ASP Encodings for the Multi Batching Problem in Logistics Networks

Racquel Dennison[1]

[1]University of Cape Town, South Africa

Abstract

Supply chain architects are often faced with a recurring routing and packing problem: given a network of suppliers, hubs, and consumers spread across several countries, one must decide which parts travel which routes, at what frequency, and how parts should be packed. This paper describes our current research on encoding this problem in Answer Set Programming (ASP). We found that a direct translation of the problem does not scale to industrial use cases, so to address this we first applied a set of encoding optimisations and then decomposed the problem into two stages. Our attempts explored using ASP and Constraint Answer Set Programming. With regards to the two stage decomposition, the first stage fixes the aggregate flows across the network, while the second assigns parts to individual trips using soft constraints that favour robust packings. Preliminary experiments show that the optimised encoding cuts solver search time substantially compared to the baseline encoding, and the two-stage decomposition offers a way to encode robustness directly into the network. Remaining work involves a full experimental study of how different encoding choices, with regards to specifying how parts should be packed and load should be distributed, trade off cost against robustness.

Keywords

Answer set programming, Constraint programming, Supply chain robustness, Multi-commodity flow

1. Introduction

Products within a production pipeline are built from parts arriving from suppliers scattered across the globe. A typical Airbus network, for example, spans dozens of locations connected by different transport resources, each equipped with its own capacity and cost. The central planning question, therefore, becomes; given the supply and demand at each location within the network, which routes should carry which parts, at what dispatch frequency, and how should those parts be loaded onto individual trips?

Flamand et al. [1] formalise this as the Transportation Problem with Packing Constraints. We will denote this problem as the Multi Batching Problem. Earlier studies on TPPC primarily optimise for cost or feasibility and use Constraint Programming methods to solve it. A particular area of interest with regards to this problem is the concept of robustness. Given the volatility of supply chain networks, one unplanned disruption could drastically affect the ability for demand to be serviced. Literature on supply chain planning has highlighted the need for encoding robustness during the pre disruption and post disruption phase in planning [13].

This research asks two related questions. First, how well does ASP handle the Multi Batching Problem, and what encoding choices make it tractable on realistic instances, particially the instances used by Airbus? Second, can soft constraints be layered into the encoding to produce solutions that are cost effective and robust to disruptions?

2. Background

2.1. Answer Set Programming

ASP is a declarative problem-solving paradigm based on the stable model semantics [5]. An ASP program consists of rules, constraints, and aggregates. The standard solving pipeline is a two-step process. First, a grounder instantiates the program against a set of facts, replacing variables to create

Doctoral Consortium of the 23rd International Conference on Principles of Knowledge Representation and Reasoning (KR 2026 DC), July 20-23, 2026, Lisbon, Portugal

published 2026-06-30

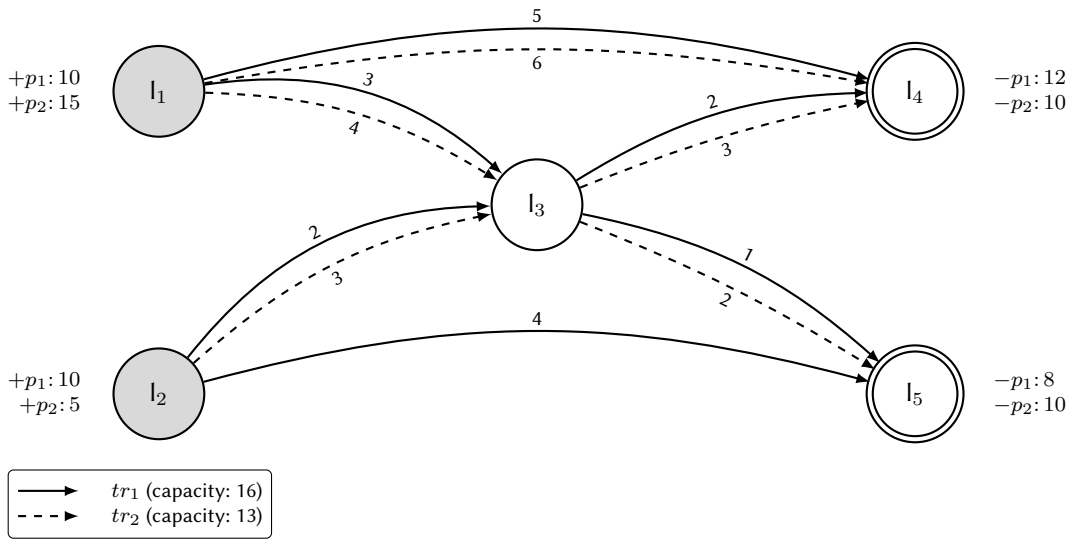

Figure 1: Example of a logistics chain network with demand and supply nodes.

