



# A GPU-based Genetic Algorithm for the Set Cover Problem

---

Andres Gonzalez

Advisor: Dr. Martin Burtscher

Fall 2025 Independent Study



Department of Computer Science  
San Marcos, Texas



# Set Cover Problem (SCP)

- A classic combinatorial **optimization** problem
- Can be framed as a graph analytics problem
  - Given a graph  $G$  with vertices  $V$  and weighted hyperedges (“sets”)  $E$ , where  $G = (V, E)$
  - Find a subset of **hyperedges** (“sets”) whose union “**covers**” all vertices and **minimizes** the weight

# Modeling an SCP as a Graph Problem

- Find a subset of **hyperedges** (“sets”) whose union “**covers**” all vertices and **minimizes** the weight

$$\textit{Weights} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \end{pmatrix}$$

$$\textit{Incidence} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

# Modeling an SCP as a Graph Problem

- Find a subset of **hyperedges** (“sets”) whose union “**covers**” all vertices and **minimizes** the weight

$$\begin{array}{l} \text{Weights} = \\ \text{Incidence} = \end{array} \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Sets	1	2	3	4	5
1	1	1	0	0	0
2	0	1	1	1	0
3	0	0	0	0	1



# Modeling an SCP as a Graph Problem

- Find a subset of **hyperedges** (“sets”) whose union “**covers**” all vertices and **minimizes** the weight

$$\begin{array}{l} \text{Weights} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \end{pmatrix} \\ \text{Incidence} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

Weight = 9

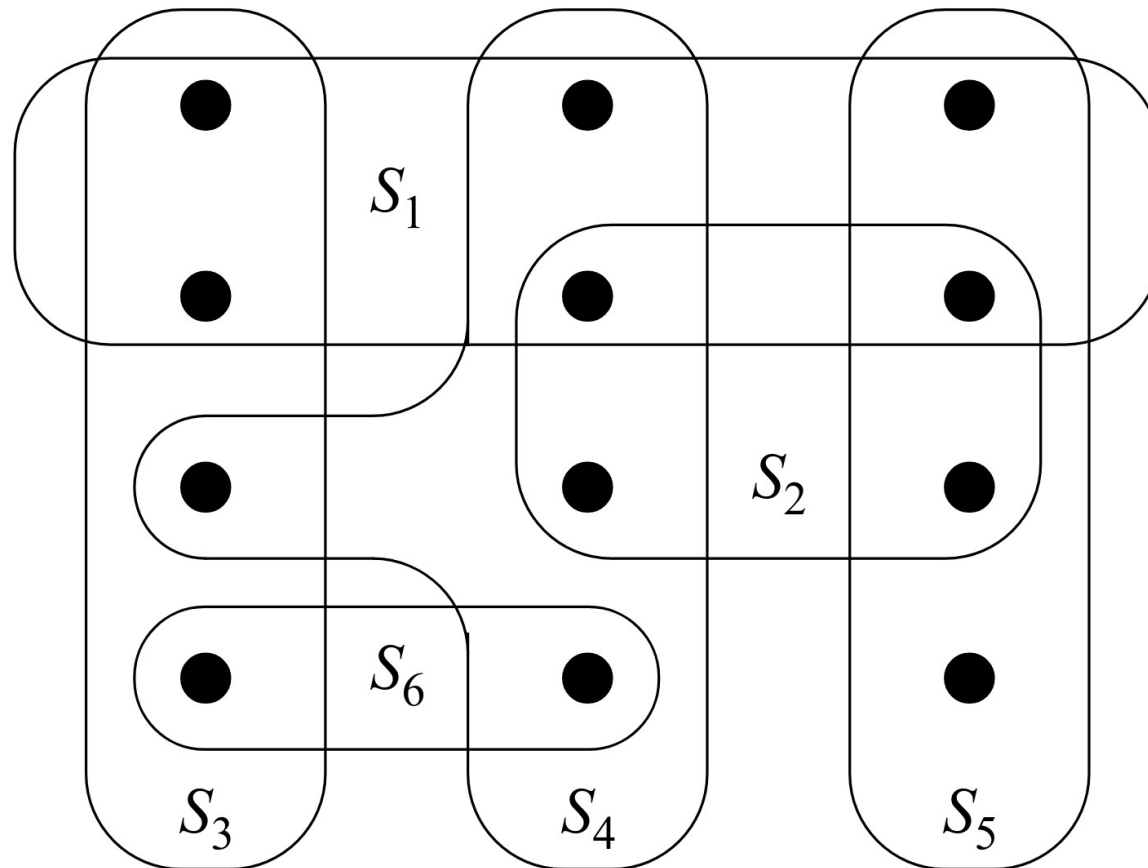
# Modeling an SCP as a Graph Problem

- Find a subset of **hyperedges** (“sets”) whose union “**covers**” all vertices and **minimizes** the weight

$$\begin{array}{l} \text{Weights} = (1 \quad 2 \quad 3 \quad 4 \quad 5) \\ \text{Incidence} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

Weight = 7

# Visualizing SCP With a Graph



# Set Cover Problem (SCP)

- NP-Hard
- Variations of SCP
  - Non-Unicost vs. Unicost
  - Partial Cover
  - Conflicts on Sets
  - Small Neighborhood Property
  - and more...



# SCP Applications

- Facility Location Selection
- Aircraft Crew Staffing
- Political Zoning
- Truck Transport
- Materials Management
- and more...



# SCP & Linear Programming (LP)

# Modeling an SCP Problem as LP

$$\textit{minimize} \quad \sum_{j=1}^n c_j x_j$$

$$\textit{s.t.} \quad x_j \in (0, 1) \quad j=1, \dots, n$$

Will explain shortly...

$$\sum_{j=1}^n a_{ij} x_j \geq 1 \quad i=1, \dots, m$$



# Linear Programming (LP) Example



# Modeling an LP Problem

- Two bookshelf models  $x_1$  and  $x_2$  each require raw material and machine time
- $x_1$ 
  - \$2 profit per unit
- $x_2$ 
  - \$4 profit per unit

$$P = 2x_1 + 4x_2$$

# Modeling an LP Problem

- Two bookshelf models  $x_1$  and  $x_2$  each require raw material and machine time

- $x_1$

- 3 m<sup>2</sup> of material

- $x_2$

- 4 m<sup>2</sup> of material

- Raw material is limited

- 1700 m<sup>2</sup> of material per week

$$3 x_1 + 4 x_2 \leq 1700$$

# Modeling an LP Problem

- Two bookshelf models  $x_1$  and  $x_2$  each require raw material and machine time

- $x_1$

- (1 / 5) hours

- $x_2$

- (1 / 2) hours

- Machine time is limited

- 160 machine hours per week

$$\frac{1}{5} x_1 + \frac{1}{2} x_2 \leq 160$$

# Modeling an LP Problem

- Two bookshelf models  $x_1$  and  $x_2$  each require raw material and machine time

- Profit (\$)

$$P = 2x_1 + 4x_2$$

- Material (m<sup>2</sup>)

$$3x_1 + 4x_2 \leq 1700$$

- Machine Time (hrs)

$$\frac{1}{5}x_1 + \frac{1}{2}x_2 \leq 160$$



# Modeling an LP Problem

$$P = 2 x_1 + 4 x_2$$

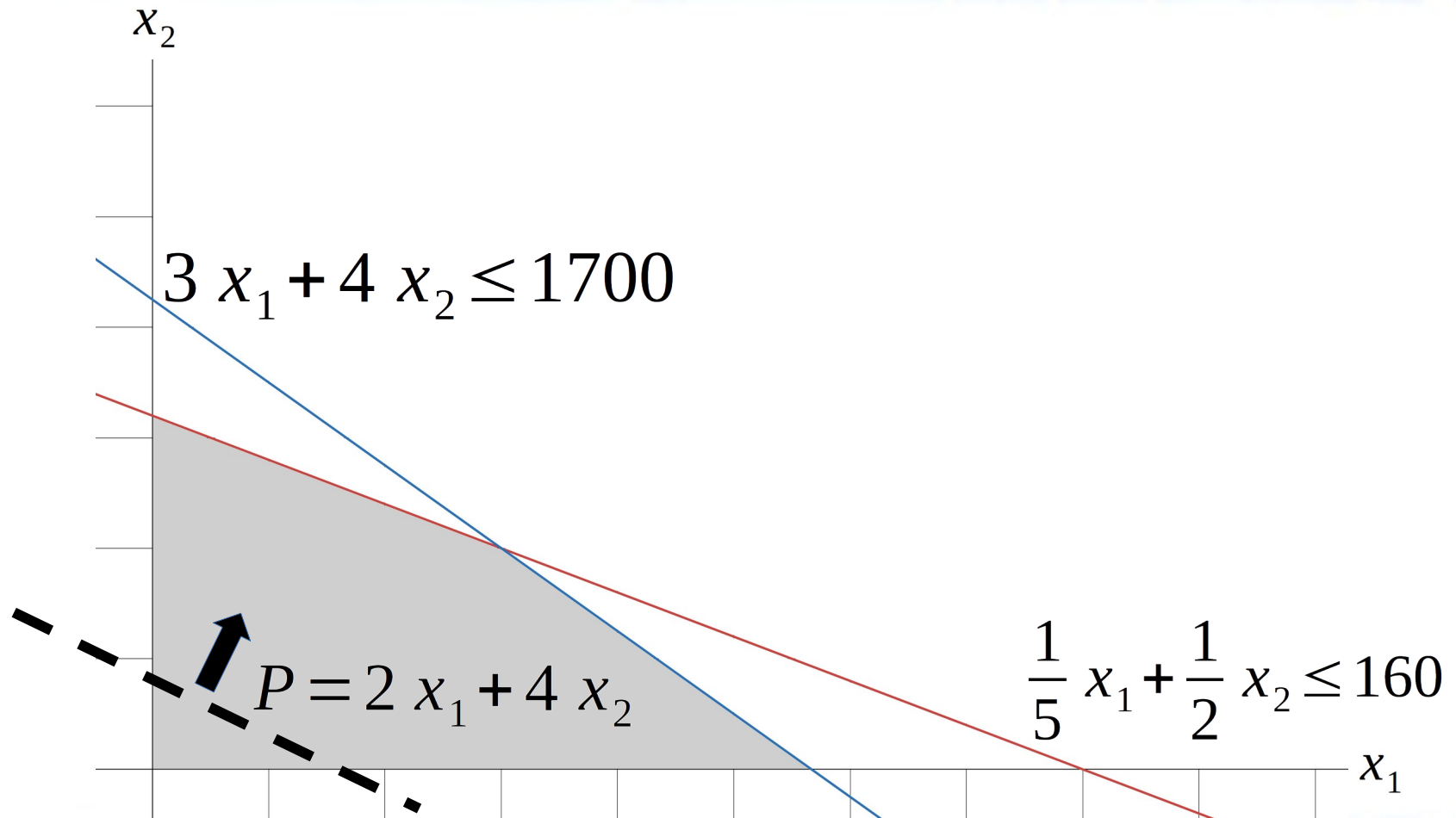
“Objective Function”

