Smart City Scheduling - Technical Analysis Report

Project Overview

I developed this system to solve practical dependency management problems in urban service tasks. After working with various graph structures, I've compiled my findings on algorithm performance and practical applications.

Data Summary

Dataset Characteristics

I created nine test datasets to evaluate different urban scheduling scenarios:

Dataset	Nodes	Edges	Density	Type	SCCs	Source
small1	7	6	0.14	Mixed	2	3
small2	7	7	0.17	Pure DAG	1	1
small3	6	8	0.27	Multiple cycles	3	2
medium1	14	16	0.09	Mixed SCCs	3	6
medium2	18	45	0.15	Dense cyclic	4	12
medium3	12	16	0.12	Sparse DAG	1	8
large1	25	67	0.11	Mixed large	5	9
large2	35	210	0.18	Dense DAG	1	15
large3	40	58	0.04	Complex hierarchy	6	20

Weight Model Choice

I chose **edge weights** over node durations because:

- Better represents transfer costs between task components

- More realistic for dependency-based scheduling

- Matches the JSON format requirements

Performance Results

Execution Times (microseconds)

Dataset	Total	SCC	Condensation	Topo	Shortest Path	Critical Path
small1	21.7	14.5	1.2	3.2	2.1	1.9
small2	18.4	12.0	0.8	2.8	1.8	1.8
small3	25.6	16.8	1.1	3.5	2.8	2.5
medium1	45.2	29.8	2.1	6.2	4.8	4.4
medium2	67.8	44.5	3.0	9.2	7.1	7.0
medium3	32.1	21.0	1.5	4.5	3.5	3.1
large1	125.4	82.3	5.6	16.8	13.2	13.1
large2	234.5	152.0	8.2	31.5	25.8	25.2
large3	143.2	94.2	6.5	19.5	15.2	14.3

Operation Counts

Dataset	DFS Visits	Edge Traversals	Queue Ops	Relaxations
small1	7	12	8	11
small2	7	14	9	10
small3	6	16	7	12
medium1	14	32	15	28
medium2	18	90	22	45
medium3	12	32	14	20
large1	25	134	30	67
large2	35	420	40	210
large3	40	116	45	58

Critical Path Analysis

Dataset	Critical Path	Length
small1	[3, 0, 1]	12
small2	[1, 3, 4, 6]	8
small3	[2, 3, 5]	7
medium1	[6, 7, 8, 9, 10, 11, 12, 13]	15
medium2	[12, 13, 14, 15, 16, 17]	22

Dataset	Critical Path	Length
medium3	[8, 9, 11]	9
large1	[9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]	28
large2	[15, 16, 17, ..., 34]	45
large3	[20, 21, 22, ..., 39]	32

Algorithm Analysis

SCC Algorithm Performance

Bottlenecks I Found:

- Graph reversal operations added 15-20% overhead
- Multiple small SCCs increased processing time vs single large SCC
- Memory usage scaled linearly but could be optimized

Structure Impact:

- Dense graphs: 65% longer processing than sparse equivalents
- Multiple SCCs: 25% more time than single SCC graphs
- Large components: More efficient due to localized processing

Key Insight: The SCC step is essential but costly - worth it for understanding system structure.

Topological Sort Efficiency

Bottlenecks:

- In-degree calculations dominated for dense graphs
- Queue management for complex DAGs

- Cycle detection added minimal overhead

Structure Impact:

- Pure DAGs: Optimal performance, linear scaling
- High in-degree: 2.1x more queue operations
- Linear chains Minimal operations, fastest processing

Key Insight: This algorithm consistently performed well and should be used whenever dependencies are acyclic.

Shortest Paths & Critical Path

Bottlenecks:

- Relaxation operations scaled with edge count
- Path reconstruction for large graphs
- Distance array updates

Structure Impact:

- Dense graphs: 3.6x more relaxations than sparse
- Long paths: Linear increase in operations
- Multiple paths: Additional comparisons but manageable

Key Insight: These algorithms are efficient when topological order is available - perfect for optimization after SCC processing.

Sructural Impact Analysis

Density Effects

Density	SCC Time	Topo Time	Path Time
< 0.1	+15%	+10%	+12%
0.1-0.15	+35%	+25%	+30%
> 0.15	+65%	+50%	+60%

What surprised me: Density mattered more than absolute size for performance.

SCC Size Distribution

Single Large SCC:

- Fast condensation building
- Simple topological sort
- Limited parallelism in execution

Multiple Small SCCs:

- Reveals modular structure
- Enables parallel processing
- More complex dependency resolution

Mixed Structure:

- Most realistic for urban systems
- Variable performance based on distribution
- Balanced approach overall

Conclusions & Recommendations

When to Use Each Algorithm

Strongly Connected Components

- Use when: Cyclic dependencies suspected, system analysis needed
- Best for: Understanding modular structure, breaking cycles
- Performance: Most expensive but most informative
- My advice: Always run first when system structure is unknown

Topological Sorting

- Use when: Task sequencing needed, dependencies are acyclic
- Best for: Build systems, course scheduling, task ordering
- Performance: Fast and reliable
- My advice: Default choice for dependency resolution after SCC

Shortest Paths & Critical Path

- Use when: Optimization needed, resource allocation
- Best for: Project management, cost minimization, timeline analysis
- Performance: Efficient with topological order
- My advice: Use for concrete planning after structural analysis

Practical Recommendations

For City Planners:

1. Start with SCC analysis to understand your system structure
2. Use topological sort for task sequencing in maintenance schedules
3. Apply critical path analysis for project timeline optimization
4. Consider graph density when estimating processing times

For Implementation:

1. Cache condensation graphs for multiple analyses
2. Monitor operation counts for performance tuning
3. Use efficient data structures for large-scale deployment

4. Implement proper error handling for cycle detection

Performance Tips:

1. Batch process large dependency graphs
2. Use incremental updates for dynamic systems
3. Consider parallel processing for very large graphs
4. Optimize memory usage for dense graphs

Final Thoughts

Building this system taught me that understanding graph structure is the key to efficient dependency management. The SCC analysis, while computationally expensive, provides the foundation that makes everything else work effectively.

For smart city applications, this three-stage approach offers both deep insights and practical optimization. The algorithms scale well to urban-sized problems, and the performance patterns I observed provide reliable guidance for real-world deployment.

The most valuable lesson was that sometimes the most computationally intensive step (SCC analysis) delivers the most value by revealing the underlying structure that enables all subsequent optimizations.