a finite, propositional representation; thereafter, a conflict-driven solver searches the variable-free space for stable models that satisfy the constraints defined. To handle optimisation, ASP employs weak constraints that express preferences, using lexicographic priority levels to dictate which objectives dominate the search.

Standard ASP struggles with large numerical domains because grounding them causes a combinatorial explosion. The clingcon extension [7] tackles this by adding linear arithmetic theory atoms to ASP. This enables integer variables to be declared and constrained directly at the solver level without grounding their full domains.

2.2. The Multi Batching Problem

The Multi Batching Problem combines two classical problems. The routing layer is a multi-commodity flow problem. This decides how many units of each product flow along each directed arc, satisfying supply and demand at each node. The packing layer assigns those units to individual trips, subject to vehicle capacity on each arc. The two layers interact by making sure that the frequency must be large enough to carry the total load, but each trip must not exceed the capacity of a given transport resource. To illustrate the Multi-Batching Problem, consider a logistics network with the locations $\{\mathrm{l}_1, \mathrm{l}_2, \mathrm{l}_3, \mathrm{l}_4, \mathrm{l}_5\}$. Locations l_1 and l_2 are supply nodes, l_3 is a transhipment hub, and l_4 and l_5 are demand nodes. Two transport resources are available: tr_1 with capacity 15, and tr_2 with capacity 10. Two parts are transported: p_1 with size 4 and p_2 with size 3. Location l_1 supplies 10 units of p_1 and 15 units of p_2, while l_2 supplies 10 units of p_1 and 5 units of p_2. On the demand side, l_4 requires 12 units of p_1 and 10 units of p_2, and l_5 requires 8 units of p_1 and 10 units of p_2. Figure 1 represents this network. An example of a package to satisfy the demand of p2 at l_5 would route 5 items of p_2 from $\mathrm{l}_1 \rightarrow \mathrm{l}_3 \rightarrow \mathrm{l}_5$ at a frequency of 2.

A cost-optimal choice might route all of p_1's supply from l_1 to l_4 exclusively via tr_1, as it is the cheaper resource. However, if tr_1 is disrupted, the entire supply of p_1 to l_4 is lost. This is a vulnerability, one of which we aim to address with ASP soft constraints.

3. Problem Formulation

3.1. Network Structure

The network is modelled as a directed graph $G = (L, E)$, where L is the set of locations and E is the set of directed arcs between them. A location is either a supplier, a consumer, or a hub through which

parts pass in transit. Each arc in E is served by one or more transport resources $\tau \in T$, where τ has a fixed capacity c_τ and a per-trip cost $cost_\tau$. Each part $p \in P$ has a size s_p and a value v_p. Every network is balanced: the total supply of each part across all supplier locations equals the total demand across all consumer locations.

3.2. Decision Variables

A valid solution determines three quantities.

- **Flow** $flow(from, to, p) \in \mathbb{Z}_{\geq 0}$: the number of units of part p routed along arc $(from, to) \in E$.
- **Frequency** $freq(k, from, to, \tau) \in \mathbb{Z}_{>0}$: the number of trips made by resource τ on arc $(from, to)$ for package k.
- **Package load** $pack(from, to, p, k, \tau) \in \mathbb{Z}_{\geq 0}$: the number of units of part p assigned to package k carried by resource τ on arc $(from, to)$.

3.3. Constraints

Three constraints govern every valid solution: parts must be conserved at each node, no individual trip may be overpacked, and the units assigned across all packages on an arc must account for the total flow on that arc.

Flow conservation. For each part p and location l, the inflow minus the outflow must equal the net demand: $\sum_{from \in L} flow(from, l, p) - \sum_{to \in L} flow(l, to, p) = \text{netflow}(p, l) \quad \forall p \in P,\ l \in L$ where $\text{netflow} : P \times L \to \mathbb{Z}$ is positive at consumer locations, negative at suppliers, and zero at hubs.