$$3 x_1 + 4 x_2 \leq 1700$$

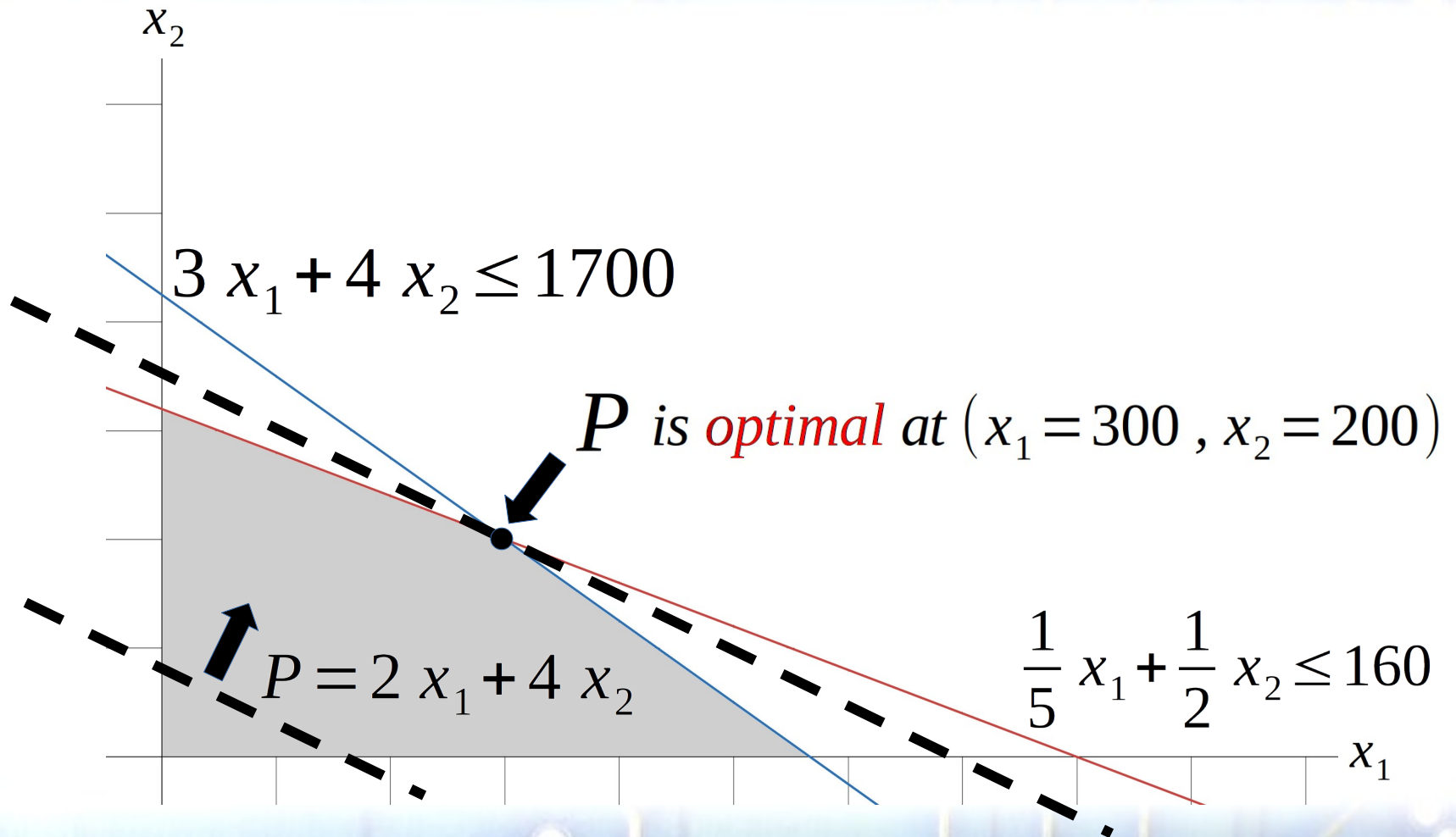
$$\frac{1}{5} x_1 + \frac{1}{2} x_2 \leq 160$$

“Constraints”

# Modeling an LP Problem



# Modeling an LP Problem



# Solving an LP Problem

- Graphical Method (aforementioned example)
- Simplex Method
- Interior Point Methods
- Branch-and-Bound Method
- Primal-Dual Hybrid Gradient
- and more...



# LP-Related Domains

- Integer Programming (IP)
- Binary Integer Programming (BIP)
- Mixed-Integer Programming (MIP)
- Mixed-Integer Non-Linear Programming (MINLP)
- [Non]Convex Quadratic Programming (QP)
- Fractional Linear Programming (FLP)
- Second-Order Cone Programming (SOCP)
- and more...



# Modeling an SCP as LP Problem

# Modeling an SCP as an LP Problem

$$P = 2 x_1 + 4 x_2$$

“Objective Function”

$$3 x_1 + 4 x_2 \leq 1700$$

$$\frac{1}{5} x_1 + \frac{1}{2} x_2 \leq 160$$

“Constraints”

# Modeling an SCP as an LP Problem

$$\text{minimize} \quad \sum_{j=1}^n c_j x_j$$

$c_j$  = cost of set

$$s.t. \quad x_j \in (0, 1) \quad j=1, \dots, n$$

$$\sum_{j=1}^n a_{ij} x_j \geq 1 \quad i=1, \dots, m$$

$a_{ij}$  = incidence matrix



# Modeling an SCP as an LP Problem

$$\sum_{j=1}^n a_{ij} x_j \geq 1 \quad i=1, \dots, m$$

$$\text{Incidence} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{matrix} \geq 1 \\ \geq 1 \\ \geq 1 \end{matrix}$$

$a_{ij}$  = incidence matrix



# SCP Benchmarks

# Representing an SCP Problem

- Balas, Ho, and Beasley
    - Published benchmark set in 1987
    - Expanded in 1990, 1997 across publications
    - Problem sets
      - “4 to 6”
      - “A – H”
- MIPLIB 2017



# Beasley **SCP** Benchmark Compositions

Problem Set	Vertices	Sets	Density
4	200	1000	2%
5	200	2000	2%
6	200	1000	5%
A	300	3000	2%
B	300	3000	5%
C	400	4000	2%
D	400	4000	5%
E	50	500	20%
NRE	500	5000	10%
NRF	500	5000	20%
NRG	1000	10000	2%
NRH	1000	10000	5%

Only Unicost  
Problem Set

Problem Instance	Vertices	Sets	Density
Rail507	507	63009	1.3%
Rail516	516	47311	1.3%
Rail582	582	55515	1.2%
Rail2536	2536	1081841	0.4%
Rail2586	2586	92683	0.3%
Rail4284	4284	1092610	0.2%
Rail4872	4872	968672	0.2%



# MIPLIB 2017

MIPLIB 2017 About Benchmark Collection Download Help

## The Set Covering Tag

All instances with set covering constraints after presolving.  
Browse below all 199 instances. Find all relevant downloads on our [Downloads](#) page.

Get Column Copy CSV Excel PDF Print Column visibility

Show **all** entries Search:

Instances tagged with the Set Covering Tag. Objective values of open instances are marked with '\*'.

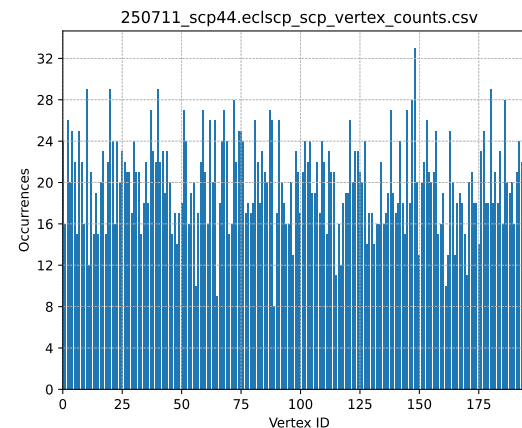
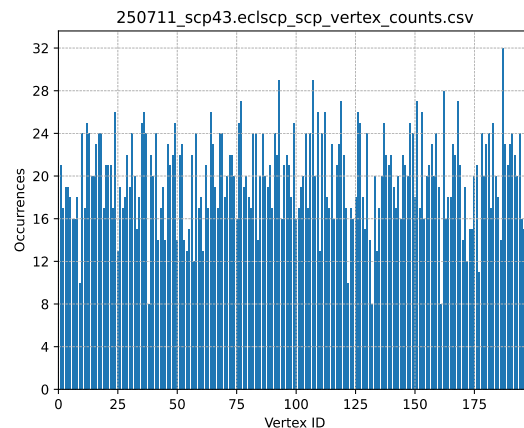
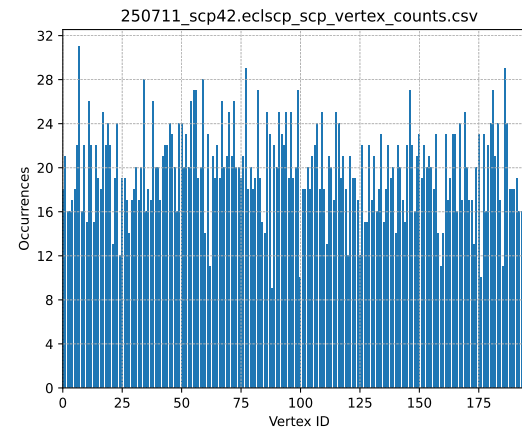
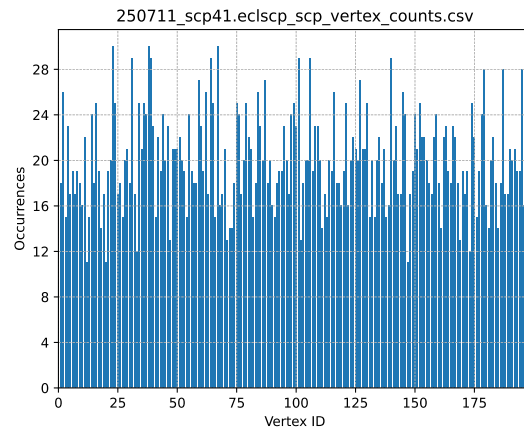
Instance Status Variables Binaries Integers Continuous Constraints Nonz. Submitter Group Objective Tags

											knapsack
<a href="#">scpj4scip</a>	open	99947	99947	0	0	1000	999893	Shunji Umetani	scp	128*	binary set_covering
<a href="#">scpk4</a>	open	1e+05	1e+05	0	0	2000	1000000	Shunji Umetani	scp	318.0*	binary set_covering
<a href="#">scpl4</a>	open	2e+05	2e+05	0	0	2000	2000000	Shunji Umetani	scp	259.0*	binary set_covering
<a href="#">scpm1</a>	open	5e+05	5e+05	0	0	5000	6250000	Shunji Umetani	scp	540.0*	binary set_covering
<a href="#">scpn2</a>	open	1e+06	1e+06	0	0	5000	12500000	Shunji Umetani	scp	485.0*	binary set_covering

