Advanced Optimisation Algorithms

SEP592, Summer 2021

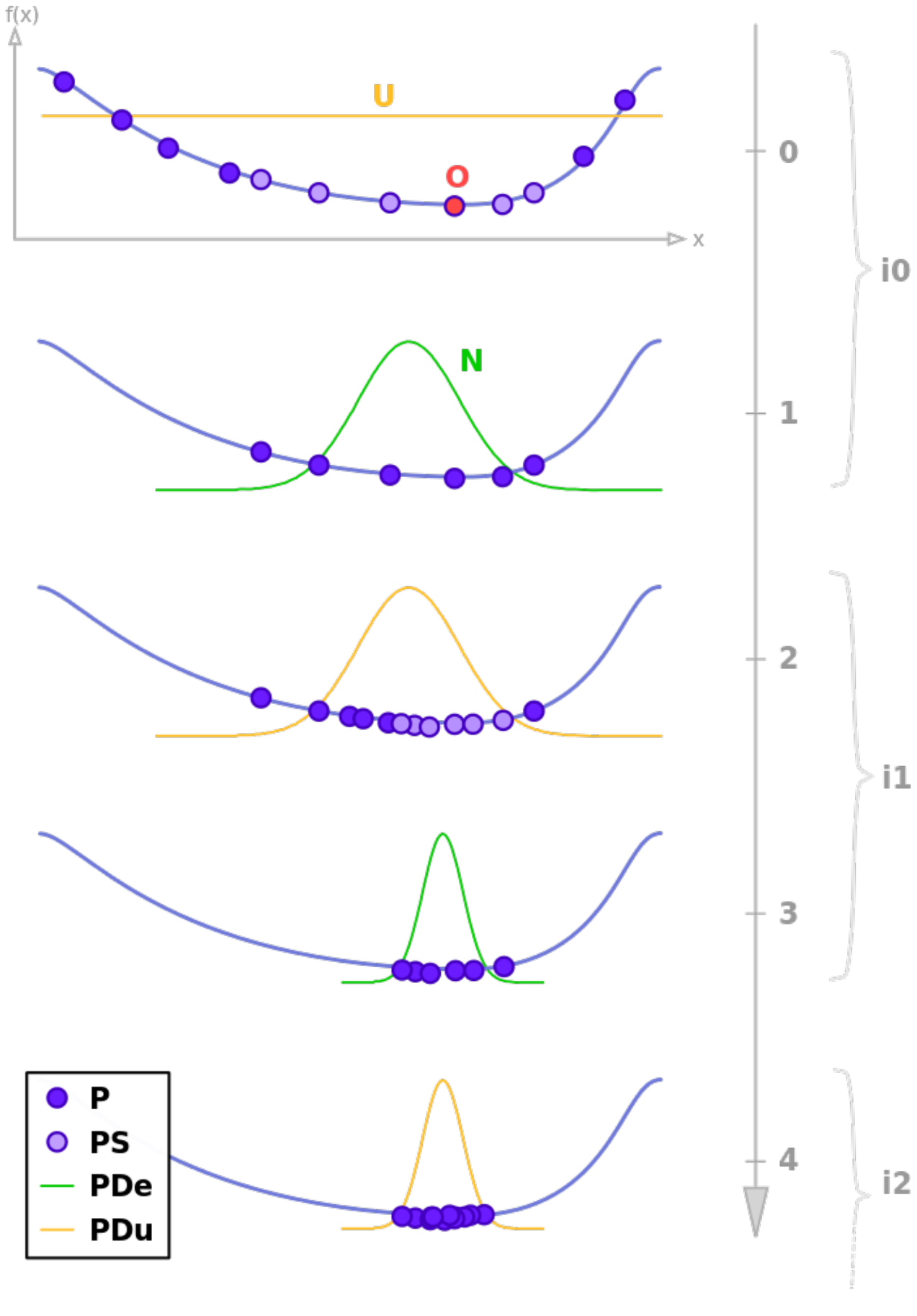
Shin Yoo

Estimation of Distribution Algorithm

- EDA mantra: avoid arbitrary operators; instead, explicitly model what desirable solutions should look like.
    - Do not evolve one solution, or even a group of solutions
    - Evolve a probability **distribution**, from which one or more solutions can be **sampled**.
- Very new; emerged in 2000s.

EDA

(1) $D_0 \leftarrow$ generate M random individuals
(2) **while** stopping criterion is not met
(3) $D^s_{l-1} \leftarrow$ select $N < M$ individuals from D_{l-1}
(4) $p_l(x) \leftarrow p(x|D^s_{l-1})$ //est. p.d. of x being in D^s_{l-1}
(5) $D_l \leftarrow$ sample M individuals from $p_l(x)$


f(x)
x
U
O
N
P
PS
PDe
PDu
0
1
2
3
4
i0
i1
i2

Independent Variables

Assume that variables are independent from each other, i.e. $p_l(\mathbf{x}) = \Pi_{i=1}^{n} p(x_i)$ for $\mathbf{x} = (x_1, \ldots, x_n)$:

$$p_l(x_i) = \frac{\sum_{j=1}^{N} \delta_j(X_i = x_i | D_{l-1}^s)}{N}$$

where $\delta_j(\mathbf{X}_i = x_i | D_{l-1}^s) = 1$ if, in the jth case of D_{l-1}^s, $\mathbf{X}_i = x_i$, 0 if not.

