



UNIVERSITY OF
ILLINOIS
URBANA-CHAMPAIGN

Improvements to CUDA LAP Solver

IE 533 S22

Samiran Kawtikwar (samiran2)

Wes Crevens (wbc3)

Nick Yang (yyang192)

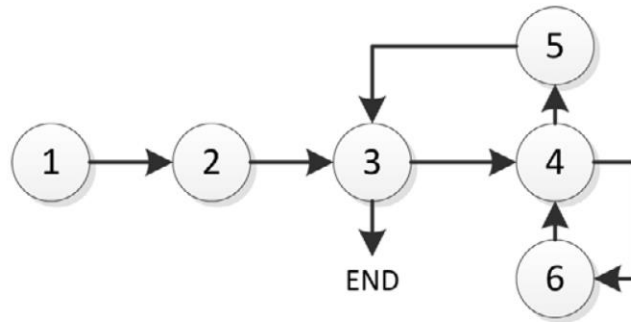
- Linear Assignment Problem
- Hungarian Algorithm for LAP
- JVC Algorithm for LAP
- Improvements and Results
- Reproducible Research and Project Infrastructure

- The linear assignment problem is a fundamental problem in combinatorial optimization and has vast applications, e.g., transportation systems, personal assignments, and graph association.
- known as the perfect matching problem in bipartite graphs.
 - Given n resources, n tasks and a cost matrix
 - Each resource is assigned to one task and each task is assigned to one resources
 - minimize total assignment cost

Hungarian Algorithm for the LAP



- Developed and Published by, Harold Kuhn (1955)
- Original Computational Complexity $O(N^4)$
- Has an alternating tree variant of $O(N^3)$
- 6 step iterative algorithm:



Flow through several steps of the algorithm

- Step 1: Min Reduction
- Step 2: Scan zeros (Star Zeros)
- Step 3: Column Cover
- Step 4: prime non covered zero/s
- Step 5: Construct Alternating prime and star zero path (Augment)
- Step 6: find the lowest value, subtracting uncrossed rows and adding crossed columns (Dual update)

Hungarian Algorithm Example



14	5	8	7
2	12	6	5
7	8	3	9
2	4	6	10



9	0	3	2
0	10	4	3
4	5	0	6
0	2	4	8



9	0	3	0
0	10	4	1
4	5	0	4
0	2	4	6

Row reduction and
column reduction



10	0	4	0
0	9	4	0
4	4	0	3
0	1	4	5

- Assignments

- $x_{33} = 1$
- $x_{41} = 1$
- $x_{24} = 1$
- $x_{12} = 1$

$$\text{Cost} = c_{12} + c_{24} + c_{33} + c_{41} = 15$$

- Published by R. Jonker and A. Volgenant (December 1987)
- While Hungarian algorithm finds any feasible augmenting path, Jonker, Volgenant and Castanon (JVC) find the shortest augmenting paths

Primal

$$\begin{aligned} \min \quad & \sum_i \sum_j x_{ij} c_{ij} \\ \text{s.t.} \quad & \sum_i x_{ij} = 1 \\ & \sum_j x_{ij} = 1 \\ & x_{ij} \geq 0, \forall (i, j) \in E \end{aligned}$$

Dual

$$\begin{aligned} \max \quad & \left\{ \sum_i u_i + \sum_j v_j \right\} \\ \text{s.t.} \quad & c_{ij} - u_i - v_j \geq 0 \end{aligned}$$

- Step 1 : *Initialization*
- Step 2: *Termination*, if all rows are assigned.
- Step 3: *Augmentation*
Construct the auxiliary network and determine from an unassigned row i to an unassigned column j an alternating path of minimal total reduced cost and use it to augment the solution.
- Step 4: *Adjust the dual solution*
to restore complementary slackness. Go to step 2

- Initialization
 - Column reduction
 - each column is assigned to minimal row element
 - some rows may not be assigned
 - Reduction transfer from unassigned to assigned rows

Column Reduction:

$$C = \begin{bmatrix} 14 & 5 & 8 & 7 \\ 2 & 12 & 6 & 5 \\ 7 & 8 & 3 & 9 \\ 2 & 4 & 6 & 10 \end{bmatrix}$$

$$\Rightarrow \underline{v}^T = [2 \quad 4 \quad 3 \quad 5]; \underline{u} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}; x_{21} = x_{42} = x_{33} = 1$$

Dual cost = 14

Row 1 and column 4 unassigned.

Reduction Transfer:

Row 2: $\mu = \min(8, 3, 0)$, $v_1 = 2$; $u_1 = 0$

Row 3: $\mu = \min(5, 4, 4)$, $v_3 = -1$; $u_3 = 4$

Row 4: $\mu = \min(0, 3, 5)$, $v_2 = 4$; $u_4 = 0$

$$\Rightarrow \underline{v}^T = [2 \quad 4 \quad -1 \quad 5]; \underline{u} = \begin{bmatrix} 0 \\ 0 \\ 4 \\ 0 \end{bmatrix}$$

Dual cost = 14

- Augmenting reduction of unassigned rows

```

LIST = {all unassigned rows}
∀i ∈ LIST do
  repeat
    k1 = min{cij - vj: j = 1, ..., n}
    select j1 with cij1 - vj1 = k1
    k2 = min{cij - vj: j = 1, ..., n; j ≠ j1}
    select j2 with w = cij2 - vj2 = k2; j2 ≠ j1
    ui = k2
    if k1 < k2 then
      vj1 = vj1 - (k2 - k1)
    else if j1 is assigned, then
      j1 = j2
      r = yj1;
      if r > 0 then
        xr = 0; xi = j1; yj1 = i; i = r
      until k1 = k2 or r = 0
  end do
  
```

- Complexity of first two initialization procedures is $O(n^2)$, and it can be shown that the augmenting reduction procedure has complexity $O(Rn^2)$, with R the range of cost coefficients

Augmentation Reduction:

Row 1: $k_1 = \min(12, 1, 9, 2) = 1; k_2 = 2$

$v_2 = 3; u_1 = 2$

$x_{12} = 1$

$x_{42} = 0$

$$\Rightarrow \underline{v}^T = [2 \quad 3 \quad -1 \quad 5]; \underline{u} = \begin{bmatrix} 2 \\ 0 \\ 4 \\ 0 \end{bmatrix}$$

Dual cost = 15

Row 4: $k_1 = \min(0, 1, 7, 5) = 0; k_2 = 1$

$v_1 = 1; u_4 = 1$

$x_{41} = 1$

$x_{21} = 0$

$$\Rightarrow \underline{v}^T = [1 \quad 3 \quad -1 \quad 5]; \underline{u} = \begin{bmatrix} 2 \\ 0 \\ 4 \\ 1 \end{bmatrix}$$

Row 2: $k_1 = \min(1, 9, 7, 0) = 0; k_2 = 1$

$v_4 = 4; u_2 = 1$

$x_{24} = 1$

$r = 0$

$$\Rightarrow \underline{v}^T = [0 \quad 3 \quad -1 \quad 4]; \underline{u} = \begin{bmatrix} 2 \\ 1 \\ 4 \\ 2 \end{bmatrix}$$

Primal and dual feasible. Done!