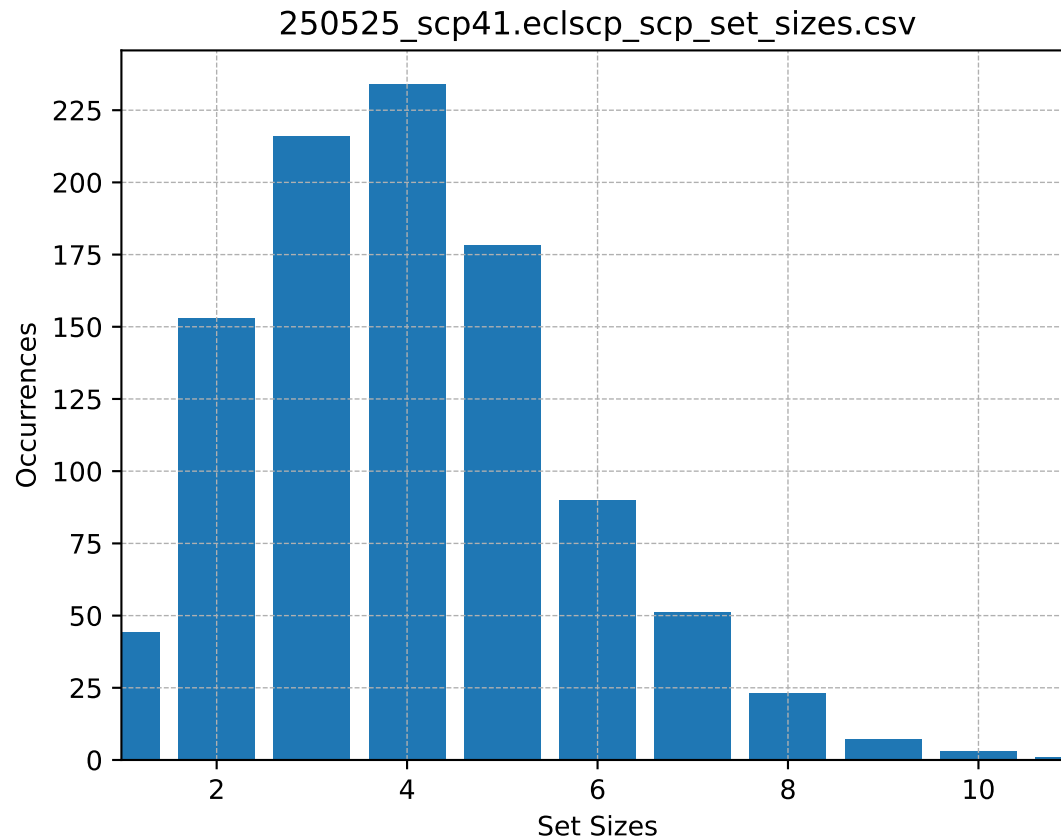
Some benchmarks are still “open” (optimal solution unknown)!

<https://miplib.zib.de/>

# Beasley **SCP** Benchmark Compositions



# Beasley **SCP** Benchmark Compositions



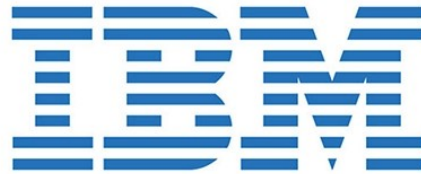


# Related Work



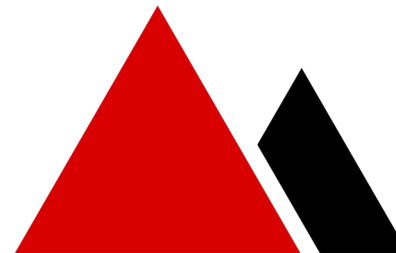
# Closed-Source Implementations

- CPLEX (IBM)
- Gurobi
- Xpress (FICO)
- Mosek



# Open-Source Implementations

- cuOpt (NVIDIA – GPU)
  - Uses Simplex Method
- CyLP / CBC (COIN-OR – CPU)
  - Uses Presolving
- HiGHS (ERGO-Code – CPU)
  - Uses Dual Simplex
- SCIP (SCIP-OPT – CPU)
  - Uses Primal / Dual





# ECL-SCP Implementations

# Experimental Methodology

- AMD Ryzen **Threadripper 2950X**, 16 cores, 3.5GHz, gcc 13.3

	PE	SM	PE/SM	Mem (GB)	NVCC
<b>RTX 4090</b>	16384	128	128	24	12.6
<b>RTX 3090</b>	10496	82	128	24	12.0
<b>A100</b>	6912	108	64	40	12.0
<b>GTX 1650</b>	1024	16	64	4	12.6

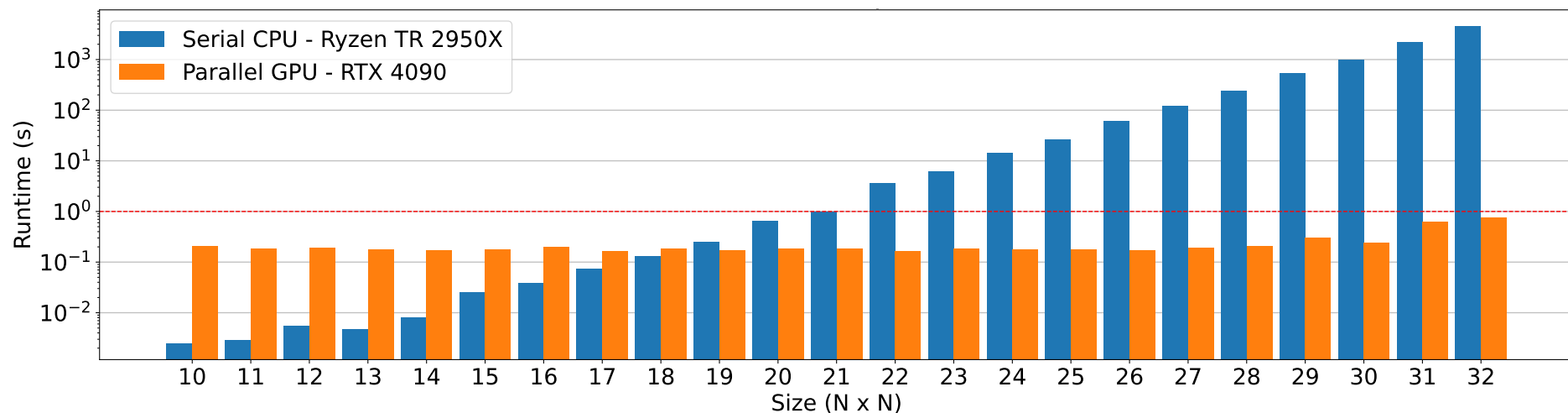




# ECL-SCP **Brute-Force** Implementation

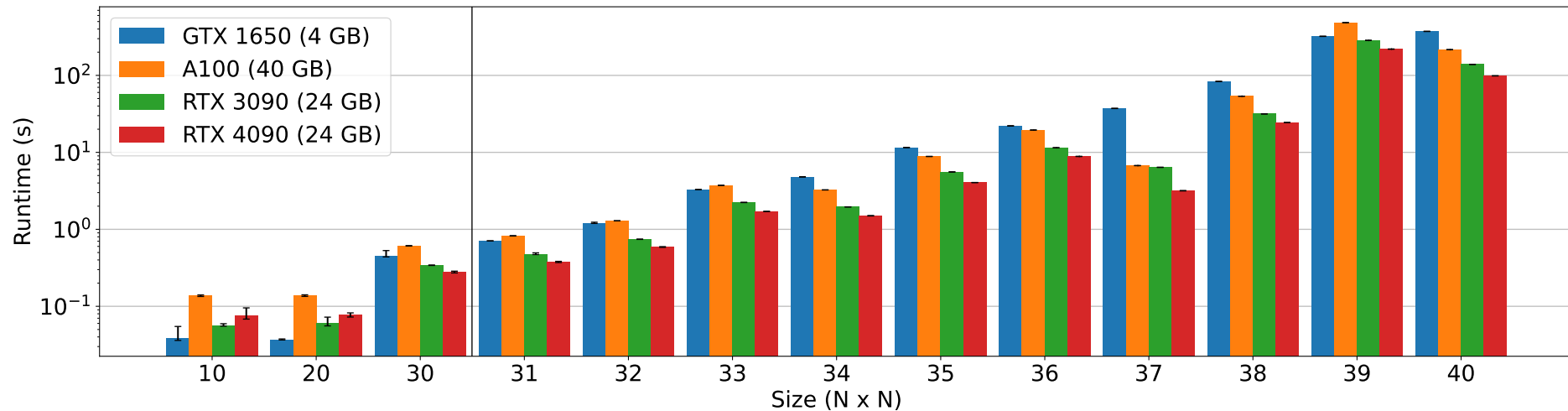
# ECL-SCP Naïve CPU/GPU Brute Force

- Exhaustively iterate over all N-bit combinations
  - Runtime (s) vs. SCP Size - Ser-CPU to Par-GPU



