LEARNING TO COMPILE RELAXED DECISION DIAGRAMS BY OPTIMIZING THE BOUND

Anonymous authors
Paper under double-blind review

ABSTRACT

Relaxed decision diagrams certify lower bounds for discrete optimization by merging states under a per-layer width cap. Bound quality depends on which states remain separate, a decision made by hand-designed ranking rules. We study learning this ranking for the traveling salesman problem with a drone. The natural approach ranks states by accumulated cost plus a learned cost-to-go fitted to backward values from a wider relaxed diagram. We show these targets are systematically biased: a relaxed node inherits the most optimistic completion of every state it represents, so its value reflects how much merging occurred rather than the quality of the state, and more data does not remove the bias. Exact targets do not close the gap either: the same policy trained directly on the bound beats exact-target imitation by 1.88 points at the size where exact targets are computable and by 3.27 at a larger one, on 88 and 99 of 100 held-out instances. We therefore discard targets and optimize a 22-parameter ranking policy against the bound compilation produces, which needs no solved instances and preserves validity for every policy. Averaged over optimizer seeds, it improves over imitation by 4.4–5.8 gap points and over the hand-designed $f{+}h$ rule it corrects by 6.1–8.8, on at least 99 of 100 test instances in every configuration from 15 to 25 customers, and by 11.8–15.7 over root distance; it reaches in one compilation what a model-free self-guided baseline needs about three times the arc cost to match. Frozen and applied to an independently defined distribution, it is not overtaken by root distance at five times its compilation budget in three of four configurations.

1 INTRODUCTION

Relaxed decision diagrams (DDs) make intractable dynamic programs tractable by capping the number of nodes per layer and merging the excess. A merged node takes the most permissive value of each state component over its members, so the shortest path remains a valid lower bound; how tight it is depends on which states remain separate. That choice is made by a hand-designed rule, and the standard one, keeping the states of smallest accumulated cost, ignores the cost still to be paid. Ranking by accumulated cost plus an estimated cost-to-go corrects this and yields substantially tighter bounds. This paper is about where that estimate should come from.

Imitating a cost-to-go introduces a systematic bias. The obvious source is a wider diagram: compile it once, run the backward pass, fit a model to the node values. But a relaxed node represents many exact states, and its backward value is the most optimistic completion available through the relaxation. A target can be low because of merging rather than because the state has a favorable continuation, and a model fitted to such targets reproduces the bias. A seventeen-fold larger training set improves the resulting baseline by about three quarters of a percentage point.

Two consequences are testable and both hold. The problem persists with exact values aggregated by minimum: fitted at $m = 8$, the largest size at which the exact cost-to-go is computable, such targets land within a third of a point of the coarse cost-to-go they replace, at $m = 8$ and at $m = 15$. And it persists without any model: querying a wider diagram's per-state values during compilation, with no charge for building it, degrades the bound by 9.24 points relative to that diagram's own table, on all 100 instances (Section 4).

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Training directly on the bound. We keep the policy class and remove the targets: the same 22-parameter linear policy is optimized against the gap compilation produces, by a black-box search costing one compilation per evaluation. Training needs no solved instances, so it is not confined to sizes an exact method reaches, and the frozen policy applies unchanged to other sizes and families.

Contributions. (i) A mechanism by which imitating cost-to-go values from relaxed diagrams produces poor rankings, separating relaxation, minimum aggregation, and multiplier mismatch (Section 4). (ii) A pre-registered control showing the gap is not explained by relaxation alone: direct optimization beats imitation of *exact* cost-to-go by 1.88 points at the training size and 3.27 at a larger one, on 88 and 99 of 100 instances (Section 6.5). (iii) A ranking policy trained against the compiled bound without targets or solved instances, beating imitation by 4.4–5.8 gap points, the round-0 $f+h$ rule by 6.1–8.8 as seed means, on at least 99 of 100 test instances in every configuration, at sizes up to 25 customers from training at 15, and root distance by 11.8–15.7 in the three cells where root distance was measured (Sections 5 and 6.4). (iv) Evidence that the signal, not capacity, drives the result: a perceptron with seventeen times the parameters and a per-state policy both land within 0.35 points of the linear policy (Section 6.5). (v) A protocol with every comparison at matched arc cost, disjoint development, training, and test seeds, and an audit of missing and duplicate cells, released with the code (Section 6.1, Appendix H).

The baseline we hold ourselves to is model-free, reusing one compilation's backward values to guide the next; at comparable bound quality it needs about three times the arc cost. The mechanism is not specific to this problem: wherever a supervised target is a by-product of a relaxation, it inherits that relaxation's compromises, and a target defined as a minimum over a merged group carries a bias set by how much merging occurred rather than by the state being scored. TSP-D is where we can measure that exactly, against exact targets and a per-state oracle, and the 22-parameter policy is the control that separates signal from capacity, not a ceiling on what the approach can express.

2 Background and related work

2.1 The traveling salesman problem with a drone

In the TSP-D (Murray & Chu, 2015; Agatz et al., 2018), a truck carrying a drone must serve all customers; the drone may launch, serve one customer, and rejoin the truck later, the first to arrive waits, and the objective is the makespan. Exact methods reach roughly 39 customers (Roberti & Ruthmair, 2021); beyond that, a lower bound is what certifies a heuristic's quality.

A state is $(\mathrm{served}, i_c, i_t, \tau)$: customers visited, the node where truck and drone were last together, the truck's position, and the elapsed desynchronization. Merging intersects served, takes the maximum τ, and unions the rendezvous points, each in the optimistic direction, so the shortest path remains a valid bound. We allow the truck to revisit a served customer solely to collect the drone, which 10 of the 39 known optima require; a merged state must then track valid rendezvous points, and Appendix A describes two defects this caused and the tests that now prevent them.

2.2 Relaxed decision diagrams, and what a width cap decides

A decision diagram represents the solution space as a layered DAG whose root-to-terminal paths are the feasible solutions and whose arcs carry decision costs, so the cheapest path is the optimum; the exact diagram is intractable. Relaxed compilation caps each layer at width W and combines excess states by a merge operator taking the most permissive value of each component, so the merged node represents a superset of its members' behavior and the shortest path can only decrease. We follow Bergman et al. (2016a;b); Tang & van Hoeve (2024) derive the DD route relaxations for the TSP-D. At fixed width, two decisions govern bound quality: which states are eligible to merge, which in our relaxation is fixed by the (i_c, i_t) bucket, and which states within an over-full bucket remain separate. This paper concerns the second. The standard rule keeps the states of smallest accumulated cost, *root distance*. Merge quality has been studied and learned (Frohner & Raidl, 2019a;b) and states clustered before merging (Nafar & Römer, 2024), but the ranking that decides which states survive the cap has remained hand-designed. It affects tightness but never validity, since any partition of an over-full layer is a relaxation, and it is made once per over-full layer under a fixed budget, so it must be cheap relative to the compile it guides.

2.3 LEARNING INSIDE EXACT SOLVERS

Learning has selected variable orderings for DDs by reinforcement learning (Cappart et al., 2019; 2022) and node selection for restricted DDs from exact Pareto frontiers (Patel et al., 2026); none learns a cost-to-go that steers merging under a fixed budget. Learned heuristics for A* must preserve admissibility (Futuhi & Sturtevant, 2026), whereas here h only decides which states remain separate and can never invalidate the bound. Training against a bound rather than labels is established for Lagrangian multipliers (Demelas et al., 2024; Parjadis et al., 2023), and differentiable surrogates exist for combinatorial solvers (Berthet et al., 2020; Pogancic et al., 2020; Elmachtoub & Grigas, 2022; Mandi et al., 2024); relaxed compilation is a non-differentiable procedure, so we optimize through it with a black-box evolution strategy. Route-relaxation memory such as ng-routes (Baldacci et al., 2011) refines the state space rather than the ranking; Section 6.6 reports it on top of our rule. Appendix J expands these threads.

3 RANKING STATES UNDER A COMPILATION BUDGET

3.1 THE DECISION, AND SCORING IT BY $f{+}h$

Let L_k be the states generated at layer k before merging. When $|L_k| > W$, the compiler partitions L_k into groups eligible to merge and decides, within each over-full group, which states remain separate. We take the partition as given and study that decision. Each state receives a score and the smallest are kept; the standard score is the root distance $f(s)$, the cheapest path cost from the root to s, which accounts only for cost already paid. We rank instead by

$$\text{score}(s) \;=\; f_\lambda(s) \;+\; h(k, i_c(s), i_t(s)), \tag{1}$$

where f_λ is the root distance in Lagrangian reduced costs under multipliers λ and h estimates the remaining cost. We call this $f{+}h$. It has the form of the A* evaluation function but a different role, as Section 3.2 explains. We index h by (k, i_c, i_t), so it is a table of $O(mn^2)$ entries built once per compile and read once per candidate; the served set is deliberately excluded, for the reason Section 4 gives. Each entry is a linear map over 22 features of the cell and the instance (Appendix F), normalized by the instance's median travel time L so that a policy fitted at one size applies at another. With $\phi(k, i_c, i_t) \in \mathbb{R}^{22}$ and parameters θ,

$$h_\theta(k, i_c, i_t) \;=\; h_0(k, i_c, i_t) \;+\; L \cdot \theta^\top \phi(k, i_c, i_t), \tag{2}$$

where h_0 is the coarse cost-to-go from the backward pass of a width-one diagram. Predicting a correction to h_0 means $\theta = 0$ recovers the existing $f{+}h$ rule, so a policy that learns nothing degrades to a strong baseline. The cost of h_0 is charged to every method that uses it.

Reduced costs put states that have served different customers on a comparable scale, and h_0 is already on it; we retain them for consistency but measure no benefit from them in the ranking itself (Appendix N). The coarse cost-to-go improves over root distance by 5.65 points; the multipliers alone do not.

3.2 WHAT $f{+}h$ DOES AND DOES NOT GUARANTEE

Proposition 1. *Let $\mathcal{D}_h$ be the diagram compiled by the procedure of Section 2.2 with candidate states scored by equation 1 for an arbitrary function h. Then the shortest root-to-terminal path in $\mathcal{D}_h$ is a lower bound on the optimum, for every h.*

The score enters the compiler only in choosing which states of an over-full group remain separate; it changes neither arc costs nor the merge operator, and any partition is a relaxation (Appendix K). Unlike learned heuristics for A*, where an inadmissible estimate invalidates optimality (Futuhi & Sturtevant, 2026), an inaccurate h here weakens the bound but cannot invalidate it, so h may be trained by any procedure, including one that never sees a feasible solution. Validity does not buy quality: 10% multiplicative noise on a good h removes 3.5 of its 5.65-point gain, 30% is worse than no cost-to-go, and dropping the dependence on i_c keeps only 1.63 points (Appendix D). The estimate must be fine-grained and free of systematic error in the ordering it induces, which Section 4 shows the supervised target is not. Bound quality is also not monotone in width or merge-key granularity, since partitions at different granularities are not nested (Appendix L); every comparison in Section 6 is therefore at matched cost, not matched width.

Table 1: Mean gap (per cent) for one linear policy fitted to nine signals at $m = 8$, on 100 held-out instances per size; the bound row averages five optimizer seeds. *min*/*mean*: how per-state values are aggregated into a cell. *own*/*eval.* λ: backward values read under the source diagram's re-optimized multipliers or under the width-one multipliers used at inference. Relaxed sources are a diagram of width $2W$ refined for 2 self-guided rounds, or one of width $8W$ refined for 8.

signal	m=8	m=15	signal	m=8	m=15
coarse h_0, no model	12.44	16.42	wide, own λ, min, 2 rounds	11.62	15.20
exact h^*, min	12.15	16.46	wide-lam0, eval. λ, min, 2	11.43	15.26
exact h^*, mean	11.07	14.47	wide-mean, own λ, mean, 2	11.95	15.44
bound, no target	**9.18**	**11.20**	wide, own λ, min, 8 rounds	12.60	16.75
			wide-lam0, eval. λ, min, 8	10.99	14.45
			wide-mean, own λ, mean, 8	15.06	19.12

4 WHY IMITATING A RELAXED COST-TO-GO FAILS

The natural source of h is a wider diagram: compile it, run the backward pass, use the node values as targets. We show these targets are poorly suited to ranking and separate three distortions: relaxed backward values are systematically optimistic; aggregating a cell by minimum reproduces the effect from exact values; and the source diagram's multipliers differ from those used at inference. Table 1 is the evidence for all three: one linear policy fitted to different signals at $m = 8$, the largest size at which the exact cost-to-go is computable, and evaluated at $m = 8$ and $m = 15$ with instances, features, and class fixed (protocol in Section 6.5).

