A Supplemental Information for "NODE"

A.1 Notation

Table 3: Notation.

Symbol	Meaning	Units
c_{od}	generalised cost from o to d	time, money, or index
$F(c)$	acceptance cumulative distribution function by cost c	probability
$f(c)$	acceptance probability distribution function by cost c	probability density
$G = \{V, E\}$	network graph	–
J_d	jobs at destination d	persons
$j_{d,k}$	residual jobs at d in step k	persons
k	step index	–
O, D	sets of origins and destinations	–
o, d	indices of origins and destinations	–
P	stationary movement kernel (when time-invariant)	probability
P_k	row-stochastic movement kernel at step k	probability
t	scenario/time index (superscript (t) denotes scenario)	–
Δt	time quantum between adjacent cells	time
W_o	workers at origin o	persons
$w_{o,k}$	residual workers at o in step k	persons
$X_{od,k}$	cumulative flow from o to d in step k	persons
$x_{od,k}$	accepted (incremental) flow from o to d in step k	persons
$\Delta X^{(t)}$	change in flows between scenarios $t-1$ and t	persons
λ	acceptance hazard	per unit cost
σ	per-step acceptance $(1 - e^{-\lambda \Delta t})$	probability
$\sigma_{d,k}$	per-step acceptance at destination d in step k	probability

A.2 Counterexample: cost-ordered allocation is not a general optimiser

Two origins, two destinations, unit supplies: $W_{O1} = W_{O2} = 1$, $J_{D1} = J_{D2} = 1$. Costs:

	$D1$	$D2$
$O1$	1.00	1.01
$O2$	1.01	100

Cost ordered allocation (the greedy algorithm) assigns $O1 \to D1$ (cost 1.00), then must send $O2 \to D2$ (cost 100), total 101.00. The min-cost-flow solution is $O1 \to D2$ and $O2 \to D1$ with total $1.01 + 1.01 = 2.02$. Hence cost-ordered differs from the global optimum. □

A.3 When does cost-ordered match the transport optimum?

Proposition A.1 (Cost-ordered optimality under Monge)**.** *Consider the Hitchcock–Koopmans problem with supplies W_o, demands J_d, and costs c_{od}. If the cost array is Monge, i.e.,*

$$c_{i+1,j+1} + c_{i,j} \leq c_{i+1,j} + c_{i,j+1} \quad \forall i, j, \tag{3}$$

then the cost-ordered allocation that processes (o, d) in non-decreasing c_{od} (with a fixed, reproducible tie-break) attains an optimal solution.

Sketch. Under the Monge property, the transport polytope is totally monotone and admits northwest-corner/cost-ordered constructions that are optimal; see Burkard et al. (1996). The NODE sweep respects this order and, because updates preserve feasibility and complementary slackness under Monge structure, the resulting plan minimises total cost. A counterexample in Appendix A.2 shows the property is sufficient for cost-ordered optimality. □

With shortest-path costs c_{od}, the global benchmark is the Hitchcock–Koopmans transport problem (a min-cost flow on the OD bipartite graph):

$$\min_{X \geq 0} \sum_{o \in O} \sum_{d \in D} c_{od} X_{od} \quad \text{s.t.} \quad \sum_{d} X_{od} = W_o, \quad \sum_{o} X_{od} = J_d. \tag{4}$$

NODE processes (o, d) in nondecreasing c_{od} and assigns until capacities bind. This coincides with the optimum of (4) only under special cost structure (e.g., Monge/convex arrays, or shared distance orderings yielding total monotonicity); otherwise NODE is a transparent least-cost *heuristic* that exposes local capacity competition (see Appendix A.2).

A.4 Proofs and unit–vs–batch equivalence

Lemma S2.1 (Termination). The algorithm terminates in a finite number of steps.
Proof. Each processed event either reduces $\sum_o w_o$ or is skipped because $w_o = 0$ or $j_d = 0$. There are at most $|O||D|$ events and at most $\sum_o W_o$ positive assignments. With nonnegative link costs and a fixed tie-break, the event order is finite, hence termination. □

Lemma S2.2 (Mass preservation). If $\sum_o W_o = \sum_d J_d$ and all pairs are mutually reachable, then $\sum_{od} X_{od} = \sum_o W_o = \sum_d J_d$.
Proof. Each positive assignment subtracts the same δ from w_o and j_d, and adds it to X_{od}. When the algorithm halts with all $w_o = 0$, column residuals must also be zero by conservation, so $\sum_{od} X_{od}$ equals the common total. □

Lemma S2.3 (Integrality). If W_o and J_d are integers for all o, d, the deterministic event-order algorithm yields an integral allocation matrix X. *Proof.* Each assignment updates $X_{od} \leftarrow X_{od} + \delta$ with $\delta = \min\{w_o, j_d\}$, which is integer when w_o, j_d are integers. All updates preserve integrality, so X is integral at termination. □

A.5 Pseudocode (priority-queue implementation (frontier-based version))

Inputs. Network $G = (V, E)$ with nonnegative link costs; origin set O with workers W_o; destination set D with jobs J_d; shortest-path cost function c_{od}; fixed tie-break on (c, o, d).
Outputs. Allocation matrix $X \in \mathbb{R}_+^{|O| \times |D|}$; residual vectors w, j.

