KAHNG AND ROBINS ROUTING

GROUP MEMBER:
DINGXING JIN
ZHONGYOU CHEN

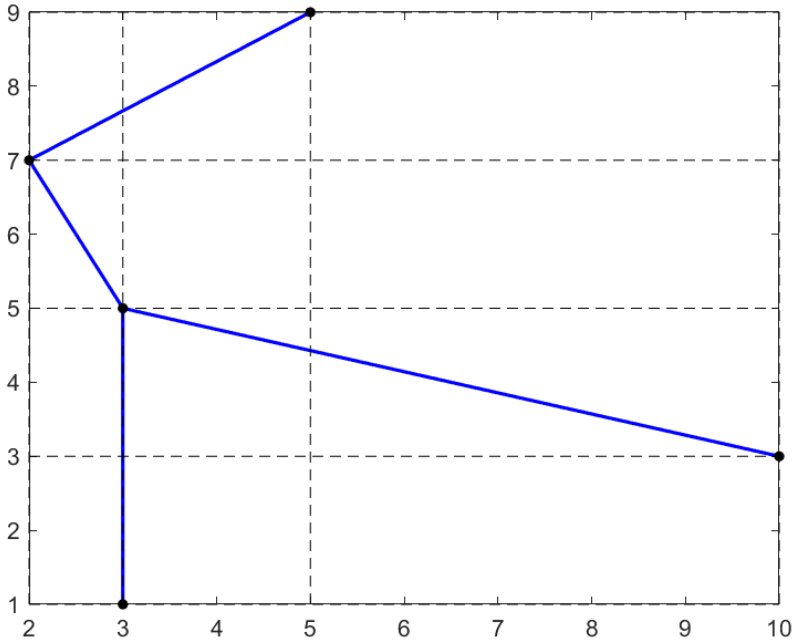
Implement

- Basic 1-Steiner algorithm as well as the Random Variant
- Compare the quality among 3 solutions(basic, random, and Prim)
- Compare Prim(Heap + Adjacency List) and Kruskal MST algorithm
- Partition Variant for the KR routing based on Prim Algorithm
- Use Python to generate extra benchmarks

