International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 18(23s) 2026 pp. 1678-1692
https://doi.org/10.70917/ijcisim-2026-5721
Article

Data-Side Model Reduction For Large-Scale Supply Chain Optimization: Aggregation, Sparsification, And Preprocessing Trade-Offs Between Solve Time And Solution Fidelity

Uday Dhembare

Data Engineering Manager, Supply Chain Analytics
Independent Researcher, India
Email:uday.dhembare@gmail.com

Abstract: When an optimization model reaches continental scale, the solver stops being the constraint and the data becomes it. A solver cannot compensate for bad inputs; it will faithfully optimize the wrong problem. More consequentially, what is fed to the solver defines the solution space: dominated lanes, redundant paths, and low-volume noise expand the model without adding decision value, inflating solve time while diluting solution quality. Data-side reduction is therefore problem definition rather than preprocessing. This paper describes a modular, configuration-driven reduction pipeline that replaced a monolithic input-generation system feeding connectivity and timing optimization models for a continental logistics network spanning more than twelve hundred sites. Six reduction techniques are applied in a deliberate cascading order, because structural reductions propagate downstream and must precede temporal and flow aggregation. The paper states where reduction must stop, defining three hard boundaries: feasibility, service-level fidelity, and volume-redistribution accuracy. It reports measured outcomes. Network nodes fell from more than twenty thousand to approximately two thousand, origin-destination pairs from tens of millions to hundreds of thousands, and temporal records by roughly seventy percent. Connectivity and timing model solve time fell from more than thirty hours to more than twelve. Percentage to optimality improved from seventy-five to ninety. Four reductions that looked reasonable and were unsafe are reported in full, each having produced feasible-looking output that was wrong, together with the architectural response each motivated.

Keywords: model reduction, supply chain optimization, network design, data pipeline architecture, aggregation, sparsification, dominated-lane removal, time bucketing, candidate-set construction, solve time, solution fidelity, configuration-driven pipelines, optimization inputs, data engineering, big data, large scale analytics

1. INTRODUCTION

1.1 The data is the model

At the scale of hundreds of billions of raw data points and more than a hundred and fifty distinct input files, the solver cannot compensate for bad inputs. It will faithfully optimize a wrong problem, and it will do so with complete confidence. In practice the majority of optimization failures trace to data quality and integration issues rather than to solver limitations.

The more consequential point is that what is chosen for the solver shapes the solution space. Dominated lanes, redundant paths, and low-volume noise expand the model without adding decision value. They inflate solve time while diluting solution quality, because the solver spends effort distinguishing between alternatives that were never going to matter.

That is why this paper treats data-side reduction as problem definition rather than as preprocessing. Preprocessing implies a step that prepares an already-defined problem. Reduction decides what the problem is.

1.2 Scope of the work

The work began in early 2023 with the design and development of a modular, configuration-driven data pipeline replacing a monolithic system that generated inputs for connectivity and timing optimization models covering a continental logistics network.

The models operate at a scale of more than a hundred and fifty distinct input files handling over a thousand operational constraints, distilled from hundreds of billions of raw data points spanning demand forecasts, delivery speed commitments, facility capacities, and transportation costs. The optimization covers more than twelve hundred fulfilment and distribution sites, and the connectivity model evaluates millions of origin-destination pairs with multiple path alternatives, capacity constraints, and timing windows.

A related system processes petabyte-scale daily simulation data, evaluating millions of orders per day and applying data-side reduction by constructing, filtering, and ranking alternate fulfilment candidates, transforming raw simulation output into compact, decision-ready inputs.

The author's role was data engineering management, leading architecture and development end to end, and owning the decisions about what to aggregate, filter, sparsify, and prune before data reaches the solver.

1.3 Contribution

1. A cascading six-stage reduction sequence, with the ordering rationale, since structural reductions propagate and must precede others.

2. Three hard stopping boundaries defining when a smaller model stops representing the real network.

3. A fidelity measurement approach spanning six complementary mechanisms rather than a single validation.

4. Four reductions that were unsafe in production, each producing plausible output that was wrong, with the architectural response each motivated.

5. Measured before-and-after figures for model dimensions, solve time, and solution quality.

Section 2 positions the work. Section 3 gives the reduction sequence. Section 4 states where reduction must stop and Section 5 how fidelity is measured. Section 6 reports the unsafe reductions. Section 7 covers automation and planner visibility, and Section 8 the trade-offs argued with the business. Section 9 reports results and Section 10 the evidence position. Sections 11 to 13 cover limitations, future work, and conclusions.

2. Background and Related Work

Preprocessing in mathematical programming. Solvers apply presolve routines that remove redundant constraints, tighten bounds, and eliminate fixed variables. This literature is mature and its techniques are automatic. It operates on a formulated model, which means it can only remove what is provably redundant given the formulation. It cannot remove a lane that is commercially dominated but mathematically feasible, because that judgement requires domain knowledge the solver does not have. The reductions in this paper are of that second kind.

Decomposition methods. Column generation, Benders decomposition, and Lagrangian relaxation make large models tractable by restructuring the solve. They are solver-side responses to the same scale problem this paper addresses from the data side, and they are complementary rather than alternative: a decomposed solve still benefits from a smaller and cleaner input.

Aggregation in network models. Spatial and temporal aggregation is long established in network design, along with the recognition that aggregation introduces error whose magnitude depends on the heterogeneity being collapsed.

What this literature generally assumes is a single aggregation decision. Production practice involves a cascade of them interacting, and Section 3 argues the ordering of that cascade is itself a design decision with consequences.