Per-trip capacity. The total size of parts loaded onto any single package must not exceed the capacity of the transport resource: $\sum_{p \in P} s_p \times pack(i, j, p, k, \tau) \leq c_\tau \quad \forall (i, j) \in E,\ \tau \in T,\ k \in K_{ij\tau}$ where $K_{ij\tau}$ is the set of packages carried by resource τ on arc (i, j).

Load conservation. The units delivered across all packages for part p on arc $(from, to)$ via resource τ must equal the total flow of p on that arc. Since each package k is dispatched $freq(k, from, to, \tau)$ times, the conservation is: $\sum_{k \in K_{from,to,\tau}} pack(from, to, p, k, \tau) \times freq(k, from, to, \tau) = flow(from, to, p)$ $\forall (from, to) \in E,\ \tau \in T,\ p \in P$

Objective. The primary objective is to minimise total transport cost across the network,
$\min \sum_{(from,to) \in E} \sum_{\tau \in T} \sum_{k \in K_{ij\tau}} freq(k, from, to, \tau) \times d_{ij} \times cost_\tau$ where d_{ij} is the distance on arc $(from, to)$.

4. Approach

Our work has gone through three phases. We first defined a baseline ASP encoding that handles flow, packing, and frequency as a single monolithic problem. This established correctness and gave us something concrete to measure against. The second phase introduced a set of structural optimisations to the baseline, such as tightening variable domains, pruning unreachable routes during grounding, and using symmerty breaking. These changes cut solver search effort substantially but still showed little improvement when considering larger networks. The third phase decomposed the problem entirely, separating the flow and frequency decisions from the packing assignments. Upon investigating the packages produced by the two stage approach, we saw that the packages themselves were not robust. We define robust in the sense that if a disruption happened within the network which affected the supply of a part by a specified percentage, the packages were robust if they were still able to serve a high demand.

4.1. Baseline Encoding

The baseline is a direct ASP translation of the problem formulation. Flow is assigned to every route that neither leaves a consumer nor enters a producer. Packing assigns units to bins on each route. In this case, a bin would represent some trip. Frequency assigns a dispatch count to each trip (bin):

```
1{ flow(From,To,Part,N) : numFlow(N) }1 :-
    route(From,To,_,_,_), part(Part),
    not is_consumer(Part,From),
    not is_producer(Part,To).

{pack(N,P,ID,From,To,TR) : numParts(N)} = 1 :-
    route(From,To,TR,_,_), bins(ID), part(P).

1{freq(ID,From,To,TR,F) : numFreq(F)}1 :-
    route(From,To,TR,_,_), bins(ID).
```

Three integrity constraints eliminate infeasible assignments. Flow must be conserved at every node, no bin may exceed vehicle capacity, and the total units dispatched across all bins on a route must cover the assigned flow:

```
:- #sum{ N,To   : flow(Loc,To,Part,N);
        -N,From : flow(From,Loc,Part,N) } != DS,
   netdemandSupply(Part,Loc,DS).
:- #sum{N*Size : pack(N,P,ID,From,To,TR), partSize(P,Size)} > C,
   transportCapacity(TR,C), route(From,To,TR,_,_), bins(ID).
:- #sum{N*F,ID : pack(N,P,ID,From,To,TR),
        freq(ID,From,To,TR,F)} < Value,
   flow(From,To,P,Value), Value > 0.
```

The objective minimises total transport cost, defined as the product of frequency, distance, and per-unit cost summed across all active routes:

```
#minimize{ F*Distance*Cost@1, ID, From, To, TR :
    freq(ID,From,To,TR,F),
    route(From,To,TR,Distance,Cost) }.
```

The encoding worked well on small instances but did not scale as the number of bins and network size grew. Two problems compounded each other. The first is bin symmetry. A solution assigning frequency f_1 to bin 1 and f_2 to bin 2 is structurally identical to one that swaps them. With B bins there are $B!$ such permutations per route to explore. The second is the flow support constraint. The aggregate:

```
:- #sum{N*F,ID : pack(N,P,ID,From,To,TR),
        freq(ID,From,To,TR,F)} < Value,
   flow(From,To,P,Value), Value > 0.
```

involves a product of two choice atoms, the pack and frequency values. Both are decision variables rather than ground facts; the grounder must instantiate every combination of pack and frequency values before the solver can evaluate the aggregate. This produces a large number of ground atoms and rules, so the solver cannot detect a violation until both have been assigned. This means that conflicts are found late, and each backtrack discards a long chain of decisions.

