

Ashera; Neural Guided Optimization Modulo Theory

*Justin Wong
Pei-Wei Chen
Tianjun Zhang
Joseph Gonzalez
Yuandong Tian
Sanjit A. Seshia*

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2023-103

<http://www2.eecs.berkeley.edu/Pubs/TechRpts/2023/EECS-2023-103.html>

May 11, 2023



Copyright © 2023, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Acknowledgement

This research is supported in part by NSF CISE Expeditions Award CCF-1730628, NSF NRI \#2024675 and under the NSF AI4OPT Center. UC Berkeley research is also supported by gifts from Alibaba, Amazon Web Services, Ant Financial, CapitalOne, Ericsson, Meta, Futurewei, Google, Intel, Microsoft, Nvidia, Sco tiabank, Splunk and VMware.

Ashera: Neural Optimization Modulo Theory

Justin Wong^{*1}, Pei-Wei Chen^{*1}, Tianjun Zhang¹,
Joseph E. Gonzalez¹, Yuandong Tian², Sanjit A. Seshia¹

¹University of California, Berkeley ²Meta AI Research

Abstract

1 Applications of Satisfiability Modulo Theories (SMT) within
2 design automation and software/hardware verification often
3 require finding models whose quantitative cost objective is
4 guaranteed to be optimal. As an example, in worst-case ex-
5 ecution time analysis, it does not suffice to simply discover
6 a feasible execution trace; we are instead interested in prov-
7 ing properties on the longest execution trace. Such problems
8 can be formulated as Optimization Modulo Theory (OMT),
9 and solving them is much more challenging than both SMT
10 problems and unconstrained optimization. Current solutions
11 struggle to scale to problems of large size, because they
12 require experts to tune solvers and carefully craft problem
13 encodings. This approach is not only problem-specific but
14 also requires manual effort. Recent progress in neural tech-
15 niques have been successfully applied to Mixed Integer Lin-
16 ear Programming (MILP) and certain instances of the Travel-
17 ing Salesman Problem (TSP). We make the case for learning-
18 based solvers in OMT and present Ashera, a neural-guided
19 OMT solver. Ashera innovates on prior art by introducing
20 Logical Neighborhood Search and neural-based warm start-
21 ing. Additionally, we introduce new benchmarks for learning-
22 based OMT techniques, targeted at real-world applications
23 including scheduling and multi-agent TSP. Ashera exhibits as
24 much as a 3x speedup and shows improved scaling compared
25 to MILP approximation as used in industry and state-of-the-
26 art OMT solvers.

1 Introduction

28 Analysis of worst-case execution time (WCET) of soft-
29 ware and hardware requires not only identifying valid ex-
30 ecutions of a program but also finding a provably slowest
31 execution. Existing solvers for Satisfiability Modulo Theo-
32 ries (SMT) have found remarkable practical success in pro-
33 viding guarantees on previously feasibility problems in do-
34 mains such as software and hardware verification (Dutertre
35 and De Moura 2006; Moura and Bjørner 2008; Leino 2010;
36 Katz et al. 2017). However, there has been limited progress
37 on Optimization Modulo Theories (OMT) problems like
38 WCET (Henry et al. 2014) that require provably cost-
39 optimal solutions not just feasible solutions. In these sce-
40 narios, ranging from software security (Bertolissi, Dos San-
41 tos, and Ranise 2018; Henry et al. 2014) to scheduling and

planning (Leofante et al. 2017; Kovásznai, Biró, and Erdélyi
2017), require OMT to support an objective function.

While OMT is a more general optimization framework,
solving it becomes more difficult. There are two major chal-
lenges. First, existing approaches to OMT either use the Big
M approximation to reduce the problem to Integer Linear
Programming ILP (Gurobi Optimization, LLC 2021) or do
iterative calls to SMT (Sebastiani and Trentin 2015; Bjørner,
Phan, and Fleckenstein 2015), which scales poorly. Second,
the solvers are designed to be general purpose and problem
agnostic. To address poor scalability in a per application ba-
sis, experts manually tune hyperparameters and problem en-
codings which often improves the solving time from days to
a matter of minutes. Such solutions, while feasible, are often
expensive and time-consuming, and ignore the past experi-
ence of solved problems.

In this paper, we present a codesigned neural OMT solver,
called *Ashera* to address these two challenges by introducing
two components: *an OMT engine* and *a neural diver*.

First, our OMT engine in Ashera iterates between calling
an SMT solver and an optimality-aware theory solver, TS^* .
Because we use SMT to decompose disjunctions into guided
search over conjunctive cases, we can avoid using Big M. In
SMT, theory solvers (TS) are domain specific subsolvers for
a conjunction of constraints. For instance, for LIA theory
specific properties to prune and speed up the enumeration
such as $x > 0 \wedge x < 1$ indicates no integer x is feasible. We
introduce *optimality-aware theory solver*, TS^* , which addi-
tionally has awareness of the optimization objective. More
concretely for Linear Integer Arithmetic (LIA), we use Z3
for quantifier-free LIA SMT and Gurobi for ILP.

Second, our neural diver learns to directly predict promis-
ing partial assignments that lead to a better cost when ex-
tended to full assignments. We do this by training a Graph
Neural Network (GNN) on existing solutions of related
problems, extracting the knowledge from past experience.
Then, the same SMT solver used in our OMT engine is
used to check the validity of these candidate assignments
to guarantee soundness. Our approach is inspired by the re-
cent success of (Nair et al. 2020) that applies similar idea to
Mixed Integer Linear Programming (MILP). The efficiency
of existing solvers can be substantially improved by trans-
ferring knowledge of solving one problem instance to an-
other similar one. We observe that in practice, optimiza-

^{*}These authors contributed equally.

tions are done repeatedly on similar instances drawn from an underlying distribution, for example on a monthly basis for network planning (Zhu et al. 2021). As such, the problems to be solved regularly differ only slightly, but the optimization procedure needs to be redone, as modified constraints change the solution space. As a result, learning from previous solved problems can speed up the next optimization without any manual human input. Our approach differs from existing work that learn end-to-end a solver to replace existing systems (e.g., learned query optimization (Yang et al. 2022) and chip placement (Mirhoseini et al. 2020)). In contrast, we co-design our OMT approach to contribute a new generalized solver while exploiting opportunities for learning-based components.

As existing benchmarks provide dozens of diverse OMT problems, we build a benchmark for evaluating learning-based OMT solvers. Our benchmark provides over 58,000 OMT problems from two families of problems: task scheduling and multi-agent Traveling Salesman Problem (TSP). Using this benchmark we demonstrate that Ashera can learn on problems with less variables and constraints and generalize to larger problems within the same problem class.

Our work makes the following contributions:

- A benchmark for evaluating learning-based OMT solving with problem families in scheduling and multi-agent traveling salesman problems.
- To our knowledge, we present the first learning-based OMT solver in Ashera. The solver focuses on support for Linear Integer Arithmetic (LIA), but the framework generalizes to other theories.
- For scheduling, Ashera in contrast is **3x faster** and solves **three more problems** than the widely-used commercial solver, Gurobi, on problems with 10 and 11 tasks, respectively. OptiMathSAT and Z3 are unable to solve any problems of 11 tasks within 1 hour timeout.
- Ashera is **18% faster** than OptiMathSAT on multiagent TSP with 15 waypoints, where Gurobi and Z3 timeout.

2 Related works

2.1 Solvers for OMT

The last decade has produced two prominent Optimization Modulo Theory (OMT) solvers: 1) OptiMathSAT (Sebastiani and Trentin 2015), which applies a binary search approach to discovering the optimal solution, and 2) vZ (Bjørner, Phan, and Fleckenstein 2015), which iteratively uses SMT to find a strictly better feasible solution and locally improve it by coordinate-wise search for strictly improving boundary solutions. OptiMathSAT further uses sorting networks to decouple the Boolean reasoning from the arithmetic solving. vZ exploits carefully engineered MaxSMT and pseudo Boolean solvers to provide competitive performance when the OMT problem lies within these domains. Neither of these works perform well at scale and applications are largely dominated by ILP approximations.

Similar to OMT, Inez (Manolios, Pais, and Papavasileiou 2015), a solver for Mathematical Programming Modulo

Theory, integrates ILP solvers with solvers for first order theories. However, Inez relies on the Big M encoding or built in constraint handlers in ILP solvers to reason about disjunctions and instead focus on extending ILP solvers to support uninterpreted functions and support for user-provided axioms. Ashera explicitly reasons about disjunctions arising from the logic structure of OMT problems.

2.2 Neural Guidance for Combinatorial Optimization

The Integer Linear Program (ILP) is a well-studied class of optimization problems in large part due to its prominence in operations research, computer vision, scheduling, and other domains. Branch and bound, one particular tree search algorithm, is the best known approach for solving ILPs. Branch and bound is able to incrementally establish a lower and upper bound via a feasible point and a solution to a linear relaxation of the ILP respectively until tightness is achieved.

In (Balcan et al. 2018), the authors propose to replace empirical heuristics for selecting variables to branch with a data-driven methodology that achieves provable complexity bounds. The authors show that they can learn a convex combination of scoring rules (which each determine the ordering of node branching) that is nearly optimal in expectation over a distribution of original ILPs. Furthermore, the optimization process over scoring rules can be seen as performing empirical risk minimization (ERM) of the original optimization objective subject to the input variable constraints.

Another approach taken by Wu et al. (Wu et al. 2021a) is to train a model to reconstruct locally optimal solutions. This model is trained by taking feasible solutions and resolving the optimization with a random subset of variables masked out. The model learns how to improve solutions locally and recognize strategies that generalize to adjacent regions in the feasible set. However, this approach fails to recognize that due to combinatorial explosion subproblem optimization is often negligible and the real challenge is identifying and proving optimal the global optima not local optima.

The approach taken by Nair et al. (Nair et al. 2020) extends Balcan et al. (Balcan et al. 2018)’s approach with a neural diver, which takes the bipartite graph representation of variables and constraints to predict plausible partial assignments. These partial assignments can then be explored in parallel by instantiating Mixed Integer Linear Programming (MILP) instances over a smaller variable space. Our neural diver extends Nair et al. to support OMT problems.

2.3 Neural Guidance for Combinatorial Applications

Using neural networks to solve TSP (Bello et al. 2016) has been extensively studied dating back to the development of Hopfield neural networks in 1985 (Hopfield and Tank 2004). More recent efforts such as Selsam et al. (Selsam et al. 2019) have attempted to learn end-to-end models for SAT solvers.

Alternatively, a presolve phase is employed before solving to explore promising regions. This approach is exemplified by NeuroPlan (Zhu et al. 2021), which applies neural guidance to large-scale network planning problem. These

198 problems often takes days or weeks for integer linear pro- 249
 199 gramming (ILP) to find even a feasible solution. For this, 250
 200 NeuroPlan learns an RL agent to predict a good initial so- 251
 201 lution to large-scale network planning problem modeled as 252
 202 ILP. The RL agent constructs the solution progressively by 253
 203 picking which network connection to be used to increase the 254
 204 capacity between two nodes to satisfy the communication 255
 205 requirements, and verify the feasibility via efficient checking 256
 206 techniques, reusing previous computational results. Once 257
 207 the initial solution is found, a follow-up ILP solving be- 258
 208 comes much faster.

209 3 Preliminaries

210 In this section, we provide some background on OMT and 261
 211 graph neural network architectures used in Ashera.

212 3.1 Optimization Modulo Theories (OMT)

213 Optimization Modulo Theories (OMT) extends SMT, guar- 262
 214 anteeing that an optimal feasible solution is returned. In ad- 263
 215 dition to the satisfaction formula, an OMT problem includes 264
 216 an objective function C which maps assignments to a total 265
 217 ordered set. When the domain of C is real or floating point, 266
 218 a tolerance δ must be specified.

219 A Satisfiability Modulo Theory (SMT) problem decides 267
 220 the satisfiability of a first-order formula within a theory (Bar- 268
 221 rett et al. 2009). We focus on the theory of Linear Inte- 269
 222 ger Arithmetic (LIA), but note that SMT and OMT extend 270
 223 to other theories, for instance, arrays and strings. For LIA, 271
 224 consider *atomic formulas* as linear inequalities of the form: 272
 225 $atom_i \triangleq \vec{a}_i \cdot \vec{x} \bowtie b_i$ where $\bowtie \triangleq \{<, \leq, >, \geq, =\}$. These 273
 226 atoms may differ when considering different theories, and 274
 227 we denote a theory *literal* as an atomic formula or the nega-
 228 tion of one.

We build up *clauses* and subsequently *formulas* in Con-
 junctive Normal Form (CNF) from these atoms as

$$clause_j \triangleq \bigvee_i literal_i \quad formula \triangleq \bigwedge_j clause_j.$$

229 With the standard interpretation of atoms, a model or assign- 275
 230 ment that satisfies a formula is one where the evaluation of 276
 231 the formula is True. We denote partial assignments as α and 277
 232 full assignments as \mathcal{A} .

233 We define a *Boolean backbone* \mathcal{B} as the set of literals 278
 234 where the polarity of each literal depends on the truth value 279
 235 of the corresponding atom when applying a full assignment 280
 236 \mathcal{A} , i.e. a literal is in positive polarity if the corresponding 281
 237 atom evaluates to true. A Boolean (backbone) assignment is 282
 238 the truth value assignment to all literals in the formula. Note 283
 239 that an assignment uniquely specifies a Boolean backbone 284
 240 but multiple assignments can share a common backbone.

241 *Lazy SMT* solves with a two stage iterative process. First, 285
 242 a SAT solver identifies a candidate Boolean assignment, 286
 243 viewing clauses as simple Boolean functions. Then, once a 287
 244 Boolean assignment is chosen, the formula can be expressed 288
 245 as a conjunct of atomic theory constraints. As such, the con- 289
 246 straints can then be passed along to a specialized theory 290
 247 solver that identifies a feasible solution that satisfies the con-
 248 junct of constraints. If this is not feasible, the solver picks

another Boolean assignment factoring in the learned con-
 flict. In some sense, this can be viewed as a two level search
 problem where Boolean backbone identifies a *logical neigh-*
borhood for the theory solver to search in.

Even though SMT is designed to efficiently explore dis-
 junctive logic structures exploiting structure and symmetry
 in the encoding, the satisfiability task only requires reason-
 ing about feasibility. In contrast, an OMT solver must search
 for other solutions once after identifying a feasible solution
 potentially with differing Boolean backbone.

In the notation as introduced for SMT, we solve problems
 of minimizing cost, $C(\vec{x})$ such that the CNF formula holds,
 where in the theory of LIA variables are constrained to be
 integral and $C(\vec{x})$ a linear function with integral coefficients.

OMT raises the specific challenges of both explicitly rea-
 soning about disjunctions and optimizing a cost function.
 Particularly, disjunctions lead to local optimas which may
 not be connected since there are no convexity guarantees.

267 3.2 Integer Linear Programming (ILP)

268 A Integer Linear Programming (ILP) problem is parameter- 269
 269 ized by the tuple $(\mathbf{A}, \vec{b}, \vec{c})$. The objective is to solve for an 270
 270 optimal choice of \vec{x} such that $\vec{c} \cdot \vec{x}$ is minimized. However, 271
 271 \vec{x} is constrained to satisfy $\mathbf{A}\vec{x} \leq \vec{b}$, where \leq represents el- 272
 272 ement wise less than or equal to. Further, these vectors can 273
 273 be over a mixture of real numbers and integers. We note that 274
 274 strict inequalities $\vec{a}_i \cdot \vec{x} < b_i$ can be encoded as $\vec{a}_i \cdot \vec{x} \leq b_i - 1$.

Although MILP does not explicitly support disjunctions,
 Big M encoding can allow practitioners to implicitly approx-
 imate disjunctions by adding an additional decision variable.
 Disjunctions of inequalities that appear in the original en-
 coding, $(LHS_1 \leq RHS_1) \vee (LHS_2 \leq RHS_2)$ can be encoded
 by adding the decision binary variable α . Since MILP must
 be expressed as a list of constraints that always hold, a large
 constant M is then added to the inequality to trivialize the
 constraints when α deselects the literal. For our example,
 the disjunction becomes encoded as the following two con-
 straints:

$$\begin{aligned} LHS_1 &\leq RHS_1 + \alpha M \\ LHS_2 &\leq RHS_2 + (1 - \alpha)M \end{aligned}$$

275 By using this encoding strategy, the assignment of α results 276
 276 in the selection of which clause must hold.

277 3.3 Graph Neural Network (GNN)

278 Recent progress in machine learning for optimization prob- 279
 279 lems have been enabled by graph neural networks. In this 280
 280 section, we provide a brief introduction and key intuition be- 281
 281 hind these models. Interested readers can refer to (Wu et al. 282
 282 2021b) for more details.

283 Graph Neural Networks (GNNs) are deep neural networks 284
 284 that take graph structured inputs and make predictions on 285
 285 both individual nodes or edges and the entire graph. As for- 286
 286 malized by (Gilmer et al. 2017), these models output a graph 287
 287 or node representation (i.e., a high-dimensional vector), via 288
 288 message passing over the graph structure. The GNN takes 289
 289 node features x_v and applies i rounds of message passing 290
 290 where the hidden state h_v^i of each node is updated based a

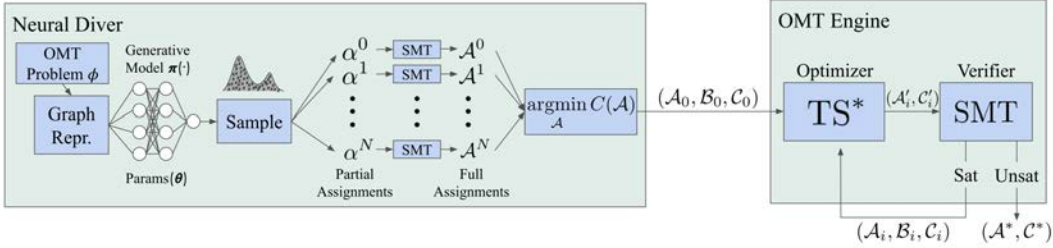


Figure 1: Ashera workflow. The neural diver serves as a warm-starter providing an initial low cost feasible solution. After neural diving, the OMT engine in Ashera alternates between a *Optimization-aware Theory Solver* (TS*) for optimization and an SMT solver for verification. Each invocation of TS* returns a tighter blocking clause, restricting the SMT solver to search for strictly lower cost solutions than the current model.

291 learned function parameterized by θ on its neighbors' hid- 322
 292 den states and the edge features: $m_v^{t+1} f_\theta(h_w^i, e_{vw})$ where 323
 293 w is a neighbor in the neighborhood of v , $N(v)$. The new 324
 294 hidden state is then $h_v^{i+1} = \text{Agg}(h_v^i, \sum_{w \in N(v)} m_w^{t+1})$.

295 Inspired by Convolutional Neural Networks (CNNs), 326
 296 which exploit the inductive bias that neighboring pixels are 327
 297 often related, Graph Convolutional Networks, or GCN (Kipf 328
 298 and Welling 2016), employ the same learned function across 329
 299 the same layer of the neural network irrespective of nodes. 330
 300 Analogous to computer vision, the shared function encour- 331
 301 ages the network to learn to recognize the same pattern oc- 332
 302 ccurring in connected subgraphs. This approach has seen wide 333
 303 success from analyzing social media graphs (Fan et al. 2022) 334
 304 to predicting molecule properties (Gilmer et al. 2017) and 335
 305 within combinatorial optimization has been used for net- 336
 306 work planning (Zhu et al. 2021) and chip placement (Mirho- 337
 307 seini et al. 2020).

308 **Graph encoding for constrained programming.** As done 329
 309 in (Gasse et al. 2019) and Nair et al. (Nair et al. 2020), one 330
 310 common graph representation for constrained programming 331
 311 is a graph, in which nodes represent constraints and vari- 332
 312 ables. Edge between constraint nodes and variable nodes 333
 313 encode the coefficient of variables that appear in the con- 334
 314 straint. Then, the message passing from variable to con- 335
 315 straint in the graph neural network can be interpreted as how 336
 316 the variable embedding influence the constraint embedding 337
 317 (and vice versa). In practice, 2-3 rounds of message passing 338
 318 suffice to model higher-order constraints, e.g., two variables 339
 319 in the same constraint, etc.

320 4 Method Overview

321 In this section, we provide an overview of our method, illus- 322
 322 trated in Fig.1. Ashera consists of two main components: an 323
 323 OMT engine (Section 5) and a neural diver (Section 6).

324 The neural diver generates an initial feasible assignments 325
 325 \mathcal{A}_0 by first using a neural heuristic trained on solutions 326
 326 of similar OMT problems. Given a \mathcal{A}_0 , the assignment 327
 327 uniquely specifies a cost C_0 and a boolean backbone \mathcal{B}_0 . The 328
 328 initial feasible solution provides a warm start to the OMT 329
 329 engine. We elaborate on how the neural diver is trained and 330
 330 how it selects initial feasible assignment in Section 6.

331 Given the tuple $(\mathcal{A}_0, \mathcal{B}_0, C_0)$, the OMT engine generates a

332 blocking clause: $C(x) < C_0$. This restricts the engine to only 333
 333 search for lower cost assignments. Using the boolean back- 334
 334 bone, our OMT engine iterates between an optimizer that 335
 335 searches for a lower cost solution than the current assign- 336
 336 ment \mathcal{A}_i , and a verifier that checks if the optimized assign- 337
 337 ment \mathcal{A}'_i is optimal. In a counterexample-guided fashion, the 338
 338 optimizer utilizes feasible solutions returned by the verifier 339
 339 to refine the search. We elaborate on the optimizer and ver- 340
 340 ifier in Section 5. When the verifier returns unsat, Ashera 341
 341 returns the best assignment, \mathcal{A}^* , and best cost, C^* .

342 Ashera can be run in a *cold-start* settings when solutions 343
 343 to similar OMT problems are not available, or as a way to 344
 344 solve related OMT problems that are then used for training. 345
 345 In this setting, the neural diver can be replaced by a SMT 346
 346 solver. Although Ashera will not be able to learn from past 347
 347 examples, the SMT solver will still provide a valid albeit 348
 348 likely high cost assignment to the OMT engine.

349 5 OMT Engine

350 For exposition purposes, we first detail the cold-start setting 351
 351 of our OMT algorithm in this section, in which the neural 352
 352 diver is substituted with an SMT solver. In Section 6, we in- 353
 353 troduce the full Ashera algorithm, including the neural diver 354
 354 and introduce Logical Neighborhood Search.

355 5.1 Logical Neighborhood Search

356 Efforts like Wu et al. (Wu et al. 2021a) approach opti- 357
 357 mization problems as a large neighborhood search problem 358
 358 where the optimizer first discovers feasible solutions. Then, 359
 359 it explores assignments which differ from the feasible as- 360
 360 signment by a ϵ -ball. Our approach identifies a more natural 361
 361 notion of logical locality for OMT.

362 After obtaining the tuple $(\mathcal{A}_i, \mathcal{B}_i, C_i)$ from the SMT 363
 363 solver, we perform optimization with an optimality-aware 364
 364 theory solver TS* which takes in a Boolean backbone \mathcal{B} as 365
 365 input. Unlike existing approaches to neighborhood search, 366
 366 we aim to search for solutions that have similar logic struc- 367
 367 ture (i.e. preserves the same Boolean literal backbone). This 368
 368 definition of locality allows us to use an off-the-shelf ILP 369
 369 solver as the optimality-aware theory solver to conduct the 370
 370 neighborhood search.