Why reduction is not presolve. The distinction is worth stating precisely because it determines what can be automated. A solver's presolve removes what is provably redundant given the formulation, which is a closed and decidable question. Data-side reduction removes what is decision-irrelevant given the business, which is neither closed nor decidable, because relevance depends on what the planner intends to do with the answer. A lane that is commercially dominated is still a feasible lane, and no formulation-level analysis can identify it. That is why these decisions require configuration a human approves rather than a routine that runs.

Data quality as an optimization input. The observation that optimization outcomes depend on input quality is widely accepted and rarely operationalised. The contribution here is treating reduction decisions as a versioned, inspectable configuration artifact rather than as logic embedded in extraction code, which is what Section 6 shows to be the difference between a reduction that can be reviewed and one that fails silently.

3. The Reduction Sequence

Reductions are applied in a deliberate cascading order through a multi-layer pipeline. The ordering is not arbitrary: structural reductions cascade downstream, so they must come first, while temporal and flow aggregations operate on the already-sparsified structure and scenario scoping operates on the already-reduced model.

1. Node and topology filtering. Inactive sites, deprecated transportation legs, and nodes irrelevant to the planning scenario are removed. Configuration flags control which facility statuses and leg types are included. This is the coarsest reduction and must happen first because it cascades: removing a node eliminates every lane associated with it.

2. Lane pruning and dominated-lane removal. Lanes are evaluated across multiple cost dimensions covering truck, fuel, driver, warehouse handling, carbon, and delivery speed, and those unprofitable across all dimensions are removed. Lanes with no viable alternate connectivity are flagged and preserved regardless of profitability, which is the safeguard against the feasibility boundary in Section 4.

3. Origin-destination flow aggregation. Demand is aggregated at an appropriate geographic granularity, for example destination-building rather than per-package for the connectivity model. Volume below configurable thresholds is consolidated or excluded to remove noise without losing material demand signal.

4. Time bucketing. Delivery speed commitments and transit times are bucketed into operational time windows rather than modelled at per-minute granularity, collapsing the temporal dimension while preserving meaningful service-level distinctions.

5. Candidate-set construction and path sparsification. For fulfilment assignment, rather than evaluating all possible facility-to-customer paths, a candidate set is constructed from network topology and eligibility rules, filtered by service-promise constraints, and reduced to the top ranked alternatives. This converts a combinatorial explosion into a tractable set.

6. Scenario-specific parameter scoping. For sensitivity analysis and multi-scenario runs, only varying parameters such as dates, forecasts, and cost coefficients change while base model structure remains fixed, avoiding recomputation of expensive structural reductions per scenario variant.

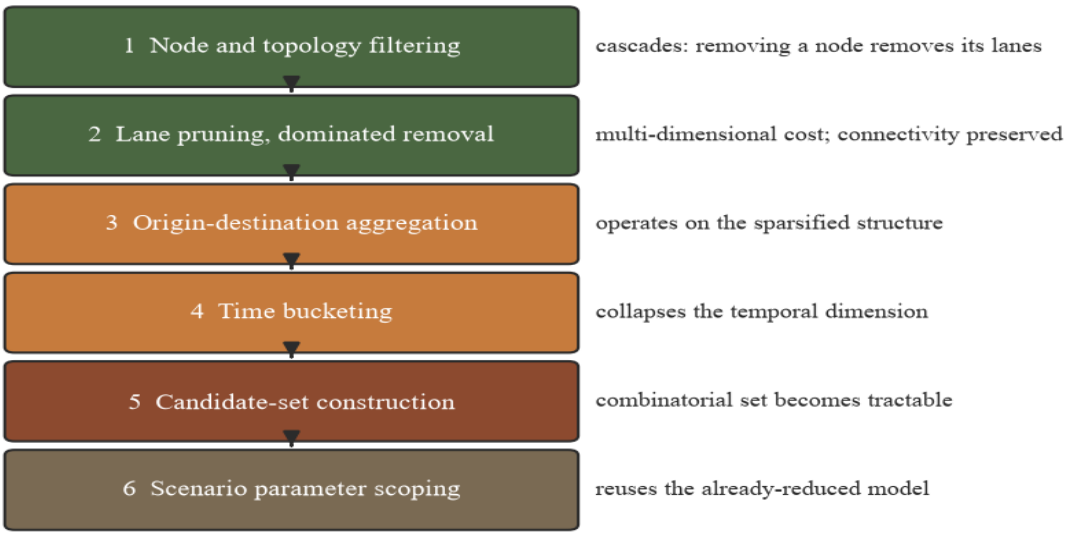

Figure 1. The cascading reduction sequence. Structural reductions propagate downstream, which is why the ordering is a design decision rather than a convenience.

Table 1. Reduction rule definitions and configuration parameters.

