Machine Learning for Combinatorial Optimization and Continuous Optimization

Combinatorial Optimization and Machine Learning | Lecture 8

Sanjeeb Dash / Parikshit Ram

June 28, 2023

JPOC'13

Les 20 ans !

Plan

1. ML for Combinatorial Optimization

2. ML for Continuous Optimization

Outline

1 ML for Combinatorial Optimization

2 ML for Continuous Optimization

A Commom Problem: MILP

Mixed Integer Linear Program or MILP

$$\min_{\theta} c^{\top}\theta \tag{1}$$

$$\texttt{subject to } A\theta \leq b, \tag{2}$$

Terminology

- ⇛ D integer variables, d continuous variables
- ⇛ c – $(D+d)$ objective coefficients
- ⇛ A – constraint coefficient matrix for p constraints
- ⇛ b – p constraint thresholds

Exact Solver: Branch and Bound

Algorithm 1 Branch-and-bound algorithm (BnB) for MILP P

Initialize: Set of open leaves $S \leftarrow \{P\}$ `// run preprocessing routines`
Initialize: UB $U \leftarrow +\infty$, LB $L \leftarrow -\infty$ respectively
while $S \neq \emptyset$ *and* $U > L$ **do**
  Select open leaf M from S `// node selection routines`
  if M *is an integral node* **then**
    Compute obj $\hat{l}$ in M, update $U \leftarrow \min\{U, \hat{l}\}$, $L \leftarrow \min\{L, \hat{l}\}$, and continue
  Relax M & solve to get node LB $\tilde{l}$ `// run primal heuristics or add cuts here`
  if $\tilde{l} > U$ **then**
    continue `// can prune this node`
  $L \leftarrow \min\{L, \tilde{l}\}$
  if *solution integral* **then**
    $U \leftarrow \min\{U, \tilde{l}\}$ and continue
  `// variable selection routines`
  Select fractional variable $j \in [D]$, split, and push child problems M_1 and M_2 into S

Configuring a Branch-and-Bound Solver

Decisions to make:

- $\Rrightarrow$ Which open leaf to consider next?
- $\Rrightarrow$ Which fractional variable to split on?
- $\Rrightarrow$ Whether and which primal heuristics to run?
- $\Rrightarrow$ Whether and which cuts to add?
- $\Rrightarrow$ Whether and which preprocessing routines to run?
- $\Rrightarrow$ ...

Decision frequency:

- $\Rrightarrow$ Once initially to globally set solver configuration – *a single "decision"*
- $\Rrightarrow$ Adaptively, during the execution at each point that needs a choice – *a "policy" to make a sequence of decisions*

Configuring a Branch-and-Bound Solver

Quantifying the quality of a decision or policy – the (sequence of) decisions

$\Rrightarrow$ Time to solve – that is, time to $U = L$

$\Rrightarrow$ Branch-and-bound tree size

Configuring a Branch-and-Bound Solver

What has been automated with machine learning?

- ⇛ Node selection
- ⇛ Variable selection
- ⇛ Cutting planes selection
- ⇛ Primal heuristic selection
- ⇛ Formulation selection
- ⇛ Neighborhood search heuristics
- ⇛ Diving heuristics

Configuring Branch-and-bound Solvers

All products / ILOG CPLEX Optimization Studio / 12.8.0.0 /

MIP strategies

Last Updated: 2021-03-08

Here are links to parameters controlling MIP strategies.

algorithm for initial MIP relaxation

Benders strategy

MIP subproblem algorithm

MIP variable selection strategy

MIP strategy best bound interval

MIP branching direction

backtracking tolerance

MIP dive strategy

MIP heuristic frequency

local branching heuristic

MIP priority order switch

MIP priority order generation

MIP node selection strategy

node presolve switch

MIP probing level

RINS heuristic frequency

feasibility pump switch

scale parameter for subMIPs

algorithm for initial MIP relaxation of a subMIP of a MIP

algorithm for subproblems of a subMIP of a MIP

All products / ILOG CPLEX Optimization Studio / 12.8.0.0 /

MIP cuts

Last Updated: 2021-03-08

Here are links to parameters controlling cuts.

These parameters set limits on cut generation.

constraint aggregation limit for cut generation

cut factor row-multiplier limit

type of cut limit

number of cutting plane passes

These parameters control specific types of cuts.

Boolean Quadric Polytope cuts

MIP cliques switch

MIP covers switch

MIP disjunctive cuts switch

MIP flow cover cuts switch

MIP flow path cut switch

MIP Gomory fractional cuts switch

- pass limit for generating Gomory fractional cuts
- candidate limit for generating Gomory fractional cuts

MIP GUB cuts switch

MIP globally valid implied bound cuts switch

MIP locally valid implied bound cuts switch

Lift-and-project cuts switch for MIP and MIQCP

MCF cut switch

MIP MIR (mixed integer rounding) cut switch

Reformulation Linearization Technique (RLT) cuts

MIP zero-half cuts switch

All products / ILOG CPLEX Optimization Studio / 12.8.0.0 /

MIP variable selection strategy

Value	Symbol	Meaning
-1	CPX_VARSEL_MININFEAS	Branch on variable with minimum infeasibility
0	CPX_VARSEL_DEFAULT	Automatic: let CPLEX choose variable to branch on; **default**
1	CPX_VARSEL_MAXINFEAS	Branch on variable with maximum infeasibility
2	CPX_VARSEL_PSEUDO	Branch based on pseudo costs
3	CPX_VARSEL_STRONG	Strong branching
4	CPX_VARSEL_PSEUDOREDUCED	Branch based on pseudo reduced costs

All products / ILOG CPLEX Optimization Studio / 12.8.0.0 /

MIP Gomory fractional cuts switch

Value	Meaning
-1	Do not generate Gomory fractional cuts
0	Automatic: let CPLEX choose; **default**
1	Generate Gomory fractional cuts moderately
2	Generate Gomory fractional cuts aggressively

Configuring Branch-and-bound Solvers

- $\Rightarrow$ Set of problems $\{P_i, i \in [n]\}$
- $\Rightarrow$ Solver configuration parameters ϕ
- $\Rightarrow$ Solution quality metric $S(P_i, \phi)$ by solving problem P_i with a BnB solver configured with ϕ

$$\min_{\phi} \sum_{i \in [n]} S(P_i, \phi) \tag{3}$$

Configuring Branch-and-bound Solvers

$$\min_{\phi} \sum_{i \in [n]} S(P_i, \phi) \tag{4}$$

Solution: SMBO – sequential model based optimization / derivative-free optimization / global optimization / kriging / Bayesian optimization

Implications

⇛ If $\bar{\phi}$ is a good configuration for all $P_i, i \in [n]$ (on average), it would be good for a new problem, provided ...

⇛ the set $\{P_i, i \in [n]\}$ is a diverse set, but ...

⇛ the score $S(\cdot, \cdot)$ needs to be calibrated properly so that we don't end up optimizing only for the problems with scores in the higher end

Configuring Solvers with a Model using Problem Features

Extension. Utilize features of the MILP problem

Given features g_i for a problem P_i,

$\Rrightarrow$ learn a ML model M that predicts the score $S(P_i, \phi)$ – that is $M(g_i, \phi) \approx S(P_i, \phi)$, and

$\Rrightarrow$ for any problem P_j with features g_j, pick solver configuration by minimizing the predicted score $M(g_j, \phi)$ over ϕ (over the space of valid configurations):

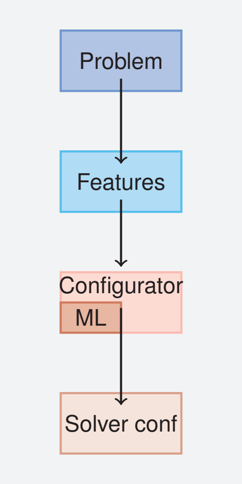
$$\min_M \sum_{i \in [n]} S(P_i, \phi_i) \quad \texttt{subject to} \quad \forall i \in [n], \phi_i \in \arg\min_{\varphi} M(g_i, \varphi) \tag{5}$$

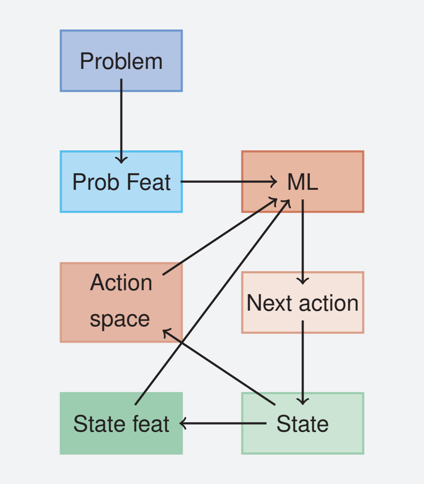
Solution. Extended version of SMBO

Limitations

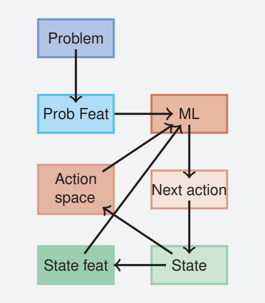
- $\Rrightarrow$ Single solver conf for all problems seems limiting
  - $\Rrightarrow$ Handled to some extent by using problem features – adapting solver conf to problem
- $\Rrightarrow$ SMBO requires multiple evaluations of $S(P,\phi)$ for different problem P and solver confs ϕ
  - $\Rrightarrow$ Each eval requires a MILP solution
  - $\Rrightarrow$ Might be computationally infeasible since we might need to *use a lot of problems* $\{P_i, i \in [n]\}$ and *obtain evals for many solver configurations* ϕ to learn a good scorer model M.

Guiding Solvers with a Policy





Policy: Imitation Learning



Imitation Learning

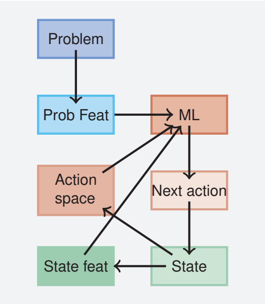
The ML model M corresponds to a "policy" – a model that makes a sequence of decisions – that **learns to mimic an expert policy**.

Why might this be useful?

⇛ Do not have access to expert policy at execution

⇛ Expert policy computationally expensive to constantly invoke during execution – for example, Strong Branching is expensive to invoke at each variable selection.

Policy: Imitation Learning



Imitation Learning

The ML model M corresponds to a “policy” – a model that makes a sequence of decisions – that **learns to mimic an expert policy**.

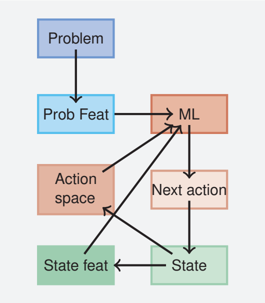
Given a problem P_i (with features g_i) solved by an expert,

$\Rightarrow$ We have a sequence of state-action pairs $\{(s_t, a_t)\}_{t \in [T_i]}$ from the execution

$\Rightarrow$ We learn M to mimic the actions taken by the expert in any state

$$\min_M \sum_{i \in [n]} \sum_{t \in [T_i]} \mathscr{L}\Big(\underbrace{a_t,}_{\text{expert action}} \quad \underbrace{M(g_i, s_t)}_{\text{action by ML}} \Big) \tag{6}$$

Policy: Imitation Learning

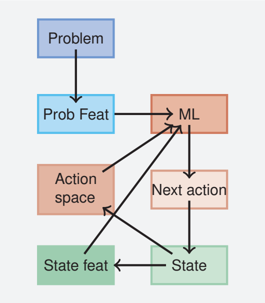


Various MILP benchmarks already provide data for this form of learning.

Opportunities for innovation

- $\Rrightarrow$ How to create problem features?
- $\Rrightarrow$ How to create features for states and action spaces?
- $\Rrightarrow$ How to model the policy M?

Policy: Reinforcement Learning



Reinforcement Learning

The ML model M corresponds to a “policy” – a model that makes a sequence of decisions – that **learns to maximize some reward**.

Given a problem P_i (with features g_i),

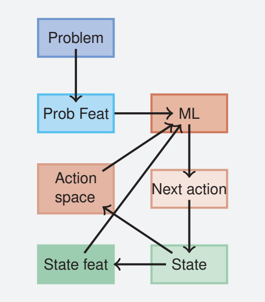
$\Rrightarrow$ Rollout policy M to solve P_i

$\Rrightarrow$ Obtain a sequence of state/action/reward tuples $\{(s_t, a_t, r_t, s_{t+1}), t \in [T_i]\}$

$\Rrightarrow$ We learn M to maximize the rewards from the actions

$$\min_{M} \sum_{i \in [n]} \sum_{t \in [T_i]} \underbrace{w_t}_{\text{state weight}} \cdot \underbrace{r_t}_{\text{reward at state } t} \tag{7}$$

Policy: Reinforcement Learning



The data for this kind of learning needs to be generated on the fly – *we will have to (partially or fully) solve MILPs for each policy rollout*.

Additional opportunities for innovation

- ⇛ How to design a useful reward mechanism?
- ⇛ How to model intermediate rewards in a MILP solution path?
- ⇛ How to learn in a "sample efficient" manner – that is, not have to solve a lot of MILPs?

Existing Literature: Leveraging ML in MILP

Paper	Decision	ML Technique+Model	Data
He et al. [2014]	Node selection & pruning	Imitation Learning w/ policy network	node (4); branching (4-5); tree-specific (5);
Khalil et al. [2016]‡	Branching var	Learning to rank w/ linear model (Imitation Learning)	static problem (18); dynamic node (54); SB scores as targets;
Khalil et al. [2017]	Heuristics which & if	Linear model per-heuristic	global (4); depth (2); node LP (8); frac-score (35);
Balcan et al. [2018]	Branching var	Linear model	Diff var selection scores

$\Rrightarrow$ ‡ Learning and deployment can be done on-the-fly

Existing Literature: Leveraging ML in MILP

Paper	Decision	ML Technique+Model	Data
Fischetti et al. [2019]	MILP resolution	Classification w/ random forests	node (4); node LP (11); tree (6); global bounds (5);
Tang et al. [2019]†	Cutting plane	Reinforcement Learning w/ policy network (attention+LSTM)	current set of csts; current sol of LP relaxation set of Gomory's cuts
Gasse et al. [2019]	Branching var	Imitation Learning w/ GNN based problem feat	MILP $\rightarrow$ var-cst BP graph; var feat; edge feat; cst feat SB choice as targets
Gupta et al. [2020]	Branching var	Imitation Learning w/ hybrid GNN+MLP	graph feat [Gasse et al., 2019] per-node feat [Khalil et al., 2016] SB choice as targets

$\Rrightarrow$ † For IP only

He et al. [2014]

Features

- ⇛ Node features: Node LB, estimated objective, depth, whether it is child/sibling of last selected node
- ⇛ Branching features: For branching var leading to current node, pseudocost, (root LP sol val - current node LP sol val), (val - current bound)
- ⇛ Tree features: global LB, global UB, integrality gap, num sols found, if $U - L = \infty$

Performance

Dataset	Ours			Ours (prune only)			SCIP (time)		Gurobi (node)	
	speed	OGap	IGap	speed	OGap	IGap	OGap	IGap	OGap	IGap
MIK	**4.69×**	**0.04‰**	**2.29%**	4.45×	**0.04‰**	**2.29%**	3.02‰	**1.89%**	0.45‰	2.99%
Regions	2.30×	**7.21‰**	**3.52%**	**2.45×**	7.68‰	3.58%	**6.80‰**	**3.48%**	21.94‰	5.67%
Hybrid	**1.15×**	**0.00‰**	**3.22%**	1.02×	**0.00‰**	3.55%	0.79‰	4.76%	3.97‰	5.20%
CORLAT	1.63×	**8.99%**	22.64%	**4.44×**	**8.91%**	**17.62%**	6.67%	fail	2.67%	fail

Khalil et al. [2016] Features

Feature	Description	Count	Reference
Static Features (18)			
Objective function coeffs.	Value of the coefficient (raw, positive only, negative only)	3	
Num. constraints	Number of constraints that the variable participates in (with a non-zero coefficient)	1	
Stats. for constraint degrees	The *degree of a constraint* is the number of variables that participate in it. A variable may participate in multiple constraints, and statistics over those constraints' degrees are used. The constraint degree is computed on the root LP (mean, stdev., min, max)	4	
Stats. for constraint coeffs.	A variable's positive (negative) coefficients in the constraints it participates in (count, mean, stdev., min, max)	10	
Dynamic Features (54)			
Slack and ceil distances	$\min\{\breve{x}^i_j - \lfloor \breve{x}^i_j \rfloor, \lceil \breve{x}^i_j \rceil - \breve{x}^i_j\}$ and $\lceil \breve{x}^i_j \rceil - \breve{x}^i_j$	2	
Pseudocosts	Upwards and downwards values, and their corresponding ratio, sum and product, weighted by the fractionality of x_j	5	(Achterberg 2009)
Infeasibility statistics	Number and fraction of nodes for which applying SB to variable x_j led to one (two) infeasible children (during data collection)	4	
Stats. for constraint degrees	A dynamic variant of the static version above. Here, the constraint degrees are on the current node's LP. The ratios of the static mean, maximum and minimum to their dynamic counterparts are also features	7	
Min/max for ratios of constraint coeffs. to RHS	Minimum and maximum ratios across positive and negative right-hand-sides (RHS)	4	(Alvarez, Louveaux, and Wehenkel 2014)
Min/max for one-to-all coefficient ratios	The statistics are over the ratios of a variable's coefficient, to the sum over all other variables' coefficients, for a given constraint. Four versions of these ratios are considered: positive (negative) coefficient to sum of positive (negative) coefficients	8	(Alvarez, Louveaux, and Wehenkel 2014)
Stats. for active constraint coefficients	An active constraint at a node LP is one which is binding with equality at the optimum. We consider 4 weighting schemes for an active constraint: unit weight, inverse of the sum of the coefficients of all variables in constraint, inverse of the sum of the coefficients of only candidate variables in constraint, dual cost of the constraint. Given the absolute value of the coefficients of x_j in the active constraints, we compute the sum, mean, stdev., max. and min. of those values, for each of the weighting schemes. We also compute the weighted number of active constraints that x_j is in, with the same 4 weightings	24	(Patel and Chinneck 2007)

Khalil et al. [2016] Performance

		CPLEX-D	SB	PC	SB+PC	SB+ML
Unsolved Instances	All (523)	11	129	66	63	**52**
	Easy (255)	0	12	15	14	**13**
	Medium (120)	2	43	22	22	**17**
	Hard (148)	9	74	29	27	**22**
Num. Nodes	All (523)	46,633	33,072	92,662	70,455	**59,223**
	Easy (255)	3,255	3,610	7,931	5,224	**5,124**
	Medium (120)	173,417	121,923	395,199	288,916	**234,093**
	Hard (148)	1,570,891	519,878	1,971,333	1,979,660	**1,314,263**
Total Time	All (523)	499	2,263	**960**	1,093	1,059
	Easy (255)	111	602	**243**	361	382
	Medium (120)	1,123	6,169	2,493	1,892	**1,776**
	Hard (148)	3,421	9,803	4,705	4,718	**4,039**

Khalil et al. [2017]

Features

Global Features (4)
Optimality gap
Infinite gap?
Root LP value / Global Lower Bound
Root LP value / Global Upper Bound
Depth Features (2)
Node Depth / Max. Depth in Tree
Node Depth / Max. Possible Depth
Node LP Features (8)
Sum of variables' LP solution fractionalities / Num. of Fractional Variables
Num. of Fractional Variable / Num. of Integer Variables
Num. Variables Roundable Up (Down) / Num. of Integer Variables (x2)
Num. of Active Constraints / Num. of Constraints
Node is root?
Root LP value / Node LP value
Root LP value / Node Estimate
Scoring Features for Fractional Variables (35)
Number of Up Locks (x5) – Number of Down Locks (x5)
Normalized Objective Coefficient (x5)
Objective Gain (x5)
Distance to root LP solution (x5)
Vector Length (x5)
Pseudocost score (x5)

Performance

MIPLIB – Num. Instances = 280	DEF	ML	ML/DEF
Primal integral	95.65	89.65	0.94
Time to first incumbent	34.23	26.60	0.78
Time to best incumbent	746.95	738.71	0.99
Total calls (ML heurs.)	755.19	514.77	0.68
Total time (ML heurs.)	124.38	101.88	0.82
Num. incumbents (ML heurs.)	1.85	2.45	1.33
Success Rate (ML heurs.)	0.00036	0.00064	1.79
Num. incs. per heur. sec. (ML heurs.)	0.00565	0.00860	1.52
Num. Instances Solved	170	172	1.01
Total time (BnB)	3,966.47	4,119.67	1.04
Total nodes (BnB)	27,458.77	27,904.43	1.02
Primal-dual integral	34,390.33	35,329.91	1.03

Fischetti et al. [2019] Features

Count	Group name	Features general description
7	Last observed global measures	Gap, global bounds ratio, fraction of nodes left attaining max/min objective estimate, comparison of max/min estimates with incumbent, primal-dual integral
4	Nodes left and pruned, iterations count	Throughput of pruned nodes, comparison with nodes left, trend w.r.t. max observed # of nodes left, simplex iterations throughput
4	Node LP integer infeasibilities (iinf)	Max/min/avg number of observed iinf, fraction of nodes with iinf below 5% quantile value
5	Incumbent	Throughput of incumbent updates, average frequency and improvement of updates (normalized), distance from last observed update (normalized), was an incumbent found before an integer feasible node (boolean)?
4	Best bound	Throughput of best bound updates, average frequency and improvement of updates (normalized), distance from last observed update (normalized)
3	Node LP objective	Fraction of nodes with objective above the 95% quantile value, normalized differences between quantile threshold and global bounds
4	Node LP fixed variables	Fraction of max/min observed # of fixed variables, fraction of nodes with # of fixed variables above 95% quantile value, normalized distance from last observed peak
6	Depth and tree traversal	Comparison of max observed depth with # of processed nodes, normalized height of last full level and waist of the tree, average length of dives (normalized), frequency of leaps in the traversal

Fischetti et al. [2019] Performance

	Dum	LR	SVM	**RF**	ExT	MLP
Accuracy	0.57	0.93	0.94	**0.94**	0.93	0.93
Precision	0.57	0.93	0.94	**0.95**	0.94	0.93
Recall	0.57	0.93	0.94	**0.94**	0.93	0.93
F1-score	0.57	0.93	0.94	**0.94**	0.93	0.93

Gasse et al. [2019] Features

Bi-partite Graph & Graph-Convolutional NN

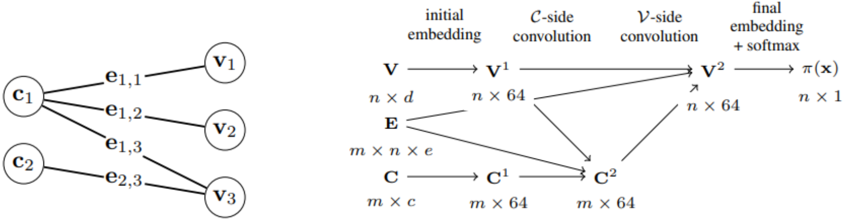


Figure 2: Left: our bipartite state representation $\mathbf{s}_t = (\mathcal{G}, \mathbf{C}, \mathbf{E}, \mathbf{V})$ with $n = 3$ variables and $m = 2$ constraints. Right: our bipartite GCNN architecture for parametrizing our policy $\pi_\theta(\mathbf{a} \mid \mathbf{s}_t)$.

Gasse et al. [2019] Performance

Model	Easy Time	Easy Wins	Easy Nodes	Medium Time	Medium Wins	Medium Nodes	Hard Time	Hard Wins	Hard Nodes
FSB	17.30 ± 6.1%	0 / 100	17 ±13.7%	411.34 ± 4.3%	0 / 90	171 ± 6.4%	3600.00 ± 0.0%	0 / 0	n/a ± n/a %
RPB	8.98 ± 4.8%	0 / 100	**54** ±20.8%	60.07 ± 3.7%	0 / 100	1741 ± 7.9%	1677.02 ± 3.0%	4 / 65	47 299 ± 4.9%
TREES	9.28 ± 4.9%	0 / 100	187 ± 9.4%	92.47 ± 5.9%	0 / 100	2187 ± 7.9%	2869.21 ± 3.2%	0 / 35	59 013 ± 9.3%
SVMRANK	8.10 ± 3.8%	1 / 100	165 ± 8.2%	73.58 ± 3.1%	0 / 100	1915 ± 3.8%	2389.92 ± 2.3%	0 / 47	42 120 ± 5.4%
LMART	7.19 ± 4.2%	14 / 100	167 ± 9.0%	59.98 ± 3.9%	0 / 100	1925 ± 4.9%	2165.96 ± 2.0%	0 / 54	45 319 ± 3.4%
GCNN	**6.59** ± 3.1%	**85** / 100	134 ± 7.6%	**42.48** ± 2.7%	**100** / 100	**1450** ± 3.3%	**1489.91** ± 3.3%	**66** / 70	**29 981** ± 4.9%

AAAI'21 tutorial on *Recent Advances in Integrating Machine Learning and Combinatorial Optimization* https://sites.google.com/view/ml-co-aaai-21/

SCIP based toolkit for Research on ML4CO.

https://doc.ecole.ai/master/index.html

Outline

1 ML for Combinatorial Optimization

2 ML for Continuous Optimization

Learning to optimize with gradients

Problem:

$$\min_{\theta \in \mathbb{R}^d} f(\theta) \tag{8}$$

Gradient descent:

$$\theta^{k+1} \leftarrow \theta^k - \alpha^k \nabla_\theta f(\theta^k) \tag{9}$$

Optimization with a ML model M_ϕ parameterized with ϕ [Andrychowicz et al., 2016]:

$$\theta^{k+1} \leftarrow \theta^k + M_\phi \left(\nabla_\theta f(\theta^k) \right) \tag{10}$$

Meta-optimization problem

For optimization objective f with *optimizee* variables θ_f and a optimization horizon K, meta-objective with respect to the *optimizer* variables ϕ:

$$\mathscr{L}(\phi) = \mathbb{E}_f \left[f\left(\theta_f^K\right)\right] \tag{11}$$

Generalization:

$$\mathscr{L}(\phi) = \mathbb{E}_f \left[\sum_{k=1}^{K} w^k f\left(\theta_f^k\right)\right] \tag{12}$$

A simple recurrent neural network

RNN are neural-networks used for learning with sequences $(\theta_1, y_1), (\theta_2, y_2), \ldots$ where we want the learner to take into account the fact that the data is sequential in nature.

Classical Elman network with input θ_{t-1} & hidden-layer "history" vector h_t:

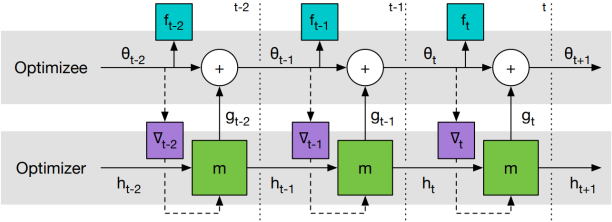
$$h_t = \sigma_h (W_h \theta_{t-1} + U_h h_{t-1} + b_h) \tag{13}$$

$$y_t = \sigma_y (W_y h_t + b_y), \tag{14}$$

where $\phi = \{W_h, U_h, b_h, W_y, b_y\}$ and σ_h and σ_y are the activation functions.

In practice, Long Short Term Memory (LSTM) networks [Hochreiter and Schmidhuber, 1997] and Gated Recurrent Unit (GRU) networks [Cho et al., 2014] are the RNN-du-jour.

Recurrent neural network as optimizer [Andrychowicz et al., 2016]



$$\mathbf{g}^{k+1}, \mathbf{h}^{k+1} \leftarrow M_\phi\left(\nabla_\theta f(\theta^k), \mathbf{h}^k\right), \tag{15}$$

$$\theta^{k+1} \leftarrow \theta^k + \mathbf{g}^{k+1} \tag{16}$$

Training the RNN

Input: Distribution of objective functions F
Input: Learning rate β, optimization horizon K, trajectory weights $w^k, k \in [K]$
Input: Initialization of the *optimizer* variables ϕ
while *not converged* **do**

Sample $f \sim F$ with *optimizee* variables θ_f

Initialize θ_f^0 randomly, $\mathbf{h}_0 \leftarrow \mathbf{0}$

Unroll K steps with RNN M_ϕ $(k = 1, \ldots, K)$: `// inner optimization`

$$\mathbf{g}^k, \mathbf{h}^k \leftarrow M_\phi \left(\nabla_{\theta_f} f(\theta_f^{k-1}), \mathbf{h}^{k-1} \right), \qquad \theta_f^k \leftarrow \theta_f^{k-1} + \mathbf{g}^k \tag{17}$$

Update the RNN parameters ϕ: `// RNN-optimization`

$$\nabla_\phi \mathscr{L} \leftarrow \frac{\partial}{\partial \phi} \sum_{k=1}^{K} w^k f(\theta_f^k), \qquad \phi \leftarrow \phi - \beta \nabla_\phi \mathscr{L} \tag{18}$$

return ϕ

Note. Need second-order derivative of f wrt θ_f or assume $\partial \nabla_{\theta_f} f / \partial \phi = 0$

Details: RNN Architecture

Coordinate-wise RNN – allows θ_f to have different dimensionalities

$\Rrightarrow$ RNN with 1-dimensional input

$\Rrightarrow$ Each dimension in θ_f shares RNN weights,

$\Rrightarrow$ But has its own (scalar) history $h^k \in \mathbf{h}^k$

$\Rrightarrow$ Andrychowicz et al. [2016] say *"... has the nice effect of making the optimizer invariant to the order of the variables (...) since the same update rule is used independently for each coordinate ..."*

Details: Distribution of Objectives

Wichrowska et al. [2017, Section 4.1] suggest the following distribution:

- ⇛ 2-dimensional problems from literature (Goldstein, Hartmann, Rosenbrock, Branin)
- ⇛ Well-behaved convex problems (quadratic bowls, logistic regression on linearly separable data)
- ⇛ Objectives with slow convergence
  - ⇛ *"many dimensional oscillating valley whose global minimum lies at* ∞*"*,
  - ⇛ *"problems with a loss consisting of a very strong coupling term between variables in a sequence"*,
  - ⇛ objective *"only depends on the minimum and maximum valued variables, so the gradients are extremely sparse and (...) has discontinuous gradients"*

Modification: Weighing the Optimization Trajectory

$$\mathscr{L}(\phi) = \mathbb{E}_f \left[\sum_{k=1}^{K} w^k f(\theta_f^k) \right] \tag{19}$$

Different choices of w^k:

$\Rrightarrow$ $w^k = \mathbb{I}[k = K] \rightarrow$ focus on final solution [Lv et al., 2017]

$\Rrightarrow$ $w^k = 1$ (or $1/K$) $\rightarrow$ focus on full trajectory [Andrychowicz et al., 2016]

$\Rrightarrow$ $w^k = k \rightarrow$ more focus on later steps; but some on earlier steps too [Ruan et al., 2020]

Modification: Transforming the RNN Training Objective

Original meta-objective:

$$\mathscr{L}(\phi) = \mathbb{E}_f\left[\sum_{k=1}^{K} w^k f(\theta_f^k)\right] \tag{20}$$

Assuming (or translating) $\min_{\theta_f} f(\theta_f) = 0 \forall f$, for some $\varepsilon > 0$, Wichrowska et al. [2017] instead consider:

$$\mathscr{L}(\phi) = \mathbb{E}_f\left[\sum_{k=1}^{K} w^k \log\left(f(\theta_f^k) + \varepsilon\right)\right] \tag{21}$$

to better encourage exact convergence to minima.

Modification: Generating Harder Functions to Train on

Transformation to f for better RNN-training [Lv et al., 2017, Wichrowska et al., 2017]:

$\Rrightarrow$ Simulate problems with sparse gradients by setting large fraction of ∇_{θ_f} to 0.

$\Rrightarrow$ Simulate different scaling across optimizee variables with a linear (randomized) transformation of the variables.

$\Rrightarrow$ Simulate different steepness profiles by applying a monotonic transformation to the objectives

$\Rrightarrow$ Simulate complex objectives with diverse parts which sums the objective values and concatenates the variables from a diverse set of objectives

$$f(\theta_f) = f_1(\theta_{f_1}) + f_2(\theta_{f_2}) + \ldots$$

Modification: Additional RNN Inputs

Instead of the gradient ∇_θ as the RNN input, consider versions of [Lv et al., 2017, Wichrowska et al., 2017]:

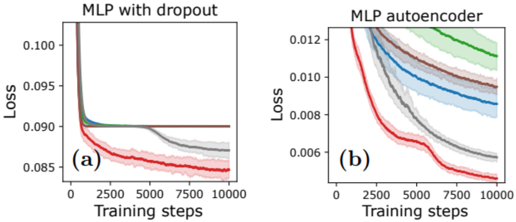
⇛ Momentum terms $m^k \leftarrow \eta_1 m^{k-1} + (1-\eta_1)\nabla_{\theta_f} f(\theta_f^k)$

⇛ Relative gradient magnitudes $v^k \leftarrow \eta_2 v^{k-1} + (1-\eta_2)\left(\nabla_{\theta_f} f(\theta_f^k)\right)^2$, where $\left(\nabla_{\theta_f} f(\theta_f^k)\right)^2$ is the element-wise square of $\nabla_{\theta_f} f(\theta_f^k)$

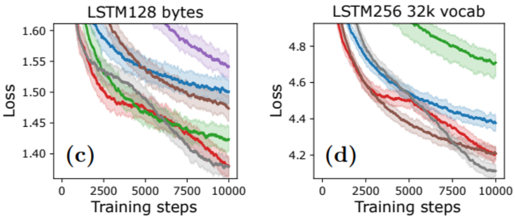
$$\mathbf{g}^{k+1}, \mathbf{h}^{k+1} \leftarrow M_\phi(\nabla_{\theta_f} f(\theta_f^k), m^k, v^k, \mathbf{h}^k) \tag{22}$$

Recent Results [Metz et al., 2022]

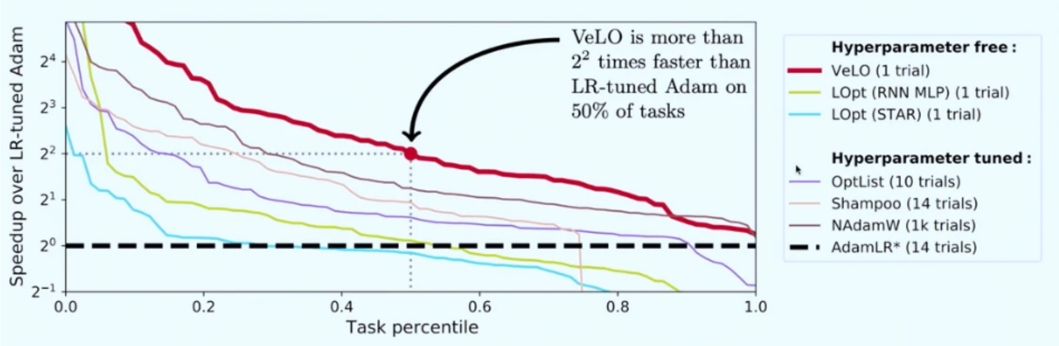
Best cases



Worst cases



Recent Results [Metz et al., 2022]



Learning to Optimize with Zeroth-order Oracle [Ruan et al., 2020]

Learned optimizer with gradients:

$$\theta_f^{k+1} \leftarrow \theta_f^k + M_\phi \left(\nabla_{\theta_f} f(\theta_f^k) \right) \tag{23}$$

Instead consider zeroth-order gradient estimates [Liu et al., 2018] with n random Gaussian directions $\mathbf{u}_i \in \mathbb{R}^d, i = 1, \ldots, n$, and a smoothing parameter $\mu > 0$:

$$\widehat{\nabla}_{\theta_f} f(\theta_f^k) = \frac{1}{\mu n} \sum_{i=1}^{n} \mathbf{u}_i \left(f(\theta_f^k + \mu \mathbf{u}_i) - f(\theta_f^k) \right). \tag{24}$$

Learned optimizer with gradient estimates:

$$\theta_f^{k+1} \leftarrow \theta_f^k + M_\phi \left(\widehat{\nabla}_{\theta_f} f(\theta_f^k) \right) \tag{25}$$

We can also “optimize” for the random directions $\mathbf{u}_i$ by modifying the diagonal of the covariance matrix of the Gaussian.

Learning to Optimize with Function Values [Chen et al., 2017]

Learned optimizer for function values:

$$\theta_f^{k+1} \leftarrow M_\phi\left(\theta_f^k, f(\theta_f^k)\right) \tag{26}$$

- ⇛ Useful for black-box optimization (like in hyperparameter optimization),
- ⇛ **But** RNN training requires first-order derivative of f w.r.t. θ_f
- ⇛ Cannot use coordinate-wise RNN, needing a RNN per optimizee variable – no parameter sharing between variables in θ_f
- ⇛ For cheap f, might be better to use L2O with (zeroth-order) gradient estimates
- ⇛ Open question: weight sharing RNN that does not use gradient estimates

Resources

- ⇛ Survey – Learning to Optimize: A Primer and A Benchmark [Chen et al., 2022]
- ⇛ Invited talk by Jascha Sohl-Dickstein (Google) at ICLR 2023:
  - ⇛ **Learned optimizers: why they're the future, why they're hard, and what they can do now**
  - ⇛ `https://iclr.cc/virtual/2023/invited-talk/14236`

References I

He He, Hal Daume III, and Jason M Eisner. Learning to search in branch and bound algorithms. In *Advances in neural information processing systems*, pages 3293–3301, 2014. URL https://papers.nips.cc/paper_files/paper/2014/file/757f843a169cc678064d9530d12a1881-Paper.pdf.

Elias Boutros Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. Learning to branch in mixed integer programming. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016. URL https://www.ekhalil.com/files/papers/KhaLebSonNemDil16.pdf.

Elias B Khalil, Bistra Dilkina, George L Nemhauser, Shabbir Ahmed, and Yufen Shao. Learning to run heuristics in tree search. In *IJCAI*, pages 659–666, 2017. URL https://www.ijcai.org/proceedings/2017/0092.pdf.

Maria-Florina Balcan, Travis Dick, Tuomas Sandholm, and Ellen Vitercik. Learning to branch. In *International Conference on Machine Learning*, pages 344–353, 2018.

Martina Fischetti, Andrea Lodi, and Giulia Zarpellon. Learning milp resolution outcomes before reaching time-limit. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 275–291. Springer, 2019. URL https://cerc-datascience.polymtl.ca/wp-content/uploads/2018/11/Technical-Report_DS4DM-2018-009.pdf.

Yunhao Tang, Shipra Agrawal, and Yuri Faenza. Reinforcement learning for integer programming: Learning to cut. *arXiv preprint arXiv:1906.04859*, 2019.

Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 15580–15592, 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/d14c2267d848abeb81fd590f371d39bd-Paper.pdf.

Prateek Gupta, Maxime Gasse, Elias B Khalil, M Pawan Kumar, Andrea Lodi, and Yoshua Bengio. Hybrid models for learning to branch. *arXiv preprint arXiv:2006.15212*, 2020.

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in neural information processing systems*, pages 3981–3989, 2016.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

References II

Olga Wichrowska, Niru Maheswaranathan, Matthew W Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Nando de Freitas, and Jascha Sohl-Dickstein. Learned optimizers that scale and generalize. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3751–3760. JMLR. org, 2017.

Kaifeng Lv, Shunhua Jiang, and Jian Li. Learning gradient descent: Better generalization and longer horizons. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2247–2255. JMLR. org, 2017.

Yangjun Ruan, Yuanhao Xiong, Sashank Reddi, Sanjiv Kumar, and Cho-Jui Hsieh. Learning to learn by zeroth-order oracle. *ICLR*, 2020.

Luke Metz, James Harrison, C Daniel Freeman, Amil Merchant, Lucas Beyer, James Bradbury, Naman Agrawal, Ben Poole, Igor Mordatch, Adam Roberts, et al. Velo: Training versatile learned optimizers by scaling up. *arXiv preprint arXiv:2211.09760*, 2022. URL `https://arxiv.org/pdf/2211.09760.pdf`.

S. Liu, J. Chen, P.-Y. Chen, and A. O. Hero. Zeroth-order online admm: Convergence analysis and applications. In *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, volume 84, pages 288–297, April 2018.

Yutian Chen, Matthew W Hoffman, Sergio Gómez Colmenarejo, Misha Denil, Timothy P Lillicrap, Matt Botvinick, and Nando de Freitas. Learning to learn without gradient descent by gradient descent. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 748–756. JMLR. org, 2017.

Tianlong Chen, Xiaohan Chen, Wuyang Chen, Zhangyang Wang, Howard Heaton, Jialin Liu, and Wotao Yin. Learning to optimize: A primer and a benchmark. *The Journal of Machine Learning Research*, 23(1):8562–8620, 2022. URL `https://jmlr.org/papers/volume23/21-0308/21-0308.pdf`.