Improving the Efficiency of Heuristic Algorithms

Andrius Gasiukevičius

Vilnius University

Faculty of Mathematics and Informatics

andrius.gasiukevicius@mif.stud.vu.lt

Outline

- Motivation
- Background
- Research questions
- Data sample
- Methods of Analysis
  - The Survey
- Results
- Discussion
- Conclusion
- References

Results:

- Heuristics for classical problems
- No Free Lunch theorem
  - Answering Q1
- Survey Responses (3)
- Algorithm Categories
- Algorithm Performance (5)
- Overall Comparison
  - Answering Q2
- Combining Heuristics (2)
  - Answering Q3

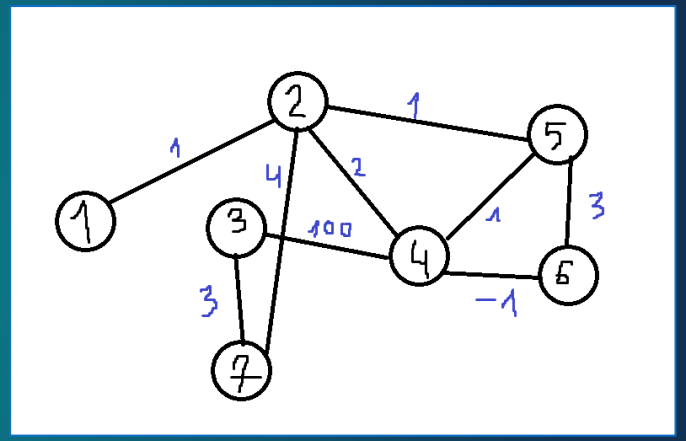
Motivation

- Why should we care?
- Research Gap
  - In most research articles, well-known heuristics are usually not compared with human intuition.
- Inspiration
  - Articles on recent advances in heuristics
  - Past mathematics and informatics Olympiad problems with heuristic solution ideas
  - “Brainstorm” YouTube series by PurpleMind

Got a question? Send an email to
andrius.gasiukevicius@mif.stud.vu.lt

Background

- History of algorithms
- Heuristics and metaheuristics
- Note on abbreviations



Gad, A.G. (2022) Particle swarm optimization algorithm and its applications: A systematic review - archives of computational methods in engineering

Clerc, M. (2012) *Standard Particle Swarm Optimisation, HAL Open Access Archive.*

Got a question? Send an email to
andrius.gasiukevicius@mif.stud.vu.lt

VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS

Research questions

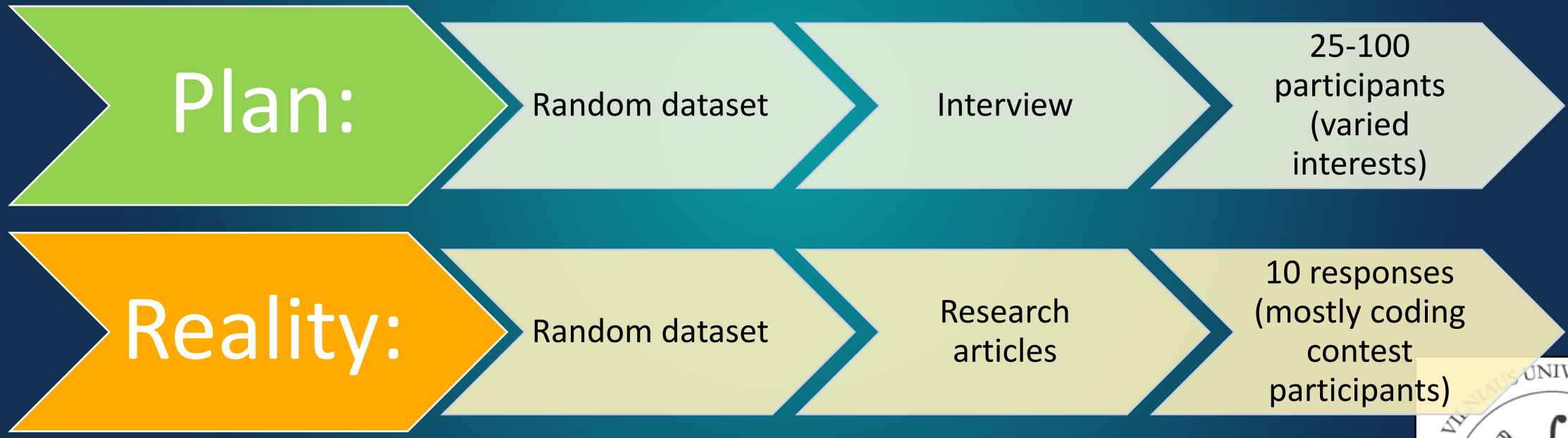
The research aims to answer the following questions:

- Which heuristic algorithms are the most effective for solving classical algorithmic problems?
- How effective are 'greedy' or personal intuition-based heuristics in practice compared to well-documented ones?
- How can we improve heuristics to increase their applicability for problem solving?

