	≡	≡	✓
	$KNN_{10}^{0.5}$	✗	✗	✗	$KNN_{10}^{0.5}$	≡	≡	✓	$KNN_{10}^{0.5}$	≡	≡	✓	$KNN_{10}^{0.5}$	≡	✓	✓	$KNN_{10}^{0.5}$	≡	≡	✓
	$KNN_{10}^{1.0}$	✗	✗	✗	$KNN_{10}^{1.0}$	✗	✗	✓	$KNN_{10}^{1.0}$	≡	≡	✓	$KNN_{10}^{1.0}$	≡	✓	✓	$KNN_{10}^{1.0}$	≡	≡	✓
	$KNN_{10}^{2.0}$	✗	✗	✗	$KNN_{10}^{2.0}$	✗	✗	✓	$KNN_{10}^{2.0}$	✗	✗	✓	$KNN_{10}^{2.0}$	≡	✓	✓	$KNN_{10}^{2.0}$	≡	≡	✓
	$KNN_{10}^{4.0}$	✗	✗	✗	$KNN_{10}^{4.0}$	✗	✗	✓	$KNN_{10}^{4.0}$	✗	✗	✓	$KNN_{10}^{4.0}$	✗	≡	✓	$KNN_{10}^{4.0}$	≡	≡	✓
K=25	$KNN_{25}^{0.25}$	✗	✗	✗	$KNN_{25}^{0.25}$	≡	≡	✓	$KNN_{25}^{0.25}$	≡	✓	✓	$KNN_{25}^{0.25}$	≡	≡	✓	$KNN_{25}^{0.25}$	≡	≡	✓
	$KNN_{25}^{0.5}$	✗	✗	✗	$KNN_{25}^{0.5}$	✗	✗	✓	$KNN_{25}^{0.5}$	≡	≡	✓	$KNN_{25}^{0.5}$	≡	≡	✓	$KNN_{25}^{0.5}$	≡	≡	✓
	$KNN_{25}^{1.0}$	✗	✗	✗	$KNN_{25}^{1.0}$	✗	✗	✓	$KNN_{25}^{1.0}$	✗	≡	✓	$KNN_{25}^{1.0}$	≡	≡	✓	$KNN_{25}^{1.0}$	≡	✓	✓
	$KNN_{25}^{2.0}$	✗	✗	✗	$KNN_{25}^{2.0}$	✗	✗	✓	$KNN_{25}^{2.0}$	✗	✗	✓	$KNN_{25}^{2.0}$	✗	≡	✓	$KNN_{25}^{2.0}$	≡	≡	✓
	$KNN_{25}^{4.0}$	✗	✗	✗	$KNN_{25}^{4.0}$	✗	✗	≡	$KNN_{25}^{4.0}$	✗	✗	✓	$KNN_{25}^{4.0}$	✗	≡	✓	$KNN_{25}^{4.0}$	≡	≡	✓
K=50	$KNN_{50}^{0.25}$	✗	✗	✗	$KNN_{50}^{0.25}$	≡	≡	✓	$KNN_{50}^{0.25}$	≡	≡	✓	$KNN_{50}^{0.25}$	✓	✓	✓	$KNN_{50}^{0.25}$	≡	≡	✓
	$KNN_{50}^{0.5}$	✗	✗	✗	$KNN_{50}^{0.5}$	≡	≡	✓	$KNN_{50}^{0.5}$	≡	≡	✓	$KNN_{50}^{0.5}$	≡	≡	✓	$KNN_{50}^{0.5}$	≡	≡	✓
	$KNN_{50}^{1.0}$	✗	✗	✗	$KNN_{50}^{1.0}$	✗	✗	✓	$KNN_{50}^{1.0}$	≡	≡	✓	$KNN_{50}^{1.0}$	≡	✓	✓	$KNN_{50}^{1.0}$	≡	≡	✓
