Union-of-predecessors bounds and a forward–backward local search for multi-door cross-dock truck scheduling

Felipe Provezano Coutinho*

Independent researcher

Abstract

In a cross-docking terminal, an outbound truck can start loading only after every inbound truck that carries its goods has been unloaded. We study the makespan of this problem with several identical doors on each side. Earlier lower bounds reason about one truck at a time. We propose the union-of-predecessors bound (UPB). For each time t, it bounds the outbound work that can have started by t. This bound is a knapsack problem over the union of the trucks' inbound predecessors. We prove that the bound is valid and give an exact finite form. On the 10,500 benchmark instances it is on average 15.8% above the best earlier bound, and 3.4% above it on preregistered sparse and block-structured instances. The bound is not tight: on 165 proven optima it lies 3.6–6.6% below the optimum. A second bound, which infers time windows, reduces this distance from 5.5% to 1.4% on 98 optima. Written as cuts in a constraint-programming solver, the knapsack argument proves optimality on 137 of 300 new instances within 300 s, against 47 without the cuts, in a preregistered test. Certified schedules come from a forward–backward iterated local search. In a preregistered test it improves on a published evolutionary algorithm by 0.63% at equal numbers of schedule evaluations, and by 1.09% at equal computing time; this second comparison pits our compiled code against the rival's interpreted code.

Keywords: Scheduling, Cross-docking, Lower bounds, Partially ordered knapsack, Constraint programming

1. Introduction

A cross-docking terminal transfers goods from inbound trucks to outbound trucks with little or no storage in between. Inbound trucks are unloaded at receiving doors, the goods are sorted, and outbound trucks are loaded at shipping doors. In the problem that we study, every outbound truck needs goods from a known set of inbound trucks, and it can start loading only after all of these inbound trucks have been unloaded. Both sides have several identical doors. The goal is to finish all trucks as early as possible. In scheduling notation the problem is $P|M_j, prec|C_{\max}$ (Nogueira et al., 2020).

The problem has a public benchmark and a line of methods built on it. Chen and Song (2009) defined the two-stage problem with several doors and gave a lower bound and constructive heuristics. Cota et al. (2016) published a time-indexed formulation and a set of 10,500 instances.

*Corresponding author. ORCID: 0000-0001-7206-5856

Email address: fpcoutinho@alumni.usp.br (Felipe Provezano Coutinho)

Fonseca et al. (2019) proposed a hybrid Lagrangian metaheuristic (HgR) and published its per-instance results. Nogueira et al. (2020) modelled the problem as parallel-machine scheduling and proposed the PCH heuristic and bound. Nogueira et al. (2025) proposed the evolutionary algorithm EACDC and its machine-learning-tuned version EACDCml. This paper extends that line. It uses the same benchmark, the same list decoder and the public code and results of the earlier work. On top of them it adds a lower bound, cuts for exact solution and a certificate for every schedule.

Reported optimality gaps on the benchmark lie between about 23% and 30%, depending on the method and on how the gap is defined (Fonseca et al., 2019; Nogueira et al., 2020, 2025). Such a gap has two possible sources: the heuristic may be far from the optimum, or the lower bound may be far below it. The published gaps cannot separate the two, because the bounds used on the benchmark reason about one truck at a time. The bound of Chen and Song (2009) adds the smallest release times of the outbound trucks to the outbound workload. The PCH bound adds to an inbound workload the outbound work that a single inbound truck releases. Neither bound uses the fact that one inbound truck feeds many outbound trucks, while an outbound truck waits for several inbound trucks at once.

Our bound uses exactly this fact. Pick any time t. The outbound trucks that have started by t must have all of their inbound trucks finished by t. The inbound doors can complete at most $m_1 t$ units of work by then. So the set of outbound trucks that can have started is limited by the total work of the *union* of their predecessor sets. The largest outbound workload that can have started is the value of a partially ordered knapsack (Kolliopoulos and Steiner, 2007). Whatever is left must be processed after t on the m_2 outbound doors. Taking the best t gives the union term LB_U. For one door on each side, Prot and Rapine (2013) use this inequality at a single time as a step inside an approximation proof. We turn it into a lower bound for any number of doors and all times and make it computable. The union-of-predecessors bound (UPB) is the maximum of LB_U and the classical bounds.

The paper makes four contributions.

1. *A lower bound and its theory.* We prove that LB_U is valid and give an exact form that needs the knapsack only at its breakpoints. With a door-start cap, the union term together with a trivial per-truck bound is never below the bound of Chen and Song (2009), so UPB contains that bound. The knapsack subproblem is strongly NP-hard, as Johnson and Niemi (1983) showed for partially ordered knapsacks. The list decoder used by all methods on the benchmark is within the classical typed-task constant $3-1/\max(m_1, m_2)$ of UPB (Jaffe, 1980; Liu and Liu, 1978), so the ratio of a schedule to the bound is an instance-level certificate.

2. *A census of the benchmark.* On all 10,500 instances, UPB is on average 15.76% above the best earlier bound, and a preregistered test on 210 new instances confirms a gain of 14.90%. The gap of the published HgR solutions falls from 26.1% to 9.1%. On average the bound is also above a two-stage energetic-reasoning bound. It is above the LP relaxation of the published time-indexed model wherever we could solve that LP. Even with strengthened makespan linking, the LP stays below the bound on 45 of 50 instances with $n_1 = 20$, is equal on 3 and above on 2, and it is about 5×10^4 times slower. The bound is also above the

published means of the Lagrangian bound on the single-door instances (one door per side). Exact optima show that it is not tight.

3. *A stronger destructive bound.* UPB-W tests trial makespans by inferring a time window for every truck, with union-based tests that couple the two stages. On 98 proven optima it lowers the mean distance to the optimum from 5.48% to 1.43%.

4. *Cuts for exact solution.* The knapsack argument can be written as valid cuts for a constraint-programming model. In CP-SAT these cuts prove optimality on 137 of 300 new instances within 300 s in a preregistered test, against 47 for the same model started from the bound alone. On the instances used to develop them the counts are 161 and 59; the 161 include 47 optima with 30 inbound trucks, where no variant without the cuts proved any.

To certify schedules and to improve the published upper bounds, we use a simple iterated local search with forward–backward passes (FB-ILS). It is an extension of the EACDC line rather than a contribution of its own, and we test it against EACDC, EACDCml and HgR at matched effort. At equal numbers of schedule evaluations, which do not depend on the implementation, it improves on EACDC by 0.63% in a preregistered test. It improves the published per-instance HgR upper bound on 10,475 of the 10,500 instances; these are the only per-instance upper bounds published for the benchmark. The bound has limits too: on the preregistered structured instances, which are sparser or block-structured, its gain is much smaller (3.4%). Section 9 states these and the other limits of the study.

2. Related work

Lower bounds for two-stage, hybrid flow shop and assembly problems. Most bounds for two-stage and hybrid flow shops relax each stage to a parallel-machine problem with heads and tails. Carlier and Néron (2000) bound a stage by the m smallest heads, the total work and the m smallest tails, divided by m. Haouari et al. (2006) and Hidri and Haouari (2011) build branch-and-bound methods and families of bounds on this principle. Energetic reasoning goes back to the window bounds of Fernandez and Bussell (1973). Néron et al. (2001) apply it to hybrid flow shops: each operation receives a time window, and the method checks whether the work that must fall inside each interval fits on the machines. Wallrath et al. (2025) derive valid inequalities that link the makespan to job start times and tighten the LP relaxations of flow shop models. In all of these models each operation has exactly one predecessor, the previous operation of the same job, so windows can be set job by job. In two-stage assembly scheduling a final operation waits for several components, but each job has its own components (Potts et al., 1995; Hariri and Potts, 1997). Lee et al. (2025) study a similar two-stage structure in sheet-metal production and name a lower bound as future work. In a cross-dock, inbound trucks are shared: one inbound truck feeds several outbound trucks, and an outbound truck waits for several inbound trucks.

Our bound uses this coupling. The set of outbound trucks that can have started by time t is chosen by a bipartite partially ordered knapsack (Kolliopoulos and Steiner, 2007) over the union of their predecessors. For one door on each side, Prot and Rapine (2013) use this knapsack at

one time point inside an approximation proof, which gives OPT $\geq p(Y) + t - p(U^*(t))$ in their notation. We generalise the inequality to any number of doors and all times, add a door-start cap and the reversed instance, prove a finite exact form and compute the bound on a benchmark. Destructive lower bounds, which fix a trial makespan and refute it by propagation and feasibility tests, are standard in project scheduling (Klein and Scholl, 1999; Brucker and Knust, 2000). Our second bound, UPB-W, instantiates this scheme for the cross-dock (Section 4.6). For a single receiving and a single shipping door, with product quantities instead of fixed predecessor sets and with temporary storage, Golshahi-Roudbaneh et al. (2017) proposed a lower bound from the total workload plus one changeover delay and used it as a gap reference for their metaheuristics. With parallel doors on both sides, and outbound trucks that wait for fixed sets of inbound trucks, a different argument is needed. We provide it: the union-of-predecessors bound, its exact breakpoint form, and its dominance over the bound of Chen and Song (2009).

List scheduling for typed task systems. Graham (1969) showed that list scheduling is within $2 - 1/m$ of the optimum on m identical processors. Liu and Liu (1978) and Jaffe (1980) extended the analysis to typed task systems with k processor types and precedence constraints: every list schedule is within $k + 1 - 1/\max_i m_i$ of the optimum, which is best possible for general precedence graphs. Liu et al. (1981) prove the constant against per-type workloads and chain lengths. The cross-dock problem is a typed task system with $k = 2$, so the constant $3 - 1/\max(m_1, m_2)$ applies to it. We use this classical result to turn UPB into an instance-level certificate (Section 4.5).

Makespan methods for cross-docks. For one door on each side, Chen and Lee (2009) proved strong NP-hardness and gave a branch-and-bound method, and Prot and Rapine (2013) studied approximation algorithms. Boysen et al. (2010) studied a single-door problem in which goods are described by product quantities, and Ladier and Alpan (2016) compare cross-dock research with industry practice. For several doors, Chen and Song (2009) introduced the two-stage problem, a bound from the smallest outbound heads applied to the instance and its time-reversed copy, and heuristics based on Johnson's rule. Cota et al. (2016) proposed the time-indexed formulation, the benchmark and the CDH heuristic, with an additive worst-case bound for CDH (their Theorem 3). Fonseca et al. (2019) developed HgR, a Lagrangian bound for the case with one door per side, and used the bound of Chen and Song (2009) for several doors. Nogueira et al. (2020) introduced the PCH heuristic, the PCH bound and the list decoder that later methods use, and also bounded the additive error of their heuristic (PCH). Nogueira et al. (2025) embedded the decoder in a $(\mu + \lambda)$ evolutionary algorithm (EACDC) and tuned its parameters per instance with a neural network (EACDCml). Other cross-dock papers add door assignment, mixed-mode doors, workforce, storage, product flows or cost objectives (Gaudioso et al., 2021; Neamatian Monemi and Gelareh, 2023; Rijal et al., 2019; Tadumadze et al., 2019; Li et al., 2026c; Ojha and Erera, 2025; Eti et al., 2025; Li et al., 2026a,b); some solve them exactly by branch-and-price (Neamatian Monemi and Gelareh, 2023) or dynamic discretization discovery (Ojha and Erera, 2025). None of them uses the Cota benchmark or a union-of-predecessors bound.

Upper-bound methods. Population-based methods dominate cross-dock truck scheduling, for example memetic algorithms (Golias et al., 2013; Dulebenets, 2021), the evolutionary algorithm of Nogueira et al. (2025) and random-key genetic algorithms with multi-parent crossover (Andrade et al., 2021). Our upper-bound method is deliberately simpler: an iterated local search over the order of the inbound trucks. Simple single-trajectory methods of this kind, such as the iterated greedy algorithm of Ruiz and Stützle (2007), are known to be effective for flow shops. Our method is an iterated local search in the sense of Lourenço et al. (2003). Its forward–backward step follows the justification principle of project scheduling (Li and Willis, 1992; Valls et al., 2005), which we transfer to the two-stage cross-dock decoder (Section 5).

3. Problem definition

Trucks, doors and precedence. A set J_1 of n_1 inbound trucks is unloaded on m_1 identical inbound doors, and a set J_2 of n_2 outbound trucks is loaded on m_2 identical outbound doors. Inbound truck i takes p_{1i} time units and outbound truck j takes p_{2j} time units. All processing times are positive integers, all trucks are available at time zero, and a truck occupies one door without interruption. Each outbound truck j has a set $N(j) \subseteq J_1$ of inbound predecessors: the trucks that carry goods for j. Truck j may start only after every $i \in N(j)$ has been completed. For a set $A \subseteq J_2$ we write $N(A) = \bigcup_{j \in A} N(j)$, and for an inbound truck i we write $S(i) = \{j \in J_2 : i \in N(j)\}$ for its successors. The makespan $C_{\max}$ is the completion time of the last truck, inbound or outbound. The objective is to minimise it. Following Nogueira et al. (2020), we denote the problem by $P|M_j, prec|C_{\max}$. It is strongly NP-hard already with one door on each side (Chen and Lee, 2009).

Aggregate quantities. For a set X of jobs, $p_1(X)$ and $p_2(X)$ denote total processing times. We use

$$P_1 = p_1(J_1), \qquad P_2 = p_2(J_2), \qquad a = \frac{P_1}{m_1}, \qquad b = \frac{P_2}{m_2},$$
$$p = \max_{i \in J_1} p_{1i}, \qquad q = \max_{j \in J_2} p_{2j}, \qquad M = \max(m_1, m_2).$$

The *head* of outbound truck j is

$$h_j = \max\Big(\max_{i \in N(j)} p_{1i},\ \frac{p_1(N(j))}{m_1}\Big), \qquad h_j = 0 \text{ if } N(j) = \emptyset.$$

No schedule can start j before h_j: its longest predecessor must be finished, and the work of all its predecessors must fit on m_1 doors. We write $r_{\min} = \min_j h_j$.

The benchmark. The benchmark of Cota et al. (2016) has 35 cells of 300 instances: seven sizes $n_1 \in \{20, 30, \ldots, 80\}$ times five door groups. In the groups $m = 2$, $m = 4$ and $m = 10$, both sides have that number of doors. In the groups U(2,4) and U(2,10), m_1 and m_2 are drawn independently and uniformly from these ranges. Processing times are integers drawn uniformly from $[10, 100]$, n_2 is drawn uniformly between $0.8n_1$ and $1.2n_1$, and each inbound–outbound pair is a predecessor pair with probability one half. Every outbound truck has at least one

predecessor and every inbound truck at least one successor. We checked these rules against the 10,500 files before generating new instances with them (Supplement S11).

Time reversal. Chen and Song (2009) observed that the problem is symmetric in time. The reversed instance $R(I)$ swaps the two sides. Its first stage consists of the outbound trucks on m_2 doors, and its second stage of the inbound trucks on m_1 doors. Inbound truck i may start only after every outbound truck in $S(i)$ has been completed.

Proposition 1 (Time reversal)**.** $\mathrm{OPT}(R(I)) = \mathrm{OPT}(I)$*. Hence, for any valid lower bound* L*,* $\max\big(L(I), L(R(I))\big)$ *is a valid lower bound for* I*.*

Proof. Let S be a feasible schedule of I with makespan C, and give every truck x the new start time $s'_x = C - s_x - p_x$. Each door processes the same trucks, mirrored in time, so no door processes two trucks at once. All new start times are non-negative because $s_x + p_x \le C$. For a pair with $i \in N(j)$, feasibility in I gives $s_j \ge s_i + p_{1i}$. Hence $s'_i = C - s_i - p_{1i} \ge C - s_j = s'_j + p_{2j}$, which is the precedence constraint of $R(I)$. The new makespan is $\max_x(C - s_x) \le C$. Therefore $\mathrm{OPT}(R(I)) \le \mathrm{OPT}(I)$. Since $R(R(I)) = I$, the reverse inequality also holds. □

The list decoder. PCH builds its second schedule, and EACDC all of its schedules, with this decoder (Nogueira et al., 2020, 2025); all heuristics in this paper use it too. For one door on each side, the decoder is optimal for a fixed inbound sequence (Prot and Rapine, 2013, Remark 1). It receives an order π of the inbound trucks.

1. *Stage 1.* Take the inbound trucks in the order π and put each one on the inbound door that becomes free first. This gives completion times C_{1i}.
2. *Release times.* Each outbound truck j is released at $r_j(\pi) = \max_{i \in N(j)} C_{1i}$.
3. *Stage 2.* Repeatedly take the outbound door that becomes free first, at time τ. Among the outbound trucks with $r_j(\pi) \le \tau$, start the longest one (ties by index). If no truck is released, move τ to the next release time.

We write $\mathrm{UB}(\pi)$ for the makespan of the resulting schedule. Stage 2 is *non-idle*: an outbound door is never idle while a released outbound truck waits. The certificate of Section 4.5 uses only this property and the list rule of stage 1. The public code of EACDC (Nogueira et al., 2022) evolves an order of the outbound trucks, derives the inbound order from it and uses the same stage-2 rule.

4. The union-of-predecessors lower bound

This section first derives the union term LB_U from what every schedule must satisfy at a fixed time t (Section 4.1). It then shows that LB_U needs the knapsack only at finitely many capacities (Section 4.2), defines the bound UPB that we compute (Section 4.3) and explains how we compute it (Section 4.4). Section 4.5 turns the bound into a certificate for schedules, and Section 4.6 adds a second, stronger bound that infers time windows.

4.1. The union term

The bound rests on one function. For a capacity $c \geq 0$, let

$$K(c) = \max\bigl\{p_2(A) : A \subseteq J_2,\ p_1(N(A)) \leq c\bigr\}. \tag{1}$$

In words, $K(c)$ is the largest outbound workload whose predecessors, taken together, need at most c units of inbound work. Choosing an outbound truck forces the choice of all its predecessors, and a predecessor shared by several chosen trucks is paid for only once. This is the partially ordered knapsack problem of Kolliopoulos and Steiner (2007), restricted to a two-level order. We also write $K(c^-) = \max\{p_2(A) : p_1(N(A)) < c\}$.

Fix a time t. By time t the inbound doors have done at most $m_1 t$ units of work. An outbound door cannot work before the head of its first truck. So the outbound work done by t is at most

$$\overline{W}(t) = \min\bigl(K(m_1 t),\ D(t)\bigr), \qquad D(t) = \sum_{k=1}^{\min(m_2,n_2)} (t - h_{(k)})^+,$$

where $x^+ = \max(x, 0)$ and $h_{(k)}$ is the k-th smallest head. The rest of the outbound work, at least $P_2 - \overline{W}(t)$, must be done after t on m_2 doors. This gives the candidate value $f(t) = t + (P_2 - \overline{W}(t))/m_2$. For $m_1 = m_2 = 1$ and a single t, $f(t)$ is the inequality of Prot and Rapine (2013, Lemma 3).

Lemma 1. *Let S be a feasible schedule with makespan C, let $t \geq 0$, and let $A_t = \{j \in J_2 : s_j < t\}$ be the outbound trucks that start before t. Then:*

(i) $p_1(N(A_t)) \leq m_1 t$ and $p_{1i} \leq t$ for every $i \in N(A_t)$;

(ii) the outbound work processed in $[0, t]$ is at most $\min\bigl(p_2(A_t),\ D(t)\bigr)$;

(iii) $C \geq t + p_{2j}$ for every $j \notin A_t$.

Proof. (i) Every $i \in N(A_t)$ finishes before some $j \in A_t$ starts, hence before t, so this work fits in $[0, t]$ on m_1 doors. (ii) Only trucks in A_t can work before t. No outbound truck starts before its head, and different doors have different first trucks, so the doors together work at most $D(t)$ before t. (iii) A truck that starts at t or later finishes at $t + p_{2j}$ or later. □

Definition 1. $\mathrm{LB}_U = \sup\{f(t) : t \geq 0,\ \overline{W}(t) < P_2\}$.

The condition $\overline{W}(t) < P_2$ says that some outbound work certainly remains after t.

Theorem 1 (Validity)**.** *For every t with $\overline{W}(t) < P_2$, $\mathrm{OPT} \geq f(t)$. Hence $\mathrm{LB}_U \leq \mathrm{OPT}$, and $\lceil \mathrm{LB}_U \rceil \leq \mathrm{OPT}$ for integer data.*

Proof. Let S be optimal with makespan C. If $C < t$, every outbound truck starts and finishes before t, so Lemma 1(i)–(ii) give $K(m_1 t) = P_2$ and $D(t) \geq P_2$, a contradiction. So $C \geq t$. The outbound work done in $[0, t]$ is at most $\overline{W}(t)$, and the rest is processed in $[t, C]$ on m_2 doors, so $C \geq f(t)$. □

Remark 1. f is non-increasing in K. Replacing K by any $\overline{K} \geq K$, for example the dual bound of an interrupted knapsack solve, keeps the bound valid. We use this when a knapsack is not solved to optimality.

4.2. An exact finite form

The supremum runs over a continuum of times, but only the capacities at which K jumps matter. Let $\Lambda = \{p_1(N(A)) : A \subseteq J_2\}$, $c_{\text{full}} = p_1(N(J_2))$, $L_{\text{CS}} = \big(\sum_{k\le\min(m_2,n_2)} h_{(k)} + P_2\big)/m_2$ and $H = \max_j(h_j + p_{2j})$.

Theorem 2 (Breakpoint form)**.** *Assume $P_2 > 0$. Then*

$$\max(\text{LB}_U, H) = \max\Big(L_{\text{CS}},\ H,\ \max_{c\in\Lambda,\, 0<c\le c_{\text{full}}} \Big\lceil \frac{c}{m_1} + \frac{P_2 - K(c^-)}{m_2} \Big\rceil\Big).$$

If $n_2 \ge m_2$ and $D(h_{(m_2)}) < P_2$, the term H can be dropped on both sides.

The proof is in Supplement S1.1. It splits f into a knapsack part, which is increasing between consecutive points of Λ, and a door-cap part, whose supremum is L_{CS} when $n_2 \ge m_2$. When $n_2 < m_2$, the door-cap part keeps rising until all outbound work fits, and its supremum lies between L_{CS} and H; the term H covers this case. On the benchmark $n_2 \ge 16 > m_2$. An equivalent inverse form uses the cheapest predecessor load $c^*(v)$ that reaches each outbound workload v (Supplement S1.1). The theorem has two consequences. First, LB_U can be computed exactly from the at most $\min(2^{n_2}, P_2)$ jumps of K, one knapsack per jump. Second, evaluating f on a grid of times can only under-estimate the bound.

Corollary 1 (Chen–Song bound)**.** $\max(\text{LB}_U, H) \ge L_{\text{CS}}$*, where L_{CS} is the bound of Chen and Song (2009). Applied to the instance and to its reversal, the union term therefore dominates their bound* $\max(L_{\text{CS}}(I), L_{\text{CS}}(R(I)))$.

The door cap $D(t)$ is needed for this. With the simpler cap $m_2(t - r_{\min})^+$ the union term does not dominate L_{CS}. For example, take $m_1 = m_2 = 2$, inbound lengths $(1, 10)$, one outbound truck of length 16 that needs the first inbound truck and ten of length 1 that need the second. Then $\lceil L_{\text{CS}} \rceil = 19 = \text{OPT}$, while the simpler bound gives 17 in both directions.

4.3. The bound we compute

The union term spreads work over doors as if trucks could be split. The classical bounds capture what it misses: the *load bound* $\text{LB}_{\text{load}} = \max(a + \underline{q},\ b)$, where $\underline{q} = \min_j p_{2j}$ if every inbound truck has a successor and 0 otherwise; the *head bound* $\text{LB}_{e2} = \max\big(\max_k[h_{(k)} + \sum_{l:\, h_l \ge h_{(k)}} p_{2l}/m_2],\ H\big)$ of the head-and-tail family (Carlier and Néron, 2000); and the *PCH bound* LB_{PCH} of Nogueira et al. (2020), which gives each inbound truck i the tail $\Delta_i = \max\big(p_2(S(i))/m_2,\ \max_{j\in S(i)} p_{2j}\big)$ and maximises the inbound work of the trucks with the k largest tails plus the k-th largest tail (Supplement S1.2 gives the formula). We define

$$\text{UPB} = \max\big(\text{LB}_U,\ L_{\text{CS}},\ \text{LB}_{\text{PCH}},\ \text{LB}_{e2},\ \text{LB}_{\text{load}}\big), \qquad \text{LB}^* = \max\big(\text{UPB}(I),\ \text{UPB}(R(I))\big).$$

Both are valid by Theorem 1 and Proposition 1. By definition, UPB is never below the earlier bounds, so on the benchmark the informative quantity is the size of its gain. By Corollary 1, the term L_{CS} is redundant: it is contained in $\max(\text{LB}_U, \text{LB}_{e2})$.

Remark 2 (A coupled version). One can couple the union term with the longest unstarted truck: for each t, minimise over head-feasible sets A the larger of $(P_2 - \min(p_2(A), D(t)))/m_2$

and $\max_{j \notin A} p_{2j}$. The resulting bound LB_U^{Φ} dominates LB_U, $\mathrm{LB}_{\mathrm{load}}$, LB_{e2} and $\mathrm{LB}_{\mathrm{PCH}}$ by one argument (Supplement S1.2). We do not compute it. On 15,000 tiny random instances it exceeded UPB on only one, so the maximum of the components is what we compute and report.

Complexity. Computing $K(c)$ is strongly NP-hard and has no fully polynomial approximation scheme unless P = NP. This follows from the two-level clique reduction of Johnson and Niemi (1983) for partially ordered knapsacks; the unit-time case with two predecessors per outbound truck already suffices (Supplement S2). $K(c)$ is easier for nested, single-predecessor, block, laminar and interval structures, and the LP relaxation of the knapsack is the upper concave envelope of the points $(p_1(N(A)), p_2(A))$, which one parametric minimum cut computes (Gallo et al., 1989). The resulting LP bound is strongly polynomial, but on all 10,500 benchmark instances and 240 structured ones it was never above the classical bounds (Supplement S2). The integer knapsack is what makes the bound strong.

4.4. Computing the bound

We evaluate Theorem 2 by a walk that goes down over integer capacities c, starting just below c_{full}. With integer data, capacity c gives the term $(c+1)/m_1 + \big(P_2 - \min(K(c),\, m_2((c+1)/m_1 - r_{\min}))\big)/m_2$. Here we may use the simpler cap $m_2(t - r_{\min})$, because the door-cap part of the bound equals L_{CS}, and UPB already contains L_{CS} as a separate term. Before solving a knapsack, the walk compares an optimistic value of the term with the best bound so far and skips capacities that cannot raise it. It solves $K(c)$ by an exact depth-first search with a node limit, and by CP-SAT beyond this limit. The optimal set at c has a cost $c' \le c$, so the walk continues at $c' - 1$. If a solve stops early, the walk uses the solver's upper bound on $K(c)$ (Remark 1). The walk starts from $L_0 = \max(L_{\mathrm{CS}}, \mathrm{LB}_{\mathrm{PCH}}, \mathrm{LB}_{e2}, \mathrm{LB}_{\mathrm{load}})$ and reports $\lceil \max(L_0, w) \rceil$, where w is its best term. This value is valid, never below L_0, and equal to $\lceil \mathrm{UPB} \rceil$ when the walk finishes.

An unfinished walk under a CPU cap gives a valid bound whose value can depend on the machine. We therefore report the deterministic version UPB-det throughout. It caps the walk at 50 steps and gives each CP-SAT solve 0.5 units of the solver's deterministic time, so it runs in pseudo-polynomial time (polynomial in n_1, n_2 and P_1), and its value depends only on the instance and the solver version. On all 10,500 instances it finished the walk on 79% (on every instance with $n_1 \le 50$), and its values and step counts were identical on two machines on the 150 instances that we checked. Supplement S4 compares it with the CPU-capped walk.

4.5. A certificate for every schedule

A bound is more useful when we know how far any heuristic schedule can be from it. The following is the classical typed-task argument (Liu and Liu, 1978; Jaffe, 1980; Liu et al., 1981) applied to the list decoder of Section 3 and to UPB.

Corollary 2 (Certificate)**.** *For every inbound order π, let ℓ be the outbound truck that finishes last and i its predecessor that finishes last. Then*

$$\mathrm{UB}(\pi) \le a + b + \Big(1 - \frac{1}{m_1}\Big) p_{1i} + \Big(1 - \frac{1}{m_2}\Big) p_{2\ell} \le \Big(3 - \frac{1}{M}\Big) \mathrm{UPB}.$$

If an inbound truck finishes last, $\mathrm{UB}(\pi) \le (2 - 1/m_1)\mathrm{UPB}$.

Proof. When truck i is assigned, every inbound door is busy until s_i, so $C_{1i} \le a + (1 - 1/m_1)p_{1i}$. Stage 2 is non-idle, so all outbound doors are busy between the release of ℓ and its start, which gives $\mathrm{UB}(\pi) \le C_{1i} + b + (1 - 1/m_2)p_{2\ell}$. Finally $a, b \le \mathrm{UPB}$ and $p_{1i} + p_{2\ell} \le h_\ell + p_{2\ell} \le \mathrm{LB}_{e2} \le \mathrm{UPB}$. □

Against the optimum, the constant $3 - 1/M$ is due to Jaffe (1980) and Liu and Liu (1978). What the corollary adds is practical: the ratio UB/UPB is a certificate computed for each instance that never exceeds $3 - 1/M$, and the middle expression is a residual that can be computed after the schedule is built. On all 10,500 instances, with the worst of three orders (the PCH order, longest first and a random order), the largest ratio UB/UPB was 1.488, and the largest ratio relative to the instance's guarantee $3 - 1/M$ was 0.576. Supplement S3 gives a family with ratio $2m/(m + 1)$, shows that the density of the benchmark alone cannot improve the constant, and reports a search for bad two-level instances.

4.6. A stronger bound by window inference

LB^* is computed once, from the data. Its distance to the optimum (Section 8.2) is mostly a lack of knowledge about time windows: on 95 solved instances, a preemptive relaxation of the outbound stage that is given the start times of an optimal schedule as release dates closes 97.7% of the gap between LB^* and the optimum (Supplement S5.2). This suggests a destructive bound. For a trial makespan T, we infer windows that every schedule with makespan at most T must respect, and we declare T infeasible when the windows become inconsistent. We call the resulting bound the *window-inference bound* UPB-W.

Setting. Data are integers, so if $\mathrm{OPT} \le T$ there is an integer schedule with makespan at most T; let $\mathcal{S}_T$ be the set of these schedules. A window set $W = (r^1, d^1, r^2, d^2)$ is *T-valid* if every $S \in \mathcal{S}_T$ satisfies $r^1_i \le s_i$ and $C_i \le d^1_i$ for every inbound truck, and $r^2_j \le s_j$ and $s_j + p_{2j} \le d^2_j$ for every outbound truck. The initial set W^0 uses heads and tails computed with dual-feasible functions (Fekete and Schepers, 2001; Clautiaux et al., 2010). Each rule below either maps a T-valid set to a T-valid set or proves $\mathcal{S}_T = \emptyset$.