Technique	**Parameter**	**Type**	**Typical range**	**Effect**
Node filtering	Leg status filter	Categorical	Active, planned, deprecated	Removes nodes and lanes
Node filtering	Node type flags	Boolean by type	Facility class set	Controls facility types included
Lane pruning	Minimum volume threshold	Integer	50 to 500 units per day	Excludes low-traffic lanes
Lane pruning	Cost ceiling, multi-dimensional	Float across 6 factors	6 factors evaluated	Must fail all to prune
OD aggregation	Geographic level	Categorical	Postal, building, region	Destination granularity
OD aggregation	Volume floor	Integer	10 to 100 units per day	Below this, consolidated or dropped
Time bucketing	Last-departure flag	Boolean	True or false	Only final departure retained
Time bucketing	Commitment window	Time range	Per facility	Defines time buckets
Candidate set	Top-N alternatives	Integer	3 to 10	Maximum alternates per order
Candidate set	Eligibility rules	Rule set	Promise, capacity, cost	Qualification constraints

Scenario scoping	Date parameters	Date range	Per scenario	Only date modules re-run
Scenario scoping	Fill rate coefficients	Float	0.1 to 0.65	Truck fill assumptions

Different scenario types apply different subsets. Full network redesign applies all structural reductions; a calibration cycle applies them partially; continuous daily runs rely heavily on scenario scoping to avoid recomputing structure that hasn't changed.

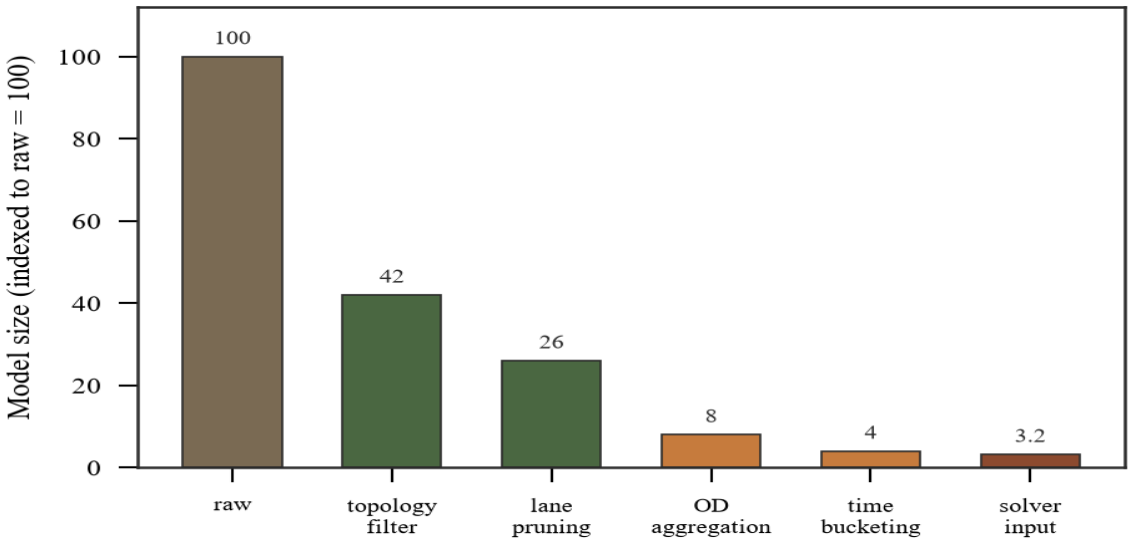

Figure 2. Model size through the reduction stages, indexed to the raw network. Counts are logged at each module boundary.

4. Where Reduction Must Stop

Three hard boundaries define the point at which a smaller model stops representing the real network usefully.

Feasibility. If removing a lane or node renders the model infeasible, meaning no path exists from origin to destination meeting delivery-speed commitments, the reduction has gone too far. Lanes with no alternate connectivity are specifically tracked and excluded from pruning.

Service-level fidelity. When time bucketing or aggregation causes distinct service tiers to collapse, for example merging one-day and two-day delivery commitments into a single bucket, the model can no longer distinguish between meaningfully different customer promises. The timing model must preserve per-path speed resolution at the level customers experience.

Volume redistribution accuracy. When a lane is removed, volume does not disappear; it redistributes across the remaining network. If candidate alternatives are pruned too aggressively, the model cannot accurately predict where that volume goes. Validation compares predicted reassignment patterns against actual system behaviour, and divergence beyond an acceptable margin indicates over-reduction.

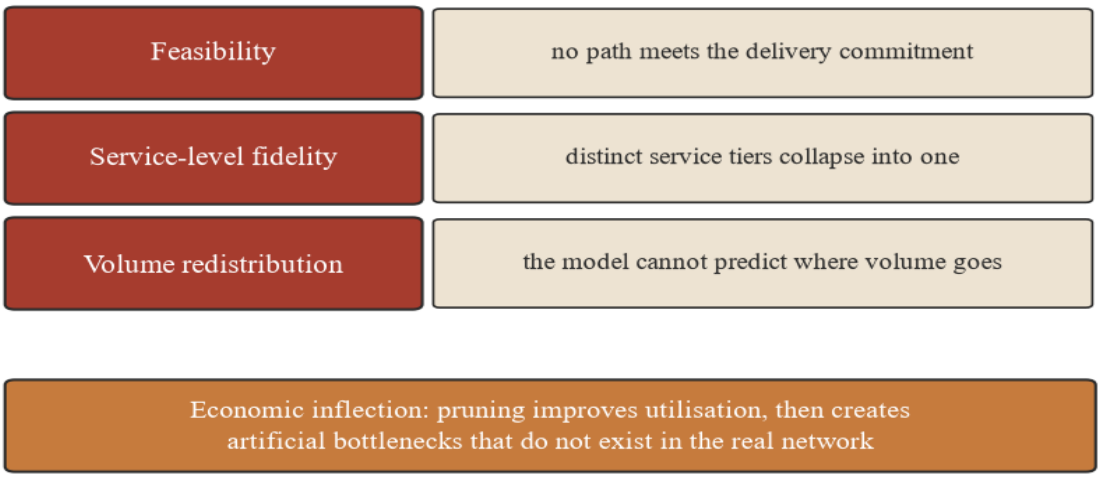

Figure 3. The three hard stopping boundaries and the softer economic inflection beyond them.