- Augmentation
 - Modified version of Dijkstra's shortest augmenting path method
 - Complexity for augmentation phase is $O(n^3)$, so this holds for the entire JVC algorithm
- Update of the dual solution
- After augmentation of the partial assignment, the values of dual variables must be updated to restore complementary slackness, that is,
 - $c_{ik} - u_i - v_k = 0$, if $x_i = k$ for assigned column k , $i = 1, \dots, n$ and
 - $c_{ik} - u_i - v_k \geq 0$

- Random costs generated with default random generator from stdlib.
- Problem size (N) and Range considered as input parameters.
- Cost matrix generated is uniformly distributed in $[0, \text{Range} \times N)$.
- Hardware Information:
 - Host – quad-core intel i7 4790 with 16GB RAM
 - Device – NVIDIA GeForce RTX 2080 Ti with 11GB vRAM (cc 7.5)
 - NVCC CUDA version 11.5
 - C++ V11, GCC Version 7.5
- For small/fast solving problems we run for 5 times and average.

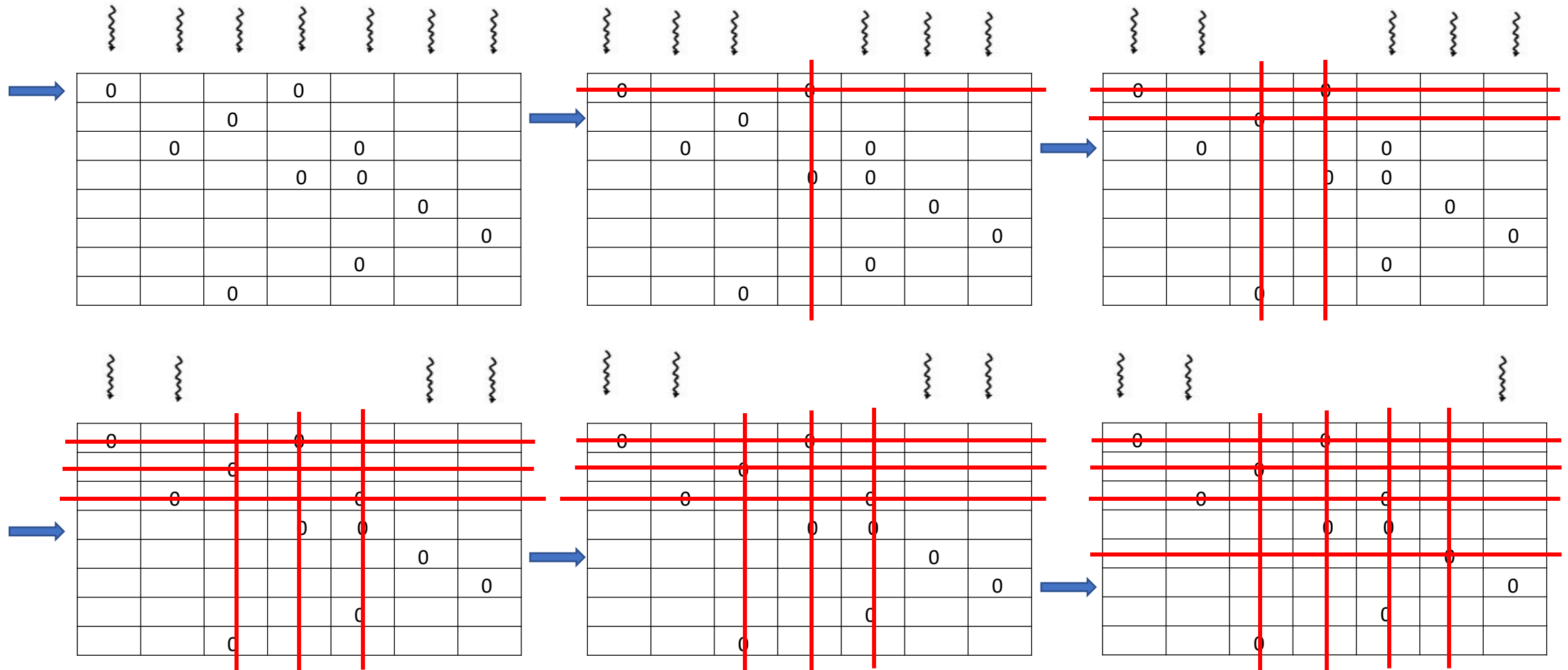
- We did an extensive survey of available implementations of linear assignment problem solvers.
- These solvers are mainly based on following techniques.
 - Max flow - (Hungarian Algorithm)
 - Shortest Path - (JVC, Auction)
 - Linear Programming - (Specialized versions of Simplex)
- Since, our focus was mainly on GPUs, we looked at both sequential and parallel implementations of these algorithms.
 - Hungarian Algorithm - (Date 16*, Lopes 19*)
 - JVC - (JVC 70, Guthe 21*)
 - Simplex - (Gurobi)
 - Auction - (Bertsekas 81, Pauc 16*)

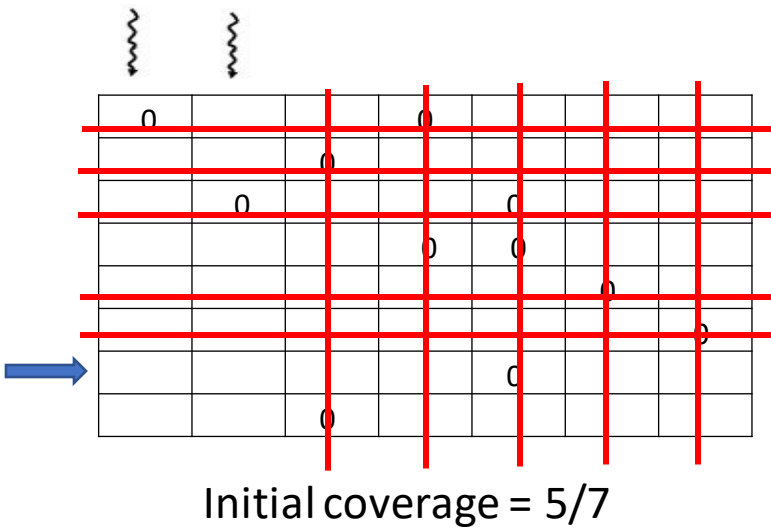
* cuda implementations

Improvement Attempt - 1



- Existing GPU Hungarian algorithms use race conditions for initial cover.





- Can we do better?
Definitely!
- Use Vogel's regret approach (as used in traditional transportation problem)
- Tradeoffs:
 - Higher initial coverage
 - Compromise in speed
 - Inherently Sequential procedure
- A middle ground is to check for column regrets in case of race conditions and chose column with highest regret.