# ECL-SCP Naïve GPU Brute Force

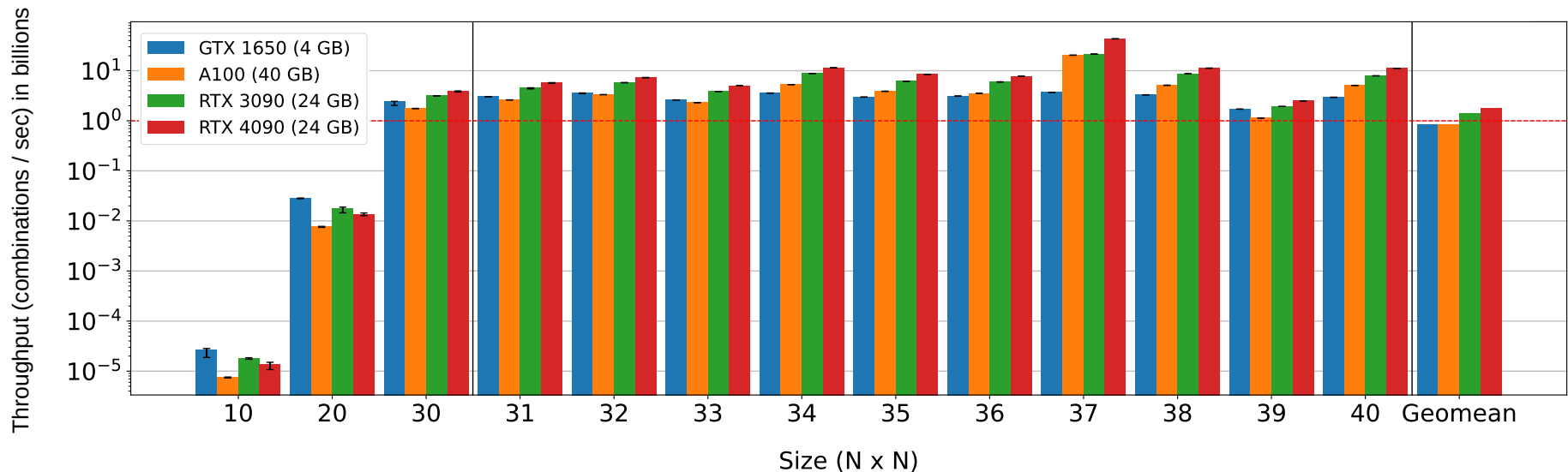
- With 40 bits, runtime is already > 100 sec!
  - Runtime (s) vs. SCP Size – Device Comparison



Only 10 more bits than CPU!

# ECL-SCP Naïve GPU Brute Force

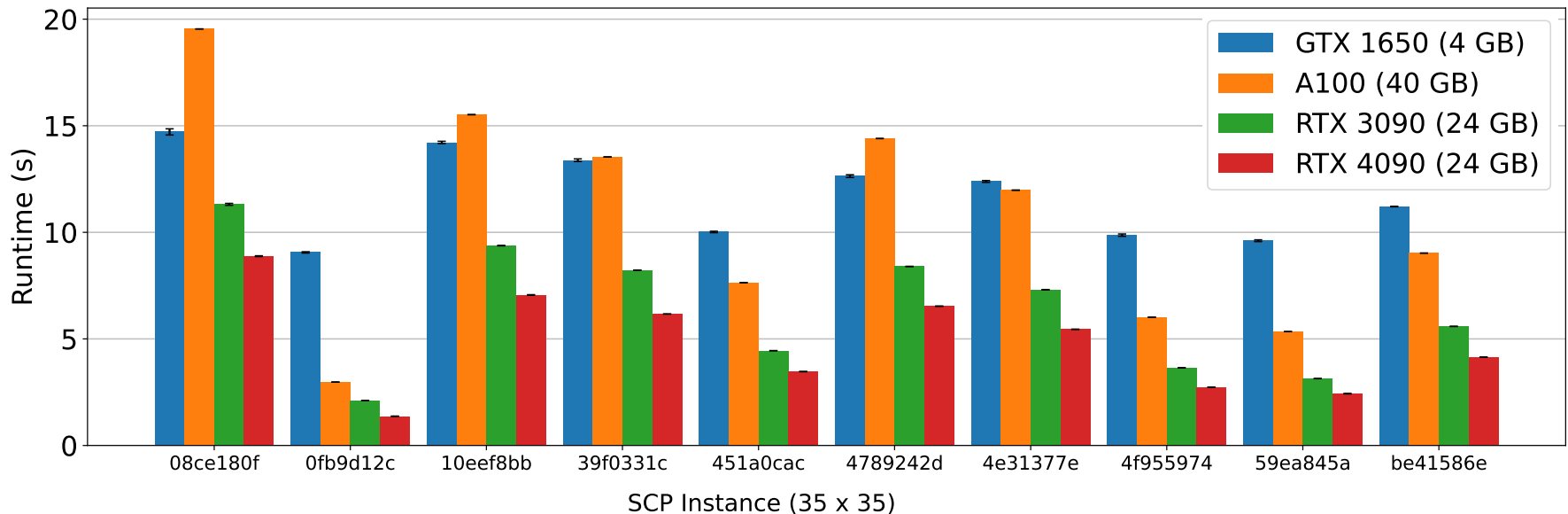
- GPUs become saturated after 30 bits
  - Throughput (combinations / s) in billions





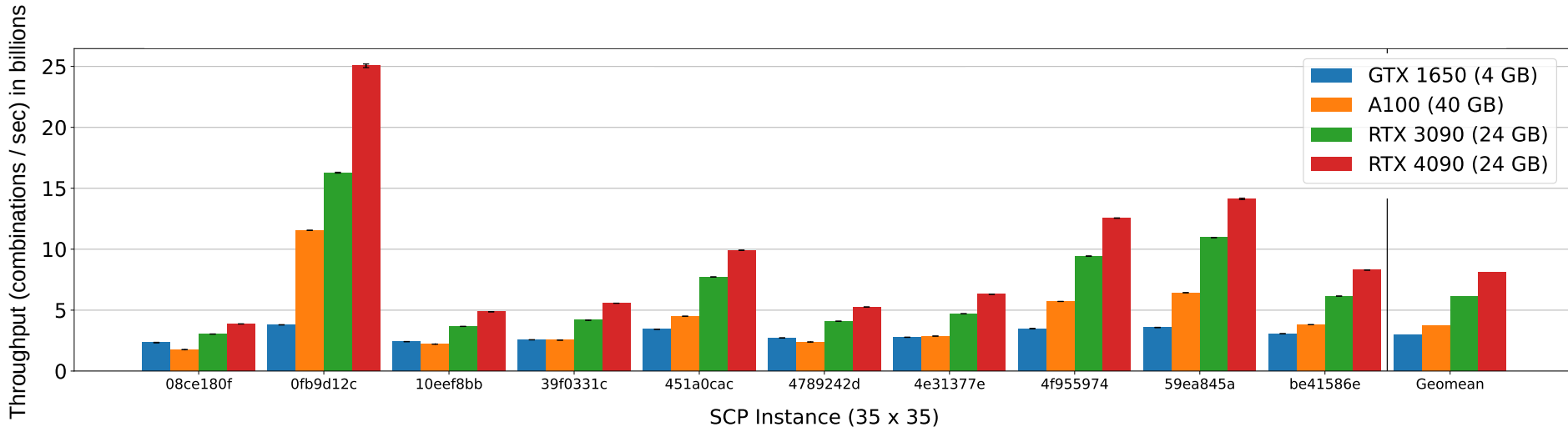
# ECL-SCP Naïve GPU Brute Force

- Benchmark **set compositions** affect GPU runtimes
  - Runtime (s) vs. SCP (35 x 35) – Device Comparison



# ECL-SCP Naïve GPU Brute Force

- Reached throughput of **25 billion 35-bit** set combinations per second!
  - Throughput (combinations / s) in billions

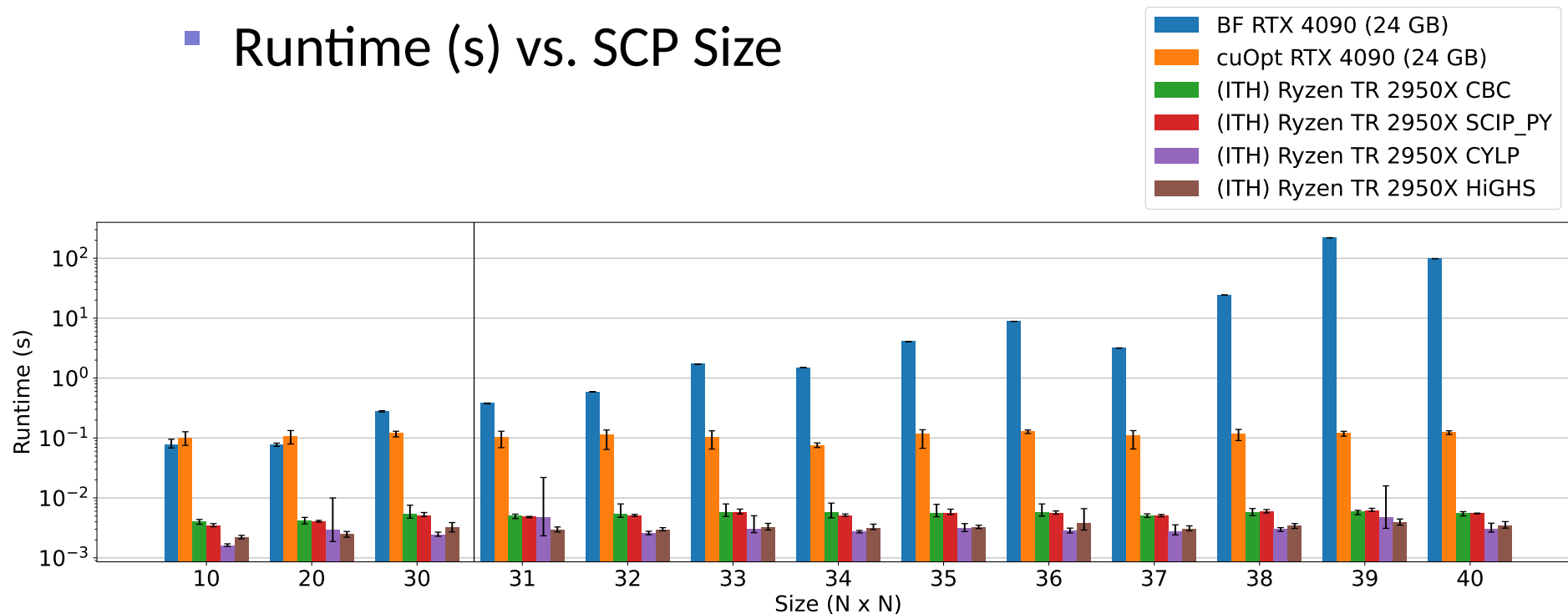




# Third-Party Comparisons

# ECL-SCP BF Against 3<sup>rd</sup> Party

- Brute Force is **not** the best approach
  - cuOpt is also order of magnitude slower
  - Runtime (s) vs. SCP Size