4.2. Optimised Encoding

The baseline exposed two problems. First, the pack domain was unconstrained as the solver would consider packing more units of a part into a bin than could physically fit. Second, bins were symmetric; swapping the labels produced a structurally identical solution, and the solver had no way to know this. We addressed both with four targeted changes.

Domain tightening. Rather than drawing pack values from a global domain, we precompute the maximum number of units of part p that can physically fit in a transport resource τ, and restrict choices to that range:

```
feasiblePackN(P,TR,N) :-
    numParts(N), partSize(P,S),
    transportCapacity(TR,C), N*S <= C.

```

```
1{ pack(N,P,ID,From,To,TR) : feasiblePackN(P,TR,N) }1 :-
    route(From,To,TR,_,_), bins(ID), part(P).
```

This removes atoms that the solver would otherwise branch over.

Reachability pruning and supply-based constraints. We added explicit guards to prevent packing on arcs where the supply cannot arrive. These fire during unit propagation rather than waiting for a downstream flow conservation failure:

```
:- pack(N,P,ID,From,To,TR), N > 0, is_consumer(P,From).
:- pack(N,P,ID,From,To,TR), N > 0, is_producer(P,To).
```

The effect is that conflicts are found earlier, and each backtrack discards less work than in the baseline.

Frequency-bin consistency. We added constraints to prevent 0 packed assignments from being considered in the final solution.

```
:- freq(ID,From,To,TR,0), pack(N,P,ID,From,To,TR), N > 0.
has_load(ID,From,To,TR) :- pack(N,P,ID,From,To,TR), N > 0.
:- freq(ID,From,To,TR,F), F > 0, not has_load(ID,From,To,TR).
```

Symmetry breaking. The baseline explored all $B!$ permutations of bin labels per route. We broke this by requiring that bins are used in order: if bin 2 is active, bin 1 must also be active:

```
used_bin(ID,From,To,TR) :- freq(ID,From,To,TR,F), F > 0.

:- used_bin(ID+1,From,To,TR),
   bins(ID),
   not used_bin(ID,From,To,TR).
```

Together, these changes reduce the effective frequency search space from $O(|D|^{RB})$ to $O\left(\binom{|D|+B-1}{B}^{R}\right)$, approaching a $B!$ reduction per route as the domain grows. However, with these optimizations, although useful, still faced scaling issues.

4.3. Two-Stage Decomposition

To address the scaling problems of the monolithic, we separated the problem into two stages. Stage 1 is responsible for the network configuration, deciding how much of each part flows on each arc and how many trips each transport resource makes. Stage 2 takes those decisions as fixed facts and handles the packing, with room to introduce soft constraints that push the solver toward distributions that survive disruption.

Stage 1 (Network level). A clingcon encoding fixes aggregate flows and transport frequencies. Theory atoms declare integer flow variables with linear constraints enforcing conservation and capacity. We made use of clingcon for these encodings as it scales well to an integer domain. The output is a set of `routeFreq/5` and `load/4` facts that feed Stage 2.

Stage 2 (Packing level). Given Stage 1 facts, a second ASP encoding assigns units to trips via theory atoms for per-trip loads. This indicates which parts get packed on which trips between two locations within the network.

4.4. Soft Constraint Architecture

As mentioned earlier, when examining the packages and the load assignments of parts produced by the answer sets, we observed that the solver was loading all parts onto a single transport resource or packing all parts of a single type into a single package. Although this would have been the most cost

efficient solution, it is not always the robust solution. If a single transport resource was to fail and there were no alternative part assignments to other resources, the total demand served for these parts would be drastically affected.

To introduce robustness into the packing decisions, we define three soft constraints that guide the solver toward distributions which hold up under disruption. Each one targets a different way a packing can be fragile.

Worst-case exposure. A transport resource is concentrated if it carries more than half the total supply of any part type on an arc. If that transport resource is lost, demand at the destination cannot be met, regardless of what else arrives. Thus we can define a predicate `highExposure/4` which penalises concentrated transport resource. Setting a high soft constraint weight, the solver would need to trade off against the cost of the network or distributing the load evenly.