Proposition 2 (Validity of UPB-W)**.** *If a finite sequence of the rules below, started from W^0, proves $\mathcal{S}_T = \emptyset$, then* $\mathrm{OPT} > T$. *Hence UPB-W* $= \max\big(\lceil \mathrm{LB}^* \rceil,\ 1 + \max\{T :\ T \text{ refuted}\}\big)$ *is a valid lower bound, where the inner maximum is $-\infty$ when no trial is refuted. It needs no monotonicity in T, and a test that stops without an answer counts as "not refuted".*

The proof is an induction over the sequence (Supplement S5). The rules are the following.

- *Union propagation* (Lemma S3 in the supplement). For an outbound truck j and a time α, the work of $N(j)$ that must lie after α is $\Sigma_j(\alpha) = \sum_{i \in N(j)} \min\big(p_{1i}, (r^1_i + p_{1i} - \alpha)^+\big)$, so r^2_j can be raised to $\alpha + \lceil \Sigma_j(\alpha)/m_1 \rceil$ whenever $\Sigma_j(\alpha) > 0$. The mirror rule lowers d^1_i over $S(i)$. This is the classical energetic adjustment of a release date, applied to the AND-predecessor set of a cross-dock truck.

- *Per-stage tests and shaving.* Each stage, with the other stage replaced by its windows, is a parallel-machine problem with release dates and deadlines. We test it by the preemptive flow network of Horn (1974) and the energetic test of Baptiste et al. (1999). Shaving refutes a half-window hypothesis such as $s_j \le t-1$ and keeps the other half, as in Haouari et al. (2012).

- *The split-cut test* (Proposition S7). At a cut time t, the set Y of inbound trucks completed by $t-1$ and the set A of outbound trucks started by $t-1$ must satisfy closure ($j \in A \Rightarrow N(j) \subseteq Y$), packing on both stages (Y fits into m_1 bins of size $t-1$, $J_2 \setminus A$ into m_2 bins of size $T-t$), work conditions on both stages and the forcing implied by the windows. If no such pair exists, or if the LP relaxation of these conditions is infeasible, then $\mathcal{S}_T = \emptyset$. This extends Lemma 1 from one set to a coupled pair.

- *Door caps* (Lemma S5). The k-th smallest outbound start is at least the k-th smallest r_j^2 and at least the time the inbound doors need for the predecessors of any k outbound trucks. This sharpens $D(t)$ from the current windows, so shaving and the caps tighten each other.

For each trial T, starting at $\lceil \mathrm{LB}^* \rceil$ and moving up by doubling and then bisection, the procedure repeats propagation, the per-stage tests, shaving and LP probes of the split-cut test at 16 cut times until no window changes or the time is used up. It then runs the 0/1 split-cut test in CP-SAT, exact single-stage tests and a nested version with several cuts. The part with propagation, flow, energetic tests and LP probes is polynomial; the full procedure is not, because its CP-SAT tests decide strongly NP-hard questions.

Credit. Window propagation across stages with shaving is due to Néron et al. (2001) and Hidri and Haouari (2011), in hybrid flow shops where each operation has one predecessor. Shaving to a fixpoint with energetic reasoning and dual-feasible functions is established for the RCPSP (Haouari et al., 2012; Carlier and Néron, 2007); energetic reasoning is used in the form of Baptiste et al. (1999, 2001); dual-feasible functions are due to Fekete and Schepers (2001) and Clautiaux et al. (2010), and Lahimer et al. (2013) apply them per stage of a hybrid flow shop. UPB-W instantiates the destructive-improvement scheme of Klein and Scholl (1999): a trial makespan is refuted by window reduction and feasibility tests. Our flow test is the identical-machine specialisation of the preemptive time-window LP that Brucker and Knust (2000) use destructively for the RCPSP (cf. Horn, 1974). What is ours is the cross-dock-specific part: propagation over predecessor unions between the two stages, the split-cut test with closure and packing on both stages, the door caps with union-based staggering, LP refutations backed by exact-arithmetic certificates, a starting point given by UPB, which dominates the bound of Chen and Song (2009), and the computation on the benchmark.

5. The upper-bound method

The bound needs good schedules to become a certificate. Our upper-bound method searches over orders of the inbound trucks and builds each schedule with the list decoder of Section 3. One decode of our implementation costs $O(n_1 m_1 + |E| + n_2 m_2 + n_2^2)$ time, where $|E| = \sum_j |N(j)|$, and is never asymptotically more expensive than one evaluation of the public EACDC code

(Supplement S7.1). It returns the same makespan as the EACDC evaluator on 1,000 random orders over 40 instances. By Proposition 1, a schedule of the reversed instance can be mirrored into a schedule of I with the same makespan, so every part of the method can run on either side. For each instance we use the side whose best constructive order is better.

Constructive orders. We use seven rules, each applied to I and to $R(I)$, which gives 14 orders at 2 to 10 ms each: the PCH order of Nogueira et al. (2020) (decreasing tail Δ_i), longest first, most successors first, and four *union-greedy* orders. A union-greedy order grows a set U of scheduled inbound trucks. At each step it adds the missing predecessors $N(j) \setminus U$ of the outbound truck j with the largest ratio of released outbound work to the cost $p_1(N(j) \setminus U)^\alpha$, for $\alpha \in \{0.5, 1, 2\}$. This is the greedy algorithm for the knapsack $K(c)$: it schedules first the inbound trucks that release the most outbound work per unit of inbound capacity.

Iterated local search. The search works on the inbound order π (Figure 1). It accepts moves that leave the makespan unchanged, so it can cross the large plateaus of this objective. Its stall limit grows with n_1, and its perturbation restarts from the best order found. It has three parameters, which were fixed on the development instances before any confirmation run. We call this procedure the *base ILS*.

Algorithm 1 (base ILS)
$\pi \leftarrow$ best of the 14 constructive orders; $f \leftarrow \mathrm{UB}(\pi)$; $\pi^\star \leftarrow \pi$; *stall* $\leftarrow 0$.
while the budget is not spent **do**
  move the truck at a random position u to a random position $v \neq u$, giving π';
  if $\mathrm{UB}(\pi') \leq f$ **then** $\pi \leftarrow \pi'$, $f \leftarrow \mathrm{UB}(\pi')$, update $\pi^\star$, and reset *stall* if f strictly improved;
  else *stall* $\leftarrow$ *stall* $+ 1$;
  if *stall* $> 40\, n_1$ **then** $\pi \leftarrow \pi^\star$ with a random segment reversed and two random moves;
$f \leftarrow \mathrm{UB}(\pi)$; *stall* $\leftarrow 0$.
return $\pi^\star$, mapped back to I if the search ran on $R(I)$.

Figure 1: The base iterated local search. A move removes one truck from the order and inserts it elsewhere.

FB-ILS. The *Forward–Backward Iterated Local Search* (FB-ILS) adds two mechanisms to the base ILS. First, a forward–backward step re-sequences the schedule on the reversed instance by decreasing outbound completion time, decodes it and maps it back, as long as the makespan strictly improves. This is the forward–backward improvement, or justification, of project scheduling (Li and Willis, 1992; Valls et al., 2005), applied to the inbound order through the reversed cross-dock instance. Second, window insertion draws 90% of the insertion targets within $\pm 25\%$ of n_1 positions from the source. FB-ILS is compiled to C, which gives about 45 times more decodes per second than our Python decoder. Every decode, including those of the forward–backward step, counts as an evaluation. FB-ILS was chosen among more than ten variants on the development instances only; the others, including iterated greedy (Ruiz and Stützle, 2007), were weaker per evaluation (Supplement S7).

6. An exact method with union cuts

A lower bound can help an exact method in two ways. It can be given once, as a bound on the objective at the root, or its argument can be written as constraints that the solver

propagates at every node. We tested both with the constraint-programming solver CP-SAT.

Base model. Each truck has an integer start variable and a fixed-length interval on its side. A cumulative constraint with capacity m_1 (respectively m_2) limits the number of trucks at the inbound (outbound) doors at any time. Each predecessor pair gives $s_j \geq s_i + p_{1i}$, and the makespan is at least every completion time. We add $s_j \geq \lfloor h_j \rfloor$ for every outbound truck and $C_{\max} \geq s_i + p_{1i} + \lfloor \tau_i \rfloor$ for every inbound truck, where τ_i is a lower bound on the time that the successors of i still need after i finishes. The floors are valid but slightly weaker than ceilings for integer start times; they are what the frozen code uses.

Union cuts. Choose an integer cut time θ. Let x_j be true exactly when outbound truck j starts before θ, and y_i true exactly when inbound truck i is finished by θ: $x_j \Leftrightarrow s_j \leq \theta - 1$ and $y_i \Leftrightarrow s_i + p_{1i} \leq \theta$. The *forward union cut* at θ consists of

$$x_j \Rightarrow y_i \quad \text{for every } i \in N(j), \tag{2}$$

$$\textstyle\sum_i p_{1i}\, y_i \leq m_1 \theta, \tag{3}$$

$$\textstyle\sum_j p_{2j}\,(1 - x_j) \leq m_2 (C_{\max} - \theta). \tag{4}$$

The *reversed union cut* is the same construction on the reversed instance.

Proposition 3. *If $\theta \leq \mathrm{OPT}$, every feasible schedule satisfies the forward and reversed union cuts at θ.*

Proof. Set the Booleans by their definitions. (2) holds because an outbound truck that starts before θ has all its predecessors finished by then. (3) holds because this inbound work was processed in $[0, \theta]$ on m_1 doors. Every feasible schedule has $C_{\max} \geq \mathrm{OPT} \geq \theta$, and an outbound truck that does not start before θ is processed in $[\theta, C_{\max}]$, which gives (4). The reversed cut is the forward cut of the mirrored schedule. □

The condition $\theta \leq \mathrm{OPT}$ matters: for a larger θ, the right-hand side of (4) could be negative. We therefore place $k = 12$ integer cut times evenly over $(r_{\min}, L]$, where L is the current proven lower bound. At the root, the cuts express the reasoning of LB_U at these times. During the search they also act on partial schedules: once some trucks are fixed, the solver can deduce that other outbound trucks cannot start before θ, or that the makespan must grow.

Search procedure. All variants get 300 s of CPU per instance and one solver worker. The lower bound starts at $\lceil \mathrm{LB}^* \rceil$, and the incumbent is the base ILS schedule after 10 s, which is also given to the solver as a hint. For at most 15% of the time, decision probes ask whether a schedule with makespan at most $L + \delta$ exists, starting with $\delta = 0$ and allowing at most 5 s per probe. A refuted probe raises L to $L + \delta + 1$ and sets δ to $\max(1, 2\delta)$. A found schedule or a timeout ends this phase. One optimisation run then minimises $C_{\max}$ subject to $C_{\max} \geq L$ for the remaining time. The variant CP-LB (LB^* as root bound) uses LB^* only in this way, at the root. The variant CP-UC (CP-SAT with union cuts) adds the forward and reversed union cuts at 12 cut times to every model that it solves. The variant CP-plain is the base model without the head and tail cuts, solved in optimisation mode with the same hint and no lower bound. Supplement S9

describes a further variant without cuts, CP-CS, started from the bound of Chen and Song (2009). Reported times include the probes but not the 10 s of ILS or the computation of LB*.

Cost of the methods. Table S7 in the supplement gives the worst-case time and memory of every method. One decode of the heuristics takes $O(n_1 m_1 + |E| + n_2 m_2 + n_2^2)$ time. The searches repeat it up to their evaluation budget, after a constructive phase of $O(n_1^2 n_2^2)$ time. UPB-det is polynomial in n_1, n_2 and P_1, and $\mathrm{LB}_U^{\mathrm{LP}}$ needs at most $2(n_1 + n_2) + 1$ maximum flows. The polynomial part of UPB-W takes $O(n^4 \log^2(P_1 + P_2))$ time plus $O(n \log(P_1 + P_2))$ linear programs, where $n = n_1 + n_2$. Its CP-SAT tests and the exact method are exponential in the worst case, and a time budget caps both. Memory is $O(n + |E|)$ for the heuristics, $O(P_1 + n + |E|)$ for UPB-det and $O(n^2)$ for the polynomial part of UPB-W; CP-SAT has no bound fixed in advance.

7. Experimental design

The experiments answer four questions. How much higher is the new bound than the earlier bounds and other bounding methods? How far is it from the optimum, and what do the union cuts do inside an exact solver? Do the schedules of FB-ILS match or improve on the earlier heuristics at equal effort? And do the results hold outside the benchmark's random structure?

Instance sets.

- *Census*: all 10,500 benchmark instances, with the per-instance HgR upper bounds and Chen–Song bounds published by Fonseca et al. (2019) (Fonseca, 2019). Census values are population means and carry no sampling error.
- *Development set*: 105 benchmark instances, three per cell. The heuristics were designed on this set, so results on it are exploratory.
- *Exact set*: 300 benchmark instances, 30 per cell for $n_1 \in \{20, 30\}$ and the five door groups.
- *Exact confirmation set*: 300 new instances with the same design, 30 per cell for $n_1 \in \{20, 30\}$ and the five door groups, generated with the benchmark's rules (generator seed 20261193) after the exact method was frozen.
- *Confirmation sets*: three sets of 210 new instances, six per cell, generated with the benchmark's rules (generator seeds 20261103, 20261105 and 20261107, the first set being reused with new run seeds for EACDCml). None duplicates a benchmark file.
- *Structured set*: 240 new instances with sparse precedence (density 0.1–0.3), block precedence (destination groups), strongly unbalanced door numbers and n_1 up to 300 (generator seed 20261104).
- *Single-door set*: the 500 instances of Fonseca et al. (2019) with one door per side.

Implementations. The bounds, the decoder of the base ILS and the HgR re-implementation are written in Python; FB-ILS is compiled to C. Knapsacks and the exact models use CP-SAT from OR-Tools 9.15 with one worker. EACDC is the authors' public `GA.py` (Nogueira et al., 2022) with the static configuration of their Table 2, run with a faster pure-Python evaluator that gives the same results at 13.4 times lower CPU cost; this favours EACDC in every CPU-matched comparison. We record for an EACDC run the best makespan it ever evaluated, which again

favours EACDC. The trained network of EACDCml is not public. We rebuilt EACDCml on top of the authors' code with their calibration budget; this rebuild is what EACDCml denotes throughout. A second rebuild follows the description in the paper (EACDCml (paper-based)). The model file of the public HgR script is not public either, so we re-implemented HgR from the paper. We say *rebuild* for EACDCml and *re-implementation* for HgR. All results reported for EACDCml and HgR therefore come from our rebuild and re-implementation, run by us; both reproduce the published results within sampling error (Supplement S7.4). All runs used one thread per process, on Apple-silicon computers and, for some runs, one x86 laptop under Linux; we never compare CPU times measured in different runs.

Matched effort. A fixed CPU budget is a poor basis for comparing a local search with EACDC, which builds its whole initial population before its first evaluation. We therefore match the effort run by run. For each instance and seed, one process runs the rival to completion and then the local search, or the other way round, in alternating order. The local search is read at the rival's CPU time and at the rival's number of schedule evaluations. The two matchings answer different questions. The evaluation-matched comparison does not depend on how fast each implementation evaluates a schedule, so it compares the algorithms. The CPU-matched comparison measures the implementations as they are; for FB-ILS it compares compiled with interpreted code. We count evaluations in two ways: *generation evaluations* exclude the rival's initial population, which favours the rival, and *all evaluations* include it.

Preregistered confirmation. Seven confirmatory families were fixed in written protocols before their instances were generated or run, with frozen code identified by hashes. Table 1 lists all 30 of their confirmatory endpoints. Each family uses Holm's correction over its own endpoints (family G over two groups of three); we did not correct across families. A further preregistered study, the anytime comparison at 1, 10 and 100 times the rivals' budgets, has its own Holm family of 15 endpoints; it is reported separately in Supplement S8, where 14 of the 15 endpoints were confirmed. An endpoint counts as confirmed if its Holm-adjusted p-value is below 0.05 and its 95% interval excludes zero in the expected direction; bound endpoints also required that no lower bound exceed the best upper bound found. Everything not in Table 1 or in that anytime study is exploratory, including the census itself, the exact-method results on the exact set, UPB-W and the new best-known upper bounds. The SHA-256 digests of the protocol documents are deposited with a public timestamp in the data release (Provezano Coutinho, 2026); the documents themselves are available on request. The deposit postdates the runs, which the deposit record states.

Statistics. The unit of analysis is the instance. For each instance we average over seeds and take the relative difference $(\mathrm{UB}_{\mathrm{ILS}} - \mathrm{UB}_{\mathrm{rival}})/\mathrm{UB}_{\mathrm{rival}}$. We report the mean with a 95% percentile bootstrap interval over instances (10,000 resamples). Tests are one-sided sign-flip permutation tests over instances with 100,000 random sign vectors, except in the HgR family, which used one-sided Wilcoxon signed-rank tests as fixed in its protocol. Wins, ties and losses (W/T/L) count instances, with a relative tolerance of 10^{-9} for ties. Gaps are $(\mathrm{UB}-\mathrm{LB})/\mathrm{LB}$ unless stated otherwise; Supplement S11 relates this convention to the ones used in earlier papers.

8. Results

We report the lower bounds first, then the exact method, then the upper bounds and certificates, and finally the structured instances. Table 1 collects the endpoints of the seven confirmatory families; the anytime study of Section 8.4 is a separate preregistered study reported in Supplement S8. All other results are exploratory.

Table 1: All confirmatory endpoints. Each family was fixed in a written protocol before its instances were generated or run, and uses Holm's correction over its own endpoints. Differences are means over instances of $(\mathrm{UB}_{\mathrm{ours}} - \mathrm{UB}_{\mathrm{rival}})/\mathrm{UB}_{\mathrm{rival}}$ in % (negative favours our method) or, for bound endpoints (marked ↑), the gain of LB* over the baseline in % (positive favours LB*). Brackets are 95% bootstrap intervals over instances. W/T/L counts instances where our method is better, equal or worse (for bounds: higher, equal, lower). CPU: matched CPU time; "gen. evals" and "all evals": matched evaluation counts without and with the rival's initial population (Section 7). Only evaluation-matched rows and the HgR row H2 are independent of the implementation language. EACDCml (paper-based) is the version built from the paper's description (Section 7); "static" is the static configuration without the network. In family G, rows S1–S3 count proven optima (W/T/L: solved only by CP-UC, by both or neither, only by the other method; exact McNemar) and rows G1–G3 compare final gaps in percentage points (paired bootstrap); Holm's correction runs within S1–S3 and within G1–G3. Endpoint IDs follow the protocols. Every endpoint in this table was confirmed. The separate preregistered anytime study (15 endpoints, 14 confirmed) is reported in Supplement S8.

Family (instances; run seeds)	Endpoint	Comparison	Matched on	Difference (%)	W/T/L; Holm p
A. First confirmation	E2↑	LB* vs $\lceil\max(L_{\mathrm{CS}}, \mathrm{LB}_{\mathrm{PCH}})\rceil$	–	+14.90 [13.78, 15.96]	–/–/0; 3×10^{-5}
(210 new; 11–13)	E1	base ILS vs EACDC	CPU	−0.57 [−0.67, −0.46]	161/5/44; 3×10^{-5}
	E3	base ILS vs EACDC	gen. evals	−0.48 [−0.58, −0.37]	158/2/50; 3×10^{-5}
B. FB-ILS	E1	FB-ILS vs EACDC	CPU	−1.09 [−1.19, −1.00]	203/2/5; 4×10^{-5}
(210 new; 21–23)	E3	FB-ILS vs EACDC	all evals	−0.63 [−0.72, −0.53]	172/3/35; 4×10^{-5}
	E5	FB-ILS vs base ILS (ablation)	CPU	−0.57 [−0.63, −0.50]	195/15/0; 4×10^{-5}
	E6	FB-ILS vs base ILS (ablation)	all evals	−0.14 [−0.19, −0.10]	131/21/58; 4×10^{-5}
C. EACDCml, base ILS	E4	base ILS vs EACDCml	CPU	−0.61 [−0.70, −0.51]	176/4/30; 5×10^{-5}
(set A; 31–33)	E5	base ILS vs EACDCml	all evals	−0.56 [−0.66, −0.46]	173/2/35; 5×10^{-5}
	E1	base ILS vs EACDCml (paper-based)	CPU	−1.52 [−1.67, −1.37]	199/2/9; 5×10^{-5}
	E2	base ILS vs EACDCml (paper-based)	all evals	−1.52 [−1.67, −1.37]	198/2/10; 5×10^{-5}
	E3	base ILS vs EACDCml (paper-based), static	CPU	−1.71 [−1.86, −1.57]	204/3/3; 5×10^{-5}
D. EACDCml, FB-ILS	E11	FB-ILS vs EACDCml	CPU	−1.15 [−1.25, −1.05]	203/3/4; 4×10^{-5}
(set A; 31–33)	E12	FB-ILS vs EACDCml	all evals	−0.63 [−0.73, −0.53]	168/3/39; 4×10^{-5}
	E9	FB-ILS vs EACDCml (paper-based)	CPU	−2.11 [−2.27, −1.96]	209/0/1; 4×10^{-5}
	E10	FB-ILS vs EACDCml (paper-based)	all evals	−1.60 [−1.76, −1.45]	200/0/10; 4×10^{-5}
E. HgR	H1	FB-ILS vs HgR, 10 s	CPU	−2.89 [−3.07, −2.71]	210/0/0; 7×10^{-36}
(210 new; 41–43;	H2	base ILS vs HgR, 10 s (both Python)	CPU	−2.54 [−2.72, −2.37]	209/1/0; 7×10^{-36}
Wilcoxon)	H3	FB-ILS vs better HgR variant, 10 s	CPU	−2.80 [−2.97, −2.63]	209/0/1; 7×10^{-36}
	H4	FB-ILS vs HgR, 60 s (60 instances)	CPU	−3.03 [−3.42, −2.67]	60/0/0; 8×10^{-12}
F. Structured	E2↑	LB* vs $\lceil\max(L_{\mathrm{CS}}, \mathrm{LB}_{\mathrm{PCH}})\rceil$	–	+3.45 [2.79, 4.15]	129/111/0; $\leq 4 \times 10^{-5}$
(240 new; 11–13)	E4↑	LB* vs energetic reasoning	–	+3.16 [2.50, 3.85]	125/41/74; $\leq 4 \times 10^{-5}$
	E1	base ILS vs EACDC (58 instances)	CPU	−0.85 [−1.09, −0.63]	49/4/5; $\leq 4 \times 10^{-5}$
	E3	base ILS vs EACDC (58 instances)	gen. evals	−0.75 [−1.00, −0.53]	47/4/7; $\leq 4 \times 10^{-5}$
G. Exact method	S1	CP-UC vs CP-LB, solved	CPU	137 vs 47	92/206/2; 5×10^{-25}
(300 new; seed 0)	S2	CP-UC vs CP-plain, solved	CPU	137 vs 45	94/204/2; 4×10^{-25}
	S3	CP-UC vs time-indexed MIP, solved ($n_1 = 20$)	CPU	100 vs 12	89/60/1; 4×10^{-25}
	G1	CP-UC vs CP-LB, final gap (pp)	CPU	−2.91 [−3.23, −2.61]	226/71/3; 3×10^{-4}
	G2	CP-UC vs CP-plain, final gap (pp)	CPU	−31.68 [−33.63, −29.73]	254/44/2; 3×10^{-4}
	G3	CP-UC vs time-indexed MIP, final gap (pp; $n_1 = 20$)	CPU	−38.95 [−42.83, −34.92]	137/11/2; 3×10^{-4}

8.1. How much higher is the new bound?

Table 2 gives the census of UPB-det on all 10,500 instances. Against the best earlier bound, $\lceil\max(L_{\mathrm{CS}}, \mathrm{LB}_{\mathrm{PCH}})\rceil$, UPB is 15.76% higher on average (median 17.74%, maximum 31.19%). By construction it is never lower. The gain grows with the number of trucks, from 4.93% at $n_1 = 20$ to 22.21% at $n_1 = 80$. It is largest with two doors per side (20.35%) and smallest in the U(2,10) group (10.69%). With the new bound, the mean gap of the published HgR upper bounds falls from 26.1% to 9.1%. The choice of the baseline changes the mean gain by less than half a point (Supplement S4).

On the 210 instances of the first confirmation set, the preregistered endpoint E2 was confirmed (Table 1). LB* is 14.90% [13.78, 15.96] above the best earlier bound, no lower bound

Table 2: Census of the deterministic bound UPB-det (forward instance) on all 10,500 benchmark instances. B is the best earlier bound, $\lceil \max(L_{\mathrm{CS}}, \mathrm{LB}_{\mathrm{PCH}}) \rceil$, with L_{CS} as reported with the published HgR results. "Gain" is $(\mathrm{UPB} - B)/B$ in %. The HgR gaps use the published HgR upper bound and $(\mathrm{UB} - \mathrm{LB})/\mathrm{LB}$. "Fin." is the share of instances on which the walk finished within its step cap. The census covers every instance, so the values are population means, not estimates.

		Gain of UPB over B (%)				HgR gap (%)			
Slice	n	mean	median	max	# lower	vs B	vs UPB	Fin. (%)	CPU s (mean / max)
All	10,500	15.76	17.74	31.19	0	26.1	9.1	79	3.0 / 36.4
$n_1 = 20$	1,500	4.93	3.83	23.51	0	17.4	11.9	100	0.0 / 0.0
$n_1 = 30$	1,500	9.55	10.83	24.71	0	22.8	12.3	100	0.0 / 0.1
$n_1 = 40$	1,500	14.51	15.70	27.05	0	26.3	10.4	100	0.1 / 0.4
$n_1 = 50$	1,500	17.91	19.10	29.14	0	28.2	8.7	100	0.4 / 2.0
$n_1 = 60$	1,500	19.91	21.06	30.98	0	29.0	7.6	95	1.7 / 11.2
$n_1 = 70$	1,500	21.30	22.72	31.19	0	29.4	6.7	49	5.3 / 16.4
$n_1 = 80$	1,500	22.21	23.67	31.03	0	29.9	6.3	8	13.4 / 36.4
$m = 2$	2,100	20.35	21.44	31.19	0	26.5	5.2	78	3.0 / 26.0
$m = 4$	2,100	18.73	20.29	30.05	0	29.1	8.9	78	3.1 / 26.4
$m = 10$	2,100	13.32	16.76	27.68	0	30.2	15.1	78	2.8 / 19.7
U(2,4)	2,100	15.71	16.40	30.49	0	23.4	6.7	79	3.2 / 36.3
U(2,10)	2,100	10.69	10.42	30.25	0	21.4	9.8	80	3.0 / 36.4

exceeded the best upper bound found, and the mean gap between the best upper bound and LB* was 6.22%. This endpoint used the CPU-capped walk. Where both walks finish, it gives the same value as UPB-det; elsewhere its value is lower or equal, except on 3 of the 10,500 instances (Supplement S4).

Comparison with other bounding methods. A gain over head-sum bounds says little if a general technique would achieve the same. Table 3 compares LB* with four other bounds. A two-stage energetic-reasoning bound gains only 1.23% over the rounded Chen–Song bound, and LB* is 14.00% above it. Energetic reasoning is higher on 7.5% of the instances, mostly with few trucks and many doors; there the union argument matters less. The LP relaxation of the published time-indexed model is much weaker (0.69 of UPB at $n_1 = 20$); its makespan linking is the weak form analysed by Wallrath et al. (2025). With strengthened linking it reaches 0.90 of UPB and is higher on 2 of 50 benchmark instances, both with ten doors per side, and on 38 of 300 tiny generated instances, but it takes minutes per instance against milliseconds (Supplement S6). The bound that CP-SAT proves by itself in 300 s is far below LB*. On the single-door instances (one door per side), LB* is above the published Lagrangian means in every size class.

8.2. Distance to the optimum and the exact method

The bound at the root only. With LB* only at the root (variant CP-LB) and the other variants without cuts, CP-SAT proved optimality on 62 of the 300 instances of the exact set within 300 s, all with $n_1 = 20$. On 30 instances, starting from LB* instead of the Chen–Song bound did not shorten the time to proof (geometric-mean time ratio 1.01), but it raised the final lower bound of the instances that stayed open (Supplement S9). Used only at the root, the bound improves the certificate but does not speed up the proof.

Union cuts inside the search. Table 4 reports CP-UC, which adds the union cuts to every model, with the same limit, starting bound and schedule. CP-UC proved optimality on 161

Table 3: LB* against other bounding methods. Full tables are in Supplement S6.

Bound	Instances	Result	CPU per instance
Two-stage energetic reasoning, style of Néron et al. (2001)	all 10,500	LB* 14.00% higher on average; energetic bound higher on 7.5% (88.7% with ten doors per side and $n_1 = 20$); the maximum of the two adds 0.14%	0.205 s vs 3.0 s (UPB-det)
LP relaxation of the published time-indexed model (Cota et al., 2016) with two valid cuts (TIM-LP)	50 with $n_1 = 20$; 9 with $n_1 = 30$–50	never above UPB; ratio to UPB 0.687 [0.640, 0.735] at $n_1 = 20$, 0.41–0.74 on the 6 larger ones solved in 30 min	23.7 s vs 0.008 s ($n_1 = 20$)
TIM-LP with, in addition, strengthened makespan linking	50 with $n_1 = 20$	above UPB on 2 of 50 (ten doors per side), equal on 3; ratio to UPB 0.896 [0.874, 0.918]	451 s vs 0.008 s
Bound proved by CP-SAT after 300 s	14 unsolved, $n_1 = 20$	25.8% below LB*; with probes from L_{CS}, 1.7% below (16 instances)	300 s vs 0.01 s
Hybrid Lagrangian bound, one door per side (Fonseca et al., 2019)	500, one door per side	LB* higher in all 10 rows of published means, by 2.0–15.1%; equal to the optimum on 86 of 200 with $n \leq 10$	– vs ≤6.4 s

of the 300 instances, against 59 for CP-LB and 62 for the variants without cuts together: 114 of 150 at $n_1 = 20$ and 47 of 150 at $n_1 = 30$, where no variant without cuts closed any. At $n_1 = 30$ it closed 24 of 30 instances with two doors per side and none with four or ten. On the 139 instances that CP-UC did not solve, its remaining gap is 5.35%, against 8.86% for the best variant without cuts. All optima agree with the earlier runs. Together with the four instances solved only by variants without cuts, 165 optima are known on the exact set (118 at $n_1 = 20$ and 47 at $n_1 = 30$). Solved counts under a CPU limit depend on the machine, and on the exact set CP-UC ran on a faster machine than CP-LB (Table 4, note a). We therefore re-timed both on one machine, on the 55 instances that both had solved, with a 60 s limit: CP-UC proved 49 and CP-LB 26. The median time to proof was 4.8 s for CP-UC; for CP-LB it is above the limit, because fewer than half of its runs finished. A search node of CP-UC is not cheaper than one of CP-LB, so the faster proofs come from the propagation of the cuts (Supplement S9).