Independent Variables

- We are looking for a bit string of length 5; there *is* a fitness function.
- M = 4; suppose we let N = 2, and choose 2 and 4.
- What should we try the next?

#	Solutions	Fitness
1	00100	1
2	11011	4
3	01101	0
4	10111	3

Independent Variables

- Let $\mathbf{x} = (x_0, x_1, x_2, x_3, x_4)$.
- $D^s_{l-1} = \{11011, 10111\}$
- $p_l(x_0 = 1) = \frac{2}{2} = 1.0$
- $p_l(x_1 = 1) = \frac{1}{2} = 0.5$
- $p_l(x_2 = 1) = \frac{1}{2} = 0.5$
- $p_l(x_3 = 1) = \frac{2}{2} = 1.0$
- $p_l(x_4 = 1) = \frac{2}{2} = 1.0$

#	Solutions	Fitness
1	00100	1
2	11011	4
3	01101	0
4	10111	3

Univariate Marginal Distribution Algorithm

Independent Variables

$$p_{l+1}(x_i) = (1-\alpha)p_l(x_i) + \alpha\frac{1}{N}\sum_{k=1}^{N} x_i^l, k : M, \alpha \in (0, 1]$$

Population Based Incremental Learning

Multivariate Modelling

- When one has to capture dependencies between variables, the joint probability distribution needs to be factorised:

$$f_{XY}(x, y) = f_{X|Y=y}(x) f_Y(y)$$

- There are also algorithms that try to group variables, so that each group can be considered as an independent marginal distribution.

Differential Evolution

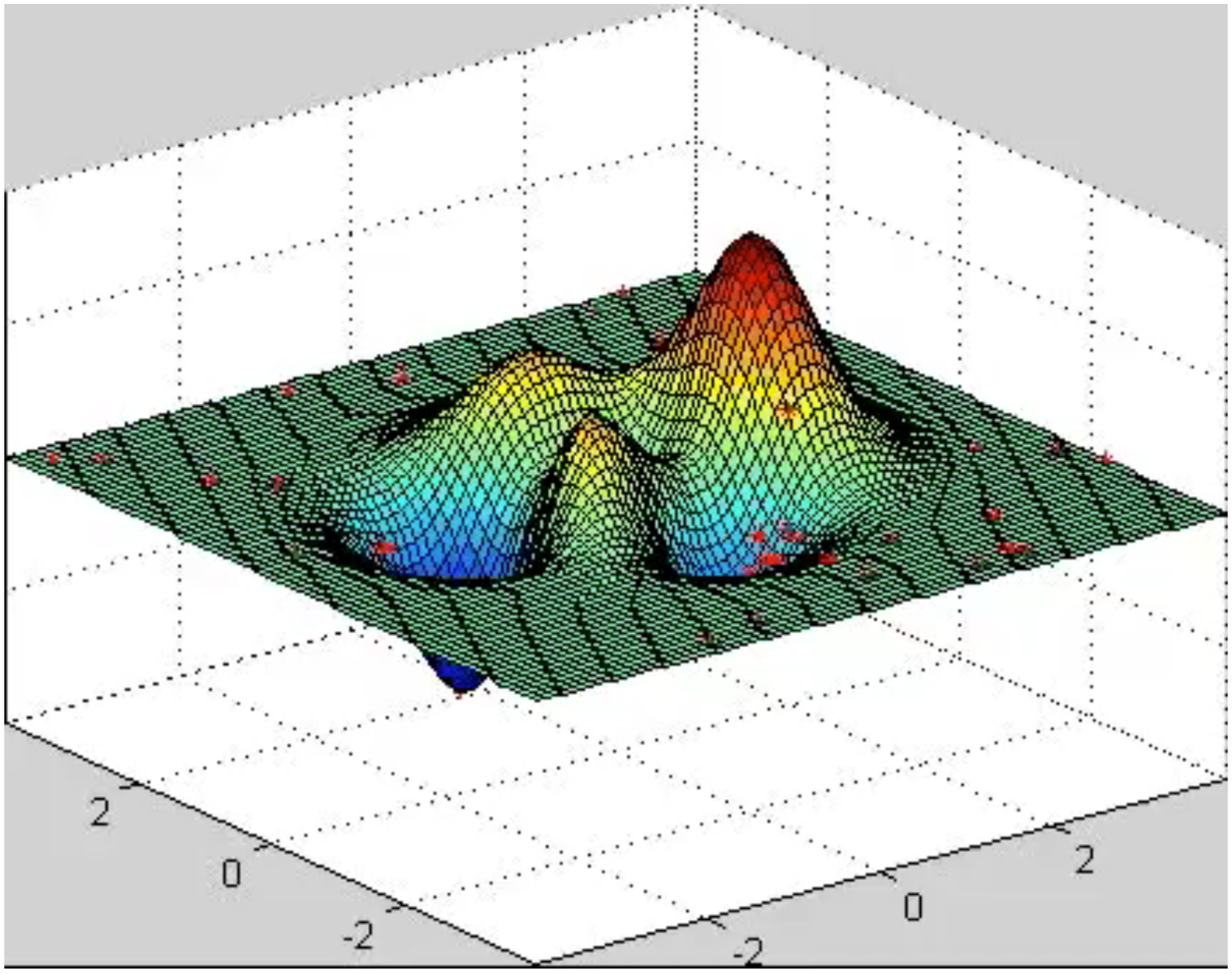
- Recently developed optimisation for real variables
- BUT! The underlying function does not need to be differentiable, i.e. it can be even discontinuous.
- DE can optimise this, unlike traditional gradient descent, for example.