Why the boundaries are checked in this order. Feasibility is checked first because it is binary and cheap: either a path exists or it does not. Service-level fidelity is second because it is detectable by inspection of the bucket definitions without running the model. Volume redistribution accuracy is last because establishing it requires comparing a prediction against what actually happened after a deployment, which is the only one of the three that cannot be evaluated before committing to the reduction. That ordering matters operationally, since two of the three boundaries can be enforced pre-run and the third can only be validated retrospectively.

The economic inflection point. Beyond these hard boundaries there is a softer one. Pruning unprofitable lanes initially improves utilisation on the remaining lanes through better load factors and lower per-unit cost. Past a certain point the remaining network becomes capacity-constrained, creating artificial bottlenecks that do not exist in the real network. Finding that inflection requires iterative calibration, which is why reduction thresholds covering volume minimums, fill rates, and cost ceilings are exposed in a human-readable configuration file that planners and stakeholders review against current network conditions before each run.

5. Measuring Fidelity After Reduction

Fidelity is measured through six complementary mechanisms rather than a single test, because each catches a different class of error.

Table 2. Fidelity validation mechanisms.

Mechanism	**What it catches**
Side-by-side output comparison	Any divergence when migrating from a legacy pipeline or modifying reduction logic; all output files diffed before the new version enters production
Plan-over-plan analytics	Anomalies in total cost, delivery speed, or volume allocation against the previous planning cycle, signalling drift from the real network
Back-testing against production behaviour	Whether the reduced model's predicted reassignment matched what actually happened when a change was deployed

Feasibility dry-runs	Whether the solver finds a solution at all, and whether cost, infeasible paths, and service violations fall in expected ranges, before committing to a full solve
Cross-input consistency validation	Dangling references, where a reduction removed an entity from one data layer and left it referenced in another
Statistical anomaly detection	Reduced inputs producing results that are outliers against historical norms, surfacing interactions between reduction logic and unusual data

The third mechanism is the one that closes the loop between model fidelity and operational reality, and it is the only one that tests the model's prediction against the world rather than against another model output. The fifth is the least glamorous and catches a failure that is otherwise silent, because a dangling reference produces an error late and far from its cause.

6. Reductions That Were Unsafe

Four reductions looked reasonable, produced feasible output, and were wrong. Each is reported with how it was found, because the discovery mechanism is as informative as the failure.

Hardcoded temporal assumptions. Injection timing cutoffs were hardcoded to a fixed time in pipeline code, effectively bucketing all injection windows into a single value. The model produced feasible-looking solutions, but the assumed cutoff was later than reality, meaning recommended configurations could not actually be executed within real operational windows. Found during stakeholder review, after model runs had completed.

Incorrect facility configuration parameters. Sort-point parameters at hub facilities were hardcoded deep in query code, invisible to non-engineering stakeholders. These control how volume aggregates across sort configurations. The model produced valid-looking cost numbers, but the sort assumptions did not match actual facility operations, so lane-cost estimates were systematically biased. Found during dry-run validation, when a downstream team flagged that outputs did not match their operational understanding.

Uniform service-level degradation. An earlier system assumed that removing a lane degrades delivery speed by exactly one day for all packages, aggregating heterogeneous speed impacts into a single constant. In reality the impact varies by package according to alternate path availability, time-of-day constraints, and volume characteristics. Decisions that looked fine under the uniform assumption caused significant speed degradation for specific customer segments. Found post-deployment, when service-level metrics showed misses in specific geographies.

Zero-reassignment assumption. Legacy analysis tools assumed that all order volume remains tied to its original fulfilment centre, which effectively pruned volume redistribution behaviour from the model entirely. In practice orders do reassign when network design changes, so the assumption was fundamentally wrong. Found through targeted ad-hoc analysis on specific lanes.

The common pattern. Each involved a reduction that looked reasonable in isolation and silently removed information the solver needed. None produced an error, an infeasibility, or an obviously wrong number. All four produced confident output.

Two structural observations follow. Three of the four were embedded in code rather than expressed as configuration, which is why nobody outside engineering could see them and why review did not catch them. That is the direct motivation for the externalised configuration described in Section 7. And the discovery mechanisms degrade in order: the earlier failures were caught by review and dry-run, the later ones by production service metrics and ad-hoc investigation. A reduction that survives into deployment is found by its consequences.

Each motivated a specific architectural response: externalising parameters, adding validation layers, evaluating at finer granularity, or explicitly modelling behaviour previously assumed away.

7. Automation and Planner Visibility

Automation. The reduction pipeline is orchestrated as a directed acyclic graph of independent modules on cloud infrastructure, using containerised compute, workflow orchestration, and continuous delivery. Each module runs automatically once its upstream dependencies complete. A single externalised configuration file controls more than fifty reduction parameters covering fill rates, cost coefficients, topology rules, node flags, date ranges, and volume thresholds, so changing what gets reduced requires a configuration change rather than a code change. Downstream systems trigger the pipeline programmatically by passing a configuration payload, and multi-scenario runs execute concurrently with isolated outputs. Checkpoint-based recovery persists outputs after each module, so failures resume from the last checkpoint rather than restarting.