Improvement Attempt – 1 (continued)



- Vogel's regret approach

Penalty

0	79	75	0	29	11	44	0
12	44	0	69	88	95	99	12
82	0	48	18	0	41	57	0
68	69	17	0	0	45	60	17
72	18	43	60	9	0	27	9
79	91	71	28	74	19	0	19
43	81	96	48	0	58	24	24
97	25	0	37	53	65	55	25
12	18	0	18	0	11	27	

0	79	75	0	29	11	44	0
12	44	0	69	88	95	99	12
82	0	48	18	0	41	57	0
68	69	17	0	0	45	60	17
72	18	43	60	9	0	27	9
79	91	71	28	74	19	0	19
43	81	96	48	0	58	24	48
97	25	0	37	53	65	55	25
12	18	0	18	0	11	27	

0	79	75	0	29	11	44	0
12	44	0	69	88	95	99	12
82	0	48	18	0	41	57	18
68	69	17	0	0	45	60	17
72	18	43	60	9	0	27	18
79	91	71	28	74	19	0	19
43	81	96	48	0	58	24	48
97	25	0	37	53	65	55	25
12	18	0	18	0	11	27	

0	79	75	0	29	11	44	0
12	44	0	69	88	95	99	12
82	0	48	18	0	41	57	18
68	69	17	0	0	45	60	45
72	18	43	60	9	0	27	18
79	91	71	28	74	19	0	19
43	81	96	48	0	58	24	48
97	25	0	37	53	65	55	25
12	18	0	18	0	11	27	

0	79	75	0	29	11	44	11
12	44	0	69	88	95	99	12
82	0	48	18	0	41	57	41
68	69	17	0	0	45	60	45
72	18	43	60	9	0	27	18
79	91	71	28	74	19	0	19
43	81	96	48	0	58	24	48
97	25	0	37	53	65	55	25
12	18	0	18	0	11	27	

0	79	75	0	29	11	44	11
12	44	0	69	88	95	99	83
82	0	48	18	0	41	57	41
68	69	17	0	0	45	60	45
72	18	43	60	9	0	27	72
79	91	71	28	74	19	0	19
43	81	96	48	0	58	24	48
97	25	0	37	53	65	55	25
12	18	0	18	0	11	27	

Improvement Attempt – 1 (continued)



- Vogel's regret approach

Penalty

0	79	75	0	29	11	44	--
12	44	0	69	88	95	99	--
82	0	43	13	0	41	57	41
68	69	17	0	0	45	60	45
72	18	43	69	0	0	27	72
70	91	71	23	74	19	0	19
43	81	95	43	0	53	24	48
97	25	0	37	53	65	55	25
12	18	0	18	0	11	27	

0	79	75	0	29	11	44	--
12	44	0	69	88	95	99	--
82	0	43	13	0	41	57	41
68	69	17	0	0	45	60	45
72	18	43	69	0	0	27	72
70	91	71	23	74	19	0	19
43	81	95	43	0	53	24	48
97	25	0	37	53	65	55	25
12	18	0	18	0	11	27	

Initial coverage = 7/7







0			0				
		0					
	0			0			
				0	0		
						0	
							0

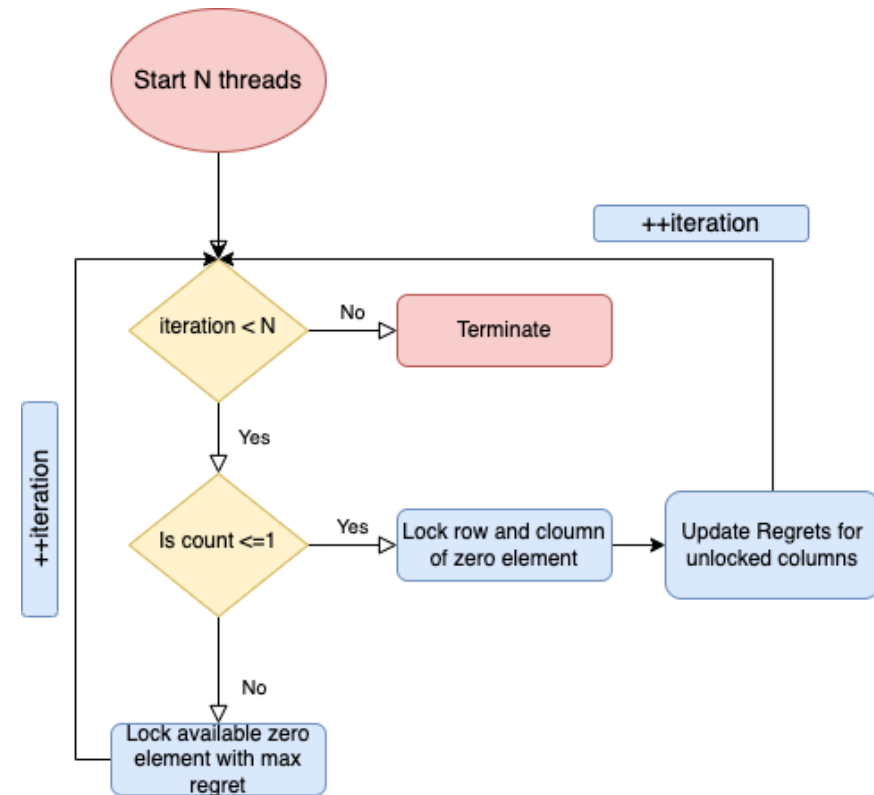
Old coverage = 5/7

Improvement Attempt – 1 (continued)



- Hybrid approach – (Vogel's regret approach is a bit slow)
- 2 phase approach
 - Phase 1 - Dry run
 - Phase 2 – Lock with regret

								
							Count	
	0	79	75	0	29	11	44	2
	12	44	0	69	88	95	99	1
	82	0	48	18	0	41	57	2
	68	69	17	0	0	45	60	1
	72	18	43	60	9	0	27	1
	79	91	71	28	74	19	0	1
	43	81	96	48	0	58	42	1
	97	24	0	37	53	65	55	1
Regret	12	18	0	18	0	11	27	



Improvement Attempt – 1 (coverage strength)



Range 0.01

Size	Baseline	2 Phase	Vogel
32	32	32	32
64	64	64	64
128	128	128	128
256	256	256	256
512	508	510	512
1024	1020	1019	1024
2048	2037	2036	2048
4096	4069	4074	4096
8192	8140	8161	8192

Range 0.1

Size	Baseline	2 Phase	Vogel
32	31	31	32
64	58	62	64
128	119	124	128
256	240	246	256
512	475	493	511
1024	956	980	1023
2048	1909	1950	2047
4096	3821	3892	4094
8192	7608	7824	

Range 1

Size	Baseline	2 Phase	Vogel
32	24	26	28
64	47	55	57
128	99	112	113
256	205	213	225
512	402	437	448
1024	801	861	892
2048	1588	1705	1754
4096	3231	3446	3566
8192	6492	6888	