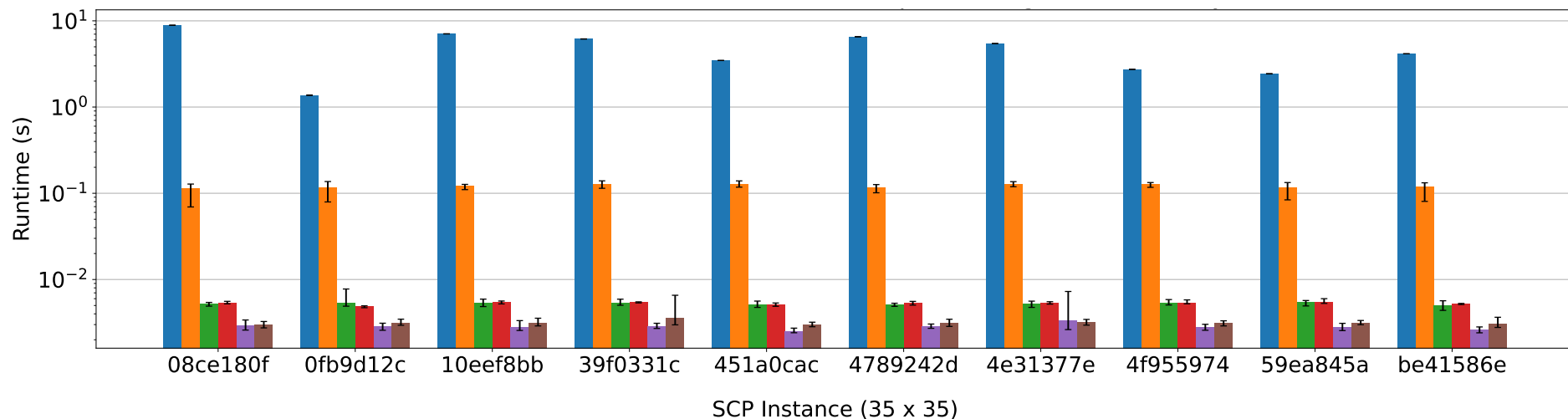




# ECL-SCP BF Against 3<sup>rd</sup> Party

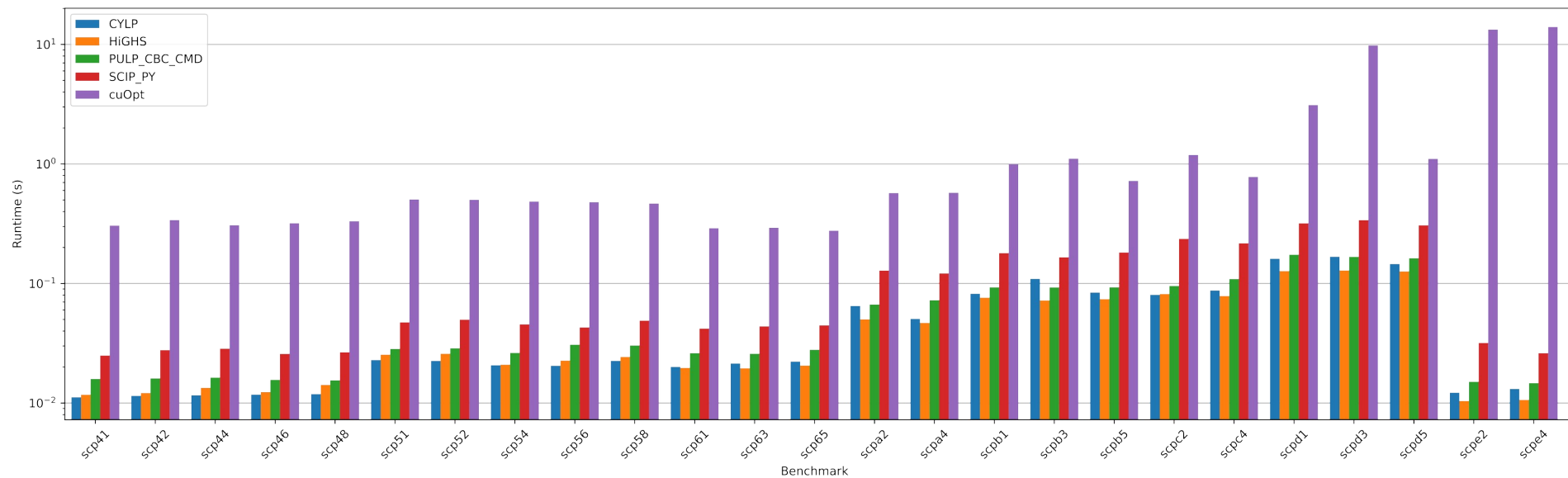
- **Less variation** in runtime against **set composition**

- Perhaps inputs are too small?
- Runtime (s) vs. SCP (35 x 35)



# 3<sup>rd</sup> Party on Beasley SCPs

- Open-source solvers take the lead on small SCPs
  - Runtime (s) vs. Beasley SCP

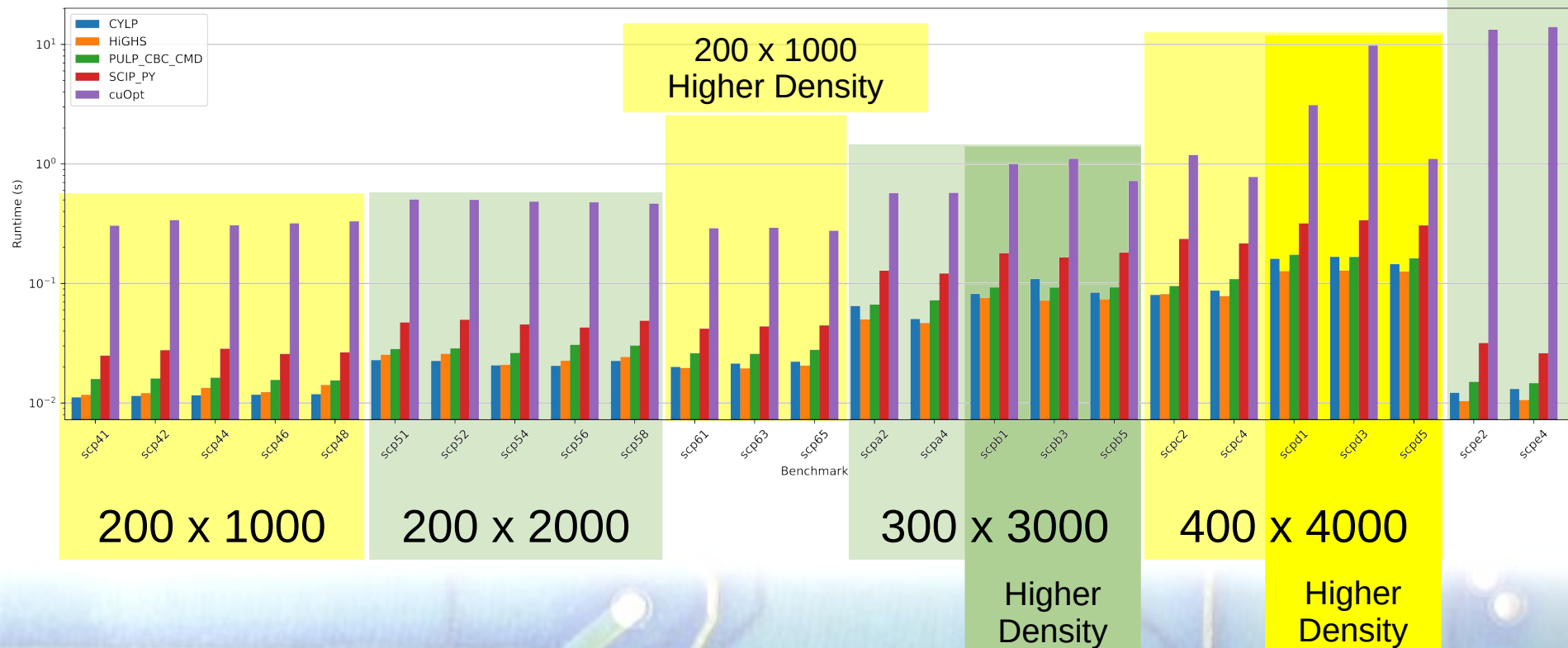


# 3<sup>rd</sup> Party on Beasley SCPs

- Open-source solvers take the lead on small SCPs

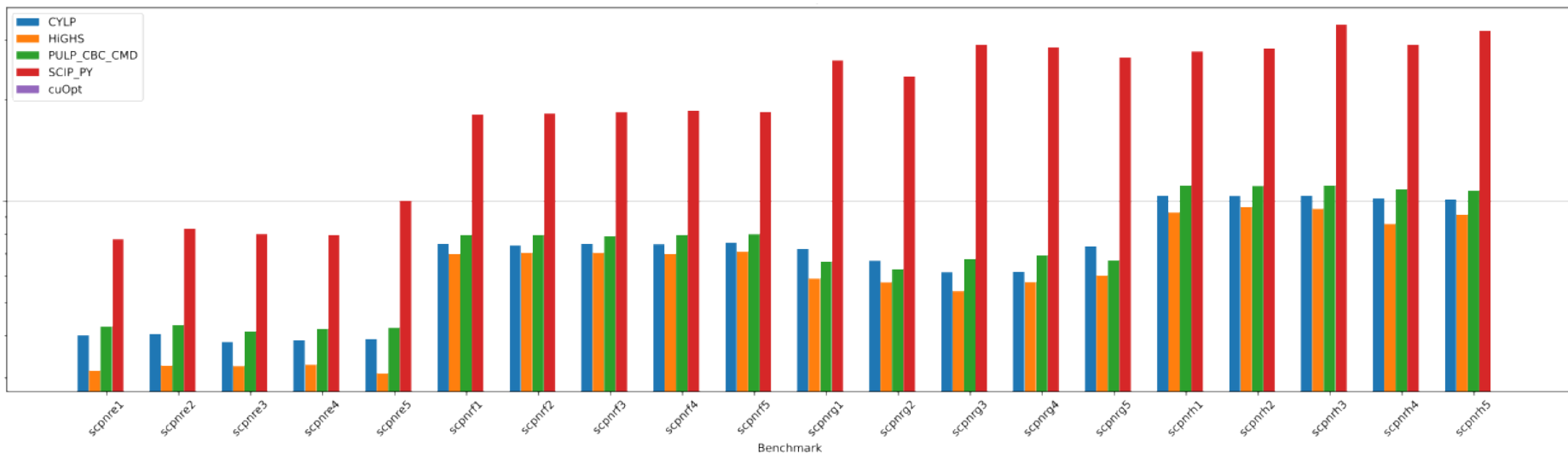
- Runtime (s) vs. Beasley SCP

50 x 500 (unicost)