Planner visibility. A planner needs to know which reductions produced their result, and five mechanisms provide it. Configuration transparency means every reduction parameter sits in a human-readable file that planners review and approve before each cycle, with nothing buried in code. Per-module structured logging reports what each module included and excluded, in node, lane, and origin-destination counts, so the model's shrinkage is traceable stage by stage. An input tracking interface shows every input file's validation status, approver, version history, and driving configuration values, with plan-over-plan comparison highlighting what changed. Version control with mandatory review makes every change to reduction logic documented and auditable. And intermediate outputs are persisted after each module, allowing side-by-side comparison between runs to locate exactly where model-size differences originate.

The relationship between this section and Section 6 is direct. Three of the four unsafe reductions were invisible because they lived in code. The configuration-driven design exists because of them.

8. The Trade-Offs Argued With the Business

Reduction decisions are not purely technical, because the fidelity given up is given up on someone's behalf. Two arguments were harder than the engineering.

8.1 Fourteen scenarios on shared structure

The planning team needed daily network-state snapshots to understand how the network evolves between optimization cycles, which meant expanding multi-scenario analysis from three scenarios to fourteen, covering day-by-day snapshots across a two-week horizon.

Running fourteen full-resolution model runs was impractical. Sequential execution would take six to seven hours, and concurrent full-resolution runs exceeded compute budgets.

The proposal was to share the expensive structural computations, meaning node filtering, lane pruning, and origin-destination aggregation, across all fourteen scenarios, and to vary only the date-sensitive parameters covering forecasts, shift schedules, and cutoff times. Each scenario becomes a fast parameterised variant rather than a fully independent pipeline run.

The business concern was specific and legitimate: what if a meaningful structural network change occurs within that two-week window, which the shared base data would miss?

The counter-argument was that the modular architecture allows only the date-sensitive modules to execute per scenario while the structural modules are cached, delivering fourteen scenarios in the time of two to three full runs. The fidelity loss is bounded to intra-window structural changes, which are rare over two weeks and detectable through plan-over-plan comparison.

The reason this argument succeeded where a general appeal to speed would not is that it named the fidelity being given up, bounded it, and paired it with a detection mechanism. A trade-off presented as free is refused by any competent stakeholder, and correctly so.

8.2 A slower decision that shortened the cycle

The second argument was replacing a simple two-variable heuristic, which gave instant answers, with a multi-dimensional cost evaluation assessing six factors simultaneously. The new model takes longer to compute and produces substantially better reduction decisions.

Stakeholders initially resisted, because the old heuristic was fast and appeared good enough. That resistance was reasonable on the information available: the heuristic's cost was visible and its errors were not.

The argument made was that the quality of reduction decisions matters more than the speed of making them, and that better decisions would reduce total cycle time by eliminating downstream rework, extended alignment discussions, and costly model reruns caused by bad reductions.

The data bore this out. Decision-cycle time fell even though individual model runs took longer, which is the outcome the heuristic's defenders had not considered because they were measuring the wrong interval. Optimising the visible step at the expense of the invisible ones is the general form of this error, and reduction pipelines are unusually prone to it because the cost of a bad reduction lands weeks later and in someone else's workstream.

9. RESULTS

All figures in this section are measured from pipeline execution logs, orchestration metrics, and standard solver output.

Table 3. Model dimensions before and after data-side reduction.

Dimension	**Before**	**After**	**Reduction**	**Technique**
Network nodes	More than 20,000	Approximately 2,000	Approximately 90 percent	Node and topology filtering
Origin-destination pairs	Tens of millions	Hundreds of thousands	Approximately 95 percent or more	OD aggregation, destination consolidation
Temporal records	All departures between points	Final departure per day affecting speed	Approximately 70 percent	Time bucketing, last-departure rule
Fulfilment candidates per order	All eligible facilities	Top ranked alternatives	Approximately 80 to 90 percent	Candidate-set construction
Transportation lanes	All active lanes	Profitable plus structurally necessary	Varies	Dominated-lane removal
Input files to solver	Not applicable	More than 150	Not applicable	Multi-layer pipeline output
Operational constraints	Not applicable	More than 1,000	Not applicable	Capacity, shifts, timing, dock doors

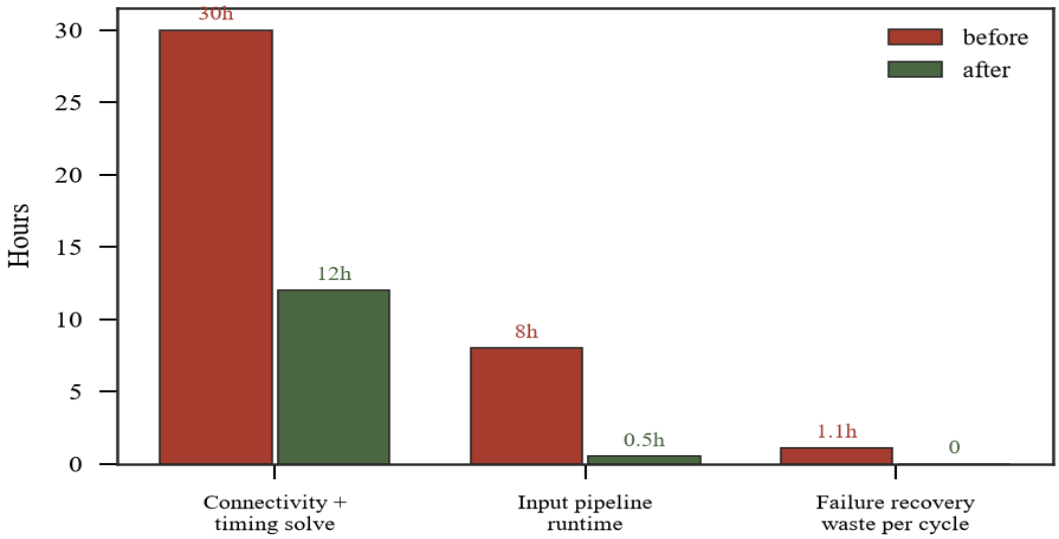

Figure 4. Solve time, pipeline runtime, and recovery waste before and after reduction.