Algorithm 1: Ashera: OMT Solver

Input: ϕ : OMT formula
Returns: $\mathcal{A}^*, \mathcal{C}^*$: best assignment and cost w.r.t. objective

- 1: $\mathcal{A}^*, \mathcal{C}^* := \emptyset, \infty$
- 2: $partialAssignSamples := neuralDiver(\phi)$
- 3: $SMT_problem := createSMT(\phi)$
- 4: **for all** α **in** $partialAssignSamples$ **do**
- 5: $isSat, (\mathcal{A}, \mathcal{B}, \mathcal{C}) := solve(SMT_problem, \alpha)$
- 6: **if** $\mathcal{C} < \mathcal{C}^*$ **and** $isSat$ **then**
- 7: $\mathcal{A}^*, \mathcal{C}^* := \mathcal{A}, \mathcal{C}$
- 8: **while** **True** **do**
- 9: $blocker := (cost \leq \mathcal{C}^* - \delta)$
- 10: $isSat, \mathcal{A}, \mathcal{B}, \mathcal{C} := solve(SMT_problem, blocker)$
- 11: **if not** $isSat$ **then**
- 12: **break**
- 13: $ILP_problem = createILP(\mathcal{B})$
- 14: $\mathcal{A}^*, \mathcal{C}^* := solve(ILP_problem)$

return $\mathcal{A}^*, \mathcal{C}^*$

371 For each literal in the Boolean backbone \mathcal{B}_i , we add the
372 negation of the false literal as an ILP constraint and the lit-
373 eral itself for true literals in the backbone. In this way, we do
374 not need to encode disjunctions into the ILP encoding using
375 Big M or convex hull. In section A.5, we discuss some sound
376 optimizations that can be done with this constraint genera-
377 tion process. By restricting to the convex region around the
378 feasible solution, we can express the optimization purely as
379 the conjunction of literals in \mathcal{B}_i . As we have identified a fea-
380 sible solution \mathcal{A}_i , we effectively use ILP to improve on the
381 found solution within the neighborhood which maintains the
382 same logic assignment. Note that the search is localized to a
383 connected region specified by the constraints, in which at
384 least one feasible solution can be found (i.e., the current
385 solution \mathcal{A}_i). However, unlike a LIA theory solver, ILP is
386 cost-aware and able to optimize with respect to our objective
387 function. The ILP solver produces a tuple $(\mathcal{A}'_i, \mathcal{C}'_i)$ as output,
388 where \mathcal{A}'_i is an assignment that achieves the optimized cost
389 \mathcal{C}'_i within the neighborhood.

390 To reach another disconnected region, we query SMT
391 with the optimized cost \mathcal{C}'_i for a feasible solution that's
392 strictly better than the solution \mathcal{A}'_i discovered by the last it-
393 eration of ILP. If this results in an unsatisfiable result, we
394 terminate knowing there exists no better feasible solution to
395 the OMT problem. Given ILP discovered the optimal solu-
396 tion for the particular logic backbone, the SMT solver will
397 find a feasible solution with a different logic backbone.

398 We present our algorithm in full in Algorithm 1 noting
399 that a tolerance, δ , can be set depending on the user's domain
400 expertise. For our integer example, $\delta = 1$ is natural.

6 Neural Diving

402 Inspired by work by Nair et al. (Nair et al. 2020), we adopt a
403 neural diver which can be thought of as a warm-starter that
404 identifies promising initial feasible solutions.

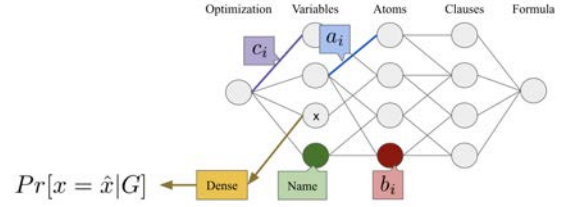


Figure 2: Graph representation. We represent the OMT problem as a graph with edges connecting variable nodes and constraint/cost nodes. The logic structure is represented similarly to an abstract syntax tree (AST), with the AST leaf nodes coinciding with the atomic constraint nodes.

6.1 Graph Representation

We translate an OMT problem into a graph by encoding each variable as a node; we encode the variables and atomic formula as is done in (Gasse et al. 2019). We encode the elements of $a_{i,j}$ as weights on edges between variables and constraints. We add $b_{i,j}$ as node attributes for constraint nodes and variable names as node attributes on variable nodes. We also have a cost node connected to each of the variable nodes with weights corresponding to \vec{c} . Each atomic formula node serves as leaves in an adjoining tree representing the clauses and final formula.

Taking from the approach in (Nair et al. 2020), we implement a neural partial assignment generator based on a graph convolutional neural network (GCN). As shown in Fig. 2, we build a graph connecting variable nodes and literal/constraint nodes with the edge weights indicating the linear coefficients in the literal. Finally, we encode the cost objective as special literal node, with the coefficients as edge weights.

In contrast to the encoding for Nair et al. (Nair et al. 2020), OMT also includes disjunctions over the literals. As such, we encode the disjunction as a tree over the literal nodes. Disjunction and conjunction nodes pass information between related clauses in rounds of message passing.

6.2 Formulating the Learning Problem

We consider the setting where an OMT solver is repeatedly solving similar problems. This means we can curate a training set of problems in this distribution based on historical queries or in simulation. Further, by design, our OMT engine can be run without the neural diver by replacing it with an initial SMT call. The cold-start OMT engine can be used to label training examples with optimal models. Using this labeled training set, we seek to reduce the required time to solve unseen problems from the same distribution.

With the graph encoding, call it G , our goal is to learn a function, f that estimates for each variable a probability distribution over potential values. We do this with a standard graph convolutional network (GCN) (Kipf and Welling 2016). We learn this function f over examples G_i labeled with x_i^* , a cost optimal variable assignment. We treat integer variable values as independent classes train the model to classify each variable.

446 We use a GCN to learn an embedding for each variable,
447 which we then pass through a linear layer to predict the class
448 corresponding to the variable assignment. We find it suffi-
449 cient to run two rounds of message passing for this appli-
450 cation. With two rounds, the variables nodes can be aggre-
451 gate information from two hop neighbors allowing the fi-
452 nal learned embedding of the variable to be both influenced
453 by variables that it shares an atomic constraint with and the
454 clause that it belongs to.

455 Using the learned embedding, we optimize the following
456 cross entropy loss: $\mathcal{L} = -\sum_{i=1}^m x_i^* \log p(x_i|G)$ where x_i^* is
457 the optimal assignment of the i -th variable and $\log p(x_i|G)$
458 is the probability of the assignment generated by the func-
459 tion f .

460 This loss intuitively maximizes the probability that the
461 optimal assignment is selected. Note for each variable the
462 GCN effectively approximate the probability distribution
463 over variable assignments conditioned on G , $p(x|G)$. This
464 is the desired f we sought to learn.

465 6.3 Partial Assignment Warm-Start

466 When the neural diver is used to solve a problem of interest,
467 the diver makes a prediction based on the input problem G .
468 This inference results in an estimated probability $P(x|G)$
469 returned as output logits. We sample variable assignments
470 based on these logits and use the KL divergence to a unifrom
471 distribution to estimate confidence. If the KL divergence is
472 larger than a confidence threshold, C , we abstain from as-
473 signing (see Appendix A.7 for implementation details). This
474 results in partial assignments α^i as only variables that are
475 easy to predict have assignments. To get full feasible solu-
476 tions \mathcal{A}^i , we call an SMT solver on each partial assignment
477 to get a complete feasible assignment. We do this by adding
478 to the existing OMT formula equality constraints $x = k$
479 where x is a variable and k is the sampled assignment from
480 the partial assignment generator.

481 We then use Ray (Moritz et al. 2018) to search for a valid
482 assignment in parallel for a user specified time, T , with K
483 parallel threads. If a thread discovers an unsatisfiable par-
484 tial assignment, it continues searching with another sample
485 from the generative model until the time expires. After run-
486 ning for T , the diver returns the best assignment discovered.
487 In the case that it does not find any feasible solutions, the
488 OMT engine runs SMT first to get a feasible assignment. By
489 default Ashera uses $T = 5s$, $K = 5$, and $C = 1$, but these
490 parameters can be tuned on a validation set in practice for
491 each problem family.

492 7 Experiment Setup

493 For this work, we look at two families of real-world OMT
494 problems: 1) DAG job scheduling, and 2) multi-agent travel-
495 ing salesman problem (TSP). Disjunctions native to these
496 two families are particularly challenging as they result in a
497 disconnected feasible set and a redundancy of equally opti-
498 mal solutions. For instance, in DAG scheduling, the each as-
499 signment of tasks to resources defines a nontrivial schedul-
500 ing subproblem, that tend to be similar but not identical
501 to each other. We describe the dataset generation, baseline

502 solvers in this section, and include experimental details such
503 as hardware setup in Appendix A.6.

Dataset Generation. In order to train and evaluate a
504 learning-based OMT solver, we generate instances for each
505 family of problems labeled with their optimal assignment. 506

Baseline Solvers. To compare with existing ILP solvers
507 (e.g., Gurobi), we encode the OMT benchmarks as ILP prob-
508 lems using the standard Big M encoding to approximate the
509 disjunctions. Big M encoding introduces an additional bi-
510 nary choice variable α to determine which clause in the dis-
511 junction is enforced as an ILP constraint. This increases the
512 number of variables combinatorially in the number of dis-
513 junctions and requires the ILP solver to optimize over
514 all disjunctive branch simultaneously. 515

516 For a fair comparison, the models were evaluated with-
517 out GPU assistance, but we expect improved performance
518 if accelerators are available at inference time. As Gurobi is
519 highly parallelized, to make fair comparison of algorithmic
520 cost, we compare results based on process time unless oth-
521 erwise noted and impose a 1 hr limit for experiments. We
522 do not include training time in our comparison as it required
523 only 2 hour on a single GPU and depends heavily on the
524 amount of data. Further, targeted applications of repeated
525 OMT solves occur on a weekly basis allowing for training
526 between invocations.

527 8 Results

528 In this section, we present our empirical results of existing
529 OMT tools, νZ , OptiMathSAT, Gurobi (ILP with Big M),
530 and Ashera. Our results show that Ashera scales to larger
531 problems, outperforming all three baselines by as much as
532 5x compared to the next best solver on the scheduling and
533 multi-agent TSP tasks. We compare performances with Par-
534 2 score, which is the solving time for solved instances and
535 two times timeout for unsolved ones.

536 8.1 Task Scheduling for Directed Acyclic Graphs

537 We generate a total of 43,596 similar DAG scheduling prob-
538 lems using the same problem encoding but varying the num-
539 ber of tasks and CPUs. We split up the evaluation based on
540 the number of tasks in the scheduling problem. For a realis-
541 tic setting, we consider two GPUs, and we have all the task
542 with the same expected runtime of 15 seconds and release
543 times of 2. This symmetry is notoriously difficult for tradi-
544 tional ILP solutions. To further make the instances compar-
545 able and always feasible, we scale up the deadline as the num-
546 ber of tasks increase, and only have one randomly placed de-
547 pendency between two tasks in the taskset. All task within
548 a problem instance have the same deadline as reported in
549 Table 4. In this section, we only consider Ashera trained
550 and tested on the *same* number of tasks and defer analysis
551 Ashera’s ability to adapt to tasks sizes not seen in training in
552 Section 8.3. We use $T = 1m$ for 10 tasks and default values
553 otherwise. 554

555 We report the performance of baseline solvers and Ashera
556 in Table 1 categorized by the number of tasks in the prob-
557 lem instance. The table indicates that existing baselines Op-
558 tiMathSAT has the best performance until 8 tasks while νZ

Table 1: OMT Solver Performance on Scheduling. We report PAR-2 scores of process time to account for timeouts and provide number of solved problems in parenthesis.

Number of Tasks	Average PAR-2 Score in Seconds (Number Solved in 1 hr)				
	Gurobi	vZ	OptiMathSAT	Ashera	Ashera Cold-start
5	2.21 (20)	0.02 (20)	0.02 (20)	2.25 (20)	2.35 (20)
6	2.30 (20)	0.11 (20)	0.03 (20)	2.31 (20)	2.40 (20)
7	2.78 (20)	1.00 (20)	0.17 (20)	2.33 (20)	2.50 (20)
8	3.84 (20)	16.27 (20)	1.28 (20)	3.04 (20)	3.44 (20)
9	26.29 (20)	149.39 (20)	31.17 (20)	16.78 (20)	13.95 (20)
10	400.01 (20)	5246.23 (9)	841.21 (20)	135.97 (20)	181.34 (20)
11	3562.15 (15)	7200.00 (0)	7200.00 (0)	3204.00 (18)	2924.77 (18)

Table 2: OMT Solver Performance on Multi-Agent TSP. We report PAR-2 scores of process time to account for timeouts and provide number of solved problems in parenthesis.

# Waypoints per Cluster	Average PAR-2 Score in Seconds (Number Solved in 1 hr)				
	Gurobi	vZ	OptiMathSAT	Ashera	Ashera Cold-start
3	0.91 (22)	0.16 (22)	0.12 (22)	2.48(22)	3.93 (22)
4	76.51 (22)	0.80 (22)	0.94 (22)	4.51(22)	7.89 (22)
5	5154.00 (10)	5.74 (22)	7.85 (22)	20.50(22)	24.29 (22)
6	7200.00 (0)	7200.00 (0)	90.54 (22)	104.00(22)	102.91 (22)
7	7200.00 (0)	7200.00 (0)	919.26 (22)	754.33 (22)	793.88 (22)

Table 3: Ablation on Neural Diving. We both test performance transfer to larger problems trained on smaller problems (Curriculum) and applying neural diving on existing solvers (Neural Diver + Gurobi/OptiMathSAT).

Number of Tasks	Average PAR-2 score in Seconds (Number Solved in 1 hr)			
	Neural Diver + Gurobi	Neural Diver + OptiMathSAT	Ashera	Ashera Curriculum
5	2.34 (20)	2.37 (20)	2.25 (20)	¹
6	2.42 (20)	2.43 (20)	2.31 (20)	2.28 (20)
7	2.66 (20)	2.44 (20)	2.33 (20)	2.30 (20)
8	4.00 (20)	3.24 (20)	3.04 (20)	2.36 (20)
9	19.44 (20)	12.03 (20)	16.78 (20)	13.97 (20)
10	288.09 (20)	288.09 (20)	135.97 (20)	174.19 (20)
11	2649.02 (17)	5728.87 (9)	3204.00 (18)	3118.62(17)

we look at two settings: 1) the performance of Ashera when tested on larger problems than seen in training, and 2) the neural diver applied to Gurobi and OptiMathSAT.

Performance transfer to larger problems. In this setting, Ashera is trained only on problems that are smaller than the test-time number of task. On scheduling, Table 3 shows Ashera performs comparably to when it is trained on the same sized problems. On TSP, the performance is also comparable (see Appendix). This ablation indicates that Ashera can be trained on smaller problems from the same family to scale to larger problems of interest.

Neural Gurobi and Neural OptiMathSAT. We further consider the performance of the learned neural diver on Gurobi and OptiMathSAT. For this setting, we replace the Ashera OMT engine with Gurobi or OptiMathSAT. As the diver provides a Z3 verified upper bound on the cost, all OMT solvers are compatible and can use the upper bound as a hint. On problems with 11 tasks, OptiMathSAT can solve 9 problems compared to none before and Gurobi can solve two additional problems. This demonstrates the use of neural guidance on OMT problems extends beyond our specific solver design.

9 Conclusions

Our work presents Ashera, a neural OMT solver, which performs up to 3x faster and solves three more problems than the widely-used commercial solver, Gurobi, on problems with 10 and 11 tasks, respectively. Traditional solvers, OptiMathSAT and Z3, are unable to solve any problems of 11 tasks within 1 hour timeout. Further, Ashera is 18% faster than OptiMathSAT on multiagent TSP with 15 waypoints, where Gurobi and Z3 timeout. As OMT problems solved in

¹We do not evaluate on 5 tasks as there are no smaller problems.

performs badly on 10 tasks. Gurobi however, continues to solve instances with 11 tasks where vZ and OptiMathSAT timeout. Ashera in contrast is 3x faster and solves three more problems than Gurobi on problems with 10 and 11 tasks, respectively. The results demonstrates the effectiveness of Ashera when the problem size scales. This suggests the benefit of applying Ashera to large-scale real-world applications, which typically have hundreds of variables.

8.2 Multi-agent TSP

In our multi-agent TSP benchmark, we generate 2500 instances of TSP with two clusters of waypoints arranged in a polygon, and additionally 22 problems for testing. We vary the distance between the center of the cluster and the origin where the vehicles start and vary the radius of the polygon. For simplicity, we provide as many vehicles as there are clusters and ensure restrictions on weight are not constraining. Additionally, the first waypoint is always the starting point for the vehicles.

Table 2 shows Ashera outperforms all three baselines on the largest problems, solving faster than OptiMathSAT while Gurobi and vZ solve none. In Table 2, we report Ashera’s performance when trained on instances with the same number of waypoints as those of the test set. We use $C = 0.5$ and otherwise use default parameters.