# 3<sup>rd</sup> Party on Beasley SCPs

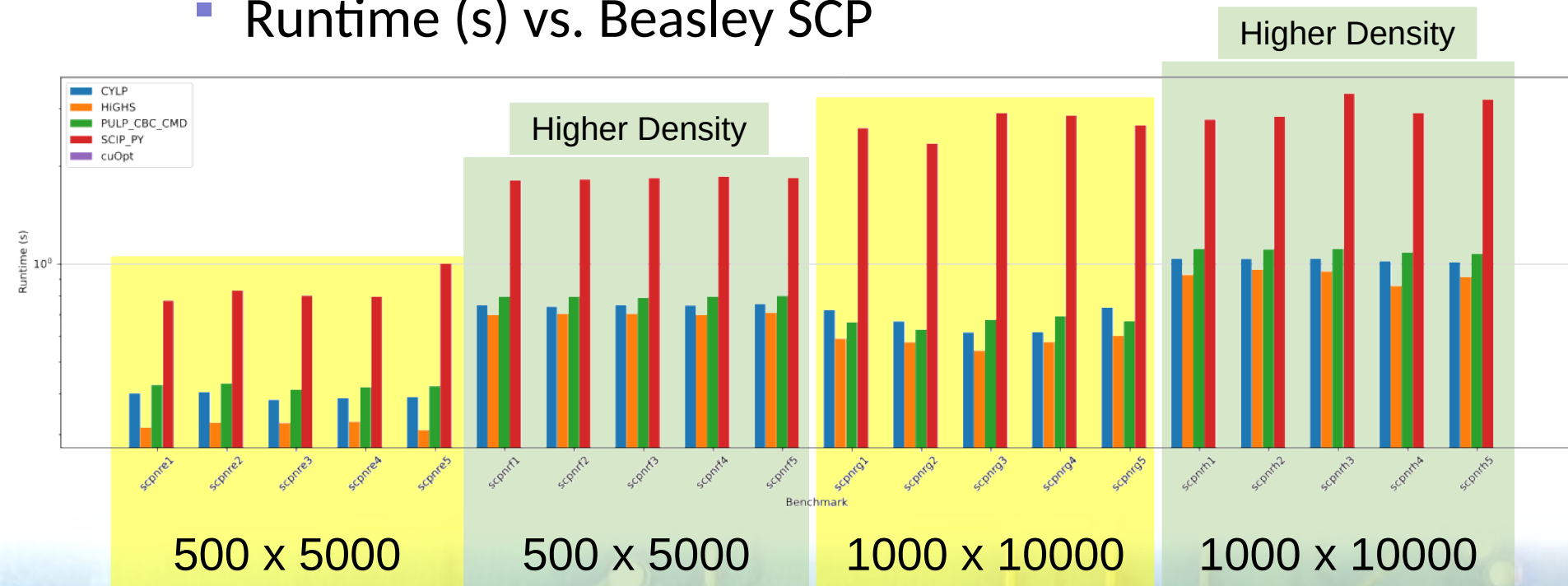
- Open-source solvers take the lead on small SCPs
  - cuOpt times out for all of these inputs at 5 minutes
  - Runtime (s) vs. Beasley SCP





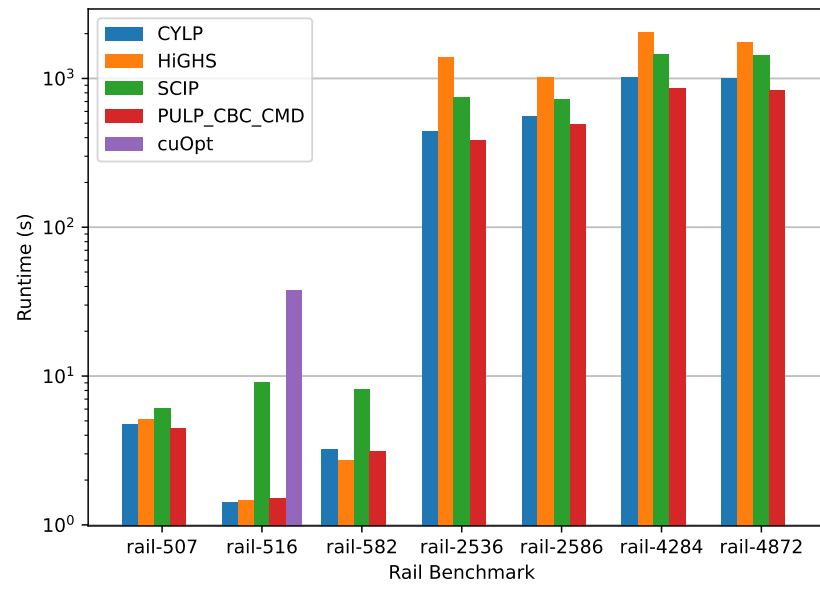
# 3<sup>rd</sup> Party on Beasley SCPs

- Open-source solvers take the lead on small SCPs
  - cuOpt times out for all of these inputs at 5 minutes
  - Runtime (s) vs. Beasley SCP



# 3<sup>rd</sup> Party on Beasley SCPs

- Rail benchmarks have room for improvement
  - cuOpt times out for all but one input at 2 hours
  - Runtime (s) vs. Beasley SCP – Rail benchmarks



# 3<sup>rd</sup> Party Codes with Large Inputs

```
scpk4.mps.SCIIP_PY.output X
SCP > project_exhaustive_brute_force_gpu > third_party_tests > PuLP > scpk4.mps.SCIIP_PY.output
51 SCIP Status      : solving was interrupted [time limit reached]
52 Solving Time (sec) : 7200.00
53 Solving Nodes      : 250 (total of 251 nodes in 2 runs)
54 Primal Bound       : +3.310000000000000e+02 (711 solutions)
55 Dual Bound        : +2.82670772247488e+02
56 Gap               : 17.10 %
57 ***** Solve time: 7202.799690224 seconds
```

```
scpk4.mps.PULP_CBC_CMD.output X
SCP > project_exhaustive_brute_force_gpu > third_party_tests > PuLP > scpk4.mps.PULP_CBC_CMD.output
285 ZeroHalf was tried 1 times and created 0 cuts (0.722 seconds)
286
287 Result - Stopped on time limit
288
289 Objective value:      359.00
290 Lower bound:         280.544
291 Gap:                 0.28
292 Enumerated nodes:     11092
293 Total iterations:     3707163
294 Time (CPU seconds):   7160.05
295 Time (Wallclock seconds): 7219.49
296
297 Option for printingOptions changed from normal to all
298 Total time (CPU seconds): 7160.45 (Wallclock seconds): 7219.92
299
```

Large inputs (2,000 x 100,000)  
time out at 2 hours

```
scpk4.mps.HIGHS.output X
SCP > project_exhaustive_brute_force_gpu > third_party_tests > PuLP > scpk4.mps.HIGHS.output
257 Model after restart has 2000 rows, 50299 cols (50299 nonzeros), 0 int., 0 impl., 0 cont., 0 dom.fix.), and 504224 nonzeros
258
259 4853 25 1 0.00% 281.552458 31 14.94% 5 1 8 3864k 7200.0s
260
261 Solving report
262 Status      Time limit reached
263 Primal bound 331
264 Dual bound 282
265 Gap 14.8% (tolerance: 0.01%)
266 P-D integral 1518.41161135
267 Solution status feasible
268 331 (objective)
269 0 (bound viol.)
270 0 (int. viol.)
271 0 (row viol.)
272 Timing 7200.02
273 Max sub-MIP depth 12
274 Nodes 4853
275 Repair LPs 0
276 LP iterations 3864254
277 1013484 (strong br.)
278 1940 (separation)
279 1024689 (heuristics)
```

```
scpk4.mps.CVLPout.output X
SCP > project_exhaustive_brute_force_gpu > third_party_tests > PuLP > scpk4.mps.CVLPout.output
194 Result - Stopped on time limit
195
196 Objective value:      376.00
197 Lower bound:         282.503
198 Gap:                 0.33
199 Enumerated nodes:     11337
200 Total iterations:     3705678
201 Time (CPU seconds):   7200.61
202 Time (Wallclock seconds): 7248.81
203
204 Total time (CPU seconds): 7200.61 (Wallclock seconds): 7248.81
205
206 stopped on time
```



# ECL-SCP

## Maximal Independent Set (“MIS”) and Genetic Algorithm (“GA”) Parallel GPU Implementation



# ECL-SCP Phase I - MIS

- First, find an **MIS** for vertices with respect to their “chosen” cheapest set
- If **all vertices agree** on that set, that set gets added to the **running solution**
  - The same applies to all other “agreed sets”
- Repeat for all vertices that remain, excluding already accounted-for vertices

## ECL-SCP Phase II – GA

- Second, feed the solution obtained from the **MIS** phase into the **GA**
- Simulate generations of the **population of organisms**, taking into account **organism fitness**
  - **Fitness** = **Feasibility** + (1 / **Cost**)
- Organisms with **higher fitness** are more likely to generate offspring in the following generation

## ECL-SCP Phase II – GA

- The **most-fit organism** always survives
- Creating the next generation of organisms (the new population) consists of
  - Generating new offspring (“crossover op”)
  - Mutating the alleles (“mutation op”)
  - Minimization
  - Ensuring feasibility

# ECL-SCP Phase II – GA Population

“Alleles” == Sets

Population of organisms

1111000001 ...

1000110111 ...

0010000110 ...

0101101001 ...

1100010101 ...

0011001101 ...

0000010011 ...