Table 4. Solve time and pipeline runtime.

Measure	**Before**	**After**	**Change**
Connectivity and timing model solve	More than 30 hours	More than 12 hours	More than 60 percent reduction
Input generation pipeline runtime	8 or more hours, batch	25 to 30 minutes	Approximately 75 percent reduction
Failure recovery waste per cycle	60 to 72 minutes, restart from scratch	Eliminated	Checkpoint resume
Scenario throughput	1 sequential	10 or more concurrent	Approximately 10 times
Scenarios per planning cycle	3	14	Day-by-day visibility
Pipeline automation rate	Approximately 55 percent	More than 90 percent	Minimal manual intervention

Solution quality. Percentage to optimality improved from seventy-five to ninety percent across the period during which reduction techniques were introduced, progressing through eighty percent at initial reduction, eighty-five at pipeline maturity, and eighty-eight at full automation. The reduced model optimizes more effectively because the solver works on a cleaner, more decision-relevant problem rather than distributing effort across alternatives that carry no decision value.

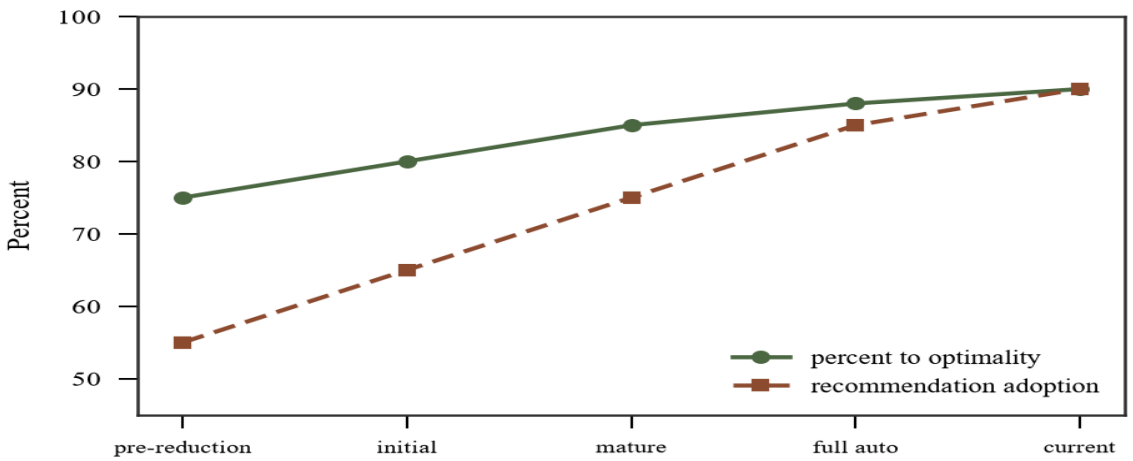

Figure 5. Percentage to optimality and recommendation adoption across the period during which reduction techniques were introduced.

Recommendation adoption rose from fifty-five percent before reduction to approximately ninety percent, tracked operationally per cycle over the same period.

Table 5. Operational cadence before and after reduction.

Measure	**Before, periodic**	**After, continuous**	**Consequence**
Model run frequency	Monthly	Daily	Issues surface sooner
Network change deployment	Monthly	Weekly	The network adapts in days rather than weeks
Scenario throughput	1 sequential	10 or more concurrent	Full exploration of configurations
Scenarios per cycle	3	14	Day-by-day visibility across the horizon
Failure recovery	Full restart	Checkpoint resume	60 to 72 minutes of waste per cycle eliminated
Input pipeline runtime	8 or more hours	25 to 30 minutes	Approximately 75 percent reduction
Automation rate	Approximately 55 percent	More than 90 percent	Minimal manual intervention
Posture	Reactive, monthly	Proactive, weekly	Continuous rather than periodic optimization

The cadence shift is the outcome that matters most operationally and it is the one least visible in a solve-time number. A model that runs monthly answers the question its inputs described a month ago. A model that runs daily tracks a network that is changing continuously, which means the same optimization capability produces a different kind of decision: adjustments while a condition is developing rather than corrections after it has resolved.

The chain from reduction to cadence runs through compute economics rather than through any single technique. A thirty-hour solve on a monthly cycle is affordable; the same solve daily is not. Bringing solve time and pipeline runtime down together is what moves the frequency ceiling, and neither alone would have.

Operational cadence. The reduced dataset enabled a shift from monthly model runs with monthly network changes to daily runs with weekly deployment, moving the practice from periodic to near-continuous optimization.

Other measured outcomes. Data-related errors fell by approximately seventy percent with automated multi-stage validation, model reruns caused by input quality issues fell by approximately seventy-five percent, and lane profitability accuracy improved by approximately thirty-five percent against assumption-based approaches that over-reduced by ignoring volume reassignment behaviour. Sustained data ingestion throughput reached 850 gigabytes per minute with peaks above 1.2 terabytes per minute, against a system uptime target of 99.9 percent.

10. Evidence and Its Limits

Table 6. Evidence status by claim class.