Got a question? Send an email to
andrius.gasiukevicius@mif.stud.vu.lt

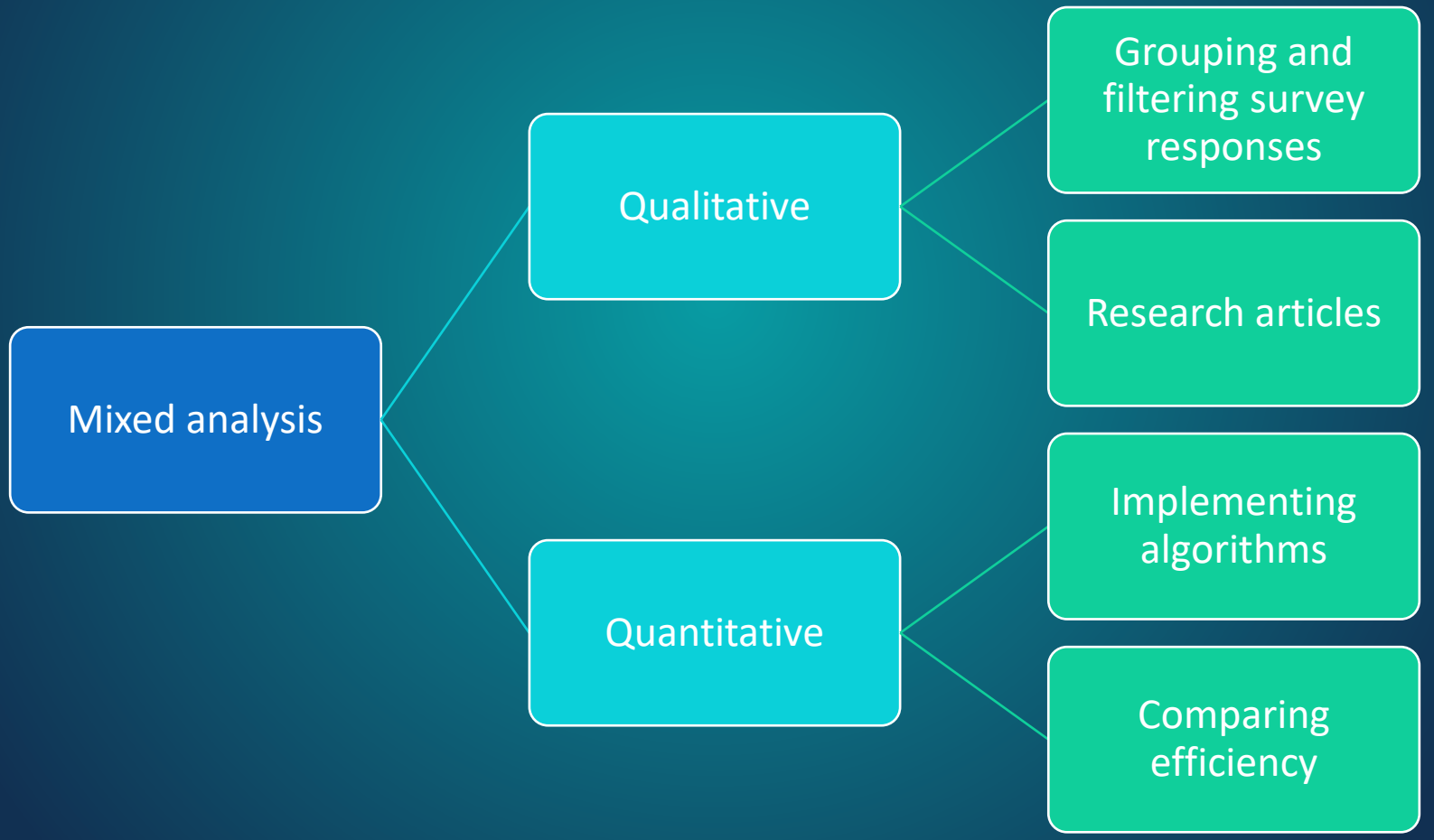
VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS

Data sample



Got a question? Send an email to
andrius.gasiukevicius@mif.stud.vu.lt

Methods of Analysis

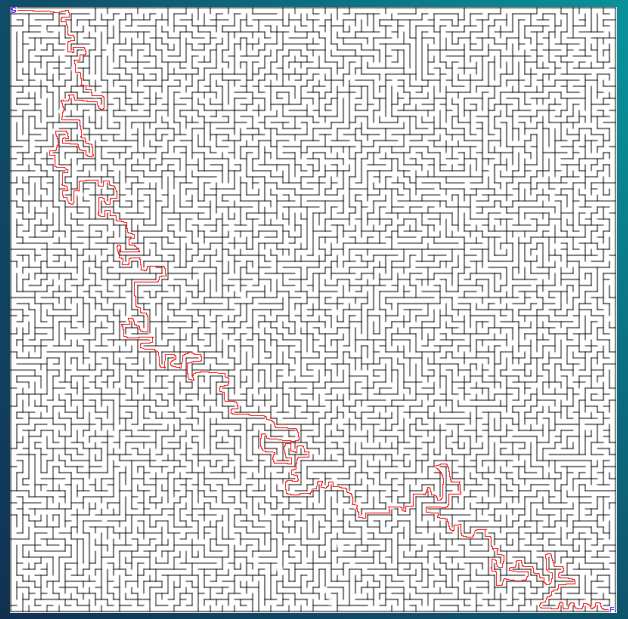


Got a question? Send an email to
andrius.gasiukevicius@mif.stud.vu.lt

VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS

The Survey

Participants were presented with a challenging problem that they had to solve. The following results will compare some of the provided solutions.



Got a question? Send an email to
andrius.gasiukevicius@mif.stud.vu.lt

VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS

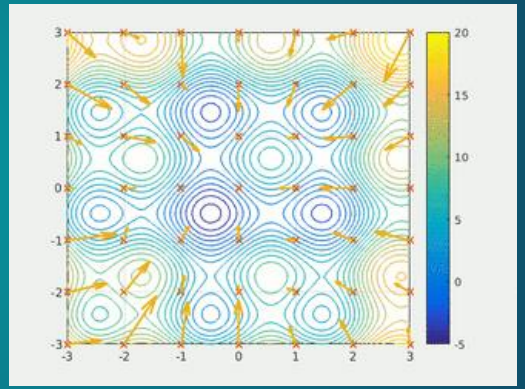
Results


VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS

Heuristics in problem-solving

To solve classical algorithmic problems, the most commonly-used heuristic algorithms are (Colorni *et al.*, 1996):

- GA (Genetic Algorithms)
- SA (Simulated Annealing)
- SC (Sampling and Clustering)
- TS (Tabu Search)
- NN (Neural Nets)
- AS (Ant System) or ACO (Ant Colony Optimisation)



source: https://en.wikipedia.org/wiki/Particle_swarm_optimization

In recent years, PSO (Particle Swarm Optimisation) and other algorithms have emerged.