Differential Evolution

- CR: crossover rate
- F ∈ [0, 2]: differential weight
- Exploration: essentially we randomly mix up vector elements, so we always do random exploration
- Exploitation: gradually we update the population with better solutions - if "better" means "in the same fitness valley", then there is increased chance of interpolating "within the valley", thereby exploiting the corresponding part of the landscape.

(1) $P \leftarrow M$ random real vectors of length n
(2) EVALUATE(P)
(3) **while** termination criterion not met
(4) **for** $j = 0$ **to** M
(5) $r_1, r_2, r_3 \leftarrow$ unique random integers $\in [1, M]$
(6) $R \leftarrow$ a random integer $\in (1, n)$
(7) **for** $i = 0$ **to** n
(8) **if** $U(0,1) < CR \vee i == R$ **then** $x'_{i,j} = x_{i,r_3} + F(x_{i,r_1} - x_{i,r_2})$
(9) **else** $x'_{i,j} = x_{i,j}$
(10) EVALUATE(x'_j)
(11) **if** IsBetter(x'_j, x_j) **then** $x_j \leftarrow x'_j$

Differential Evolution



Seeding

- Ensure that whatever knowledge of solution building blocks gets incorporated into your optimisation.

- Often this takes the form of initialisation of population based algorithms.

- Seed is strong, indeed.

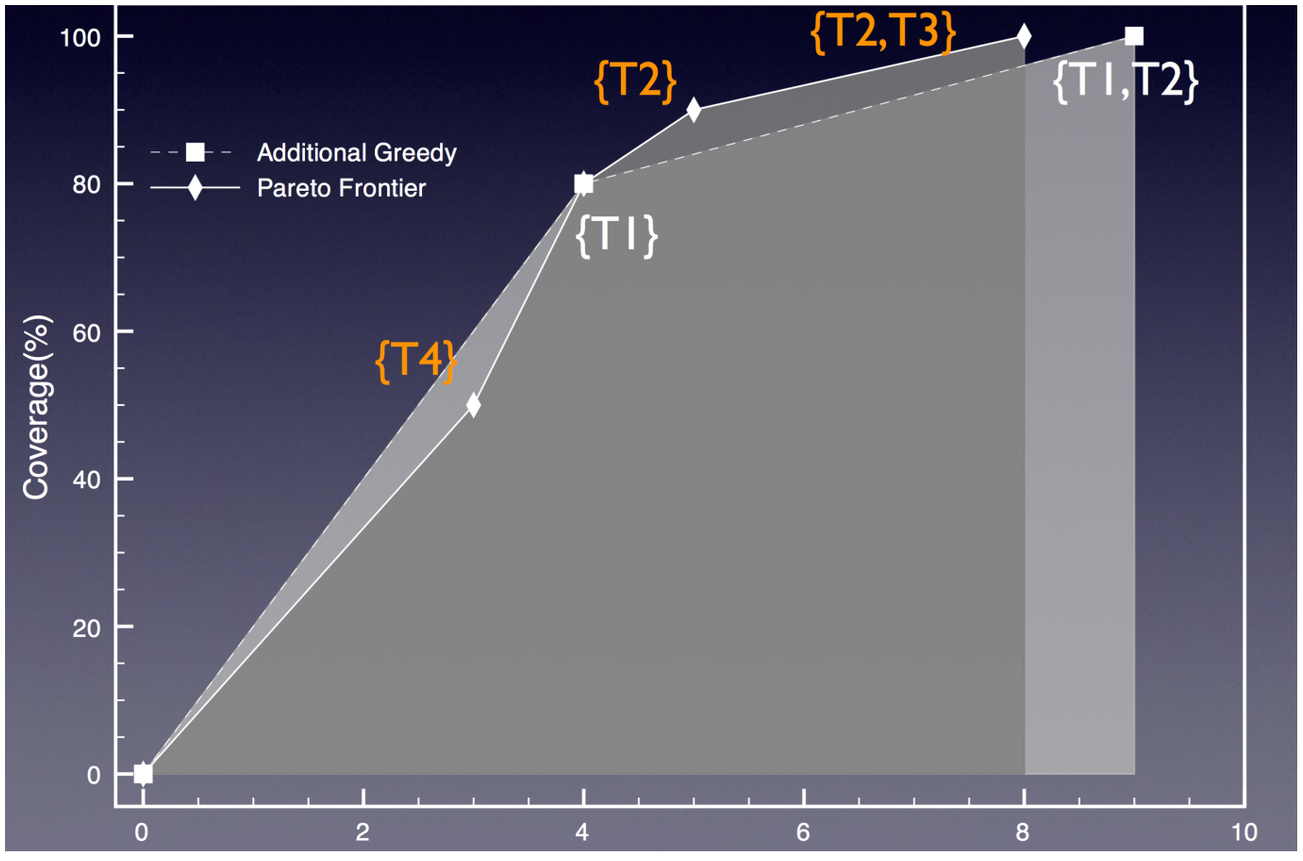
Example 1. Test Suite Generation for Java

- EvoSuite is a whole test suite generation tool for Java.
- Test data generation for OO languages is extra-challenging.
  - Two layers of problems: method sequence for state building, and parameter selection for method calls themselves.
- Idea: find constants in the source code, and seed the initial population with them.