Range 10

Size	Baseline	2 Phase	Vogel
32	26	27	27
64	49	53	53
128	103	110	110
256	197	208	209
512	405	421	422
1024	806	849	851
2048	1593	1679	1684
4096	3144	3290	3297
8192	6348	6637	6661

Range 100

Size	Baseline	2 Phase	Vogel
32	26	27	27
64	49	52	52
128	105	109	109
256	196	205	205
512	403	421	421
1024	797	846	846
2048	1588	1656	1656
4096	3194	3325	3326
8192	6345	6591	6592

Range 1000

Size	Baseline	2 Phase	Vogel
32	26	27	27
64	49	52	52
128	105	109	109
256	202	207	207
512	398	417	417
1024	785	809	809
2048	1591	1661	1661
4096	3209	3312	3312
8192	6351	6612	6613

Improvement Attempt – 1 (Total Runtimes)



- Though the regret-based strategies provide promising coverage strength, it doesn't reflect as sharply in run times.
- For, $\text{range} \leq 1$ these strategies prove to be overhead.
- 2 phase strategy is “just fine” in some cases without much overhead.
- On checking counts per step, the small change in that hints that the smart initialization strategies take away the parallelism from augmentations.

Size	Range	Baseline	2 phase	Vogel
4096	0.01	25.84	23.25	291.25
4096	0.10	18.70	18.14	269.58
4096	1.00	220.47	231.73	496.35
4096	10.00	1207.61	1145.48	1992.75
4096	100.00	6524.84	6720.88	15482.30
4096	1000.00	32261.00	32121.00	38744.00

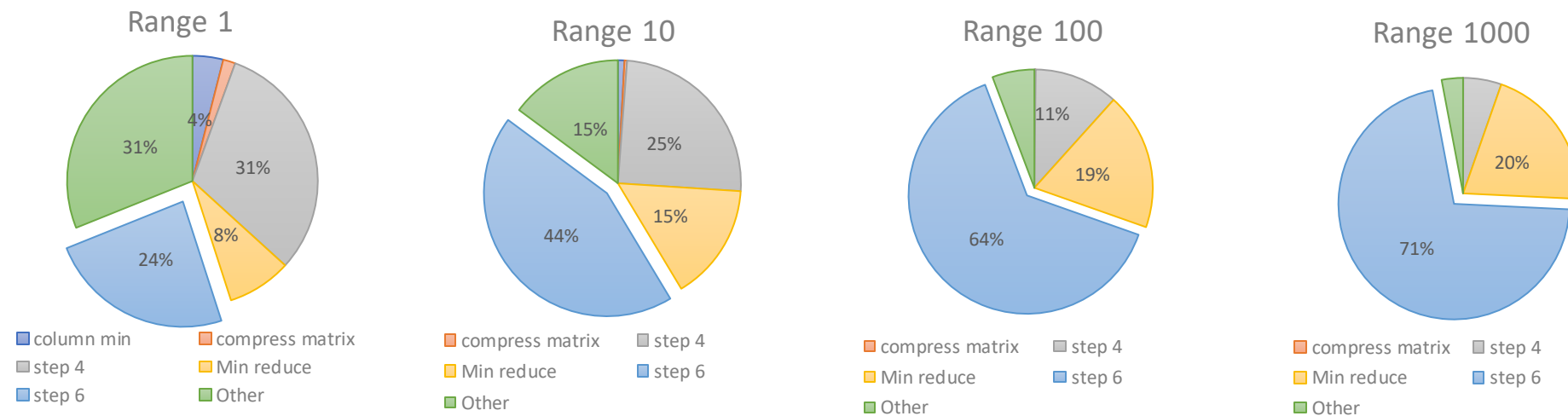
Total Times (in ms) for different initialization strategies

- Used preprocessing ideas from Munapo et. All
- Claim to increase parallelism in the LAP cost structure
- Though the cost update is only valid when augmentation is done across shortest path.
- One augmentation provides multiple pivots, hence the term coined as parallelism.
- The idea is best suited for CPU implementation, since shortest paths required and can only do single pivot per iteration.

Improvement Attempt 3



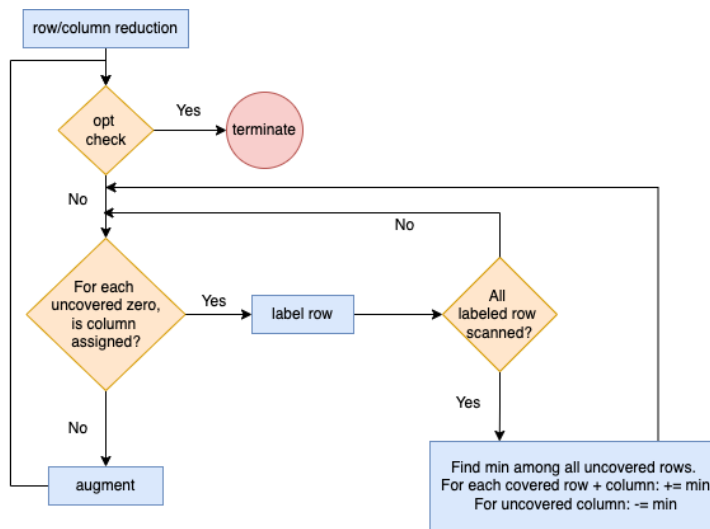
- CUDA improvements
- Used NVIDIA system profiler to identify bottlenecks in the code and used basic techniques like privatization, kernel fusion etc. to improve performance by upto 7.8x
- This improvement also brought light to most time-consuming steps in the implementation.



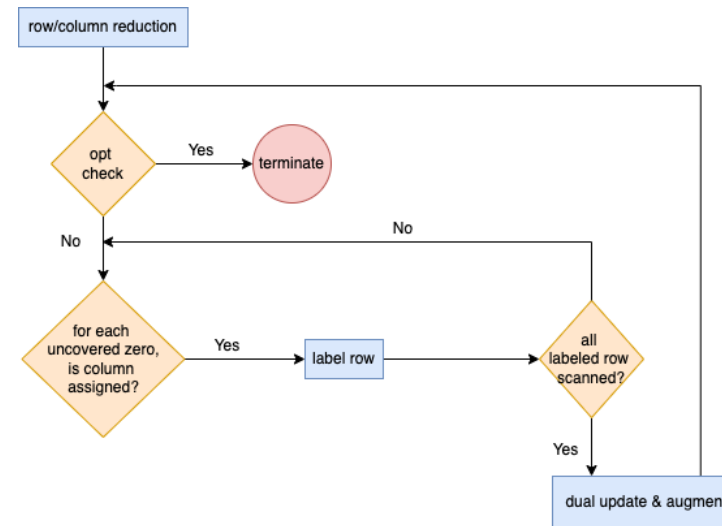
Improvement Attempt 4 - JVC vs Hungarian Algorithm



- Clearly step 6 is the most time-consuming step as cost range increases.
- This step is referred as dual update step and is the weakness of Hungarian Algorithm. (confirmed by many sources) ($O(N^2)/iteration$)
- JVC shortest path algorithm reduces this step to ($O(N)/iteration$), though with this update the augmentation restricts to finding shortest paths.