Got a question? Send an email to andrius.gasiukevicius@mif.stud.vu.lt

VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS

No Free Lunch theorem

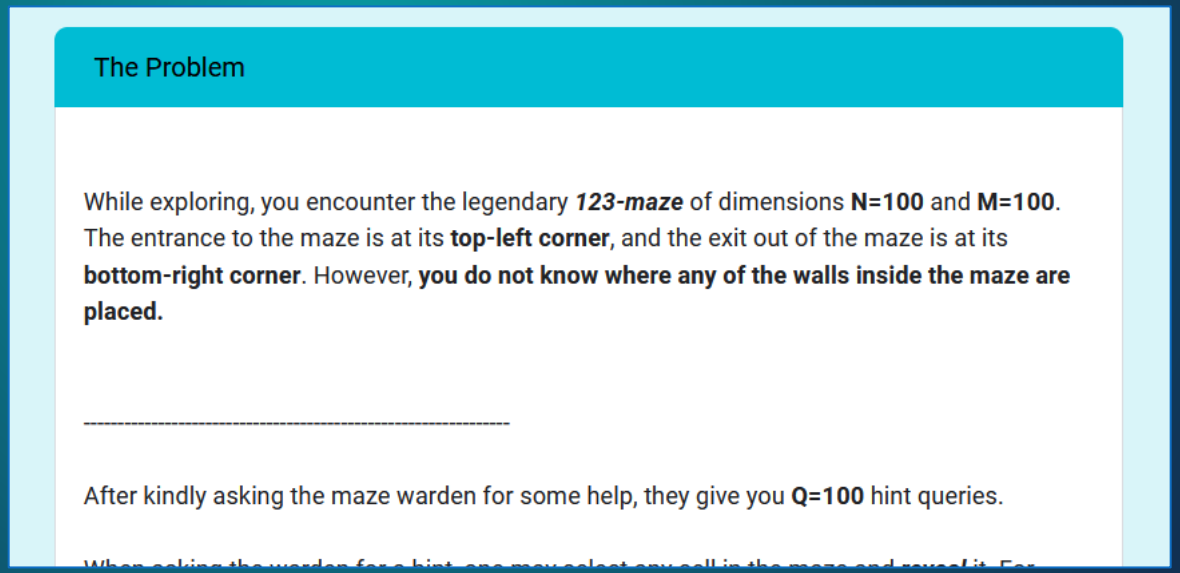
However:

- Just as there is no such thing as a free lunch, there is no universal algorithm to solve every optimisation problem.
- This gives reason to why the study of heuristics and optimisation algorithms is so important.

> "There is a question here as to why we need more algorithms despite the many algorithms proposed so far.
> The answer to this question is in the No Free Lunch (NFL) theorem [28] that logically has proven that there is no optimisation technique for solving all optimisation problems. In other words, algorithms in this field perform equally on average when considering *all* optimisation problems." (Saremi *et al.*, 2017)

Survey Responses (1)

- Sample, quality and quantity
- Participant's Background
- Non-responses
- Solutions



Got a question? Send an email to
andrius.gasiukevicius@mif.stud.vu.lt

VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS

“After you ask warden to reveal the starting cell (topmost left cell) you will know what is the minimum path_travel_time of the maze because all paths go through that cell, so all the good paths, as well as the good path which takes the least time to complete. That means, that the warden will tell you that at the moment total score is 1/x, where is x the minimum path_travel_time. Then you ask warden to reveal one of the adjacent nodes; one should start from asking to reveal the cell which takes less time to travel to from the starting node. If warden gives total score equal to 2/x, the this cell is part of the optimal route, because the denominator did not change (it is still the optimal path_travel_time) and numerator increased by one (path_revealed_nodes acquired a new cell). Let's consider if the total score after the reveal of the second cell was 2/y or 1/x instead, (where y is not equal x), there may be two cases:
1.(2/y) the total score is the path which contains both revealed cells, yet since the y is not equal x, it must be larger than x, since the path with the length of the truly shortest path_travel_time was already revealed, and if this cell had been a part of it, the total score would have been 2/x, and since it is not, then y is part of a slower path, hence this cell is not a part of the optimal path.
2. (1/x) if the total score did not change, it means, that the optimal path, which the warden previously had in mind, is still optimal to gather largest score, however, the score would have been larger if the second cell, which was revealed, was a part of this path, thus the total score can be increased.
Either way, since we were able to choose between two adjacent second cell options (to go right or to go below), then the unchosen cell was a part of optimal path . Then we choose that cell, see that the total score is 2/x and gather new information about the adjacent cells. Later the algorithm basically stays the same. We still have two adjacent unrevealed cells. We start with the one which takes less to travel to, if that does not yield 3/x total score then we choose others (if some of the adjacent cells are separated by walls, then we ignore them). Now depending on which was the optimal cell, we either have three or two adjacent optimal cells and we continue our algorithm until our queries end. If after using the last query, the total score is not 100/x, we drink the potion. Since we remember which revealed cells yielded which total scores [there might be problem with how I interpret this problem, but from what I understand, we remember what total score was received at each step], we mark our way up to the node, which we ended last time (because we remember which cells gave us what scores, then we can reveal only those cells which were part of the optimal path). Since the score was not 100/x, there might be between 99 and 34 nodes revealed (according my calculations, 34 is the worst case scenario where we always pick the wrong adjacent nodes and get into the path, where there are three adjacent nodes as soon as possible). Let's consider the worst case scenario. We reveal 34 cells, which are a part of optimal path. Then we start from the node which we ended last time, perform same algorithm moves, find at least 34 new optimal cells and so on. During 4 rerun we know more than 100 optimal cells so we just reveal them and get optimal total score. If I made some interpretation mistakes, I still believe that using this algorithm we can find 100 cells which belong to the optimal path or even find the path from start to finish since, we can reveal up to 101*100 cells, and there are only 100*100 cells in the maze.”