G. Fraser and A. Arcuri. The seed is strong: Seeding strategies in search-based software testing. In Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on, pages 121–130, April 2012.

Example 2. MO Test Suite Minimisation

- Solutions produced by single-objective greedy heuristic is not always Pareto optimal
  - But quite close!
- Idea: when initialising the population, include (partial) solutions generated by greedy.
  - Faster convergence



S. Yoo and M. Harman. Using hybrid algorithm for pareto effcient multi-objective test suite minimisation. Journal of Systems Software, 83(4):689–701, April 2010.

Memetic Algorithm

"Examples of memes are tunes, ideas, catch-phrases, clothes fashions, ways of making pots or of building arches. Just as genes propagate themselves in the gene pool by leaping from body to body via sperms or eggs, so memes propagate themselves in the meme pool by leaping from brain to brain via a process which, in the broad sense, can be called imitation."

- "Memetic" as in internet memes 😟
- As I understand: what individual learns, gets propagated through the population.
- Intuitively, memetic algorithm is evolutionary algorithm combined with occasional local search.

Memetic Algorithm

- In GA, individuals in the population reach their current location by the means of crossover and mutation, i.e. not necessarily knowing the neighbourhood.

- Why not apply local search (e.g. hill climbing) to get a little bit better?

- Parameters to consider:

  - How often do you perform individual learning?

  - What is the budget for local search?

Example: Test Suite Generation for Java

- The same problem with OO that we have seen earlier.
- If GA can evolve the correct method sequence, we can try to optimise the method parameters locally.

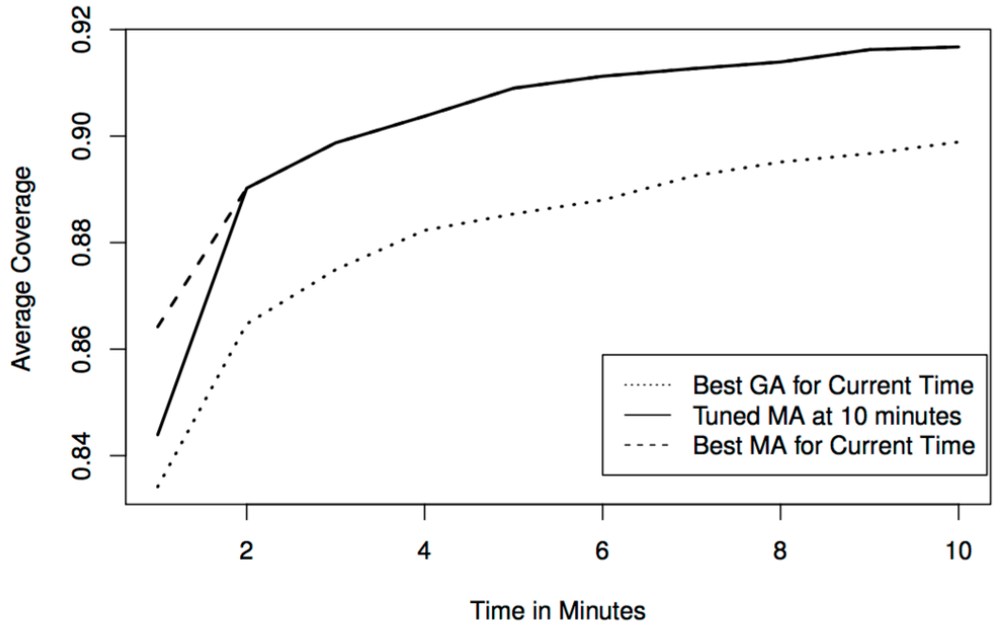


Figure 4: Average coverage at different points in time for the "best" GA (at each minute interval), the MA tuned at 10 minutes, and "best" MA configuration at each minute interval.

G. Fraser, A. Arcuri, and P. McMinn. A memetic algorithm for whole test suite generation. Journal of Systems and Software, 103:311 – 327, 2015.

Hyper-heuristic

- Q: I want to solve my problem using metaheuristic, but I don't know which one to use.

- A: Simple! Use a metaheuristic to choose one.

- 😵

A well-known scientist (some say it was Bertrand Russell) once gave a public lecture on astronomy. He described how the earth orbits around the sun and how the sun, in turn, orbits around the centre of a vast collection of stars called our galaxy. At the end of the lecture, a little old lady at the back of the room got up and said: "What you have told us is rubbish. The world is really a flat plate supported on the back of a giant tortoise." The scientist gave a superior smile before replying, "What is the tortoise standing on?" "You're very clever, young man, very clever," said the old lady. "But it's turtles all the way down!"

–Stephen Hawking, “A Brief History of Time”

But more seriously, HH

- Selecting hyper-heuristics: for a given problem, what is the best heuristic among the ones in the given library?
    - Can only learn online, based on their actual performance

- Generation hyper-heuristic: combine heuristic component to build a domain-specific heuristic.

    - IMHO, this is beginning to look like GP…

Example: HH Simulated Annealing

- Inside: the simulated annealing, we generate a neighbourhood solution by applying one of many local move operators.

- Outside: we learn which operator is improving the fitness the best, using machine learning, as the optimisation progresses.