The cut design was developed on these 300 instances, so we confirmed it on 300 new instances with the same sizes and door groups, generated after the method was frozen, in a preregistered test. Within 300 s, CP-UC proved optimality on 137 of them, against 47 for CP-LB, 45 for CP-plain and 12 of 150 ($n_1 = 20$) for the time-indexed model of Cota et al. (2016) solved as a MIP with HiGHS (one thread, same start schedule). In each comparison CP-UC solved almost every instance that the other method solved (92:2, 94:2 and 89:1 in the discordant pairs; Holm-adjusted McNemar $p < 10^{-24}$), and its final gap was 2.91 [2.61, 3.23] points smaller than that of CP-LB. All arms of an instance ran in one process on one machine. The new instances ran on two other machines, one x86 laptop and one Apple-silicon computer, on which CP-UC solved 37% and 57% of its instances (46% overall, against 54% on the exact set). Solved shares therefore depend on the machine and are not compared across runs; the paired comparisons hold on each machine separately (Supplement S9).

How far is the bound from the optimum?. On the 165 proven optima, LB* equals the optimum on 3 of 118 instances at $n_1 = 20$ and on none of 47 at $n_1 = 30$. It lies 6.62% and 3.63% below

Table 4: CP-SAT with reified union cuts (CP-UC) against the variants without cuts on the 300 instances of the exact set, 300 s per instance, same starting bound and schedule. "Without cuts" counts an instance if any variant without cuts solved it (CP-LB, CP-CS or CP-plain; Supplement S9). The rows on LB* and on the gaps to the optimum use the 165 known optima (118 and 47), which include four instances solved only without cuts. Exploratory: the cut design was developed on these instances; the preregistered confirmation on 300 new instances is in Table 1 (row G).

	$n_1 = 20$	$n_1 = 30$
Proved optimal by CP-UC (of 150)	114	47
Proved optimal by CP-LB / by any variant without cuts	59 / 62	0 / 0
two doors per side (of 30): CP-UC / without cuts	30 / 21	24 / 0
four doors per side	14 / 0	0 / 0
ten doors per side	25 / 24	0 / 0
U(2,4)	30 / 5	14 / 0
U(2,10)	15 / 12	9 / 0
Solved by both CP-UC and CP-LB (55): median CPU to proof, main runs[a]	4.7 s vs 47.0 s	
same machine, 60 s limit: solved / median CPU	49 vs 26; 4.8 s vs above the limit[b]	
Unsolved by CP-UC: remaining gap, CP-UC / best without cuts	5.35% / 8.86% (139 instances)	
LB* = OPT on the known optima	3 of 118	0 of 47
Mean (OPT − LB*)/OPT	6.62%	3.63%
Gap to OPT: HgR / EACDC (one run) / ILS 10 s	3.53 / 1.59 / 0.40%	2.73 / 0.90 / 0.10%

[a] CP-UC on an Apple M5 Pro, CP-LB on an Apple M3 Max (PassMark single-thread ratio 1.23). [b] Censored: CP-LB solved fewer than half of the 55 runs within the limit.

the optimum on average. The solved instances are the easier part of these sizes. An exact fluid relaxation at $n_1 = 20$, in which each stage is one machine of speed m and only the order of the inbound trucks matters, locates two sources of this slack. With two doors per side and in U(2,4), the fluid relaxation is above LB* on 97% of the instances: the bound loses because it may pick an unrelated set of outbound trucks at each time, while a schedule needs a nested sequence of sets. With ten doors per side the fluid relaxation is never above LB*: there the slack comes from the few trucks per door, which a fluid argument cannot see. On the same optima, the base ILS after 10 s is 0.40% ($n_1 = 20$) and 0.10% ($n_1 = 30$) above the optimum, one EACDC run 1.59% and 0.90%, and the published HgR solutions 3.53% and 2.73%. At these sizes almost all of the remaining certified gap is therefore due to the bound.

8.3. The window-inference bound

We evaluated UPB-W on the 98 instances of the exact set whose optimum was known when these experiments were designed and on the development set. Before any run we wrote down a go/no-go test: UPB-W had to close at least 70% of the gap between LB* and the optimum on the 98 instances, and its mean gain over LB* at $n_1 = 60$ had to have a 95% interval above zero, with a nominal budget of 120 s. The search checks its budget only between trial tests, so it used up to 168 s at $n_1 = 60$ and 244 s on the optima.

Table 5 gives the results. UPB-W closes 73.9% [70.2, 77.4] of the gap, against 61.0% [57.3, 64.6] for the previous best, and the mean distance to the optimum falls from 5.48% to 1.43%. It equals the optimum on 11 of the 98 instances, against 3 for LB*. At $n_1 = 60$ it gains 0.24% [0.09, 0.41], so both parts of the test were met. At $n_1 = 70$ and 80 the bound does not rise within 120 s, but it does with 600 s (0.98% and 0.39%), so its gain there is limited by the budget. An oracle experiment shows that, given the heads and tails of an optimal schedule, the polynomial tests close all of the gap, against 21.6% with the inferred windows. The inference of windows, not the tests, is the bottleneck (Supplement S5.2).

Validity. We re-derived all 285 reported values of UPB-W above LB* (the 98 optima and the development set at 120 s, and the 300 and 600 s runs). Each time the floating-point LP solver reported infeasibility, we checked a Farkas certificate in exact rational arithmetic for the system with integer right-hand sides. Any report without a certificate would have counted as "not refuted". All 2,949 reports in the deciding refutations, and all 20,784 reports in full replays of the 120 s runs, are certified, so no value changes. Replays on a different machine ended 1–2 units lower on 4 of 116 runs, because the CP-SAT sub-tests run under time limits. For three of them the deciding refutation was found and certified on that machine when tested directly; for the fourth it is found and certified only on the machine of the original run. So one reported value depends on the platform. On 610 tiny instances with optima computed by enumeration, where the search was forced to test $T = \mathrm{OPT}$, no variant of the procedure refuted the optimum, and an exact check confirmed that all 4,880 final window sets contain every schedule with makespan at most T. That no bound in our runs exceeds the optimum is not evidence, because the search never tests trial values at or above a known upper bound.

Table 5: The window-inference bound UPB-W. Top: 98 proven optima. "Closed" is $(\mathrm{LB} - \mathrm{LB}^*)/(\mathrm{OPT} - \mathrm{LB}^*)$ in %, over the 95 instances with $\mathrm{LB}^* < \mathrm{OPT}$; "dist." is the mean of $(\mathrm{OPT} - \mathrm{LB})/\mathrm{OPT}$ in %; LB^* equals the optimum on 3 instances. "Previous best" is the maximum of the split-cut, nested, single-stage and shaving tests without window inference. Bottom: development set, 15 instances per n_1, mean gain over LB^* in % (instances raised). Nominal budget 120 s of one CPU thread, checked only between trial tests; the 600 s column is an exploratory run on one machine. Intervals are 95% bootstrap intervals. Supplement S5.2 gives the full table.

Proven optima, 120 s nominal

Subset	n	LB^* dist.	Prev. best closed	UPB-W closed	UPB-W dist.	= OPT (prev./UPB-W)
All	98	5.48	61.0 [57.3, 64.6]	73.9 [70.2, 77.4]	1.43	7 / 11
$n_1 = 20$	77	6.05	64.9	75.8	1.52	7 / 11
$n_1 = 30$	21	3.42	47.3	67.0	1.11	0 / 0
$m = 2$	36	4.21	57.6	78.0	0.88	0 / 1
$m = 10$	24	8.74	62.9	69.2	2.54	4 / 5

Development set, gain over LB^* *(%)*

n_1	n	Prev. best	UPB-W, 120 s	UPB-W, 600 s
20–30	30	3.16	4.67 (29)	–
60	15	0	0.24 [0.09, 0.41] (10)	1.12 [0.89, 1.37] (15)
70	15	0	0 (0)	0.98 [0.76, 1.22] (15)
80	15	0	0 (0)	0.39 [0.18, 0.64] (11)
All	105	1.01	1.71 [1.23, 2.21] (61)	–

8.4. Upper bounds and certificates

Matched comparisons. Table 1 (families A–E) gives the confirmatory comparisons. At equal evaluations, which do not depend on the implementation, FB-ILS improves on EACDC by 0.63% [0.53, 0.72] and on EACDCml by 0.63% [0.53, 0.73]. The base ILS, which is written in Python like the rivals, improves on EACDC by 0.48% at equal generation evaluations and on HgR by 2.54% at equal CPU time. At equal CPU time, FB-ILS improves on EACDC by 1.09% and on EACDCml by 1.15%. These figures compare compiled with interpreted code. Most of the extra gain of FB-ILS over the base ILS at equal CPU comes from the compiled decoder: the ablation gives 0.57% at equal CPU (E5) but only 0.14% at equal evaluations (E6). These 0.14% are the part due to the two search mechanisms. Against EACDCml (paper-based) the margins are larger (1.60% at equal evaluations), but the main claims use the rebuild on the authors' code, the stronger rival. The margins shrink with the instance size but stay negative

in every size class, and they hold against EACDCml (paper-based) recalibrated with ten times the budget (Supplement S7.4). In a further preregistered test at 1, 10 and 100 times the rivals' evaluation counts, FB-ILS stayed better than EACDC with restarts (−0.75%, −0.63%, −0.67%), than EACDCml with scaled generations (−0.71% to −0.88%) and than HgR (−2.16% to −2.84%), so its advantage is not only faster convergence. An iterated greedy tuned on the development set beats EACDC from ten times the budget on, but remains 0.3–0.6% behind FB-ILS (Supplement S8). This test is a separate preregistered study with its own Holm family of 15 endpoints, reported in full in Supplement S8; 14 of them were confirmed, and the one that was not is the tuned iterated greedy against EACDC at the original budget. On the development set, the memetic combination of EACDC and the local search did not beat the local search alone (Supplement S7).

New best-known bounds and certificates. We ran FB-ILS once on every benchmark instance with 10 s of CPU. It improves the published HgR upper bound on 10,475 instances (99.8%), ties on 21 and is worse on 4. On average it is 3.06% below the published value. The HgR values are the only per-instance upper bounds published for the benchmark. Taking for each instance the best known upper bound and the best proven lower bound, the certified ratio UB/LB has mean 1.060 and median 1.048. It is within 1.05 on 53.1% of the instances, against 14.1% with the published HgR upper bounds and LB^*, and equal to 1 on 188. The per-instance table of best-known bounds is part of the data release. In an exploratory rerun outside the protocol (60 s of CPU, at most two further seeds), FB-ILS also found schedules below the published HgR bound on the 4 instances where the census run was worse; the release contains these schedules, and the statistics above do not use them.

8.5. Structured and larger instances

On the 240 structured instances (family F of Table 1), LB^* is 3.45% [2.79, 4.15] above the best earlier bound and 3.16% [2.50, 3.85] above energetic reasoning. The gain is much smaller than on the benchmark and depends on the structure. With random precedence of density 0.2–0.3 it reaches 5.9–12.8% at $n_1 = 60$–100 and 8–10% at $n_1 = 150$–300. With density 0.1 it is below 1%, and with block precedence 0–3.7%. When the predecessor sets are small or nearly disjoint, the union argument adds little to the head sums, and energetic reasoning is sometimes higher. The bound took 0.5 s per direction at $n_1 = 30$ and 125 s at $n_1 = 300$. On the 58 matched instances, the base ILS is 0.85% below EACDC at equal CPU and 0.75% at equal generation evaluations, more than on the benchmark. An exploratory analysis in Supplement S10 shows that the margin over EACDC grows with the certified headroom $(\mathrm{UB}_{\mathrm{EACDC}} - \mathrm{LB}^*)/\mathrm{LB}^*$ and that the bound predicts where search pays better than instance features do.

9. Discussion

Why the union matters. Energetic and head-and-tail bounds reason about the time windows of single jobs. In flow shops this loses little, because each operation has one predecessor. In a cross-dock, whether a set of outbound trucks can start early depends on the total work of the union of their predecessors, which no single window describes. The data agree with this

explanation. The gain of UPB over the earlier bounds grows with n_1, from 5% to 22%, while the energetic bound gains about 1% at every size. When predecessor sets are small or nearly disjoint, as with sparse or block precedence, or when there are many doors and few trucks, the coupling matters less and the gain shrinks.

What the bound does not do. The bound is far from tight: it equals the optimum on 3 of 165 solved instances. The fluid relaxation separates two causes. With few doors, the bound may pick an unrelated set of outbound trucks at each time, while a schedule needs one nested sequence of sets. With many doors, the slack comes from the small number of trucks per door. A bound that keeps one inbound order for all times, and one that counts trucks per door, are therefore the two natural next steps. UPB-W is a step of the second kind: it counts trucks per door through shaving and door caps, and it closes 74% of the slack where the optimum is known.

Practical use. Dividing the makespan of any plan by LB* bounds how far the plan can be from the best possible one, at a cost of milliseconds to seconds. With the best bounds now known, 53.1% of the benchmark instances are certified within 5%. A planner who sees a certified gap of a few percent knows that little can be gained by searching further. At small terminals, most of the remaining gap comes from the bound, so extra computation is better spent on proving the plan good, for example with the union cuts, than on improving it. Where doors are many, energetic reasoning is sometimes stronger, and planners should take the maximum of both bounds. The bound also ranks door investments without any schedule. On 350 benchmark instances, adding one door on the side that most reduces LB* agreed with the choice made from schedules on 93% of the 335 instances where the two options differ (Supplement S10).

Scope and limits. All multi-door results except the structured set use the benchmark's random instances, in which each inbound–outbound pair is a predecessor pair with probability one half; on sparse and block-structured instances the gain of the bound is 3.4%. Exact optima are known only for the easier part of the instances with $n_1 \leq 30$. UPB-W was evaluated on 98 optima and 105 development instances, not on the whole benchmark; at $n_1 \geq 70$ its gain depends on the budget, and its CPU use can exceed the nominal budget by about a factor of two. The exact-method design was developed on the exact set; the preregistered confirmation on 300 new instances covers only $n_1 \leq 30$ and ran on two machines. The upper-bound gain is small: about 0.6% at equal evaluations. EACDCml and HgR are compared through our rebuild and re-implementation, because the trained network and the HgR model file are not public; both reproduce the published results within sampling error. The CPU-matched comparisons mix algorithm and implementation. The constant $3 - 1/M$ is classical, and its tightness on two-level instances is open. Our results say nothing about budgets much longer than those of the rival runs. Supplement S12 lists these limits in full.

10. Conclusion

We studied makespan scheduling at a cross-docking terminal with several doors on each side, where every outbound truck waits for a set of inbound trucks. The main contribution is a lower

bound that asks which outbound trucks can have started by each time, which requires a knapsack over the union of their predecessors. On the public benchmark this bound is 15.76% above the best earlier bound, and a preregistered test on new instances confirmed a gain of 14.90%. Written as cuts in a constraint-programming solver, the same argument proves optimality on 137 of 300 new instances instead of 47 in a preregistered test. A second bound that infers time windows closes about three quarters of the remaining distance to the optimum where the optimum is known. Together with the schedules of a simple local search, the bound certifies more than half of the benchmark instances within 5%.

Three questions remain open. Can a bound that keeps one inbound order for all times close the nestedness slack? How can the bound be strengthened on sparse and block-structured precedence, where its gain is small? And can the worst-case constant against the bound be lowered for the two-level precedence of a cross-dock?

Supplementary material

The online supplement contains the long proofs, the theory extensions, the details of the bound computation, the full upper-bound benchmarking, the exploratory analyses, the reproducibility notes and the full list of limitations. It is cited as "Supplement Sx".

Data and code availability

The code, the per-instance bounds and best schedules, the generated instances, the SHA-256 digests of the preregistration protocols and the scripts that produce every table are deposited at Zenodo (Provezano Coutinho, 2026), DOI 10.5281/zenodo.23933868; the code is also on GitHub (github.com/provezano/crossdock-union-bounds). The protocol documents are available from the author on request and can be checked against the deposited digests.

Declaration of competing interest

The author declares no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Funding

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Declaration of generative AI and AI-assisted technologies in the manuscript preparation process

During the preparation of this work the author used Claude (Anthropic) to generate research code and to draft and edit parts of the manuscript text. All code and text were subsequently reviewed in full by the author, and edited as needed. The author takes full responsibility for the content of the publication.

CRediT authorship contribution statement

Felipe Provezano Coutinho: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Writing – original draft, Writing – review & editing, Visualization.

References

Andrade, C.E., Toso, R.F., Gonçalves, J.F., Resende, M.G.C., 2021. The multi-parent biased random-key genetic algorithm with implicit path-relinking and its real-world applications. European Journal of Operational Research 289, 17–30. doi:10.1016/j.ejor.2019.11.037.

Baptiste, P., Le Pape, C., Nuijten, W., 1999. Satisfiability tests and time-bound adjustments for cumulative scheduling problems. Annals of Operations Research 92, 305–333. doi:10.1023/A:1018995000688.

Baptiste, P., Le Pape, C., Nuijten, W., 2001. Constraint-Based Scheduling. International Series in Operations Research & Management Science, Springer US, Boston, MA. doi:10.1007/978-1-4615-1479-4.

Boysen, N., Fliedner, M., Scholl, A., 2010. Scheduling inbound and outbound trucks at cross docking terminals. OR Spectrum 32, 135–161. doi:10.1007/s00291-008-0139-2.

Brucker, P., Knust, S., 2000. A linear programming and constraint propagation-based lower bound for the RCPSP. European Journal of Operational Research 127, 355–362. doi:10.1016/S0377-2217(99)00489-0.

Carlier, J., Néron, E., 2000. An exact method for solving the multi-processor flow-shop. RAIRO – Operations Research 34, 1–25. doi:10.1051/ro:2000103.

Carlier, J., Néron, E., 2007. Computing redundant resources for the resource constrained project scheduling problem. European Journal of Operational Research 176, 1452–1463. doi:10.1016/j.ejor.2005.09.034.

Chen, F., Lee, C.Y., 2009. Minimizing the makespan in a two-machine cross-docking flow shop problem. European Journal of Operational Research 193, 59–72. doi:10.1016/j.ejor.2007.10.051.

Chen, F., Song, K., 2009. Minimizing makespan in two-stage hybrid cross docking scheduling problem. Computers & Operations Research 36, 2066–2073. doi:10.1016/j.cor.2008.07.003.

Clautiaux, F., Alves, C., Valério de Carvalho, J., 2010. A survey of dual-feasible and superadditive functions. Annals of Operations Research 179, 317–342. doi:10.1007/s10479-008-0453-8.

Cota, P.M., Gimenez, B.M.R., Araújo, D.P.M., Nogueira, T.H., de Souza, M.C., Ravetti, M.G., 2016. Time-indexed formulation and polynomial time heuristic for a multi-dock truck scheduling problem in a cross-docking centre. Computers & Industrial Engineering 95, 135–143. doi:10.1016/j.cie.2016.03.001.

Dulebenets, M.A., 2021. An adaptive polyploid memetic algorithm for scheduling trucks at a cross-docking terminal. Information Sciences 565, 390–421. doi:10.1016/j.ins.2021.02.039.

Eti, N.A., Ferrell, W.G., Huynh, N., 2025. Multi-door cross-dock scheduling under flexible doors mode and material handling resource restrictions. Computers & Operations Research 183, 107183. doi:10.1016/j.cor.2025.107183.

Fekete, S.P., Schepers, J., 2001. New classes of fast lower bounds for bin packing problems. Mathematical Programming 91, 11–31. doi:10.1007/s101070100243.

Fernandez, E.B., Bussell, B., 1973. Bounds on the number of processors and time for multiprocessor optimal schedules. IEEE Transactions on Computers C-22, 745–751. doi:10.1109/TC.1973.5009153.

Fonseca, G.B., 2019. Cross-docking problems: instances, scripts and per-instance results. https://github.com/GabrielaBragaFonseca/Cross-docking-Problems. Accessed 2026-11-01.

Fonseca, G.B., Nogueira, T.H., Ravetti, M.G., 2019. A hybrid Lagrangian metaheuristic for the cross-docking flow shop scheduling problem. European Journal of Operational Research 275, 139–154. doi:10.1016/j.ejor.2018.11.033. arXiv:1702.05603.

Gallo, G., Grigoriadis, M.D., Tarjan, R.E., 1989. A fast parametric maximum flow algorithm and applications. SIAM Journal on Computing 18, 30–55. doi:10.1137/0218003.

Gaudioso, M., Monaco, M.F., Sammarra, M., 2021. A Lagrangian heuristics for the truck scheduling problem in multi-door, multi-product cross-docking with constant processing time. Omega 101, 102255. doi:10.1016/j.omega.2020.102255.

Golias, M.M., Saharidis, G.K.D., Ivey, S., Haralambides, H.E., 2013. Advances in truck scheduling at a cross dock facility. International Journal of Information Systems and Supply Chain Management 6, 40–62. doi:10.4018/ijisscm.2013070102.

Golshahi-Roudbaneh, A., Hajiaghaei-Keshteli, M., Paydar, M.M., 2017. Developing a lower bound and strong heuristics for a truck scheduling problem in a cross-docking center. Knowledge-Based Systems 129, 17–38. doi:10.1016/j.knosys.2017.05.006.

Graham, R.L., 1969. Bounds on multiprocessing timing anomalies. SIAM Journal on Applied Mathematics 17, 416–429. doi:10.1137/0117039.

Haouari, M., Hidri, L., Gharbi, A., 2006. Optimal scheduling of a two-stage hybrid flow shop. Mathematical Methods of Operations Research 64, 107–124. doi:10.1007/s00186-006-0066-4.

Haouari, M., Kooli, A., Néron, E., 2012. Enhanced energetic reasoning-based lower bounds for the resource constrained project scheduling problem. Computers & Operations Research 39, 1187–1194. doi:10.1016/j.cor.2011.05.022.

Hariri, A.M.A., Potts, C.N., 1997. A branch and bound algorithm for the two-stage assembly scheduling problem. European Journal of Operational Research 103, 547–556. doi:10.1016/S0377-2217(96)00312-8.

Hidri, L., Haouari, M., 2011. Bounding strategies for the hybrid flow shop scheduling problem. Applied Mathematics and Computation 217, 8248–8263. doi:10.1016/j.amc.2011.02.108.

Horn, W.A., 1974. Some simple scheduling algorithms. Naval Research Logistics Quarterly 21, 177–185. doi:10.1002/nav.3800210113.

Jaffe, J.M., 1980. Bounds on the scheduling of typed task systems. SIAM Journal on Computing 9, 541–551. doi:10.1137/0209040.

Johnson, D.S., Niemi, K.A., 1983. On knapsacks, partitions, and a new dynamic programming technique for trees. Mathematics of Operations Research 8, 1–14. doi:10.1287/moor.8.1.1.

Klein, R., Scholl, A., 1999. Computing lower bounds by destructive improvement: An application to resource-constrained project scheduling. European Journal of Operational Research 112, 322–346. doi:10.1016/S0377-2217(97)00442-6.

Kolliopoulos, S.G., Steiner, G., 2007. Partially ordered knapsack and applications to scheduling. Discrete Applied Mathematics 155, 889–897. doi:10.1016/j.dam.2006.08.006.

Ladier, A.L., Alpan, G., 2016. Cross-docking operations: Current research versus industry practice. Omega 62, 145–162. doi:10.1016/j.omega.2015.09.006.

Lahimer, A., Lopez, P., Haouari, M., 2013. Improved bounds for hybrid flow shop scheduling with multiprocessor tasks. Computers & Industrial Engineering 66, 1106–1114. doi:10.1016/j.cie.2013.08.004.

Lee, J.Y., Jeon, I.C., Han, J.H., 2025. Scheduling for two stages with a single machine and a job shop. Computers & Operations Research 183, 107167. doi:10.1016/j.cor.2025.107167.

Li, B., Borowska-Stefańska, M., Wiśniewski, S., Guo, Q., Lau, Y.y., Dulebenets, M.A., 2026a. A hybridized Thompson Sampling hyperheuristic for the truck scheduling problem at a cross-docking facility. Swarm and Evolutionary Computation 107, 102502. doi:10.1016/j.swevo.2026.102502.

Li, B., Moses, R., Ozguven, E.E., Ozkul, S., Fathollahi-Fard, A.M., Dulebenets, M.A., 2026b. A sequential fix-and-optimize framework and pricing-guided search for efficient truck scheduling at a cross-docking terminal. Transportation Research Part E: Logistics and Transportation Review 216, 105264. doi:10.1016/j.tre.2026.105264.

Li, K.Y., Willis, R.J., 1992. An iterative scheduling technique for resource-constrained project scheduling. European Journal of Operational Research 56, 370–379. doi:10.1016/0377-2217(92)90320-9.

Li, Y., Mohammadi, M., Zhang, X., Lan, Y., van Jaarsveld, W., 2026c. Integrated trucks assignment and scheduling problem with mixed service mode docks: A Q-learning based ALNS. European Journal of Operational Research 333, 117–137. doi:10.1016/j.ejor.2025.12.036.

Liu, J.W.S., Liu, C.L., 1978. Performance analysis of multiprocessor systems containing functionally dedicated processors. Acta Informatica 10, 95–104. doi:10.1007/BF00260927.

Liu, J.W.S., Liu, C.L., Miyazaki, N., Yamazaki, H., 1981. Performance evaluation of multiprocessor systems containing special purpose processors. DTIC report ADA102227. University of Illinois. URL: https://web.archive.org/web/20170301014705/http://www.dtic.mil/dtic/tr/fulltext/u2/a102227.pdf.

Lourenço, H.R., Martin, O.C., Stützle, T., 2003. Iterated local search, in: Glover, F., Kochenberger, G.A. (Eds.), Handbook of Metaheuristics. Kluwer Academic Publishers, Boston, MA. volume 57 of *International Series in Operations Research & Management Science*, pp. 320–353. doi:10.1007/0-306-48056-5_11.

Manurangsi, P., 2017. Almost-polynomial ratio ETH-hardness of approximating densest k-subgraph, in: Proceedings of the 49th Annual ACM Symposium on Theory of Computing (STOC 2017), pp. 954–961. doi:10.1145/3055399.3055412.

Neamatian Monemi, R., Gelareh, S., 2023. Dock assignment and truck scheduling problem; consideration of multiple scenarios with resource allocation constraints. Computers & Operations Research 151, 106074. doi:10.1016/j.cor.2022.106074.

Néron, E., Baptiste, P., Gupta, J.N.D., 2001. Solving hybrid flow shop problem using energetic reasoning and global operations. Omega 29, 501–511. doi:10.1016/S0305-0483(01)00040-8.

Nogueira, T.H., Coutinho, F.P., Peixoto, M.G.M., Carrano, E.G., Ravetti, M.G., 2025. Evolutionary algorithm framework for optimizing truck scheduling in multi-dock truck cross-docking centers. Evolutionary Intelligence 18, 2. doi:10.1007/s12065-024-00992-x.

Nogueira, T.H., Coutinho, F.P., Ribeiro, R.P., Ravetti, M.G., 2020. Parallel-machine scheduling methodology for a multi-dock truck sequencing problem in a cross-docking center. Computers & Industrial Engineering 143, 106391. doi:10.1016/j.cie.2020.106391.

Nogueira, T.H., et al., 2022. EA-ML: public code of the EACDC/EACDCml evolutionary algorithm. https://github.com/yotitinogs/EA-ML. Accessed 2026-11-01.

Ojha, R., Erera, A., 2025. Cross-dock trailer scheduling with workforce constraints: A dynamic discretization discovery approach. Transportation Science 59, 165–186. doi:10.1287/trsc.2023.0406.

PassMark Software, 2026. CPU benchmarks: Single thread performance. https://www.cpubenchmark.net/singleThread.html. Apple M5 Pro 15-core 5,898; Apple M3 Max 16-core 4,781. Accessed 2 November 2026.

Picard, J.C., 1976. Maximal closure of a graph and applications to combinatorial problems. Management Science 22, 1268–1272. doi:10.1287/mnsc.22.11.1268.

Potts, C.N., Sevast'janov, S.V., Strusevich, V.A., Van Wassenhove, L.N., Zwaneveld, C.M., 1995. The two-stage assembly scheduling problem: Complexity and approximation. Operations Research 43, 346–355. doi:10.1287/opre.43.2.346.

Prot, D., Rapine, C., 2013. Approximations for the two-machine cross-docking flow shop problem. Discrete Applied Mathematics 161, 2107–2119. doi:10.1016/j.dam.2013.02.030.

Provezano Coutinho, F., 2026. Code and data for: Union-of-predecessors bounds and a forward–backward local search for multi-door cross-dock truck scheduling. Zenodo, version 1.0.0 [dataset]. URL: https://doi.org/10.5281/zenodo.23933868, doi:10.5281/zenodo.23933868.

Rijal, A., Bijvank, M., de Koster, R., 2019. Integrated scheduling and assignment of trucks at unit-load cross-dock terminals with mixed service mode dock doors. European Journal of Operational Research 278, 752–771. doi:10.1016/j.ejor.2019.04.028.

Ruiz, R., Stützle, T., 2007. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European Journal of Operational Research 177, 2033–2049. doi:10.1016/j.ejor.2005.12.009.

Tadumadze, G., Boysen, N., Emde, S., Weidinger, F., 2019. Integrated truck and workforce scheduling to accelerate the unloading of trucks. European Journal of Operational Research 278, 343–362. doi:10.1016/j.ejor.2019.04.024.

Valls, V., Ballestín, F., Quintanilla, S., 2005. Justification and RCPSP: A technique that pays. European Journal of Operational Research 165, 375–386. doi:10.1016/j.ejor.2004.04.008.

Vandevelde, A., Hoogeveen, H., Hurkens, C., Lenstra, J.K., 2005. Lower bounds for the head-body-tail problem on parallel machines: A computational study of the multiprocessor flow shop. INFORMS Journal on Computing 17, 305–320. doi:10.1287/ijoc.1040.0082.

Wallrath, R., Franke, M., Walter, M., 2025. Improved linear programming relaxations for flow shop problems with makespan minimization. Computers & Operations Research 177, 106970. doi:10.1016/j.cor.2024.106970.

Online supplement

This supplement contains the material that the main text summarises: the long proofs and extensions of the bound theory (S1–S3), the computation of the bound (S4), the window-inference bound in full (S5), the comparison with other bounding methods (S6), the upper-bound method and its benchmarking against EACDC, EACDCml and HgR (S7), details of the anytime comparison of the upper-bound methods at larger budgets (S8), details of the exact method (S9), exploratory analyses (S10), reproducibility notes (S11) and the full list of limitations (S12). Numbers without the prefix S (sections, theorems, tables) refer to the main text.