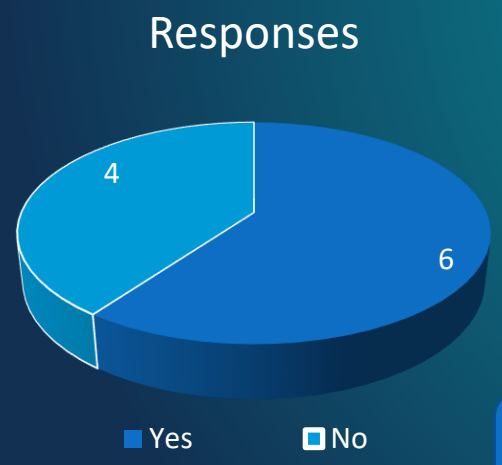
“I will get all of the times between squares with the checkered pattern (as mentioned above). Then apply a modified version of Dijkstra's algorithm that finds not the shortest path, but the largest score.”

“My algorithm is:
Plz set me free
Aaaaaappakakakakakakaakakakakakakakakakakaakkaaka
kakakakakakaakskakakakakakakakaakakakakakakakaka
kakakakakakakakaakHELPHELPHELPHELP”

Survey Responses (2)

- Intuition and heuristics

Do you think that it is possible to always pick the optimal cells to reveal such that score is consistently maximised, no matter the starting configuration of the maze?



"Difficult for people who are not competitive programmers or hardcore leetcoders I think"

"semi-random revealing, or some fixed patter of revealing"

"Try as many paths as possible."

"Start by focusing on revealing useful areas and use potions to try again when needed."

Got a question? Send an email to
andrius.gasiukevicius@mif.stud.vu.lt

Survey Responses (3)

- More responses

Try to reveal every other cell to save hints and follow an open path heading bottom right. When i am out of hints i use the potion and retry, choosing a different path until hopefully finding the exit.

Reveal the starting cell and set it as the head (H) of my query path.
Reveal all cells which are connected to H.
The cell which maximised the total score is set as H.
Repeat these steps until we run out of queries.
Now reveal all of the cells which were set as H in the previous run and with the remaining queries continue the "reveal -> max set as head" algorithm.
We end when the query path is of length Q.

Reveal random tiles but try to spread them out, so perhaps just manually pick some corner pieces, some in the middle, etc. keep track which reveals produce the best score, so after drinking the potion, accordingly either build up on your previous or start new pattern.

Categorising Problems and Algorithms

- Free/rigid
  - The problem was quite rigid
- Global/local
  - Most solutions used a combination of global and local heuristics
- Classical optimisation/Intuitive heuristic
  - Two classical optimisation algorithms (ACO and PSO) were used for comparison

Got a question? Send an email to
andrius.gasiukevicius@mif.stud.vu.lt

VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS

Algorithm Performance (1)

10 strategies

- DFS-based: multi-dfs, single-dfs, skip-dfs, greedy, improve, crossroad
- Global: catalan, pattern, corners, checker

2 optimisation algorithms

- PSO, ACO

Repeated random cell selection as baseline

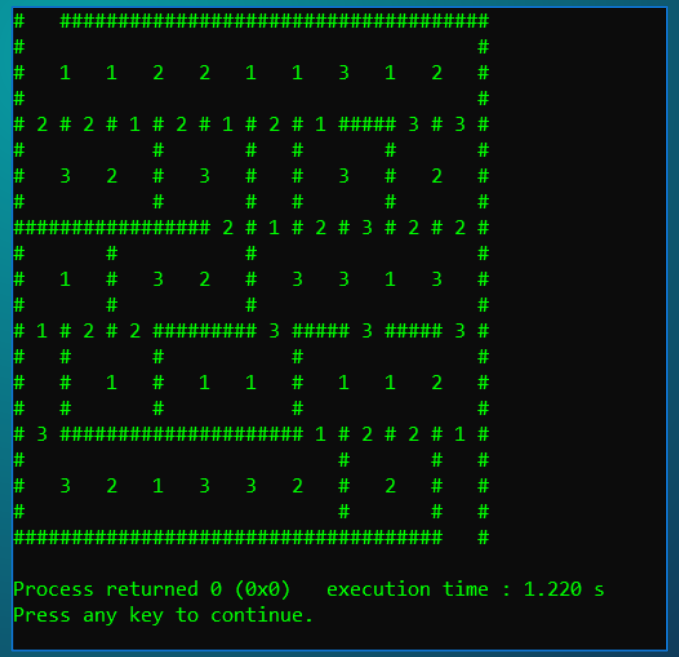
Got a question? Send an email to
andrius.gasiukevicius@mif.stud.vu.lt

VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS

Algorithm Performance (2)

- Implemented a 123-maze generation algorithm
- Implemented common strategies
- 1706 lines of code