4.1 VALUES FROM A RELAXED DIAGRAM ARE SYSTEMATICALLY OPTIMISTIC

Let u be a relaxed node and $X(u)$ the exact states it represents; its backward value is the cheapest completion from u. Since merging takes the most permissive value of each component, every completion available to any $x \in X(u)$ is available from u at no greater cost, so

$$h(u) \leq \min_{x \in X(u)} h^*(x), \tag{3}$$

where h^* is the exact cost-to-go. The inequality makes the bound valid; for ranking, the right-hand side is the problem. It is a minimum over a group whose membership the merging procedure decided, not the state being scored, so the value falls with group size for a reason unrelated to whether the states were worth keeping. Figure 1 draws the consequence: two candidates with equal reduced cost, one in a small block and one in a large one, receive different scores, and the ranker keeps the one merging already judged interchangeable with many others. A fitted model reproduces the pattern because the error is systematic, not noise; seventeen times the training data moves the supervised baseline by three quarters of a point (Section 6.3).

4.2 AGGREGATING A CELL BY ITS MINIMUM

The same structure arises without relaxation. The policy of Section 3.1 reads h from a table indexed by (k, i_c, i_t), so state-level values must be aggregated into cells. If the aggregate is a minimum, equation 3 holds with $X(u)$ replaced by the cell, even when every $h^*(x)$ is exact. Table 1 isolates this: exact values aggregated by minimum reach 12.15% at $m = 8$ and 16.46% at $m = 15$, the latter marginally worse than the coarse table's 16.42%. Holding the values fixed and changing the aggregate to the mean moves the policy to 11.07% and 14.47%, nearly two points at $m = 15$ from one change of aggregate. Exactness itself is worth less: at $m = 15$ the exact mean beats the relaxed mean-aggregated target at two rounds by under a point.

4.3 MULTIPLIER MISMATCH BETWEEN FITTING AND INFERENCE

Backward values from a wider diagram are costs in *that* diagram's multipliers, re-optimized during its compilation; the evaluator ranks in the width-one multipliers h_0 uses. For the $2W$ source at two refinement rounds the mismatch is 0.06 points (15.26% against 15.20% at $m = 15$) and 0.19 at

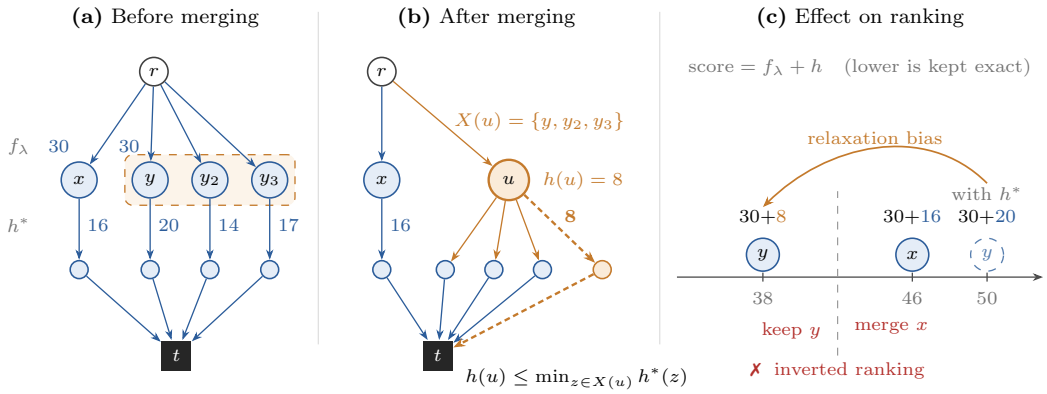


Figure 1: Why a relaxed backward value misranks. *(a)* Two candidates with equal accumulated cost $f_\lambda = 30$: x, which will survive alone, and y, which sits in a group with y_2 and y_3. Their true cost-to-go is $h^*(x) = 16$ against $h^*(y) = 20$, so x is the better state. *(b)* Merging the group into u takes the most permissive value of every component, so u admits completions no member had (dashed) and its backward value $h(u) = 8$ falls below even the smallest h^* in the group, which is Equation 3. *(c)* Ranking by $f_\lambda + h$ scores y at 38 and x at 46, so y is kept separate and x is merged, although y's true score is 50. The target is low because merging happened, not because the state has a good continuation, and the ordering inverts. Nothing in the argument uses relaxation: the same picture describes exact values aggregated into a table cell by minimum. Values are illustrative.

$m = 8$. For the $8W$ source at eight rounds the mismatch dominates: *wide* degrades from 15.20% to 16.75% while the same values under the evaluator's multipliers improve from 15.26% to 14.45%, and the gap opens to 2.30 points at $m = 15$ and 1.61 at $m = 8$. A target closer to what the wider diagram achieves is further from a useful ranking unless priced in the units the ranking is read in. This is the only distortion with a free repair, and it does not close the gap: the best relaxed signal, *wide-lam0* from the $8W$ source, reaches 14.45% against 11.20% for the bound-trained policy. The distortions also interact: *wide-mean* degrades from 15.44% to 19.12% with the stronger source, since averaging removes min-aggregation bias but spreads the mismatch across the cell.

4.4 A PER-STATE ORACLE DOES NOT HELP EITHER

A remaining hypothesis is that the table's indexing, not its values, is the bottleneck. We test it by removing the model and querying a $2W$ diagram's per-state values directly during compilation, under three lookup rules with no charge for that diagram, so the oracle has strictly more information than any fitted model. All three degrade the bound on every instance: at $m = 10$ over 100 instances the wide diagram's own (k, i_c, i_t) *table* reaches 2.42% and the coarse table 4.90%, while exact-match and containing-node lookup of its per-state values give 11.66% and 11.60%, losing 9.24 (95% CI $[-9.91, -8.58]$) and 9.18 points ($[-9.84, -8.53]$) against the wide table, paired within instance; lookup after self-guided refinement gives 11.76%, 6.86 points ($[-7.37, -6.38]$) behind the coarse table it starts from. The information is there, and exposing per-state values discards it (Appendix C).

5 TRAINING ON THE BOUND

5.1 THE OBJECTIVE AND THE POLICY CLASS

Let $\mathcal{I}$ be a set of training instances and $\mathrm{gap}(\theta, I)$ the relative gap from compiling instance I at the operating width with h_θ. We minimize

$$\mathcal{L}(\theta) \;=\; \frac{1}{|\mathcal{I}|} \sum_{I \in \mathcal{I}} \mathrm{gap}(\theta, I). \tag{4}$$

There are no targets. Evaluating $\mathcal{L}$ costs one compilation per instance, the operation performed at deployment, and needs no solved instance, so training is not confined to sizes an exact method

reaches. Proposition 1 holds at every iterate, so every θ the search visits yields a valid bound. Figure **??** in Appendix I sets this beside the imitation pipeline; the two differ only in what θ is fitted to and both cost one compile at inference.

The policy is Equation 2: a linear correction to h_0 over the 22 features of Appendix F, which describe the cell (depth, distances among i_c, i_t, and the depot, the isolation and Lagrangian price of each position) and the instance, and include h_0 itself; the served set is absent, for the reason Section 4.4 gives. The linear class is deliberate: the claim concerns the signal, and Section 6.5 finds no material gain from a 385-parameter perceptron or a 30-parameter per-state policy.

5.2 The search, and what training and inference cost

Compilation has discrete selection steps, so $\mathcal{L}$ has no useful gradient; differentiable surrogates (Berthet et al., 2020; Pogancic et al., 2020) need a smoothed selection operator, and at 22 parameters a derivative-free method suffices. We use a $(1 + 1)$ evolution strategy: propose a Gaussian perturbation of the incumbent, accept if the pooled gap improves, adapt the step by the one-fifth rule with a floor and re-inflation after repeated rejections (the plain rule drove the step to 10^{-4} within sixty steps and stalled). Search starts at the fitted ridge weights, so it cannot return something worse than imitation on the training set.

Training is one compilation plus bound evaluation (200 subgradient iterations) per instance per step, paid once: about two hours on one core at $m = 15$ and thirteen at $m = 25$ with eight instances and 120 steps, roughly 8 and 49 seconds per evaluation. At inference, filling the table is one $O(mn^2) \times 22$ matrix product, about 0.1 seconds against a compile of 21–39 seconds at the largest sizes; the precompute of h_0 is the expensive part, a $\Theta(m^5)$ width-one backward pass measured at 0.6, 2.9, and 9.2 million arcs at $m = 15$, 20, and 25, or 19, 30, and 39 per cent of the arcs charged to an $f{+}h$ compile. Both are charged everywhere; because the precompute grows faster than the compile, the advantage over root distance narrows with size at matched arcs even as it grows at matched width (Section 7).

6 Experiments

6.1 Setup

Instances. Two families are ours, *uniform* and *single-centre* (Appendix B). The third is the generator of Tang & van Hoeve (2024): locations uniform on a 1000×1000 plane, the first the depot, drone times divided by $\alpha \in \{2, 4\}$, ten instances per configuration as in their protocol. Their m counts locations including the depot and we count customers, so their $m \in \{15, 20, 25, 30\}$ are our $n \in \{14, 19, 24, 29\}$. We do not compare against their published numbers: their seeds are unavailable and their gaps are against a CP Optimizer incumbent.

Seeds, width, and cost. Seeds 0–29 are development, used for every design choice and never reported; 100+ are training; 1000–1099 are test, 100 instances per cell, used once, disjointness checked before any experiment runs (Appendix G). The compiler merges only states in the same (i_c, i_t) bucket, so a width cap is satisfiable only when $W \geq (m + 1)^2$; we fix one width from the largest size in the main study, $(30 + 1)^2 = 961$, hence $W = 1024$ for 12–30 customers. Cost is compiled arcs; every method is charged its auxiliary compilations, including the width-one diagram for h_0, with inference converted to arcs at the compiler's measured throughput. Rules at different widths are compared at matched cost by interpolating the cheaper rule along its width ladder, only where that count is bracketed by two ladder points, never extrapolating or clamping.

Upper bound and intervals. Gaps are $(\mathrm{UB} - \mathrm{LB})/\mathrm{UB}$, with UB the best of nine restricted-diagram constructions, cached per instance; its slack is measured exactly at $m \leq 12$ and bounded below at $m \in \{15, 20\}$ (Appendix E), and it rescales differences between methods but cannot reorder them. Intervals are paired 95% bootstraps, 10,000 resamples. Every experiment whose reading could depend on its outcome had that reading fixed before it ran (Appendix M).

6.2 DOES THE RANKING RULE MATTER?

We first evaluate the rule without learning: root distance along its width ladder against $f{+}h$ with the coarse h_0, the configuration we call *round* 0 throughout, interpolated to the same arc count with h_0 charged, on 100 uniform instances per cell (Table 9, Appendix O). At matched cost and width 1024, $f{+}h$ gains $+4.49$ points at $m = 15$ (95% CI $[+3.87, +5.10]$, $92/100$), $+3.87$ at $m = 20$ ($[+3.35, +4.40]$, $90/100$), and $+3.09$ at $m = 25$ ($[+2.63, +3.55]$, $92/100$); at width 2048, $+4.70$ ($[+4.12, +5.26]$, $91/100$) and $+4.51$ ($[+3.95, +5.07]$, $92/100$) at $m = 15$ and 20. Every instance is bracketed by the ladder. How the gain moves with size depends on how cost is controlled: at matched arcs it falls with m, while at equal width it rises, $+5.65$, $+5.75$, $+6.08$. The difference is the $\Theta(m^5)$ precompute against the $\Theta(Wm^3)$ compile: 0.6M, 2.9M, and 9.2M arcs at the three sizes, a fitted exponent of 5.3 in m, and 19%, 30%, and 39% of the total charged. At $m = 15$, width 2048, ng-2 route memory adds $+1.75$ points for 5% more arcs; two remembered customers quadruple the bucket floor to $4(m + 1)^2$, which equals 1024 at $m = 15$ and exceeds it from $m = 16$, so at the operating width the ng row is the one configuration allowed to exceed the cap (Section 6.6).