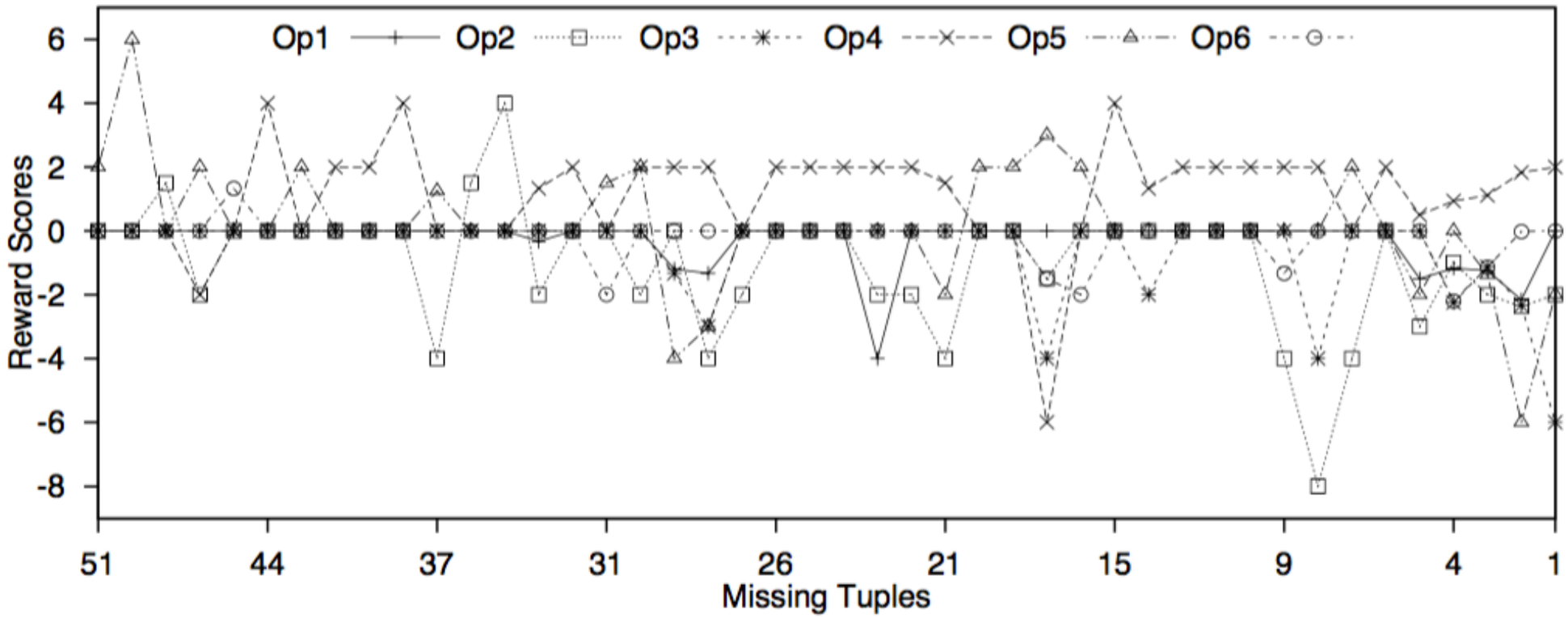
- That is, selection of low-level heuristic baed on dynamic learning.

Input : $\hat{t}$, k, v, c, $\tilde{N}$, *MaxNoImprovment*
Output: covering array A
$A \leftarrow$ `initial_array` (t,k,v,N)
$no\_improvement \leftarrow 0$
$curr\_missing \leftarrow$ `countMissingTuples` (A)
while $curr\_missing \neq 0$ *and* $MaxNoImprovment \neq no\_improvement$ **do**
  $op \leftarrow$ `rl_agent_choose_action` ($curr\_missing$)
  A' = `local_move` (op, A)
  while `fix_cons_violation` (A', c) **do**
    A' = `local_move` (op, A)
  end
  $new\_missing \leftarrow$ `countMissingTuples` (A')
  $\Delta fitness = curr\_missing - new\_missing$
  `rl_agent_set_reward` (op, $\Delta fitness$)
  if $e^{\Delta fitness/Temp} >$ `rand_0_to_1` () **then**
    if $\Delta fitness = 0$ **then**
      $no\_improvement \leftarrow no\_improvement + 1$
    else
      $no\_improvement \leftarrow 0$
    end
    $A \leftarrow A'$
    $curr\_missing \leftarrow new\_missing$
  end
  $Temp \leftarrow$ `cool` ($Temp$)
end

ALGORITHM 1: HHSA

Y. Jia, M. Cohen, M. Harman, and J. Petke. Learning combinatorial interaction test generation strategies using hyperheuristic search. In IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE), volume 1, pages 540–550, May 2015.

Example: HH Simulated Annealing



- Different operators receives better reward in different stages of optimisation, meaning that we are indeed learning and choosing operators wisely.

Y. Jia, M. Cohen, M. Harman, and J. Petke. Learning combinatorial interaction test generation strategies using hyperheuristic search. In IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE), volume 1, pages 540–550, May 2015.

Problem Space Exploration

- Originally introduced for Job-Shop Scheduling (Storer et al. ’92)
- Do not search for a solution; **search for a problem!**
    - You have a difficult problem.
    - Change the problem a little bit, see if you can solve it more easily.
    - If solved, apply answer to the original problem and check if it’s solved too.
    - Rinse, lather, repeat.