1. Initialise $w \leftarrow W$, $j \leftarrow J$, $X \leftarrow 0$.

2. For each $o \in O$, start a Dijkstra-like expansion with source o. Maintain a min-heap keyed by tentative cost. When a destination d is first settled for o, append the event (c_{od}, o, d) to a global event stream.

3. Process events from the stream in non-decreasing order of (c, o, d). For each event:

   (a) If $w_o = 0$ or $j_d = 0$, *skip*.

   (b) Let $\delta = \min\{w_o, j_d\}$. Set $X_{od} \leftarrow X_{od} + \delta$, $w_o \leftarrow w_o - \delta$, $j_d \leftarrow j_d - \delta$.

4. Stop when all $w_o = 0$ or no admissible pairs remain. Report residuals if $\sum_o W_o \neq \sum_d J_d$.

A.6 Complexity and implementation notes

- **Frontier (recommended).** Run Dijkstra's algorithm from each origin (multi-source); record each destination once per origin. Complexity dominated by shortest paths, $O((|E| + |V| \log |V|)|O|)$ on sparse graphs.

- **All-pairs + sort (didactic).** Generate all (o, d) pairs, sort by c_{od}, then sweep. Work $O(|O||D| \log(|O||D|))$ may dominate.

- **Reproducibility.** Fix lexicographic tie-break on (c, o, d); fix random-number seed for stochastic runs.

- **Disconnected pairs and imbalance.** Ignore unreachable origin-destination pairs. If $\sum_o W_o \neq \sum_d J_d$, stop and report residuals.

A.7 Stochastic acceptance and the gravity link

Let the cumulative acceptance be $p(c) = 1 - \exp\{-\int_0^c \lambda(s)ds\}$ with hazard $\lambda(c) \geq 0$. The acceptance *density* is $f(c) = \lambda(c) \exp\{-\int_0^c \lambda(s)ds\}$. With constant λ, $p(c) = 1 - e^{-\lambda c}$ and $f(c) = \lambda e^{-\lambda c}$.

Expected flows without capacity limits. Consider one origin o with W_o workers and destinations $d \in D$ with job counts J_d. Treat each job as a competitor in an "exponential race": destination-d jobs draw i.i.d. $E \sim \text{Exp}(\lambda)$ and reveal arrival times $T_{d,k} = c_{od} + E_{d,k}$. The worker goes to the job with the minimum $T_{d,k}$. For any fixed o, the probability that the winner lies at destination d is

$$\Pr\{\text{winner at } d\} = \frac{J_d e^{-\lambda c_{od}}}{\sum_{d'} J_{d'} e^{-\lambda c_{od'}}}. \tag{5}$$

Sketch. For shifted exponentials $c_i + E_i$, $\Pr\{\arg\min i\} \propto e^{-\lambda c_i}$; aggregating J_d i.i.d. jobs at d multiplies the weight by J_d. Thus the expected allocation from o is W_o times (5), i.e., the singly-constrained exponential-gravity form. With nonconstant $\lambda(c)$, replace $e^{-\lambda c}$ by $\exp\{-\int_0^c \lambda(s)ds\}$.

With capacity limits. When J_d bind, the process becomes a sequence of races with shrinking sets; the expectation still prefers low c_{od}, but the closed form (5) no longer holds. NODE keeps the frontier and capacities explicit, which is the point of using it.

Define survival $S(c) = e^{-\lambda c}$ and hazard $h(c) = \lambda$ for $c > 0$, so $p(c) = 1 - S(c)$. Embedding this acceptance in the deterministic event order yields a stochastic variant of NODE; the main text analyses the deterministic limit.

A.8 Worked stochastic realisation for the toy network

For $\lambda = 0.35$ and a fixed RNG seed, one draw (illustrative) produced:

	D1	D2	D3	Total from origin
O1	1	1	2	4
O2	3	2	0	5
Total to dest.	4	3	2	9

This preserves workers and jobs, but differs from the deterministic table in the main text.

Data and code. Minimal code to reproduce the figure, the deterministic table, and a stochastic draw is included at the end of the SI. Tie-breaks and seeds are set in the script for full reproducibility.