S1. Proofs and extensions of the bound theory

S1.1. Proof of the breakpoint form

Proof of Theorem 2. Write $x(t) = K(m_1 t)$ and $y(t) = D(t)$. Since $P_2 - \min(x, y) = \max(P_2 - x, P_2 - y)$, we have $f(t) = \max(g_1(t), g_2(t))$ with $g_1(t) = t + (P_2 - x(t))/m_2$ and $g_2(t) = t + (P_2 - y(t))/m_2$. A time t is *admissible* if $\min(x(t), y(t)) < P_2$.

The term g_1. On admissible times with $x(t) = P_2$, we have $y(t) < P_2$ and $g_1(t) = t < g_2(t)$, so g_1 does not decide f there. Every time with $x(t) < P_2$, that is with $m_1 t < c_{\text{full}}$, is admissible. The function K is non-decreasing, right-continuous and piecewise constant, and it jumps only at points of Λ. Let $0 = c_0 < c_1 < \cdots < c_r = c_{\text{full}}$ be the points of $\Lambda \cup \{0\}$ up to c_{full}. On $[c_k/m_1, c_{k+1}/m_1)$ the value $K(m_1 t) = K(c_k)$ is constant and g_1 increases with t. Its supremum on this interval is the left limit $c_{k+1}/m_1 + (P_2 - K(c_k))/m_2$, and $K(c_k) = K(c_{k+1}^-)$. So the supremum of g_1 over the times with $x(t) < P_2$ is the inner maximum G_1 of the theorem, and $G_1 \le \text{LB}_U$.

The term g_2. The slope of g_2 is $1 - k(t)/m_2$, where $k(t) \le \min(m_2, n_2)$ is the number of the $\min(m_2, n_2)$ smallest heads below t. If $n_2 \ge m_2$, then g_2 does not decrease until $t = h_{(m_2)}$ and is constant afterwards, with value L_{CS}, so $g_2 \le L_{\text{CS}}$ everywhere. If $n_2 < m_2$, the slope stays positive, so g_2 increases for ever. On the times with $y(t) < P_2$ its supremum is the time t^* with $D(t^*) = P_2$. If every outbound truck j starts at h_j on its own door, all outbound work is done by $H = \max_j(h_j + p_{2j})$, so $D(H) \ge P_2$ and $t^* \le H$. In both cases, the supremum of g_2 over admissible times is at most $\max(L_{\text{CS}}, H)$.

Lower side. If $n_2 \ge m_2$ and $y(h_{(m_2)}) < P_2$, then $t = h_{(m_2)}$ is admissible and $f(t) \ge g_2(t) = L_{\text{CS}}$, so $\text{LB}_U \ge L_{\text{CS}}$. Otherwise $L_{\text{CS}} = g_2(h_{(\min(m_2,n_2))})$ is at most the supremum of g_2 over the times before y reaches P_2, or at most $h_{(\min(m_2,n_2))} \le H$; in both cases $L_{\text{CS}} \le \max(\text{LB}_U, H)$.

Combining the three parts, $\max(\text{LB}_U, H) = \max(L_{\text{CS}}, H, G_1)$. If $n_2 \ge m_2$ and $y(h_{(m_2)}) < P_2$, the lower side gives $\text{LB}_U \ge L_{\text{CS}}$ and the g_2 part gives $\sup g_2 \le L_{\text{CS}}$, so $\text{LB}_U = \max(L_{\text{CS}}, G_1)$ and H can be dropped. □

Inverse form. For integer p_2, let $c^*(v) = \min\{p_1(N(A)) : p_2(A) \ge v\}$ be the cheapest predecessor load that reaches outbound workload v. Then $G_1 = \max_{1 \le v \le P_2,\ c^*(v) > 0} \big[c^*(v)/m_1 + (P_2 - v + 1)/m_2\big]$. Indeed, let $c \in \Lambda$ be a jump with $K(c^-) = v'$. Every $v \in (v', K(c)]$ has $c^*(v) = c$, and the largest term among these values is the one with $v = v' + 1$, which equals the term of c in G_1. Values of v that are not of this form give terms that are no larger than the term of the jump that covers them.

Why the term H is needed. Without the term H, the breakpoint form is exact only when $n_2 \ge m_2$ and $D(h_{(m_2)}) < P_2$. For $n_2 < m_2$ the door-cap part rises above L_{CS} (its supremum is t^*), so a formula with L_{CS} alone can understate LB_U. This cannot happen on the benchmark ($n_2 \ge 16 > 10 \ge m_2$), and it never affects validity. It can occur in the structured families with strongly unbalanced doors and small n_1; since $H \le \text{LB}_{e2}$ is a component of UPB, the computed bound is unaffected.

S1.2. The classical bounds and a coupled version of the union bound

The bounds of Section 4.3 are, in full:

- the *load bound* $\mathrm{LB}_{\mathrm{load}} = \max(a + \underline{q},\ b)$, where $\underline{q} = \min_j p_{2j}$ if every inbound truck has a successor and $\underline{q} = 0$ otherwise;
- the *head bound* on stage 2,

$$\mathrm{LB}_{e2} = \max\Big(\max_k \Big[h_{(k)} + \sum_{l:\, h_l \geq h_{(k)}} \frac{p_{2l}}{m_2}\Big],\ \max_j (h_j + p_{2j})\Big),$$

  where $h_{(k)}$ is the k-th smallest head. It belongs to the head-and-tail family of Carlier and Néron (2000);
- the *PCH bound* (Nogueira et al., 2020). Each inbound truck i receives the tail $\Delta_i = \max\big(p_2(S(i))/m_2,\ \max_{j\in S(i)} p_{2j}\big)$, with $\Delta_i = 0$ if $S(i) = \emptyset$. Let T_k be the set of the k inbound trucks with the largest tails and $\Delta_{(k)}$ the k-th largest tail. Then

$$\mathrm{LB}_{\mathrm{PCH}} = \max\Big(\max_k \Big[\frac{p_1(T_k)}{m_1} + \Delta_{(k)}\Big],\ \max_i (p_{1i} + \Delta_i)\Big).$$

  We use this form, which follows Algorithms 1 and 2 of Nogueira et al. (2020).

Proposition S1. LB_U *dominates none of* $\mathrm{LB}_{\mathrm{load}}$, LB_{e2} *and* $\mathrm{LB}_{\mathrm{PCH}}$.

Proof. Two instances suffice. (a) Let $m_1 = 1$, $m_2 = 2$, $p_1 = (5, 5)$, $p_2 = (4, 4)$ and $N(j_k) = \{i_k\}$ for $k = 1, 2$. Then $\mathrm{OPT} = \mathrm{LB}_{\mathrm{load}} = 14$, while $\mathrm{LB}_U = 12$, because LB_U spreads the last outbound truck over both outbound doors. (b) Let $m_1 = 2$, $m_2 = 1$, $p_1 = (10, 1)$, $p_2 = (1, 1)$ and $N(j_k) = \{i_k\}$. Then $\mathrm{OPT} = \mathrm{LB}_{e2} = \mathrm{LB}_{\mathrm{PCH}} = 11$, while $\mathrm{LB}_U = 6.5$, because LB_U treats the long inbound truck as if its work could be split over two doors. □

Both failures come from the same source. LB_U ignores that a truck cannot be split. The refined bound adds this information. For a time t, call a set $A \subseteq J_2$ *head-feasible* if $p_1(N(A)) \leq m_1 t$ and $\max_{i\in N(A)} p_{1i} \leq t$, and let $\mathcal{F}(t)$ be the family of these sets. Define

$$\Phi(t) = \min_{A\in\mathcal{F}(t)} \max\Big(\frac{P_2 - \min\big(p_2(A),\ D(t)\big)}{m_2},\ \max_{j\notin A} p_{2j}\Big),$$

with the maximum over an empty set equal to zero, and

$$\mathrm{LB}_U^{\Phi} = \max\big(a,\ p,\ q,\ \sup\{t + \Phi(t) : \Phi(t) > 0\}\big).$$

This is the coupled bound of Remark 2 in the main text. The bound UPB that we compute is the maximum of its components (Section 4.3). The first term inside Φ is the remaining outbound work spread over the doors, as in LB_U. The second term is the longest outbound truck that has not started. The two terms are evaluated on the *same* set A; this coupling is what makes the dominance proof below work.

Theorem S1 (Validity and dominance of the coupled bound)**.** LB_U^{Φ} *is a valid lower bound, and* $\mathrm{LB}_U^{\Phi} \geq \max(\mathrm{LB}_U, \mathrm{LB}_{\mathrm{load}}, \mathrm{LB}_{e2}, \mathrm{LB}_{\mathrm{PCH}})$.

Proof. Validity. The terms a, p and q are clearly valid. Let S be optimal with makespan C, fix t, and let A_t be as in Lemma 1. By part (i), $A_t \in \mathcal{F}(t)$. If $C \geq t$, parts (ii) and (iii) give $C \geq t + \max(\text{both terms at } A_t) \geq t + \Phi(t)$. If $C < t$, then $A_t = J_2$, all outbound work is done by t, so $D(t) \geq P_2$ by part (ii), and both terms vanish at $A = J_2 \in \mathcal{F}(t)$. Hence $\Phi(t) = 0$. So $\Phi(t) > 0$ implies $C \geq t$ and therefore $C \geq t + \Phi(t)$.

LB_U. Since $\mathcal{F}(t) \subseteq \{A : p_1(N(A)) \le m_1 t\}$ and the maximum only adds a term, $\Phi(t) \ge (P_2 - \overline{W}(t))/m_2$. Whenever $\overline{W}(t) < P_2$ this is positive, so $t + \Phi(t) \ge f(t)$.

$\mathrm{LB}_{\mathrm{load}}$. The term b is at most $L_{\mathrm{CS}} \le \mathrm{LB}_U^{\Phi}$ (Proposition S2 below). Suppose every inbound truck has a successor and let $t < a$. Every $A \in \mathcal{F}(t)$ has $p_1(N(A)) \le m_1 t < P_1$, so some inbound truck i is not in $N(A)$. Its successors are then not in A, so the second term is at least $\min_j p_{2j}$. Letting $t \uparrow a$ gives $a + \min_j p_{2j}$.

LB_{e2}. Fix $\tau = h_{(k)} > 0$ and let $t < \tau$. A truck l with $h_l \ge \tau > t$ has either $p_1(N(l)) > m_1 t$ or a predecessor longer than t, so no $A \in \mathcal{F}(t)$ contains it. Hence $p_2(A) \le P_2 - \sum_{l:h_l \ge \tau} p_{2l}$, and the first term is at least $\sum_{l:h_l \ge \tau} p_{2l}/m_2$. Letting $t \uparrow \tau$ gives the first part of LB_{e2}. For the second part, a truck j with $h_j > 0$ is not in any $A \in \mathcal{F}(t)$ when $t < h_j$, so $\Phi(t) \ge p_{2j}$ and $t \uparrow h_j$ gives $h_j + p_{2j}$. If $h_j = 0$, the term p_{2j} is at most q.

$\mathrm{LB}_{\mathrm{PCH}}$. Let $t < p_1(T_k)/m_1$. Every $A \in \mathcal{F}(t)$ has $p_1(N(A)) < p_1(T_k)$, so some $i \in T_k$ is not in $N(A)$, and then $S(i) \cap A = \emptyset$. For this same set A, the first term is at least $p_2(S(i))/m_2$ and the second term is at least $\max_{j \in S(i)} p_{2j}$. The maximum of the two terms is therefore at least $\Delta_i \ge \Delta_{(k)}$. Since this holds for every $A \in \mathcal{F}(t)$, $\Phi(t) \ge \Delta_{(k)}$, and $t \uparrow p_1(T_k)/m_1$ gives the first part of $\mathrm{LB}_{\mathrm{PCH}}$. The term $p_{1i} + \Delta_i$ follows in the same way with $t \uparrow p_{1i}$, because then no head-feasible set can contain a successor of i. If we had minimised the two terms of Φ separately, we would only obtain $\max(\min_A \cdot, \min_A \cdot)$, which can be smaller than $\Delta_{(k)}$. □

Proposition S2. $\mathrm{LB}_U^{\Phi} \ge L_{\mathrm{CS}}$, *where* $L_{\mathrm{CS}} = (\sum_{k \le \min(m_2, n_2)} h_{(k)} + P_2)/m_2$ *is the bound of Chen and Song (2009). Hence* LB^* *dominates their bound* $\max(L_{\mathrm{CS}}(I), L_{\mathrm{CS}}(R(I)))$.

Proof. If $n_2 < m_2$, then $L_{\mathrm{CS}} \le \max_j(h_j + p_{2j}) \le \mathrm{LB}_{e2} \le \mathrm{LB}_U^{\Phi}$, by the LB_{e2} part of the proof of Theorem S1, which does not use this proposition. Otherwise let $\tau = h_{(m_2)}$. For every set A, the first term of $\Phi(\tau)$ is at least $(P_2 - D(\tau))/m_2$, and $\tau + (P_2 - D(\tau))/m_2 = L_{\mathrm{CS}}$. If $\Phi(\tau) > 0$, then $\mathrm{LB}_U^{\Phi} \ge \tau + \Phi(\tau) \ge L_{\mathrm{CS}}$. If $\Phi(\tau) = 0$, then $D(\tau) \ge P_2$, so $L_{\mathrm{CS}} \le \tau \le h_j + p_{2j} \le \mathrm{LB}_U^{\Phi}$ for the truck j with head τ. The reversed case is the same argument on $R(I)$. □

The door cap $D(t)$ is needed: with the simpler cap $m_2(t - r_{\min})^+$ the bound does not dominate L_{CS}, as the counterexample after Corollary 1 shows. On the 3,000 benchmark instances with $n_1 \le 30$, the head-sum part decides the bound on 26% of them.

S1.3. Small-instance checks

We checked Theorems 1, 2 and S1 by brute force on 1,200 tiny random instances with six precedence structures ($n \le 10$), with no violation. On 15,000 further tiny instances, LB_U^{Φ} was strictly larger than $\max(\mathrm{LB}_U, \mathrm{LB}_{\mathrm{PCH}}, \mathrm{LB}_{e2}, \mathrm{LB}_{\mathrm{load}}, q)$ in only one. In practice, therefore, LB_U^{Φ} equals the maximum of LB_U and the classical bounds, which is UPB and is what we compute.

S2. Complexity of the knapsack subproblem

The bound needs $K(c)$ at several capacities. In general this is hard, but the structure of the predecessor sets decides how hard.

Proposition S3 (Johnson and Niemi, 1983)**.** *Computing $K(c)$ is strongly NP-hard, even when all processing times are 1 and every outbound truck has exactly two predecessors. In that case it admits no fully polynomial approximation scheme unless P = NP, and it is as hard to approximate as the densest k-subgraph problem.*

Proof. Given a graph $G = (V, E)$ and an integer k, create one inbound truck per vertex and one outbound truck per edge, all with unit times, and let the predecessors of an edge be its two end vertices. Then $K(k)$ is the largest number of edges induced by k vertices. Deciding whether $K(k) \ge k(k-1)/2$ is the clique problem, so the problem is NP-hard, and strongly so because all numbers are 1. Since $K(k)$ is an integer at most n_2, an approximation scheme with error

$\varepsilon < 1/n_2$ would solve it exactly in polynomial time. The reduction preserves the objective of densest k-subgraph exactly, so its approximation hardness (Manurangsi, 2017) transfers. □

The result is classical. Johnson and Niemi (1983) proved strong NP-hardness of the partially ordered knapsack with the same two-level reduction from clique and noted that no fully polynomial approximation scheme exists, and Kolliopoulos and Steiner (2007) relate the problem to densest k-subgraph. The proof above only records the unit-time case in which every outbound truck has two predecessors. We do not know whether evaluating LB_U itself is NP-hard.

Proposition S4 (Tractable structures)**.** *$K(c)$ can be computed*

(a) in $O(n_1 n_2)$ time after sorting, if the predecessor sets are nested, $N(j_1) \subseteq \cdots \subseteq N(j_{n_2})$;

(b) by a 0/1 knapsack, with a pseudo-polynomial algorithm and an approximation scheme, if every outbound truck has one predecessor, or if the trucks form disjoint blocks in which every outbound truck needs all the inbound trucks of its block;

(c) in pseudo-polynomial time, with an approximation scheme, if the predecessor sets form a laminar family or are intervals of one fixed order of the inbound trucks.

Proof. Parts (a) and (b) are proved in full; for part (c) we describe the dynamic programs. Part (c) belongs to the family of dynamic programs for ordered knapsacks: Johnson and Niemi (1983) treat tree partial orders, and Kolliopoulos and Steiner (2007) give a pseudo-polynomial program and an approximation scheme for two-dimensional partial orders. We state the programs directly for the two cross-dock structures. (a) With nested sets, a set A has $N(A) = N(j_{\max A})$, where $j_{\max A}$ is its member with the largest index. Adding all j_l with $l \le \max A$ does not change the cost. The optimal sets are therefore prefixes $\{j_1, \ldots, j_k\}$, and $K(c)$ is the largest prefix workload with $p_1(N(j_k)) \le c$. Consequently LB_U has at most $n_2 + 1$ candidate values in Theorem 2. (b) If $|N(j)| = 1$ for all j, choosing an inbound truck i costs p_{1i} and makes all of $S(i)$ available, worth $p_2(S(i))$; distinct inbound trucks unlock disjoint sets. This is a 0/1 knapsack. Disjoint blocks are the same with blocks as items. (c) For a laminar family, form the forest of distinct predecessor sets ordered by inclusion. Give node u with set U_u the cost $p_1(U_u)$ minus the total cost of its children's sets, which are disjoint, and the value of the outbound trucks whose predecessor set is exactly U_u. Paying for all of U_u makes every truck whose predecessor set lies below u available, so an optimal choice is a union of whole subtrees. A dynamic program over the forest that either takes a whole subtree or merges the tables of the children by knapsack convolution solves the problem in pseudo-polynomial time; Johnson and Niemi (1983) give such programs for tree knapsack problems. For intervals of a fixed order, $N(A)$ is a union of disjoint runs of consecutive inbound trucks, and the value of a choice of runs is the workload of the trucks whose interval lies inside one run. A dynamic program that scans the order and decides where each run starts and ends, with the capacity used as the second index, runs in $O(n_1^2 c)$ time. In both cases a version of the program indexed by value instead of capacity, combined with the standard rounding of values, gives an approximation scheme. □

On the benchmark, the predecessor sets are random with density one half and have none of these structures.

Proposition S5 (LP relaxation)**.** *Let $\widehat{K}(c)$ be the optimal value of the LP $\max\{\sum_j p_{2j} x_j : 0 \le x_j \le y_i \le 1 \ (i \in N(j)),\ \sum_i p_{1i} y_i \le c\}$. Then $\widehat{K}$ is the upper concave envelope of the points $\big(p_1(N(A)), p_2(A)\big)$, $A \subseteq J_2$. All its breakpoints can be found by one parametric minimum cut, and the bound $\mathrm{LB}_U^{\mathrm{LP}}$, obtained by replacing K with $\widehat{K}$ in LB_U, is valid and can be computed in strongly polynomial time.*

Proof. Without the budget row, every constraint $x_j - y_i \le 0$ has one coefficient $+1$ and one -1. The constraint matrix is therefore the transpose of a network matrix and is totally unimodular, so the polytope Q defined by these rows and the bounds is integral. Its integral points are the pairs (x, y) with x the indicator of a set A and $y \supseteq$ the indicator of $N(A)$; the best of them for a given y has x equal to all trucks whose predecessors are in y. Adding the single budget row cuts Q by a hyperplane. An optimal vertex of the cut polytope is either a vertex of Q or lies on an edge of Q, so it is a convex combination of at most two integral points, and its value is at most the concave envelope at c. Conversely, any convex combination of integral points of cost at most c lies in Q and satisfies the budget, so the LP value is at least the envelope. Hence $\widehat{K}$ equals the envelope. For a multiplier $\lambda \ge 0$ on the budget, $\max\{p_2 \cdot x - \lambda p_1 \cdot y : (x, y) \in Q\}$ is a maximum-weight closure problem, which is one minimum cut (Picard, 1976). The function $\widehat{K}$ is the Lagrangian dual of this knapsack-constrained closure problem, which Gallo et al. (1989) compute by one parametric minimum cut. Their algorithm returns all breakpoints together with one optimal closure for each linear piece. The closures are nested, so there are at most $n_1 + n_2 + 1$ of them, and they are the vertices of the envelope. Validity follows from $\widehat{K} \ge K$ and Remark 1. □

We implemented $\widehat{K}$ with one maximum flow per breakpoint search step, which is strongly polynomial, and checked it against an LP solver and against brute force on small instances. On the benchmark the envelope is a single segment: with density one half, every proper set of outbound trucks needs most of the inbound trucks, so the best ratio of outbound to inbound work is reached by the whole set. $\mathrm{LB}_U^{\mathrm{LP}}$ then reduces to $\max(a, L_{\mathrm{CS}})$, a classical bound. It did not exceed the classical bounds on any of the 10,500 instances, nor on the 240 structured ones of Section 8.5. The integer knapsack is what makes the bound strong.

S3. The certificate: checks and limits

An a-posteriori certificate. The following lemma is the first step of Corollary 2. Its right-hand side can be computed after the schedule is built.

Lemma S1 (Release certificate)**.** *For every inbound order π,*

$$\mathrm{UB}(\pi) \le \max\Big(C_1(\pi),\ \max_j \Big[r_j(\pi) + p_{2j} + \frac{P_2 - p_{2j}}{m_2}\Big]\Big),$$

where $C_1(\pi)$ is the completion time of the last inbound truck.

Proof. If an inbound truck finishes last, $\mathrm{UB}(\pi) = C_1(\pi)$. Otherwise let ℓ be the outbound truck that finishes last. It is released at $r_\ell(\pi)$. Because stage 2 is non-idle, all m_2 outbound doors process other outbound trucks at every moment of $[r_\ell(\pi), s_\ell)$. Hence $s_\ell - r_\ell(\pi) \le (P_2 - p_{2\ell})/m_2$ and $\mathrm{UB}(\pi) = s_\ell + p_{2\ell}$. □

The right-hand side can be computed after the schedule is built, so it serves as an a-posteriori certificate. Corollary 2 turns it into an a-priori guarantee.

We checked Corollary 2 with our decoder and our bound code. On 100,000 random tiny instances with the benchmark's condition (every truck has a predecessor or successor), the largest value of $(\mathrm{UB}/\mathrm{UPB})/(3 - 1/M)$ was 0.887. On 100,000 tiny instances without that condition it was 0.944. On all 10,500 benchmark instances, with the worst of three orders (the PCH order, longest first and a random order), the largest ratio UB/UPB was 1.488, and the largest value of $(\mathrm{UB}/\mathrm{UPB})/(3 - 1/M)$ was 0.576; the two maxima need not come from the same instance. No case violated the corollary.

How tight can such a certificate be?. The following facts limit what any certificate against UPB can achieve.

Remark S1 (Complete precedence). If every outbound truck needs every inbound truck, the instance is two identical-machine problems in series. Every list order then satisfies UB/UPB $\leq 2 - 1/M$: by Graham's argument (Graham, 1969) applied to each stage, UB $\leq a + b + (1 - 1/M)(p + q)$, while $a + b \leq \mathrm{LB}_{e2}$ and $p + q \leq \mathrm{LB}_{e2}$ because every head is at least $\max(a, p)$. This is tight for every fixed M. This case does not occur on the benchmark, and we do not count it as a contribution.

Proposition S6 (A family with a large ratio)**.** *Let $m_1 = m_2 = m$, let every outbound truck need every inbound truck, and let both sides have $m+1$ trucks of length κ. Then* $\mathrm{OPT} = 4\kappa$ *and the fractional values of* UPB *and* LB_U^{Φ} *are both $2\kappa(m + 1)/m$, so* $\mathrm{OPT}/\mathrm{UPB} = 2m/(m + 1)$. *For the rounded bound the ratio tends to $2m/(m + 1)$ as $\kappa \to \infty$.*

Proof. No outbound truck can start before all inbound trucks finish, and $m+1$ trucks of length κ need 2κ on m doors. So $\mathrm{OPT} = 2\kappa + 2\kappa = 4\kappa$. For the bound, all heads equal $\max(a, p) = (m+1)\kappa/m$. For $t < (m+1)\kappa/m$ only $A = \emptyset$ is head-feasible, so $\Phi(t) = \max(b, q) = (m+1)\kappa/m$ and $t + \Phi(t)$ tends to $2(m + 1)\kappa/m$. For larger t, the set $A = J_2$ is head-feasible and $t + \Phi(t) \leq L_{\mathrm{CS}} = r_{\min} + b$ (all heads are equal). The terms a, p, q are smaller. Hence $\mathrm{LB}_U^{\Phi} = 2\kappa(m+1)/m$. Since $\mathrm{UPB} \leq \mathrm{LB}_U^{\Phi}$ (Theorem S1 and Proposition S2) and $\mathrm{LB}_{e2} \geq h + P_2/m_2 = 2\kappa(m + 1)/m$, also $\mathrm{UPB} = 2\kappa(m + 1)/m$. □

So no certificate against UPB can be better than $2m/(m + 1)$, which tends to 2 as m grows. The two statements concern different ratios. The remark bounds UB/UPB for the worst list order and is tight for that ratio; the proposition bounds OPT/UPB, which limits every certificate. For fixed m they leave a gap of $(m - 1)/(m(m + 1))$ under complete precedence, between what any certificate can achieve and what list schedules achieve.

Lemma S2 (Density does not help)**.** *Consider instances with integer lengths in which every outbound truck has a predecessor and every inbound truck a successor. Let $B(I) = \lceil \max(a, b, \mathrm{LB}_{e2}, \mathrm{LB}_{\mathrm{PCH}}, \mathrm{LB}_U) \rceil$, which is the computed bound without the terms L_{CS} and $\mathrm{LB}_{\mathrm{load}}$. Here LB_U is taken with the simpler cap $m_2(t - r_{\min})^+$ in place of $D(t)$; with the door cap, L_{CS} would become part of LB_U (Theorem 2), and the lemma is stated for the bound without it. The supremum of OPT/B over such instances with exactly $\lceil n_1 n_2/2 \rceil$ predecessor pairs equals the supremum of OPT/B over all such instances.*

The proof (Section S3.1) scales a given instance by a factor λ and adds unit-length inbound trucks until the density is exactly one half. Every term of B then moves by an amount that does not depend on λ, so the ratio of the padded instance tends to the ratio of the original one. The lemma concerns the forward bound only. It means that the density one half of the benchmark cannot, by itself, give a better worst-case guarantee than Corollary 2. It does not apply to the benchmark itself, whose lengths lie between 10 and 100 and therefore cannot be scaled.

Tightness on two-level instances. The tight examples of Jaffe (1980) use long chains of tasks, which a cross-dock does not have. A hill-climbing search over all orders of tiny two-level instances ($n_1 \leq 6$, $n_2 \leq 5$, lengths up to 20; 3,901 restarts) found worst-order ratios UB/UPB of 2.185 for $m_1 = m_2 = 2$ (guarantee 2.5), 2.333 for $(m_1, m_2) = (2, 3)$ (guarantee 2.667), 2.000 for $(3, 2)$ and 1.964 for $(3, 3)$. These values exceed Graham's $2 - 1/M$, so the two-level structure does not reduce the constant to the single-stage value. Whether $3 - 1/M$ is attained on two-level instances is open.

Conjecture S1. $\sup \mathrm{OPT}/\mathrm{UPB} = 2$ *over all instances.*

The evidence is weak: an adversarial search over instances with $n \leq 5$ found at most 1.5. We list the conjecture because a proof would give a much better certificate than Corollary 2.

S3.1. Proof of Lemma S2

Let $\mathcal{I}$ be the set of instances with integer lengths ≥ 1 in which every outbound truck has at least one predecessor and every inbound truck at least one successor. Let E be the set of predecessor pairs. Write $B^f(I) = \max(a, b, \mathrm{LB}_{e2}, \mathrm{LB}_{\mathrm{PCH}}, \mathrm{LB}_U)$ for the unrounded bound, so that $B(I) = \lceil B^f(I) \rceil \in [B^f(I), B^f(I) + 1)$. We use

$$\Psi(I) = \mathrm{LB}_U(I) = \sup\Big\{ t + \frac{P_2 - W(t)}{m_2} : W(t) < P_2 \Big\},$$
$$W(t) = \min\big(K(m_1 t),\, m_2 (t - r_{\min})^+\big),$$

where K is evaluated at real capacities ($K(c) = K(\lfloor c \rfloor)$ for integer data).

The inequality "$\leq$" in the lemma is immediate, because the density-one-half instances form a subset of $\mathcal{I}$. For "$\geq$", fix $I \in \mathcal{I}$ and an integer $\lambda \geq 2$, and let λI be the instance with every length multiplied by λ.

Construction. We build I' from λI by adding trucks of length 1.

(1) If $n_2 \leq 2$, add $g = 3 - n_2$ outbound trucks, each with every inbound truck of I as predecessor. Otherwise $g = 0$. Let $n_2' = n_2 + g \geq 3$.

(2) Add inbound trucks one at a time, each with a non-empty successor set among the n_2' outbound trucks. A new inbound truck with s successors adds s pairs and n_2' cells to the $n_1 \times n_2'$ table of possible pairs. Use $s = n_2'$ while the number of pairs is below half the number of cells, and $s = 1$ while it is above. One added truck changes the difference between half the cells and the pairs by any integer amount in $[n_2'/2 - n_2',\ n_2'/2 - 1]$, which contains both signs because $n_2' \geq 3$. The last added truck can therefore be chosen to make the number of pairs exactly $\lceil n_1' n_2'/2 \rceil$.

The number k of added inbound trucks and the number $g \leq 2$ of added outbound trucks depend only on $(n_1, n_2, |E|)$, not on λ. Every added inbound truck has a successor and every added outbound truck has predecessors, so $I' \in \mathcal{I}$. Let $\delta = k/m_1$.

Step 1: optimum. Deleting trucks keeps a schedule feasible, so $\mathrm{OPT}(I') \geq \mathrm{OPT}(\lambda I) = \lambda\,\mathrm{OPT}(I)$. Conversely, process the k added inbound trucks first on one door, shift an optimal schedule of λI by k, and process the g added outbound trucks at the end on one outbound door. This is feasible, so $\mathrm{OPT}(I') \leq \lambda\,\mathrm{OPT}(I) + k + g$.

Step 2: homogeneity. Each term of B^f is positively homogeneous of degree one in the lengths. In particular $K_{\lambda I}(\lambda c) = \lambda K_I(c)$, and heads, tails, workloads and $r_{\min}$ scale by λ. Hence $B^f(\lambda I) = \lambda\, B^f(I)$.

Step 3: each term moves by a constant. We show $B^f(\lambda I) - \delta \leq B^f(I') \leq B^f(\lambda I) + C$ with $C = k/m_1 + g/m_2 + 1$. Below, unprimed quantities refer to λI and primed ones to I'. The added trucks have length $1 \leq \lambda$, which is at most every original length, so no maximum-length term changes, except through the added outbound trucks, whose length 1 is at most q.

