Boosting optimization via machine learning

Michele Lombardi and Michela Milano
Dipartimento di Informatica – Scienza e Ingegneria
Università di Bologna

IJCAI-ECAI 2018
Tutorial overview: part 1

• Optimization for boosting Machine Learning: few details
• Machine Learning for boosting optimization: an overview
  ✓ Supporting the modeling activity:
    • Machine Learning to detect constraints (e.g. Constraint Acquisition, Model Seeker)
    • Machine Learning models as constraints (e.g. Empirical Model Learning, partially defined constraints)
    • Objective function learning
  ✓ Boosting Search
    • No good learning and recording
    • Reinforcement learning for search
    • Approximating branching heuristics
  ✓ Algorithm selection and tuning
    • Using ML to predict the run time
    • Using ML to select the best approach
  ✓ Similar approaches
    • Surrogate based optimization
    • Black box optimization
    • LION (Learning and Intelligent OptimizatioN) approach, metaheuristics
    • Model predictive control
• Concluding remarks and future research directions
Tutorial overview: part 2

- Hands on session (about Empirical Model Learning)
  - Overview of the target problem (Off-line/on-line optimization)
  - Overview of the tools
  - Sampling the on-line input space to build a training set
  - Approximating the on-line behavior via ML
  - Embedding the ML model in a combinatorial model
We are very grateful to:

- Nicolas Beldiceanu
- Bistra Dilkina
- Gene Freuder
- Arnaud Gotlieb
- Tias Guns
- Lars Kotthoff
- Arnaud Lallouet
- Nadjib Lazaar

for providing material used to prepare this tutorial
Machine learning and optimization

• A lot of research activity going on in two directions:
  
  • Helping ML algorithms through optimization techniques
  
  • Boosting optimization via ML
    • this is what this tutorial is about
Improving ML algorithms via optimization

• Pattern mining:
  • Optimization methods such as SAT/CP/ILP enable to express complex constraints on patterns and combination thereof
  • Itemset mining and extensions (cost constraints, closed itemset, max itemset)


Improving ML algorithms via optimization

• Pattern mining:
  • Optimization methods such as SAT/CP/ILP enable to express complex constraints on patterns and combination thereof

• Sequence mining with constraints on syntax, data covered, relationship


Improving ML algorithms via optimization

• Pattern mining:
  • Optimization methods such as SAT/CP/ILP enable to express complex constraints on patterns and combination thereof

• Pattern set mining


Improving ML algorithms via optimization

• Clustering:


Berg, Jeremias, and Matti Jarvisalo. "Optmal correlaton clustering via MaxSAT , AIJ Journal 2017


Boosting optimization via ML

• Supporting the modeling activity:
  • Machine Learning is used to learn problem model components:
    • Constraints
    • Objective function

• Boosting Search/Solving
  • Machine Learning is used to speed up search:
    • Variable/value selection heuristics
    • New search strategies
    • Enabling specific algorithms during search

• Algorithm selection and tuning
  • Machine learning is used to select the best algorithm in a portfolio
  • Machine learning is used for parameter tuning
ML for supporting the modeling activity

M. Lombardi, M. Milano, Boosting Combinatorial Problem Modeling with Machine Learning, IJCAI2018
Supporting the modeling activity

• Use implicit information from a set of examples to obtain part of a combinatorial model
  – a constraint
  – an objective function.

• There are three main approaches:
  – extracting information using the native constraint language of the solver;
  – embedding a fully-fledged Machine Learning model in a combinatorial approach.
  – extreme case, the ML model makes up (almost) all of the combinatorial model.
Supporting the modeling activity

• Traditionally, combinatorial optimization problems are models from iterative interactions between a domain and an optimization expert.

• ML could replace (or support)
  – the optimization expert with a constraint acquisition algorithm,
  – the domain expert with an example generator.