6.3 BASELINES: SELF-GUIDED $f{+}h$ AND IMITATION

Self-guided $f{+}h$. A compilation at width W yields backward values and multipliers that can guide another at the same width, and the loop iterates with no model and no data. We registered a rule before training anything: with C the gain of a reference cost-to-go over round 0 and B the best self-guided gain within twice the round-0 arcs, $B/C \geq 0.70$ would mean the headroom is reachable without learning. It fired at 96%, 94%, and 98% on uniform $m = 15$, uniform $m = 20$, and single-centre $m = 15$, and it is not sound, because the reference is not a ceiling: on uniform $m = 15$ it reaches 12.49% while rounds 2 and 3 reach 10.70% and 9.42% (Appendix N). The comparison that should have been registered is cost against accuracy, and Table 11 (Appendix O) and Figure 4 make it. Throughout, *the learned policy* is the per-instance mean over five optimizer seeds of the policy trained at $m = 15$; single-seed figures are labelled. It reaches 10.31% at 2.88M arcs; round 0 gives 16.42% at that cost, and the loop needs about 8.3M arcs, 2.9 times the compilation, to reach 10.31%. Within instance it improves over round 0 by 6.11 points (95% CI $[5.74, 6.49]$) and over ridge by 4.37 ($[4.05, 4.69]$), on $100/100$ for both; the best single seed of either training size, an $m = 25$ policy, reaches 10.08%.

Imitation of relaxed targets. The supervised baseline fits a scalar offset (*const*), ridge, or gradient-boosted trees (*gbt*) to backward values from a relaxed diagram of width $2W = 2048$, aggregated to (k, i_c, i_t) by minimum, compared with h_0, written r_0, on 500 held-out instances (uniform $m \in \{12, 15, 20\}$, single-centre $m \in \{12, 15\}$, 100 each). *const* shifts every cell by the same amount, so it cannot change the ranking except where the shift resolves a tie in floating point; it gains -0.00 ($[-0.00, +0.00]$, $3/500$), the sanity check it was included as. On eight training instances of Table 8, ridge gains 1.77 ($[1.67, 1.86]$, $487/500$) and gbt 2.03 ($[1.91, 2.15]$, $477/500$); with 140 instances, seventeen times the learned policy's budget, ridge gains 2.53 ($[2.42, 2.64]$, $498/500$) and gbt 2.71 ($[2.58, 2.84]$, $492/500$). The nonlinear class buys little: gbt beats ridge by 0.26 ($[0.16, 0.36]$) and 0.18 ($[0.08, 0.27]$), ridge captures 87% and 93% of the tree gain, and at the largest size the trees are 0.56 worse ($[-0.77, -0.33]$, $28/100$); ridge is the supervised baseline.

6.4 FROZEN POLICIES ACROSS INSTANCES, SIZES AND FAMILIES

Policies are trained on eight uniform instances at $m = 15$ or $m = 25$, frozen, and evaluated on the 100 test instances at $m \in \{15, 20, 25, 30\}$ on uniform and at $m = 15$ on single-centre, five optimizer seeds per configuration. We report the per-instance mean over seeds whose evaluation completed on all 100 instances, with the seed count and spread (Table 2). The registered criterion, at least 2.0 points over the fitted baseline with a paired interval excluding zero, is met in every cell.

Transfer across size and family. Fitted at $m = 15$ and applied at $m = 20$, the policy reaches 13.76% against 22.08% for round-0 $f{+}h$, 8.32 points (95% CI $[7.89, 8.76]$, $100/100$); at $m = 25$, 18.85% against 27.66%, 8.82 ($[8.36, 9.28]$, $100/100$); on single-centre, which differs in geometry and depot placement, 13.94% against 21.50%, 7.56 ($[7.13, 8.03]$, $100/100$). At $m = 30$ one seed completed: 28.11%, 8.20 over round 0 ($[7.74, 8.67]$, $100/100$) but 2.80 over ridge ($[2.36, 3.24]$, $85/100$), so the margin over imitation narrows on that seed at twice the training size.

Table 2: Frozen policies on instances, sizes and families they were not fitted on: mean gap over 100 test instances, per cent. Learned columns: per-instance mean over n complete optimizer seeds (spread from best seed to worst). Last column: gain of the $m = 15$-trained seed mean over ridge, paired 95% interval, instances won. All policies were trained on *uniform*, so single-centre rows are cross-family and $m \geq 20$ rows cross-size; root distance at width 1024 was measured for the three cells of the self-guided study. Gains quoted in the text are paired per-instance means computed before rounding, so they can differ by 0.01 from differences of the rounded means here. One of the five $m = 25$-trained seeds completed no evaluation cell, hence $n = 4$ throughout that column; at $m = 30$ the single $m = 25$-trained seed evaluated reached 99 of 100 instances and is not reported.

	root	$f + h$	ridge	learned, trained at		learned (m=15)
evaluated on	distance	round 0	(imitation)	$m = 15$	$m = 25$	over ridge
uniform $m = 15$	22.07	16.42	14.68	10.31 (0.48, n=5)	11.17 (1.90, n=4)	4.37 [4.05, 4.69], 100/100
uniform $m = 20$	27.83	22.08	19.21	13.76 (0.37, n=5)	14.11 (1.29, n=4)	5.45 [5.08, 5.83], 100/100
uniform $m = 25$	—	27.66	23.93	18.85 (0.32, n=5)	18.27 (1.13, n=4)	5.09 [4.66, 5.52], 99/100
uniform $m = 30$	—	36.31	30.91	28.11 (n=1)	—	2.80 [2.36, 3.24], 85/100
single-centre $m = 15$	29.59	21.50	19.74	13.94 (1.23, n=5)	15.05 (2.89, n=4)	5.81 [5.42, 6.21], 100/100
mean arcs, uniform $m = 15$	2.47M	3.05M	—	2.88M	2.89M	

Against imitation and self-guidance. Over ridge the seed mean gains 4.37 on uniform $m = 15$, 5.45 at $m = 20$, 5.09 at $m = 25$, and 5.81 on single-centre, on at least 99/100 in each. Against the loop of Table 11, matching the seed mean by interpolation between rounds, self-guidance needs about 8.3M arcs to match the policy's 2.88M on uniform $m = 15$ (2.9×) and 8.5M against 2.92M on single-centre (2.9×); on uniform $m = 20$ three rounds reach 14.53% after 29.19M arcs against 9.31M (3.1×), still 0.8 points behind (Figure 4).

Training at $m = 25$ is noisier, and better only at its own size. At $m = 15$, $m = 20$, and on single-centre, policies fitted at $m = 25$ supply the best single seed in two cells (10.08%, 13.30%) and the worst in all three, with spreads of 1.90, 1.29, and 2.89 points against 0.48, 0.37, and 1.23 for $m = 15$ training; at $m = 25$ itself they lead by 0.58 with spread 1.13 against 0.32. Since 120 steps buy less when each is an $m = 25$ compile and $m = 15$ policies transfer upward within 0.6 points of policies fitted there, $m = 15$ training is the method.

6.5 Controls: policy class, and exact targets

Capacity is not the explanation. Holding signal, instances, and evaluation fixed, we vary the class: *linear*, the 22 table features; *state*, those plus eight per-state features from the served set and τ (30 parameters); and *MLP*, a two-layer perceptron with 16 tanh units over the 22 table features (385 parameters). This is a separate run: one optimizer seed per class, 120 steps (240 for the perceptron, which starts from small random weights), the same eight training instances. On uniform $m = 15$ the three classes improve over ridge by 4.53, 4.87, and 4.32 points, the linear figure agreeing with the 4.37–4.60 of Table 2. The registered bar for capacity not binding was linear within 1.0 of the perceptron; it is within 0.21, and the largest class is the worst. The bar for per-state information being usable under direct optimization was *state* ahead by 2.0; it gains 0.34, the third measurement pointing the same way after Sections 4.4 and 6.3. The perceptron result is confounded with the optimizer (385 parameters is a harder black-box search at fixed budget); the per-state conclusion rests on the two small classes.

Exact targets do not rescue imitation. The exact cost-to-go is computable at $m = 8$, where the exact diagram has 595,456 nodes; at $m = 9$ it exceeds memory. We train every signal of Table 1 at $m = 8$ on the same instances, features, and class, and evaluate on 100 held-out instances at $m = 8$ and $m = 15$, the bound averaged over five optimizer seeds. *exact-mean*, exact and not min-aggregated, is the fairest imitation target that exists and the best of the four, and it is not competitive: 11.07% against 9.18% at $m = 8$, 1.88 points ([1.56, 2.22], 88/100), and 14.47% against 11.20% at $m = 15$, 3.27 points ([2.98, 3.57], 99/100). The registered strong reading required at least 1.0 point with an interval excluding zero and is met at both sizes. Exact targets exist only far below the sizes the method is for, so this is a control on the objective, not an alternative to it.

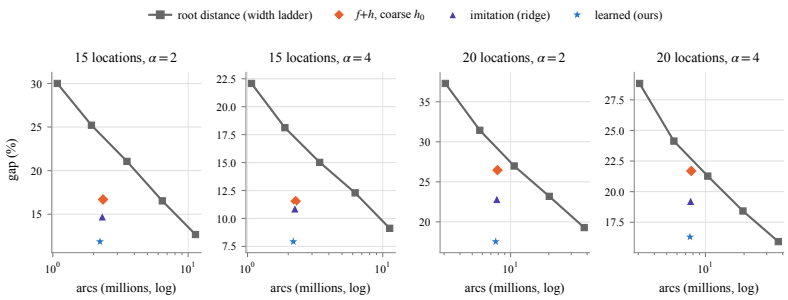


Figure 2: The distribution of Tang & van Hoeve (2024), 10 instances per configuration, gap against arcs (log). Root distance is drawn along its full width ladder (512–8192); the learned policy, fitted on a different generator at a different size and frozen, sits below the whole curve in three of four configurations; in the fourth, 19 customers at $\alpha = 4$, root distance at width 8192 passes it by 0.38 points at about five times the arcs. Values in Table 10, Appendix O.

6.6 A DISTRIBUTION FROM THE LITERATURE

We evaluate on the generator of Tang & van Hoeve (2024), reproduced as their Section 8 specifies, at both drone speeds, with one policy fitted on our uniform family at 25 customers and frozen; it has never seen this generator and α is not among its features (Figure 2; values and paired intervals over 10 instances in Table 10).

At matched arcs the frozen policy beats root distance by 12.57, 9.60, 12.18, and 6.69 points, round-0 $f{+}h$ by 4.83, 3.62, 8.93, and 5.39, and ridge by 2.87 to 5.30. In three of four configurations root distance does not reach it on the mean at any width we can compile, trailing by 0.81, 1.19, and 1.75 points at 8192, five times the budget, though only the last interval excludes zero (Table 10). In the fourth, 19 customers at $\alpha = 4$, it overtakes at $4.9\times$ for the 8192 rung, or $4.5\times$ (34.3M arcs) interpolating between 4096 and 8192, a point 85% of the way across a bracket from 19.6M to 36.9M on which no qualitative claim rests. ng-route memory with two remembered customers, which refines the state space rather than the ranking, quadruples the bucket floor (1600 at 19 customers), so the ng row alone may exceed the requested width of 1024 and its excess is charged in arcs; it helps $f{+}h$ at $\alpha = 4$ by 0.88 and 3.50 points, costs 0.02 and 1.23 at $\alpha = 2$ for 6–25% more arcs, and trails the learned policy on every mean, by 1.89 to 10.16 points, with the interval including zero only at 19 customers and $\alpha = 4$. The ng row is our implementation and these are our draws from their distribution, so the claim is about the distribution, not their solver; our 25- and 30-location blocks were incomplete and are not reported.

7 LIMITATIONS AND CONCLUSION

Appendix N reports what did not work. The limitations that bear on the claims: the $\Theta(m^5)$ precompute is charged, so the matched-arc advantage narrows with size and we do not know where it crosses zero; transfer is evaluated with five seeds only to 25 customers, and the one seed at 30 shows the margin over imitation narrowing; everything here is one problem and one relaxation; the uniform family's depot sits at a corner of the customer region, as in the public benchmarks, so depot placement is not isolated from geometry; one matched-cost figure rests on a wide bracket; the perceptron result is confounded with the optimizer; and slack is unmeasured above 20 customers, so absolute gaps there are to our bound, though paired comparisons are unaffected.

The ranking that governs merging in a relaxed decision diagram is worth learning, and the way to learn it is not to imitate a stronger relaxation: a target defined as a minimum over a group of states carries a bias proportional to how much merging has already happened, and it survives exactness, survives removing the model, and worsens as the source is strengthened. Fitting the same policy to the bound it produces removes the target and the bias with it. Where the objective is a non-differentiable procedure whose by-product is the obvious supervised target, that target encodes the procedure's own compromises; here, optimizing through the procedure beat every imitation target we constructed, exact ones included.

AI USE STATEMENT

We used generative AI assistants (ChatGPT and Claude) in preparing this submission. Their use was in three areas: drafting and revising manuscript text, including section drafts that the authors then edited; writing the scripts that aggregate experiment outputs; and auditing the manuscript for internal consistency between its tables, figures and text.