- *Workloads.* $a \leq a' \leq a + k/m_1$ and $b \leq b' \leq b + g/m_2$.
- *Heads.* For an original outbound truck j, we have $p_1(N'(j)) \in [p_1(N(j)),\, p_1(N(j)) + k]$, so $h_j' \in [h_j, h_j + \delta]$. An added outbound truck has head $\max(P_1'/m_1, p)$, which is at least every h_j'. Hence $r_{\min}' \in [r_{\min}, r_{\min} + \delta]$.
- LB_{e2}. Its first part equals

$$\sup_{\theta} \Big[\theta + \sum_{l:\, h_l \geq \theta} \frac{p_{2l}}{m_2} \Big].$$

Raising each head by at most δ cannot lower this supremum and raises it by at most δ, since $\{l : h'_l \geq \theta\} \subseteq \{l : h_l \geq \theta - \delta\}$. The added outbound trucks add at most g/m_2 to the sums. Their own term $h + p_2$ is at most $\max(P'_1/m_1, p) + 1 \leq B^f(\lambda I) + \delta + 1$. The terms $h_j + p_{2j}$ of original trucks move by at most δ.

- $\mathrm{LB}_{\mathrm{PCH}}$. Tails of original inbound trucks do not decrease and grow by at most g/m_2, and prefix workloads do not decrease. Hence $\mathrm{LB}_{\mathrm{PCH}}(I') \geq \mathrm{LB}_{\mathrm{PCH}}(\lambda I)$. For the upper side, take any position in the order of I' and let θ be the tail at that position. The trucks up to that position are added trucks, of total length at most k, and a set O of original trucks whose tails in I' are at least θ, hence at least $\theta - g/m_2$ in λI. If O is empty, the term at this position is at most $k/m_1 + \theta \leq k/m_1 + \max(b', q)$, because every tail is at most $\max(b', q)$. Otherwise let $\mathcal{T} \supseteq O$ be the set of all inbound trucks of λI with tail at least $\theta - g/m_2$. The PCH term of λI at the last truck of $\mathcal{T}$ is at least $p_1(O)/m_1 + \theta - g/m_2$. So the term of I' is at most $(p_1(O) + k)/m_1 + \theta \leq \mathrm{LB}_{\mathrm{PCH}}(\lambda I) + k/m_1 + g/m_2$. The terms $p_{1i} + \Delta_i$ behave in the same way, and for an added truck this term is at most $1 + \max(b', q)$. Since $b, q \leq B^f(\lambda I)$, we get $\mathrm{LB}_{\mathrm{PCH}}(I') \leq B^f(\lambda I) + k/m_1 + g/m_2 + 1$.

- *The union term Ψ.* For a set A of original outbound trucks, its predecessor cost in I' lies in $[\,\mathrm{cost}(A), \mathrm{cost}(A) + k]$. A set that contains an added outbound truck costs P'_1, and the added trucks add at most g to its value. Hence $K(c - k) \leq K'(c) \leq K(c) + g$, and $P'_2 = P_2 + g$.

  Upper side. Fix t with $W'(t) < P'_2$ and put $t'' = t - k/m_1$. Since $\delta = k/m_1$, $W'(t) \geq \min(K(m_1 t''), m_2(t'' - r_{\min})^+) = W(t'')$. If $W(t'') < P_2$, the term of I' at t is at most $t'' + (P_2 - W(t''))/m_2 + k/m_1 + g/m_2 \leq \Psi(\lambda I) + k/m_1 + g/m_2$. If $W(t'') \geq P_2$, then $P'_2 - W'(t) \leq g$ and the term is at most $t + g/m_2$. In this case either $K'(m_1 t) < P'_2$, which forces $m_1 t < P'_1$ and $t < a' \leq a + k/m_1$, or $m_2(t - r'_{\min}) < P'_2$, in which case the term equals $r'_{\min} + b' \leq r_{\min} + b + \delta + g/m_2$. Since $a \leq B^f(\lambda I)$ and $r_{\min} + b \leq \Psi(\lambda I)$, every case gives $\Psi(I') \leq B^f(\lambda I) + k/m_1 + g/m_2$.

  Lower side. For every t, $K'(m_1 t) \leq K(m_1(t + \delta)) + g$ and $m_2(t - r'_{\min})^+ \leq m_2(t + \delta - r_{\min})^+$, so $W'(t) \leq W(t + \delta) + g$. Whenever $W(t + \delta) < P_2$, the guard $W'(t) < P'_2$ holds and the term of I' at t is at least $t + (P_2 - W(t + \delta))/m_2$, which is the term of λI at $t + \delta$ minus δ. Hence $\Psi(I') \geq \Psi(\lambda I) - \delta$.

Taking the maximum over the terms gives $B^f(\lambda I) - \delta \leq B^f(I') \leq B^f(\lambda I) + C$.

Step 4: limit. By Steps 1–3, $B(I') \leq B^f(I') + 1 \leq \lambda\, B^f(I) + C + 1$ and $\mathrm{OPT}(I') \geq \lambda\, \mathrm{OPT}(I)$. Hence

$$\frac{\mathrm{OPT}(I')}{B(I')} \;\geq\; \frac{\lambda\, \mathrm{OPT}(I)}{\lambda\, B^f(I) + C + 1} \;\xrightarrow[\lambda \to \infty]{}\; \frac{\mathrm{OPT}(I)}{B^f(I)} \;\geq\; \frac{\mathrm{OPT}(I)}{B(I)}.$$

The padded instances have density one half, so the supremum over density-one-half instances is at least $\mathrm{OPT}(I)/B(I)$ for every $I \in \mathcal{I}$. The upper estimate $\mathrm{OPT}(I')/B(I') \leq (\lambda\, \mathrm{OPT}(I) + k + g)/(\lambda\, B^f(I) - \delta)$ shows that the padded ratios converge to $\mathrm{OPT}(I)/B^f(I)$, so nothing is lost in the limit either. □

Scope. (i) The lemma concerns the forward bound. For the maximum with the reversed bound, an added inbound truck with one successor becomes, in the reversed instance, an outbound truck with one predecessor, whose head can lower $r_{\min}$. The upper side of Step 3 still holds, but the lower side needs $r'_{\min} \geq r_{\min}$ and fails, so we do not claim the reversed version. (ii) Benchmark lengths lie in $[10, 100]$, which excludes the limit $\lambda \to \infty$.

S4. Computing the bound: caps, determinism and the CPU-capped census

Caps and determinism. For the CPU-capped census the walk had a budget of 4 s of CPU per direction, and each knapsack solve had 1 s. With these caps the walk finished on every instance

Table S1: Lower-bound census on the 10,500 benchmark instances. "Gain" is (UPB − L_0)/L_0, where L_0 is the largest earlier bound available for the instance (the Chen–Song-type bound reported with the HgR results, $\mathrm{LB_{PCH}}$, and the cheap bounds LB_{e2} and $\mathrm{LB_{load}}$). The last three gap columns use the published HgR upper bound and the gap (UB − LB)/LB. "Exact" is the share of instances on which the knapsack walk finished within the 4 s CPU cap. The census covers every instance, so the values are population means, not estimates.

		Gain of UPB over L_0 (%)			HgR gap (%)			
Slice	n	mean	max	# lower	vs L_0	vs UPB	Exact (%)	CPU s (mean / max)
All	10,500	15.00	29.14	0	26.1	9.8	66	1.2 / 10.4
$n_1 = 20$	1,500	4.97	23.03	0	17.4	11.9	100	0.0 / 0.0
$n_1 = 30$	1,500	9.56	23.89	0	22.8	12.3	100	0.0 / 0.1
$n_1 = 40$	1,500	14.48	25.58	0	26.3	10.4	100	0.2 / 0.5
$n_1 = 50$	1,500	17.89	29.14	0	28.1	8.7	99	0.6 / 2.0
$n_1 = 60$	1,500	19.36	28.81	0	28.9	8.0	55	1.6 / 5.3
$n_1 = 70$	1,500	19.41	28.37	0	29.3	8.3	7	2.6 / 7.2
$n_1 = 80$	1,500	19.35	27.48	0	29.8	8.8	0	3.2 / 10.4
$m = 2$	2,100	19.34	28.81	0	26.2	5.8	65	1.2 / 8.6
$m = 4$	2,100	18.02	28.32	0	29.1	9.5	65	1.2 / 8.2
$m = 10$	2,100	12.75	25.65	0	30.3	15.8	66	1.2 / 8.2
U(2,4)	2,100	14.88	29.14	0	23.3	7.4	66	1.2 / 9.3
U(2,10)	2,100	10.02	27.70	0	21.5	10.4	67	1.1 / 10.4

with $n_1 \le 40$, on 99% at $n_1 = 50$, and on 55%, 7% and 0% at $n_1 = 60$, 70 and 80. An unfinished walk still gives a valid bound, but its value can depend on the machine. We therefore also define a deterministic version, UPB-det, which caps the walk at 50 steps and gives each CP-SAT solve 0.5 units of the solver's deterministic time. The capped walk can solve up to $c_{\mathrm{full}} \le P_1$ strongly NP-hard knapsacks; the deterministic version does at most 50 steps of bounded work, so it runs in pseudo-polynomial time (polynomial in n_1, n_2 and P_1) and returns a value that depends only on the instance and the solver version. The LP version of Proposition S5 needs at most $2(n_1 + n_2) + 1$ maximum flows.

Deterministic against CPU-capped values. We computed UPB-det (50 walk steps, 0.5 deterministic seconds per knapsack) on all 10,500 instances. Its values and step counts were identical on two machines on the 150 instances of an earlier check. Where both walks finish (6,909 instances), UPB-det equals the CPU-capped value. At $n_1 \ge 60$ it is higher on 78% of the instances, by 1.44% on average (0.43%, 1.53% and 2.35% at $n_1 = 60$, 70 and 80), and lower on 3 of 4,500, by at most 1.1%. The walk finished on 50% of these instances, against 21% with the CPU cap. It took 6.8 s on average (maximum 36 s). With UPB-det, the mean census gain over the best earlier bound is 15.76% instead of 15.03%, and 21.1% instead of 19.4% at $n_1 \ge 60$. We propose UPB-det as the reference value for future comparisons.

The CPU-capped census. The CPU-capped walk (4 s per direction) gives the following census. We report it for completeness; the main text uses UPB-det throughout. Table S1 gives the census on all 10,500 instances, and Table S2 repeats the overall gain against several baselines. Against the best earlier bound that does not come from this paper, $\lceil \max(L_{\mathrm{CS}}, \mathrm{LB_{PCH}}) \rceil$, UPB is 15.03% higher on average. The quantiles over instances are 0.00, 0.76, 17.20, 23.66 and 30.81% at 0, 10, 50, 90 and 100%. Against the Chen–Song-type bound as reported with the HgR results, the mean gain is 15.22%. Adding the reversed instance raises the gain over the best non-own bound to 15.26%. By construction, the bound cannot fall below these baselines. Across the baselines of Table S2 the mean gain varies by less than half a point.

Table S2: Gain (%) of the new bound over several earlier baselines, on all 10,500 instances. L_{CS} is the Chen–Song-type bound as reported with the public HgR results. Quantiles are over instances. No instance has a negative gain under any baseline.

New bound	Baseline	mean	q_0	q_{10}	q_{50}	q_{90}	q_{100}
UPB	$\max(L_{\mathrm{CS}}, \mathrm{LB}_{\mathrm{PCH}})$	15.10	0.00	0.88	17.28	23.69	30.82
UPB	$\lceil \max(L_{\mathrm{CS}}, \mathrm{LB}_{\mathrm{PCH}}) \rceil$	15.03	0.00	0.76	17.20	23.66	30.81
UPB	L_{CS} (as reported)	15.22	0.00	1.16	17.38	23.86	30.82
UPB	$\lceil L_{\mathrm{CS}} \rceil$	15.15	0.00	1.07	17.30	23.83	30.81
UPB	L_0 (adds LB_{e2}, $\mathrm{LB}_{\mathrm{load}}$)	15.00	0.00	0.88	17.19	23.52	29.14
LB*	$\max(L_{\mathrm{CS}}, \mathrm{LB}_{\mathrm{PCH}})$	15.34	0.00	1.17	17.56	23.89	30.82
LB*	$\lceil \max(L_{\mathrm{CS}}, \mathrm{LB}_{\mathrm{PCH}}) \rceil$	15.26	0.00	1.11	17.48	23.86	30.81
LB*	L_{CS} (as reported)	15.46	0.00	1.47	17.63	24.08	31.10
LB*	$\lceil L_{\mathrm{CS}} \rceil$	15.39	0.00	1.34	17.57	24.05	31.08
LB*	L_0 (adds LB_{e2}, $\mathrm{LB}_{\mathrm{load}}$)	15.24	0.00	1.17	17.44	23.70	29.14

S5. The window-inference bound in full

This section gives the full statements, proofs, procedure and results of the window-inference bound UPB-W of Section 4.6.

Setting. Data are integers, so if OPT $\le T$ there is an integer schedule with makespan at most T. Let $\mathcal{S}_T$ be the set of these schedules. A window set $W = (r^1, d^1, r^2, d^2)$ is *T-valid* if every $S \in \mathcal{S}_T$ satisfies $r_i^1 \le s_i$, $C_i \le d_i^1$ for every inbound truck and $r_j^2 \le s_j$, $s_j + p_{2j} \le d_j^2$ for every outbound truck. The initial set W^0 has $r^1 = 0$, $d_i^1 = T - \hat{q}_i$, $r_j^2 = \hat{h}_j$ and $d^2 = T$, where $\hat{h}_j \ge h_j$ is a lower bound on the makespan of $N(j)$ alone on m_1 doors (we use dual-feasible functions, Fekete and Schepers, 2001; Clautiaux et al., 2010) and $\hat{q}_i$ is the same quantity for $S(i)$ on m_2 doors. Each rule below either maps a T-valid set to a T-valid set or proves $\mathcal{S}_T = \emptyset$.

Proof of Proposition 2. By induction over the sequence, every window set produced is T-valid, because each rule preserves T-validity (Lemmas S3–S5 and Proposition S7). A refutation therefore shows that $\mathcal{S}_T$ is empty, so no integer schedule has makespan at most T, and OPT $> T$ by integrality. Each refuted T is a proof by itself, so the largest refuted value gives the bound whatever the order of the tests. □

Cross-stage propagation. The first rule moves windows from one stage to the other through the union of predecessors.

Lemma S3 (Propagation)**.** *Let W be T-valid. For an outbound truck j and a time α, let $\Sigma_j(\alpha) = \sum_{i \in N(j)} \min\big(p_{1i}, (r_i^1 + p_{1i} - \alpha)^+\big)$. Then*

$$r_j^2 \leftarrow \max\Big(r_j^2,\ \max_{i \in N(j)} (r_i^1 + p_{1i}),\ \max_{\alpha:\, \Sigma_j(\alpha) > 0} \big(\alpha + \lceil \Sigma_j(\alpha)/m_1 \rceil\big)\Big)$$

keeps W T-valid, and so does the mirror update of d_i^1 over $S(i)$ on m_2 doors. If a window becomes shorter than its truck, $\mathcal{S}_T = \emptyset$.

Proof. Every $i \in N(j)$ starts at or after r_i^1 and completes by s_j, so at least $\min(p_{1i}, (r_i^1 + p_{1i} - \alpha)^+)$ of its work lies in $[\alpha, s_j]$. If $\Sigma_j(\alpha) > 0$, some predecessor ends after α, so $s_j > \alpha$, and the work $\Sigma_j(\alpha)$ fits on m_1 doors in $[\alpha, s_j]$. Since s_j is an integer, $s_j \ge \alpha + \lceil \Sigma_j(\alpha)/m_1 \rceil$. The mirror is the same argument on the reversed instance (Proposition 1). □

Per-stage tests and shaving. Each stage, with the other stage replaced by its windows, is a parallel-machine problem with release dates and deadlines. We test it in two ways: the preemptive relaxation by the flow network of Horn (1974), which is exact for $P|pmtn, r_j, d_j|$ and which Vandevelde et al. (2005) apply per stage of a hybrid flow shop, and the energetic test of Baptiste et al. (1999) on intervals bounded by window ends. Both are necessary conditions for $\mathcal{S}_T \neq \emptyset$, because every $S \in \mathcal{S}_T$ restricted to one stage is a non-preemptive schedule inside the windows. Windows are then narrowed by shaving.

Lemma S4 (Shaving)**.** *Let W be T-valid and let $\mathcal{H}$ be a half-window hypothesis on one truck, $s_j \leq t-1$ or $s_j \geq t$ for an outbound truck, $C_i \leq t-1$ or $C_i \geq t$ for an inbound truck. If any rule of this section refutes $W \cap \mathcal{H}$, then $W \cap \neg\mathcal{H}$ is T-valid. If both halves are refuted, $\mathcal{S}_T = \emptyset$.*

Proof. Every rule is a statement about all schedules inside the given windows. Applied to $W \cap \mathcal{H}$, it shows that no $S \in \mathcal{S}_T$ satisfies $\mathcal{H}$. Hence every $S \in \mathcal{S}_T$ satisfies $\neg\mathcal{H}$, which is again a half-window. □

We search for refutable hypotheses on all four window sides by bisection over t. Bisection is used only to find refutations, and each refuted t is a proof by itself, so no monotonicity is needed.

The split-cut test. The strongest test couples the two stages at one cut time. It extends the argument of Lemma 1: instead of a set A of outbound trucks started by t alone, it constrains A together with the set Y of inbound trucks completed by $t-1$, with packing conditions on both stages.

Proposition S7 (Split-cut test)**.** *Let W be T-valid, $S \in \mathcal{S}_T$ and $t \in \{1, \dots, T-1\}$. Put $Y = \{i : C_i \leq t-1\}$ and $A = \{j : s_j \leq t-1\}$. Then:*

- *(a) (closure) $j \in A$ implies $N(j) \subseteq Y$;*
- *(b) (packing) Y fits into m_1 bins of size $t-1$ and $J_2 \setminus A$ into m_2 bins of size $T-t$; in particular every dual-feasible function gives a valid linear row for each of the two sets;*
- *(c) (inbound work) for $\tau \in \{t-1, t\}$, $P_1 - m_1\tau \leq \min\big(\sum_{i \notin Y} \min(p_{1i}, (d_i^1 - \tau)^+),\ \overline{F}(\tau)\big)$;*
- *(d) (outbound work) $P_2 - \min\big(\sum_{j \in A} \min(p_{2j}, (t - r_j^2)^+),\ \overline{D}(t)\big) \leq m_2(T-t)$;*
- *(e) (forcing) $r_i^1 + p_{1i} > t-1 \Rightarrow i \notin Y$; $d_i^1 \leq t-1 \Rightarrow i \in Y$; $r_j^2 \geq t \Rightarrow j \notin A$; $d_j^2 - p_{2j} \leq t-1 \Rightarrow j \in A$;*

where $\overline{D}$ and $\overline{F}$ are the door caps of Lemma S5 (or the trivial caps $m_2(t - \min_j r_j^2)$ and $m_1(\max_i d_i^1 - \tau)$). Hence, if no pair (Y, A) satisfies (a)–(e), or if the linear relaxation of (a)–(e) over $[0,1]$ variables is infeasible, then $\mathcal{S}_T = \emptyset$.

The proof is in Section S5.1. Deciding (a)–(e) contains bin packing, so we use it in two forms. In the first, CP-SAT decides the 0/1 system; a timeout means "not refuted". In the second, a linear-programming solver (GLOP) decides its relaxation, and *probes* fix one variable: if the relaxation with $x_j = 1$ ($j \in A$) is infeasible, then $s_j \geq t$ for every $S \in \mathcal{S}_T$ (Lemma S4), and the three other fixings narrow d_j^2, r_i^1 and d_i^1 in the same way. The linear form also carries two families of rows that are valid for 0/1 points, avoidance rows and a union threshold (Section S5.1). A nested version with several cuts and monotone sets $Y_1 \subseteq \cdots \subseteq Y_\nu$, $A_1 \subseteq \cdots \subseteq A_\nu$ is valid by the same argument.

Door caps. The last rule sharpens the capacity terms in (c) and (d) by counting trucks per door.

Lemma S5 (Door-start and door-end caps)**.** *Let W be T-valid, let $U_2(k)$ be a lower bound on $\min\{p_1(N(A)) : A \subseteq J_2, |A| = k\}$ and $U_1(k)$ one on $\min\{p_2(S(B)) : B \subseteq J_1, |B| = k\}$. Put $\eta_k = \max\big(r^2_{(k)}, \lceil U_2(k)/m_1 \rceil\big)$, where $r^2_{(k)}$ is the k-th smallest r^2_j, and $F_k = \min\big(d^1_{[k]}, T - \lceil U_1(k)/m_2 \rceil\big)$, where $d^1_{[k]}$ is the k-th largest d^1_i. Then every $S \in \mathcal{S}_T$ satisfies:*

(i) the k-th smallest outbound start is at least η_k, and the k-th latest inbound completion is at most F_k;

(ii) at every time θ at most $|\{k : \eta_k \le \theta\}|$ outbound trucks are in process;

(iii) the outbound work done in $[0, t]$ is at most $\overline{D}(t) = \sum_{k \le \min(m_2, n_2)} (t - \eta_k)^+$, and the inbound work done after τ is at most $\overline{F}(\tau) = \sum_{k \le \min(m_1, n_1)} (F_k - \tau)^+$.

With $\eta_k = h_{(k)}$, the cap $\overline{D}$ is the cap $D(t)$ of Section 4.1. Here it is recomputed from the current shaved windows, so shaving and the caps tighten each other. The proof is in Section S5.1.

The procedure. For each trial T, starting at $\lceil \mathrm{LB}^* \rceil$ and moving up by doubling steps and then by bisection, the procedure starts from W^0 and repeats until no window changes or the time is used up: propagation (Lemma S3); the flow and energetic tests on each stage; shaving of the four window sides with these tests; and linear probes of the split-cut test at 16 cut times per round. It then runs shaving with the 0/1 split-cut test in CP-SAT, exact single-stage tests $P|r_j|C_{\max}$ on the outbound windows and on the reversed inbound windows, and the nested split-cut test with window forcing. The procedure that uses only propagation, flow, energetic tests and linear probes is polynomial; the full procedure is not, because its CP-SAT tests decide strongly NP-hard questions. Every timeout counts as "not refuted", which keeps the bound valid (Proposition 2). The version with door caps in every test gave the same bound on all 98 solved instances of Section S5.2, so we report the version without them.

Credit in full. Most components of UPB-W are established, and we credit them precisely.

- *Window propagation across stages* in hybrid flow shops is due to Néron et al. (2001), who adjust release dates and deadlines per stage by energetic tests and propagate them along job chains, with dichotomic shaving inside branch and bound, and to Hidri and Haouari (2011), who propagate windows forward and backward between centres to a fixpoint and raise T until the windows are consistent. In both, each operation has one predecessor, the previous operation of the same job. Lemma S3 is the classical energetic adjustment of a release date, applied to the AND-predecessor set of a cross-dock truck, which can be large and shared with many other trucks.

- *Shaving to a fixpoint* with energetic reasoning and dual-feasible functions inside feasibility tests is established for the RCPSP: Haouari et al. (2012) give a dichotomous shaving loop (their Algorithm 1), and Carlier and Néron (2007) introduce redundant resources from dual-feasible functions together with a linear program on time segments. Lemma S4 is their shaving principle written for our window sets.

- *Energetic reasoning* is used in the form of Baptiste et al. (1999) (see also Baptiste et al., 2001). Energetic reasoning originates in the work of Erschler, Lopez and Esquirol in the early 1990s, as recounted by Baptiste et al. (2001).

- *Destructive improvement.* Fixing a trial makespan, reducing windows and refuting the trial by feasibility tests is the destructive-improvement scheme of Klein and Scholl (1999); UPB-W instantiates it for the cross-dock. Brucker and Knust (2000) refute trial makespans for

the RCPSP by constraint propagation followed by a preemptive time-window LP solved by column generation. With identical doors and no further resources, every set of at most m trucks is feasible, and their LP reduces to preemptive feasibility with time windows on m identical machines, the max-flow test of Horn (1974) that we use per stage. Our split-cut LP is different: it relaxes closure and packing conditions over the AND-predecessor sets of both stages, and its refutations carry exact-arithmetic certificates.

- *Dual-feasible functions* and their validity as packing rows are due to Fekete and Schepers (2001), and their relation to the configuration LP is surveyed by Clautiaux et al. (2010). Lahimer et al. (2013) apply dual-feasible functions per stage of a hybrid flow shop with multiprocessor tasks.

- *The union knapsack at one time* is the inequality of Prot and Rapine (2013) for one door per side, which LB_U already generalises (Section 4.1).

What is ours is the coupling of these techniques with the union-of-predecessors structure. Specifically: the split-cut test, which constrains the inbound trucks completed and the outbound trucks started at one cut jointly, under closure and packing on both stages (Proposition S7); the cross-stage inference for the AND-precedence sets of a cross-dock, where an outbound truck waits for all of a large set of inbound trucks and each inbound truck feeds many outbound trucks (Lemma S3 and the probes); the multi-door caps with union-based staggering (Lemma S5); and the computation on the benchmark of Cota et al. (2016) (Section S5.2).

S5.1. Proofs

Throughout, W is a T-valid window set, $S \in \mathcal{S}_T$ is an integer schedule with makespan at most T, and s, C denote its start and completion times. The statements below were checked by brute force on 610 tiny instances with exact optima (Section S5.2).

Proof of Proposition S7. (a) If $j \in A$, every $i \in N(j)$ completes by $s_j \le t-1$, so $i \in Y$. (b) Every $i \in Y$ lies inside $[0, t-1]$ on its door, so Y fits into m_1 bins of size $t-1$. Every $j \notin A$ starts at t or later and ends by T, so $J_2 \setminus A$ fits into m_2 bins of size $T-t$. For a dual-feasible function f (normalised to capacity 1), a set that fits into m bins of size c satisfies $\sum f(p/c) \le m$ (Fekete and Schepers, 2001); applied to the 0/1 indicator of the set, this is a linear row (Clautiaux et al., 2010, Proposition 2). (c) The inbound work done in $[0, \tau]$ is at most $m_1\tau$, so at least $P_1 - m_1\tau$ is done after τ. Trucks of Y complete by $t-1 \le \tau$ and do no work after τ. A truck $i \notin Y$ completes at $C_i \le d_i^1$, so its work after τ is at most $\min(p_{1i}, (d_i^1 - \tau)^+)$. All inbound work after τ is also bounded by $\overline{F}(\tau)$ (Lemma S5), or by the trivial cap $m_1(\max_i d_i^1 - \tau)$, since all inbound work ends by $\max_i C_i \le \max_i d_i^1$. (d) Only trucks of A work before t. A truck $j \in A$ starts at $s_j \ge r_j^2$, so it does at most $\min(p_{2j}, (t - r_j^2)^+)$ before t. The outbound work done before t is also at most $\overline{D}(t)$, or at most the trivial cap $m_2(t - \min_j r_j^2)$. The rest is done in $[t, T]$ on m_2 doors. (e) Immediate from the definitions of Y, A and T-validity. For the linear relaxation: the indicator vectors of Y and A are a 0/1 point of the system, so if its relaxation over $[0, 1]$ is infeasible, no $S \in \mathcal{S}_T$ exists. □

The nested test. For cuts $t_1 < \cdots < t_\nu$, the sets $Y_k = \{i : C_i \le t_k - 1\}$ and $A_k = \{j : s_j \le t_k - 1\}$ are nested, and each pair satisfies (a)–(e) at t_k. A truck that starts in $[t_{l-1}, t_l - 1]$ has done at most $t_k - \max(r_j^2, t_{l-1})$ by t_k, and an inbound truck that completes in $[t_{l-1}, t_l - 1]$ does at most $\min\big(p_{1i}, (\min(t_l - 1, d_i^1) - \tau)^+\big)$ after τ. These bucket caps are valid for the same reason as (c) and (d). Removing a cut only coarsens the buckets, so refutation is monotone under adding cuts.

Avoidance and threshold rows. Fix a cut t and write x, y for the indicators of A and Y. Let $c_j = \min(p_{2j}, (t - r_j^2)^+)$ (zero if x_j is forced to 0), $w = \sum_j c_j x_j$, $W_t = \sum_j c_j$ and $a_i = \sum_{j:\, i \notin N(j)} c_j$.

- *Avoidance.* If $w > a_i$, then A contains some j with $i \in N(j)$, so $y_i = 1$ by closure. The row $(W_t - a_i)\, y_i \geq w - a_i$ expresses this and holds at both values of y_i. The mirror row uses $c'_i = \min(p_{1i}, (d_i^1 - t)^+)$, $z = \sum_i c'_i (1 - y_i)$, $Z = \sum_i c'_i$ and $e_j = \sum_{i \notin N(j)} c'_i$: if $z > e_j$, some predecessor of j is outside Y, so $x_j = 0$, which gives $(Z - e_j)\, x_j \leq Z - z$.

- *Union threshold.* Y contains every i with $a_i < w$ and every truck forced into Y. Sort the inbound trucks by a_i. If the forced trucks together with the first $k+1$ trucks of this order fail a dual-feasible packing row for m_1 bins of size $t-1$, then $w \leq a_{(k+1)}$, because otherwise all of them would be in Y and Y would not fit (a subset of a fitting set fits). The procedure tests the failing prefix directly, and each failing test is a proof by itself; no monotonicity of the packing test in the prefix length is needed. The mirror bound on z uses $J_2 \setminus A \supseteq \{j : e_j < z\}$ and bins of size $\max_j d_j^2 - t$.

These rows never cut the uniform fractional point $x = y = \alpha$, because $(\alpha W_t - a_i)/(W_t - a_i) \leq \alpha$. Their role is to cut points that mix the empty and the full closure.