Flow of Hungarian Algorithm



Flow of JVC shortest path

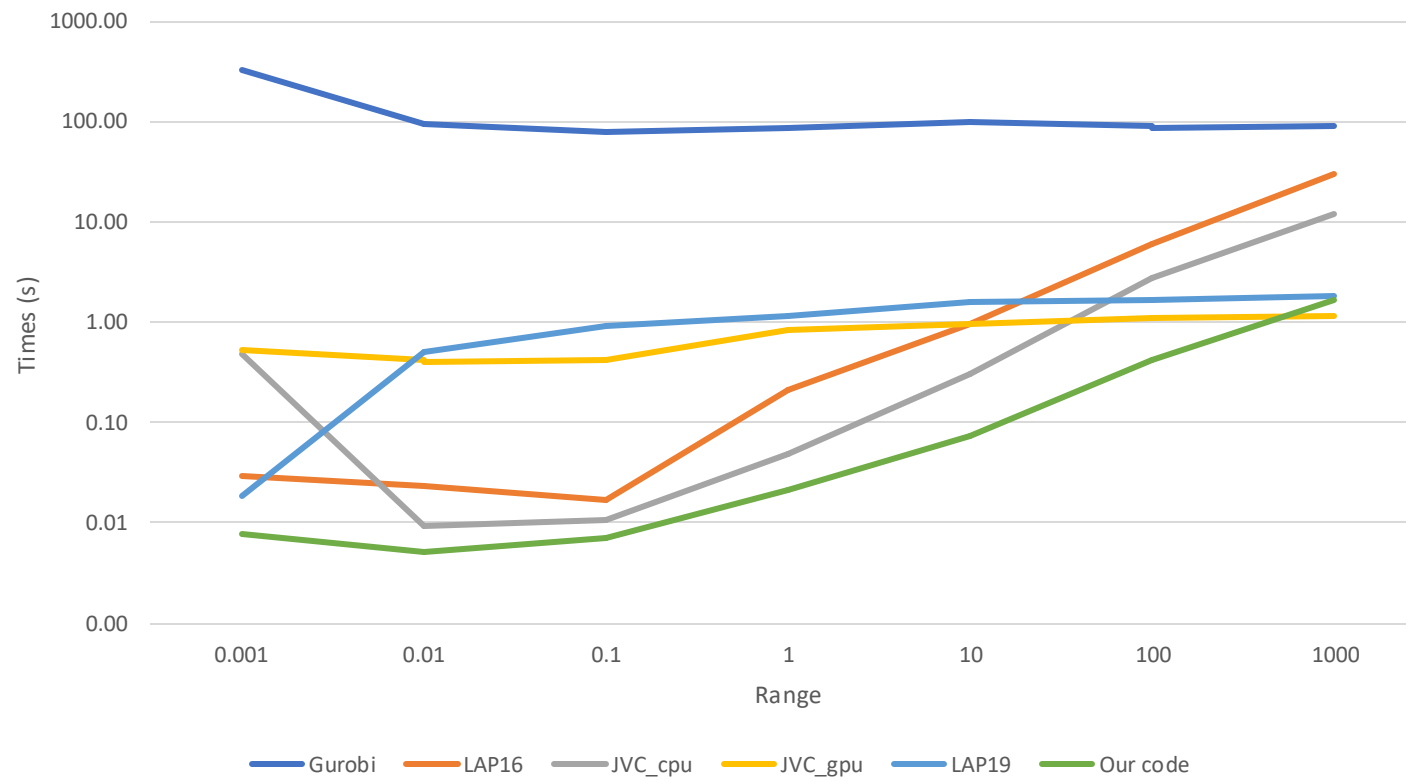
- We observe that both problems have very similar structure.
- JVC paper uses Augmentation transfer as preprocess for optimizing their run time. Though it is not necessary.
- Though a transformation from Hungarian to JVC exists, finding shortest augmenting paths on GPUs is difficult.
- Guthe et. Al in a recent paper (2021), implemented shortest augmenting paths over GPU, thought the performance is slower than CPU in many cases.*

*After ignoring epsilon scaling improvements

Improvement Attempt 4 - JVC vs Hungarian Algorithm

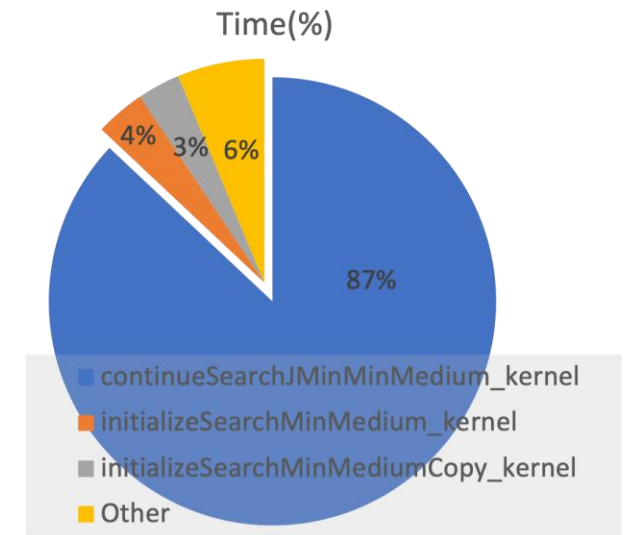


- Due to the parallelism offered, Hungarian algorithm performs better for low range matrices while JVC performs well for High range (Dense) cost inputs.



Runtime trend of different algorithms wrt range, N = 4096

- Shortest augmenting path finding dominates runtime in the GPU implementation of JVC.
- Hints towards, shortest path better to be done on CPU for JVC.
- This improvement is not fully implemented and explored.



- We have significantly improved state of the art implementation for linear assignment problem for low range matrices.

Name	64	128	256	512	1024	2048	4096	8192
Gurobi	8.44	32.65	151.95	706.99	3614.36	58170.10	324376.00	out of mem
LAP16	0.20	0.25	0.41	0.74	1.46	10.68	29.26	58.21
JVC_cpu	0.16	1.34	2.69	4.41	26.32	125.49	474.60	1842.94
JVC_gpu	174.00	164.00	177.00	190.00	201.00	279.00	521.00	1452.00
LAP19	0.37	0.44	0.56	1.03	2.20	3.64	18.79	43.11
Our code	0.41	0.57	0.62	3.49	4.81	3.41	7.89	16.64
speedup	0.39	0.45	0.66	0.21	0.30	1.07	2.38	2.59

We achieved Geo mean speedup of 2.62x for problem size 8192 wrt the best performing competitor algorithms!