S. Yoo. A novel mask-coding representation for set cover problems with applications in test suite min- imisation. In Proceedings of the 2nd International Symposium on Search-Based Software Engineering (SSBSE 2010), 2010.

Optimisation based on PSE

- A perturbation **M** alters a problem instance **P** into **P'**. Similarly, function **m** perturbs solution **S** to **S'**.

- We assume we have a cheap (but perhaps not optimal) heuristic **A** that can solve the class of problems that include **P** and **P'**s.

- Our genotype individuals are instances of **M**, not **S**. Fitness of a perturbation **M** is evaluated as **fitness(S)** s.t. $\mathbf{S = m^{-1}(S'), S' = A(P'), P' = M(P)}$.

- As we manipulate the genotype individuals (i.e. perturbations), we obtain different instances of **S** by exploring the problem space, i.e. different **P'**s.

PSE Intuition

- This is just another way to modify one solution into another.
- We do not define the modification operator in the solution space.
- Instead, we modify the problem, solve it using another simpler heuristic, take the solution, and bring it back to the original problem.
- We then evaluate the obtained solution against the original problem.

Optimisation based on PSE

- Pros
    - It is much less restricted by the features of search landscape formulated by the original problem and the fitness function.
- Cons
    - Hardly intuitive :)
    - The use of construction heuristic not only requires additional computation but also adds complexity.

Example

	S_1	S_2	S_3	S_4	S_5
T_1	x	x	x		
T_2		x	x		
T_3				x	
T_4					x

- Given a test suite minimisation problem, how would represent your solution?
- Given the representation above, how would you initialise your population?

- Binary string representation: a string of length 4; digit s_i corresponds to the inclusion of T_i
- Randomly set each digit of individual solution with $P(s_t=0) = P(s_t=1) = 0.5$

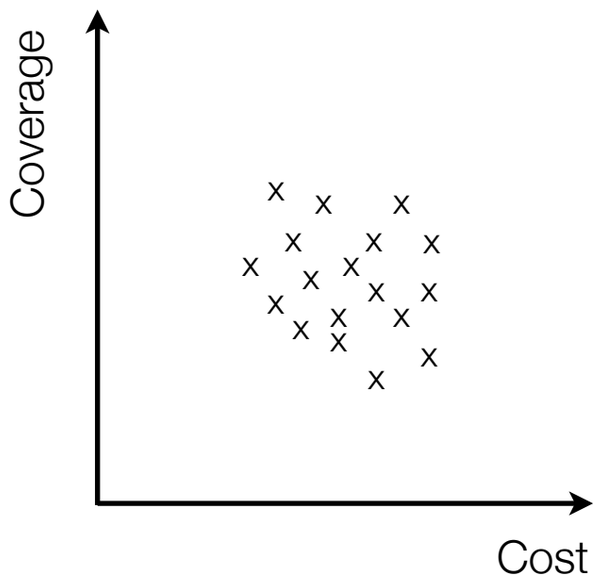
Problem: Dimensional Plateau

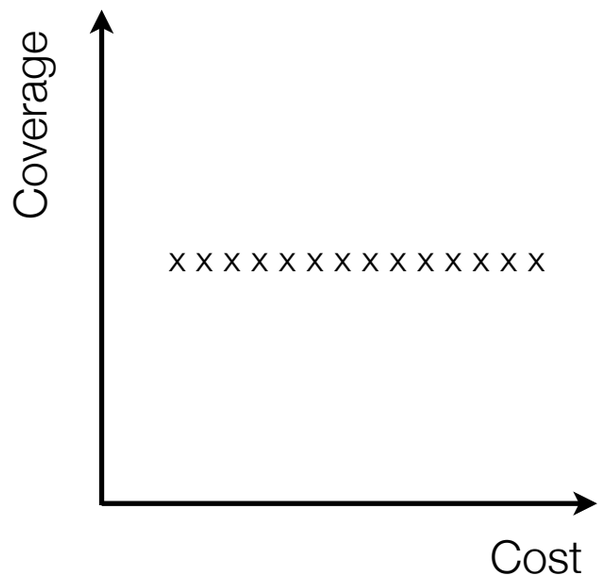
- Real minimisation looks very different from toy examples.
    - Program structure affects the frequency each element is covered, i.e. some parts of the SUT are covered by a large number of tests in similar patterns
    - There are some highly redundant test suites
- If we sample **random** test sets, the **variance** of coverage may be **very small**.

	S_1	S_2	S_3	S_4	S_5
T_1	x	x	x		
T_2		x	x		
T_3				x	
T_4					x

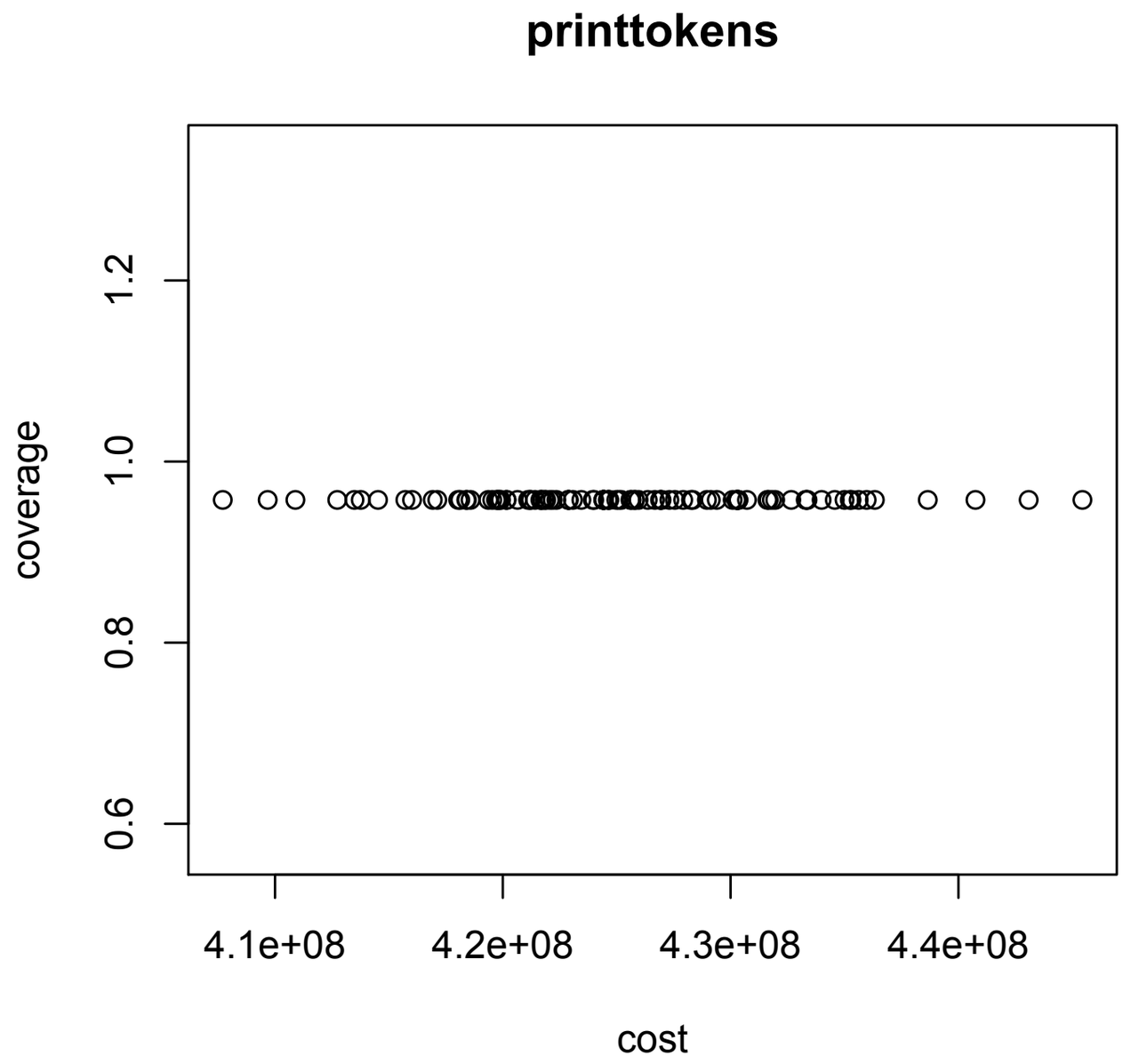
Problem: Dimensional Plateau

- Dimensional Plateau occurs when, in multi-objective optimisation, one of the objectives manifests a plateau in the search space while others do not.
    - What you think is random may not be random
    - Hard to escape





Dimensional Plateau



Perturbation for Set-Cover Problem

- Existing representations based on PSE are often called Weight Coding because the perturbation is a vector of weights applied to **P**; however, we are dealing with sets.

- **Mask-Coding**: perturbs an instance of Set-Cover problem by either masking parts of the universe (i.e. the SUT) or the sets (i.e. the tests).

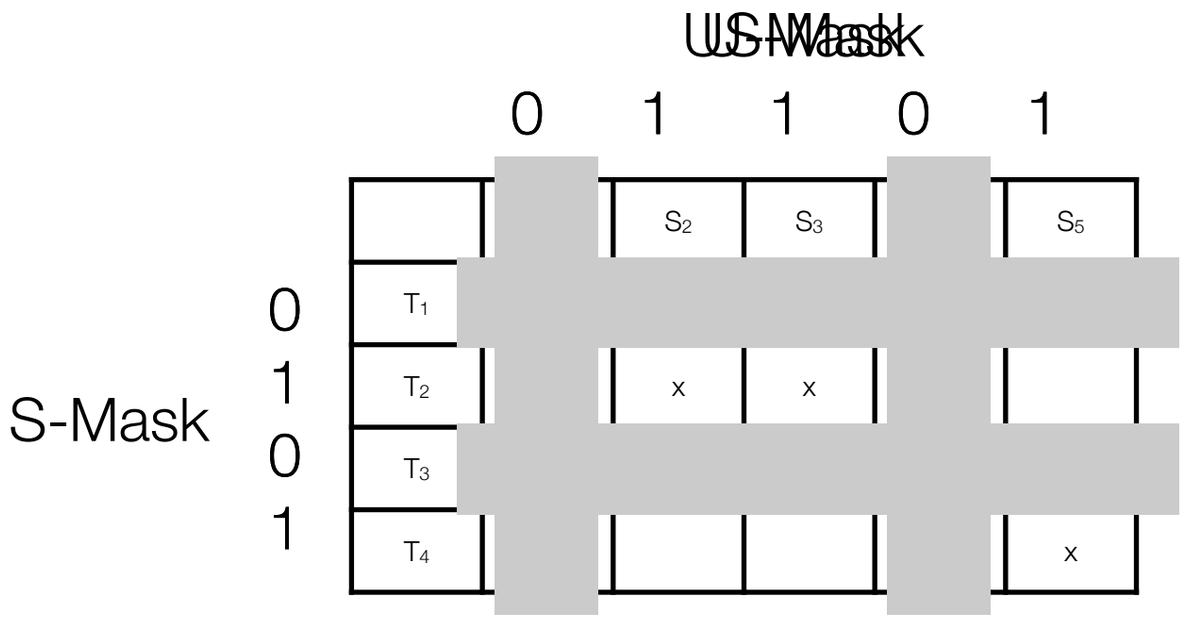
- Ironically, our masks are binary strings :)