	$KNN_{50}^{2.0}$	✗	✗	✗	$KNN_{50}^{2.0}$	✗	✗	≡	$KNN_{50}^{2.0}$	✗	✗	✓	$KNN_{50}^{2.0}$	✗	≡	✓	$KNN_{50}^{2.0}$	≡	≡	✓
	$KNN_{50}^{4.0}$	✗	✗	✗	$KNN_{50}^{4.0}$	✗	✗	≡	$KNN_{50}^{4.0}$	✗	✗	✓	$KNN_{50}^{4.0}$	✗	≡	✓	$KNN_{50}^{4.0}$	≡	≡	✓
K=100	$KNN_{100}^{0.25}$	✗	✗	✗	$KNN_{100}^{0.25}$	≡	≡	✓	$KNN_{100}^{0.25}$	≡	≡	✓	$KNN_{100}^{0.25}$	≡	✓	✓	$KNN_{100}^{0.25}$	≡	≡	✓
	$KNN_{100}^{0.5}$	✗	✗	✗	$KNN_{100}^{0.5}$	✗	✗	✓	$KNN_{100}^{0.5}$	≡	≡	✓	$KNN_{100}^{0.5}$	≡	✓	✓	$KNN_{100}^{0.5}$	≡	≡	✓
	$KNN_{100}^{1.0}$	✗	✗	✗	$KNN_{100}^{1.0}$	✗	✗	≡	$KNN_{100}^{1.0}$	≡	≡	✓	$KNN_{100}^{1.0}$	≡	✓	✓	$KNN_{100}^{1.0}$	≡	≡	✓
	$KNN_{100}^{2.0}$	✗	✗	✗	$KNN_{100}^{2.0}$	✗	✗	≡	$KNN_{100}^{2.0}$	✗	✗	✓	$KNN_{100}^{2.0}$	≡	≡	✓	$KNN_{100}^{2.0}$	≡	≡	✓
	$KNN_{100}^{4.0}$	✗	✗	✗	$KNN_{100}^{4.0}$	✗	✗	≡	$KNN_{100}^{4.0}$	✗	✗	✓	$KNN_{100}^{4.0}$	✗	≡	✓	$KNN_{100}^{4.0}$	≡	≡	✓
K=1000	$KNN_{1000}^{0.25}$	✗	✗	✗	$KNN_{1000}^{0.25}$	≡	≡	✓	$KNN_{1000}^{0.25}$	≡	✓	✓	$KNN_{1000}^{0.25}$	≡	✓	✓	$KNN_{1000}^{0.25}$	≡	≡	✓
	$KNN_{1000}^{0.5}$	✗	✗	✗	$KNN_{1000}^{0.5}$	✗	✗	✓	$KNN_{1000}^{0.5}$	≡	≡	✓	$KNN_{1000}^{0.5}$	≡	✓	✓	$KNN_{1000}^{0.5}$	≡	≡	✓
	$KNN_{1000}^{1.0}$	✗	✗	✗	$KNN_{1000}^{1.0}$	✗	✗	≡	$KNN_{1000}^{1.0}$	≡	≡	✓	$KNN_{1000}^{1.0}$	≡	✓	✓	$KNN_{1000}^{1.0}$	≡	✓	✓
	$KNN_{1000}^{2.0}$	✗	✗	✗	$KNN_{1000}^{2.0}$	✗	✗	✗	$KNN_{1000}^{2.0}$	✗	✗	≡	$KNN_{1000}^{2.0}$	≡	✓	✓	$KNN_{1000}^{2.0}$	≡	✓	✓
	$KNN_{1000}^{4.0}$	✗	✗	✗	$KNN_{1000}^{4.0}$	✗	✗	✗	$KNN_{1000}^{4.0}$	✗	✗	✗	$KNN_{1000}^{4.0}$	≡	≡	✓	$KNN_{1000}^{4.0}$	≡	✓	✓