Value-weighted concentration. Not all parts carry equal risk. Losing a cheap, easily replaced component is manageable; however, losing an expensive part can halt production. To address this we define `overloaded/4` which determines which trips have a high value ratio load.

Heterogeneity. On arcs that carry multiple part types, we prefer trips that carry a mix of parts rather than a single type. If parts are consumed together at the destination, a trip carrying only one type provides little immediate value on arrival. A mixed trip means partial production can begin straight away rather than waiting for the matching shipment. The predicate `monotrip/4` defines trips which have single loaded parts which then allows for the encodings to penalise it using soft constraints.

5. Robustness Metrics

To determine the effectiveness of our soft constraints, we define the following measurements for robustness inspired by the literature [13]. We define robustness as the fraction of total demand that can still be satisfied after a disruption. We evaluate this across two metrics.

Transport-resource robustness R_{TR} models a if all trips made by a given transport resource on a given arc fail simultaneously. This captures events such as a vehicle type being grounded or a carrier going on strike. For each such failure, we construct a surviving flow network whereby we remove the failed resource's capacity on that arc, and compute the maximum flow that can still reach all demand nodes using the formula:

$$R_{TR} = \min_{(F,T,TR)} \min_{P} \frac{\text{maxflow}\left(\mathcal{N}_P^{(F,T,TR)},\, S,\, T\right)}{\text{totalDemand}(P)}$$

Parametric trip robustness R_α generalises R_{TR} to partial disruptions, where only a fraction $\alpha \in [0, 1]$ of trips on a disrupted arc survive. Let $\mathcal{K}(from, to, \tau)$ denote the set of dispatched trips on arc $(from, to, \tau)$ with frequency $f = |\mathcal{K}(from, to, \tau)|$. Under survival fraction α, the $\lfloor \alpha \cdot f \rfloor$ heaviest-loaded trips are removed as the worst case, and $\mathcal{S}(\mathcal{K}, \alpha)$ denotes the set of surviving trips. The surviving load for part p on the disrupted arc is: $\hat{c}(from, to, \tau, p) = \sum_{k \in \mathcal{S}(\mathcal{K}(from,to,\tau),\,\alpha)} q_k(p)$ where $q_k(p)$ is the quantity of part p carried by trip k.

Single-arc scope. One arc is disrupted at a time, while all others remain fully operational. For a disrupted arc $(from^*, to^*, \tau^*)$, the residual flow network is constructed identically to R_{TR} (Section 4.1.1), with arc capacities: $c^D(from, to, \tau, p) = \begin{cases} \hat{c}(from^*, to^*, \tau^*, p) & \text{if } (from, to, \tau) = (from^*, to^*, \tau^*) \\ load(from, to, \tau, p) & \text{otherwise} \end{cases}$

The single-arc metric is the minimum coverage over all disrupted arcs and all parts:

$$R_\alpha^{\text{single}} = \min_{(from,to,\tau)\in\mathcal{A}} \min_{p\in P} \frac{\text{maxflow}\left(\mathcal{N}_p^{(from,to,\tau,\alpha)},\, S,\, T\right)}{\text{totalDemand}(p)}$$

Global scope. All arcs are disrupted simultaneously to fraction α, so only $\lfloor \alpha \cdot |\mathcal{K}(from, to, \tau)| \rfloor$ trips survive on every arc. The residual flow network uses surviving loads on every arc:

$$c^D(from, to, \tau, p) = \hat{c}(from, to, \tau, p) \qquad \forall\, (from, to, \tau) \in \mathcal{A}$$

The global metric is the minimum coverage over all parts:

$$R_\alpha^{\text{global}} = \min_{p \in P} \frac{\text{maxflow}\left(\mathcal{N}_p^{(\alpha)},\, S,\, T\right)}{\text{totalDemand}(p)}$$

Together, the two scopes satisfy $R_\alpha^{\text{single}} \geq R_\alpha^{\text{global}}$, and both recover R_{TR} when $\alpha = 0$.

6. Evaluation

6.1. Encoding Comparison

Experiments on the baseline and optimised encodings used synthetic instances using a Python random instance generator across seven series, varying locations, products, transport types, bins, and capacity tightness. All runs used clingo with a 300-second timeout across four solver configurations (jumpy, tweety, handy, parallel).