Proof of Lemma S5. (i) Let $s_{(k)}$ be the k-th smallest outbound start. The k trucks that start by $s_{(k)}$ have k distinct lower window ends, so $s_{(k)} \geq r^2_{(k)}$. The union of their predecessors has work at least $U_2(k)$ and is complete by $s_{(k)}$ on m_1 doors, so $s_{(k)} \geq \lceil U_2(k)/m_1 \rceil$ by integrality. The statement on inbound completions is the mirror: the k trucks that complete last, at or after the k-th latest completion $C^{(k)}$, have successors of total work at least $U_1(k)$ that run inside $[C^{(k)}, T]$ on m_2 doors, and $C^{(k)} \leq d^1_{[k]}$. (ii) The argument is pointwise in time. A truck in process at time θ has started by θ, so the number of trucks in process at θ is at most $|\{k : s_{(k)} \leq \theta\}| \leq |\{k : \eta_k \leq \theta\}|$ by (i). This count is constant on every interval that contains no η_k in its interior; the flow test with these capacities uses every η_k (and every F_k) as a breakpoint. The mirror count for inbound trucks is $|\{k : F_k > \theta\}|$. (iii) Door d is idle before its first start f_d. First starts on different doors belong to different trucks, so after sorting, $f_{(k)} \geq s_{(k)} \geq \eta_k$, and door (k) works at most $(t - \eta_k)^+$ in $[0, t]$. Only $\min(m_2, n_2)$ doors can work. The inbound statement is the mirror with last completions. □

Floating-point refutations. The linear form is solved in floating point with every right-hand side relaxed outward by 10^{-3}. Solver tolerances make a floating-point solver more lenient, not less, so they do not exclude a false report of infeasibility. We therefore do not rely on them. Instead, for every infeasibility report in the sample of Section S5.2, we rounded every row bound to its intended integer right-hand side, which gives a system that every integer point of the test satisfies, and verified a Farkas certificate of its infeasibility in exact rational arithmetic. CP-SAT answers are trusted as exact, because CP-SAT works in integer arithmetic; they were not independently re-certified. We then certified every refutation behind every reported number. For each of the 285 reported values above LB* we re-ran the refutation of $T = \text{LB} - 1$ with the same version and budget, treating any infeasibility report without an exact certificate as "not refuted"; all 285 were refuted again, with 2,949 of 2,949 reports certified (92 of these refutations use no linear program). Full replays of all 120 s runs certified 20,784 of 20,784 reports. Every exported system had integer coefficients and variable bounds, the row bounds deviated from integers by exactly the 10^{-3} relaxation, and the smallest normalised Farkas margin was $1.0 \cdot 10^{-3}$. Four of 116 replays on a different machine ended 1–2 units lower because of CP-SAT time limits. For three of them the deciding refutation was found and certified on that machine when tested directly; for the fourth it is found and certified only on the machine of the original run. One reported value therefore depends on the platform, as the main text states.

S5.2. Full results

Design. We evaluated UPB-W (Section 4.6) on the 98 instances of the exact set whose optimum was proven when these experiments were designed (77 with $n_1 = 20$ and 21 with $n_1 = 30$, a subset of the 165 optima above) and on the 105 instances of the development set. Before any run we wrote down a go/no-go test: UPB-W had to close at least 70% of the gap between LB* and the optimum on the 98 instances, and its mean gain over LB* at $n_1 = 60$ on the development set had to have a 95% interval above zero, with a nominal budget of 120 s of one CPU thread per instance. The search checks its budget only between trial tests, and one test may use up to half of it, so the CPU actually used can exceed the nominal value. At $n_1 = 60$ it used 116 s on average and at most 168 s against the nominal 120 s; on the proven optima, at most 244 s. The previous best bound in the comparison is the maximum of the split-cut, nested, single-stage and shaving tests computed without window inference.

Results. Table S3 gives the results. UPB-W closes 73.9% [70.2, 77.4] of the gap between LB* and the optimum, against 61.0% [57.3, 64.6] for the previous best, and the mean distance to the optimum falls from 5.48% to 1.43%. It equals the optimum on 11 of the 98 instances, against 3 for LB*. The gain over the previous best is largest at $n_1 = 30$ (67.0% against 47.3%) and with two doors per side (78.0% against 57.6%), and smallest with four or ten doors per side, where the previous best is already strong. On the development set, UPB-W raises LB* by 1.71% on average and on 61 of the 105 instances, against 1.01% and 34 instances for the previous best. At $n_1 = 60$ it gains 0.24% [0.09, 0.41] at the nominal 120 s, so both parts of the go/no-go test were met. At $n_1 = 70$ and 80 the bound does not rise within 120 s: the first trial test uses its whole share of the budget without an answer. An exploratory run with 300 and 600 s (door caps in every test, one machine; Table S3) shows that this is a matter of budget: doubling it adds 0.51 points [0.31, 0.75] at $n_1 = 70$ (to 0.98%, all 15 raised) and 0.33 [0.15, 0.55] at $n_1 = 80$ (to 0.39%, with 4 of 15 runs still stuck in the first test), but only 0.01 at $n_1 = 40$, where the bound has saturated. An oracle experiment on all 95 solved instances with LB$^* <$ OPT locates the remaining slack. A first diagnostic, before UPB-W was built, gave the preemptive relaxation of the outbound stage the start times of an optimal schedule as release dates; it closes 97.7% [94.9, 99.6] of the gap between LB* and the optimum. With UPB-W itself, given the heads and tails of an optimal schedule, the polynomial tests close all of the gap, against 21.6% [16.0, 27.9] with the inferred windows, whose heads at $T =$ OPT fall 38.3% short in aggregate (figures that depend on which optimal schedule is used). The tests are not the bottleneck; the inference of windows is.

Validity. Two checks support the validity of UPB-W beyond the proofs. First, the linear refutations rely on a floating-point solver. On 38 of the 98 solved instances, stratified by door group and CPU time and all raised by UPB-W, we re-ran the bound and verified every infeasibility report of the solver in exact rational arithmetic (Section S5.1). All 5,551 reports, 4,994 of them inside the 244 refuted trial values, have a Farkas certificate for the system with integer right-hand sides and no tolerance. Second, on 610 tiny instances with exact optima computed by enumeration (520 random instances with up to 11 trucks and 1–3 doors per side, and 90 harder ones with up to 12 trucks on which the simplest polynomial version stays below the optimum), we started the search with an upper limit far above the optimum, so that $T =$ OPT was actually tested. No variant of the procedure refuted the optimum or the optimum plus one, and an exact CP-SAT check confirmed that the final windows contain every schedule with makespan at most T for all 4,880 window sets examined. There was no violation, and UPB-W reached the optimum on 519 of the 520 random and 80 of the 90 harder instances. By contrast, the observation that no bound in our runs exceeds the optimum or the best upper bound is not evidence: the search never tests trial values at or above the known optimum or upper bound, so it holds by construction.

Table S3: The window-inference bound UPB-W. Top: 98 proven optima ($n_1 \in \{20, 30\}$). "Gap closed" is (LB − LB*)/(OPT − LB*) in %, averaged over the 95 instances with LB* < OPT; "dist." is the mean of (OPT − LB)/OPT in % over all instances in the row. "Previous best" is the maximum of the split-cut, nested, single-stage and shaving tests without window inference. "= OPT" counts instances where the bound equals the optimum (3 already with LB*). Bottom: development set, 15 instances per n_1, gain over LB* in % with the number of instances raised; "previous best" is the split-cut and nested portfolio without shaving. CPU is seconds of one thread, mean / maximum per instance, at a nominal budget of 120 s that is checked only between trial tests. The 300 s and 600 s columns come from a later exploratory run on one machine with the door caps in every test; there the actual CPU time reached up to 1.5 times the nominal budget (2.0 times on the optima with $n_1 = 30$ at 120 s, where the maximum was 244 s). Intervals are 95% bootstrap intervals over instances.

Proven optima, 120 s nominal							
Subset	n	LB* dist.	Prev. best closed	UPB-W closed	UPB-W dist.	= OPT (prev./UPB-W)	CPU UPB-W
All	98	5.48	61.0 [57.3, 64.6]	73.9 [70.2, 77.4]	1.43	7 / 11	60 / 244
$n_1 = 20$	77	6.05	64.9	75.8	1.52	7 / 11	38 / 108
$n_1 = 30$	21	3.42	47.3	67.0	1.11	0 / 0	141 / 244
$m = 2$	36	4.21	57.6	78.0	0.88	0 / 1	87 / 244
$m = 4$	4	9.43	68.2	71.2	2.74	0 / 0	75 / 108
$m = 10$	24	8.74	62.9	69.2	2.54	4 / 5	13 / 47
U(2,4)	18	5.39	63.7	75.4	1.32	0 / 0	83 / 193
U(2,10)	16	2.57	61.2	69.7	0.81	3 / 5	41 / 160

Development set, gain over LB* *(%)*						
n_1	n	Prev. best	UPB-W, 120 s	CPU	UPB-W, 300 s	UPB-W, 600 s
20	15	4.04	5.77 [4.15, 7.64] (14)	79 / 153	–	–
30	15	2.27	3.57 [2.52, 4.72] (15)	140 / 175	–	–
40	15	0.71	1.99 [1.22, 2.83] (14)	141 / 171	3.10 [2.25, 4.06] (15)	3.11 [2.24, 4.12] (15)
50	15	0.07	0.39 [0.14, 0.65] (8)	103 / 178	1.54 [1.23, 1.88] (15)	1.64 [1.31, 1.98] (15)
60	15	0	0.24 [0.09, 0.41] (10)	116 / 168	0.97 [0.77, 1.23] (15)	1.12 [0.89, 1.37] (15)
70	15	0	0 (0)	60 / 62	0.48 [0.26, 0.72] (12)	0.98 [0.76, 1.22] (15)
80	15	0	0 (0)	60 / 61	0.06 [0.00, 0.14] (3)	0.39 [0.18, 0.64] (11)
All	105	1.01	1.71 [1.23, 2.21] (61)	100 / 178	–	–

S6. Other bounding methods in full

A gain over head-sum bounds says little if a general bounding technique would achieve the same. We compared LB* with four other bounds.

Energetic reasoning. Table S4 compares LB* with a two-stage energetic-reasoning bound, which we implemented in the style of Néron et al. (2001). On all 10,500 instances, the energetic bound is only 1.23% above the rounded Chen–Song bound, while LB* is 14.00% above the energetic bound. The energetic bound is higher on 7.5% of the instances, mostly with few trucks and many doors. With ten doors per side and $n_1 = 20$ it is higher on 88.7% of the instances and LB* is 3.38% below it on average. Taking the maximum of the two adds 0.14% to LB* on average.

LP relaxation of the time-indexed model. We implemented the LP relaxation of the time-indexed model of Cota et al. (2016) equation by equation. On the paper's worked example it gives 90.06, against the published 90.0609. We added two valid cuts, which can only make the LP stronger; we call this LP TIM-LP. It keeps the published makespan linking, which the strengthened LP below replaces. Table S5 shows the results. On the 50 benchmark instances with $n_1 = 20$, the LP is never above UPB, its value is 0.687 [0.640, 0.735] of UPB on average, and it takes 23.7 s per instance against 0.008 s. At $n_1 = 30$–50 it solved six of nine instances within 30 minutes, with ratios between 0.41 and 0.74, and gave no bound on the other three. On 300 small generated instances with known optima the two bounds are closer (ratio 0.896), and the LP is higher on one of them.

The makespan linking of the published model, $C_{\max} \geq s_j + p_j$, is the weak form analysed by Wallrath et al. (2025, Prop. 1). We therefore also solved a strengthened LP. It adds three families of valid inequalities that link $C_{\max}$ to the time-indexed variables. First, step variables $z_t = [C_{\max} > t]$ with $C_{\max} \geq \sum_t z_t$, $z_t \geq z_{t+1}$ and one occupancy row per time slot and stage (if a door is busy in slot t, then $C_{\max} > t$; for an inbound door, $C_{\max} > t$ plus the shortest outbound tail). Second, one row per job and slot, $z_t \geq \sum_{s > t - p_j - q_j} x_{js}$ ($x_{js} = 1$ if job j starts at

Table S4: LB^* against a two-stage energetic-reasoning bound (ER) on all 10,500 instances. ER follows the style of Néron et al. (2001) (window reasoning per stage, precedence propagation between stages, binary search on the makespan; no shaving). Columns give mean relative differences in %, shares of instances in %, and the mean CPU of ER.

Slice	n	$\mathrm{ER}/\lceil L_{\mathrm{CS}} \rceil - 1$	$\mathrm{LB}^*/\mathrm{ER} - 1$	$\mathrm{ER} > \mathrm{LB}^*$	$\max(\mathrm{LB}^*, \mathrm{ER})/\mathrm{LB}^* - 1$	ER CPU (s)
All	10,500	1.23	14.00	7.5	0.14	0.205
$n_1 = 20$	1,500	2.27	2.83	35.1	0.90	0.012
$n_1 = 30$	1,500	1.30	8.18	14.6	0.07	0.034
$n_1 = 40$	1,500	1.14	13.29	2.2	0.01	0.088
$n_1 = 50$	1,500	1.04	16.78	0.3	0.00	0.155
$n_1 = 60$	1,500	1.01	18.51	0.0	0.00	0.251
$n_1 = 70$	1,500	0.96	19.08	0.0	0.00	0.363
$n_1 = 80$	1,500	0.86	19.33	0.0	0.00	0.531
$m = 2$	2,100	1.88	17.91	0.0	0.00	0.315
$m = 4$	2,100	1.20	16.77	1.0	0.01	0.191
$m = 10$	2,100	1.03	11.75	19.3	0.55	0.117
U(2,4)	2,100	1.31	13.94	3.0	0.02	0.239
U(2,10)	2,100	0.70	9.65	14.0	0.12	0.162
$m = 10$, $n_1 = 20$	300	4.11	−3.38	88.7	3.59	0.003

s; q_j is the longest successor of inbound j, and 0 for outbound trucks). Third, the time-reversed Queyranne inequalities of Wallrath et al. (2025, Thm. 3), summed over the m identical doors of a stage: $\sum_{j \in S} p_j (C_{\max} - \theta - s_j) \geq \frac{1}{2}\big(p(S)^2/m + \sum_{j \in S} p_j^2\big)$ for job sets S whose tails are at least θ. The last two families are added as cuts. The cutting-plane loop stops after 30 minutes, and the bound is the value of the last LP solved to optimality, which is valid at every round. On the 300 small instances the strengthened LP never exceeds the optimum.

The strengthened linking matters. On the 50 benchmark instances with $n_1 = 20$ it raises the LP by 36% on average, to 0.896 [0.874, 0.918] of UPB; it is above UPB on two instances, equal on three and below on 45. Both have ten doors per side, and the LP is higher by 3 and 4 time units. With two doors per side the ratio rises from 0.51 to 0.80. It costs 451 s per instance on average (maximum 1117 s), against 23.7 s for the published model and 0.008 s for UPB; 17 of the 50 runs stopped at the time limit or on a solver error before the loop converged. At $n_1 = 30$–50 it gives a bound on eight of the nine instances, with ratios between 0.62 and 0.98, and is never above UPB. On the small instances its ratio is 0.977 [0.973, 0.982], and it is above UPB on 38 of 300. So the weak linking explains a large part of the gap, but not all of it: on the benchmark UPB stays higher on 45 of 50 instances, mostly with few doors, and is computed about 5×10^4 times faster.

The bound of the CP-SAT solver. After 300 s, the bound that plain CP-SAT proves by itself is 25.8% below LB^* on average, on the 14 instances with $n_1 = 20$ that neither plain CP-SAT nor CP-LB solved. CP-SAT with decision probes started from the Chen–Song bound ends 1.7% below LB^* after 300 s on 16 such instances. LB^* itself takes about 0.01 s at $n_1 = 20$.

The Lagrangian bound on single-door instances. Table S6 compares LB^* with the hybrid Lagrangian bound of Fonseca et al. (2019) on their 500 instances with one door per side. Only row means of the Lagrangian bound are published, so the comparison is between means over the same 50 instances. LB^* is higher in all ten rows, by 2.0% to 15.1%, and the difference grows with n. At $n = 60$ the published Lagrangian bound equals the mean of the second bound of Chen and Lee (2009), while LB^* is 15% higher. No LB^* exceeded the ILS upper bound. Unlike the rest of the paper, Table S6 gives the ILS gap as $(\mathrm{UB} - \mathrm{LB}^*)/\mathrm{UB}$. On the 200 instances with $n \leq 10$, where CP-SAT found all optima in at most 2 s, LB^* equals the optimum on 86. The bound takes at most 0.01 s at $n \leq 20$ and 6.4 s on average at $n = 60$.

Table S5: LP relaxation of the time-indexed model of Cota et al. (2016) against UPB: the published model (TIM-LP) and the same model with strengthened makespan linking (strengthened TIM-LP). Both are solved with HiGHS on one thread. Ratios are means of the rounded-up LP value over UPB with 95% bootstrap intervals. On the benchmark instances the horizon is the published HgR upper bound; on the small generated instances it is the CP-SAT optimum.

	TIM-LP		Strengthened TIM-LP		
Instances	LP > UPB	LP / UPB	LP > UPB	LP / UPB	CPU per instance
300 generated, $n_1 = 4$–8, optimum known	1/300	0.896 [0.884, 0.909]	38/300	0.977 [0.973, 0.982]	TIM-LP 0.06 s; strengthened 11.6 s (max 108); UPB 0.001 s
50 benchmark, $n_1 = 20$	0/50	0.687 [0.640, 0.735]	2/50	0.896 [0.874, 0.918]	TIM-LP 23.7 s (max 108); strengthened 451 s (max 1117); UPB 0.008 s
9 benchmark, $n_1 = 30$–50, 30 min limit	0/6 (3 no bound)	0.41–0.74 on 6	0/8 (1 no bound)	0.62–0.98 on 8	TIM-LP 3–1800 s; strengthened 238–1499 s

Table S6: Single-door set (one door per side) of Fonseca et al. (2019) (authors' 500 files, 50 per row). Columns 3–4 are means over the row. L_{HgR} is the hybrid Lagrangian bound as published (row means only); LB* is our bound, here the maximum of UPB, its reversed version and the two bounds of Chen and Lee (2009). "Exact" is the share of instances on which the knapsack walk finished (forward / reversed). The ILS gap is $(\mathrm{UB} - \mathrm{LB}^*)/\mathrm{UB}$ after 4 s. The last column counts instances with LB* equal to the CP-SAT optimum ($n \le 10$ only).

p range	n	L_{HgR}	LB*	LB$^*/L_{\mathrm{HgR}} - 1$	Exact (%)	ILS gap (%)	LB* = OPT
[1, 10]	5	34.7	35.5	+2.31%	100 / 100	2.88	24/50
[1, 10]	10	70.0	71.7	+2.46%	100 / 100	1.94	23/50
[1, 10]	20	143.4	155.7	+8.59%	100 / 100	1.74	–
[1, 10]	40	294.9	330.4	+12.04%	100 / 100	1.28	–
[1, 10]	60	444.3	511.3	+15.07%	72 / 94	1.19	–
[10, 100]	5	347.0	354.0	+2.02%	100 / 100	2.48	22/50
[10, 100]	10	706.9	727.1	+2.86%	100 / 100	1.96	17/50
[10, 100]	20	1442.4	1568.9	+8.77%	100 / 100	1.74	–
[10, 100]	40	2883.2	3239.3	+12.35%	100 / 100	1.42	–
[10, 100]	60	4452.3	5120.4	+15.01%	66 / 88	1.30	–

S7. The upper-bound method and its benchmarking in full

S7.1. Cost of one decode

Proposition S8. *One decode of our compiled implementation takes $O(n_1m_1 + |E| + n_2m_2 + n_2^2)$ time and $O(n_1 + n_2 + M + |E|)$ memory. One evaluation of the public EACDC code takes $\Theta(n_1n_2 + |E| \log |E| + n_1m_1 + n_2m_2 + n_2^2)$ time and $\Theta(m_1n_1 + m_2n_2 + |E|)$ memory, and the faster evaluator that we use for it takes $\Theta(n_1n_2 + n_1 \log n_1 + n_1m_1 + n_2m_2 + n_2^2)$ time and $\Theta(n_1n_2)$ memory. The forward–backward step of FB-ILS decodes the reversed instance, which costs $O(n_2m_2 + |E| + n_1m_1 + n_1^2)$.*

Proof. Our decoder scans the m_1 inbound doors for each inbound truck, reads each predecessor list once to set the release times, and for each of the n_2 outbound picks scans the m_2 doors and at most twice the remaining outbound list. The EACDC evaluator visits all n_1n_2 cells of the precedence matrix and sorts the $|E|$ predecessor pairs; for each outbound pick it scans all n_2 outbound trucks. The faster evaluator replaces the sort by two passes over the matrix and a sort of the n_1 inbound trucks. □

Since $|E| \le n_1n_2$, one of our decodes is never asymptotically more expensive than one EACDC evaluation, in time or in memory. For FB-ILS this needs $n_1 = O(n_2)$, which holds on the benchmark ($0.8n_1 \le n_2 \le 1.2n_1$). With $n_2 = \Theta(n_1)$ and fixed density the advantage

over the faster evaluator is asymptotically a constant factor, and over the original code a factor $\Theta(\log n_1)$. With $|E| = o(n_1 n_2)$ and $n_2 = o(n_1)$ our decode is strictly cheaper: with in-trees and $n_2 = 2\sqrt{n_1}$ the measured operation-count ratio grows as about $n_1^{0.43}$ (exploratory), and at fixed density 0.05 it still grows at practical sizes, because our stage-2 scan stops at the first released truck (Section S10). For a reversed decode, n_1 and n_2 swap. Every decode that ILS performs is counted in the evaluation-matched comparison, including those of the forward–backward step. The comparison at equal evaluation counts is therefore a comparison of the two algorithms that does not depend on how fast either decoder is implemented, up to constant factors. Measured on one machine (Table S11), the operation count of an EACDC evaluation is 2.0 to 2.8 times ours at $n_1 = 20$ to 80, and the ratio grows with n_1.

Table S7 gives the worst-case cost of every method that we compare or cite, from the authors' own lemmas where they state one and derived from their descriptions otherwise. Against the fast EACDC evaluator, our decode is strictly cheaper in memory whenever the precedence is sparse, $|E| = o(n_1 n_2)$. In time it is strictly cheaper only if, in addition, $n_2 = o(n_1)$ and $m_1 = o(n_2)$, because both evaluators contain the n_2^2 term of the outbound picks. Against the public EACDC code, the sort of the $|E|$ predecessor pairs makes its evaluation a factor $\Theta(\log n)$ more expensive than ours on dense precedence with $n_1 = \Theta(n_2)$ and bounded M, as on the benchmark; in memory the two are equal up to constants. Against HgR, one decode costs the same order as ours, but each HgR round also builds and solves two linear programs with $\Theta(\bar{p}\, n^2/m)$ columns, where $\bar{p}$ is the mean processing time. That size grows with the magnitude of the processing times, and the solve has only the generic bound of a linear-programming method. One iteration of our search is strongly polynomial and, for bounded M, $|E| = \Theta(n_1 n_2)$ and $n_1 = \Theta(n_2)$, linear in the size of the input. On the benchmark itself ($|E| \approx n_1 n_2/2$, $0.8 n_1 \le n_2 \le 1.2 n_1$, $M \le 10$), our decode and the fast EACDC evaluator are therefore equal up to constants in time and memory, and the measured advantage of Table S11 is a constant-factor one. The one-pass constructive heuristics CDH and PCH cost less than our 14 constructive orders, and L_{CS} and $\mathrm{LB}_{\mathrm{PCH}}$ cost less than any of our bounds: UPB is exponential in the worst case, its deterministic version is polynomial with large constants, and $\mathrm{LB}_U^{\mathrm{LP}}$ needs a few maximum flows. The polynomial part of UPB-W is $O(n^4 \log^2 P)$ plus $O(n \log P)$ linear programs; its CP-SAT tests are capped by a budget.

S7.2. Variants and design choices

FB-ILS. The *Forward–Backward Iterated Local Search* (FB-ILS) extends the procedure above with two mechanisms: forward–backward improvement passes and window insertion. We call the procedure above, which has neither, the *base ILS*; it is the ablation of FB-ILS without these mechanisms. In the results, "ILS" without a prefix refers to the base ILS. FB-ILS was tested in a separate preregistered run. It is compiled to C (about 45 times more decodes per second than our Python decoder), it draws 90% of the insertion targets within ±25% of n_1 positions from the source, and it applies a forward–backward improvement step: the schedule is re-sequenced on the reversed instance by decreasing outbound completion time, decoded, and mapped back, as long as the makespan strictly improves. Every decode, including those of this step, is counted. It was chosen among more than ten variants on the development instances only. Iterated greedy (Ruiz and Stützle, 2007), bundle moves on trucks with shared successors, late acceptance and moves aimed at the critical predecessor were all weaker per evaluation; iterated greedy is kept as a comparator. Section 8.4 reports the confirmation.

Memetic EACDC. To test whether the population of EACDC adds value when combined with the local search, we built a memetic variant. EACDC, with its published static configuration, runs for 40% of the budget. The local search then runs from the best of three starting orders: the inbound order derived from the best EACDC individual, the best constructive order and an order derived from the bound (described under "A negative result: ordering by knapsack solutions" in Section S7.3).

Table S7: Worst-case cost of the methods compared or cited. $n = n_1 + n_2$, $M = \max(m_1, m_2)$, $|E| = \sum_j |N(j)|$ (about $n_1 n_2/2$ on the benchmark), $P = P_1 + P_2$ the total processing time (the horizon T of the time-indexed models), $\bar{p}$ the mean processing time, B the evaluation budget of a search. Basis: S, stated and proved by the authors; D, derived by us from the authors' description or from the code behind our numbers; G, generic solver, no polynomial bound. Measured costs are in Table S11 and Section 8.

Method	Time	Memory	Basis	Source
Prior work on the same problem				
L_{CS}, both directions	$O(\|E\|+n_1+n_2 \log n_2)$; the paper states no bound	$O(n)$ beyond the input	D	Chen and Song (2009), Eqs. (11)–(12)
JRH, JLPTH	$O(n_1 n_2 + n_1 \log n_1 + n_1 m_1 + n_1 n_2 m_2 \log n_2)$; JLPTH with $n_1 n_2^2 m_2 \log n_2$ in the last term	$O(n + \|E\|)$	S	Chen and Song (2009), Lemmas 1–2
DJRH, DJLPTH	$O(n_1 n_2^2 + n_1 n_2 \log n_1 + n_1 m_1 + n_2 m_2 \log n_2)$; DJLPTH with $n_2^2 m_2 \log n_2$ in the last term	$O(n + \|E\|)$	S	Chen and Song (2009), Lemmas 3–4
MIP (CTP)	$O(n_1 m_1 + n_2 m_2 + n_1^2 + n_2^2)$ binaries, $\Theta(m_1 n_1^2 + m_2 n_2^2 + \|E\|)$ big-M rows; branch and bound, exponential	size of the model	D, G	Chen and Song (2009), Eqs. (1)–(8)
TIM	$(n_1+n_2)(T+1)$ binaries and $2n_2+1$ continuous variables; $\Theta(n+T+\|E\|)$ rows and $\Theta(PT+\|E\|T)$ nonzeros, i.e. $\Theta(\bar{p}^2 n^2 + \bar{p}\, n\|E\|)$ with $T = P$; pseudo-polynomial size, exponential solve	$\Theta(\bar{p}^2 n^2 + \bar{p}\, n\|E\|)$	D, G	Cota et al. (2016), Eqs. (9)–(18)
CDH	$O(\max\{n_1 n_2, n_2 \log n_2\})$ for $n_i > m_i$	$O(n + \|E\|)$	S	Cota et al. (2016), Thm. 2
PCH	$O(\max\{n_2 m_2, n_1 \log n_1, n_2 \log n_2, n_1 n_2 m_1, n_1 n_2 \log n_1\})$	$O(n + \|E\|)$	S	Nogueira et al. (2020), Lemma 1
$\mathrm{LB}_{\mathrm{PCH}}$ (Steps 1–2)	$O(\|E\| + n_1 \log n_1)$	$O(n)$	D	Nogueira et al. (2020), Thm. 1, Lemma 1
Lagrangian bound, one door per side	per Volume iteration $O(\|E\| + n_1 \log n_1 + n_2 \log n_2)$, at most 1000 iterations; $\Theta(nT)$ per iteration if the averaged primal $\bar{x}$ is stored time-indexed, as described	$O(\|E\|)$; $\Theta(nT)$ for $\bar{x}$	D	Fonseca et al. (2019), Secs. 3.1, 4.2
HgR, multi-door	per round two LPs with $\Theta(\bar{p} n^2/m)$ columns (no strongly polynomial bound), one decode per H1/H2, swap search $O((T_0 - \mathrm{LB})n^2)$ decodes; at most 5 rounds	$\Theta(\bar{p} n^2/m)$	D, G	Fonseca et al. (2019), Sec. 5.3, Alg. 1; public script
EACDC evaluation (public code)	$\Theta(n_1 n_2 + \|E\| \log \|E\| + n_1 m_1 + n_2 m_2 + n_2^2)$	$\Theta(m_1 n_1 + m_2 n_2 + \|E\|)$	D	Nogueira et al. (2022); Prop. S8
EACDC evaluation (fast)	$\Theta(n_1 n_2 + n_1 \log n_1 + n_1 m_1 + n_2 m_2 + n_2^2)$	$\Theta(n_1 n_2)$	D	Prop. S8
EACDC run	$\mu + \mathrm{NGEN} \cdot \lambda$ evaluations plus $\Theta(\lambda n_2 + \mu k)$ per generation	$\Theta((\mu+\lambda) n_2)$	D	Nogueira et al. (2025); Nogueira et al. (2022)
EACDCml inference	$O(n_1 n_2)$ features and a constant-size network ($\approx 10^6$ multiply-adds); then one EACDC run with $\leq$ 40,200 evaluations	$\approx 10^6$ weights	D	Nogueira et al. (2025)
CP-SAT, any model	exponential in the worst case; no polynomial bound on time or memory	not bounded a priori	G	–
This paper				
Decoder	$O(n_1 m_1 + \|E\| + n_2 m_2 + n_2^2)$; on $R(I)$ with n_1^2 in place of n_2^2	$O(n + M + \|E\|)$	D	Prop. S8
Base ILS	14 constructive orders, $O(n_1^2 n_2^2)$ for union-greedy; then per iteration $O(n_1)$ plus one decode; B iterations	$O(n_1 n_2)$ construction, $O(n + \|E\|)$ search	D	Sec. 5
FB-ILS	as base ILS; an improving iteration adds at most 9 counted decodes and $O(n_1^2 + n_2^2)$ sorting	$O(n + \|E\|)$	D	Sec. 5
UPB, exact	up to $c_{\mathrm{full}} \leq P_1$ knapsacks, each strongly NP-hard: exponential	$O(P_1 + n + \|E\|)$	D	Prop. S3, Sec. 4.4
UPB-det	at most 50 steps of bounded work (DFS of $\leq$ 30,000 nodes at $O(n_1 n_2)$ each, or CP-SAT capped at 0.5 deterministic units), $O(P_1)$ array work per step: polynomial in n_1, n_2, P_1	$O(P_1 + n + \|E\|)$	D	Sec. 4.4
UPB-W	trials T by doubling, then bisection: $O(\log P)$ trials. Per trial at most 18 shaving rounds, each with $O(n \log P)$ window tests (propagation, at most 50 passes of $O(n_1^2 n_2)$; flow and energetic tests, $O(n^3)$ each) and linear probes at 16 cut times ($O(n)$ linear programs with $O(n)$ variables and $O(n + \|E\|)$ rows). Polynomial part: $O(n^4 \log^2 P)$ plus $O(n \log P)$ linear programs (polynomial with an interior-point method; we use the simplex code GLOP). Then CP-SAT tests (shaving, two single-stage tests, a nested split-cut test with up to 16 cut times), each capped: exponential in the worst case. The whole search stops at a CPU budget, checked between tests	$O(n^2)$ for the polynomial part; CP-SAT models with $O(16(n + \|E\|))$ variables and constraints	D, G	Sec. 4.6, Sec. S5
$\mathrm{LB}_U^{\mathrm{LP}}$	at most $2(n_1+n_2)+1$ maximum flows on $n+2$ nodes and $n+\|E\|$ arcs: strongly polynomial; one parametric flow would suffice (Gallo et al., 1989)	$O(n + \|E\|)$	D	Prop. S5
Exact method (CP-LB, CP-UC)	UPB once, 10 s of ILS, then CP-SAT on a model with $O(n+\|E\|)$ variables and constraints ($O(k(n+\|E\|))$ more for the $k = 12$ cut times of CP-UC): exponential, capped at 300 s	as CP-SAT	D, G	Sec. 6

Table S8: ILS against the static-configuration EACDC, matched run by run. Values are the mean paired difference $(UB_{ILS} - UB_{EACDC})/UB_{EACDC}$ in %, with 95% bootstrap intervals over instances. Negative values favour ILS. W/T/L counts the instances (seed means) on which ILS is better, equal (relative tolerance 10^{-9}) or worse. The confirmation instances were generated after the protocol was registered. The development instances were used to design ILS.

	Confirmation: 210 fresh instances, seeds 11–13		Development: 105 instances, seeds 1–5	
Budget matched on	Mean [95% CI]	W/T/L	Mean [95% CI]	W/T/L
Evaluations (generations)	−0.48 [−0.58, −0.37]	158/2/50	−0.66 [−0.80, −0.53]	91/1/13
Evaluations (with initial pop.)	−0.53 [−0.63, −0.42]	161/5/44	−0.70 [−0.84, −0.57]	94/1/10
CPU time	−0.57 [−0.67, −0.46]	161/5/44	−0.74 [−0.88, −0.62]	94/1/10
By n_1, mean only, %: CPU-matched / evaluation-matched (generations)				
$n_1 = 20$	−0.97 / −0.92		−1.20 / −1.10	
$n_1 = 30$	−0.86 / −0.78		−0.75 / −0.64	
$n_1 = 40$	−0.50 / −0.37		−0.49 / −0.46	
$n_1 = 50$	−0.36 / −0.27		−0.90 / −0.82	
$n_1 = 60$	−0.39 / −0.29		−0.72 / −0.66	
$n_1 = 70$	−0.52 / −0.44		−0.73 / −0.64	
$n_1 = 80$	−0.36 / −0.28		−0.42 / −0.33	
EACDC CPU per run, s: mean (median, max)	4.42 (3.15, 23.01)		5.66 (3.54, 29.39)	
Evaluations per run: EACDC / ILS	30,476 / 16,200		30,924 / 16,935	

S7.3. Further upper-bound results

Preregistered confirmation. Table S8 gives the matched comparison of ILS and the static configuration of EACDC. On the 210 confirmation instances, the mean relative difference between ILS and EACDC is −0.48% [−0.58, −0.37] when both methods evaluate the same number of schedules (W/T/L 158/2/50). This is the comparison that does not depend on the speed of either implementation. At equal CPU time the difference is −0.57% [−0.67, −0.46] (W/T/L 161/5/44). Both endpoints were confirmed, with Holm-adjusted $p = 3 \times 10^{-5}$. The difference is largest at $n_1 = 20$ (−0.97% CPU-matched) and between −0.36% and −0.52% for $n_1 \geq 40$. The order in which the two methods ran in the process did not matter (−0.55% at run level with EACDC first and −0.59% with ILS first). On the development set the differences were larger (−0.66% evaluation-matched and −0.74% CPU-matched). Part of this excess is probably due to the fact that ILS was designed on those instances.

Distance to the optimum. On the 62 instances with proven optima (Table S15), ILS after 10 s is 0.41% [0.29, 0.54] above the optimum on average and reaches it on 41.9% of the instances. One default EACDC run, which takes 1–2 s at this size, is 1.71% [1.35, 2.05] above the optimum, and the published HgR solutions are 3.38% [2.93, 3.84] above it. On the 165 known optima of the exact set (118 instances with $n_1 = 20$ and 47 with $n_1 = 30$, solved by CP-UC or by a variant without cuts; Section 8.2), ILS after 10 s is 0.40% and 0.10% above the optimum, one EACDC run 1.59% and 0.90%, and HgR 3.53% and 2.73%.

CP-SAT as an upper-bound method. Table S9 compares ILS with CP-SAT at the same CPU budget. CP-SAT started from the same constructive schedule is 1.73% worse at 4 s and 1.81% worse at 10 s, and without the start it is about 6% worse. Even at $n_1 = 20$, warm-started CP-SAT is 1.35% worse at 10 s and better on only 3 of 15 instances. At these budgets CP-SAT is not competitive as an upper-bound method, although with 300 s it does prove optimality for many $n_1 = 20$ instances.

The memetic variant. On the development set with a 10 s budget (seeds 1–3), the memetic EACDC differs from EACDC with restarts by −0.65% [−0.84, −0.48] (W/T/L 90/2/13), so it is better, and from ILS alone by +0.18% [+0.09, +0.27] (27/16/62), so it is worse. Spending 40% of the budget on the population phase thus costs more than it gains at these budgets.

Table S9: CP-SAT (one worker) as an upper-bound method at the same CPU budget as ILS on the 105 development instances, seeds 1–3. Values are mean $\mathrm{UB}_{\text{CP-SAT}}/\mathrm{UB}_{\text{ILS}} - 1$ in % with 95% bootstrap intervals; positive values favour ILS. "Warm" receives the same constructive schedule as ILS as a hint, and its constructive time is counted.

Budget	CP-SAT cold vs ILS (W/T/L of CP-SAT)	CP-SAT warm vs ILS (W/T/L of CP-SAT)
4 s	+6.62 [6.12, 7.14] (2/0/103)	+1.73 [1.54, 1.94] (2/1/102)
10 s	+6.17 [5.66, 6.69] (3/1/101)	+1.81 [1.60, 2.02] (3/1/101)

Table S10: Tuning burden. Both methods were tuned with Optuna (TPE) for the same number of trials on 25 training instances that are disjoint from the 105 development instances. The objective was the mean ratio to the HgR upper bound at 4 s of CPU. The test row compares tuned and default configurations on the development instances (2 seeds, 4 s); positive values mean that the tuned configuration is worse.

	ILS	EACDC (static)
Tunable parameters	3	6
Tuning cost	1.11 CPU-h (40 trials)	1.26 CPU-h (39 trials)
Training objective, default → best	0.9761 → 0.9749	0.9919 → 0.9871
Spread over 11 random configurations	0.27%	0.51%
Test, tuned vs default	+0.03% [−0.02, +0.07]	+0.00% [−0.07, +0.09]

A negative result: ordering by knapsack solutions. We also tried a constructive order built from optimal knapsack sets at nine capacities, taken in increasing order. Its makespan differed from that of the best of the 14 cheap orders by +1.16% [+0.84, +1.49] (W/T/L 21/3/81), so it was worse, and it took 2.9 s instead of milliseconds. Optimal knapsack sets at different capacities are not nested, so their union brings large blocks of inbound trucks forward too early. The greedy union orders already capture the useful structure of the bound at negligible cost.

Tuning, variability and cost. Table S10 reports the tuning experiment. EACDC is about twice as sensitive to its configuration as ILS (spread 0.51% against 0.27%), but for neither method did a tuned configuration beat the default on the development set. Across seeds, the mean coefficient of variation was 0.14% for ILS after 10 s and 0.26% for one EACDC run. ILS has a solution after 0.05 s in every run. In fixed-budget runs on the development set, EACDC had a solution in 14% of runs at 0.05 s, 55% at 1 s, 84% at 4 s and 97% at 10 s. Table S11 gives the measured cost per evaluation and the memory of both methods.

FB-ILS, preregistered (with the ablation). FB-ILS (Section 5) was confirmed on 210 new instances (generator seed 20261105), with three seeds each (21–23) and all four methods in one process per run: EACDC first, then the base ILS, FB-ILS and iterated greedy in rotated order, each read at the CPU time and at the evaluation count of that EACDC run. All four endpoints were confirmed (Holm-adjusted $p = 4 \times 10^{-5}$ each). Against EACDC, FB-ILS differs by −1.09% [−1.19, −1.00] at equal CPU time (W/T/L 203/2/5) and by −0.63% [−0.72, −0.53] at equal evaluation counts (172/3/35). The comparison with the base ILS is the ablation of the forward–backward mechanism. Against the base ILS, FB-ILS differs by −0.57% [−0.63, −0.50] at equal CPU (195/15/0) and by −0.14% [−0.19, −0.10] at equal evaluations (131/21/58). The margin over EACDC at equal CPU is between −1.38% ($n_1 = 30$) and −0.84% ($n_1 = 80$), and between −1.78% (ten doors per side) and −0.66% (two doors per side); at equal evaluations it falls from −0.99% at $n_1 = 30$ to −0.39% at $n_1 = 80$. On the same runs the base ILS is −0.53% and −0.48% from EACDC, and iterated greedy is −0.87% at equal CPU but only −0.05% at equal evaluations. Most of the gain of FB-ILS at equal CPU therefore comes from the compiled decoder. At equal evaluations, where the implementation does not matter, the gain over the base ILS comes from the two search mechanisms; on the development instances, the forward–backward passes alone gave most of it (−0.79% against EACDC, against −0.72% with window insertion alone and −0.68% with neither). All runs used the frozen code (hashes checked; one

Table S11: Measured cost on one instance of the $m = 4$ group per size, one thread, fresh process. Peak memory includes about 80 MB of library imports that both methods share.

n_1	ILS evaluations in 4 s (μs per evaluation)	ILS peak memory	EACDC CPU for one run	EACDC peak memory
20	116,049 (34)	84 MB	1.36 s	159 MB
40	53,272 (75)	84 MB	2.72 s	160 MB
60	28,343 (141)	85 MB	5.83 s	165 MB
80	20,677 (194)	87 MB	14.28 s	182 MB

later edit of the library-loading line of the Python wrapper did not affect them, because each run imported the module before the edit).

Certified ratios on the whole benchmark. With the published HgR upper bounds, the ratio UB/LB* over all 10,500 instances has mean 1.096, median 1.080, 90th percentile 1.166, 99th percentile 1.243 and maximum 1.311. It is within 1.05 on 14.1% of the instances. On the development set, with the best upper bound we found (minimum over HgR, ILS and EACDC with restarts, five seeds), the mean is 1.063, the median 1.052, the maximum 1.186, and 47.6% of the instances are within 1.05. These are measured certificates, not worst-case guarantees. The ratio is larger at small n_1 (1.082 at $n_1 = 20$) than at large n_1 (1.053 at $n_1 = 80$), although the walk computes the bound exactly for every instance with $n_1 \leq 40$. The exact optima above indicate that most of this residual at small n_1 is due to the structure of the bound, not to the heuristic. We also ran FB-ILS (Section 5) once on every one of the 10,500 instances, with 10 s of CPU and seed 1, on four machines (29.2 CPU hours in total; the machine is recorded for each instance, and the per-machine means differ by at most 0.07 points). It improves the published HgR upper bound on 10,475 instances (99.8%), ties on 21 and is worse on 4. On average it is 3.06% below the published value (-3.31% at $n_1 = 20$, -3.58% at $n_1 = 30$ and -2.49% at $n_1 = 80$; -2.48% with two doors per side and -3.98% with ten). Taking for each instance the best upper bound known (FB-ILS, HgR and, on the 300 instances of the exact set, the exact runs) and the best proven lower bound (LB* or the exact runs), the certified ratio has mean 1.060, median 1.048, 90th percentile 1.115 and maximum 1.222. It is within 1.05 on 53.1% of the instances, against 14.1% with the published upper bounds, and equal to 1 (optimality proved) on 188. The per-instance table of best-known bounds is part of the data release.

S7.4. The rivals: EACDC, HgR and EACDCml

EACDC. The bounds, the decoder of the base ILS and the base ILS itself are written in Python; FB-ILS is compiled to C. Knapsacks and the exact models use CP-SAT from OR-Tools 9.15 with one worker. The time-indexed LP is solved with HiGHS (highspy 1.15.1, one thread). EACDC is the authors' public `GA.py` (Nogueira et al., 2022) with the static configuration of their Table 2 (tournament size 3, 150 generations, $\mu = 75$, $\lambda = 100$, crossover probability 0.95, mutation probability 0.4) and DEAP 1.4.4. We changed two things. First, we replaced the evaluator and the initial-population builder by faster pure-Python versions with the same tie rules. On 18 complete runs they gave the same best makespan and the same best generation as the original code, at 13.4 times lower CPU cost. On 1,000 random permutations the evaluator returned the same values, and a known-answer test compares our harness with the unmodified `GA.py`. Using the faster version favours EACDC in every CPU-matched comparison. Second, the public code fixes the random seed; we pass the run's seed instead, to measure variation across seeds. The value we record for an EACDC run is the best makespan it ever evaluated. This is never worse than the value that the code returns, which again favours EACDC.

All runs used one thread per process at low priority, on Apple-silicon computers and, for some runs, one x86 laptop under Linux (Section S11 lists the versions). CPU time is process time. We observed that the CPU time of identical deterministic EACDC runs differed by a median factor of 1.40 between two run windows with different machine load. For this reason we never compare CPU times measured in different runs or windows.

Table S12: Preregistered comparison with our re-implementation of HgR on 210 new instances at equal CPU time (one machine, seeds 41–43). Difference is the mean over instances of $(UB_{ILS} - UB_{HgR})/UB_{HgR}$ in %, each instance averaged over seeds, with a 95% bootstrap interval; negative favours ILS. HgR-best is the better of HgR with and without its linear program in each run. p is the one-sided Wilcoxon signed-rank p-value, Holm-adjusted over the four endpoints. Only H2 compares two Python implementations.

Endpoint	Comparison	CPU	n	Difference (%)	W/T/L	Holm p
H1	FB-ILS vs HgR	10 s	210	−2.89 [−3.07, −2.71]	210/0/0	7×10^{-36}
H2	base ILS vs HgR	10 s	210	−2.54 [−2.72, −2.37]	209/1/0	7×10^{-36}
H3	FB-ILS vs HgR-best	10 s	210	−2.80 [−2.97, −2.63]	209/0/1	7×10^{-36}
H4	FB-ILS vs HgR	60 s	60	−3.03 [−3.42, −2.67]	60/0/0	8×10^{-12}

HgR at matched time (preregistered). The public code of HgR (Fonseca, 2019) is an AMPL script whose model file is not public. We re-implemented its multi-door loop (a linear relaxation, truck weights, schedules from H1 and H2, swap local search), rebuilding the linear program from eqs. (37)–(49) of Fonseca et al. (2019) with the reading most favourable to HgR where the description leaves room. Our version reproduces the published HgR upper bounds on the 105 development instances (mean difference 0.00% [−0.22, +0.24]; 51 better, 3 equal, 51 worse). Exploratory runs there (ILS 2.58% and FB-ILS 2.98% below HgR at 10 s) led to the preregistered test of Section 7, run on one machine, which confirmed all four endpoints (Table S12). The base ILS, which like our HgR is written in Python, is 2.54% below HgR. Giving HgR its better version in each run changes little, because the linear program makes no measurable difference (−0.07% [−0.17, +0.02] with against without). HgR does not use a longer budget: at 60 s it stops by itself, when a round brings no improvement, in 167 of 180 runs (median 45 s). Every door group and every n_1 has a negative mean, from −2.25% with two doors per side to −4.14% with ten. The only runs over budget are HgR runs in which HiGHS exceeded its time limit, which favours HgR, and all 2,880 final schedules were checked. The results concern our re-implementation, not the authors' AMPL and CPLEX runs.

The published HgR solutions. On the development set, ILS after 10 s is 2.56% [2.33, 2.80] below the published HgR upper bounds (W/T/L 104/0/1), and one default EACDC run 1.63% [1.42, 1.83] below (96/0/9); the HgR runs took 1 to 2.5 minutes per instance on other hardware, so these numbers are not matched. Of the HgR gap against LB* on this set (9.38%), one EACDC run closes 19% and ILS 30%, leaving 6.54% [5.81, 7.32].

Comparison with EACDCml. EACDCml (Nogueira et al., 2025) sets the five parameters of EACDC for each instance with a neural network trained on Optuna calibrations. The trained network is not public, so we rebuilt the method and gave it the authors' calibration budget: 875 Optuna trials over 35 training instances (one per door group and n_1), which is at or above the about 565 trials that the authors' time cap allowed. We built two versions. The code-based rebuild, called simply EACDCml in the paper, drives the authors' public EACDC code with a network trained on calibrations of that code. EACDCml (paper-based) follows the paper's description of the algorithm; the end of this section lists the choices we had to make where the description leaves room. On 210 benchmark instances that were used neither for design nor for training, with the paper's gap convention, EACDCml reproduces the published mean gap to −0.10 percentage points and EACDCml (paper-based) to +0.66 points. Run seeds move the mean gap by only 0.012–0.026 points, while a sample of 210 instances carries a sampling interval of about ±0.75 points, so both versions agree with the published value within the instance-sampling error.

We then compared FB-ILS and the base ILS with both versions in a preregistered test on the 210 instances of the first confirmation set, with new seeds (31–33), all arms in one process per run and one machine for all runs. Each search was read at the CPU time and at the evaluation

count of each rival run. The main claim uses EACDCml, the stronger rival. Against it, FB-ILS differs by −1.15% [−1.25, −1.05] at equal CPU (W/T/L 203/3/4) and by −0.63% [−0.73, −0.53] at equal evaluations (168/3/39); the base ILS differs by −0.61% [−0.70, −0.51] and −0.56% [−0.66, −0.46]. Against EACDCml (paper-based) the differences are larger: −2.11% and −1.60% for FB-ILS, and −1.52% at both matchings for the base ILS, which is −1.71% from the static configuration of EACDCml (paper-based). All nine endpoints were confirmed after Holm's correction within their family ($p \leq 5 \times 10^{-5}$). The margin of FB-ILS over EACDCml shrinks with the instance size, from about −1.4% at $n_1 = 30$ to about −0.9% at $n_1 = 70$ at equal CPU, and from about −1.0% to −0.5% at equal evaluations, but it is negative in every size class. On the same runs, the network of EACDCml (paper-based) improved on its static configuration by 0.19% of the makespan. In points of the gap, its gain in the reproduction above was 0.15, against the 0.27 points reported by Nogueira et al. (2025). The conclusions are robust to a ten times larger calibration budget. With 8,750 Optuna trials instead of 875 and the network retrained, the gap of EACDCml (paper-based) moves from 23.83% to 23.64% [22.86, 24.39] (23.17% in Nogueira et al. (2025)), and the gain of the network over the static configuration grows from 0.15 to 0.34 points of gap [0.28, 0.39], which recovers the 0.27 points reported by the authors. Against this recalibrated version, FB-ILS differs by −1.89% [−2.03, −1.75] at equal CPU (208/1/1) and by −1.43% [−1.58, −1.29] at equal evaluations (195/1/14), and the base ILS by −1.34% at both matchings, so the margins shrink by about 0.2 points only. These figures are read from the preregistered traces; in about 157 of the 630 runs the recalibrated rival used more CPU or evaluations than our search had recorded, and there our search is read at its last value, which biases the comparison against it. Even at ten times the budget, EACDCml (paper-based) remains 0.77% [0.61, 0.92] above EACDCml, so the main claims, made against the authors' code, do not depend on the calibration budget.

The EACDCml rebuild: choices D1–D9. The description of EACDCml in Nogueira et al. (2025) and the public code (Nogueira et al., 2022) differ in some details, and the trained network is not public. We followed the paper wherever it is explicit. Where the paper leaves room, we made the following choices (D1–D9). The code-based EACDCml uses the public code as it is and only adds the network.

D1. *PCH seed order.* The trucks are sorted by decreasing tail, with longest-processing-time ties, as in PCH (Nogueira et al., 2020) and in the public code.

D2. *Swap search from the PCH order.* Swaps are visited in lexicographic order. The final version uses the first-improvement search of the public code, which restarts after each improvement and keeps every evaluated neighbour.

D3. *Crossover offspring.* Each call keeps the first child, as the evolutionary framework of the public code does.

D4. *Displacement mutation.* The segment length is uniform in $\{1, \ldots, \lfloor n_1/2 \rfloor\}$, and its start and insertion point are uniform.

D5. *Operator success.* A crossover counts as successful if its child is better than both parents.

D6. *Operator probabilities.* They are updated immediately after each operation, as in the paper's worked example.

D7. *Calibration sampler.* Optuna's Gaussian-process sampler with ten random start-up trials, which corresponds to the Bayesian sampler of the public code. The trial count per instance is $\max(5, \mathrm{round}(n_1/2))$, as in the public code, with the paper's 3,600 s cap per instance; the cap never bound.

D8. *Network training.* The architecture, optimiser and scaling of the public code, trained by full-batch Adam for 2,000 epochs with a fixed seed.

D9. *Predicted parameters.* They are clipped to the paper's ranges and rounded to integers where needed.

EACDCml (paper-based) also runs all generations (the public code does not stop early). The 35 training instances follow the paper's rule (one random instance per door group and n_1) with our own draw, because the authors' list is not public.

S8. Anytime profile and a tuned iterated greedy

All earlier comparisons were made at the rivals' own, short budgets. To test whether FB-ILS is better or only converges faster, a further preregistered experiment read every method at 1, 10 and 100 times the rival's evaluation count. It used 70 instances of the first confirmation set (the first two of each cell) and one new seed. The rivals were extended as follows. EACDC (static configuration) was restarted: level k is the best of k independent runs. EACDCml was run with its number of generations multiplied by k. HgR was run to its own stopping rule, which it reaches on 66 of the 70 instances; it cannot use more effort, so its value is the same at every level. FB-ILS and iterated greedy were run once to the largest budget and read on their anytime traces at the evaluation count of each rival and level, counting every decode. Iterated greedy (Ruiz and Stützle, 2007) was tuned on the development instances only: a full grid of 72 settings (destruction size $d \in \{1, 2, 3, 4, 6, 8\}$, temperature $T \in \{0, 0.1, 0.2, 0.4, 0.8, 1.6\}$, with and without local search), followed by a final round of the five best settings at 100 times the budget. This is a larger tuning budget than FB-ILS received. The winner was $d = 3$, $T = 0$, with local search. All 15 endpoints of Table S13 were tested together with Holm's correction. This study is a separate preregistered family, not part of Table 1; 14 of its 15 endpoints were confirmed.

FB-ILS remained better than every rival at every budget (Table S13). Against EACDC with restarts the margin was -0.75%, -0.63% and -0.67% at 1, 10 and 100 times the evaluations. It did not shrink: the change from 1 to 100 times is $+0.07$ points $[-0.10, +0.27]$. Against EACDCml with scaled generations the margin grew from -0.71% to -0.88%, and against HgR from -2.16% to -2.84%. At 100 times the EACDC evaluations (3.1 million on average), FB-ILS is on average 0.02% above the best schedule any method found on the instance. EACDC with 100 restarts is 0.70% above it, and EACDCml with 100 times the generations 1.00%. The tuned iterated greedy beats EACDC with restarts at 10 and 100 times the budget (-0.35% and -0.38%), but not at the original budget (-0.17% $[-0.43, +0.09]$). It stays behind FB-ILS at every level (-0.58%, -0.28%, -0.29%). The development tuning did not transfer: on the test instances the setting chosen in an earlier development round ($d = 4$, $T = 0.4$) was as good at 1 and 10 times the budget and slightly better at 100 times (0.12 points). The advantage of FB-ILS is therefore not only an advantage in convergence speed. Within 100 times the rivals' budgets it is a stable gap of 0.6–0.9% to the EACDC line and of 2–3% to HgR. On two instances, HgR with the linear program made only two decodes before the solver exhausted its budget; there FB-ILS is read at its first schedule. Excluding the two instances changes the HgR margins by at most 0.13 points.

S9. The exact method: variants, timing and slack

Variants. We compare four variants. CP-plain is the base model without the head and tail cuts, in optimisation mode with the same hint and no lower bound (Section 6). CP-CS is the procedure above started from the bound of Chen and Song (2009) instead of LB*. CP-LB starts from LB*; it uses the bound only at the root. CP-UC is CP-LB with the forward and reversed union cuts at 12 cut times added to every model that it solves.

Table S13: Anytime profile at matched evaluations (preregistered; 70 instances of the first confirmation set, one seed). Values are mean paired differences $(\mathrm{UB}_A - \mathrm{UB}_R)/\mathrm{UB}_R$ in % with 95% bootstrap intervals over instances; negative values favour A. k multiplies the rival's evaluation count: EACDC is restarted (k independent runs), EACDCml runs k times its generations, HgR runs to its own stopping rule (its value does not change with k) and the searches are read at k times its decodes. Iterated greedy (IG) was tuned on the development instances. All 15 endpoints are Holm-adjusted together; every adjusted $p \leq 2 \times 10^{-4}$ except IG vs EACDC at $k = 1$ ($p = 0.22$).

A vs R	$k = 1$	$k = 10$	$k = 100$
FB-ILS vs EACDC (restarts)	−0.75 [−0.96, −0.54]	−0.63 [−0.76, −0.49]	−0.67 [−0.80, −0.55]
FB-ILS vs EACDCml (generations $\times k$)	−0.71 [−0.92, −0.51]	−0.78 [−0.96, −0.60]	−0.88 [−1.04, −0.72]
FB-ILS vs HgR (own stopping rule)	−2.16 [−2.48, −1.85]	−2.58 [−2.91, −2.26]	−2.84 [−3.16, −2.52]
Tuned IG vs EACDC (restarts)	−0.17 [−0.43, +0.09]	−0.35 [−0.51, −0.19]	−0.38 [−0.51, −0.26]
FB-ILS vs tuned IG	−0.58 [−0.73, −0.43]	−0.28 [−0.40, −0.17]	−0.29 [−0.41, −0.18]
W/T/L of FB-ILS vs EACDC: 53/5/12, 61/5/4, 65/3/2; vs tuned IG: 59/6/5, 37/24/9, 31/36/3			
Mean evaluations of EACDC (restarts)	31,409	314,093	3,140,934
Mean CPU, s: EACDC (Python) / FB-ILS (C) at the same evaluations	3.5 / 0.04	34.7 / 0.31	350 / 2.2
Mean distance to the best schedule found, %: FB-ILS / EACDC / tuned IG	0.56 / 1.32 / 1.15	0.25 / 0.89 / 0.53	0.02 / 0.70 / 0.31

Table S14: Effect of starting CP-SAT from LB^* as a root bound only (no cuts inside the search), on 30 instances with $n_1 = 20$ and 300 s per run. CP-LB starts its decision probes and its optimisation run from LB^*; CP-CS is the same procedure started from the Chen–Song bound; CP-plain is plain CP-SAT with the same warm start and no bound. Times are CPU seconds on instances that both arms solve.

A vs B	Solved	Median CPU, both solved	Time ratio A/B (geometric mean)	Final bound higher: for A / for B
CP-LB vs CP-CS	13 vs 14	47.5 vs 49.9 s ($n = 13$)	1.01	40% / 3%
CP-LB vs CP-plain	13 vs 12	35.7 vs 10.2 s ($n = 9$)	0.96	60% / 10%

The bound at the root only. Table S14 compares the variants that use a bound only at the root, on 30 instances with $n_1 = 20$. Starting from LB^* instead of the Chen–Song bound does not shorten the time to prove optimality (geometric-mean time ratio 1.01; 13 against 14 instances solved). Compared with plain CP-SAT the ratio is 0.96, and plain CP-SAT was faster in median on the instances that both solved. What the root bound does change is the final lower bound on the instances that remain open: it is 2.85% higher than with probes from the Chen–Song bound and 40.1% higher than with plain CP-SAT. Used only at the root, the bound therefore improves the certificate but does not speed up the proof.

Timing on one machine. The main CP-UC runs and the variants without cuts were timed on different machines (an Apple M5 Pro and an Apple M3 Max). Because all variants are our own implementations, we re-timed CP-UC and CP-LB on one machine, in the same process for each instance and in alternating order, on the 55 instances that both had solved, with a 60 s limit (0.75 CPU hours in total). There CP-UC proved optimality on 49 of the 55 instances within 60 s and CP-LB on 26. Its median time to proof was 4.8 s. The median for CP-LB is censored: 29 of its 55 runs stopped at the limit without a proof, so its median time to proof lies above the limit. No instance was solved by CP-LB alone, and on the 26 easy instances that both solved within 60 s the two times were similar (medians 2.0 and 3.4 s). On the original runs the median times were 4.7 s for CP-UC and 47.0 s for CP-LB. Scaled by the ratio of the PassMark single-thread ratings of the two processors (PassMark Software, 2026), 5,898 against 4,781, the CP-UC median corresponds to about 5.8 s on the machine of the variants without cuts. The union cuts of Section 6 add $O(k(n_1 + n_2 + |E|))$ Boolean variables and constraints to the model ($k = 12$ cut times, both directions), so a search node of CP-UC is not cheaper than one of CP-LB(about 18,000 against 29,000 branches per CPU second); the faster proofs come from the propagation of the cuts. The CP-SAT search counts agree (one re-run on a second machine, exploratory): over the 55 instances the median number of branches is 76,539 for CP-UC against 954,098 for CP-LB, and on the 29 instances that both solve there within 60 s CP-UC needs a

Table S15: Distance to proven optima on the 62 instances ($n_1 = 20$) that CP-SAT solved within 300 s, in %, (UB − OPT)/OPT. Intervals are 95% bootstrap intervals over instances. The solved instances are the easier part of $n_1 = 20$, so the values describe that subset only.

Method	Mean [95% CI]	Median	Max	At optimum (%)
HgR (published; unmatched reference)	3.38 [2.93, 3.84]	3.39	10.64	1.6
EACDC (one default run)	1.71 [1.35, 2.05]	1.30	5.19	6.5
ILS, 4 s	0.43 [0.31, 0.56]	0.25	2.07	40.3
ILS, 10 s	0.41 [0.29, 0.54]	0.25	2.07	41.9
LB* (lower bound; (OPT − LB)/OPT)	5.74	5.14	15.38	4.8
Chen–Song bound (same measure)	9.57			

median of 0.27 times the branches and 0.08 times the conflicts of CP-LB.