Name	64	128	256	512	1024	2048	4096	8192
Gurobi	10.93	33.17	258.19	509.85	2466.77	11718.00	95954.10	out of mem
LAP16	0.51	0.25	0.42	3.23	4.75	8.54	22.94	56.57
LAP19	0.31	0.41	0.49	2.76	6.46	6.44	9.45	18.13
JVC_cpu	0.09	0.30	1.71	6.21	19.59	92.64	411.71	1617.43
JVC_gpu	223.00	134.00	134.00	140.00	164.00	242.00	503.00	1436.00
Our code	1.50	0.88	1.45	2.38	2.64	6.26	5.09	9.96
speedup	0.06	0.28	0.29	1.16	1.80	1.03	1.86	1.82

Thank you!

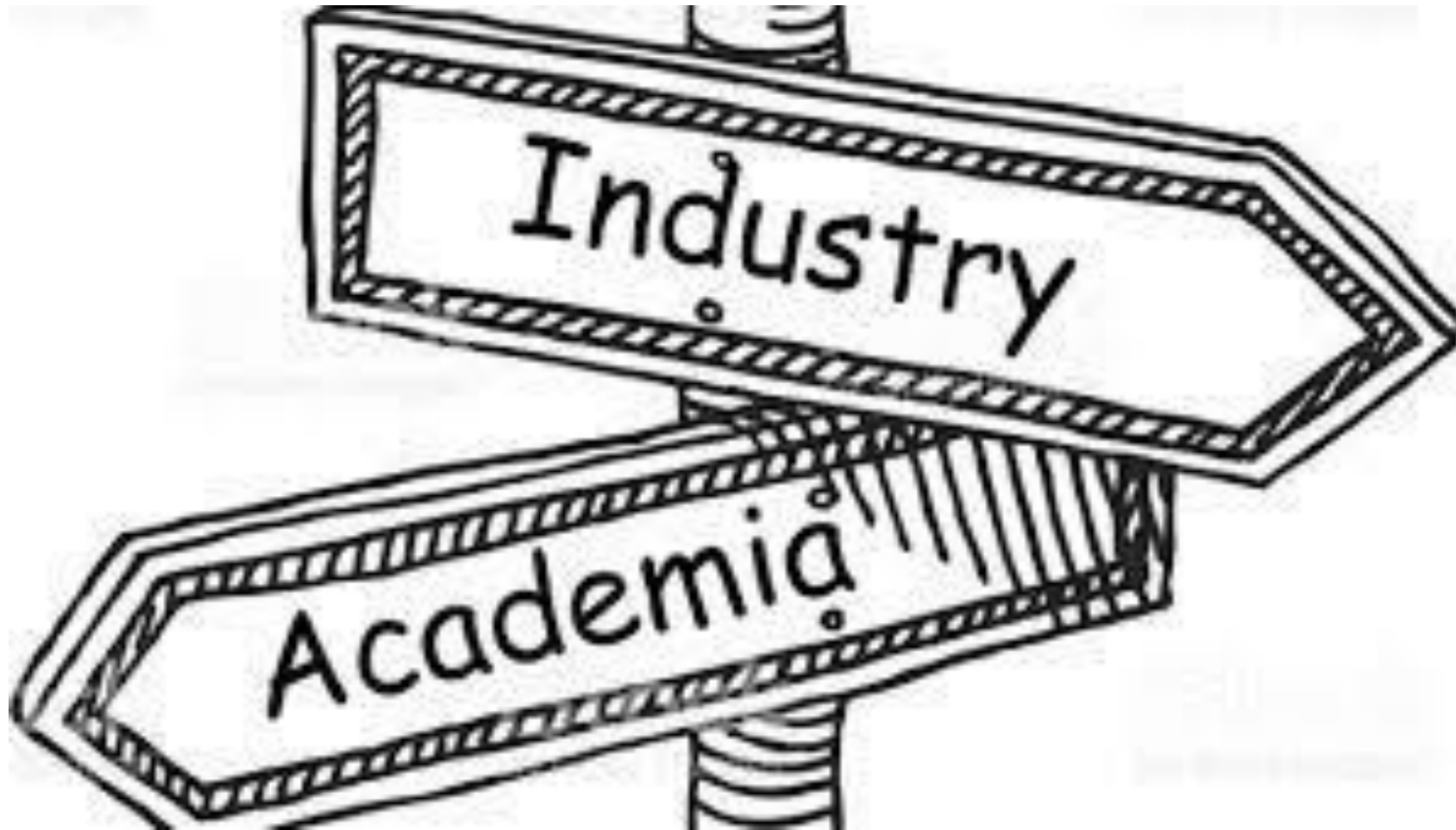
You can Relax now... we're into the good part.