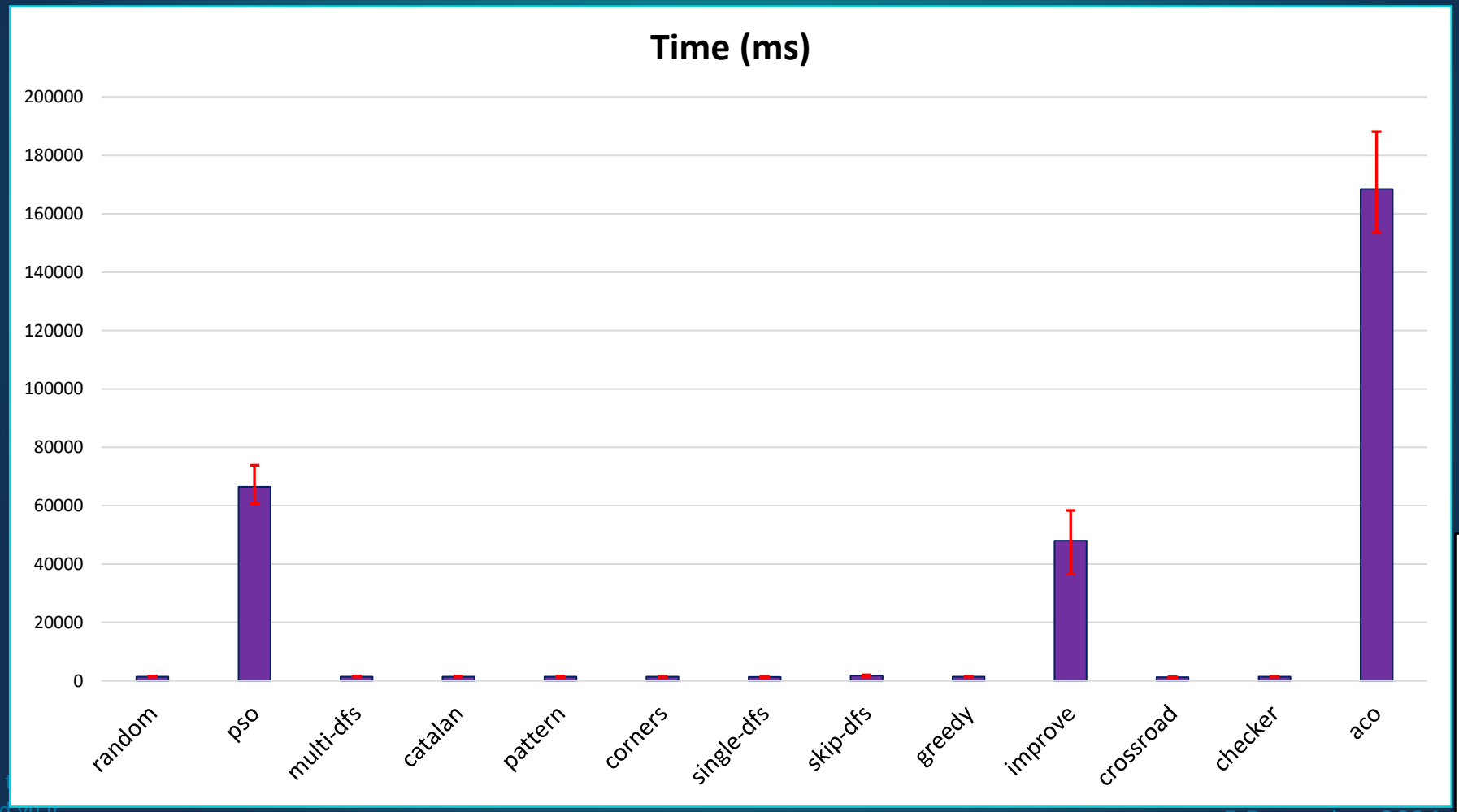
Available at:



Got a question? Send an email to
andrius.gasiukevicius@mif.stud.vu.lt

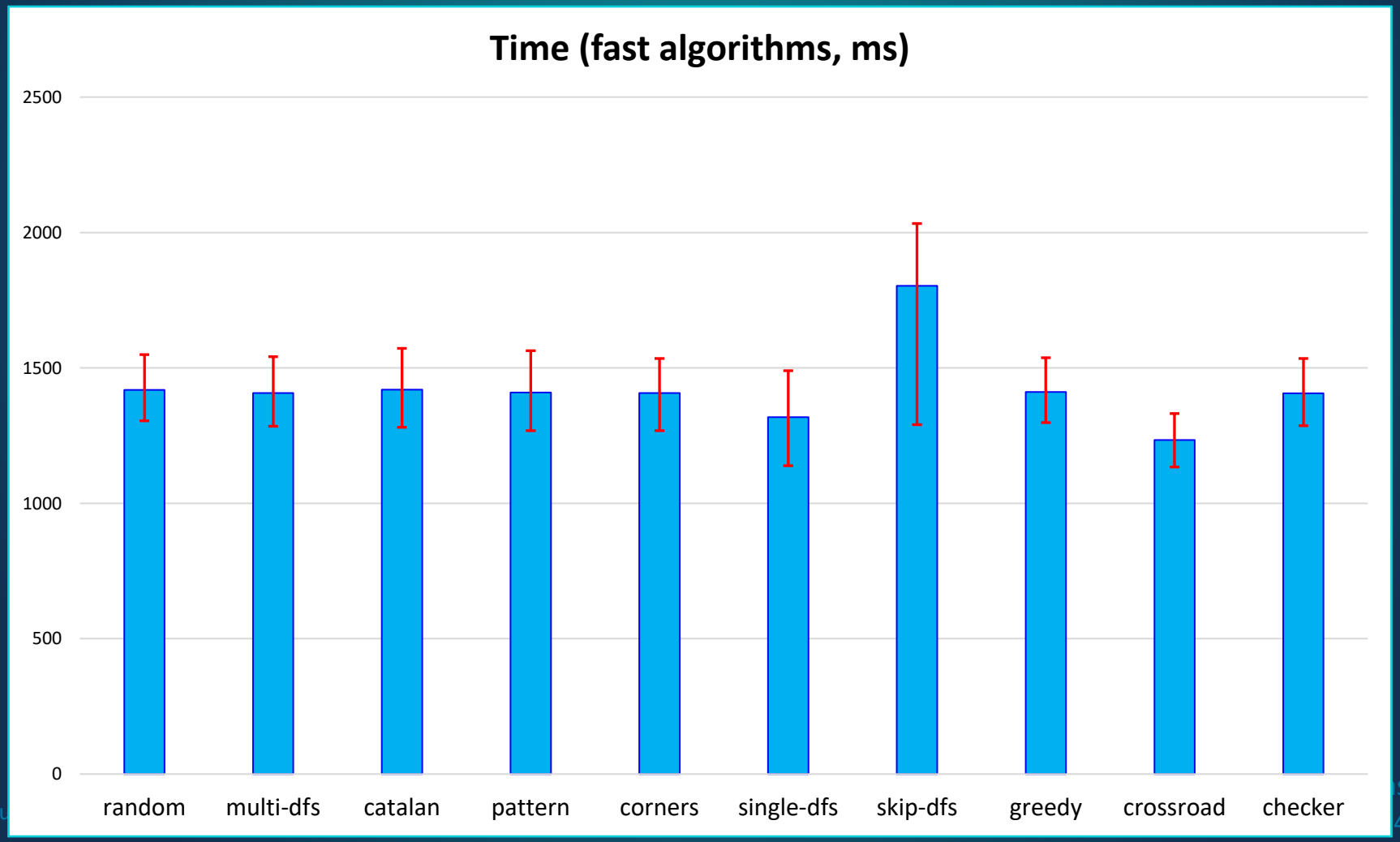
VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS

Algorithm Performance (3)



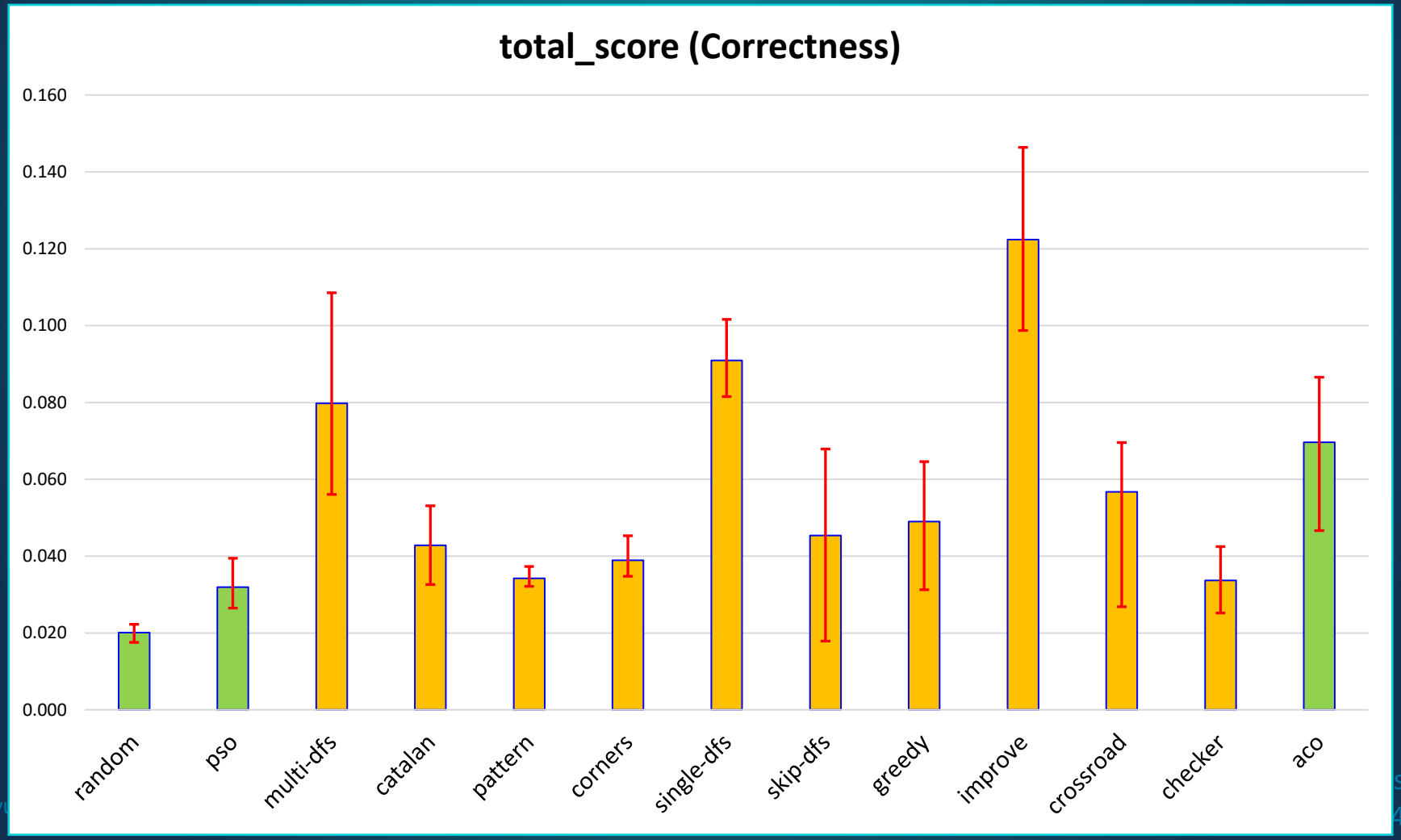
Got a question? Send an email t
andrius.gasiukevicius@mif.stud.vu.lt

Algorithm Performance (4)