Key findings:

- **Large search space dominates.** Grounding never exceeds 17% of total runtime. Solve time grows much faster as the instance size increases.
- **Choices proxy search effort reliably.** Near-linear proportionality between choices and solve time holds across all runs. Optimised runs clustered near the origin, while baseline runs reached 3.2 million choices and 120 seconds on the hardest instances.
- **Symmetry breaking drives the speedup.** From $B = 1$ to $B = 3$, the baseline grows 35–302× while the optimised encoding grows only 7–14×.
- **Conflict quality improves.** The optimised encoding has a 2–3× lower conflict rate at $B = 3$, this signaled that adding more constraints guided the solver better, which is in line with what the ASP literature says.

7. Progress and Plan

Completed

- Baseline and optimised ASP encodings with correctness verification.
- Full search space analysis, including formal reduction factors.
- Experimental evaluation across seven instance series and four solver configurations.
- Stage 1 clingcon encoding with frequency and load output.
- Stage 2 ASP encoding with hard and soft diversification constraints.
- Four robustness metrics defined.

Remaining

- **Systematic experimental running** of the full two-stage pipeline across 15 instances varying in size.
- **Evaluation** of the results and write-ups.

8. Related Work

The multi-commodity flow problem has been studied extensively via LP/MIP formulations [8, 9]. Those solvers will likely outperformASP on pure cost minimisation. The case for ASP lies in the modelling flexibility it provides for the packing layer and the non-monotonic extensions. Supply chain robustness is reviewed by Hosseini et al. [2]. The packing-level diversification approach is novel in that literature, which focused on inventory positioning and supplier diversification rather than trip-level load assignment.

9. Conclusion

This research examines ASP as a framework for a logistics problem in which the aim is not merely to minimise cost but to build robustness directly into the plan. The work produced a baseline and optimised encoding, along with a two-stage decomposition that separates flow planning from packing diversification. Supply chains frequently face unpredictable, often unavoidable, disturbances, making a cost-optimal plan alone vulnerable to unforeseen circumstances. ASP addresses this directly by providing a modelling language in which robustness can be enforced through soft constraints, making it a particularly attractive tool for incorporating these features into the network. The remaining work is to close the gap between our hypothesis and preliminary results to a full experimental understanding of this, which will, in fact, push the network in the direction we hope.

Declaration on Generative AI

Generative AI (Claude Sonnet 4.6) was used to provide a skeleton structure of the paper and to generate boiler content, which the authors then reformatted and refined. All conceptual, technical, and experimental contributions are the author's own.

References

[1] T. Flamand, M. Iori, and M. Haouari. The transportation problem with packing constraints. *Computers & Operations Research*, 157:106278, 2023.

[2] S. Hosseini, D. Ivanov, and A. Dolgui. Review of quantitative methods for supply chain robustness analysis. *Transportation Research Part E*, 125:285–307, 2019.

[3] A. Dolgui, D. Ivanov, and B. Sokolov. Ripple effect in the supply chain: an analysis and recent literature. *International Journal of Production Research*, 56:1–17, 2017.

[4] J. B. Rice and F. Caniato. Building a secure and resilient supply network. *Supply Chain Management Review*, 7(5):22–30, 2003.

[5] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. ICLP/SLP*, pages 1070–1080, 1988.

[6] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. Multi-shot ASP solving with clingo. *Theory and Practice of Logic Programming*, 19(1):27–82, 2019.

[7] M. Banbara, B. Kaufmann, M. Ostrowski, and T. Schaub. Clingcon: The next generation. *Theory and Practice of Logic Programming*, 17(4):408–461, 2017.

[8] F. L. Hitchcock. The distribution of a product from several sources to numerous localities. *Journal of Mathematics and Physics*, 20(1–4):224–230, 1941.

[9] T. C. Koopmans. Optimum utilization of the transportation system. *Econometrica*, 17:136, 1949.

[10] Y. Guo, F. Liu, J.-S. Song, and S. Wang. Supply chain resilience: A review from the inventory management perspective. *Fundamental Research*, 5(2):450–463, 2024.

[11] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[12] Scaling Industrial Logistics: Tackling Multi-Batching Problems via Sequential Solving. Submitted

[13] B. Adenso-Díaz, J. Mar-Ortiz, and S. Lozano. Assessing supply chain robustness to links failure. *International Journal of Production Research*, 56(15):5104–5117, 2018.