8.3 Cold-Start Ashera

We ablate the learned component of Ashera and see that Ashera performs faster on scheduling problems with 5 to 8 tasks. We attribute this to a 2 second overhead incurred in order to perform neural network inference on CPU and initialize Ray (Moritz et al. 2018). We leave for future work improvements afforded by hardware accelerators such as GPUs. Cold-start Ashera outperforms neural Ashera on scheduling problems with 11 tasks. For multi-agent TSP, Table 2 shows neural diving provides modest improvement compared to cold-start due to a larger number of variables and constraints. We note that Cold-start Ashera already outperforms or matches performances of the baselines.

8.4 Neural Diver Performance

One important aspect for neural-based solvers is its performance to transfer to similar but new problems. Specifically,

⁰We generated scheduling benchmarks of 11 to 20 tasks but all methods timeout with 1 hour at 11 tasks. Our benchmark contains multi-agent TSP problems of 8 to 10 waypoints per cluster but all methods timeout with 1 hour at 8 waypoints per cluster.

629 practice tend to be solved on a regular basis, we make the case
630 for learned-based OMT solver that train on a set of similar
631 problems encountered previously in the application or
632 in simulation. We contribute benchmark of problem families
633 including DAG task scheduling and multi-agent TSP for
634 evaluating learning-based OMT solvers.

635 10 Acknowledgements

636 This research is supported in part by NSF CISE Expeditions
637 Award CCF-1730628, NSF NRI #2024675 and under the NSF
638 AI4OPT Center. UC Berkeley research is also supported by
639 gifts from Alibaba, Amazon Web Services, Ant Financial,
640 CapitalOne, Ericsson, Meta, Futurewei, Google, Intel,
641 Microsoft, Nvidia, Sco tiabank, Splunk and VMware.

642 References

643 Balcan, M.; Dick, T.; Sandholm, T.; and Vitercik, E. 2018.
644 Learning to Branch. *CoRR*, abs/1803.10150.

645 Barrett, C.; Sebastiani, R.; Seshia, S. A.; and Tinelli, C.
646 2009. Satisfiability Modulo Theories. In Biere, A.; van
647 Maaren, H.; and Walsh, T., eds., *Handbook of Satisfiability*,
648 chapter 26, 825–885. IOS Press.

649 Beauchemin, M. 2014. Apache Airflow.

650 Bello, I.; Pham, H.; Le, Q. V.; Norouzi, M.; and Bengio,
651 S. 2016. Neural combinatorial optimization with reinforcement
652 learning. *arXiv preprint arXiv:1611.09940*.

653 Bertolissi, C.; Dos Santos, D. R.; and Ranise, S. 2018. Solv-
654 ing multi-objective workflow satisfiability problems with
655 optimization modulo theories techniques. In *Proceedings*
656 *of the 23rd ACM on Symposium on Access Control Models*
657 *and Technologies*, 117–128.

658 Bjørner, N.; Phan, A.-D.; and Fleckenstein, L. 2015. νZ -
659 An Optimizing SMT Solver. In Baier, C.; and Tinelli, C.,
660 eds., *Tools and Algorithms for the Construction and Analy-*
661 *sis of Systems*, 194–199. Berlin, Heidelberg: Springer Berlin
662 Heidelberg. ISBN 978-3-662-46681-0.

663 De Moura, L.; and Bjørner, N. 2008. Z3: An efficient
664 SMT solver. In *International conference on Tools and Algo-*
665 *rithms for the Construction and Analysis of Systems*, 337–
666 340. Springer.

667 Dutertre, B.; and De Moura, L. 2006. The yices smt solver.
668 *Tool paper at <http://yices.csl.sri.com/tool-paper.pdf>*, 2(2):
669 1–2.

670 Fan, W.; Ma, Y.; Li, Q.; Wang, J.; Cai, G.; Tang, J.; and Yin,
671 D. 2022. A Graph Neural Network Framework for Social
672 Recommendations. *IEEE Transactions on Knowledge and*
673 *Data Engineering*, 34(5): 2033–2047.

674 Gasse, M.; Chételat, D.; Ferroni, N.; Charlin, L.; and
675 Lodi, A. 2019. Exact combinatorial optimization with
676 graph convolutional neural networks. *arXiv preprint*
677 *arXiv:1906.01629*.

678 Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; and
679 Dahl, G. E. 2017. Neural message passing for quantum
680 chemistry. In *International conference on machine learn-*
681 *ing*, 1263–1272. PMLR.

Gog, I.; Kalra, S.; Schafhalter, P.; Gonzalez, J. E.; and Sto- 682
ica, I. 2022. D3: A Dynamic Deadline-Driven Approach 683
for Building Autonomous Vehicles. In *Proceedings of the* 684
Seventeenth European Conference on Computer Systems, 685
453–471. Association for Computing Machinery. 686

Gurobi Optimization, LLC. 2021. Gurobi Optimizer Refer- 687
ence Manual. 688

Henry, J.; Asavoae, M.; Monniaux, D.; and Maïza, C. 2014. 689
How to Compute Worst-Case Execution Time by Optimiza- 690
tion modulo Theory and a Clever Encoding of Program Sem- 691
antics. LCTES '14, 43–52. Association for Computing 692
Machinery. 693

Hopfield, J. J.; and Tank, D. W. 2004. “Neural” computation 694
of decisions in optimization problems. *Biological Cybernet-* 695
ics, 52: 141–152. 696

Katz, G.; Barrett, C.; Dill, D. L.; Julian, K.; and Kochen- 697
derfer, M. J. 2017. Reluplex: An efficient SMT solver for 698
verifying deep neural networks. In *International conference* 699
on computer aided verification, 97–117. Springer. 700

Kipf, T. N.; and Welling, M. 2016. Semi-supervised classi- 701
fication with graph convolutional networks. *arXiv preprint* 702
arXiv:1609.02907. 703

Kovácszai, G.; Biró, C.; and Erdélyi, B. 2017. Generating 704
Optimal Scheduling for Wireless Sensor Networks by Using 705
Optimization Modulo Theories Solvers. In *SMT*, 15–27. 706

Leino, K. R. M. 2010. Dafny: An automatic program verifier 707
for functional correctness. In *International conference on* 708
logic for programming artificial intelligence and reasoning, 709
348–370. Springer. 710

Leofante, F.; Abraham, E.; Niemueller, T.; Lakemeyer, G.; 711
and Tacchella, A. 2017. On the synthesis of guaranteed- 712
quality plans for robot fleets in logistics scenarios via opti- 713
mization modulo theories. In *2017 IEEE International Con-* 714
ference on Information Reuse and Integration (IRI), 403– 715
410. IEEE. 716

Manolios, P.; Pais, J.; and Papavasileiou, V. 2015. The Inez 717
Mathematical Programming Modulo Theories Framework. 718
In Kroening, D.; and Păsăreanu, C. S., eds., *Computer Aided* 719
Verification, 53–69. Springer International Publishing. 720

Mirhoseini, A.; Goldie, A.; Yazgan, M.; Jiang, J.; Songhori, 721
E.; Wang, S.; Lee, Y.-J.; Johnson, E.; Pathak, O.; Bae, S.; 722
et al. 2020. Chip placement with deep reinforcement learn- 723
ing. *arXiv preprint arXiv:2004.10746*. 724

Moritz, P.; Nishihara, R.; Wang, S.; Tumanov, A.; Liaw, R.; 725
Liang, E.; Elibol, M.; Yang, Z.; Paul, W.; Jordan, M. I.; and 726
Stoica, I. 2018. Ray: A Distributed Framework for Emerging 727
AI Applications. In *13th USENIX Symposium on Operating* 728
Systems Design and Implementation (OSDI 18), 561–577. 729

Moura, L. d.; and Bjørner, N. 2008. Z3: An efficient SMT 730
solver. In *International conference on Tools and Algo-* 731
rithms for the Construction and Analysis of Systems, 337– 732
340. Springer. 733

Nair, V.; Bartunov, S.; Gimeno, F.; von Glehn, I.; Lichocki, 734
P.; Lobov, I.; O’Donoghue, B.; Sonnerat, N.; Tjandraat- 735
madja, C.; Wang, P.; et al. 2020. Solving Mixed In- 736
teger Programs Using Neural Networks. *arXiv preprint* 737
arXiv:2012.13349. 738

739 Sebastiani, R.; and Trentin, P. 2015. OptiMathSAT: A Tool
740 for Optimization Modulo Theories. In Kroening, D.; and
741 Păsăreanu, C. S., eds., *Computer Aided Verification*, 447–
742 454. Springer International Publishing.