A.9 Matrix form and symbols

Let $w^{(t)} \in \mathbb{R}_+^{|O|}$, $j^{(t)} \in \mathbb{R}_+^{|D|}$ be residuals after step t, and M be a binary indicator matrix (mask) of admissible pairs. At event (o, d), the update is

$$X_{od}^{(t+1)} = X_{od}^{(t)} + \delta^{(t)}, \quad \delta^{(t)} = \min\{w_o^{(t)}, j_d^{(t)}\}, \tag{6}$$

with component-wise reductions to $w^{(t+1)}$, $j^{(t+1)}$. Shapes are explicit and the min is component-wise; masked pairs have $\delta^{(t)} = 0$.

Let $\mathbb{1}_{\mathcal{F}^{(t)}}$ be a binary mask of OD pairs discovered at step t.

$$M^{(t)} = \min\left[\mathbb{1}_{\mathcal{F}^{(t)}} \odot \operatorname{diag}(\mathbf{w}^{(t)})\mathbf{1}^\top, \mathbf{1}(\mathbf{j}^{(t)})^\top\right] \tag{7}$$

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - M^{(t)}\mathbf{1}, \qquad \mathbf{j}^{(t+1)} = \mathbf{j}^{(t)} - M^{(t)\top}\mathbf{1} \tag{8}$$

$$T = \sum_t M^{(t)}. \tag{9}$$

Note: $\odot$ denotes element-wise multiplication; diag forms a diagonal matrix.

A.10 CTM realisation of NODE

NODE can be implemented in discrete time on a time-expanded network. To do this, partition space so that adjacent cells are separated by a fixed travel-time quantum Δt, or alternatively

use a time-expanded graph whose edges represent Δt of travel. At each step k, worker mass in a cell moves to Δt-adjacent cells according to a row-stochastic movement kernel P_k (reflecting network topology and impedance), encounters the cell's remaining jobs $j_{d,k}$, and accepts with per-step probability $\sigma_{d,k}$, drawing down $j_{d,k}$. Here o and d index cells in the time-expanded network, with O and D meaning 'worker-holding cells' and 'job-holding cells'; similarly the variable k is remapped to index time steps, rather than assignment events. This can be thought of as an agent-based intervening opportunities model.

With a constant hazard λ the discrete acceptance is $\sigma = 1 - e^{-\lambda \Delta t}$. If capacities do not bind and P_k is stationary in k (i.e., $P_k = P$), the expected first-acceptance time is geometric, yielding an exponential deterrence in travel time and reproducing the origin-constrained gravity form. Binding capacities, heterogeneous hazards, or state-dependent P_k recover NODE's competitive allocation and its deviations from gravity. Capacity is enforced at the destination level before allocating to origins. This is shown in Algorithm 2. We use the shorthand $\sum_O$ to denote $\sum_{o \in O}$, and similarly $\sum_D$ for $\sum_{d \in D}$.

Algorithm 2 CTM–NODE (move then accept; summation shorthand)

Inputs: Δt, $w_{o,0} = W_o$, $j_{d,0} = J_d$, $X_{od,0} = 0$, P_k, $\sigma_{d,k}$
for $k = 1, 2, \ldots$ **do**
  for all $d \in D$ **do** ▷ accept only from mass that arrives at d this step
    if $\sigma_{d,k} \sum_O w_{o,k-1}(P_k)_{od} \le j_{d,k-1}$ **then**
      for all $o \in O$ **do**
        $x_{od,k} \leftarrow \sigma_{d,k} w_{o,k-1}(P_k)_{od}$
      end for
    else ▷ Capacity binds
      for all $o \in O$ **do**
        $x_{od,k} \leftarrow j_{d,k-1} \dfrac{w_{o,k-1}(P_k)_{od}}{\sum_O w_{k-1}(P_k)_d}$
      end for
    end if
    $j_{d,k} \leftarrow j_{d,k-1} - \sum_O x_{od,k}$
    for all $o \in O$ **do**
      $X_{od,k} \leftarrow X_{od,k-1} + x_{od,k}$
    end for
  end for
  for all $o \in O$ **do** ▷ Workers update, subtract acceptances into cell o
    $w_{o,k} = \sum_i w_{i,k-1}(P_k)_{io} - \sum_d x_{od,k}$
  end for
  Stop if $\sum_O w_{o,k} = 0$ or $\sum_D j_{d,k} = 0$
end for

A.11 Code: A short, reproducible implementation for the Toy Network

Listing 1: Minimal NODE implementation for the toy network.

```
def node(W, J, C):
    # W: dict o->int, J: dict d->int, C: dict (o,d)->cost or inf
    from math import inf
    X = {(o, d): 0 for (o, d) in C if C[(o, d)] < inf}
    w = W.copy(); j = J.copy()
    events = sorted([(C[(o, d)], o, d) for (o, d) in X.keys()])
    for _, o, d in events:
        m = min(w[o], j[d])
        if m > 0:
            X[(o, d)] += m
            w[o] -= m
            j[d] -= m
    return X, w, j
```