Discussion

RQ1: Can kNN-Avg reduce outlier-effect?

kNN-Avg reduces the outlier effect

RQ2: Are solutions by kNN-Avg better/worse?

kNN-Avg tends to be slightly worse or equal generally

RQ3: Relation noise-level and kNN-Avg's efficiency

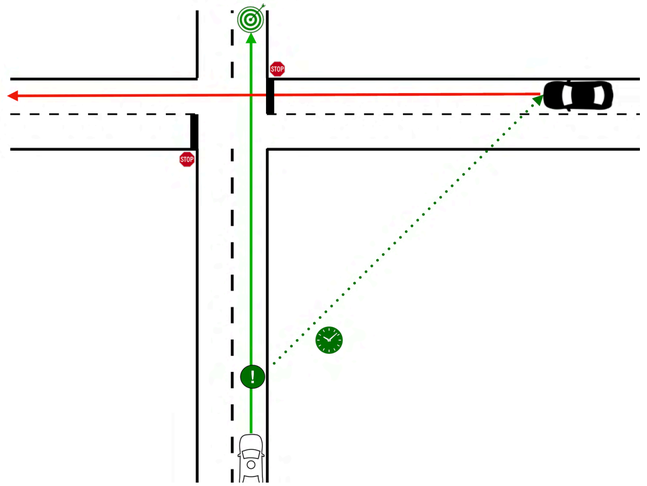
kNN-Avg performs better with higher noise, equal / worse on low noise

RQ4: Influence of hyper-parameters

***k* seems not very significant, smaller MD yields better results**

Future

XinSheYang02	2		“go-xinsheyang02”
XinSheYang03	2		“go-xinsheyang03”
XinSheYang04	2		“go-xinsheyang04”
Xor	9		“go-xor”
YaoLiu04	2		“go-yaoliu04”
YaoLiu09	2		“go-yaoliu09”
Zacharov	2		“go-zacharov”
ZeroSum	2		“go-zerosum”
Zettl	2		“go-zettl”
Zimmerman	2		“go-zimmerman”
Zirilli	2		“go-zirilli”

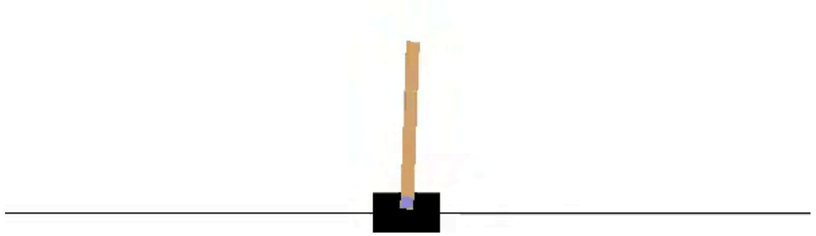


[Klikovits, Arcaini, PRDC'21]

kNN-Averaging

pymc

Noisy Controls

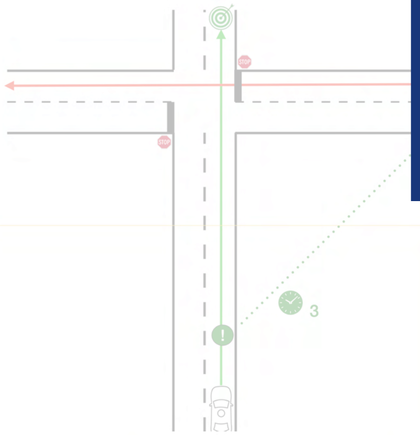


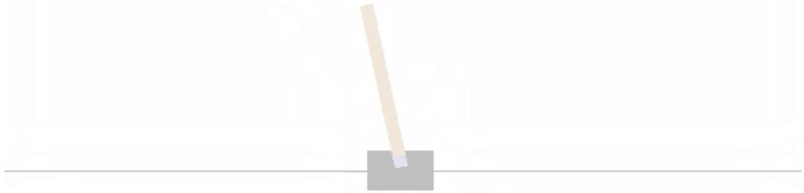
Future

XinSheYang02	2		“go-xinsheyang02”
XinSheYang03	2		“go-xinsheyang03”
XinSheYang04	2		“go-xinsheyang04”
Xor	9		“go-xor”
YaoLiu04	2		“go-yaoliu04”
YaoLiu09	2		“go-yaoliu09”
Zacharov	2		
ZeroSum	2		
Zettl	2		
Zimmerman	2		
Zirilli	2		