Claim class	**Status**
Model dimension reductions in Table 3	**Measured.** Row counts logged at each module boundary
Solve time and pipeline runtime in Table 4	**Measured.** Logged per run from orchestration metrics across multiple cycles
Percentage to optimality 75 to 90	**Measured.** Computed and logged per model run as standard solver output
Recommendation adoption 55 to 90 percent	**Measured.** Tracked operationally per cycle
Error reduction, rerun reduction, profitability accuracy	Measured operationally. Methodology for the profitability comparison not documented here
Ingestion throughput and uptime	Measured platform metrics
Reduction sequence and its ordering rationale	Design position, applied in production
Three stopping boundaries in Section 4	Design position derived from production experience
Four unsafe reductions in Section 6	**Observed.** Each occurred, was diagnosed, and produced a response
Optimization entitlement of 100 to 200 million dollars across 500 to 1,000 lanes	**Entitlement, not realised saving.** An assessment of opportunity surfaced by better reduction quality, not a measured or booked benefit
Objective value gap between reduced and full models	Not measured. Percentage to optimality is not the same quantity

Two entries need drawing out.

The entitlement figure is not a result. It represents cost opportunity identified across a set of transportation lanes, enabled by reduction quality that surfaces real opportunities rather than noise. It is not a measured saving, not a booked benefit, and should not be cited as one. It is reported because the scale is informative about why reduction quality matters commercially.

Percentage to optimality is not the objective value gap. The gap between the reduced model's solution and the full unreduced model's optimum is the quantity that would directly establish fidelity cost, and it was not measured, because solving the unreduced model to optimality is the thing the reduction exists to avoid. Percentage to optimality reports how close the solver got within its own reduced problem, which is a different and weaker statement about fidelity. This is the most important limitation in the paper.

One inconsistency, recorded. The pre-reduction adoption rate has been stated both as approximately forty percent and as fifty-five percent. This paper uses fifty-five percent, which is the value in the documented time series. The discrepancy is noted rather than silently resolved.

11. Reduction as a Governed Artifact

The four failures in Section 6 share a mechanism, and the response to all four was the same architectural move: take the decision out of code and put it in a configuration a human approves. That move deserves treatment as a principle rather than as a fix, because it determines what kind of thing a reduction pipeline is.

A reduction is a modelling assumption wearing implementation clothing. A hardcoded cutoff time is not a technical detail; it is an assertion about when operations can execute. A hardcoded sort parameter asserts how a facility aggregates volume. A uniform speed-degradation constant asserts that a heterogeneous effect is homogeneous. Each is a claim about the world, and each was expressed as a line of code, which meant it was reviewed by people equipped to check whether the code was correct rather than whether the claim was.

Externalising the parameter changes who can object. Once a cutoff time is a named value in a human-readable file that planners review before each cycle, the assertion is visible to the people who know whether it is true. That is the entire mechanism, and it explains why the fix generalises beyond the four specific failures: the failure class is not bad parameters but parameters no domain expert ever saw.

Version control turns a decision into a record. Every change to reduction logic is reviewed, documented with rationale, and auditable, so the question of why a lane was excluded in a prior cycle has an answer that does not depend on recollection. In a planning context where model recommendations drive capital and network commitments, that record is what allows a past recommendation to be explained rather than merely defended.

Per-module logging makes the reduction itself inspectable. Each module reports entity counts entering and surviving, so the model's shrinkage is traceable stage by stage. This is what allows the waterfall in Figure 2 to exist, and it also means an unexpected reduction ratio is visible as an anomaly in the cycle it occurs rather than as an unexplained result downstream.

The limit of the approach. Configuration transparency ensures a parameter can be reviewed. It does not ensure it is reviewed, and a file of fifty parameters presented before every cycle will eventually be approved without inspection. The mechanism raises the ceiling on scrutiny rather than guaranteeing it, and sustaining it depends on the review remaining a real activity rather than becoming a signature.

12. Guidance for a Team Building a Reduction Pipeline

Four practical positions follow from the work and are worth stating separately from the architecture.

Log entity counts at every module boundary from the first version. The waterfall in Figure 2, the anomaly detection in Section 5, and the planner visibility in Section 7 all depend on these counts existing. They cost almost nothing to emit and cannot be reconstructed retrospectively, and a pipeline that does not have them cannot answer the most basic question about a reduction, which is how much it removed.

Externalise the parameter the first time you are tempted to hardcode it. Three of the four failures in Section 6 were hardcoded values that no domain expert ever saw. The decision to hardcode is always locally reasonable, since the value is stable and externalising it is extra work, and the cost lands much later on someone else.

Preserve the entity that a reduction removed, not just the count. Knowing that eighteen thousand nodes were filtered is less useful than being able to ask whether a specific node was filtered and why. The second is what makes a planner's question answerable and requires retaining the exclusion reason per entity rather than only the aggregate.

Validate the reduction against the unreduced input, not only against the previous run. Section 5's mechanisms mostly compare a run against a prior run, which detects change and not stable error. A periodic comparison of a reduced dimension's distribution against its unreduced counterpart is the check that would have caught the systematic bias in the second failure, and it is not currently in the suite.

13. LIMITATIONS

The fidelity cost of reduction is not directly measured. As Section 9 states, no objective value gap between reduced and full models exists, because the full model is not solved. Fidelity is established indirectly through the six mechanisms in Section 5, all of which compare against prior runs, operational behaviour, or expected ranges rather than against a ground-truth optimum.