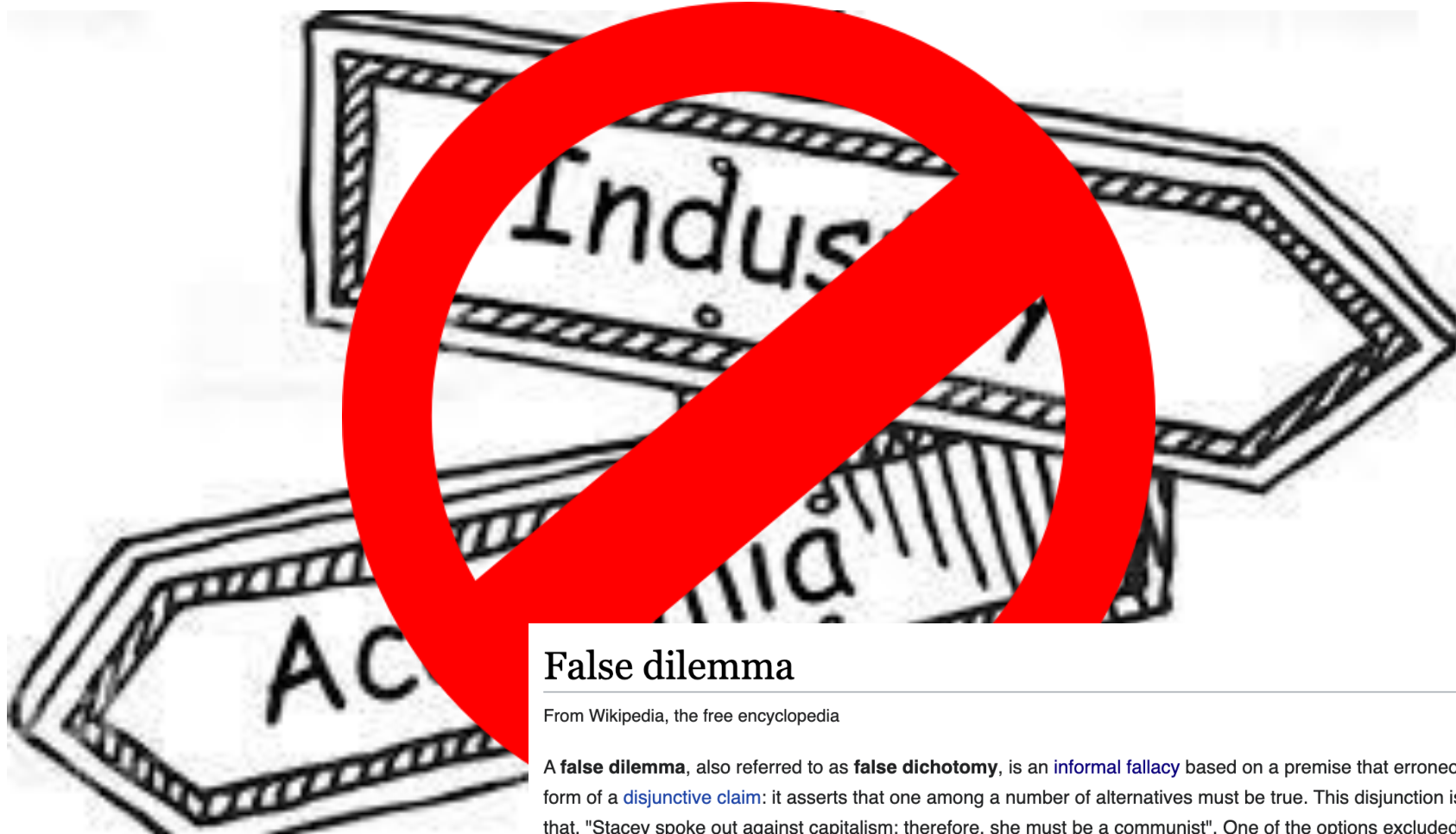


Note: These slides are not the content (thankfully).

The talking part is the content.

If you would like notes/bullet points/references later just let us know.

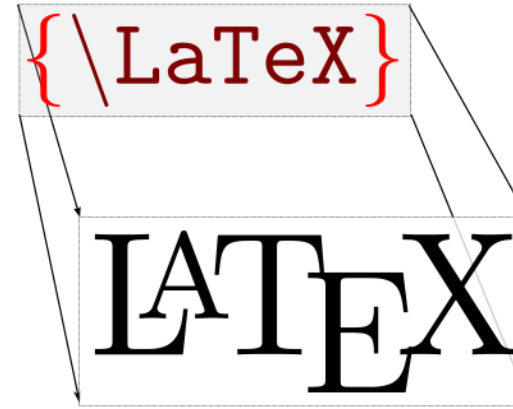




False dilemma

From Wikipedia, the free encyclopedia

A **false dilemma**, also referred to as **false dichotomy**, is an [informal fallacy](#) based on a premise that erroneously limits what options are available. The source form of a [disjunctive claim](#): it asserts that one among a number of alternatives must be true. This disjunction is problematic because it oversimplifies the choice that, "Stacey spoke out against capitalism; therefore, she must be a communist". One of the options excluded is that Stacey may be neither communist nor capitalist, as [contradictories](#), of which one is necessarily true. Various inferential schemes are associated with false dilemmas, for example, the [constructive dilemma](#), the terms of [deductive arguments](#). But they can also occur as [defeasible arguments](#). Our liability to commit false dilemmas may be due to the tendency to simplify language. This may also be connected to the tendency to insist on clear distinction while denying the vagueness of many common expressions.



GNU Make



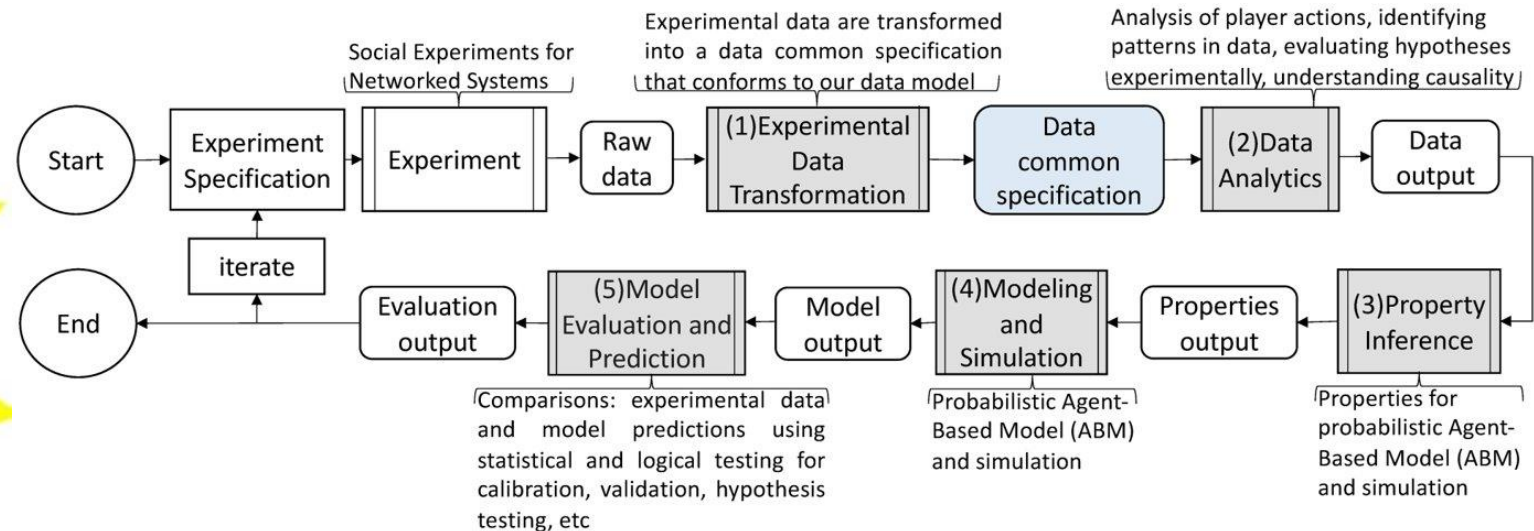
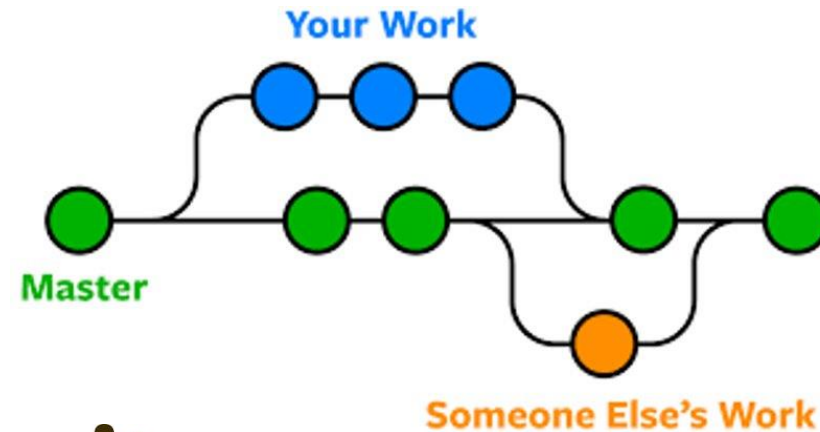
...