• We need data:
  – Either historical data
  – Data from the expert
  – Data extracted from the real system or a simulator
Learning native constraint language components

Aimed at learning models in the general form:

\[
\begin{align*}
\min z &= x_0 \\
\text{subject to: } \pi_i(x) &\quad \forall i \in I \\
\vec{x} &\in D_{\vec{x}}
\end{align*}
\]

- where \( \vec{x} \) are problem variables, \( D_{\vec{x}} \) their domain and \( x_0 \) is the variable representing the objective value, \( \pi_i(\vec{x}) \) are problem constraints (predicates)

- Notably the predicates \( \pi_i(\vec{x}) \) are defined using the building blocks from the hosting approach
Learning native constraint language components

• Relevant approaches
  – Building over the constraint programming paradigm
    • CONACQ [Bessiere et al., 2017a] (in its various versions),
    • QUACQ [Bessiere et al., 2013],
    • Model seeker [Beldiceanu and Simonis, 2012]
  – Building over Inductive logic programming
    • [Lallouet et al., 2010]
Version space learning [Mitchell82]

• Let $X=x_1,..,x_n$ a set of attributes of domains $D=D_1,..,D_n$

• A concept is a Boolean function
  • $f(x_i)=0 \Rightarrow x_i$ is a negative instance
  • $f(x_j)=1 \Rightarrow x_j$ is a positive instance

• Given a set of hypothesis $H$, any subset of $H$ represents a version space

• A concept to learn is the set of positive instances that can be represented by a version space
Version space learning [Mitchell82]

- Most specific concept to learn:
  \[ f : (\forall x_i \in E^+ : f(x_i) = 1) \land (\forall x_i \in E^- : f(x_i) = 0) \]

  \[ f \equiv h_2 \land h_6 \land h_9 \]
Constraint acquisition as version space learning

Constraint Programming:

Constraint network

Concept Learning
Constraint acquisition

• Inputs:
  • (X,D): Vocabulary
  • Γ: Constraint language
  • B: Bias (constraints/hypothesis)
  • C_T: Target Network (concept to learn)
  • (E^+,E^-): training set

• Output:
  • C_L: Learned network such that: all positive and no negative example covered
Convergence/collapse state
Convergence/collapse state

Collapse state

\[ C_T \]

\[ E^- \]

\[ E^+ \]
Acquisition via membership queries

- **CONACQ** [Bessiere et al. AIJ17]
  - SAT-Based constraint acquisition
  - Bidirectional search using Membership queries
  - Conacq1.0 (passive learning)
  - Conacq2.0 (active learning)

\[ K = (\neg x_1 \land \neg x_2 \land \neg x_3) \land (x_4 \lor x_5 \lor x_6 \lor x_7) \ldots \]

Non-learnability using Membership queries [Constraint Acquisition, AIJ17]
Quick acquisition

- **QUACQ** [Bessiere et al. IJCAI13]
  - Active learning approach
  - Bidirectional search
    - But it can be top-down search if no positive example
  - Based on partial queries to elucidate the scope of the constraint to learn
  - Learnability using partial queries
Membership queries

\[ \text{ask}(2, 8, 4, 2, 6, 5, 1, 6) \]

\[ \text{ask}(2, 8, 4, 2, 6, 5, 1, 6) = \text{No} \]
Partial queries

ask(2, 8, 4, 2, -, -, -, -) = No
Partial queries

```
ask(2, 8, -, -, -, -, -) = Yes
```
Partial queries

\[ \text{ask}(2, 8, 4, -, -, -, -) = \text{No} \]
Partial queries

- The number of queries required to find the target concept is in:

\[ O(|C_T| \cdot (\log |X| + |\Gamma|)) \]

- The number of queries required to converge is in:

\[ O(|B|) \]

QUACQ needs more than 8000 queries to learn the Sudoku model
Limitations

- **Too many queries to generate a model**
  - More than 8000 queries for generating a SUDOKU model

- **To overcome the limitation:**
  - Either elicit more knowledge when asking queries to the user
  - Or learn structured problems

- Constraint acquisition works learning only binary constraints otherwise the bias would grow exponentially
Acquisition via more complex queries

- **Matchmaker agents** [Freuder and Wallace wAAAI97]
- **Argument-Based CONACQ** [Friedrich et al. 09]
- **ILP-Based Acquisition** [Lallouet et al. 10]
Matchmaker agents

- Matchmaker agents are based on the Constraint Acquisition and Satisfaction Problem (CASP) paradigm

- Two agents: the solver and the customer
  - The customer knows the problem but not so explicitly that it can tell the solver outright
  - The solver suggests solutions based on the constraints it knows about
  - The customer then gives the solver the complete set of constraints violated by the last solution

Goal: limit the number of interactions
Matchmaker agents

Goal: limit the number of interactions

Two strategies:

SATISFACTION STRATEGIES
• Try to find solutions that satisfy additional constraints w.r.t. the ones known by the solver

VIOLATION STRATEGIES
• Try to find solutions that violate most the constraints

Argumentation based learning generates a logical theory $C$ based on a set of arguments ARC provided by an domain expert.

- An argument $arg$ is a set of constraints.
- We have positive, negative, too strong, necessary, sufficient, too weak arguments.

Implementation based on
- QUICKXPLAIN finding the minimal conflict set
- HD-TREE: breadth first search to find all min conflict sets + hitting set
Argument-based acquisition

- The constraint acquisition algorithm continues to generate solutions until all the constraints in the given bias are fixed.
- Three stages:
  - generation of an example
  - validation of the solution by an expert
  - learning of the version space
ILP-based acquisition

Use ILP to learn the model

• Search state = conjunction of litterals (clause)
• A rule r covers an example e if there exists \( \sigma \) such that \( \sigma(r) \subseteq e \)
• State evaluation = number of covered positive examples, number of negative examples rejected, size of the conjunction, etc.

• Search for a rule using classical ILP techniques (separate and conquer)
• Bidirectional search
ILP-based acquisition

\[
\begin{align*}
\text{spec} & ::= \text{rule} \mid \text{spec} \land \text{spec} \\
\text{rule} & ::= \forall \text{variables} : \text{body} \rightarrow \text{head} \\
\text{variables} & ::= \text{vs} \in \text{DOMAIN} \mid \text{variables}, \text{variables} \\
\text{vs} & ::= \text{VARIABLE} \mid \text{vs}, \text{vs} \\
\text{body} & ::= \text{BODY}_\text{ATOM} \mid \text{body} \land \text{body} \\
\text{head} & ::= \text{HEAD}_\text{ATOM} \mid \neg\text{HEAD}_\text{ATOM} \mid \text{head} \lor \text{head}
\end{align*}
\]

Arnaud Lallouet, Matthieu Lopez, Lionel Martin, Christel Vrain:
On Learning Constraint Problems. ICTAI (1) 2010: 45-52
ILP-based acquisition

∀Q₁, Q₂ ∈ Queens, ∀X₁, X₂, Y₁, Y₂ ∈ Int :
position(Q₁, X₁, Y₁) ∧ position(Q₂, X₂, Y₂)
→ Q₁ = Q₂ ∨ X₁ ≠ X₂
∧
∀Q₁, Q₂ ∈ Queens, ∀X₁, X₂, Y₁, Y₂ ∈ Int :
position(Q₁, X₁, Y₁) ∧ position(Q₂, X₂, Y₂)
→ Q₁ = Q₂ ∨ Y₁ ≠ Y₂
∧
∀Q₁, Q₂ ∈ Queens, ∀X₁, X₂, Y₁, Y₂ ∈ Int :
position(Q₁, X₁, Y₁) ∧ position(Q₂, X₂, Y₂)
ecart(X₁, X₂, L₁) ∧ ecart(Y₁, Y₂, L₂) → Q₁ = Q₂ ∨ L₁ ≠ L₂
Structured problem acquisition: ModelSeeker

- **ModelSeeker** [Beldiceanu and Simonis, CP11’12]
  - Generates constraint models for structured problems from positive examples
  - Based on global constraint catalogue (≈1000)
  - Bottom-up search
  - ModelSeeker learns constraints underlying the scheduling of the Bundesliga (the German Football Liga) from a single example schedule.
Structured problem acquisition: ModelSeeker

- Transformation: transforms inputs in suitable representations
- Sequence generation: transforms vars in regular sets
- Argument creation: creates patterns
- Constraint Seeker: finds matching constraints using the catalogue
- The following three steps reduce the set of constraints that are output of the previous step
Embedding an ML model into a combinatorial model

Aimed at learning models in the general form:

\[
\begin{align*}
\min z &= x_0 \\
\text{subject to: } \pi_i(\vec{x}) &\quad (P2) \\
\nu_m(\vec{x}_{m,\text{in}}, \vec{x}_{m,\text{out}}) &\quad \forall m \in M \\
\vec{x} &\in D_{\vec{x}} \\
\end{align*}
\]

- where \( \vec{x} \) are problem variables, \( D_{\vec{x}} \) their domain and \( x_0 \) is the variable representing the objective value, \( \pi_i(\vec{x}) \) are problem constraints (predicates) and \( \nu_m(\vec{x}_{m,\text{in}}, \vec{x}_{m,\text{out}}) \) is a proper encoding of a machine learning model in the hosting language
Motivating example problem: traffic light placement

- Add/remove traffic light over a whole city
- Traffic lights can be connected (green wave)
- Every operation has a cost (and there is a budget)
- Probably other constraints (not considered)
- Improve traffic figures

HP: we want to solve it! What is the toughest obstacle?
An example problem: traffic light placement

- Add/remove traffic light over a whole city
- Traffic lights can be connected (green wave)
- Every operation has a cost (and there is a budget)
- Probably other constraints (not considered)
- Improve traffic figures

HP: we want to solve it! What is the toughest obstacle?

Assessing the effect of traffic light placement on the traffic levels:

Impossible via expert-designed declarative models

Human behaviour – complex system
Assessing the effect of traffic light placement on the traffic levels

Traffic light placement: decidables
Traffic levels: observables

WE WANT TO LEARN THE IMPACT OF DECIDABLES ON OBSERVABLES and CAST THIS INTO AN OPTIMIZATION MODEL
A second example: policies for PV adoption

- Given a target of PV adoption to be achieved
- Decide which incentives to give
- Every incentive has a cost (and there is a budget)
- Probably other constraints (not considered)

HP: we want to solve it! What is the toughest obstacle?
A second example: policies for PV adoption

- Given a target of PV adoption to be achieved
- Decide which incentives to give
- Every incentive has a cost (and there is a budget)
- Probably other constraints (not considered)

HP: we want to solve it! What is the toughest obstacle?

Assessing the effect of incentives on PV adoption:
Impossible via expert-designed declarative models
Human behaviour – Social Dynamics
Assessing the effect of incentives on PV adoption

Incentives: decidables
PV adoption levels: observables

WE WANT TO LEARN THE IMPACT OF DECIDABLES ON OBSERVABLES

and

CAST THIS INTO AN OPTIMIZATION MODEL
A third example: thermal-aware workload dispatching

- Given a target heterogeneous platform and a workload
- Decide workload (tasks) dispatching
- Every task has a thermal dynamics (and there is a thermal controller)
- Load balancing constraints
- **With thermal limits**
A third example: thermal-aware workload dispatching

- Given a target heterogeneous embedded platform and a workload
- Decide task allocation and scheduling
- Every task has a thermal dynamics (and there is a thermal controller)
- Probably other constraints (not considered)

HP: we want to solve it! What is the toughest obstacle?
A third example: thermal-aware workload dispatching

- Given a target heterogeneous embedded platform and a workload
- Decide task allocation and scheduling
- Every task has a thermal dynamics (and there is a thermal controller)
- Probably other constraints (not considered)

HP: we want to solve it! What is the toughest obstacle?

Assessing the effect of workload allocation on thermal dynamics:

Possible with simulation based on differential equations of the thermal dynamics BUT not manageable by combinatorial optimization techniques
Assessing the effect of workload allocation on thermal dynamics:

Workload allocations: decidables
Core temperatures: observables

WE WANT TO LEARN THE IMPACT OF DECIDABLES ON OBSERVABLES

and

CAST THIS INTO AN OPTIMIZATION MODEL
What do these problems have in common?

A strong combinatorial structure

Part of the problem is not easily described through a manageable declarative model

Decisions influence predictions and vice versa

We have access to problem data (historical, simulated, real) to create a training set
Empirical Model Learning - EML

- Start from a set of observations extracted from runs on a real system
  - Sample the allocation space by running the system with different inputs of workload dispatching
  - Produce a training set

![Task sequence - Temperature](image)

![Task sequence - Power](image)

We computed optimized workload allocation and frequency assignments for both the target platforms, by running each approach for 90 seconds on an Intel Core 2 T7200, 2GHz; the resulting solutions were executed on the simulation framework, with all cores starting from a room temperature of 26.5 °C. Since all considered variants make use of approximated thermal model, there is no theoretical guarantee for the dominance of one approach over another: the use of simulation to evaluate the results ensures a fair comparison and a reliable effectiveness assessment. Moreover, since the optimized solution are evaluated via simulation, the results are unaffected by any numerical issue in the models.

The typical thermal behavior exhibited by the NN approach solutions is depicted in Figure 2, showing both temperature and power dynamics for a single experiment; each line in the graphs corresponds to a PE: as one can see, after an initial transient behavior the temperature becomes pretty stable, thanks to the thermal aware allocation.

Next, we compared the final (simulated) peak temperature obtained by each of the considered approaches; the results are shown in Figure 3, depicting for the 40 instances the distribution (histogram) of the difference $T_{HH} - T_{NN}$ (in dark grey) and $T_{PP} - T_{NN}$ (in light grey). For the considered configuration, discrepancies of around 1-2 °C we found to be relevant; also, in some cases, the network-based approach is consistently better than the HH one and even improves (on average) the PP approach, which is known to use a very good temperature proxy measure.

In order to investigate the effect of non-homogeneity, we performed a second evaluation after having asymmetrically pre-heated the target platforms; in this case the starting temperature for each core are 31.05 °C, 33.55 °C, 35.75 °C, 36.48 °C. The resulting differences in the final peak temperature are shown in Figure 4; as one can see the advantage of the NN approach becomes more relevant, due to the inability of the surrogate objective functions to capture the initial asymmetry.
Empirical Model Learning - EML

- Start from a set of monitoring data from the real system:
- Learn an approximate function via Machine Learning

\[ f: \text{dispatching} \rightarrow \text{temperature} \]
Empirical Model Learning - EML

- Start from a set of observations
- Learn an approximate function via Machine Learning
- Embed this “empirical model” inside an optimization model

\[
\min z = f(\bar{y})
\]

subject to:  \( \bar{y} = g(\bar{x}) \)

all manner of complex constraints

::
In principle, we can have a combinatorial problem solver and a black box tool to test the solution. Can be very expensive.
In principle...... we can have a combinatorial problem solver and a black box tool to test the solution.

Combinatorial Optimization component

evaluation

System Simulator

solution

Can be very expensive
In principle.....
we can have a combinatorial problem solver
and a black box tool to test the solution

Scatter search +
predictors of the
simulation outcome

Can be very expensive

System
Simulator

evaluation

solution

Note: these predictors enable a selection of solutions to be sent to the simulator

[OptQuest M. Laguna, 2011]
In principle……
we can have a combinatorial problem solver
and a black box tool to test the solution

What if the black box tool is a machine learning model?

Combination Optimization component

---

Machine learning model

Evaluation

Solution

Can be expensive to train but after training it is very efficient to use

Surrogate models: [Henao, Maravelias, AIChE 2011], [Cozad, Sahinidis, Miller, 2014]
In principle……
we can have a combinatorial problem solver
and a black box tool to test the solution

What if the black box tool is a machine learning model?