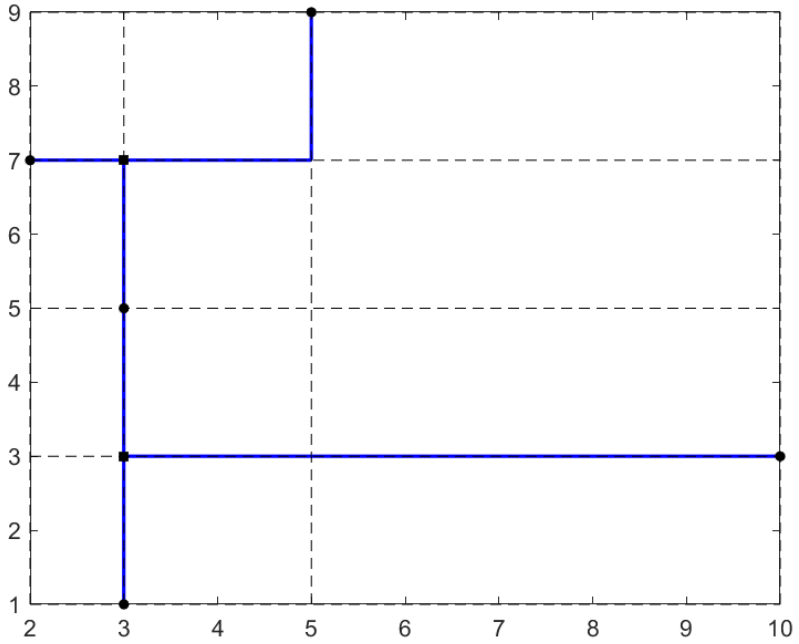
Basic 1-Steiner algorithm result

Grid Size 10, Number of Point 5

Initial MST



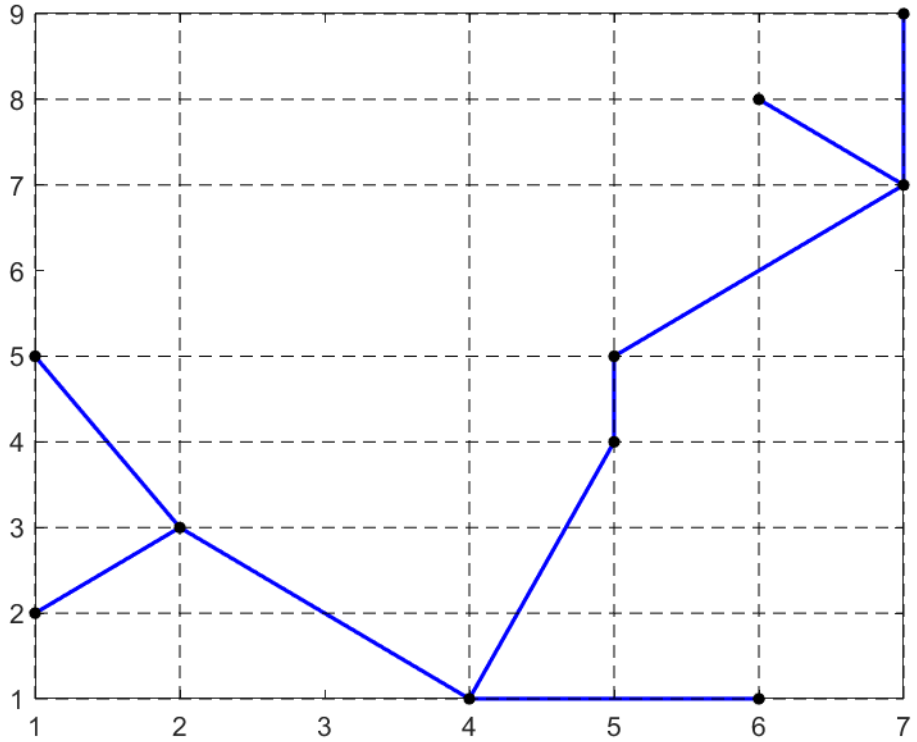
Final MRST



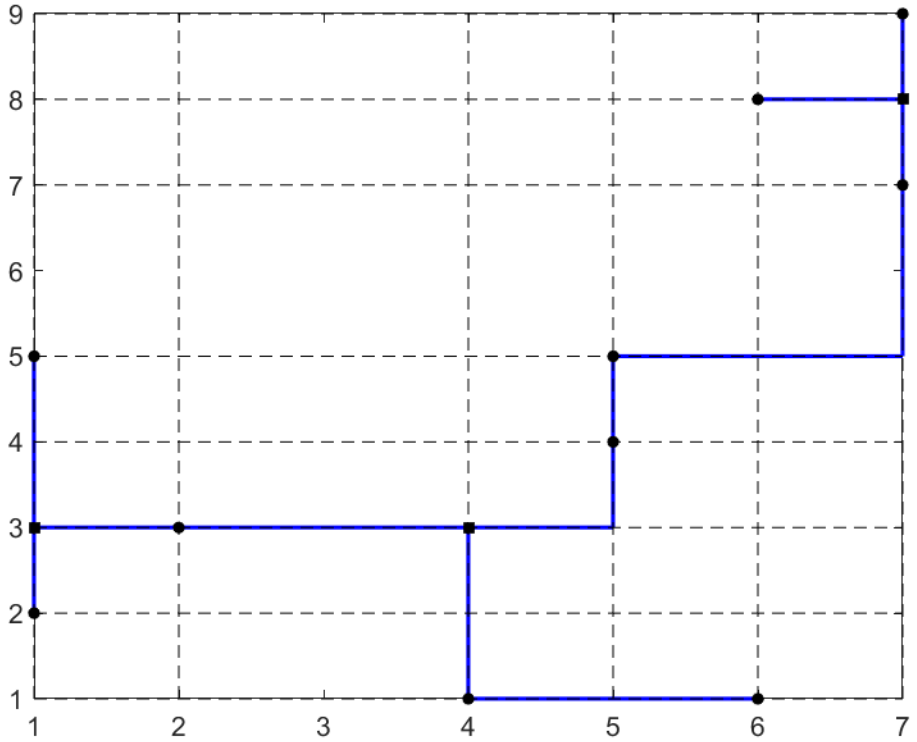
Basic 1-Steiner algorithm result

Grid Size 10, Number of Point 10

Initial MST



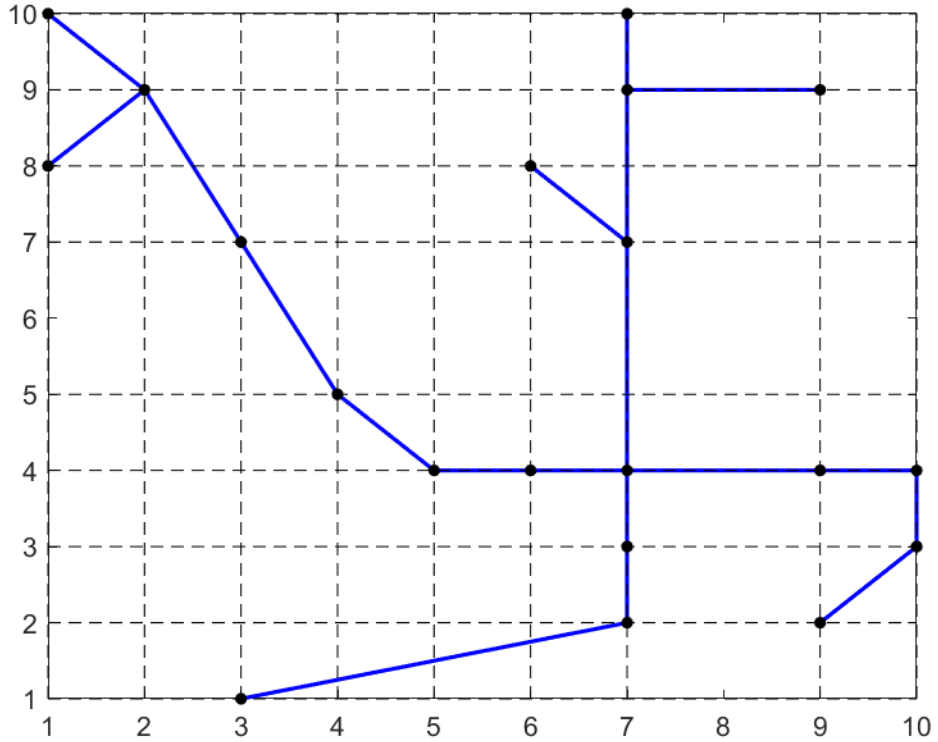
Final MRST



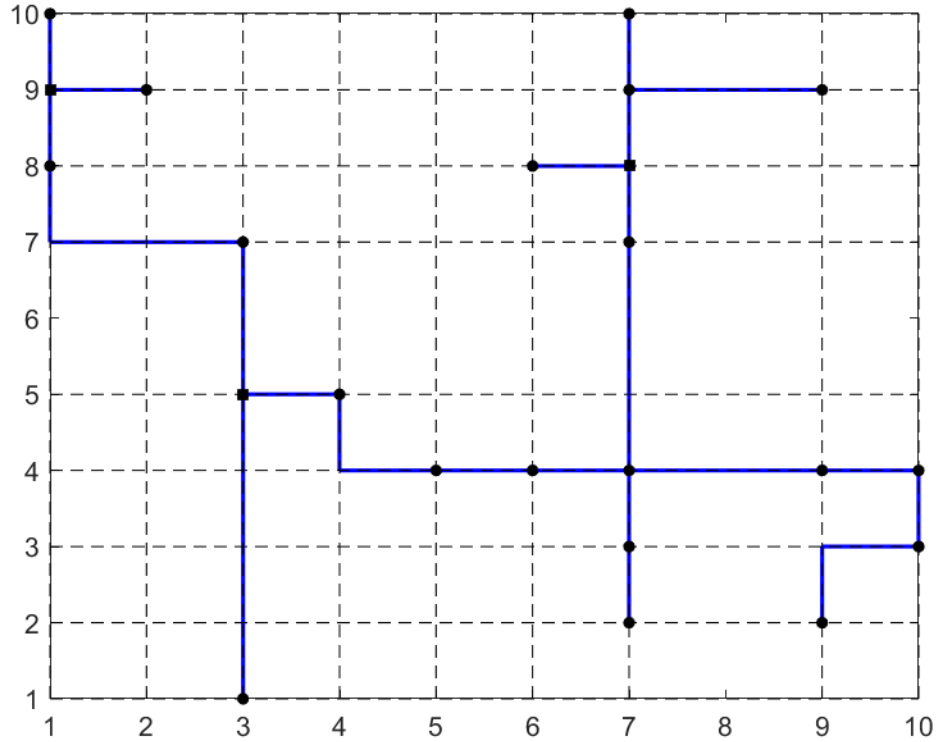