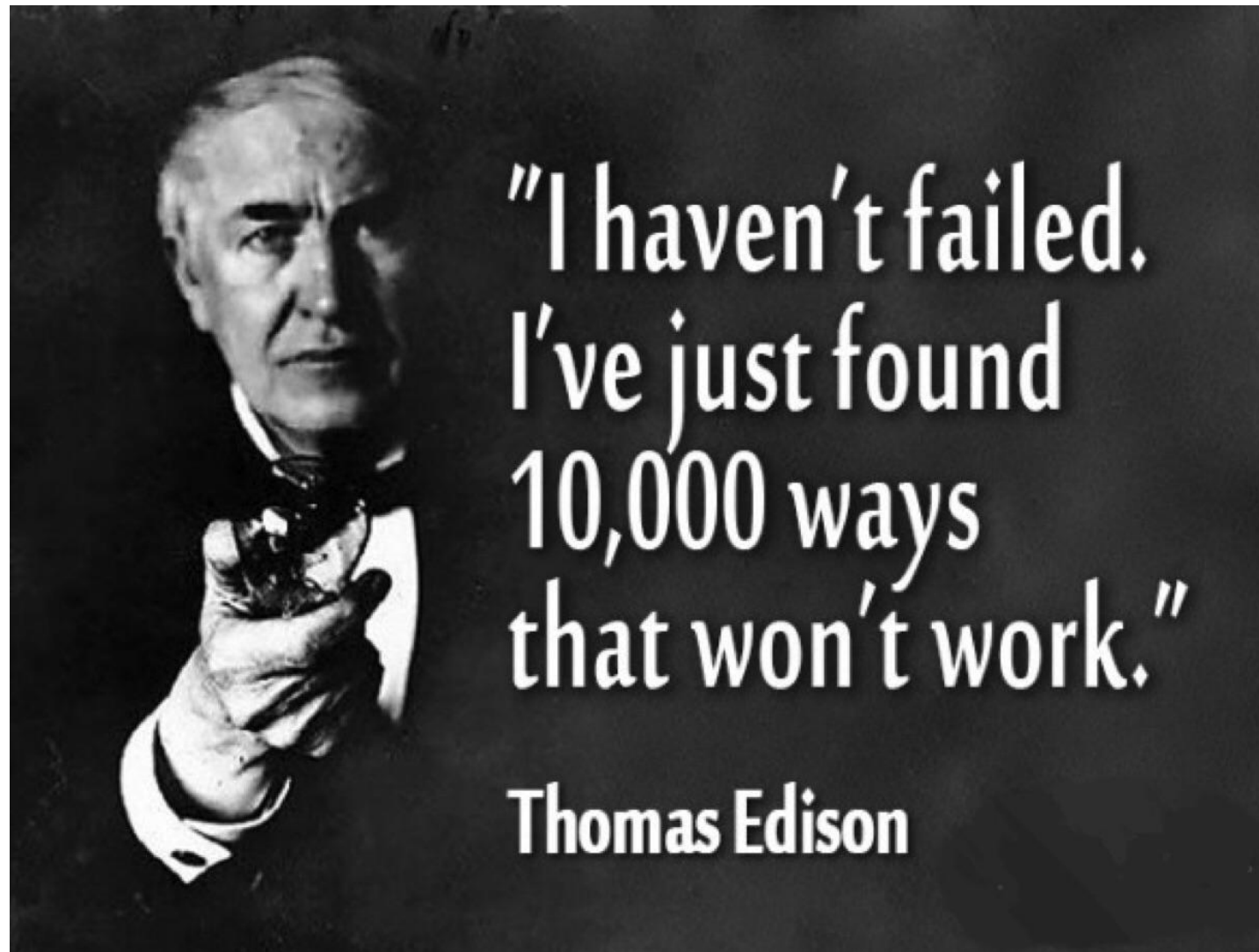
Reproducible Research and Project Infrastructure



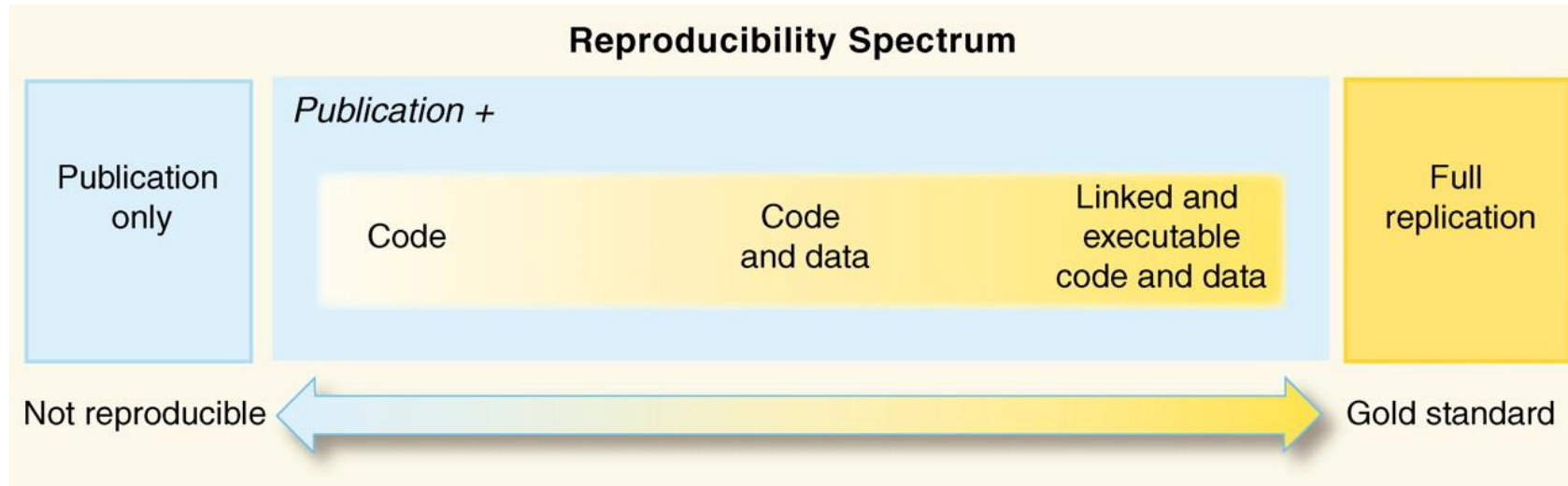
RECORD







The Red Shiny Button



SCIENCE • 2 Dec 2011 • Vol 334, Issue 6060 • pp. 1226-1227 • [DOI: 10.1126/science.1213847](https://doi.org/10.1126/science.1213847)