kNN-Averaging

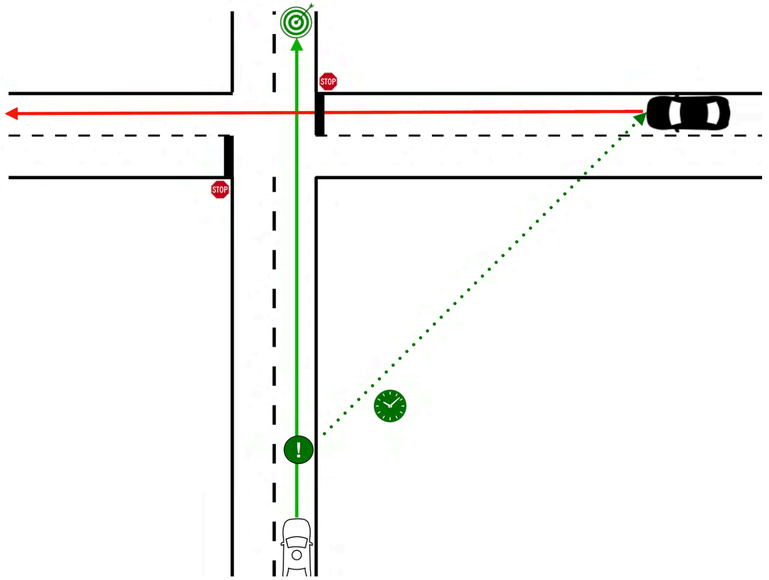
Let's try your Noisy System

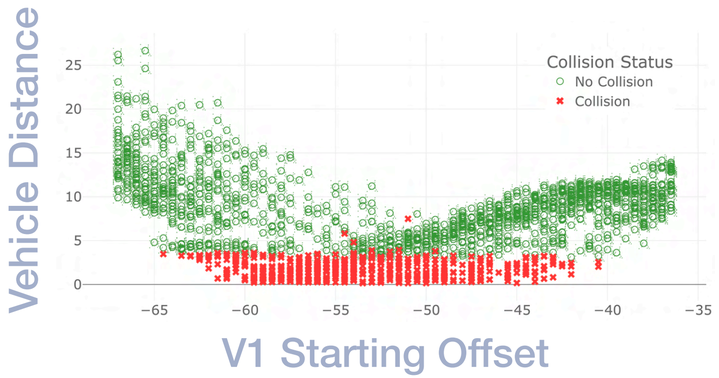




[Klikovits, Arcaini, PRDC'21]

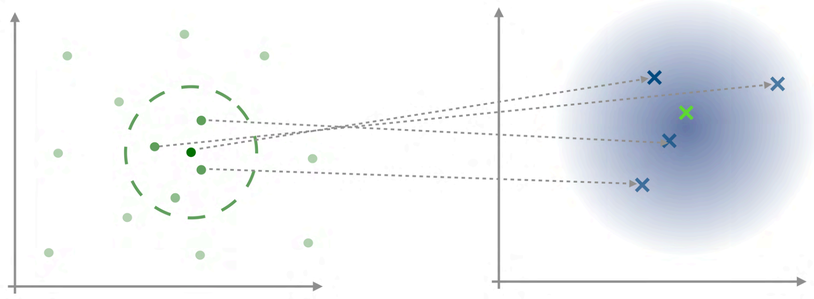
Conclusion





Let's repeat without repeating!

kNN-Averaging



KNN-Averaging for Noisy Multi-Objective Optimisation

Stefan Klikovits*, Paolo Arcaini

ERATO MMSD Hasuo Project
National Institute of Informatics, Tokyo

*supported by JSPS Kakenhi Grant-in-Aid JP20K23334