Basic 1-Steiner algorithm result

Grid Size 10, Number of Point 20

Initial MST



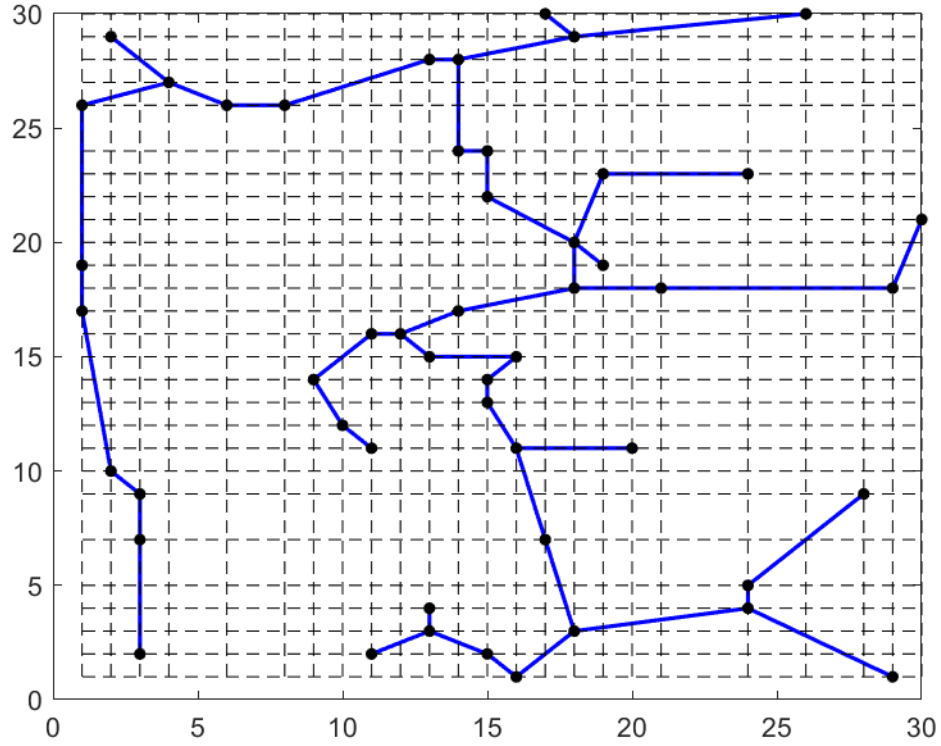
Final MRST



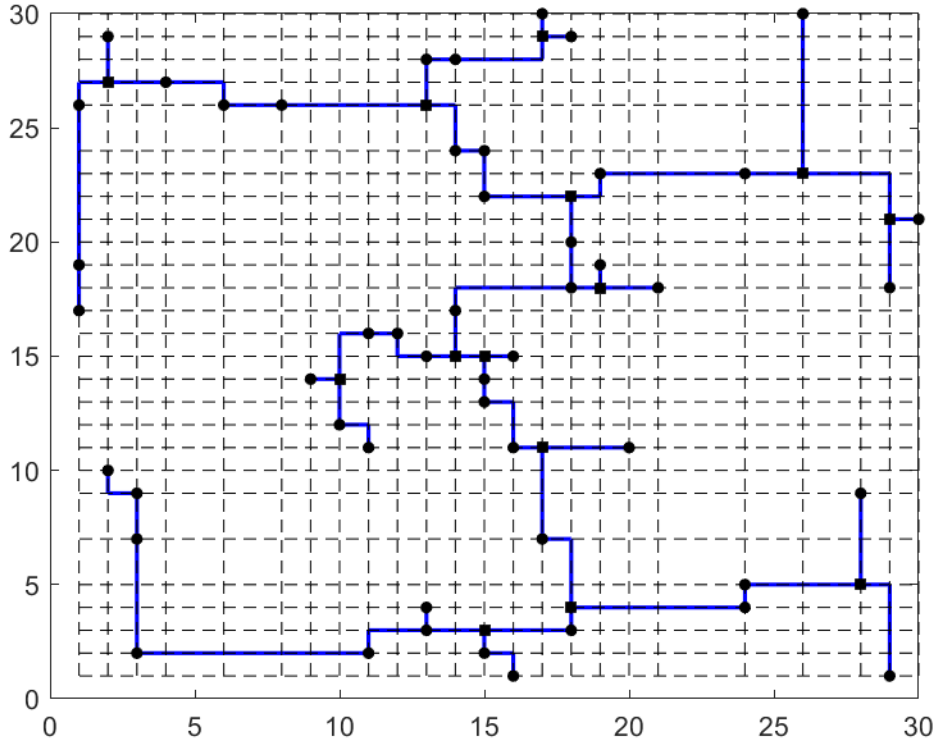
Basic 1-Steiner algorithm result

Grid Size 30, Number of Point 50

Initial MST



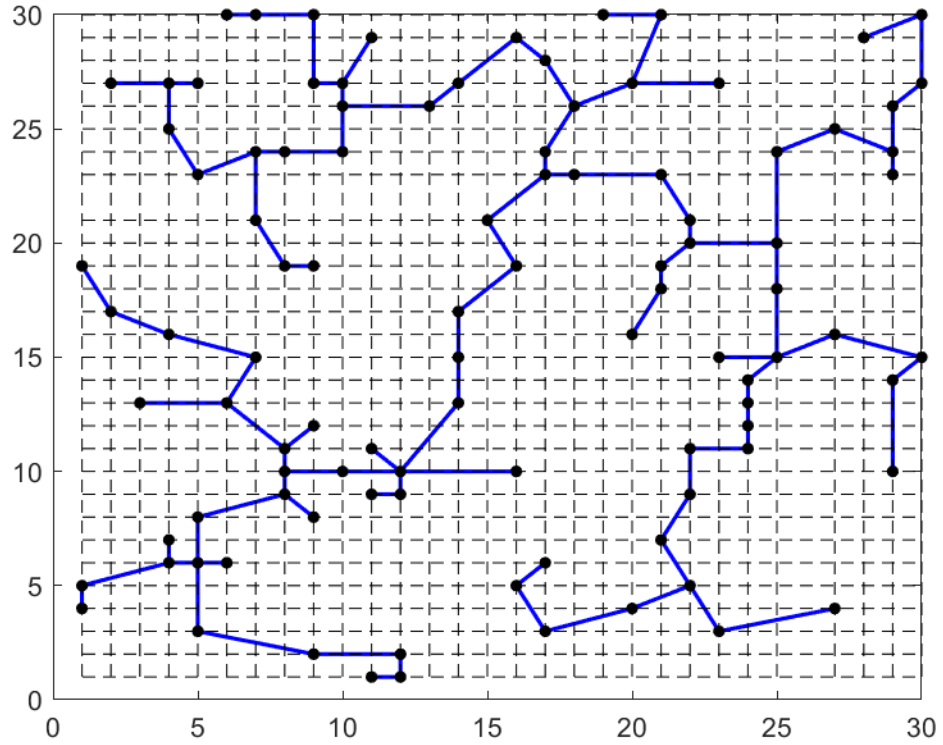
Final MRST



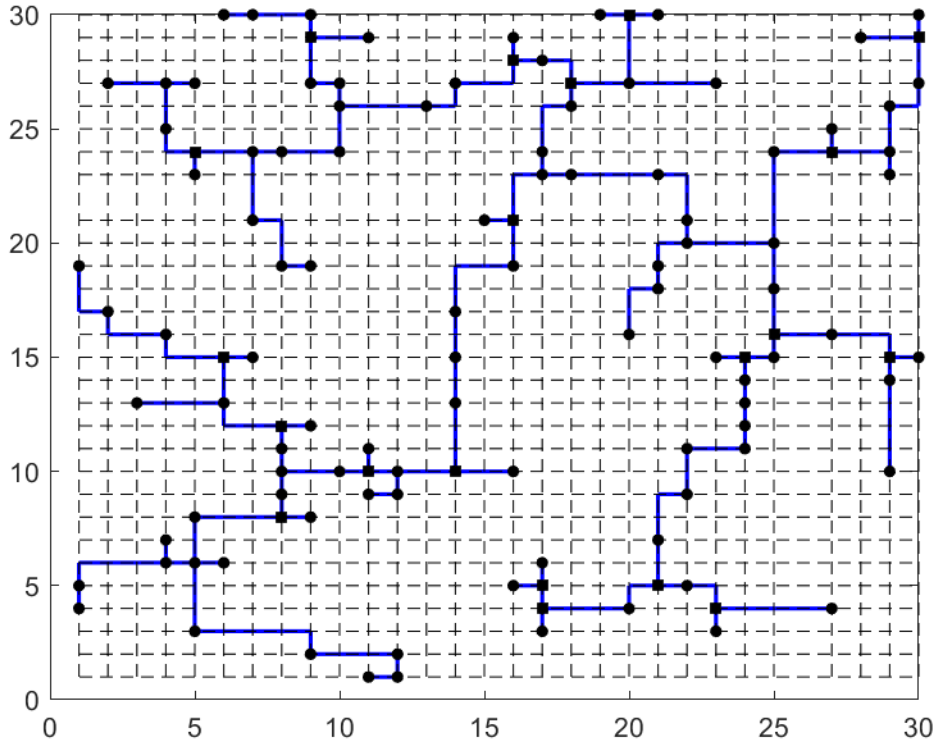