Generative AI was not used to generate the research question, the relaxation, the merge operator, the compiler or the experiment code, or the proof of Proposition 1, and it produced no experimental results: every number reported here comes from runs of the authors' own code.

All AI-assisted text and scripts were read, checked and re-run by the authors. We take full responsibility for the content of this paper, including its claims, numbers and artifacts.

ETHICS STATEMENT

This paper studies lower-bound certification for synthetic routing benchmarks. It involves no human subjects and no personal data, and we see no application-specific risks beyond those of optimization methods generally.

REPRODUCIBILITY STATEMENT

- **Theoretical results.** Proposition 1, that the compiled bound is valid for every ranking function h, is stated in Section 3.2 and proved in Appendix K. Equation 3, the bias on which Section 4 rests, is derived in Section 4.1. Appendix L states the non-monotonicity of bound quality in width and merge-key granularity that requires every comparison to be matched on cost rather than on width.
- **Algorithm and models.** The ranking score is Equation 1 and the policy class Equation 2 (Sections 3 and 5.1); the training objective is Equation 4; the search, its step-size rule and its warm start are in Section 5.2, with training and inference costs. Appendix F defines all 22 table features and the 8 per-state features of the extended class, including one feature that is constant in the implementation used for every reported run.
- **Experimental protocol.** Appendix G lists every experimental constant, imported by the scripts from one module. Development (0–29), training (100+) and test (1000–1099) seeds are disjoint, and disjointness is asserted before any experiment runs (Section 6.1). Instances are determined by family, size, seed and drone speed ratio (Appendix B), so they are regenerated rather than distributed. The upper bound against which all gaps are computed, and the measurement of its slack, are in Appendix E.
- **Analyses fixed in advance.** Every experiment whose reading could depend on its outcome had that reading registered beforehand; the criteria are in Appendix M. Appendix N reports the analyses that did not come out as specified, including one pre-registered criterion we judge unsound and therefore do not use, and the warm-start and training-set-size studies.
- **Completeness and known defects.** A result is reported only for evaluation cells complete on all 100 test instances; incomplete cells are named and excluded rather than averaged (Section 6.4, Appendix H). Appendix A documents two defects found in the relaxation, the results withdrawn because of them, and the property tests that fail on the defective implementation and pass on the corrected one.
- **Source code and instructions.** The supplementary material contains the full source code in anonymised form: the compiler, the policy and training code, one shell script per experiment, the instance generators, the upper-bound ensemble, the completeness audit and the test suite. Appendix H describes how experiments are run and checked; reproducing a table means running the numbered script for that experiment and then the completeness check.

REFERENCES

Niels Agatz, Paul Bouman, and Marie Schmidt. Optimization approaches for the traveling salesman problem with drone. *Transportation Science*, 52(4):965–981, 2018.

Roberto Baldacci, Aristide Mingozzi, and Roberto Roberti. New route relaxation and pricing strategies for the vehicle routing problem. *Operations Research*, 59(5):1269–1283, 2011.

Saverio Basso and Matteo Salani. Learning dominant states in elementary resource constrained shortest path problems. *arXiv preprint arXiv:2605.03760*, 2026.

David Bergman, Andre A Cire, Willem-Jan Van Hoeve, and John Hooker. *Decision diagrams for optimization*, volume 1. Springer, 2016a.

David Bergman, Andre A Cire, Willem-Jan Van Hoeve, and John N Hooker. Discrete optimization with decision diagrams. *INFORMS Journal on Computing*, 28(1):47–66, 2016b.

Quentin Berthet, Mathieu Blondel, Olivier Teboul, Marco Cuturi, Jean-Philippe Vert, and Francis Bach. Learning with differentiable perturbed optimizers. *Advances in neural information processing systems*, 33:9508–9519, 2020.

Natashia Boland, Mike Hewitt, Luke Marshall, and Martin Savelsbergh. The continuous-time service network design problem. *Operations Research*, 65(5):1303–1321, 2017.

Quentin Cappart, Emmanuel Goutierre, David Bergman, and Louis-Martin Rousseau. Improving optimization bounds using machine learning: Decision diagrams meet deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 1443–1451, 2019.

Quentin Cappart, David Bergman, Louis-Martin Rousseau, Isabeau Prémont-Schwarz, and Augustin Parjadis. Improving variable orderings of approximate decision diagrams using reinforcement learning. *INFORMS Journal on Computing*, 34(5):2552–2570, 2022.

Jingyi Chen, Xinyuan Zhang, and Xinwu Qian. Neural certificate pricing for combinatorial optimization problems. *arXiv preprint arXiv:2607.01185*, 2026.

Francesco Demelas, Joseph Le Roux, Mathieu Lacroix, and Axel Parmentier. Predicting lagrangian multipliers for mixed integer linear programs. In *ICML*, pp. 10368–10384, 2024.

Adam N Elmachtoub and Paul Grigas. Smart "predict, then optimize". *Management Science*, 68(1): 9–26, 2022.

Nikolaus Frohner and Günther R Raidl. Merging quality estimation for binary decision diagrams with binary classifiers. In *International Conference on Machine Learning, Optimization, and Data Science*, pp. 445–457. Springer, 2019a.

Nikolaus Frohner and Günther R Raidl. Towards improving merging heuristics for binary decision diagrams. In *International Conference on Learning and Intelligent Optimization*, pp. 30–45. Springer, 2019b.

Ehsan Futuhi and Nathan Sturtevant. Learning admissible heuristics for a*: theory and practice. In *International Conference on Learning Representations*, volume 2026, pp. 78781–78811, 2026.

Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. *Advances in neural information processing systems*, 32, 2019.

Ryo Kuroiwa and J Christopher Beck. Domain-independent dynamic programming: Generic state space search for combinatorial optimization. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 33, pp. 236–244, 2023.

Jayanta Mandi, James Kotary, Senne Berden, Maxime Mulamba, Victor Bucarey, Tias Guns, and Ferdinando Fioretto. Decision-focused learning: Foundations, state of the art, benchmark and future opportunities. *Journal of Artificial Intelligence Research*, 80:1623–1701, 2024.

Chase C Murray and Amanda G Chu. The flying sidekick traveling salesman problem: Optimization of drone-assisted parcel delivery. *Transportation Research Part C: Emerging Technologies*, 54: 86–109, 2015.

Mohsen Nafar and Michael Römer. Using clustering to strengthen decision diagram bounds for discrete optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 8082–8089, 2024.

Augustin Parjadis, Quentin Cappart, Bistra Dilkina, Aaron Ferber, and Louis-Martin Rousseau. Learning lagrangian multipliers for the travelling salesman problem. *arXiv preprint arXiv:2312.14836*, 2023.

Rahul Patel, Elias B Khalil, and David Bergman. Heuristic multiobjective discrete optimization using restricted decision diagrams. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pp. 378–403. Springer, 2026.

Marin Vlastelica Pogancic, Anselm Paulus, Vit Musil, Georg Martius, and Michal Rolinek. Differentiation of blackbox combinatorial solvers. In *ICLR*, 2020.

Stefan Poikonen, Bruce Golden, and Edward A Wasil. A branch-and-bound approach to the traveling salesman problem with a drone. *INFORMS Journal on Computing*, 31(2):335–346, 2019.

Giovanni Righini and Matteo Salani. New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks: An International Journal*, 51(3):155–170, 2008.

Roberto Roberti and Mario Ruthmair. Exact methods for the traveling salesman problem with drone. *Transportation Science*, 55(2):315–335, 2021.

Isaac Rudich, Quentin Cappart, and Louis-Martin Rousseau. Peel-and-bound: Generating stronger relaxed bounds with multivalued decision diagrams. *arXiv preprint arXiv:2205.05216*, 2022.

Ziye Tang and Willem-Jan van Hoeve. Dual bounds from decision diagram-based route relaxations: An application to truck-drone routing. *Transportation Science*, 58(1):257–278, 2024.

Madison Van Dyk and Jochen Koenemann. Dynamic discretization discovery under hard node storage constraints. *arXiv preprint arXiv:2303.01419*, 2023.

A Two defects in the relaxation, and the tests that pin them

Allowing the truck to revisit an already served customer to collect the drone is necessary for the model, as 10 known optimal solutions contain such a move. This requirement also complicates the relaxation because a merged node must retain the set of nodes that remain valid rendezvous points. We identified two implementation defects related to this mechanism. Both are now covered by property tests that fail on the defective implementation and pass on the corrected one. The first affected previously computed results, while the second concerned the conditions required to certify the lower bound.

A.1 Defect 1: phantom rendezvous

Invariant. For every state reachable in the exact diagram, the truck is located either at a served customer or at the depot:

$$i_t \in \mathrm{served} \cup \{0\},$$

and the same property holds for i_c. This follows by induction over the transition rules. A truck or combined arc to customer j sets $i_t = j$ and adds j to served. A drone arc sets $i_t = r$ for a rendezvous node $r \in \{i_t, 0\} \cup \mathrm{served}$, which is already served. No other transition changes the truck position.

Violation under merging. The merge operator intersects the served sets of its members and unions their rendezvous sets. A merged node can therefore retain a rendezvous point r that belongs to the union but not to the intersection. A subsequent drone transition to r then produces a state in which the truck is located at a customer absent from the merged served set.

This violation creates a zero-cost transition. Since the diagonal entries of the travel-time matrix satisfy $T[r, r] = 0$, the truck can transition to its current position and mark that customer as served while advancing one layer. If this state is subsequently merged with others, the intersection can again remove r from served, allowing the same effect to propagate to descendants.

Effect on the apparent value of memory. Route-relaxation methods can restrict merging by requiring states to agree on selected customer information, as in ng-routes (Baldacci et al., 2011) and decremental state space relaxation (Righini & Salani, 2008). Our implementation of the size-one geometric memory rule ranked candidates using $T[\mathrm{node}, 1{:}]$ with a stable argsort. Because $T[c, c] = 0$, the nearest customer to node c is c itself. The rule therefore retained only the bit corresponding to the truck's current node. This bit is constant for exact states and, unintentionally, prevents the invariant violation described above.

At 12 customers and width 2048, the defective implementation produced a gap of 8.9% without memory and 3.1% with the size-one geometric rule. After correcting the merge operator to preserve the invariant, both configurations produce a gap of 2.55%. The apparent improvement from memory in this experiment was therefore caused by the memory rule suppressing the defect rather than by a tighter route relaxation.

This result concerns only the size-one geometric rule evaluated on the defective implementation. The ng-route memory of size two reported in Section 6.6 uses a different rule and is evaluated on the corrected implementation. Its measured effect is therefore separate from this defect: it improves the bound at $\alpha = 4$ and increases cost without improving the bound at $\alpha = 2$.

Three oracle searches had also identified the size-one rule as unimprovable. Their search spaces could only remove or replace the retained bit and could not introduce additional structure. The result was therefore determined by the restricted move set and is not used as evidence in this paper.

A.2 Defect 2: rendezvous containment

Simulation property. Validity of the Lagrangian and linear programming bounds requires a simulation property between exact and relaxed states. For every exact state x represented by a relaxed node u, each transition available from x must also be available from u at no greater cost. For our state representation, this requires

$$\mathrm{served}(u) \subseteq \mathrm{served}(x)$$

and

$$\mathrm{rendez}(u) \supseteq \mathrm{served}(x),$$

since every node served on the path to x is a valid rendezvous point for x.

Violation. The defective implementation constructed successors using served alone and therefore discarded rendezvous options introduced by merging. We exhaustively checked the simulation property at 5 customers for widths from 2 to 128. It failed for $9{,}772$ of $271{,}361$ checked $(\text{exact state}, \text{relaxed node})$ pairs. A minimal counterexample occurs on the uniform instance with seed 0 and drone speed factor $\alpha = 1$, using root-distance ranking at width 32. The first violation occurs in layer 3.

Correction and computational effect. The corrected implementation propagates the rendezvous set through every transition. When two arcs produce identical raw targets in $(\mathrm{served}, i_c, i_t, \tau)$, their rendezvous masks are unioned. This union preserves the state variables while retaining all available rendezvous options.

After the correction, we observe 0 violations among $180{,}641$ checked pairs; the count differs from the pre-fix figure because the pairs are enumerated from the relaxed diagrams actually compiled, and the corrected merge operator produces different diagrams. Exact diagrams and restricted upper bounds remain unchanged. Lower bounds change by at most 0.4%. The corrected relaxed diagrams are, however, substantially larger: diagram size increases by 65% at 12 customers and by 88% at 15 customers. We therefore recomputed all cost-matched comparisons that had been obtained with the defective implementation.