743 Selsam, D.; Lamm, M.; Bünz, B.; Liang, P.; de Moura, L.;
744 and Dill, D. L. 2019. Learning a SAT Solver from Single-Bit
745 Supervision. In *7th International Conference on Learning
746 Representations, ICLR 2019, New Orleans, LA, USA, May
747 6-9, 2019*.

748 Wu, Y.; Song, W.; Cao, Z.; and Zhang, J. 2021a. Learn-
749 ing Large Neighborhood Search Policy for Integer Program-
750 ming. In *Advances in Neural Information Processing Sys-
751 tems*.

752 Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; and Yu,
753 P. S. 2021b. A Comprehensive Survey on Graph Neural Net-
754 works. *IEEE Transactions on Neural Networks and Learn-
755 ing Systems*, 32(1): 4–24.

756 Yang, Z.; Chiang, W.-L.; Luan, S.; Mittal, G.; Luo, M.; and
757 Stoica, I. 2022. Balsa: Learning a query optimizer without
758 expert demonstrations. *arXiv preprint arXiv:2201.01441*.

759 Zhu, H.; Gupta, V.; Ahuja, S. S.; Tian, Y.; Zhang, Y.; and
760 Jin, X. 2021. Network Planning with Deep Reinforcement
761 Learning. In *Proceedings of the 2021 ACM SIGCOMM
762 2021 Conference, SIGCOMM '21*, 258–271. Association
763 for Computing Machinery.

Appendix A

765 A.1 DAG Scheduling with Deadlines

766 The scheduling problem belongs to a widely applicable fam-
767 ily of scheduling problems. It requires discovering the opti-
768 mal placement of tasks to resources as well as assigning start
769 times to tasks. Further, assignments must satisfy constraints
770 on both resources and dependencies between tasks. We de-
771 sire to maximize slack – the buffer time before the deadline
772 that a task is expected to complete. As such it’s often non-
773 trivial to find any feasible schedule, much less an optimal
774 one.

775 This family of problems appears in both the workflow
776 management platform Apache Airflow (Beauchemin 2014)
777 and in the DAG scheduler used for the dynamic deadline-
778 driven execution model for self driving (Gog et al. 2022).

779 We consider a set of N tasks, $T = \{t_i | i \in [1, N]\}$, and a
780 dependency matrix M where $M_{ij} = 1$ if t_i must complete
781 before t_j otherwise 0. We further consider a set of deadlines
782 and expected runtimes denoted as d_i and e_i , respectively.

783 In addition to runtime, we also consider resource require-
784 ments and placements. We denote r_i to be 1 if t_i requires a
785 GPU and 0 otherwise.

786 We seek to optimize with respect to two sets of variables
787 s_i and p_i which denote the start time and placement of t_i . Let
788 N_G and N_C be the number of GPUs and CPUs, respectively.
789 We encode $p_i = k$ to be in $[1, N_G]$ if it’s placed on the k^{th}
790 GPUs of N_G and $(k - N_G)^{th}$ CPU if it’s in $[N_G, N_G + N_C]$.

Our cost objective is

$$\sum_{i=0}^N d_i - (s_i + e_i)$$

791 with the following constraints:

- 792 • **Basic constraints.** For all i , $0 \leq s_i$ and $0 < p_i$.
- 793 • **Finish before deadline.** For all i , $s_i + e_i \leq d_i$.
- 794 • **Placement constraints.** For all i , if $r_i = 1$, $1 \leq p_i \leq$
795 N_G . Otherwise, $r_i = 0$, $1 \leq p_i \leq N_G + N_C$.
- 796 • **Dependency Respecting.** For all i, j , if $M_{ij} = 1$, $s_i +$
797 $e_i \leq s_j$.
- 798 • **Exclusion.** For all i, j , $p_i = p_j \implies (s_i + e_i \leq s_j \vee$
799 $s_j + e_j \leq s_i)$.

800 **Big M for Scheduling.** Unfortunately, the exclusion con-
801 straint requires a disjunction. In order to compare against
802 ILP solvers, we use the Big M strategy as presented in 2. We
803 introduce an additional two variables per pair of tasks i, j to
804 1) choose if task i and task j utilize the same resources and
805 2) choose if task i completes execution before task j begins
806 or vice versa. We provide the Big M version of the exclusion
807 constrain in Appendix A.3.

808 A.2 Multi-Agent Traveling Salesman Problem

The multi-agent Traveling Salesman Problem (TSP) appears
in practical route planning applications including package
deliveries in warehouse operations. A multi-agent TSP is
specified by the distances between the W waypoints and the
number of vehicles V . In this problem, the optimizer must

find an ordering o_i in which waypoint, i , is visited by vehi-
cle, v_i . We denote the starting waypoint as s . Our objective
is to minimize the sum of the times t_i when a waypoint is
visited:

$$\sum_{i=0}^N t_i$$

Due to space constraints, we highlight the following con- 809
straints and present the full encoding for multi-agent TSP in 810
Appendix A.4: 811

- 812 • **Visited.** All waypoints w must be visited by at least one 813
vehicle.
- 814 • **Deterministic.** After visiting a waypoint, w , a vehicle 815
visits at most one waypoint w' immediately afterwards.
- 816 • **Ordering.** The starting waypoint has $o_s = 0$. For all 817
waypoints, w , visited in order, o_w , the waypoint’s prede- 818
cessor p_w must have order $o_{p_w} = o_w - 1$. This prevents 819
tours that do not include the starting point.
- 820 • **Weight Constraint.** The sum of the weights of the vehi- 821
cles is less than a given value M .
- 822 • **Visit Time.** For all waypoints w , the visit time t_w if vehi- 823
cle v_w visits it is at least the $t_{p_w} + \tau_{v,p,w}$ where t_p is 824
the time when the preceding waypoint was visited and τ 825
is the travel time from p to w by vehicle v .
- 826 • **Exclusion.** If vehicle v is traveling from w to w' from t_w 827
to $t_{w'}$, there cannot be a waypoint w'' visited by v while 828
it’s traveling.

829 A.3 Big M for multi-agent TSP.

We again use the Big M encoding to encode disjunctions.
The most complex disjunction requiring the disjunction of
conjunctions is the ordering condition for a waypoint w :

$$\bigvee_{w' \in W, v \in V} \mathbf{M}_{v,w',w} \wedge (o_{w',v} = o_{w,v} - 1)$$

In the disjunction of conjuncts, all the inequalities in the con- 830
junct share the same choice variable, ensuring that all con- 831
straints in the disjunctive case hold simultaneously if cho- 832
sen. 833

Big M Exclusion Constraint We use Big M to encode the
exclusion constraint for MILP. The implication

$$p_i = p_j \implies (s_i + e_i \leq s_j \vee s_j + e_j \leq s_i)$$

can be encoded as

$$(p_i < p_j) \vee (p_j < p_i) \vee (s_i + e_i \leq s_j) \vee (s_j + e_j \leq s_i).$$

By creating two binary variables $\alpha_{i,j}$ and $\beta_{i,j}$ per con- 834
straint, we introduce the following constraints in replace- 835
ment using Big M: 836

- **Case 1** ($\alpha_{i,j} = 0, \beta_{i,j} = 0$) :
$$p_i - p_j < M\alpha_{i,j} + M\beta_{i,j}$$
- **Case 2** ($\alpha_{i,j} = 0, \beta_{i,j} = 1$) :
$$p_j - p_i < M\alpha_{i,j} + M(1 - \beta_{i,j})$$
- **Case 3** ($\alpha_{i,j} = 1, \beta_{i,j} = 0$) :
$$s_i + e_i - s_j \leq M(1 - \alpha_{i,j}) + M\beta_{i,j}$$
- **Case 4** ($\alpha_{i,j} = 1, \beta_{i,j} = 0$) :
$$s_j + e_j - s_i \leq M(1 - \alpha_{i,j}) + M(1 - \beta_{i,j})$$

837 A.4 Multi-Agent Traveling Salesman Problem

838 Full Encoding

839 The multi-agent traveling salesman problem can be defined
840 as an optimization problem where the aggregation over the
841 time when the waypoints are visited is minimized.