Basic 1-Steiner algorithm result

Grid Size 30, Number of Point 100

Initial MST



Final MRST

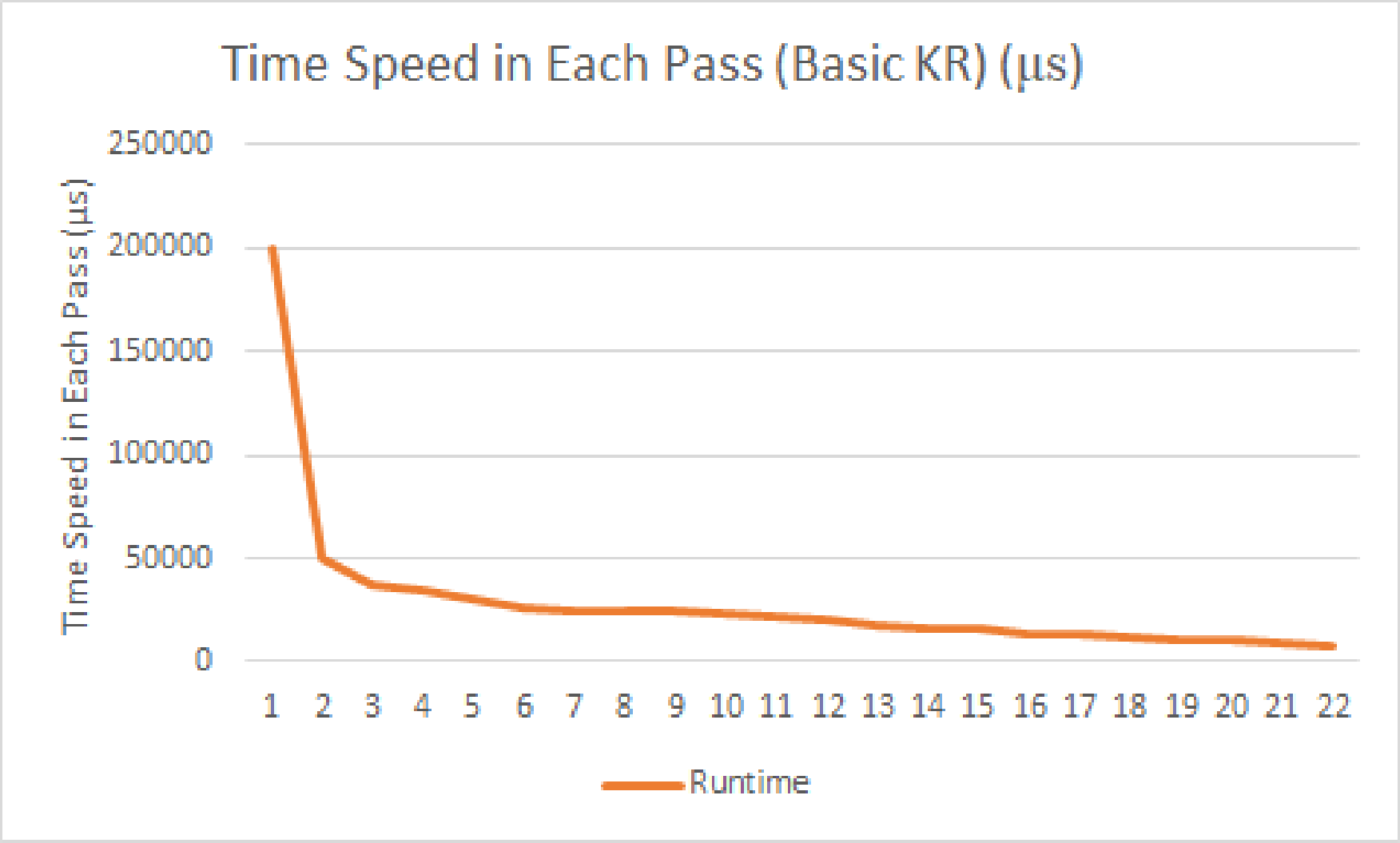


Basic 1-Steiner algorithm result

Total run time - 644213 μs

First pass - 200256 μs

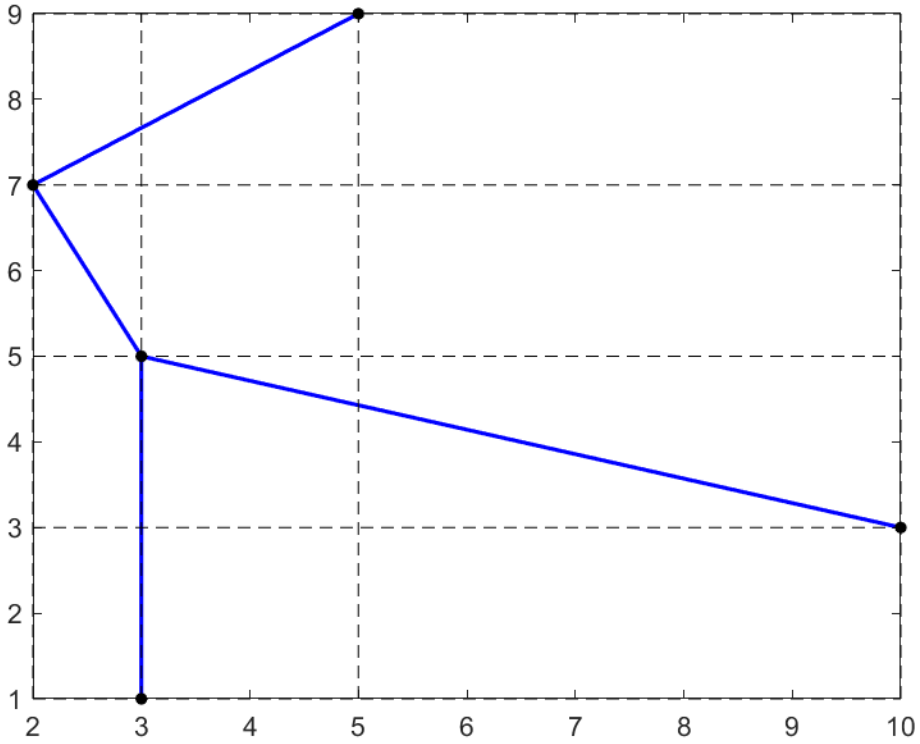
Lowest wirelength - 219



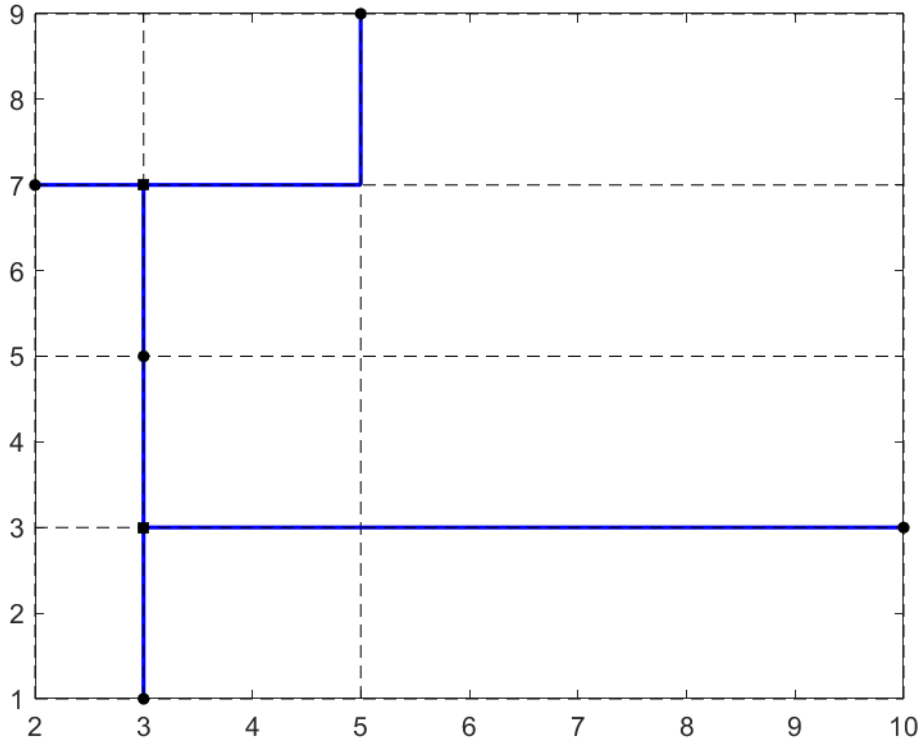
Random version 1-Steiner algorithm result