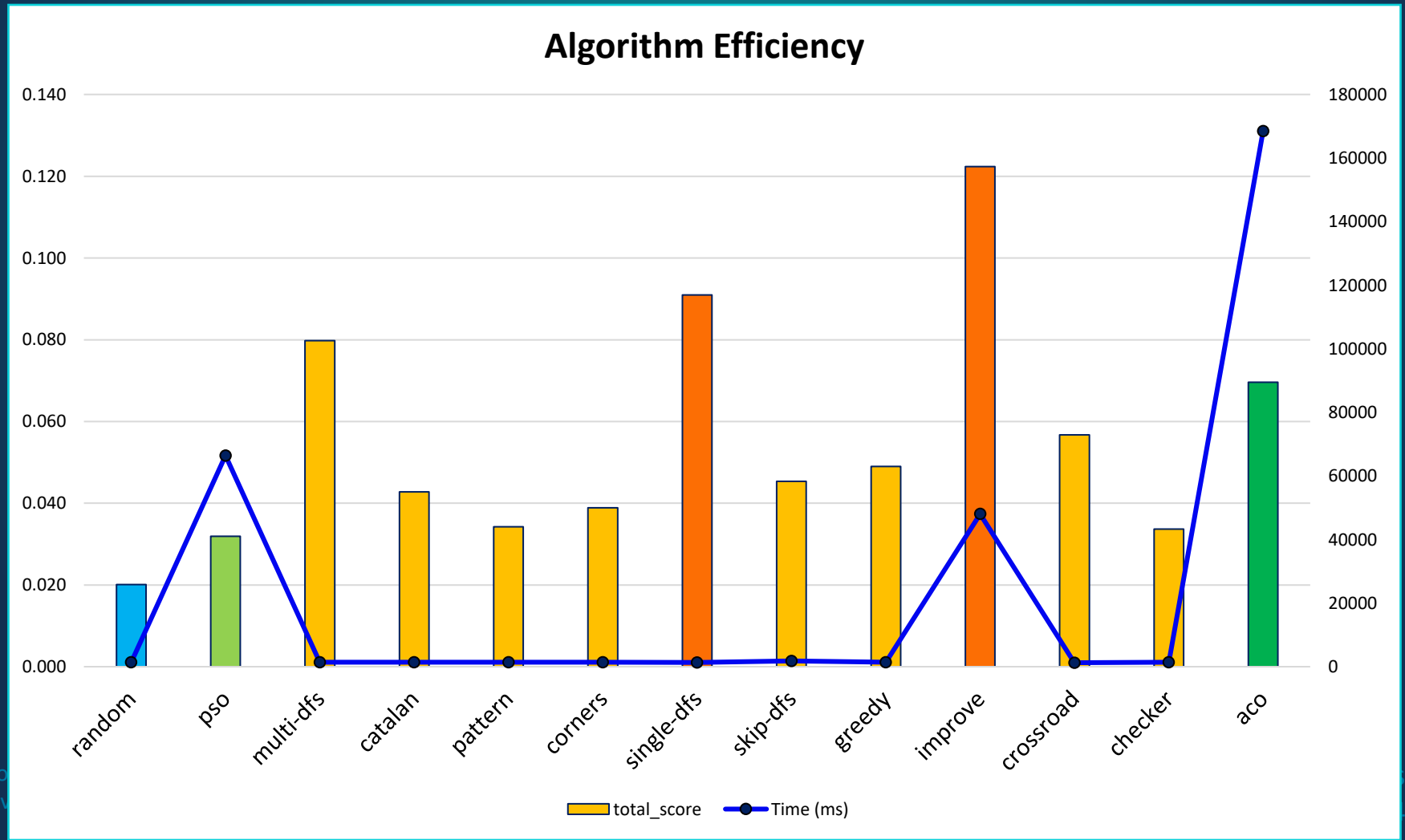
Got a question? Send an email to
andrius.gasiukevicius@mif.stud.vu

Algorithm Performance (5)