We did not observe an invalid bound before the correction. In particular, all 566 checked optimal-path embeddings succeeded. These empirical checks are insufficient to certify validity, however, because the required simulation property was violated. The corrected implementation restores this property.

A.3 Regression tests

Both defects are covered by regression and property tests that fail on the pre-fix implementation and pass on the corrected implementation.

- `test_truck_node_always_served` compiles relaxed diagrams under several policies, widths, and instance families and checks the invariant of Appendix A.1 for every reachable node.
- `test_geometric_memory_is_the_self_bit` reproduces the indexing behavior of the size-one geometric rule and preserves the explanation of the withdrawn results.
- `test_neighbourhood_memory_is_a_true_neighbour` verifies that the corrected memory rule selects a customer other than the node itself.
- `test_simulation_holds` exhaustively checks the containment property of Appendix A.2 at small instance sizes across multiple widths and families.
- `test_known_counterexample_is_fixed` fixes the instance, width, and layer of the original simulation violation as a regression case.

The complete test suite contains $1{,}217$ tests. It also cross-checks four independent sources on every benchmark instance: the published optimum, brute-force search, a subset dynamic program, and the decision diagram. These independent comparisons provide regression checks for both the optimization model and its implementation.

A.4 Implications for the reported results

Results obtained with the defective relaxation are not reported in this paper. All results that depend on relaxed compilation were recomputed using the corrected implementation. The supplementary code includes both merge operators for reproducibility, with the corrected implementation selected by default. The regression tests above fail on the defective implementation and pass on the corrected one. This makes it possible to reproduce the defect and its effect separately from the corrected results reported in the paper.

B The instance generators

Each instance is determined by its family, number of customers, random seed, and drone speed ratio α. We include all four quantities in instance identifiers, cache keys, and result files. For example, `gen-uniform-m15-s1000-a2` denotes a uniform instance with 15 customers, seed 1000, and $\alpha = 2$. Instances that differ only in α are treated as distinct problems because they have different travel times and bounds.

The two synthetic families. In *uniform*, the n customer locations are sampled independently and uniformly from $[0, 100]^2$, while the depot is sampled from $[0, 1]^2$. The depot is therefore concentrated near a corner of the customer region rather than distributed among the customers. This construction follows the convention of the public TSP-D benchmark instances of Agatz et al. (2018). We reproduce the underlying spatial distribution rather than the published instances themselves: the

benchmark files use integer-valued customer coordinates, whereas we sample customer locations continuously from the same region.

In *single-centre*, both the customers and the depot are sampled from an isotropic Gaussian distribution centred at the origin with standard deviation 50. The depot is therefore drawn from the same spatial distribution as the customers. Both synthetic families use a drone speed ratio of $\alpha = 2$. A third *double-centre* family is implemented in the released code but is not used in the experiments reported here.

The two families differ in both the spatial distribution of customers and the location of the depot relative to them. We therefore evaluate them as separate instance families and do not interpret differences between their results as effects of customer geometry alone.

The published distribution. The third family follows the generator of Tang & van Hoeve (2024) as specified in their Section 8. For a given number of locations and drone speed ratio α, locations are sampled uniformly from a 1000×1000 Euclidean plane. The first sampled location is designated as the depot and the remaining locations are customers. Truck travel distances are Euclidean, while drone travel distances are obtained by dividing the corresponding Euclidean distances by α. Following their experimental protocol, we generate 10 instances for each configuration with $\alpha \in \{2, 4\}$.

The random seeds used by Tang & van Hoeve (2024) are not published. We therefore generate new instances from the same distribution rather than reproducing their specific instances.

Instance-size convention. The size convention differs between our work and Tang & van Hoeve (2024). Their m denotes the total number of locations, including the depot, whereas we use n to denote the number of customers. Their sizes $m \in \{15, 20, 25, 30\}$ therefore correspond to $n \in \{14, 19, 24, 29\}$ in our notation. We report both conventions in all tables involving this distribution.[1]

The published ng-route comparison is available only for 15, 20, and 25 locations, as column generation exceeded the reported time limit for larger instances. Consequently, there is no published ng-route result for comparison with our largest reported instance size.

C THE PER-STATE ORACLE IN FULL

Configurations. We evaluate five configurations at $m = 10$ and width 1024 on the same 100 test instances, using a common upper bound.

table-coarse uses h_0, the cost-to-go table obtained from the width-one diagram. This is the inexpensive estimate used throughout the paper.

table-wide uses the same (k, i_c, i_t) lookup on a diagram of width $2W = 2048$, the same source width as the imitation baseline of Section 6.3. We do not charge the cost of constructing this diagram. This configuration isolates the effect of obtaining a stronger cost-to-go table and serves as the reference for the corresponding per-state lookups.

state-exact queries the same wide diagram using the full state representation $(\text{layer}, \text{served set}, i_c, i_t)$. An exact match is available for 18–21% of states across instances. When no exact match is available, the lookup falls back to the coarse value.

state-subset uses the exact match when available and otherwise selects the tightest node whose served set contains that of the query state. Together, exact and containing-node matches cover essentially all queries, with approximately 20% handled by exact matches and 80% by containing-node matches.

state-boot applies the state-subset lookup after self-guided refinement of the guiding diagram. This configuration tests whether refining the source of the per-state values can recover the performance of the coarse estimate.

[1] Section 8 of Tang & van Hoeve (2024) describes the generation loop using $n \in \{15, 20, 25, 30, 40, 50\}$ while indexing configurations by (m, α), despite defining $m = n + 1$. We interpret these values as numbers of locations because their results table reports the same values under "Number of locations," and the surrounding text refers to the corresponding runs as $m \in \{15, 20, 25\}$. Our customer counts are therefore one smaller than the sizes reported in their tables.

Table 3: Per-state lookup at $m = 10$ and width 1024 over 100 test instances. The cost of constructing the wide diagram is excluded. Differences are paired within instance, with negative values indicating worse performance than the corresponding reference.

configuration	addressed by	mean gap	vs. ref.	95% CI	wins
table-wide	cell	2.42%	—	—	—
table-coarse	cell	4.90%	—	—	—
state-subset	state	11.60%	-9.18	$[-9.84, -8.53]$	$0/100$
state-exact	state	11.66%	-9.24	$[-9.91, -8.58]$	$0/100$
state-boot	state	11.76%	-6.86	$[-7.37, -6.38]$	$0/100$

The reference is *table-wide* for *state-subset* and *state-exact*, and *table-coarse* for *state-boot*. The latter comparison tests whether self-guided refinement can improve over the inexpensive estimate from which it starts.

Compiler control. The per-state configurations use a separate compilation loop, which introduces a potential implementation confound. We therefore verify that `compile_with_state_h` exactly reproduces the production compiler when supplied with a table-valued cost-to-go function. This test is executed before the oracle experiments. It ensures that the observed degradation is attributable to the per-state values rather than to differences between the two compilation implementations.

D How accurate does a cost-to-go have to be?

Section 4 argues that a systematically biased cost-to-go estimate can be worse than using no estimate. We examine how this behavior depends on the accuracy of the estimate. Starting from h_0 at $m = 15$ and width 1024, we introduce controlled degradations while keeping the instances, upper bounds, and compiler settings fixed. We use root distance at the same width and on the same instances as the no-estimate baseline.

Table 4: Effect of degrading the cost-to-go estimate at $m = 15$ and width 1024 over 100 instances. Differences are paired within instance against root distance at the same width. Positive values favor the variant.

variant	mean gap	vs. no estimate	95% CI	wins
h_0, λ dropped from f	16.27%	$+5.80$	$[+5.41, +6.21]$	$100/100$
h_0	16.42%	$+5.65$	$[+5.00, +6.30]$	$93/100$
10% multiplicative noise	19.95%	$+2.12$	$[+1.38, +2.86]$	$75/100$
averaged over i_c	20.44%	$+1.63$	$[+1.11, +2.13]$	$78/100$
no estimate (root distance)	22.07%	0	—	—
λ only, $h = 0$	22.74%	-0.67	$[-1.26, -0.11]$	$41/100$
30% multiplicative noise	24.24%	-2.17	$[-2.92, -1.43]$	$30/100$

The improvement comes from the cost-to-go estimate. Retaining the Lagrangian multipliers while setting $h = 0$ performs worse than root distance by 0.67 percentage points, with a 95% confidence interval of $[-1.26, -0.11]$. This indicates that the improvement from $f+h$ is attributable to the cost-to-go estimate rather than to the use of Lagrangian multipliers in the ranking score.

Performance degrades with the quality of the estimate. The full h_0 improves over root distance by 5.65 percentage points. Adding 10% multiplicative noise reduces this improvement to 2.12 points, while averaging over i_c reduces it to 1.63 points. Both variants remain better than the no-estimate baseline. With 30% multiplicative noise, the effect reverses: the resulting gap is 2.17 points worse than root distance, with worse performance on 70 of the 100 instances. These results show that sufficiently degraded cost-to-go estimates can be detrimental rather than merely uninformative. This behavior is relevant to imitation from biased targets, where errors can systematically alter the ordering used during compilation.

Reduced costs have no detectable effect on ranking in this experiment. Removing λ from the root-distance term, while retaining the priced cost-to-go estimate, changes the gap by $+0.15$

percentage points. The corresponding 95% confidence interval is $[-0.42, +0.74]$, and the variant wins on 47 of 100 instances. We therefore do not observe a measurable effect from using reduced rather than raw accumulated costs in the ranking score on these instances. Reduced costs remain the principled representation when states within a layer have served different customers and their raw accumulated costs are not directly comparable. This ablation concerns only their role in state ranking. All configurations retain the Lagrangian relaxation used to construct the bound and to obtain h_0.

Role of the accumulated-cost term. These experiments do not isolate the contribution of f itself. The current compiler evaluates states using $f+h$ and does not support an h-only ranking rule. We therefore cannot determine from this ablation whether h alone would provide comparable ranking performance.

E THE UPPER BOUND: THE ENSEMBLE AND ITS SLACK

All gaps reported in this paper are computed relative to an upper bound that we construct. The quality of this upper bound therefore affects the magnitude of the reported absolute gaps. This appendix evaluates its slack where an independent reference is available.

The upper bound UB is the minimum over an ensemble of nine constructions. The ensemble contains two anytime primal heuristics and restricted $f+h$ diagrams at three widths, each evaluated with the coarse cost-to-go and with a cost-to-go refined for two self-guided rounds. To assess slack, the comparison must use constructions outside this ensemble. We therefore use a restricted diagram of width $65{,}536$, which is wider than any diagram in the ensemble, and refine its cost-to-go for five self-guided rounds rather than two. If these more expensive constructions do not improve UB, they provide no evidence of additional slack. An earlier version of this analysis compared UB with members of the ensemble itself. Since no member can improve on the minimum over that ensemble, we do not use that comparison here.

Exact measurement at $m = 12$. At $m = 12$, the optimum can be computed by a subset dynamic program, allowing the slack to be measured exactly. Across 30 uniform instances, UB equals the optimum on every instance. The mean slack is 0.00% and the maximum slack is 0.00%. The wider construction also fails to improve UB on any instance. The weakest ensemble member, the anytime heuristic at width 2048, is nevertheless above the optimum on individual instances, by 0.95% on one of them, so the minimum over the ensemble is doing work. At this size, the reported gaps are therefore gaps to the exact optimum.

Lower bounds on slack at $m = 15$ and $m = 20$. At $m = 15$ and $m = 20$, the optimum is unavailable. We can therefore measure only the improvement obtained by constructions outside the ensemble, which provides a lower bound on the slack of UB. Across 30 uniform instances, this improvement is 0.00 points on average at $m = 15$, with a maximum of 0.00 points. At $m = 20$, the mean improvement is 0.12 points and the maximum is 1.18 points. Thus, the wider and more refined construction does not improve UB at $m = 15$. At $m = 20$, it improves UB on a subset of instances, with an improvement of up to 1.18 points.

Table 5: Slack in the upper bound over 30 uniform instances per size. At $m = 12$, the optimum is available and slack is measured exactly. At $m = 15$ and $m = 20$, the reported values are lower bounds on slack obtained from constructions outside the upper-bound ensemble.

	$m = 12$	$m = 15$	$m = 20$
mean slack (points)	0.00	0.00	0.12
maximum (points)	0.00	0.00	1.18
measured against	optimum	wider search	wider search

Scope of the slack analysis. The pre-registered threshold for reporting absolute gaps was one percentage point. The mean slack measurements for all three evaluated sizes fall below this threshold. The absolute gaps reported in Section 6.2 are therefore reported under this criterion.

This analysis has two limitations. First, it extends only to $m = 20$. At $m = 25$ and $m = 30$, the ensemble is the strongest upper-bound construction available in our experiments. We therefore do not report a slack estimate at these sizes, since comparison with the ensemble itself would not provide an independent measurement.

Second, the main paired comparisons do not rely on the upper bound being exact. Methods evaluated on the same instance use the same UB. For two lower bounds LB_1 and LB_2, the difference between their reported gaps is

$$\frac{\mathrm{LB}_2 - \mathrm{LB}_1}{\mathrm{UB}}.$$

The common upper bound therefore preserves the ordering of the two lower bounds, although its slack can rescale the magnitude of the reported difference. This applies to the comparisons in Section 6.2, the ablations in Appendix D, and the controls in Section 6.5. The slack analysis is therefore required for interpreting absolute gaps, but not for determining which method produces the tighter lower bound at matched cost.

F THE FEATURE SET

The policy in Equation 2 maps a feature vector $\phi(k, i_c, i_t) \in \mathbb{R}^{22}$ to a correction of the coarse cost-to-go h_0. Features are computed once per instance and evaluated once for each table cell, allowing the complete table to be obtained with a single matrix product. All time-valued quantities are normalised by L, the median of the positive entries of the truck travel-time matrix T for the corresponding instance. This normalisation allows the same policy to be applied across instance sizes.

Let n denote the number of nodes including the depot, and let m denote the number of customers. We write $\lambda \in \mathbb{R}^m$ for the Lagrangian multipliers obtained from the width-one diagram and $\lambda^+ \in \mathbb{R}^n$ for the same vector padded with zero at the depot index.

Table 6: The 22 features of the policy, computed once per table cell (k, i_c, i_t). Every quantity with units of time is divided by L, the median of the positive entries of the truck travel time matrix, which is what allows a policy fitted at one problem size to be applied at another.

#	definition	role
1	k/m	depth through the diagram
2	$(m-k)/m$	customers still to serve
3	$T[i_c, i_t]/L$	distance between the two position components
4	$T[i_t, i_c]/L$	reverse distance, which differs when T is asymmetric
5	$T[0, i_c]/L$	depot distance of the combined node
6	$T[0, i_t]/L$	depot distance of the truck node
7	$\min_{j \neq i_c} T[i_c, j]/L$	minimum outgoing distance from i_c
8	$\min_{j \neq i_t} T[i_t, j]/L$	minimum outgoing distance from i_t
9	$\overline{T[i_c, \cdot]}/L$	mean outgoing distance from i_c
10	$\overline{T[i_t, \cdot]}/L$	mean outgoing distance from i_t
11	$\max_j T[i_c, j]/L$	maximum outgoing distance from i_c
12	$\max_j T[i_t, j]/L$	maximum outgoing distance from i_t
13	$\lambda^+[i_c]/L$	Lagrangian multiplier at i_c
14	$\lambda^+[i_t]/L$	Lagrangian multiplier at i_t
15	$\overline{\lambda^+}/L$	mean Lagrangian multiplier
16	$\max \lambda^+/L$	maximum Lagrangian multiplier
17	$h_0(k, i_c, i_t)/L$	coarse cost-to-go being corrected
18	$\mathbf{1}[i_c = i_t]$	indicator that the drone is with the truck
19	$\mathbf{1}[i_c = 0]$	indicator that the vehicles were last together at the depot
20	$\mathbf{1}[i_t = 0]$	indicator that the truck is at the depot
21	$m/25$	normalized problem size
22	constant 0.5	constant feature; see below

Feature 22 is constant. Feature 22 was intended to encode the drone speed ratio. Due to the way this field is read in the implementation, its default value is always used, making the feature equal to

0.5 for every instance and table cell. We retain the feature in the description because all reported experiments were run with this implementation, and removing it would require rerunning them. Since the feature is constant across instances and cells, it carries no information about the drone speed ratio. In particular, the policy has no direct access to this parameter in the reported transfer experiments.

Features excluded from the table policy. The served set is not included in the table-level feature representation. A per-state cost-to-go that depends on the served set can be obtained from a wider relaxed diagram, but Section 4.4 shows that using such values degrades the resulting bound. The elapsed desynchronisation time τ is also absent because the table is indexed only by (k, i_c, i_t).

We separately evaluate an extended policy with eight additional features computed from each raw state's served set and τ, listed in Table 7. On uniform instances at $m = 15$, this policy improves by 0.34 percentage points over the 22-feature policy. This is below the pre-specified threshold of 2.0 points for concluding that per-state information provides a meaningful improvement. Section 6.5 reports the full comparison. The per-state oracle in Section 4.4 provides a complementary test of the same question.

Table 7: Eight additional features used by the extended policy. These features are computed for each raw state rather than for each table cell. Here U is the set of customers not yet served by the state.

#	definition
23	$\lvert U\rvert/m$
24	$\sum_{j\in U} T[i_t, j]/L$
25	$\min_{j\in U} T[i_t, j]/L$
26	$\max_{j\in U} T[i_t, j]/L$
27	$\min_{j\in U} T[i_c, j]/L$
28	$\sum_{j\in U} \lambda^{+}[j]/L$
29	τ/L
30	$\lvert\text{served}\rvert/\max(k, 1)$

Computational cost. The instance-level quantities require $O(n^2)$ computation and are evaluated once per instance. The complete table is then produced by a single matrix product of size $O(mn^2) \times 22$. On the largest instances considered, inference takes approximately 0.1 seconds, compared with 21–39 seconds for diagram compilation. We nevertheless include inference in every cost-matched comparison by converting its runtime to arc equivalents using the compiler throughput measured on the same instance.

G Protocol constants and how they are fixed

All experimental constants are defined in a single protocol module and imported by the scripts that use them. Experiment-specific modules do not maintain separate copies. The test suite checks agreement with the protocol before experiments are run, preventing inconsistent hard-coded values across scripts. This centralization ensures that experiments intended to share a configuration use the same settings.

The protocol module also defines constants for two analyses that were not completed for this version and are not reported: a σ sweep over $\{0.025, 0.05, 0.2, 0.4\}$ and the attainable-improvement reference search of Appendix M (200 compilations per instance); the 32-instance point of the sample-efficiency curve is likewise missing. They are omitted from Table 8 so that the table lists only what the paper reports. Two entries in Table 8 are determined by the implementation rather than treated as independent experimental choices. The operating width follows the bucket-floor constraint described in Section 6.1. The arc ceiling is read by the compiler from the environment at import time and is re-exported by the protocol module rather than duplicated.

The protocol also distinguishes between the self-guided rounds used in the upper-bound ensemble and those used in the frontier experiment. These quantities serve different experimental roles and are defined separately in the implementation, even when their numerical values coincide. Keeping

Table 8: Protocol constants used throughout the experiments.

constant	value	what it fixes
Instances		
development seeds	0–29	design and development; excluded from reported results
training seeds	100+	supervised fits and policy search
test seeds	1000–1099	final reported results
N_{test}	100	instances per evaluation cell
N_{pub}	10	instances per published configuration
N_{slack}	30	instances for the upper-bound slack check
Compilation		
W	1024	operating width for 12–30 customers
RD ladder	512–8192	root-distance widths for matched-arc comparisons
max arcs	6×10^7	compiler arc ceiling; re-exported from the compiler
Lagrangian iterations	200	subgradient steps per bound
ng memory	2	customers retained in the ng-route configuration
Training		
training set	8 instances	equal at $m = 15$ and $m = 25$
optimizer seeds	0–4	five seeds per headline policy
steps	120	evolution-strategy steps
σ_0	0.1	initial step size
warm starts	ridge, zero	ridge is the method; zero reported in Appendix N
sample-efficiency	4, 8, 16, 24	training-set sizes, one seed, Appendix N
Controls		
exact-target size	8 customers	size at which the exact cost-to-go is computed
bootstrap	10,000 resamples	paired within instance

them separate prevents changes to one experiment from implicitly changing the upper bounds used throughout the paper.

H Reproducing the results

The supplementary code provides one shell script for each experiment. Each script invokes a single Python driver and validates the resulting output. Scripts are numbered according to their dependencies and can also be executed independently. For example,

```
./sh/run_03_ranking.sh
```

Experiment logic is separated from job scheduling. Each driver can either execute its commands directly or emit them as independent jobs for external scheduling. The execution environment does not change the experimental configuration. In particular, every command receives the first test seed explicitly rather than relying on a default value. The same instances are therefore evaluated whether the commands are run manually, submitted through a job array, or executed through the complete pipeline.

Completeness checks. We verify experimental completeness at the level of individual $(\text{instance}, \text{configuration})$ pairs rather than by file size or row count. For each experiment, the repository defines the expected evaluation grid and checks the output for both missing and duplicate cells using

```
python scripts/check_complete.py
```

An experiment is considered complete only when every expected cell occurs exactly once.

This check is necessary because some evaluation modules append results to existing files. Rerunning such a module can create duplicate rows, while an interrupted evaluation can omit complete configurations without producing an obviously incomplete file. The audit detects both cases. Each

experiment script also specifies whether it can be safely re-run or requires its previous output to be removed.

Computational cost. All experiments reported in this paper were run on a single workstation with 24 CPU cores and 32 GB of memory. No shared computing cluster was used. The complete experimental pipeline requires approximately 250 CPU-hours, with most of the computation spent on the transfer grid and policy search. Independent runs were parallelized across the available cores when memory permitted.

Memory, rather than the number of CPU cores, is the main constraint at larger instance sizes. The cost-to-go precomputation scales as $\Theta(m^5)$ and requires approximately 6 GB per process at 30 customers. Consequently, the largest evaluation cells run only three or four instances concurrently, despite the availability of 24 CPU cores.

I THE TWO TRAINING PIPELINES

Figure **??** draws the imitation pipeline of Section 4 and the direct-optimization pipeline of Section 5 side by side. Both end at the same inference procedure.

J EXTENDED RELATED WORK

Learning inside decision diagrams. Cappart et al. (2019) introduced machine learning for directly improving DD relaxation bounds by using deep reinforcement learning to select variable orderings for relaxed and restricted DDs. This approach was later extended to branch and bound (Cappart et al., 2022). Patel et al. (2026) learn node-selection heuristics for restricted DDs using supervision from exact Pareto frontiers. Other approaches to tightening DD bounds include peel-and-bound (Rudich et al., 2022). These methods do not learn a cost-to-go function for ranking states during merging under a fixed compilation budget. Existing learned approaches instead act on discrete ordering or selection decisions, often using imitation or reinforcement learning and, in some cases, exact solutions. We instead rank states using $f+h$ in Lagrangian reduced costs and train h directly against the bound produced by compilation.

Learned cost-to-go for search. Learning cost-to-go estimates to guide search is well established. In A*, admissibility is required to preserve optimality, and recent work formulates heuristic learning as a constrained optimization problem to maintain this property (Futuhi & Sturtevant, 2026). Domain-independent dynamic programming and beam search also use learned state evaluations to guide exploration (Kuroiwa & Beck, 2023). Related learning-to-search methods include graph-network policies for branching (Gasse et al., 2019). In these settings, the learned estimate changes the search trajectory and can affect the returned solution or the effort required to prove optimality, so accuracy and admissibility are central requirements. Our setting differs because h is used only to rank states before merging. It changes which states remain separate and therefore affects the tightness of the relaxed bound, but not its validity. Any ranking function can be used without invalidating the relaxation. The accuracy requirements are nevertheless consequential for bound quality: Appendix D shows that 10% multiplicative noise on h removes 3.5 of the 5.65 percentage-point improvement obtained from the cost-to-go estimate, while 30% noise performs worse than using no cost-to-go estimate.

Optimizing a bound rather than imitating labels. Several recent approaches train models directly against an optimization bound rather than supervised labels. Demelas et al. (2024) predict Lagrangian multipliers for mixed-integer linear programs by optimizing the resulting dual bound without supervision. Parjadis et al. (2023) similarly learn multipliers for the TSP by directly maximizing the bound. Related unsupervised approaches to pricing and state dominance have been studied for shortest-path and combinatorial pricing problems (Chen et al., 2026; Basso & Salani, 2026). When gradients through the solver are unavailable, alternative approaches construct differentiable surrogates: perturbed optimizers (Berthet et al., 2020), black-box differentiation through combinatorial solvers (Pogancic et al., 2020), and the broader decision-focused and predict-then-optimize frameworks (Elmachtoub & Grigas, 2022; Mandi et al., 2024). These methods typically

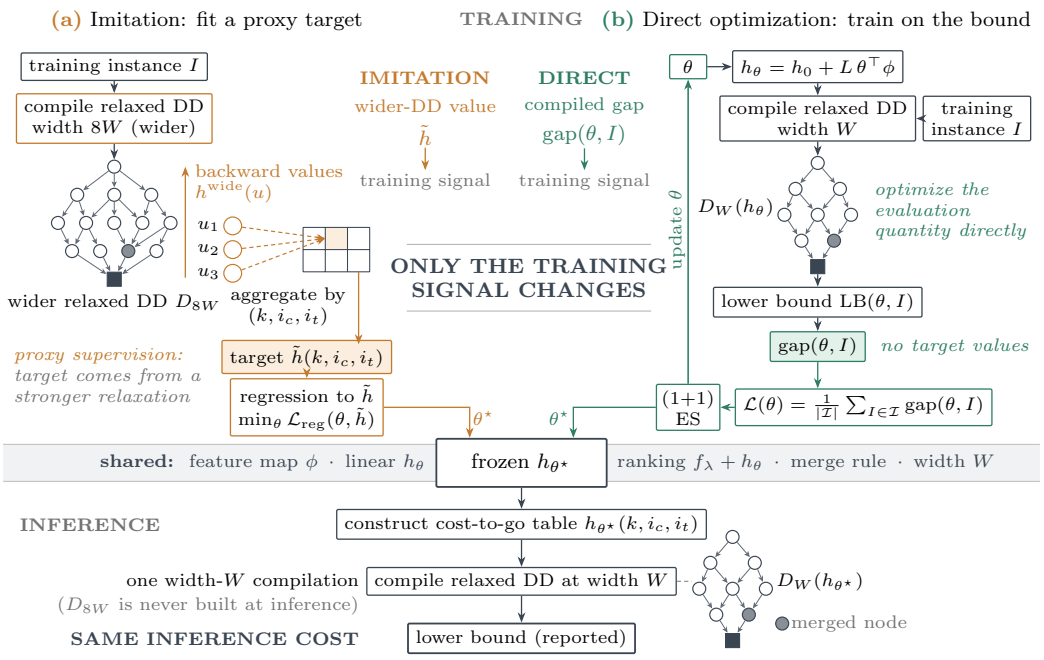


Figure 3: **Imitation fits an intermediate cost-to-go; direct optimization trains on the compiled bound.** Both methods train the *same* policy $h_\theta = h_0 + L\,\theta^\top \phi$, a cost-to-go table indexed by (k, i_c, i_t), and deploy it identically; only the training signal differs. **(a) Imitation.** For each training instance I, a wider relaxed DD D_{8W} is compiled and a backward pass yields values $h^{\mathrm{wide}}(u)$ for its nodes. These values are aggregated over all states sharing a cell (k, i_c, i_t) to form the target $\tilde{h}$, and θ is fit by regression to $\tilde{h}$. The supervision is therefore an intermediate quantity produced by a different, stronger relaxation, not the bound that is ultimately evaluated. **(b) Direct bound optimization.** The current h_θ is used by the operating compiler at width W to build $D_W(h_\theta)$, whose lower bound $\mathrm{LB}(\theta, I)$ defines $\mathrm{gap}(\theta, I)$. A (1+1)-ES minimizes $\mathcal{L}(\theta) = \frac{1}{|\mathcal{I}|} \sum_{I \in \mathcal{I}} \mathrm{gap}(\theta, I)$ by treating the compiler as a black box: no target values are constructed and no gradients are propagated through the compiler. **Shared components.** The feature map ϕ, the linear policy class, the ranking rule $f_\lambda + h_\theta$, the merge rule, and the operating width W are identical in both arms. **Inference.** Either frozen $h_{\theta^\star}$ is converted into a cost-to-go table and used in a single width-W compilation, so deployment cost is the same for both methods; D_{8W} is used only during imitation training. Solid arrows denote computation, dashed arrows denote aggregation or correspondence, and the loop arrow denotes the ES update. Gray nodes are merged nodes; black squares are terminals.

optimize objectives associated with LP or Lagrangian duals for which subgradients or differentiable surrogates can be constructed. Relaxed DD compilation is instead a non-differentiable combinatorial procedure. We therefore optimize the ranking policy through compilation using a black-box $(1 + 1)$ evolution strategy, with one compilation per objective evaluation. We also study why the natural supervised alternative fails in this setting: backward values extracted from a wider relaxed diagram become systematically optimistic for heavily merged states, so training on them can assign favorable rankings to states whose low target values arise from relaxation rather than from better futures.

Refining the state space instead of the ranking. An alternative way to strengthen a relaxation is to refine the state space itself. Dynamic discretization discovery iteratively introduces time points into time-expanded networks (Boland et al., 2017; Van Dyk & Koenemann, 2023). Route-relaxation methods such as ng-routes (Baldacci et al., 2011) and decremental state-space relaxation (Righini & Salani, 2008) retain additional partial-path information to eliminate short cycles. These approaches

operate on a different component of the relaxation: they determine which distinctions are retained in the state representation, whereas we hold the state representation fixed and learn the ranking used when merging becomes necessary. The two mechanisms can be combined; Section 6.6 evaluates ng-route memory in addition to our ranking rule. Section 4.4 tests the distinction more directly: even with free access to per-state values from a wider diagram, finer cost-to-go addressing produces a weaker bound on every instance evaluated.

Why this matters for a problem that is hard to solve exactly. Exact TSP-D methods reach roughly 39 customers (Roberti & Ruthmair, 2021), while the branch-and-bound method of Poikonen et al. (2019) is evaluated on smaller benchmark instances. Methods trained by imitating optimal solutions are consequently limited to instance sizes for which such solutions can be obtained. Our approach requires no solved training instances because the training objective is the bound produced by compilation. A policy can therefore be trained at one instance size and applied unchanged at another: Section 6.4 reports a policy trained at $m = 15$ reaching 13.76% at $m = 20$ against 22.08% for the round-0 rule, and Section 6.6 reports the same frozen policy on an independently defined distribution.

K Proof of Proposition 1

The score affects only the selection of states within an over-full group. It determines which states remain as separate nodes and which are assigned to merge blocks. Changing h therefore changes this partition but does not alter the arc costs or the merge operator. For any such partition, each merged node takes the most permissive value of every state component over its members. Every transition available from a member state is therefore represented by the merged node at no greater cost. Consequently, every feasible solution of the original problem has a corresponding root-to-terminal path in $\mathcal{D}_h$ whose cost is no greater. The shortest root-to-terminal path in $\mathcal{D}_h$ is therefore a lower bound on the optimum. This is the standard validity argument for relaxed decision diagrams (Bergman et al., 2016b) and does not depend on how the partition is selected. □

L Refinement is not monotone

Increasing the granularity of the merge key or the compilation budget does not necessarily produce a tighter bound. The partitions induced at different widths or granularities need not be nested. A finer merge key can therefore produce a different partition rather than simply subdividing the blocks obtained with a coarser key, and the resulting shortest path can be weaker.

Our relaxation partitions each layer into at least $(m + 1)^2$ buckets before applying the width cap, with one bucket for each pair of positions. Remembering k customers in the merge key multiplies this minimum number of buckets by 2^k. As this floor approaches W, an increasing fraction of the width budget is required simply to represent the partition, leaving fewer nodes available to retain promising states separately. Section 6.1 derives the operating width from this constraint.

The operating width. The floor is a lower bound on W, not a choice of it. For the main experiments we fix one width across instance sizes so that the compilation budget is comparable, and set it from the largest size in the study: for $m = 30$, $(m+1)^2 = 961$, hence $W = 1024$, the next power of two, for every instance with 12–30 customers. At $m = 12$ this is roughly six times the minimum admissible width of 169, so merging is less frequent and the ranking policy has less influence; we use $m = 12$ chiefly for the upper-bound and imitation experiments rather than the main comparisons. For larger instances we increase W only when 1024 no longer satisfies the floor: the thresholds are $(40 + 1)^2 = 1681$ and $(50 + 1)^2 = 2601$, giving widths of 2048 and 4096.

This behavior follows from imposing a fixed per-layer width, as in restricted diagrams and beam search. Tang & van Hoeve (2024) use a different budget, limiting the number of nodes retained within each bucket to M. Their Remark 4 gives a diagram size of $O(Mn^3)$, without the same minimum-width constraint. The two budgets are comparable near $M = W/(m+1)^2$ but diverge as m increases. We retain the fixed-width formulation because the learned ranking operates directly on this budget: the width cap determines how capacity is allocated across the layer and therefore makes state ranking consequential. As a result, bound quality should not be assumed to vary monotonically

with compilation budget, and comparisons between ranking or merging rules must control for computational cost rather than relying on equal widths. Section 6 follows this matched-cost protocol throughout.

M ANALYSES FIXED IN ADVANCE

Several experiments admit multiple plausible interpretations depending on their outcomes. To avoid choosing these interpretations retrospectively, we specified the criteria below before running the corresponding experiments. The criteria were implemented in the experimental scripts before evaluation, and the repository history records their implementation.

Transfer criterion. We consider a frozen policy to transfer to an evaluation cell if it improves the gap over the fitted baseline by at least 2.0 percentage points and the paired bootstrap confidence interval excludes zero. Results that do not satisfy both conditions are classified as failures to transfer, irrespective of the sign of the mean difference. Section 6.4 reports all evaluation cells.

Policy class. We compare three policy classes under the same objective, training instances, and optimizer: the linear model over table features used throughout the paper; the same model extended with features computed for individual states; and a two-layer multilayer perceptron over the table features, which has approximately seventeen times as many parameters and receives twice the optimization budget to reduce the effect of under-training. If the linear model is within 1.0 percentage point of the multilayer perceptron, we treat capacity as unlikely to be the limiting factor and retain the linear model. If the perceptron improves over the linear model by at least 2.0 points, we repeat the seed-sensitivity analysis using the perceptron, in which case the comparison with imitation is no longer fully controlled for policy class. If the per-state model improves over the linear model by at least 2.0 points, this provides evidence that per-state information can be useful under direct optimization even when it is ineffective under imitation. The perceptron comparison has an additional limitation: the optimizer is derivative-free, and optimizing several hundred parameters with only a few hundred evaluations may be difficult, so a weak perceptron result reflects the combination of policy class and optimization procedure. This limitation does not apply to the lower-dimensional linear and per-state models.

Exact-target control. At $m = 8$ the exact cost-to-go can be computed, allowing training targets to be compared while holding features and policy class fixed. We evaluate this control at the training size and at $m = 15$, with three outcomes specified in advance. If direct bound optimization improves over the best available imitation target, including exact and mean-aggregated targets, by at least 1.0 percentage point with a confidence interval excluding zero, we interpret the result as evidence that direct optimization outperforms imitation in this setting. If the difference is smaller than 1.0 point or its interval includes zero, we restrict the conclusion to the bias of relaxation-derived or min-aggregated targets. If exact-mean imitation improves over direct optimization by at least 1.0 point, we treat the target-bias mechanism rather than direct optimization as the primary contribution of the experiment.

Share of attainable improvement. To quantify the fraction of available improvement a method recovers, we define the attainable span as $\mathrm{gap}(h_0) - \mathrm{gap}(\text{reference search})$ and, for a method M, the share $\big(\mathrm{gap}(h_0) - \mathrm{gap}(M)\big)/\big(\mathrm{gap}(h_0) - \mathrm{gap}(\text{reference search})\big)$, computed as a ratio of means so that all methods use the same denominator. The reference search performs 200 compilations independently for each instance. We call it a reference search rather than the optimum of the policy class because it may optimize directly on the instance it is evaluated on, and we report its incumbent curve to assess whether it has stabilized: if the incumbent continues to improve at 200 compilations, the span is underestimated and the shares overestimated. Because the reference search requires 200 additional compilations per instance, this analysis uses 10 instances rather than 100, with numerator and denominator computed from the same 10. The reference search was not completed for this version and no share is reported; we keep the specification so that the eventual result cannot be read selectively.

Self-guided baseline. Self-guided $f{+}h$, in which the cost-to-go from one compilation guides the next, provides a non-learning baseline. We compare it with the learned policy at matched arc cost,

including all auxiliary compilations. If self-guided $f+h$ achieves comparable bound quality at comparable cost, we do not attribute the improvement to learning. The specific ratio criterion originally attached to this baseline, and why it was found unsound, is described in Appendix N.

N WHAT DID NOT WORK

We evaluated several alternatives that did not improve the final method, including two analyses whose interpretation was specified in advance. We report them because they distinguish failures of the training signal from failures of model capacity, target construction, and experimental design.

Imitating a cost-to-go. Training the ranking policy on backward values from a wider diagram improves over the hand-designed cost-to-go: with 140 training instances, ridge gains 2.53 percentage points over h_0. Directly optimizing the same policy class against the bound performs substantially better, improving by a further 4.37 points on all 100 test instances (Section 6.3). The difference is not specific to relaxed targets. At the size where exact backward values are computable, imitation of exact targets remains 1.88 points worse than direct optimization, and evaluated at $m = 15$ the difference is 3.27 points (Section 6.5).

Improving the imitation target. Replacing the $2W$ source refined for two self-guided rounds by an $8W$ source refined for eight increases the fitted model's gap from 15.20% to 16.75% at $m = 15$. Aggregating the refined values by mean rather than minimum increases it from 15.44% to 19.12%. Exact values aggregated by minimum yield 16.46%, against 16.42% for the coarse table they replace. Improving the fidelity of the target to the source diagram does not produce a better ranking policy, because the ranking objective depends on which states remain separate after merging rather than on pointwise accuracy. The exception is evaluating the values under the multipliers used at inference, the correction Section 4.3 predicts and the only variant that improves with the stronger source.

Increasing policy capacity. Removing the learned representation entirely and querying per-state values from a wider diagram degrades the bound on all 100 instances, by 9.24 points for exact-match lookup and 9.18 for the containing-node rule relative to the wide diagram's own table (Section 4.4). Increasing capacity does little in the other direction: a multilayer perceptron and a per-state policy both perform within 0.35 points of the linear policy (Section 6.5), and gradient-boosted trees improve over ridge by 0.18 points overall but are 0.56 points worse at the largest size tested ($[-0.77, -0.33]$, better on $28/100$).

A pre-registered criterion with an invalid reference. Before training any policy we specified a criterion for whether learning was necessary: a self-guided procedure recovering at least 70% of the gain available from a reference cost-to-go while using no more than twice the arc count. Let C be the improvement of the reference over round 0 and B the largest improvement of a self-guided round whose cumulative arcs do not exceed twice those of round 0; the reference for C was a width-2048 diagram whose construction was not charged. The procedure recovered B/C of 96% on uniform $m = 15$, 94% on uniform $m = 20$, and 98% on single-centre $m = 15$, improving over round 0 on $100/100$ instances in each cell. The ratio is nevertheless not a valid measure of attainable improvement because the reference is not a ceiling: on uniform $m = 15$ it reaches 12.49% while self-guided rounds 2 and 3 reach 10.70% and 9.42%, and the procedure exceeds the reference by its second round in every cell. We retain the experiment because the criterion was specified in advance, but do not use it to assess whether learning is necessary. The relevant comparison is bound quality against compilation cost, in Table 11.

An invalid upper-bound slack control. An initial slack experiment compared the upper bound against constructions drawn from the same ensemble whose minimum defines that bound. No member of an ensemble can improve on its minimum, so the resulting zero spread followed from the definition of the baseline rather than from evidence that the bound was tight. Appendix E replaces this control with constructions strictly outside the ensemble. Because no ensemble member can improve on the ensemble minimum, that comparison yields zero spread by construction, and its output is indistinguishable from a genuine tightness result; only the design reveals the difference.

Reduced costs alone. The two terms of the score must be on a comparable scale, and pricing the covering constraints into the arc costs is the standard way to achieve it; h_0 is already on that scale. The measurement does not support attributing any improvement to it. With $h = 0$, ranking by reduced root distance gives a mean gap of 22.74% against 22.07% for raw root distance, a difference of 0.67 points (95% CI $[-1.26, -0.11]$, better on 41 of 100 instances). Removing the multipliers from the root-distance term while keeping the cost-to-go gives 16.27% against 16.42%, a difference of 0.15 points ($[-0.42, +0.74]$, better on 47 of 100). We retain reduced costs for consistency with the construction of h_0 and make no claim that they contribute to bound quality.

Warm start and training-set size. Two analyses fixed in the protocol were run on uniform $m = 15$ and did not change the method. Starting the search from $\theta = 0$ instead of the ridge weights, with the same eight instances, 120 steps, and five seeds, ends at 10.48% against 10.31% for the ridge start (seed spread 0.64 against 0.48), still 5.94 points over round 0 ($[5.58, 6.31]$, $100/100$): the warm start is worth under a fifth of a point, and the result does not depend on imitation having been run first. Varying the training set with one optimizer seed gives 10.92% with 4 instances, 10.15% with 8, 10.11% with 16, and 10.01% with 24; the planned 32-instance run was not completed. Eight instances recover most of the gain, and tripling them buys 0.14 points.

Sensitivity to cost-to-go accuracy. Adding 10% multiplicative noise removes 3.5 of the 5.65 percentage-point improvement the cost-to-go provides; at 30% noise the ranking performs worse than using no cost-to-go (Appendix D). Moderate errors in the estimated values produce larger changes in the induced ordering, so approximations intended to reduce the cost of h must be evaluated by their effect on ranking and bound quality rather than by prediction error alone.

O SUPPLEMENTARY FIGURES AND TABLES

Table 9: $f{+}h$ against root distance at matched compilation cost, uniform family, 100 instances per row; positive favors $f{+}h$. Every instance is bracketed by the ladder.

size	width	gain (points)	95% CI	wins	no bracket
$m = 15$	1024	$+4.49$	$[+3.87, +5.10]$	$92/100$	0
$m = 15$	2048	$+4.70$	$[+4.12, +5.26]$	$91/100$	0
$m = 20$	1024	$+3.87$	$[+3.35, +4.40]$	$90/100$	0
$m = 20$	2048	$+4.51$	$[+3.95, +5.07]$	$92/100$	0
$m = 25$	1024	$+3.09$	$[+2.63, +3.55]$	$92/100$	0

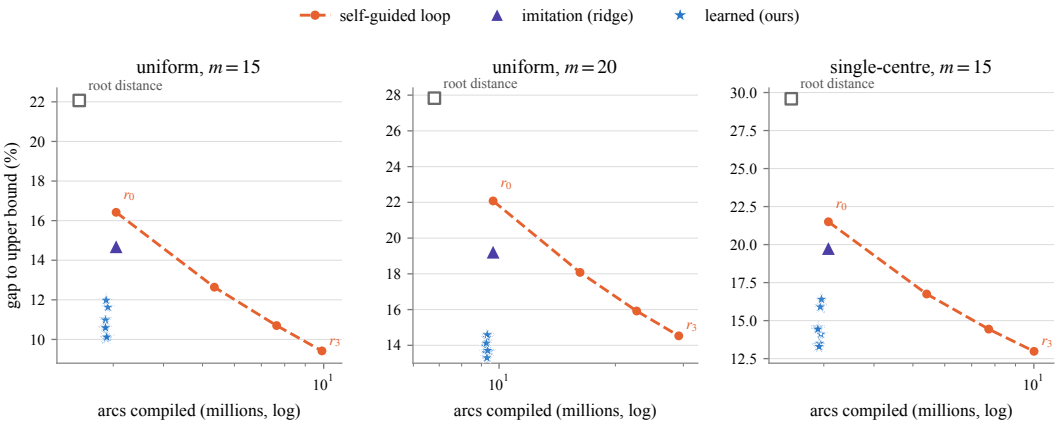


Figure 4: The cost–accuracy frontier on our own families, gap against cumulative arcs on a log axis. The self-guided loop of Section 6.3 is a curve from round 0 to round 3; root distance and imitation are single points; each star is one optimizer seed of the learned policy. Table 11 gives the uniform $m = 15$ values.

Table 10: The distribution of Tang & van Hoeve (2024), 10 instances per configuration, mean gap in per cent; the learned policy was fitted on a different generator at a different size. All rows are compiled at requested width 1024 except the root-distance ladder and the ng row, whose fourfold bucket floor (900 at 14 customers, 1600 at 19) is allowed to exceed the cap and is charged in arcs. *RD at matched arcs* interpolates the ladder (512–8192) at the learned policy's arc cost; *RD@8192* is the widest rung, with its own cost as a multiple of the learned policy's. Paired 95% intervals for the learned policy's gain, by column: over coarse $f{+}h$ $[4.14, 5.54]$, $[2.48, 4.85]$, $[7.44, 10.42]$, $[3.72, 7.21]$; over ridge $[2.24, 3.47]$, $[1.81, 4.34]$, $[4.16, 6.48]$, $[1.43, 4.58]$; over ng-route $[3.61, 6.15]$, $[0.38, 5.12]$, $[8.02, 12.29]$, $[-1.99, 5.69]$; over RD@8192 $[-0.35, 2.08]$, $[-0.48, 2.77]$, $[0.51, 3.15]$, $[-2.60, 1.84]$.

	14 customers (m=15)		19 customers (m=20)	
	α=2	α=4	α=2	α=4
RD@1024	25.21	18.12	31.44	24.13
RD at matched arcs	24.43	17.53	29.72	22.99
$f + h$, coarse h_0	16.69	11.55	26.47	21.69
ng-route memory 2, $f + h$	16.71	10.67	27.70	18.19
ridge (imitation)	14.73	10.88	22.84	19.23
learned, frozen	**11.86**	**7.93**	**17.54**	**16.30**
RD@8192	12.67	9.12	19.29	15.92
its own arc cost	5.1×	5.2×	5.0×	4.9×

Table 11: Self-guided loop against the learned policy, uniform $m = 15$, $W = 1024$, 100 instances. Arcs are cumulative and include the precompute. The reference row is not a method: its source diagram is not charged. The learned row is the mean over five optimizer seeds of the policy trained at $m = 15$ (best seed 10.11%).

	mean gap	mean arcs
root distance, $W = 1024$	22.07%	2.47M
$f + h$, round 0	16.42%	3.05M
$f + h$, h from $W = 256$, charged	15.15%	3.78M
$f + h$, h from $W = 512$, charged	13.05%	4.34M
self-guided, round 1	12.64%	5.35M
self-guided, round 2	10.70%	7.64M
self-guided, round 3	9.42%	9.90M
reference, cost not charged	12.49%	3.02M
learned policy, mean of 5 seeds	**10.31%**	**2.88M**

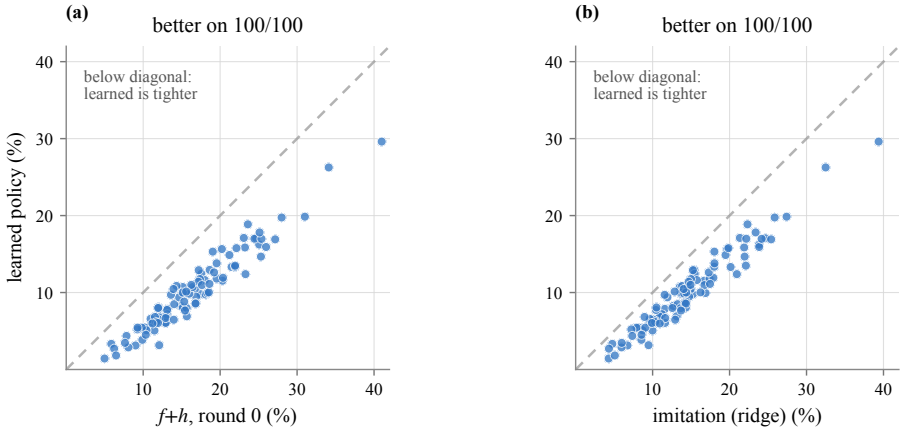


Figure 5: Per-instance paired comparison on uniform $m = 15$, 100 test instances, for the single policy with the lowest mean gap in this cell (one seed, not the seed mean of Table 2). *(a)* Learned policy against round-0 $f{+}h$. *(b)* Learned policy against ridge imitation. Every point lies below the diagonal: the margins of Section 6.4 are not averages over a few large wins.

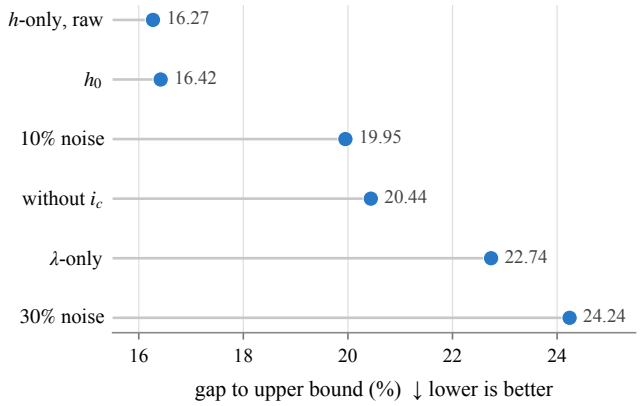


Figure 6: The cost-to-go ablation of Appendix D, uniform $m = 15$, 100 instances. The dashed line is ranking with no cost-to-go; 30% multiplicative noise and the multipliers alone both fall past it.