Grid Size 10, Number of Point 5

Initial MST



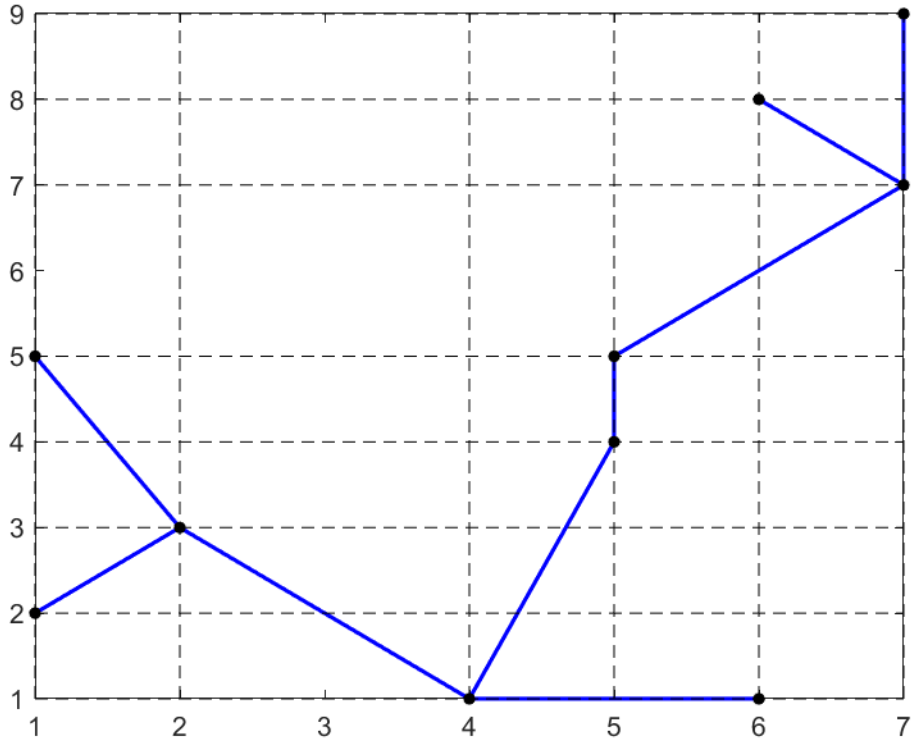
Final MRST



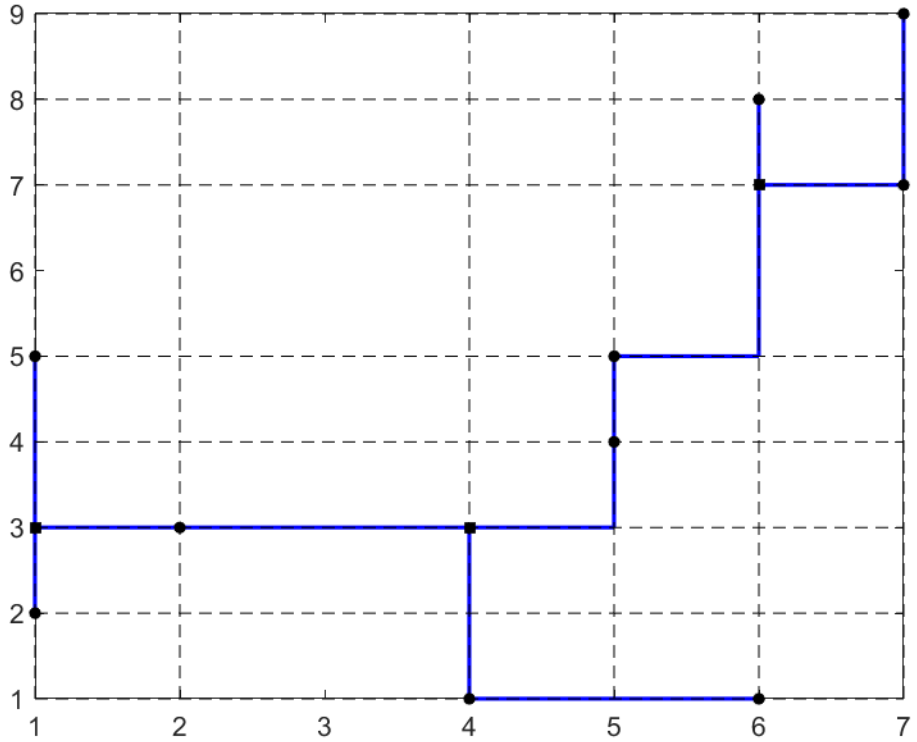
Random version 1-Steiner algorithm result

Grid Size 10, Number of Point 10

Initial MST



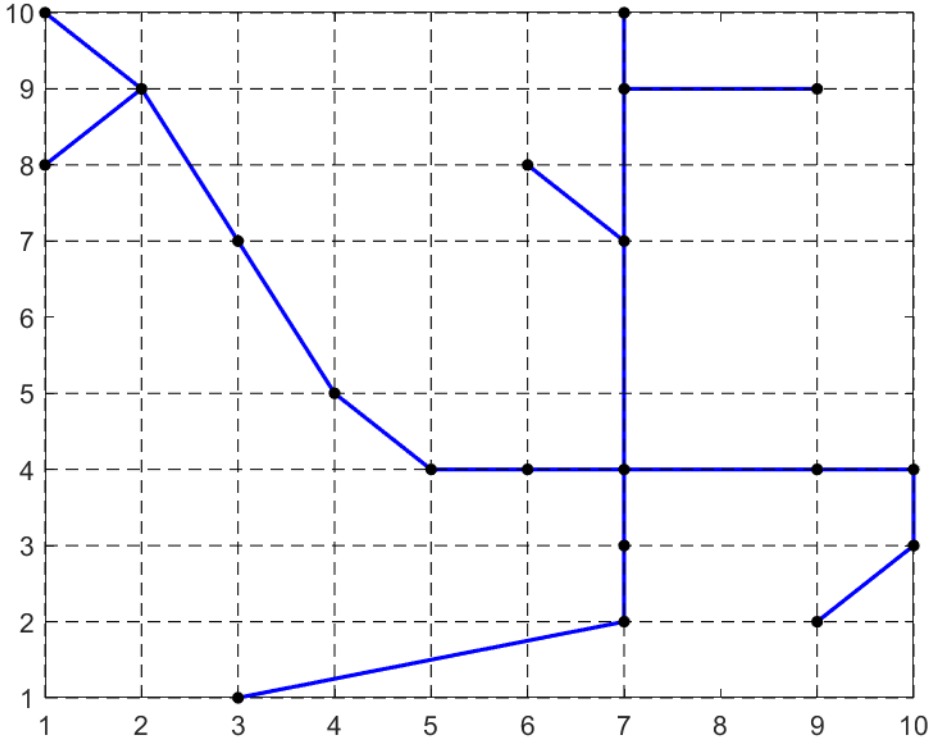
Final MRST



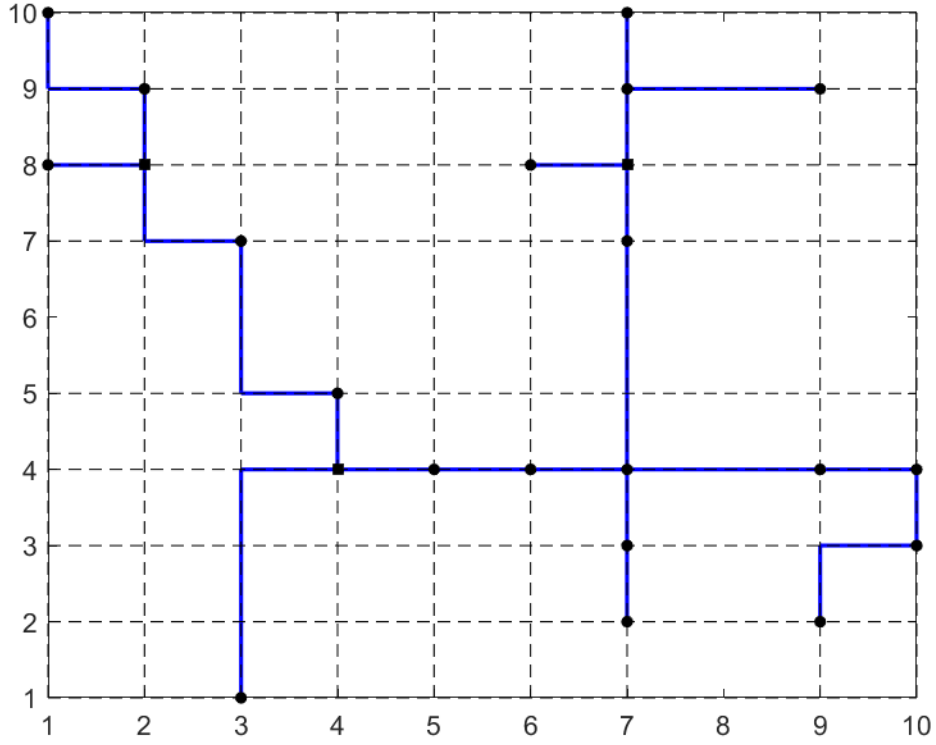
Random version 1-Steiner algorithm result