Got a question? Send an email to
andrius.gasiukevicius@mif.stud.vu

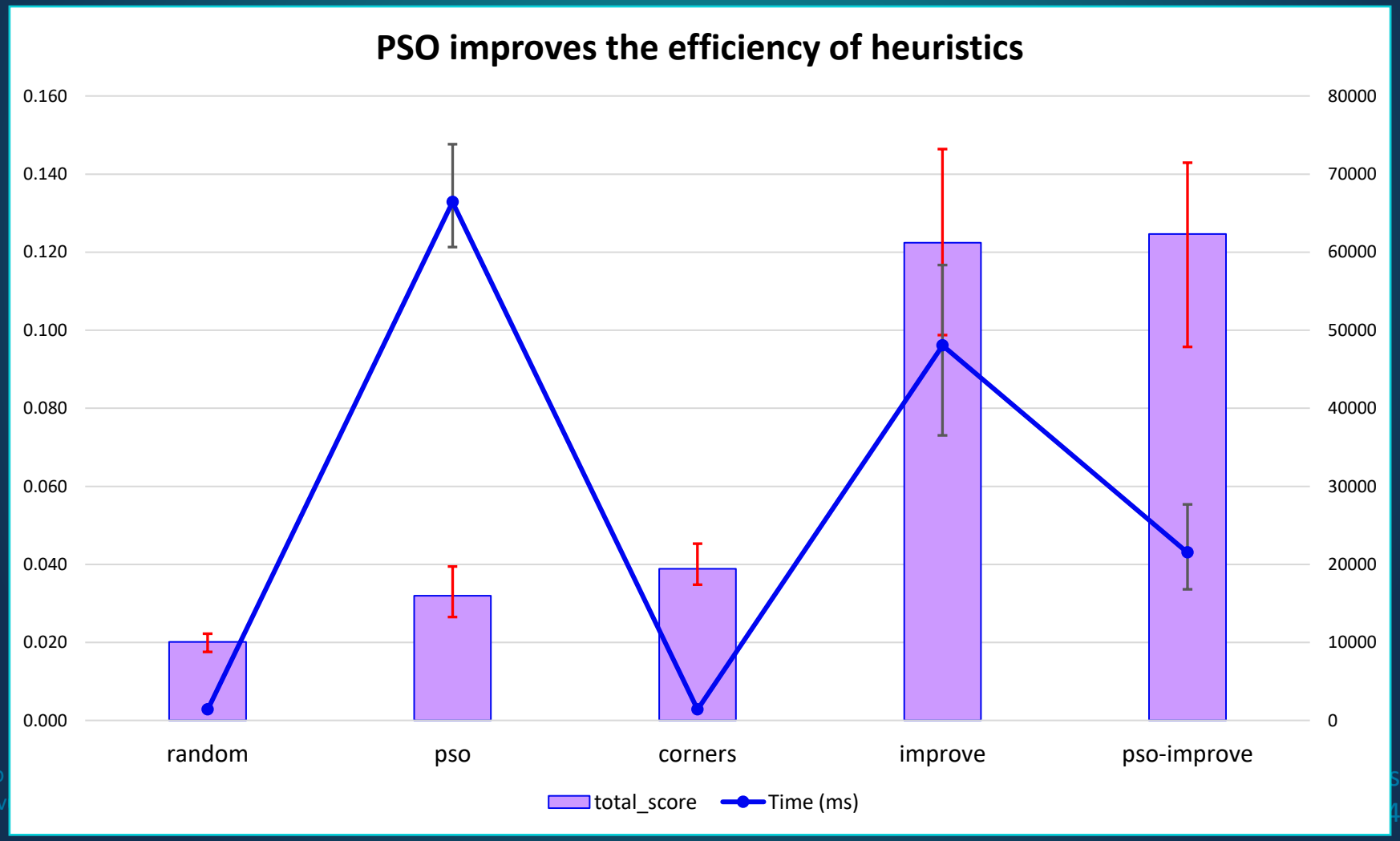
Overall Comparison



Got a question? Send an email to
andrius.gasiukevicius@mif.stud.v

VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS

Combining Heuristics



Got a question? Send an email to
andrius.gasiukevicius@mif.stud.v

Discussion

- Limitations
  - Data Sample
  - Potential for future research
- Heuristics for modern problems
- No Free Lunch, No Free Solutions
- More responses $\Longrightarrow$ more ideas

Got a question? Send an email to
andrius.gasiukevicius@mif.stud.vu.lt

Conclusion

- Lots of potential, but we need to do more research.
- A variety of heuristics can be used in problem-solving.
- In certain cases, intuition beats classical heuristics.
- Combining heuristics can improve their efficiency.
  - A good way to improve a ‘greedy’ intuitive algorithm is to combine it with a classical heuristic algorithm.

Got a question? Send an email to
andrius.gasiukevicius@mif.stud.vu.lt

VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS

References

Clerc, M. (2012) *Standard Particle Swarm Optimisation, HAL Open Access Archive.* Available at: https://hal.science/hal-00764996v1 (Accessed: 4 December 2024).

Colorni, A. *et al.* (1996) *Heuristics from nature for hard combinatorial optimization problems*, *International Transactions in Operational Research*. Available at: https://www.sciencedirect.com/science/article/abs/pii/0969601696000044 (Accessed: 26 October 2024).

Gad, A.G. (2022) *Particle swarm optimization algorithm and its applications: A systematic review - archives of computational methods in engineering*, *SpringerLink*. Available at: https://link.springer.com/article/10.1007/S11831-021-09694-4 (Accessed: 26 October 2024).

Saremi, S. *et al.* (2017) *Grasshopper optimisation algorithm: Theory and application*, *Advances in Engineering Software*. Available at: https://www.sciencedirect.com/science/article/abs/pii/S0965997816305646 (Accessed: 26 October 2024).

https://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms

https://imgflip.com/memegenerator

Got a question? Send an email to andrius.gasiukevicius@mif.stud.vu.lt

Referenced websites

https://www.youtube.com/watch?v=w6xl0IWrZYc&list=PLYrUuzjPm0uAMNkcC8wwet5ygaEc4EQqG&index=5

https://forms.gle/ewEBZCS1N4odH4Ku7

https://github.com/Andrius-G/Maze-optimisation-project/tree/master

Got a question? Send an email to
andrius.gasiukevicius@mif.stud.vu.lt

VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS

Thanks for listening!

- Questions???
- andrius.gasiukevicius@mif.stud.vu.lt

Additional Content

```
f(k,0,POTIONS-PARTICLES)
{
    f(i,0,PARTICLES-1)
    {
        f(d,0,GUESSES-1)
        {
            float& v_x = v[i][d][0];
            float& v_y = v[i][d][1];
            float r_p = frand(), r_g = frand(); //random cognitive and social parameters
            //update velocity <-> move towards 'best' pos
            v_x=(pso_inertia_weight*v_x)+(pso_phi_p*r_p*(x_best[i][d][0]-x[i][d][0]))+(pso_phi_g*r_g*(G[d][0]-x[i][d][0]));
            v_y=(pso_inertia_weight*v_y)+(pso_phi_p*r_p*(x_best[i][d][1]-x[i][d][1]))+(pso_phi_g*r_g*(G[d][1]-x[i][d][1]));
            //update particle position
            x[i][d][0]+=v_x;
            if(x[i][d][0]<0)x[i][d][0]=0;
            if(x[i][d][0]>=n)x[i][d][0]=n-1;
            x[i][d][1]+=v_y;
            if(x[i][d][1]<0)x[i][d][1]=0;
            if(x[i][d][1]>=n)x[i][d][1]=n-1;
        }//update each dimension of particle
```

Got a question? Send an email to
andrius.gasiukevicius@mif.stud.vu.lt

VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS