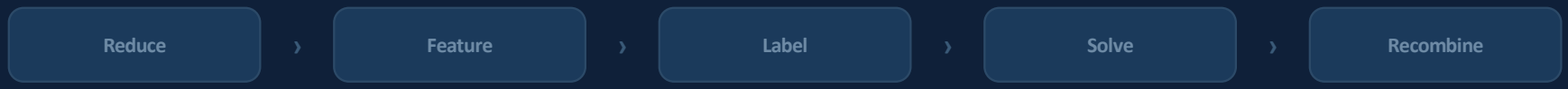
# Learning to Prune for Scalable Combinatorial Optimization

*Leveraging algorithmic knowledge and machine learning together to solve large optimization problem instances*

**Deepak Ajwani**

**Joint work with Noah Coleman, Sourav Dutta, Marco Grassia, Juho Lauri , Ryan O' Connor, Saurabh Ray, Darren Strash, Dena Tayebi and Jiwei Zhang**

ICERM Workshop · Applied & Computational Discrete Algorithms in AI and Data Science



MOTIVATION

# Combinatorial optimization: decades of algorithmic firepower

Combinatorial Optimization: Search for an optimal grouping, ordering or assignment over a discrete, finite set under constraints.

For NP-hard problems, the community has built a deep, hard-won toolbox.

Structural insight, worst-case guarantees, and engineering know-how live inside these methods.

**The premise of this talk: don't discard that toolbox in the era of AI — put it at the centre.**

**ILP / CP / SAT solvers**

**Exact & parameterized algorithms**

**Approximation algorithms**

**Metaheuristics, Heuristics & local search**

MOTIVATION FOR ML

# The same problems are solved again and again

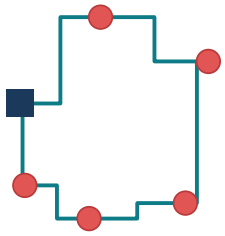
In many applications, one problem is solved repeatedly on instances from the same distribution.

- Logistics & service routing, day after day.
- Nurse and crew rostering, every cycle.

**Natural question: can we learn from historical solutions to solve new instances faster?**

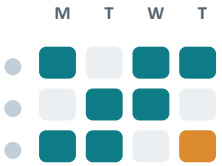
**Alternative question: can we learn from solutions of small instances and solve larger instances?**

Learn the structure of an input distribution, then exploit it.



**Vehicle routing**

*Same depots, shifting demand each day*




**Staff rostering**

*Recurring constraints, fresh week*

MOTIVATION

# Where end-to-end ML hits a wall

End-to-end models (Pointer Nets, attention-RL) learn to emit solutions directly.

- Each training label is itself an NP-hard solve — data is very expensive.
- Deep, opaque architectures: hard to interpret or trust.
- Brittle generalization to larger and out-of-distribution instances.

**Data-hungry**

**Opaque**

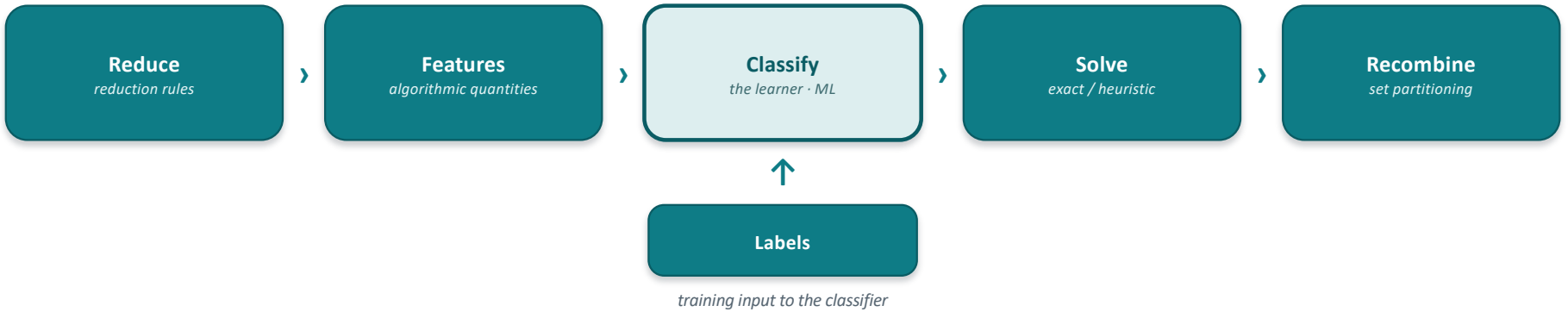
**Brittle**

THE IDEA

# Fold algorithms into ML for solving CO

Given that end-to-end ML had limited success in solving CO — how do we fold decades of algorithmic research *into* the learning loop?

**Answer: Don't learn the whole solution. Prune what's easy, hand the hard core back to algorithms.**




*solid = algorithm* *outlined = the learner (ML)*

AT A GLANCE

# One idea, three frameworks

*Integrate algorithmic knowledge with ML for pruning large instances of CO problems.*

PART 1

## Prune

Algorithmic features rank variables; an exact / MILP solver finishes the pruned core.

PIPELINE STAGES

**Feature + Solve**

EXAMPLES

k-median · set cover · facility location · maximum clique · EV routing

›

PART 2

## Reduce & proxy-label

Reduction rules, MWUA features and heuristic proxy-labels feed the pruner; a SoTA solver closes.

PIPELINE STAGES

**Reduce + Feature + Label + Solve**

EXAMPLES

independent set · vertex-cover variants

›

PART 3

## Decompose & stitch

A heuristic decomposes; ML solves the pieces; a set-partitioning ILP stitches them back.

PIPELINE STAGES

**Decompose + Recombine**

EXAMPLES

capacitated vehicle routing

PART 1 · PRUNE

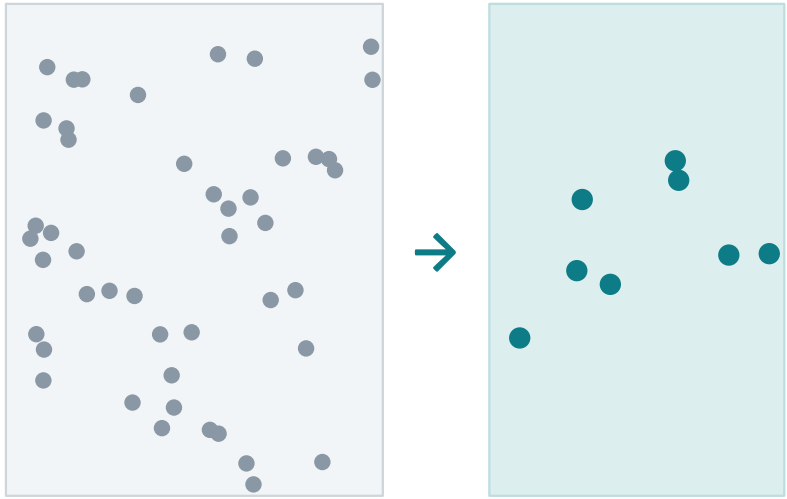
# Reframe: predict and prune, don't predict the whole solution

Subset / binary CO: one indicator variable per element (in the optimum or not).

View pruning as classification — separate elements that are 0 in the optimum solution from those that are 1.

**We don't fix every variable. We confidently prune the easy ones and shrink the instance.**

A small, hard residual is all that remains for an exact solver.



Full instance

**Hard core**

PART 1 · PRUNE

# Two places algorithmic knowledge enters


*solid = algorithm* *outlined = the learner (ML)*

*Algorithms supply the FEATURES that rank variables, and the SOLVER that finishes the core.*

## Algorithms as features

Quantities from approximation algorithms, LP relaxation surrogates and structural measures predict how likely each variable is to be in optimal solution.

## Algorithms as base-case solver

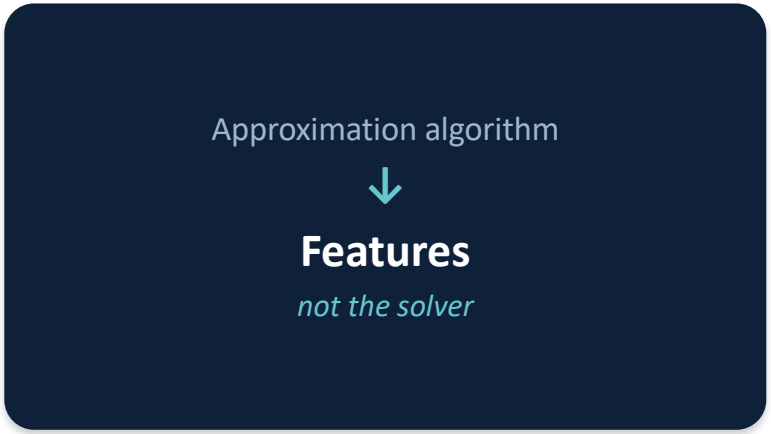
After pruning, the small residual 'hard core' goes to a state-of-the-art exact / MILP solver.

# Worked example: k-median

Quantities from the Charikar–Li k-median approximation algorithm become features.

Add LP-relaxation values and limited-depth branch-and-bound signals.

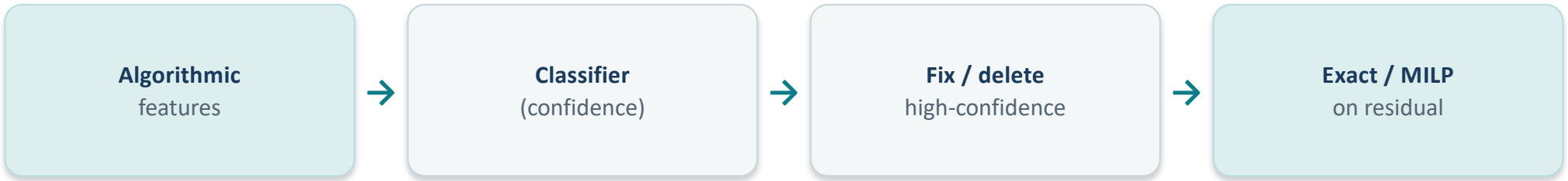
Features transfer to facility location and set cover with little change.




*D. Tayebi, S. Ray, D. Ajwani. Learning to Prune Instances of k-median and Related Problems. ALENEX 2022: 184–194.*

PART 1 · PRUNE

# The overall pipeline




**One knob θ sets how aggressively we prune — tracing a runtime-vs-quality trade-off curve.**

Hard pruning removes elements; soft pruning caps how much of the optimum may come from the pruned set.

Feasibility safeguards: keep a known-algorithm fallback solution, never prune non-zero-LP variables, re-prune with gentler thresholds if the residual is infeasible.

Classifier choice barely matters (RF / SVM / XGBoost); we use Random Forest for interpretable feature importance.

PART 1 · PRUNE

# Maximum clique enumeration

Light structural features: average $\chi^2$-neighbour-degree, local clustering coefficient.

Prune over 99% of nodes on sparse graphs.

Handle graphs ~100× larger than specialized solvers.

Robust across very different graph domains.

**40 min → 7 s**

soc-fb-B-anon graph

**>99%**

nodes pruned

**100×**

larger graphs

| Instance | Max Clique size | # Max Cliques | Pruning |
|---|---|---|---|
| Bio-WormNet-v3 | 121/121 | 18/18 | **0.987** |
| Ia-wiki-user-edits-page | 15/15 | 15/15 | **0.997** |
| Rt-retweet-crawl | 13/13 | 26/26 | **0.997** |
| Soc-digg | 50/50 | 192/192 | **0.998** |
| Soc-flixster | 31/31 | 752/752 | **0.999** |
| Soc-google-plus | 66/66 | 24/24 | **0.998** |
| Soc-lastfm | 14/14 | 324/330 | **0.993** |
| Soc-pokec | 29/29 | 6/6 | **0.975** |
| Soc-themarker | 22/22 | 40/40 | **0.972** |
| Soc-twitter-higgs | 71/71 | 14/14 | **0.986** |
| Soc-wiki-Talk-dir | 26/26 | 141/141 | **0.999** |

*J. Lauri, S. Dutta, M. Grassia, D. Ajwani. Learning fine-grained search space pruning and heuristics for combinatorial optimization. Journal of Heuristics 29(2): 313–347, 2023.*

# Generalization: train on small, test on large

Train on small RK instances; test on much larger ones.

Bootstrapping with local search reaches RK(1000, 200, 100) — where commercial solvers give no feasible solution in 3 hours.

*k-median, train RK(100,10,10) → test RK(200,20,10)*

**Learning to Prune**

**420,988**

in 45 seconds

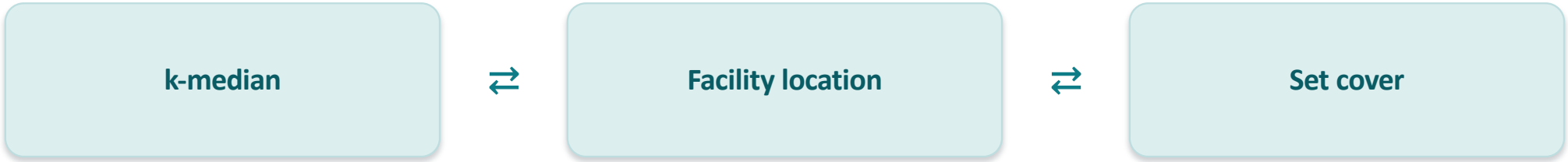
**FICO Xpress-MP**

**441,013**

after many hours

*RK(L, D, t): random k-median instances — L facilities and L clients, each facility linked to ~D clients on average, k = t.*

# The same features transfer across variants




**One feature set, learned on minimal data, lifts all three problems.**

Insight from a well-studied variant carries over to its siblings.

Features derived for metric instances still speed up non-metric instances where the approximation algorithm has no guarantee.

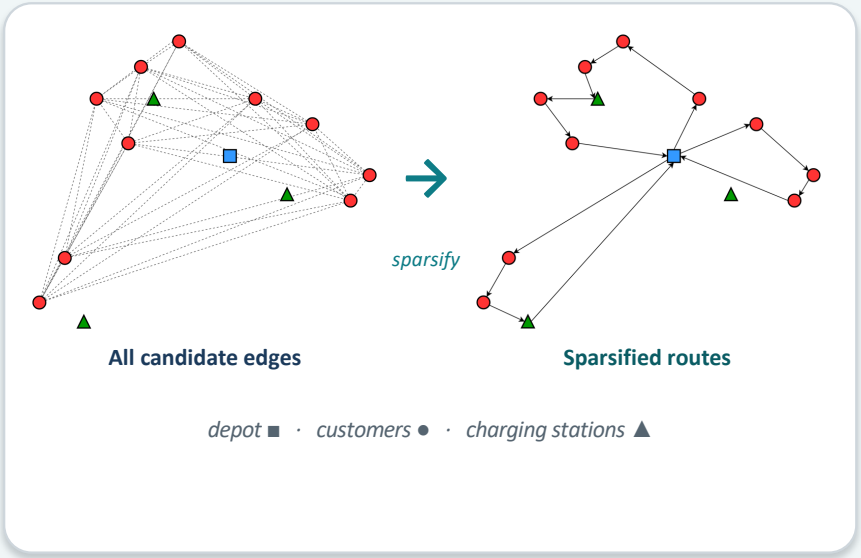
Steiner tree: a model trained on a few Track-1 instances transfers to small-treewidth Track-2 instances it was never told about.

PART 1 → PART 2

# EVRP: pruning when exact labels don't exist

Electric VRP with nonlinear charging: the classifier acts as a variable sparsifier inside an exact branch-and-bound matheuristic.

The twist: exact optimal labels are unobtainable, so we train on pseudo-labels from a heuristic.




*"We give up on exact labels, use algorithm engineering approaches to create labels for training the ML."*

*J. Fitzpatrick, D. Ajwani, P. Carroll. Learning to Prune Electric Vehicle Routing Problems. LION 2023, LNCS 14286: 378–392.*

PART 2 · REDUCE · PRUNE · PROXY-LABEL

# Algorithms now do four stages


*Same template — but algorithms now run the reductions, the features, and the labelling.*

Scaling MIS and vertex-cover variants to graphs with millions of vertices (CPAIOR 2026).

PART 2

# CO problems considered

Maximum Independent Set (MIS) and Minimum Vertex Cover (MVC) on million-vertex graphs.

k-path independent set — denser, far less reducible at k = 2.

> Largest set of vertices that are pairwise more than two hops apart

3-path vertex cover ($VCP_3$) — the natural hitting-set formulation, our focus.

> The minimum set of vertices that hits every three-vertex path

*Residual solvers*

**ReduMIS**
reductions + local search + evolutionary

**PACE 2025 winner**
hitting-set solver for $VCP_3$

*R. O'Connor, N. Coleman, D. Strash, S. Ray, D. Ajwani. A Scalable Learning Approach for Efficient Computation of Independent Set and Cover Variants. CPAIOR 2026.*

PART 2

# The scaling wall — and three algorithmic responses

### Push scalability further

Pruning alone leaves too much.

**Compose with reduction rules; target the irreducible core.**

Reduce

### LP doesn't scale

LP relaxations blow the budget on million-vertex graphs.

**Use MWUA as an LP surrogate.**

Feature

### No exact labels

Exact MIS is hopeless beyond a few thousand vertices.

**Train on heuristic (ReduMIS) proxy labels.**

Label

Reduce › Feature › Label › Solve

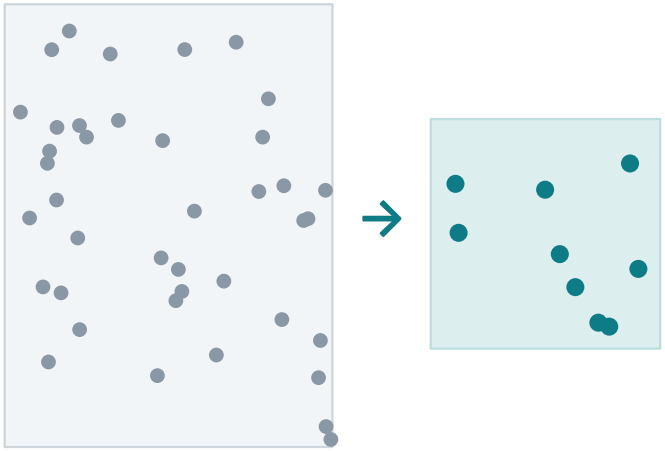
PART 2 · STAGE = REDUCE

# Reduction rules: algorithm engineering as presolve

Data-reduction rules provably preserve optimal solutions — the analogue of presolving in MILP/CP.

They shrink the instance before any learning happens.

**The classifier then faces only the irreducible core — exactly where prediction is worth the effort.**



PART 2 · STAGE = FEATURE

# MWUA: an algorithm becomes the feature

Plotkin–Shmoys–Tardos view: constraints are objects, violation is reward, weights track constraint hardness.

Violated constraints gain weight; satisfied ones fade — effort focuses on the hardest parts.

The averaged iterate is approximately feasible after T rounds.

**Yields an approximate fractional solution — the strongest predictive feature.**

update rule

$$w_i \leftarrow w_i \cdot (1 + \eta \cdot \ell_i)$$

*$\ell_i$ = violation of constraint $i$*

**~90% cheaper**

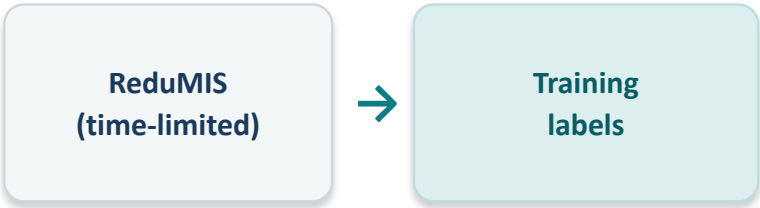
than solving the LP relaxation

PART 2 · STAGE = LABEL

# Proxy labels instead of an exact solver

Exact MIS labels are out of reach past a few thousand vertices.

Use time-limited ReduMIS / heuristic outputs as noisy-but-useful labels.



PART 2 · FEATURES

# Per-vertex features: cheap signals at million-vertex scale

**Local structure**

- Local clustering coefficient
- Core / k-shell number
- Luby maximal-IS membership frequency

**Degree & centrality**

- Degree centrality and normalized rank
- Min / max / avg neighbour-degree rank
- PageRank score

**MWUA as LP surrogate**

- Approximate fractional value
- Min / max / avg incident constraint weight

**Deliberately excluded**

Betweenness centrality
GNN embeddings
*— too expensive at this scale*

PART 2 · STAGE = SOLVE

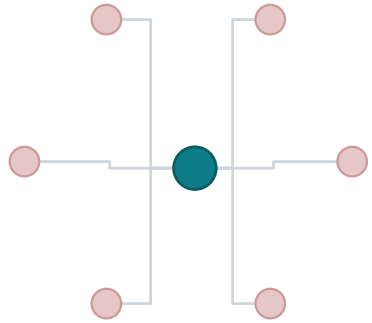
# Pruning mechanics, then the residual solver

Sort vertices by classifier confidence; fix the most certain.

MIS: predicted-IN keeps the vertex and deletes its neighbours; predicted-OUT removes it.

Operate directly on the graph — the residual is smaller and sparser — then call ReduMIS.

$VCP_3$ handled at the constraint level; empty 3-paths force their least-confident vertex in.



Keep IN-vertex, delete neighbours

PART 2 · RESULTS

# Where the method earns its keep

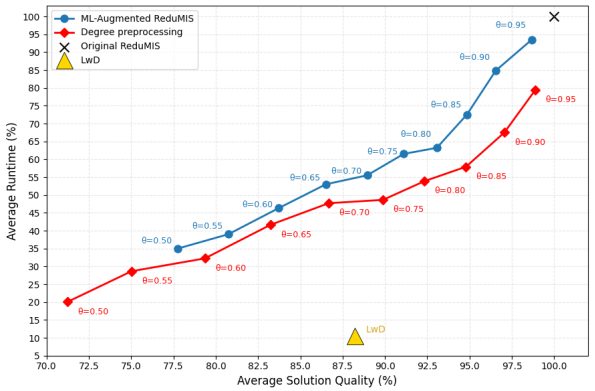



**Fig. 1 — MIS (k=1)**

Degree pruning competitive on easy inputs

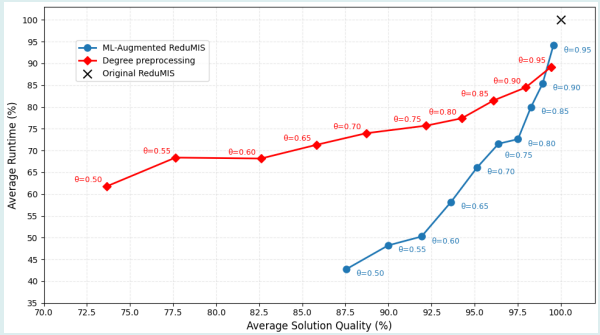



**Fig. 2 — MIS (k=2, dense)**

ML converges faster for equal quality

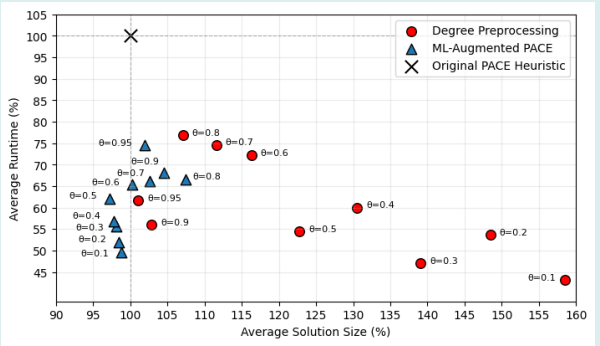



**Fig. 3 — $VCP_3$**

Smaller covers AND lower runtime vs PACE

**MWUA with ML is particularly useful in the hard, dense, non-reducible regime.**

TAKEAWAYS

# The main advantages

### Interpretable

Few-parameter models; feature importance you can read.

### Data-thrifty

A handful of training instances suffices.

### Generalizes

Robust to larger and out-of-distribution sizes.

### Principled

Features are algorithmic quantities, not black-box signals.

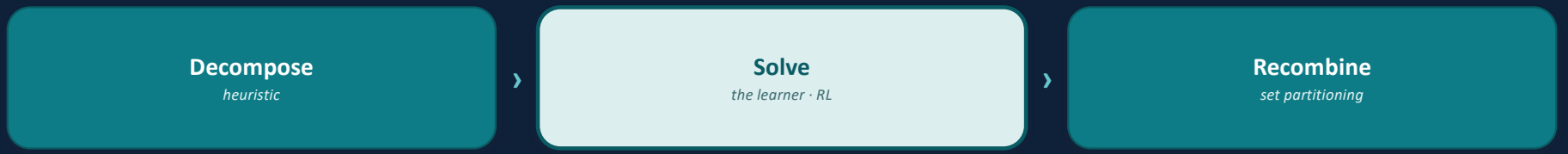
Significant gains in scalability at some cost of solution quality

Bonus: concentrated feature importance becomes new heuristics — avg $\chi^2$-neighbour-degree gave a max-clique heuristic that outperforms SoTA on dense graphs.

*S. Dutta, J. Lauri. Finding a Maximum Clique in Dense Graphs via $\chi^2$ Statistics. CIKM 2019: 2421–2424.*

PART 3 · DECOMPOSE & STITCH

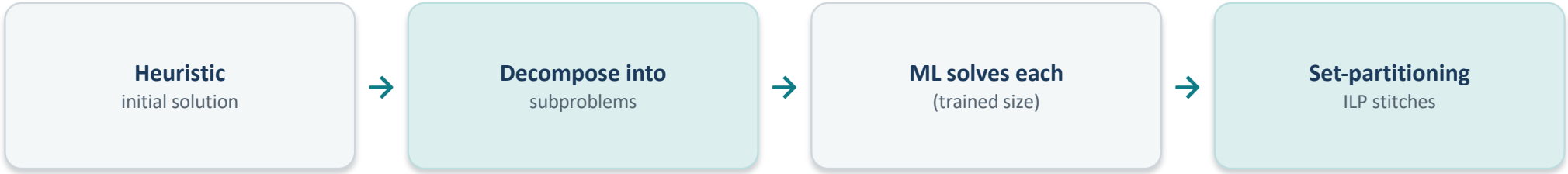
# Algorithms wrap the learner




*A heuristic decomposes, ML solves the pieces, and an optimization model stitches them back*

PART 3 · DECOMPOSE & STITCH

# Scaling CVRP by decompose-and-stitch




End-to-end RL technique for CVRP didn't generalize well to larger size instances.

Use a heuristic initial solution to decompose the large instances into smaller subproblems.

Subproblems are sized to where the RL heuristic is reliable — it always works in its comfort zone.

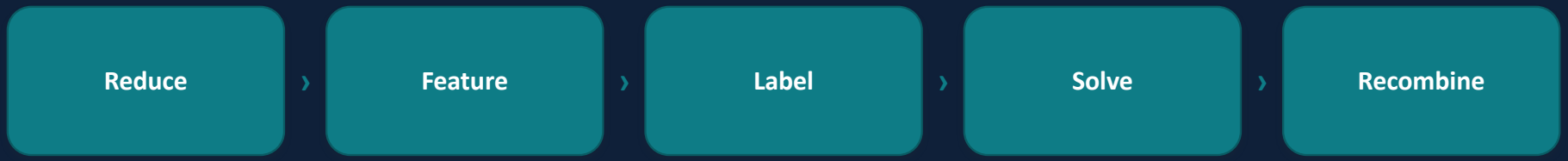
The set-partitioning ILP recombines candidate routes and enforces fleet-size constraints.

**< 3% gap**
to best-known; generalizes to green VRP

*J. Fitzpatrick, D. Ajwani, P. Carroll. A scalable learning approach for the capacitated vehicle routing problem. Computers & Operations Research 171 (2024) 106787.*

SYNTHESIS

# Integrating algorithmic insights




Algorithmic knowledge can be leveraged for different stages — **reducing, featurizing, labelling, solving, recombining.**

OPEN DIRECTION · TOWARD OPTIMALITY

# From near-optimal to provably optimal

Every framework so far returns close-to-optimal solutions — but none certifies optimality.

Pruning, MWUA surrogates and decomposition are heuristic at the core: a quality–tractability trade-off.

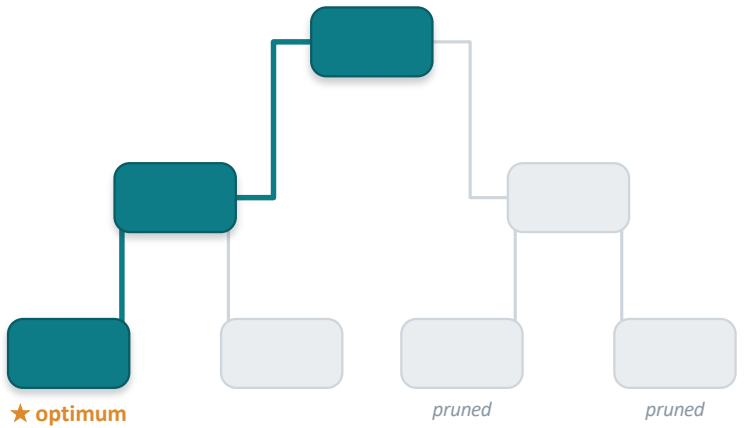
**But the same per-variable features can do more than prune — they can steer an exact search.**

**Learn a variable / branching order that guides an exact search to reach and prove the optimum faster.**

**Connecting to the previous talk**

Langedal & Manne use graph neural networks as ordering heuristics for parallel graph coloring — a learned vertex ordering driving a classical graph algorithm, the same idea applied here to branching.

*K. Langedal, F. Manne. Graph Neural Networks as Ordering Heuristics for Parallel Graph Coloring. ALENEX 2025: 56–67.*




*A feature-driven branching order explores the promising branch first — reaching and certifying the optimum with far fewer explored nodes.*

FINAL THOUGHTS

## The durable recipe is deep integration of algorithm engineering and ML — not bigger black boxes.

**Don’t learn the solution — learn what to prune.** ML fixes the easy variables; algorithms finish the hard core.

**Algorithmic knowledge enters wherever ML is unreliable** — features, reductions, proxy-labels, solver — staying interpretable and data-thrifty.

**Drastic runtime cuts at near-optimal quality** — strongest on hard, dense, least-reducible inputs; an honest quality–tractability trade-off.

**Trains small, generalizes large** — transfers across problem variants and distributions from a handful of instances.

*Potential: Able to feed new algorithms back to this community*

# Thank you · Questions?