Grid Size 10, Number of Point 20

Initial MST



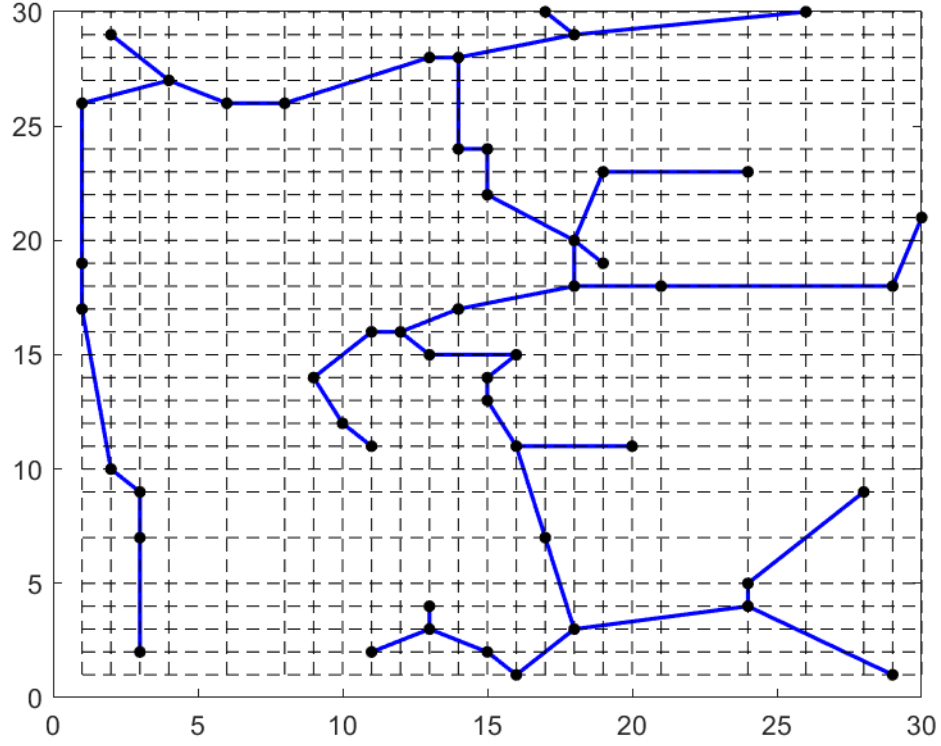
Final MRST



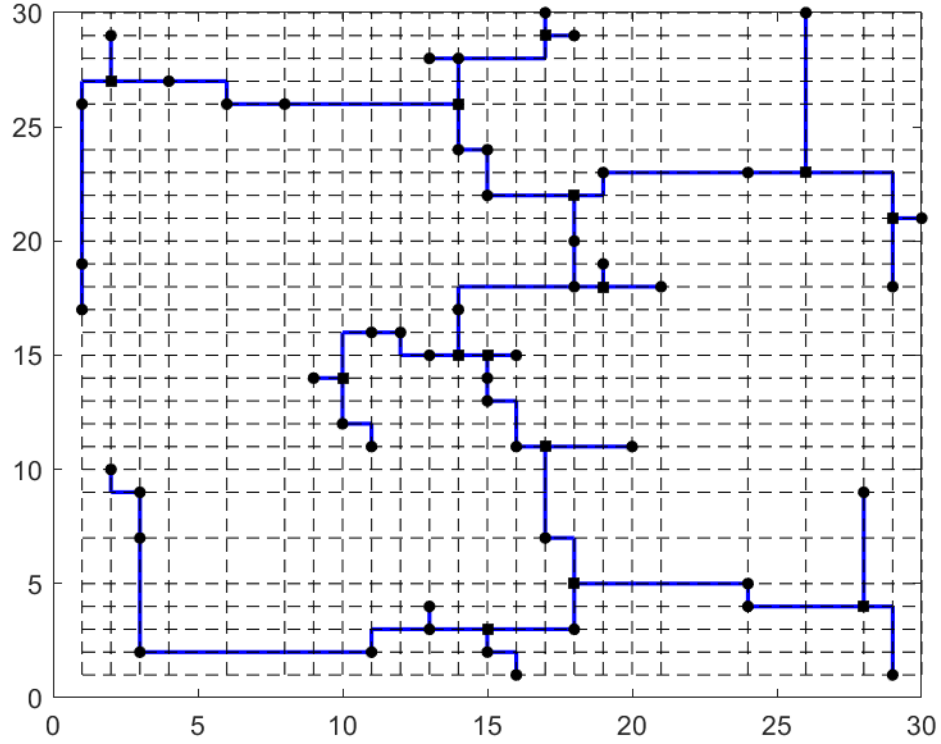
Random version 1-Steiner algorithm result