842 We index the vehicles and waypoints respectively from
843 0 to $|V| - 1$ and $|W| - 1$, where V and W are the set of
844 vehicles and waypoints and $|\cdot|$ denotes the cardinality of a
845 set. The constraints are defined over the following variables:

- 846 • u_v - whether or not a vehicle is being used
- 847 • $\mathbf{M}_{v,w,w'}$ - a Boolean 3D array indicating if vehicle v
848 travels from waypoint w to waypoint w'
- 849 • p_w - the preceding waypoint from which the vehicle vis-
850 its w
- 851 • x_w - the vehicle that visits waypoint w
- 852 • h - starting waypoint
- 853 • o_w - order that waypoint w is visited by a vehicle v . Note
854 that this is an ordering per vehicle not globally for all
855 vehicles.
- 856 • t_w - the time when waypoint w is visited
- 857 • m_{max} - total mass allowed

858 We get the following constants from an oracle.

- 859 • $\tau_{v,w,w'}$ - time for agent v to travel from w to w'
- 860 • $c_{v,w,w'}$ - energy consumption
- 861 • γ_v - vehicle weight.

862 Our optimization seeks to minimize the aggregated time
863 t when the waypoints are visited under the following con-
864 straints:

1. Each waypoint except the harbor must be visited by a vehicle:

$$\forall w' \in W \setminus \{h\}. \sum_{v \in V, w \in W} \mathbf{M}_{v,w,w'} = 1$$

The harbor has to be visited by the vehicles that are used.

$$\forall v \in V. \sum_{w \in W} \mathbf{M}_{v,w,h} = u_v$$

- 865 2. From one waypoint only one other waypoint is visited
866 next (determinism), and according to fixed order:

$$\forall w \in W \setminus \{h\}. \sum_{v \in V, w' \in W \setminus \{h\}} \mathbf{M}_{v,w,w'} = 1$$

Vehicles that are used should leave the harbor.

$$\forall v \in V. \sum_{w' \in W} \mathbf{M}_{v,h,w'} = u_v$$

3. No self loop allowed.

$$\forall v \in V, w \in W. \overline{\mathbf{M}_{v,w,w}}$$

4. If a point has order o then it must have been reach from another point with order $o - 1$. For the harbor starting point we have,

$$\forall v \in V. o_{h,v} = 0$$

for $o_{w,v}$ where v is unused we similarly constrain it to be zero:

$$\forall w \in W \setminus \{h\}, v \in V. \\ (o_{w,v} = 0) \vee \bigvee_{w' \in W} \mathbf{M}_{v,w,w'}$$

and also

$$\forall w \in W \setminus \{h\}. \\ \bigvee_{w' \in W, v \in V} (\mathbf{M}_{v,w',w} \wedge o_{w',v} = o_{w,v} - 1)$$

5. For each waypoint w , a vehicle must visit another way-
point w' from w if it travels from some waypoint w'' to
 w :

$$\forall v \in V, w \in W. \\ \left(\sum_{w'' \in W} \mathbf{M}_{v,w'',w} = 1 \rightarrow \sum_{w' \in W} \mathbf{M}_{v,w,w'} = 1 \right)$$

6. Constraint for p_w :

$$\forall w \in W. p_w = \sum_{v \in V, w' \in W} w' \cdot \mathbf{M}_{v,w',w}$$

7. Constraint for x_w :

$$\forall w \in W. x_w = \sum_{v \in V, w' \in W} v \cdot \mathbf{M}_{v,w',w}$$

8. A vehicle is used when:

$$\forall v \in V. (u_v \leftrightarrow \bigvee_{w, w' \in W} \mathbf{M}_{v,w,w'})$$

9. The total weight is less than a given value:

$$\sum_{v \in V} u_v \cdot \gamma_v < m_{max}$$

10. The total time each agent takes is equal to t , which is in the minimization problem:

$$\sum_{v \in V, w \in W, w' \in W \setminus \{h\}} \mathbf{M}_{v,w,w'} \cdot \tau_{v,w,w'} = t$$

A.5 Literal Dropping

To further improve performance, we recognize that literals can be dropped safely if the formula is expressed as negation normal form (NNF), in which the negation operator is only applied to atoms and the only allowed Boolean operators are conjunction and disjunction. We formally state the observation as follows.

Observation 1. Given a satisfying assignment \mathcal{A} to an SMT formula ϕ in NNF over LIA, let L^+ and L^- be the sets of true and false (theory) literal that appear in ϕ by applying \mathcal{A} . Literals $l \in L^+$ define a valid solution space for variables in ϕ . That is, $l \in L^-$ can be dropped while maintaining soundness.

867

868

869

870

871

872

873

874

875

876

877

878

879

880

881

882

883 *Proof.* First, observe that removing false literals $l \in L^-$ in
 884 an NNF ϕ does not affect the satisfiability of ϕ (for $l \in L^+$
 885 alone satisfies ϕ). Thus, passing L^+ to an LIA theory solver,
 886 any solution returned by the theory solver must satisfy $l \in$
 887 L^+ and hence will satisfy ϕ . \square

888 Moreover, since $L^+ \cup L^-$ imposes more constraints than
 889 L^+ does, the optimal value found in L^+ can be greater (resp.
 890 smaller) than that found in $L^+ \cup L^-$ when maximizing (resp.
 891 minimizing) an objective. This allows us to push bounds
 892 even further in the ILP solving phase.

893 **Summary of Benchmarks** Table 4 and Table 5 provide a
 894 summary of the number of variables and constraints for each
 895 size of problem in DAG scheduling and multiagent TSP, re-
 896 spectively.

Table 4: Scheduling Benchmark Summary.

Number of Tasks ²	Number of Training Cases	Number of Test Cases	Average Variable Count	Average Constraint Count	Deadline
5	811	20	16	48	50
6	1459	20	19	63	57
7	2383	20	22	80	65
8	3630	20	25	99	72
9	5250	20	28	120	80
10	7291	20	31	143	87
11	9801	20	-	-	-
12	12831	20	-	-	-

Table 5: Multi-Agent TSP Benchmark Summary.

# Waypoints per Cluster	Number of Training Cases	Number of Test Cases	Average Variable Count	Average Constraint Count
3	2500	22	137	428
4	2500	22	211	654
5	2500	22	301	928
6	2500	22	401	1250
7	2500	22	519	1620
8	2500	22	-	-

897 A.6 Hardware Setup and Software Versions

898 For evaluation, we used c5.xlarge AWS cloud instances,
 899 which have 3.0 GHz Intel Xeon Platinum processors with
 900 4 vCPUs and 8 GiB of RAM. For training, we used two
 901 Quadro GV100 GPUs with 32GB GPU Memory. We imple-
 902 mented parallel assignment search using Ray 1.10 (Moritz
 903 et al. 2018), a distributed programming framework. Our
 904 evaluation uses the following solver versions as base-
 905 lines: Gurobi v9.5.1 (Gurobi Optimization, LLC 2021), Z3
 906 4.8.16 (De Moura and Bjørner 2008), and OptiMathSAT
 907 1.7.3 (Sebastiani and Trentin 2015). The same version of
 908 Gurobi is used for Logical Neighborhood Search and like-
 909 wise of Z3 as the verifier in completing partial assignments
 910 and verifying soundness in the OMT engine.

911 A.7 Implementation details of diver.

912 **KL divergence.** We use the KL divergence to compare our
 913 learned distribution from a uniform distribution over the al-
 914 lowed variables values. We use static analysis to extract sin-
 915 gle variable inequalities ($k < x$) to obtain simple upper and
 916 lower bounds.

917 If the learned distribution is similar to this restricted uni-
 918 form distribution it is likely the variable is symmetric in the

919 data and an assignment to the variable is likely to precipitate
 920 simpler subproblems.

Parallel search. To augment the parallel search, we addi-
 921 tionally ran a thread that simply ran the OMT engine in par-
 922 allel with the diving exploration. This way if the problem is
 923 small and easy to solve, the solver will not be penalized to
 924 severely for diving.

925 The diver on the other hand explored up to 5 partial as-
 926 signments at a time as sampled from the generative model.
 927 This pool of partial assignments always includes the highest
 928 probability partial assignment. This assignment is gotten by
 929 taking the highest probability class from generative model
 930 using an argmax. We found this practically useful as this as-
 931 signment is an often promising partial assignment but as the
 932 number of predicted variables increases the odds of getting
 933 this particular sample decreases.
 934