U, S, US-Mask

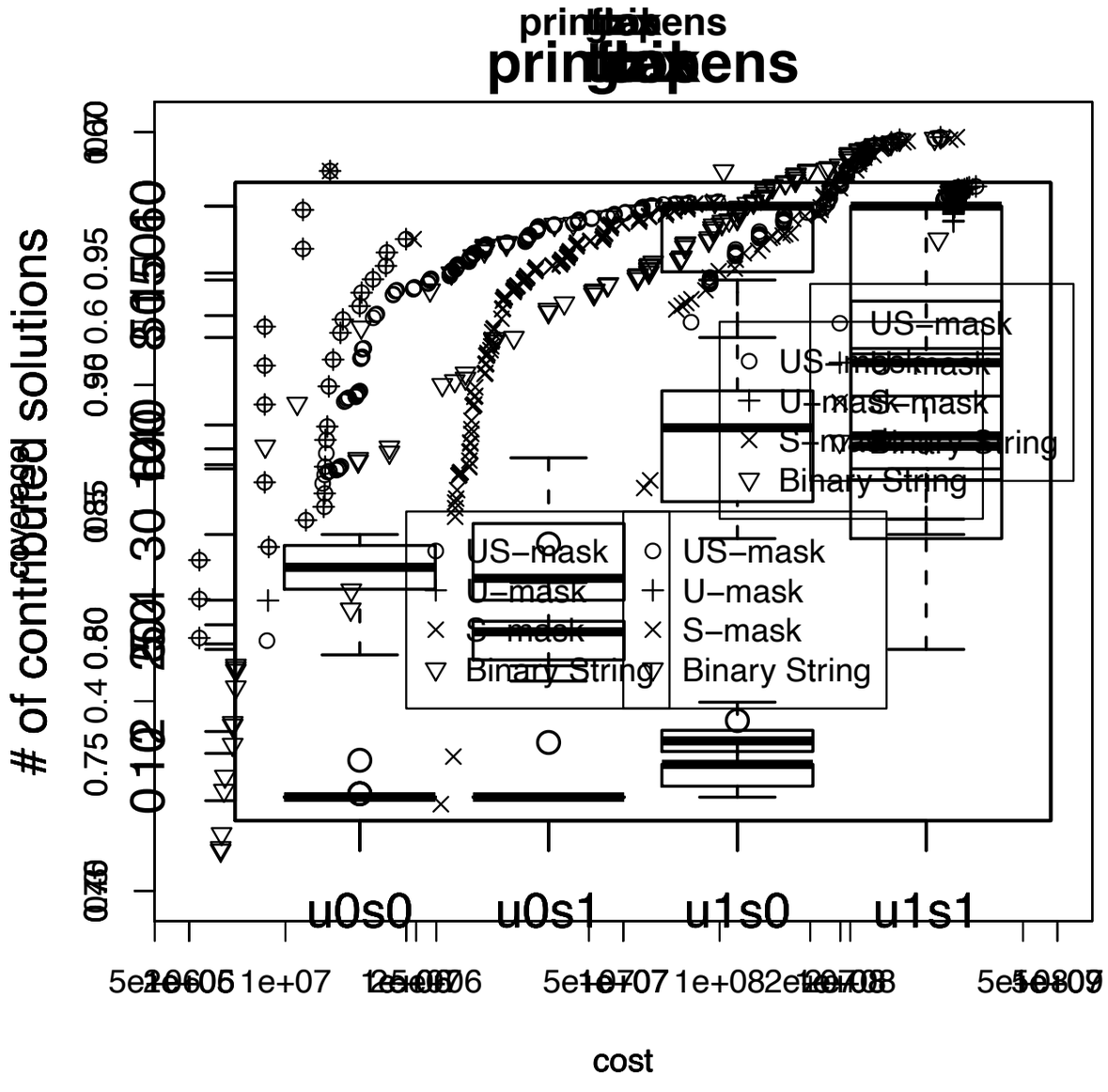
- U-Mask : perturbs the Set-Cover problem by masking (i.e. hiding) some of the elements that are to be covered (in our case, some structural elements of the SUT)

- S-Mask : perturbs the Set-Cover problem by masking some of the sets that are used to form the cover (in our case, some of the tests in the test suite)

- US-Mask : combines U- and S-Mask in a single binary string



Diversity



- For search-spaces with dimensional plateau, Mask-Coding improves the search significantly
- Binary string entirely fails to escape the plateau
- When there is no plateau, it is not as successful as binary string