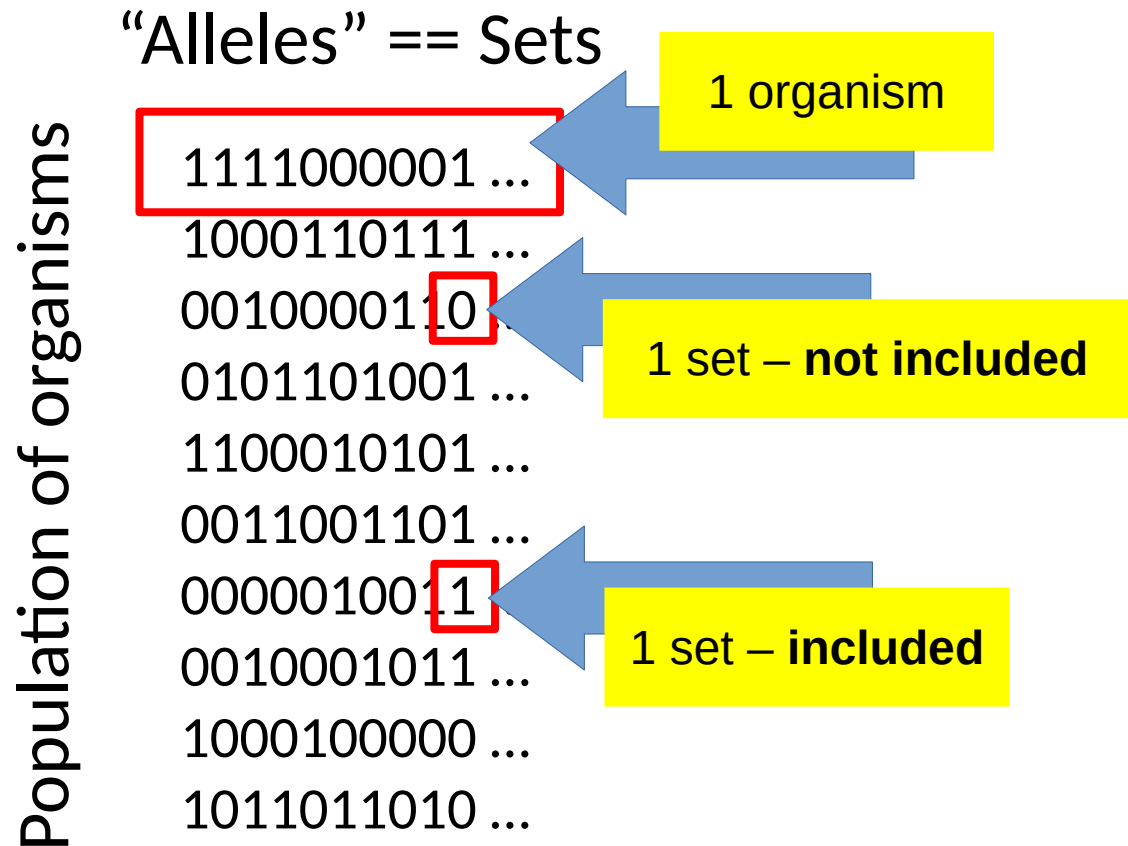
0010001011 ...

1000100000 ...

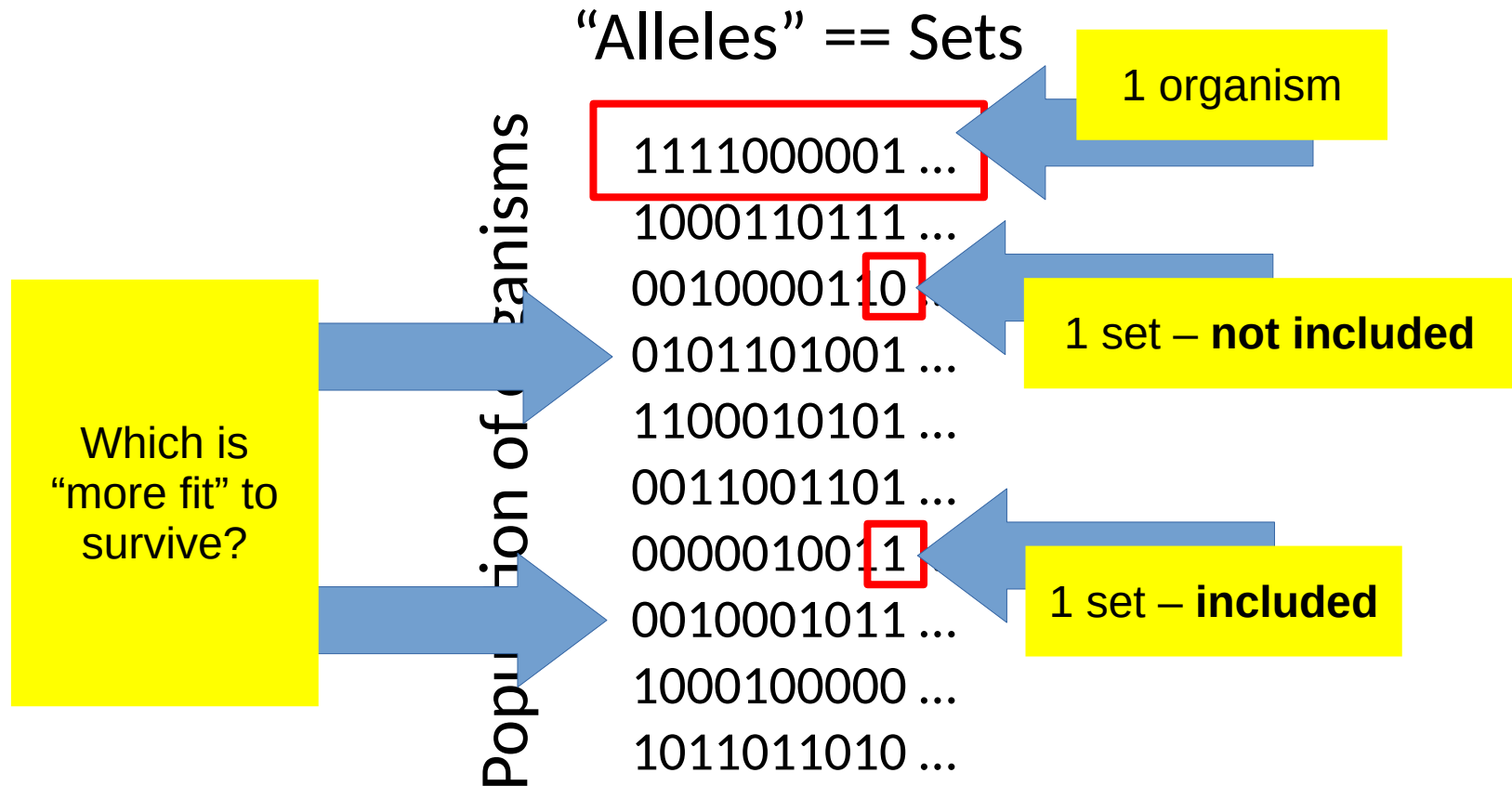
1011011010 ...