Grid Size 30, Number of Point 50

Initial MST



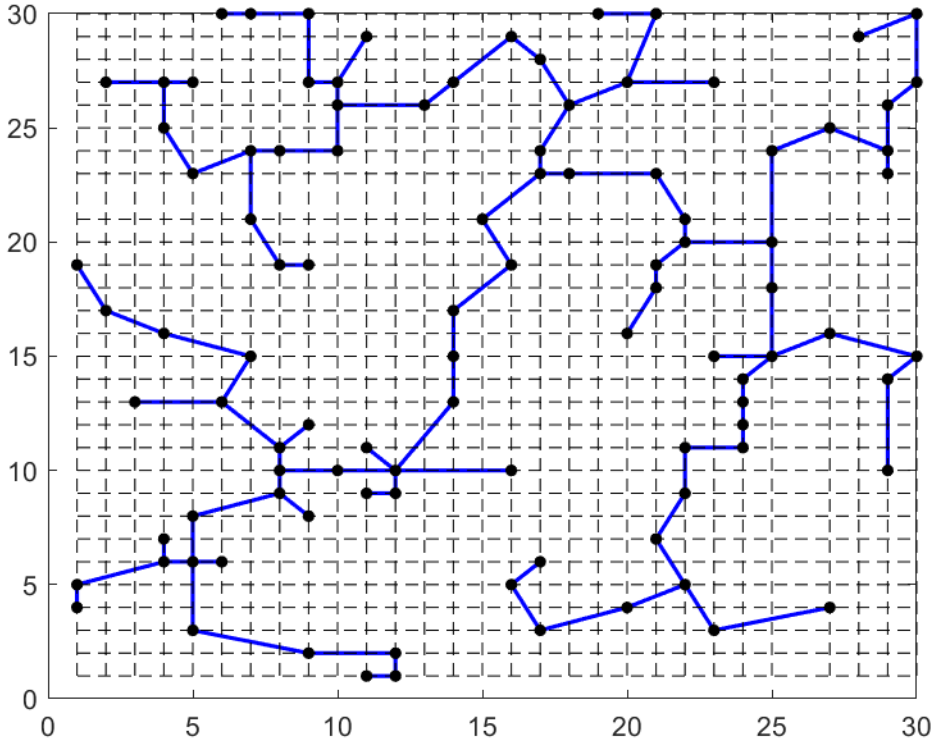
Final MRST



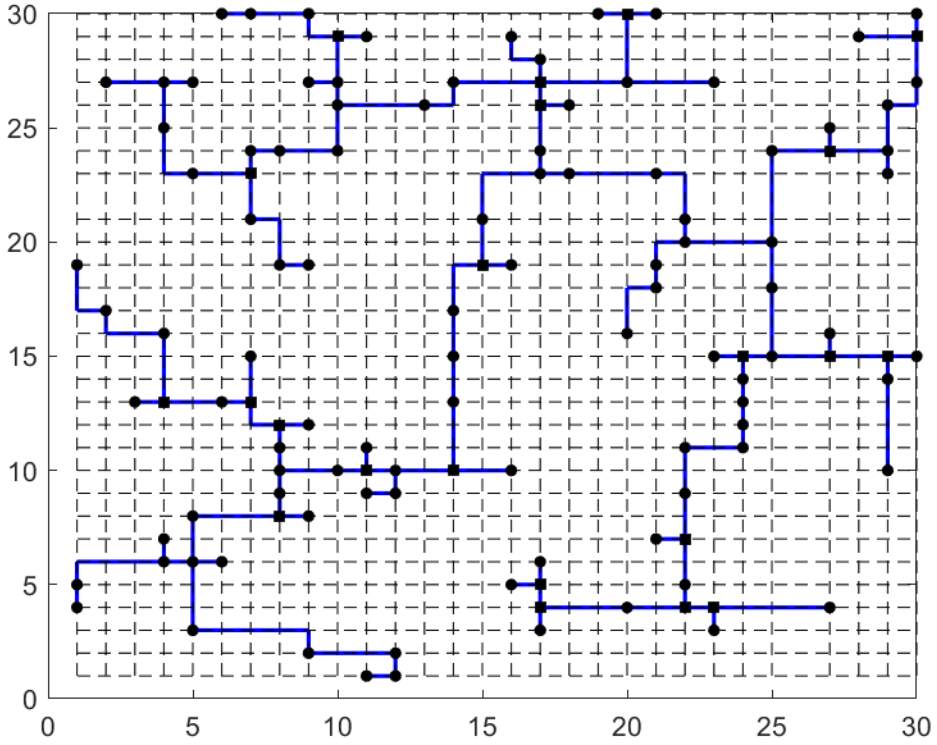
Random version 1-Steiner algorithm result

Grid Size 10, Number of Point 10

Initial MST



Final MRST

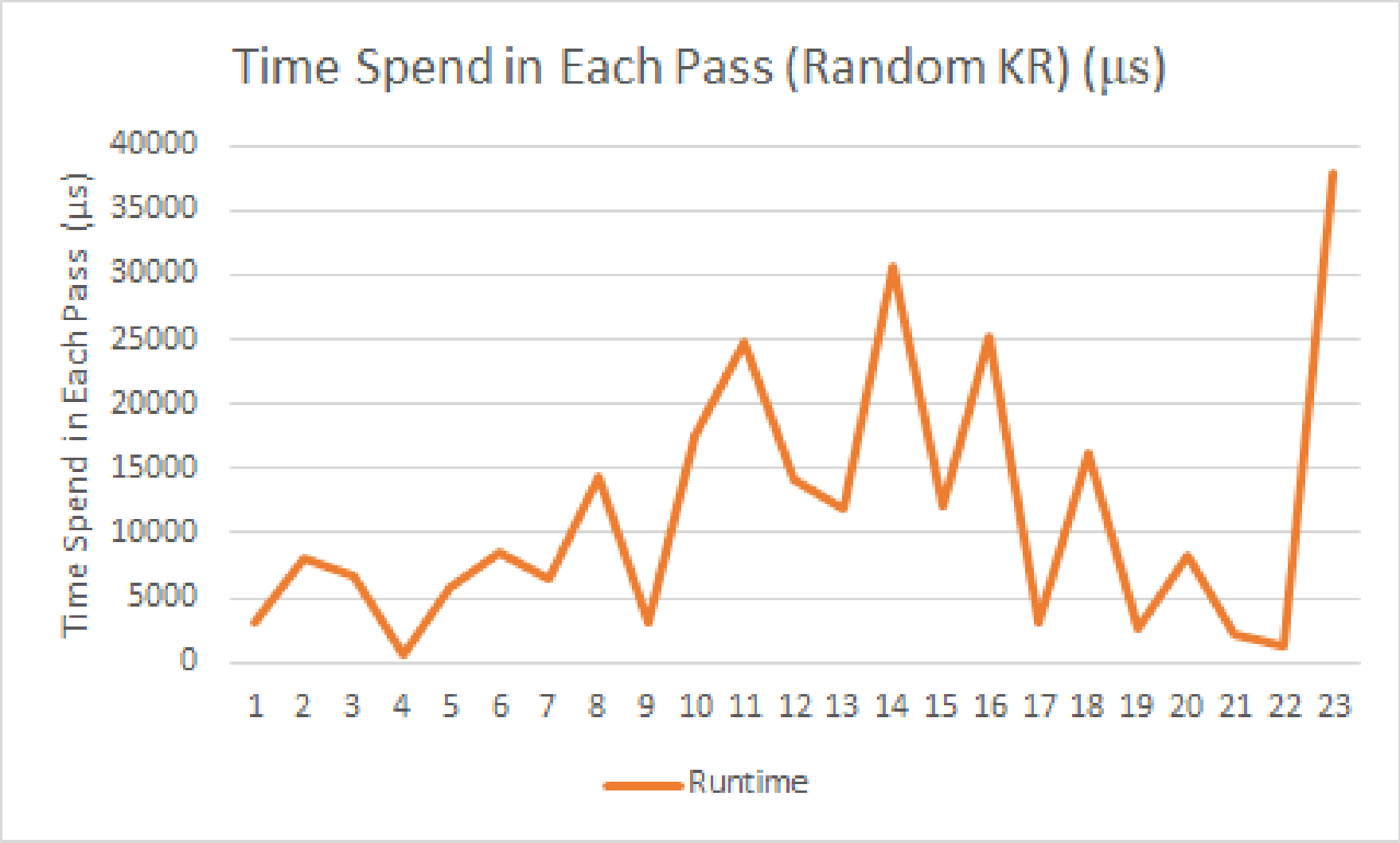


Basic 1-Steiner algorithm result

Total run time - 269446 μs

Slowest pass - 25215 μs

Lowest wirelength - 219



Quality comparison among 3 solutions(basic, random, and Prim)

Benchmark	Prim		Basic			Random		
	Avg. Runtime (us)	Wirelength	Avg. Iteration	Avg. Runtime (us)	Avg. Wirelength	Avg. Iteration	Avg. Runtime (us)	Avg. Wirelength
10_5	3862.6	21	2	3990.20	18	2	3659.90	18
10_10	4210.4	24	3	4984.80	20	3	4354.80	20.2
10_20	4104.6	37	3	6870.00	34	3	6051.80	34
30_50	4115.5	183	14.7	157352.60	163.3	14.7	103689.70	163.1
30_100	4928.6	242	21.6	738650.10	219.6	21.4	297927.20	220.1

Comparison among Prim and Kruskal MST algorithm

Benchmark	Basic Prim			Kruskal		
	Avg. Iteration	Avg. Runtime (us)	Avg. Wirelength	Avg. Iteration	Avg. Runtime (us)	Avg. Wirelength
10_5	2	3990.20	18	2	4064.90	18
10_10	3	4984.80	20	3	6164.30	20
10_20	3	6870.00	34	3	14430.80	34
30_50	14.7	157352.60	163.3	14.5	994669.70	163.4
30_100	21.6	738650.10	219.6	21.4	6134645.00	219.1