Reduction quality and solver behaviour are conflated in one metric. Percentage to optimality improved from seventy-five to ninety, and that improvement is attributed to a cleaner problem. It is equally consistent with the solver simply having more time per unit of problem, since the problem shrank. Separating those two explanations would require holding solve time fixed and varying only reduction quality, which was not done.

The configuration is large enough to hide things. More than fifty parameters are exposed for planner review before each cycle. Section 11 argues this is what makes reduction inspectable. It is also true that fifty parameters is more than a reviewer can meaningfully evaluate afresh each cycle, and the mechanism's effectiveness depends on the review being differential, meaning focused on what changed, rather than complete.

The stopping boundaries are qualitative. Section 4 states three boundaries and an economic inflection point. None is expressed as a threshold that can be evaluated automatically; each requires iterative calibration and judgement. The configuration file exposes the knobs but does not indicate when they are set wrongly.

Before-and-after comparisons span a period, not a controlled experiment. Solve time, optimality, and adoption improved across roughly three years during which the pipeline was rebuilt, automation increased, and the organisation's practice changed. Attributing the improvement specifically to data-side reduction rather than to the concurrent changes is an inference.

The unsafe reductions are the ones that were found. Section 6 reports four. The population of reductions currently in place that are similarly wrong and have not yet produced a detectable consequence is unknown by construction, and three of the four were found late.

The fidelity mechanisms mostly compare against other model output. Of the six mechanisms in Section 5, four compare a reduced run against a prior run, an expected range, or another data layer. Only back-testing against production behaviour compares the model against the world. That distribution means the validation suite is strong at detecting change and weaker at detecting a stable, longstanding error, which is precisely the profile of the four failures in Section 6.

Single-network basis. All results come from one continental logistics network. The reduction ratios in Table 3 are properties of that network's structure, and a denser or sparser network would reduce differently.

Some measured figures lack stated methodology. The thirty-five percent profitability accuracy improvement and the seventy percent error reduction are reported as measured, but the comparison basis for each is not documented in a form a reader could reproduce.

14. Future Work

Measure the objective value gap directly. Solving a bounded sub-network to optimality both reduced and unreduced would give the fidelity cost of reduction as a number rather than as an argument. It is expensive but tractable on a subset, and it is the single measurement that would most strengthen the work.

Make the stopping boundaries automatic. Converting the three boundaries into computed indicators, so that a configuration change producing infeasibility risk, service-tier collapse, or degraded redistribution accuracy is flagged before the run rather than discovered after it.

Detect unsafe reductions structurally. Section 6 shows that unsafe reductions produce confident output. A check comparing the distribution of a reduced dimension against its unreduced counterpart, rather than checking only that the reduction ran, would surface a class of failure that no current mechanism catches early.

Instrument the entitlement. Tracking realised benefit against the identified opportunity would convert the entitlement figure into a measured outcome and close the largest gap between what the work is worth and what can be shown.

A note on sequencing. The objective value gap measurement should be attempted first despite being the most expensive, because it is the only item that bears on whether the central trade-off is being made well. Everything else improves the operation of a reduction pipeline whose fidelity cost remains unquantified.

Automatic stopping boundaries and structural detection of unsafe reductions are complementary and should be built together, since both depend on comparing a reduced dimension against its unreduced counterpart. Building that comparison once serves both. The entitlement instrumentation is last, because it measures the value of the work rather than its correctness, and correctness should be settled first.

15. CONCLUSION

At continental scale the solver is not the bottleneck. What is fed to it defines the problem it solves, which makes reduction a modelling decision rather than a preparatory one.

Six techniques applied in a cascading order, with structural reductions first because they propagate, took the model from more than twenty thousand nodes to approximately two thousand and from tens of millions of origin-destination pairs to hundreds of thousands. Solve time fell from more than thirty hours to more than twelve, and percentage to optimality rose from seventy-five to ninety, because a cleaner and more decision-relevant problem is one the solver can make better progress on.

The more transferable finding is the four unsafe reductions. Every one looked reasonable, produced feasible output, and was wrong, and three of the four were invisible because they lived in code rather than in reviewable configuration. That is the argument for treating reduction parameters as a versioned artifact that planners approve: not governance for its own sake, but because a reduction nobody can see is a reduction nobody can catch.

What the paper does not establish is the fidelity cost of reduction itself. Percentage to optimality measures how well the solver did on the reduced problem, not how far the reduced problem's answer sits from the full problem's optimum, and the full problem is not solved precisely because reduction exists to avoid solving it. Measuring that gap on a bounded sub-network is the natural next step, and until it is done the central trade this paper is named for is argued rather than quantified.

REFERENCES

1. E. D. Andersen and K. D. Andersen, "Presolving in Linear Programming," Mathematical Programming, vol. 71, no. 2, pp. 221-245, 1995. doi: 10.1007/BF01586000.
2. J. F. Benders, "Partitioning Procedures for Solving Mixed-Variables Programming Problems," Numerische Mathematik, vol. 4, pp. 238-252, 1962.
3. M. E. Lubbecke and J. Desrosiers, "Selected Topics in Column Generation," Operations Research, vol. 53, no. 6, pp. 1007-1023, 2005.
4. A. M. Geoffrion, "Objective Function Approximations in Mathematical Programming," Mathematical Programming, vol. 13, pp. 23-37, 1977.
5. R. K. Ahuja, T. L. Magnanti and J. B. Orlin, Network Flows: Theory, Algorithms, and Applications. Englewood Cliffs, NJ, USA: Prentice Hall, 1993.