# ECL-SCP Phase II – GA Population

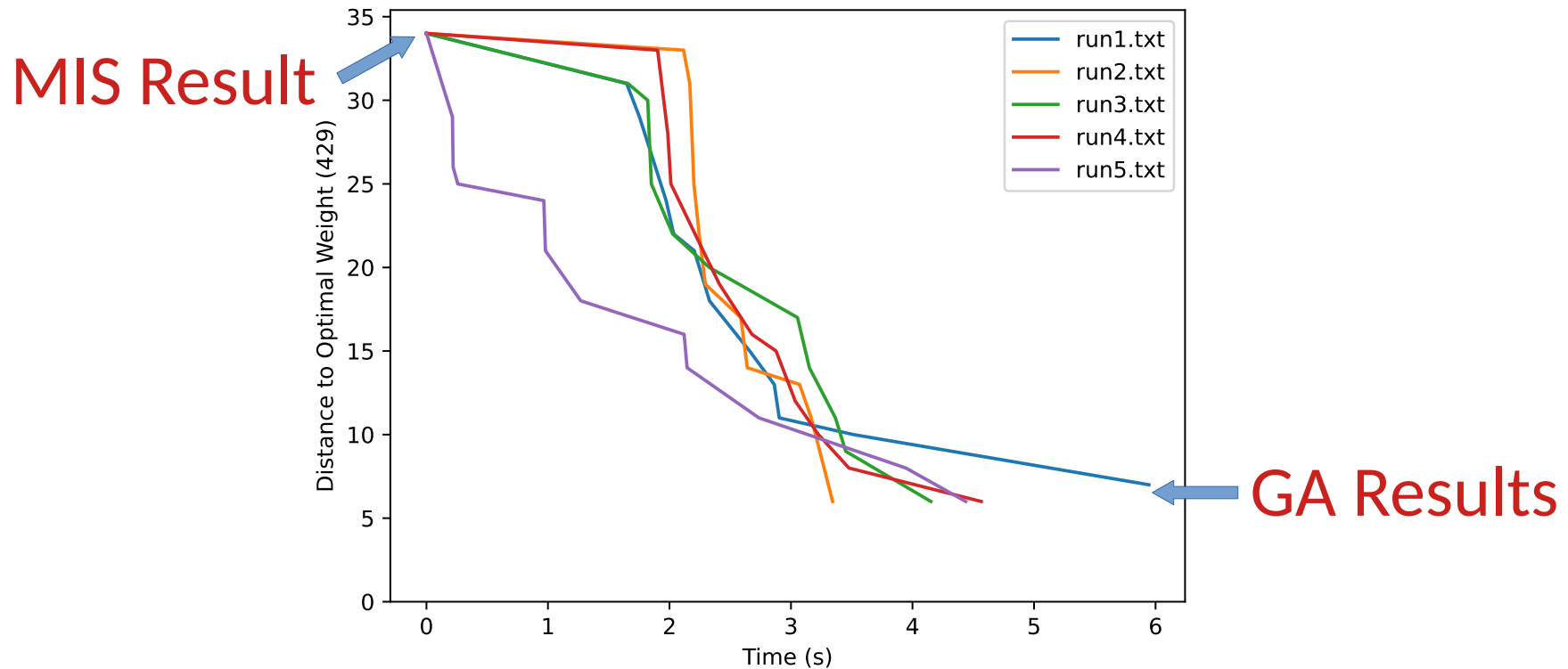


# ECL-SCP Phase II – GA Population



# ECL-SCP MIS + GA on (200 x 1000)

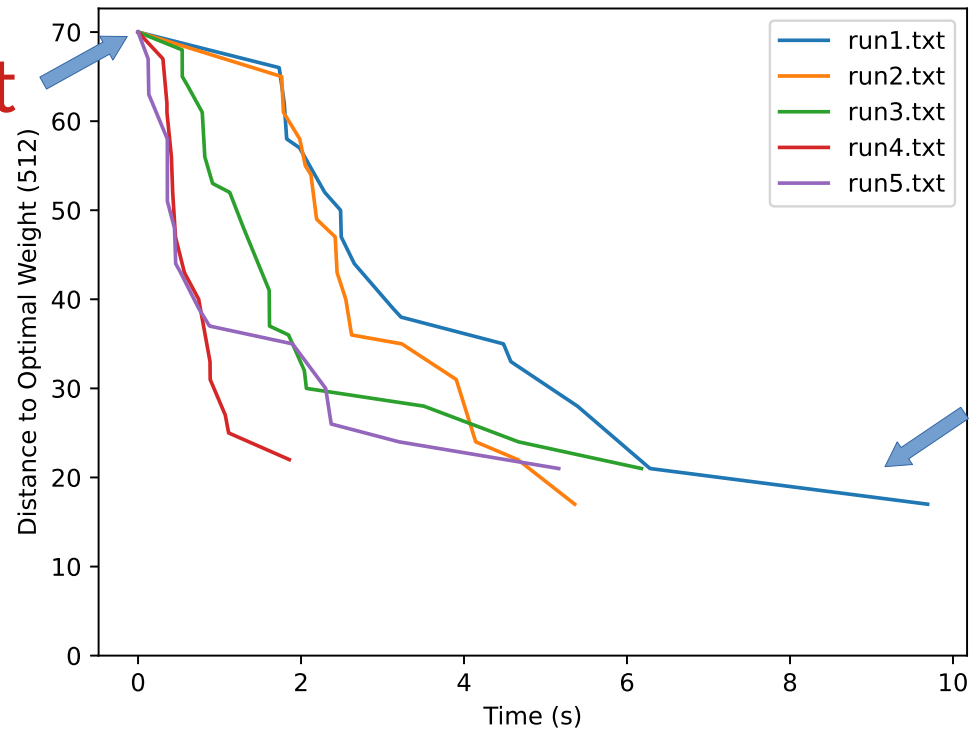
## Distance to Optimal vs. Time - SCP41



# ECL-SCP MIS + GA on (200 x 1000)

## Distance to Optimal vs. Time - SCP42

MIS Result

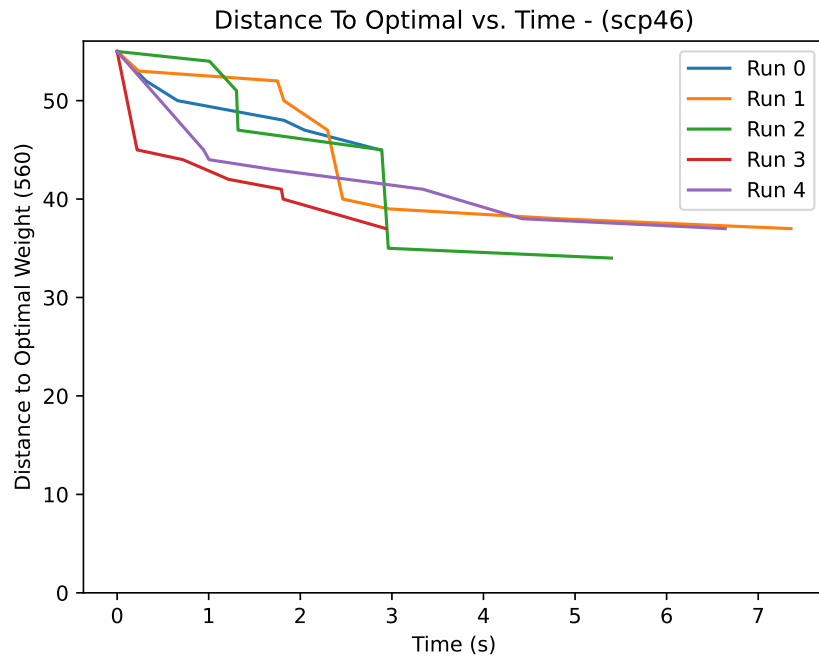


GA Results

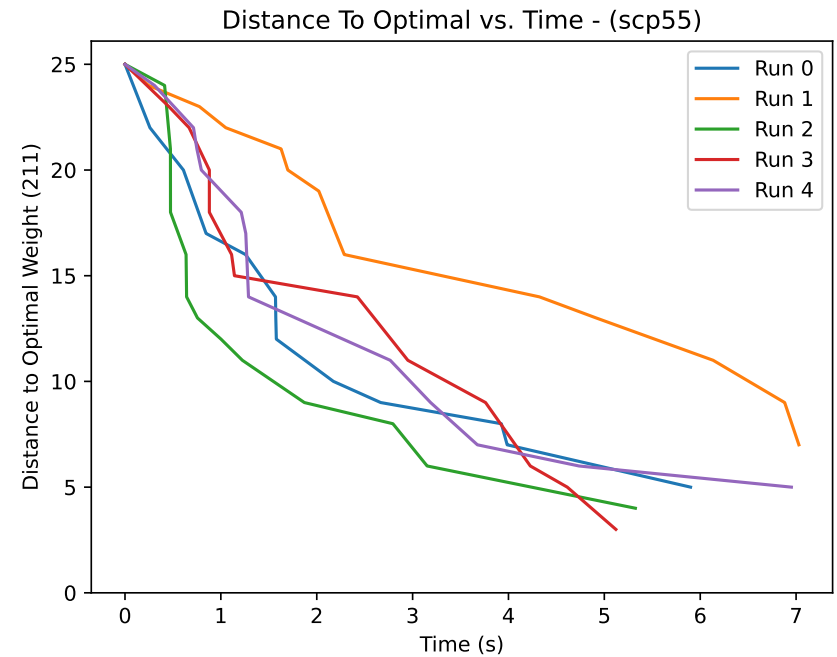


# ECL-SCP MIS + GA

(200 x 1000)

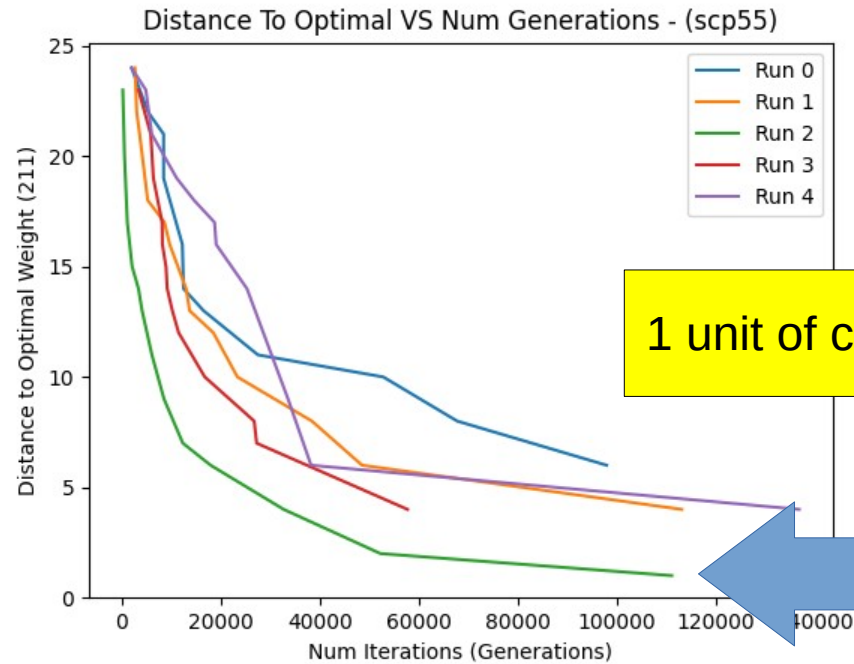


(200 x 2000)



# ECL-SCP MIS + GA

(200 x 2000)



1 unit of cost from optimal!!

# ECL-SCP **MIS** Rail Results

Problem Instance	Vertices	Sets	Optimal	Factor
<b>Rail507</b>	507	63009	172.14	1.24x
<b>Rail516</b>	516	47311	182.00	1.10x
<b>Rail582</b>	582	55515	209.71	1.19x
<b>Rail2536</b>	2536	1081841	688.39	1.28x
<b>Rail2586</b>	2586	92683	935.92	1.24x
<b>Rail4284</b>	4284	1092610	1054.05	1.31x
<b>Rail4872</b>	4872	968672	1509.63	1.26x

# Future Work

- Take into account how **valuable** a particular **allele** may be with respect to the population
  - “The **more-fit organisms** tend to have this set as part of their genes”
- Optimize solution before feeding into Phase II GA (e.g., **removing redundant sets**)
- Process each **connected-component** as a separate problem and subsequently combine



# Future Work

- Further comparisons against 3rd-party codes
- Optimize implementations
  - Can establish **lower** and **upper** bounds
- Tune GA configurations
- Looking to apply to problems with more than **5,000 vertices** and **1,000,000 sets**!

# References

- <https://people.brunel.ac.uk/~mastjjb/jeb/orlib/scpinfo.html>
- Cormen, T. H. (2009). Introduction to algorithms / Thomas H. Cormen [and others] (Third edition.). MIT Press.
- Bunday, B. D. (1984). Basic linear programming / Brian D. Bunday. Edward Arnold.
- Beasley, J. E. (1987). An algorithm for set covering problem. European Journal of Operational Research, 31(1), 85–93. [https://doi.org/10.1016/0377-2217\(87\)90141-X](https://doi.org/10.1016/0377-2217(87)90141-X)
- Balas, E., & Ho, A. (1980). Set Covering Algorithms Using Cutting Planes, Heuristics, and Subgradient Optimization - a Computational Study. MATHEMATICAL PROGRAMMING STUDY, 12(APR), 37–60.
- <https://www.ibm.com/docs/en/icos/22.1.2?topic=formats-working-mps-files>
- <https://www.gurobi.com/resources/linear-optimization-explained/>
- <https://github.com/NVIDIA/cuopt>
- <https://github.com/coin-or/CyLP>
- <https://github.com/ERGO-Code/HiGHS>
- <https://github.com/coin-or/Cbc>
- <https://github.com/scipopt/scip>

# Summary

- Brute-force approaches become untenable even with the smallest of inputs (40 vertices x 40 sets)
- We present **ECL-SCP**, a **parallel GPU** Maximal Independent Set + Genetic Algorithm SCP implementation
- Within **1 unit of cost** of the optimal within **seconds** and **match 97.7%** chosen sets with solver output!



[andres.gonzalez@txstate.edu](mailto:andres.gonzalez@txstate.edu)