The effect of Size and Sparsity

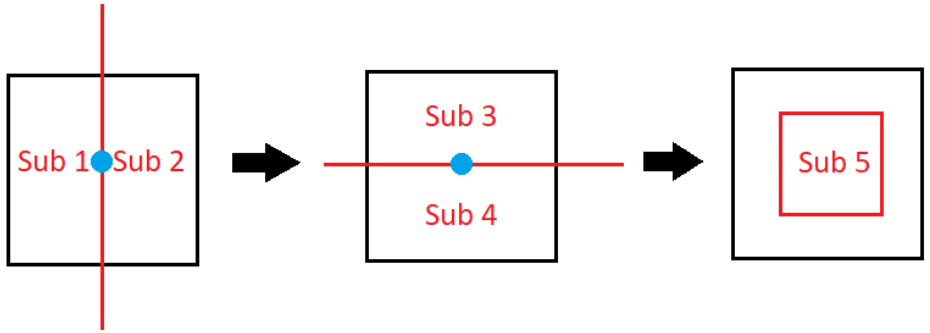
benchmark	Prim		Basic			Random		
	Runtime (ms)	Wirelength	Iteration	Runtime (ms)	Wirelength	Iteration	Runtime (ms)	Wirelength
30_100	4.93	242	21	779.73	219	22	289.66	221
50_500	10.54	895	59	48586.44	833	59	10522.78	835
50_1000	18.67	1283	70	120198.30	1212	71	26228.60	1212
Dense↑ Sparse↓								
100_500	12.55	1843	142	456875.28	1661	168	63412.73	1670
1000_100	5.76	8212	43	13990.82	7381	115	760697.65	7502

Modification – Advanced KR routing based on partition

- *Basic* KR can achieve good solution both in dense and sparse grid, but suffers from low speed…
- *Random* KR has good performance in dense grid, but cannot handle sparse grid…

- An attempt to partition the grid and do KR routing in each sub-region has been made. It reduces the runtime fairly at the cost of losing some optimality.

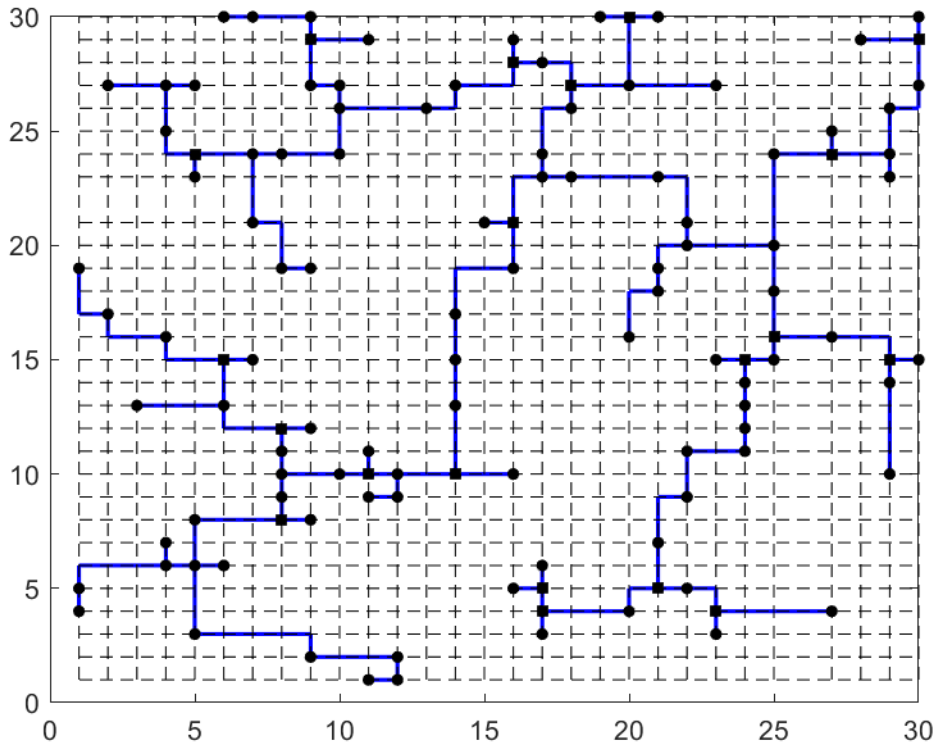
- We cut the grid with cutset, put a terminal on the cutline, and find 1-Steiner Points in each sub-region.



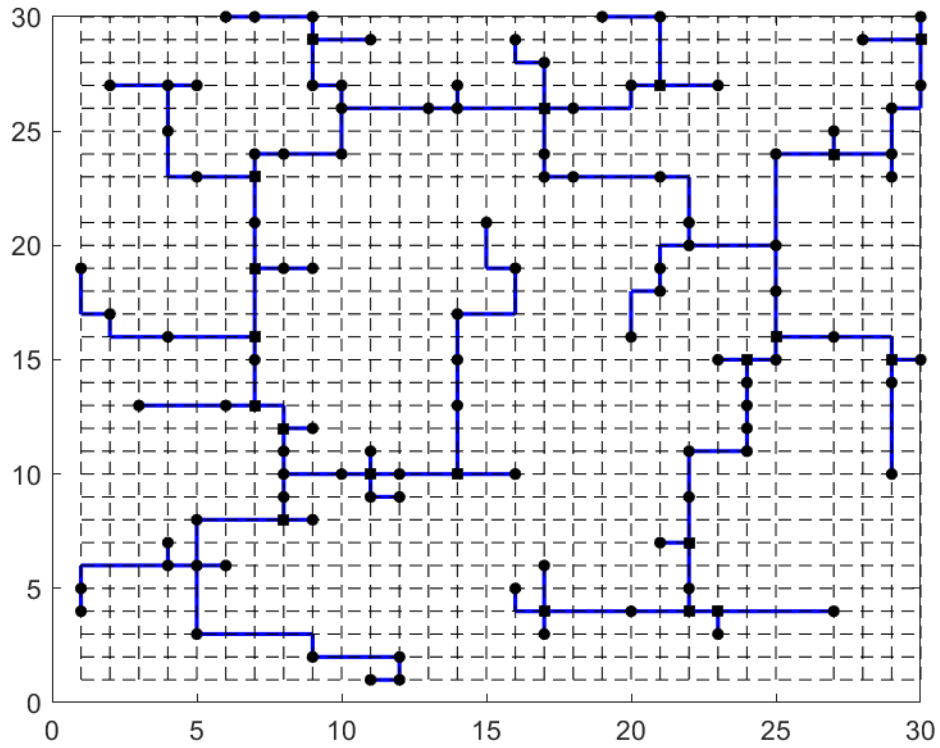
Modification – Advanced KR routing based on partition

Comparison between basic and partition 1-steiner algorithm

Original WL = 220 Runtime = 720785 μs



Partition WL = 221 Runtime = 278430 μs



Modification – Advanced KR routing based on partition

Comparison among basic, random and partition 1-steiner algorithm

benchmark	Basic			Random			Partition		
	Avg. Iteration	Avg. Runtime (us)	Avg. Wirelength	Avg. Iteration	Avg. Runtime (us)	Avg. Wirelength	Avg. Iteration	Avg. Runtime (us)	Avg. Wirelength
10_5	2	3763.90	18	2	3659.90	18	4	485.60	19
10_10	3	4984.80	20	3	4354.80	20.2	5	1097.80	20.6
10_20	3	6870.00	34	3	6051.80	34	5	2779.20	34
30_50	14.7	157352.60	163.3	14.7	103689.70	163.1	29.5	68590.10	163.5
30_100	21.6	738650.10	219.6	21.4	297927.20	220.1	43	271899.40	220.6

Modification – Comparison among Basic, Random and Partition KR

benchmark	Basic			Random			Partition		
	Iteration	Runtime (ms)	Wirelength	Iteration	Runtime (ms)	Wirelength	Iteration	Runtime (ms)	Wirelength
50_500	59	48586.44	833	59	10522.78	835	116	18789.35	837
50_1000	70	120198.30	1212	71	26228.60	1212	145	53773.53	1213
Dense↑ Sparse↓									
100_500	142	456875.28	1661	168	63412.73	1670	270	124828.85	1666
1000_100	43	13990.82	7381	115	760697.65	7502	97	4894.87	7405

Improvement

- The cutline for Partition is set manually, and will affect the performance. We can try to find a way to automatically find the best cutline.

- Considering Basic, Random and Partition is good at processing different types of grid, we could let the program itself decide which KR algorithm variant to use for the best performance.