The preregistered confirmation on new instances. The frozen method was run on the exact confirmation set (300 new instances, generator seed 20261193; Section 7) with four arms at 300 s of CPU each: CP-UC, CP-LB, CP-plain and the time-indexed model of Cota et al. (2016) solved as a MIP with HiGHS 1.15.1 (one thread, the ILS schedule as start solution, $n_1 = 20$ only). All arms of an instance ran in one process on one machine. The run was moved to a second machine part-way, as a recorded protocol deviation, so 171 instances ran on an x86 laptop under Linux and 129 on an Apple-silicon computer; no instance was split between them. On the x86 laptop, CP-UC solved 63 instances against 17 for CP-LB (discordant pairs 47:1, exact McNemar $p = 3 \times 10^{-13}$), 16 for CP-plain (49:2) and, at $n_1 = 20$, 47 of 84 against 3 for the MIP (44:0). On the Apple-silicon computer it solved 74 against 30 (45:1, $p = 1 \times 10^{-12}$), 29 (45:0) and 53 of 66 against 9 (45:1). The final gap of CP-UC was smaller than that of CP-LB by 2.63 [2.27, 3.00] points on the first machine and by 3.29 [2.77, 3.85] points on the second. The effect therefore holds on each machine; only the solved share differs (37% and 57%). Over all 300 instances CP-UC solved 100 of 150 instances with $n_1 = 20$ and 37 of 150 with $n_1 = 30$; CP-LB and CP-plain solved none with $n_1 = 30$. On the 45 instances solved by both CP-UC and CP-LB, the median times to proof were 7.1 and 35.1 s. Of the 150 MIP runs, 12 ended optimal, 72 were stopped by the CPU limit and 66 by the HiGHS wall limit, which we set to the remaining CPU budget because HiGHS presolve cannot be interrupted; the longest used 318 s. The planned ablation of the cut families was not run, because the run reached its CPU cap first.

Distance to the optimum on the first 62 optima. CP-SAT proved optimality for 62 of the 300 instances of the exact set within 300 s with the variants without cuts: 62 of the 150 instances with $n_1 = 20$ (41%) and none of the 150 with $n_1 = 30$. Table S15 gives the results. LB* equals the optimum on only 3 of the 62 instances. On average it lies 5.74% below the optimum (median 5.14%, maximum 15.38%), against 9.57% for the Chen–Song bound. On the 59 instances solved by CP-LB, the mean distance by door group ranges from 2.18% in U(2,10) to 8.74% with ten doors per side. The solved instances are the easier part of $n_1 = 20$, so these numbers describe that subset.

These results let us split the certified gap into its two sources. On the solved instances, the gap of ILS after 10 s against LB* is 6.68% on average. Of this, 6.24 points come from the bound (the distance from LB* to the optimum) and 0.44 points from the heuristic. At $n_1 = 20$, therefore, almost all of the remaining gap is in the bound. On the 238 unsolved instances, the gap between the best upper bound and the best proven lower bound is 7.53% on average. None of these 238 instances was certified optimal by LB* alone.

The fluid relaxation. On the enlarged set of 165 proven optima, LB* equals the optimum on 3 of 118 instances at $n_1 = 20$ and on none of 47 at $n_1 = 30$. It lies 6.62% and 3.63% below

the optimum on average. The slack depends on the door counts. To see why, we computed an exact fully fluid relaxation at $n_1 = 20$, in which each stage is one machine of speed m and only the order of the inbound trucks matters (a dynamic program over subsets). With two doors per side and in U(2,4) this relaxation is above LB* on 97% of the instances and reduces the slack from 4.49% to 2.71% and from 7.35% to 5.63%: there the bound loses because it chooses an unrelated set of outbound trucks at each time, whereas a schedule needs a nested sequence of sets. With ten doors per side the fluid relaxation is 24% below the optimum and never above LB*: there the slack comes from the few trucks per door, which a fluid argument cannot see. The maximum of the two bounds is 5.52% below the optimum on average, against 6.62% for LB*.

S10. Exploratory analyses

All analyses in this section are exploratory.

S10.1. Structured and larger instances in detail

The preregistered test on 240 sparse, block-structured, unbalanced and large instances (Section 7) confirmed both lower-bound endpoints. LB* is 3.45% [2.79, 4.15] above the best non-own bound (E2; higher on 129 instances, equal on 111, never lower) and 3.16% [2.50, 3.85] above the energetic bound (E4; higher on 125, equal on 41, lower on 74), with one-sided $p = 10^{-5}$ for both, which stays below 0.05 after Holm over the four endpoints. No lower bound exceeded an upper bound. The gain is much smaller than on the benchmark and depends on the structure. With random precedence of density 0.2–0.3 it reaches 5.9–12.8% at $n_1 = 60$–100, and 8–10% on the large sparse instances ($n_1 = 150$–300). With density 0.1 it is below 1%, and with block structure it is 0–3.7%: when the predecessor sets are small or nearly disjoint, the union argument adds little to the head sums, and energetic reasoning is sometimes higher. With strongly unbalanced doors it grows with n_1, to 3–10% at $n_1 = 100$. The bound took 0.5 s per direction on average at $n_1 = 30$, 20 s at 60, 59 s at 100 and 125 s at $n_1 = 300$ with the deterministic caps (the walk finished on 27% of the instances at $n_1 = 100$ and on none at 200–300); energetic reasoning took at most 6 s. On the 58 instances with $n_1 \leq 150$ of the matched subset (seeds 11–13, one machine), the base ILS was below EACDC by 0.85% [0.63, 1.09] at equal CPU (E1; lower on 49 instances, equal on 4, higher on 5) and by 0.75% [0.53, 1.00] at equal evaluations (E3; 47/4/7), both confirmed after Holm over the four endpoints ($p = 10^{-5}$). No EACDC run exceeded the 600 s cap. The margin is largest on random precedence at $n_1 = 100$ (about -2%) and smallest with strongly unbalanced doors, where one cell ($m_1 = 2$, $m_2 = 10$, $n_1 = 30$, two instances) favours EACDC by 0.5–0.7%. With the same method, rival and protocol, the margin is larger than on the benchmark (-0.57% and -0.48% for the base ILS on the first confirmation set), so the upper-bound part of the method transfers to structured instances. The next subsection explains where the margin comes from. The ILS upper bound after 10 s is 9–23% above LB* at $n_1 = 150$–300, so at that size 10 s is too short for the heuristic, the bound, or both.

S10.2. When does search pay off? A bound-based explanation

The margins over EACDC vary a lot between instances: from zero to more than 3% on the structured set, and from +0.2% to -1.9% between cells of the benchmark. We now ask what explains this variation. The analysis is exploratory: it re-uses the preregistered matched runs of the base ILS against EACDC on the 210 instances of the first confirmation set and on the 58 structured instances (same protocol, seeds 11–13), adds exact optima for the structured instances with $n_1 = 30$, and adds two small diagnostic runs. We wrote the hypotheses down before computing any of the per-instance quantities. For FB-ILS, which was not part of the structured preregistration, we read the search at the evaluation count of the same EACDC runs.

This matching does not depend on the machine, so the EACDC runs need not be repeated. FB-ILS is then -0.93% [-1.20, -0.69] from EACDC (W/T/L 48/4/6), against -0.75% for the base ILS on the same runs and -0.63% for FB-ILS on the benchmark.

Headroom decides the margin. A search can only win where the rival leaves room. We measure this room by the certified headroom $\rho = (\mathrm{UB}_{\mathrm{EACDC}} - \mathrm{LB}^*)/\mathrm{LB}^*$. Table S16 shows that the margin grows steadily with ρ on both sets: on the structured set from -0.23% for $\rho < 3\%$ to -2.06% for $\rho \geq 10\%$. The rank correlation between margin and ρ is -0.79 on the structured set and -0.45 on the benchmark. Each point of headroom is worth about 0.07 (benchmark) to 0.09 (structured) points of margin in a linear fit. The exact optima make this sharper. CP-UC (Section 6) solved 13 of the 18 structured instances with $n_1 = 30$ within 300 s. On these, EACDC is 0.84% above the optimum on average and equal to it on four, and the base ILS closes 60% of this distance; the rank correlation between the margin and EACDC's true distance to the optimum is -0.80. Where EACDC is already optimal, no method can win.

We compared two ways of predicting the margin by leave-one-out R^2 of a linear model. Raw instance features (precedence density, m_1, m_2, $\log(m_1/m_2)$, n_1 and the dispersion of the processing times) explain 0.04 of the variance on the structured set and 0.12 on the benchmark. Quantities from the bound theory explain 0.34 and 0.16: the headroom ρ, the gain of the union term over the head-sum bounds, the binding term, the residual $(a+b+(1-1/m_1)\max p_1+(1-1/m_2)\max p_2)/\mathrm{LB}^* - 1$ of Corollary 2 and the load ratios a/LB^* and b/LB^*. Given the theory block, the raw features add nothing on the structured set and at most 0.04 on the benchmark. Most of the explained variance comes from ρ alone (0.32 and 0.16). The bound is therefore useful in a second way: from one run of a heuristic and one bound evaluation it tells where further search can pay. The same holds, exploratorily, on 88 instances with chain-like, tree and series-parallel precedence (densities 0.025–0.10, $n_2/n_1 \in \{0.25, 0.5, 2, 4\}$, seeds 71–73): the base ILS is -0.33% [-0.45, -0.22] from EACDC, the rank correlation of margin and ρ is -0.80, the theory block reaches a leave-one-out R^2 of 0.73 against 0.25 for raw features including the structure type, and the structure adds nothing once ρ is known. Where $n_2 > n_1$, $\rho \leq 0.4\%$ and the margins vanish.

The slack of the bound explains the difference between the sets. At equal ρ, the margin is larger on the structured set by 0.37 points [0.17, 0.57] (pooled linear fit with a set indicator). The reason is the slack of the bound. On the solved structured instances LB^* is only 1.71% below the optimum, against 3.63% on the benchmark at $n_1 = 30$ (Section 8.2). On half of the structured instances (28 of 58, against 21 of 210 on the benchmark) the head-sum term L_{CS} decides the bound (Corollary 1), mostly because sparse or block precedence leaves the union term with nothing to add. There the bound is close to the optimum, and ρ is mostly room that a search can use. On the benchmark the union term decides the bound, and with few doors the nestedness slack of Section 8.2 inflates ρ without giving the search anything to find. Which term binds does not predict the margin by itself once ρ is known (partial rank correlation of the union gain with the margin $+0.21$ on the benchmark and 0.00 on the structured set). Our prior expectation that search pays more where the union term binds is therefore not supported; what matters is how much of ρ is real.

The cell that favours EACDC. The two methods use the same list decoder, and neither encodes a door assignment: every truck goes to the door that becomes free first. What differs is the search space. EACDC searches orders of the outbound trucks and turns each into an inbound order by the first outbound truck that needs each inbound truck. Our ILS searches inbound orders directly, on the side of the time reversal whose best constructive order is better. We re-ran EACDC on all 18 structured instances with $n_1 = 30$ (54 runs) and kept its best solution. Our decoder gives the same makespan for the inbound order derived from it in 54 of 54 runs. In the cell $m_1 = 2$, $m_2 = 10$, the base ILS started from this order matches or improves EACDC

Table S16: Margin of the base ILS over EACDC (mean of $(\mathrm{UB_{ILS}} - \mathrm{UB_{EACDC}})/\mathrm{UB_{EACDC}}$, %, 95% bootstrap interval) by the certified headroom $\rho = (\mathrm{UB_{EACDC}} - \mathrm{LB}^*)/\mathrm{LB}^*$ of the EACDC run. Benchmark: the 210 instances of the first confirmation set; structured: the 58 matched instances. Same protocol and seeds (11–13) in both sets. Exploratory analysis of preregistered runs. FB-ILS on the structured set is read at the evaluation count of the same EACDC runs.

	Benchmark			Structured				
ρ (%)	n	equal CPU	equal evals	n	mean ρ	equal CPU	equal evals	FB-ILS, evals
$[0, 3)$	25	−0.13 [−0.31, +0.04]	−0.04	24	1.67	−0.23 [−0.37, −0.09]	−0.17	−0.29
$[3, 6)$	94	−0.40 [−0.53, −0.27]	−0.30	16	4.07	−0.84 [−1.10, −0.57]	−0.76	−0.81
$[6, 10)$	38	−0.62 [−0.95, −0.32]	−0.55	7	7.95	−1.11 [−1.48, −0.77]	−0.85	−1.32
≥ 10	53	−1.03 [−1.19, −0.87]	−0.95	11	17.37	−2.06 [−2.66, −1.50]	−1.95	−2.26
All	210	−0.57 [−0.67, −0.46]	−0.48	58	6.07	−0.85 [−1.09, −0.63]	−0.75	−0.93

(−0.10%), and the same ILS run over EACDC's outbound encoding does too (−0.04%). Neither the moves nor the encoding is the cause. The cause is the choice of side. In this cell the reversed instance has the better constructive order (931 against 944 and 872 against 876), but the base ILS ends worse there (913–915 against 908, and 864–866 against 858). EACDC always searches the forward side. The base ILS on the forward side is never worse than EACDC in this cell. Over all 54 runs, the side rule costs 0.12% on average and picks the worse side in 13 runs. Its cost is concentrated in this cell (0.72%) and in the mirror cell $m_1 = 10$, $m_2 = 2$ (0.25%). The true headroom in the cell is also small: EACDC is 0.33% and 0.55% above the optimum on its two instances. Searching both sides with half of the budget each, or switching sides at restarts, is the obvious remedy; we leave it untested.

Cost per evaluation and precedence density. Proposition S8 suggests that our decoder gains most on sparse precedence, where $|E| \ll n_1 n_2$. This is true only up to constants on all our instance sets. Because $0.8n_1 \leq n_2 \leq 1.2n_1$, the term n_2^2 of both decoders is $\Theta(n_1 n_2)$, so against the faster EACDC evaluator our decode is cheaper by a constant factor whatever the density. It would be strictly cheaper asymptotically only if $n_2 = o(n_1)$ and $|E| = o(n_1 n_2)$. Against the original EACDC code the situation is reversed: its sort of the $|E|$ predecessor pairs makes our decode cheaper by a factor $\Theta(\log n_1)$ when the density is fixed, as on the benchmark, and only by a constant when $|E| = O(n_1 n_2 / \log n_1)$. The measurements agree. On the structured runs (one machine), EACDC used 1.6–1.8 times more CPU per counted evaluation than the base ILS (with its Python decoder) at every density from 0.1 to 0.5 (rank correlation with density −0.07), and the difference between the CPU-matched and the evaluation-matched margins does not change with density either. On the exploratory family above, the operation-count ratio of the fast evaluator to our decode is 3.1–4.9 for n_2/n_1 from 4 to 0.25, flat in the density (15.4 at $n_2/n_1 = 4$ on the reversed side); our compiled decoder is 31–105 times faster per decode, a Python port only 1.4–2.9.

S10.3. Practical use of the bound

A plan can come with a certificate. LB^* takes about 0.01 s at $n_1 = 20$ and a few seconds on average at $n_1 = 60$–80 (Section S4). Dividing the makespan of any plan by it gives an upper bound on how far the plan can be from the best possible one. On the development set, the best plans we found were certified within 5% on 47.6% of the instances. For the published HgR plans the share is 14.1% over the whole benchmark. With the earlier bounds, the same plans would have appeared to be about 26% from optimal on average. A planner who sees a certified gap of a few percent knows that little can be gained by searching further. A planner who sees a larger gap knows that part of it may be due to the bound (next point).

At small terminals, better heuristics bring little. On the instances with 20 inbound trucks where the optimum is known, ILS is 0.41% above it on average, and 6.24 of the 6.68 points of the

certified gap come from the bound. For terminals of this size, more effort spent on the heuristic has little to gain. The value of more computation lies in proving that the plan is good, for example with the union cuts of Section 6, not in improving the plan.

Where doors are many, the certificate is weaker. The certified gap of the HgR plans is 5.8% with two doors per side and 15.8% with ten (CPU-capped census, Table S1; 5.2% and 15.1% with UPB-det, Table 2). With ten doors and few trucks, LB^* is 8.74% below the optimum on the solved instances, and energetic reasoning gives a better bound on most instances (Table S4). Our interpretation is that with many doors the limiting factor is the length of individual trucks and their timing, which the union argument captures less well than the shared workload. Planners of terminals with many doors should therefore take the maximum of both bounds. The energetic bound adds 0.205 s of CPU time per instance on average.

The bound points to the bottleneck. The bound is attained at some time t^*. At that time, the knapsack selects a set A^* of outbound trucks that can have started, and these need the inbound trucks $N(A^*)$. Together, t^*, A^* and $N(A^*)$ describe where the schedule is congested. We traced the bound on 350 benchmark instances, ten per door group and n_1. The union term decides the bound on 341 of them; on the other nine, all with ten doors on a side and $n_1 \leq 40$, the longest truck after its head decides it. At the critical time the outbound work is limited by what the inbound doors can feed, never by the outbound doors, on all 350 instances, and t^* lies at about half of the bound (t^*/LB^* between 0.47 and 0.52 on average in every cell). Neither side is the bottleneck by itself: the outbound doors wait for the inbound work during the first half of the shift and are saturated afterwards.

We then recomputed LB^* and a 2 s ILS schedule with one more door on the inbound side and, separately, on the outbound side. One more door shortens the makespan by about 15–17% with two doors per side, 9–10% with four, 10–14% in U(2,4), 8–10% in U(2,10) and 3–5% with ten. On the 101 instances with unequal door numbers, the extra door on the side with fewer doors gives the larger reduction of the schedule's makespan on all 101, and the larger reduction of the bound on 100. On the 335 instances where the two options differ, the choice made from the bound alone, without any schedule, agrees with the choice made from the schedules on 93%. The bound can therefore rank door investments in milliseconds to seconds.

A schedule is available immediately. ILS has a feasible plan after 0.05 s in every run, while EACDC needs to build its initial population first (Section S7). For re-planning during operations, when trucks arrive late, this difference may matter more than the final quality.

S11. Reproducibility notes

This section records the conventions and settings that a reader needs to compare our numbers with earlier ones. The earlier papers on the benchmark made reasonable choices that differ from ours. We list them to make comparisons possible.

Gap conventions. Table S17 lists the gap measures. When numbers from different rows are compared, the bound and the denominator must be converted first.

Published HgR values. The per-instance HgR upper and lower bounds in the public repository (Fonseca, 2019) are the values we use. Our Python port of the authors' lower-bound script reproduces the reported lower bound on all 10,500 instances. The recorded solution times have a median of 54 to 149 s per door group.

Table S17: Gap measures used on the benchmark.

Source	Lower bound	Gap
Fonseca et al. (2019) (HgR, multi-door)	Chen–Song-type bound, rounded up	(UB − LB)/LB
Nogueira et al. (2020) (PCH)	LB_{PCH}	(UB − LB)/UB
Nogueira et al. (2025) (EACDC, EACDCml)	LB_{PCH}	(UB − LB)/UB
This paper	UPB or LB*, as stated	(UB − LB)/LB, unless stated otherwise

EACDC. We ran the public `GA.py` with the static configuration of Table 2 of Nogueira et al. (2025). On 70 instances, the gap of this configuration against LB_{PCH}, in the convention of that paper, was 23.0% [21.4, 24.7], against the published 23.4%. The public code encodes an order of the outbound trucks and derives the inbound order from it; it runs all 150 generations; and it fixes the random seed. We kept the encoding and the number of generations, and we passed the run's seed.

Settings of our methods. Bounds: walk budget 4 s of CPU per direction for the census, 1 s per knapsack solve; deterministic version 50 steps and 0.5 deterministic seconds per solve. ILS: stall limit $40n_1$, perturbation by a segment reversal and two random insertions, random seed $100 + s$ for run seed s. Exact method: 300 s per instance, at most 15% of it for probes of at most 5 s, 12 cut times, one CP-SAT worker.

Code and data. The code, the per-instance bounds (UPB-det, LB*), the best schedules, the SHA-256 digests of the preregistration documents and the scripts that produce every table are deposited at Zenodo, DOI 10.5281/zenodo.23933868. The protocol documents themselves are available from the author on request and can be checked against the deposited digests.

Instance sets in detail.

- *Benchmark census.* All 10,500 instances of Cota et al. (2016), with the per-instance HgR upper bounds and lower bounds published by Fonseca et al. (2019) in their public repository (Fonseca, 2019). Every instance is scored, so the census results are distributions over the population and carry no sampling error.
- *Development set.* 105 benchmark instances, three per cell, drawn by stratified sampling. The local search was designed on this set, so results on it are exploratory.
- *Confirmation set.* 210 new instances, six per cell, generated with the benchmark's rules (Section 3) and generator seed 20261103. We checked the generator against the 10,500 files. Two examples: the standardised density of the precedence relation has a standard deviation between 0.98 and 1.03 across files, as independent pairs would give. In the U(2,4) group, $m_1 = m_2$ holds in only 833 of 2,100 files, so the two door numbers are drawn separately. The protocol, the frozen code and the endpoints were written down before these instances were generated.
- *Exact set.* 300 benchmark instances, 30 per cell for $n_1 \in \{20, 30\}$ and the five door groups.
- *Exact confirmation set.* 300 new instances with the same design (30 per cell for $n_1 \in \{20, 30\}$ and the five door groups), generated with the benchmark's rules and generator seed 20261193 after the exact method and the protocol were frozen; the frozen code is identified by hashes in the protocol. CP-SAT used seed 0 and one worker in every arm.

- *Single-door set.* The 500 instances of Fonseca et al. (2019) for the problem with one door per side, taken from the authors' files. Recomputing the two bounds of Chen and Lee (2009) reproduces the row means of Table 4 of Fonseca et al. (2019) in all ten rows, which confirms that the files are the published instances.

- *Structured set.* 240 new instances with sparse precedence, block precedence (destination groups), strongly unbalanced door numbers, and n_1 up to 300, generated under a second preregistration (generator seed 20261104). The endpoints are E2 and E4 (gain of LB* over the best non-own bound and over energetic reasoning, all 240 instances) and E1 and E3 (ILS against EACDC at equal CPU and at equal evaluations, 58 instances, seeds 11–13), with Holm's correction over the four.

- *Second upper-bound confirmation set.* 210 new instances (generator seed 20261105, same rules and cells as the first confirmation set) for the preregistered test of FB-ILS, seeds 21–23.

- *HgR confirmation set.* 210 new instances (generator seed 20261107, same rules and cells), with a subset of 60 drawn once in advance for a 60 s endpoint. None duplicates a file of the benchmark or of the earlier sets.

Computing environment. All runs used one thread per process at low priority, on Apple-silicon Mac computers and, for some runs, an x86 laptop under Linux (WSL2), with CPython 3.12.3, 3.12.9 and 3.12.14 and the same library versions on every machine: NumPy 2.5.4, SciPy 1.18.1, OR-Tools 9.15.6755, HiGHS (highspy) 1.15.1, DEAP 1.4.4, PyTorch 2.15.0 and Optuna 5.0.0 (the HgR rebuild used CPython 3.14). The pinned versions are part of the data release. CPU time is process time. We observed that the CPU time of identical deterministic EACDC runs differed by a median factor of 1.40 between two run windows with different machine load. For this reason we never compare CPU times measured in different runs or windows.

Matching the effort of ILS and EACDC. A fixed CPU budget is a poor basis for comparing ILS with EACDC. EACDC builds its whole initial population before its first evaluation, and this step alone can exceed several seconds at $n_1 \geq 70$. A short budget would stop EACDC before it has any solution, and a medium one would stop it in the middle of its run. We therefore match the effort run by run.

1. For each instance and seed, one process runs EACDC to completion, without truncation or restarts, and then runs ILS (or the other way round). The order alternates from task to task.

2. ILS is read at exactly the CPU time used by that EACDC run, with its constructive phase included.

3. ILS is also read at the number of schedules that EACDC evaluated in its generations. This count excludes the evaluation of the initial population, which makes it conservative for ILS. A second count includes it.

The two matchings answer different questions. The evaluation-matched comparison does not depend on how fast each implementation evaluates a schedule, so it compares the algorithms. The CPU-matched comparison measures practical performance of the two implementations as they are. Our ILS evaluates schedules in plain Python, while the EACDC evaluator processes populations, so in equal CPU time ILS makes about half as many evaluations as EACDC.

Preregistered confirmation. The confirmation run had three endpoints, fixed in advance.

- E1, primary: the paired difference between ILS and EACDC in makespan, CPU-matched.

- E3, secondary: the same, evaluation-matched (generation calls only).

- E2, primary: the gain of LB* over $\lceil \max(L_{\mathrm{CS}}, \mathrm{LB}_{\mathrm{PCH}}) \rceil$, the best bound on these instances that does not come from this paper.

ILS kept its constants and EACDC its static configuration. The seeds were 11, 12 and 13, which no earlier run had used. An endpoint counts as confirmed if its Holm-adjusted p-value over the three endpoints is below 0.05 and its 95% interval excludes zero in the expected direction. For E2 we also required that no lower bound exceeds the best upper bound found. A second preregistration of the same form covers FB-ILS (Section 5). It adds two endpoints that compare FB-ILS with the base ILS, which form the ablation of the forward–backward mechanism, and it names the evaluation-matched endpoints as the ones that do not depend on the implementation.

A third preregistration compares FB-ILS and the base ILS with our re-implementation of HgR at equal CPU time on the HgR confirmation set (seeds 41–43, all methods of a task in one process, rotated order). The endpoints are FB-ILS against HgR at 10 s (H1, primary), the base ILS against HgR at 10 s (H2, both in Python), FB-ILS against the better of HgR with and without its linear program at 10 s (H3), and FB-ILS against HgR at 60 s on the subset (H4), with one-sided Wilcoxon signed-rank tests, Holm's correction and the criterion above. One amendment, before any method ran, fixed a logging line that stopped the runner at start-up.

The protocol documents use the labels "ILS v1" for the base ILS, "ILS v2" for FB-ILS and "LB^*_U" for UPB.

S12. Limitations in full

We list the limitations that a reader should weigh when using these results.

1. *One instance family.* All multi-door results come from random instances in which each inbound–outbound pair is a predecessor pair with probability one half. The single-door set (one door per side) is a second family, but there the comparison with the Lagrangian bound uses row means only. On the preregistered sparse, block-structured, unbalanced and larger instances the gain of the bound is much smaller (3.4% on average, near zero with very sparse or block precedence), while the base ILS improves on EACDC there by more than on the benchmark (0.85% at equal CPU, against 0.57%). The explanation of the margins by the headroom and the slack of the bound (Section S10) is an exploratory re-analysis of 268 instances, with exact optima for only 13 structured ones, and the remedy for the side rule that loses one unbalanced cell is untested. Chain, tree and series-parallel precedence were tested only on an exploratory development family.

2. *The bound is not tight.* It is 6.62% ($n_1 = 20$) and 3.63% ($n_1 = 30$) below the optimum on average where the optimum is known, and equal to it on 3 of 165 instances. The known optima come from the easier part of the instances with $n_1 \leq 30$; none is known for $n_1 \geq 40$.

3. *Window-inference bound.* UPB-W was evaluated on 98 proven optima with $n_1 \leq 30$ and on the 105 development instances, not on the whole benchmark. At $n_1 = 70$–80 its gain is bound by the budget: none at 120 s, 0.98% and 0.39% at 600 s, still rising at $n_1 = 80$ (an exploratory run on 15 instances per size). An oracle experiment shows that the inference of windows, not the tests, is what limits it. Its strength at large n_1 comes from CP-SAT tests inside the loop, so it is not polynomial, and its value depends on the budget and the machine. Its CPU use can exceed the nominal budget, in our runs by about a factor of two at most (244 s against 120 s), because the budget is checked only between trial tests.

4. *Deterministic bound with a step cap.* At $n_1 \geq 60$ the walk that computes the bound often stops at its cap, so the bound is not exact there. We report the deterministic version, whose value depends only on the instance and the solver version. The LP version of Proposition S5 is exact and strongly polynomial, but it is no stronger than the classical bounds on our instances.

5. *Small upper-bound gain, rebuilt comparator.* The confirmed gain over EACDC and EACDCml is about half a percent of the makespan for the base ILS and about one percent at equal CPU (0.63% at equal evaluations) for FB-ILS. EACDCml is compared through our rebuild, because its trained network is not public; the rebuild reproduces the published mean gap within the instance-sampling error (Section S7.4). ILS was designed on the development instances. Each confirmation used 210 instances from one generator seed.

6. *Implementation effects.* The CPU-matched comparison mixes algorithm and implementation: our first decoder is plain Python, FB-ILS is compiled, and the EACDC evaluator processes populations. The evaluation-matched comparison removes this effect but counts a decode of either method as one unit. Proposition S8 shows that our decode is never asymptotically more expensive than theirs, and every decode of ILS, including the forward–backward passes, is counted. Some work is not counted on either side: the 14 constructive orders of ILS cost about 50 EACDC evaluations, and the variation operators of EACDC take 21–35% of each generation. The faster EACDC evaluator was shown to give identical results on 18 runs and 1,000 permutations, not proved identical.

7. *Re-implemented reference.* We compare with HgR through our re-implementation of the authors' public multi-door script. The model file that the script loads is not public, so we rebuilt its linear program from the paper, choosing the reading most favourable to HgR where the description leaves room. Our version reproduces the published HgR upper bounds on the 105 development instances. The preregistered advantage of 2.5–3.0% is therefore confirmed against this re-implementation at equal CPU time on one machine, not against the authors' AMPL and CPLEX runs, whose run times are not comparable with ours. Only the base ILS (H2) is matched with it in language; FB-ILS is compiled, while our HgR is Python with its linear program solved by HiGHS.

8. *Theory.* The constant of Corollary 2 is classical against the optimum, and the strong NP-hardness of the knapsack is due to Johnson and Niemi (1983). Its tightness on two-level instances is open; the worst ratios we found are 2.185 for $M = 2$ and 2.333 for $(m_1, m_2) = (2, 3)$. The padding lemma covers the forward bound and the computed form B, not the reversed maximum, and it does not apply to lengths bounded in $[10, 100]$.

9. *Exact timings.* The main runs of CP-UC and of the variants without cuts were timed on different machines. Solved counts under a CPU limit depend on the machine. The same-machine re-timing confirms the speed-up, but it covers only the 55 instances that both variants had solved, with a 60 s limit, and the PassMark factor that we use for the original times is a single-thread rating, not a measurement of CP-SAT on these models. The branch counts that support the speed-up come from one exploratory re-run.

10. *Exact result.* The union cuts, their number ($k = 12$) and the probe schedule were developed on the 300 instances of the exact set. A preregistered run of the frozen method on 300 new instances confirmed the gain over the root-bound and plain models and over a time-indexed MIP. Two machines were used for that run because one became unavailable; the comparisons are paired within instance and hold on each machine. We did not ablate the cut families, so we cannot say which of them, forward or reversed, matters more.

11. *Floating-point refutations.* The LP refutations of UPB-W are certified in exact arithmetic for every reported value; CP-SAT refutations are trusted as exact. The search uses CPU time limits, so the bound can differ by a few units between machines (Supplement S5.2).

12. *Verifiability of the preregistrations.* The protocols are internal documents; their SHA-256 digests are deposited with a public timestamp in the data release (DOI 10.5281/zenodo.23933868), after the runs, and the documents themselves are available on request, not deposited.