Still….. a generate and test mechanism
Empirical Model Learning – EML

In EML the learned model is embedded into the optimization component and used to actively reduce the search space during execution.
Empirical Model Learning – EML

In EML the learned model is embedded into the optimization component and used to actively reduce the search space during execution.

*Not limited to objective functions, but also constraints and any relation between decidables and observables*

Lombardi, Milano, Bartolini, Empirical Decision Model Learning, AIJ (244), 2017
Empirical Model Learning – EML

What is the difference between EML and the traditional use of ML models?

Traditional approaches work forward
Empirical Model Learning – EML

What is the difference between EML and the traditional use of ML models?

*EML works forward and backward*

The ML model becomes a constraint: given temperature limits we remove combinations of workload decisions that lead to inconsistent temperatures
Thermal-aware workload dispatching

COMBINATORIAL COMPONENT

Given n tasks m cores

\[ \sum_j \sum_i x_{ij} = 1 \]

\[ \sum_i x_{ij} = \frac{n}{m} \quad \text{Load balancing} \]

Other constraints (e.g. scheduling)

\( x_{ij} \) is 1 if task \( i \) is allocated on core \( j \)

MACHINE LEARNING MODEL

One NN per core

\[ \min \left( \max \text{temp}_j \right) \]

\[ \sum_j \sum_i x_{ij} \leq 1 \]

\[ \sum_i x_{ij} = \frac{n}{m} \]

\( \text{EM}(x, \text{temp}) \)

Other constraints (e.g. scheduling)

\( x_{ij} \) is 1 if task \( i \) is allocated on core \( j \)
Empirical Model Learning – EML

The questions are:

1) how do we embedded Machine learning models into a combinatorial model?

2) How do we provide operational semantics to these models so that they can actively prune the search space?

We will explore:

• Neural Networks
• Decision Trees

In different optimization techniques: CP in the talk, but also MINLP/ILP, SMT
Artificial Neural Networks

- Computational model, consisting of parallel computation units (neurons) connected in a network structure
- Usually: no cycles (feed-forward) and multiple layers
Artificial Neural Networks

Artificial Neuron = scalar function with vector input

- "a" is the neuron activity
- "f" is the activation function
- weights are obtained in the learning phase

The activation function is a **monotonic non-decreasing function** of the neuron activity

\[ a = \sum_i w_i \cdot x_i + b \]

\[ y = f(a) \]

Neural Networks in a Combinatorial Model

A Neural Network is a declarative (nonlinear) model

• Can be embedded in a MINLP model directly

1. We should insert variables for NN inputs and outputs in the model
2. Insert the NN non linear equations in the model
Neural Networks in a Combinatorial Model

A Neural Network is a declarative (nonlinear) model

- Can be embedded in a CP model via variables and constraints
- Here: artificial neuron = a global "neuron constraint"

\[
\text{act.function}([X], Y, [W], b)
\]

**Input variables**

**Output variable**

**Weights**

**Bias**
Structure of the Neuron Constraint

- An internal variable $A$ for the neuron activity
- Bound consistency on the equality: $A = \sum_i w_i \cdot X_i + b$

Filtering for the activation function:

1. $\max(A)$ updated $\Rightarrow \max(Y) = f(\max(A))$
2. $\max(Y)$ updated $\Rightarrow \max(A) = f^{-1}(\max(Y))$

- Similar rules for minima
- The value $f^{-1}(\max(Y))$ is replaced by $\max\{a' \mid f(a') = \max(Y)\}$ to avoid precision errors
- Filtering from $Y$ to $A$ and from $A$ to $Y$

We can combine many neuron constraints in an artificial neural network global constraint: $\text{ann}(\text{output}, \text{inputs})$
How it works at search time

Allocation Variables

Tasks

NN In Vars

NN Out Vars

NN In Vars

NN Out Vars

NN Out Vars

NN In Vars

Feature computation

temp

acpi_3

acpi_2

acpi_1

acpi_0

Feature computation

temp

acpi_3

acpi_2

acpi_1

acpi_0

Feature computation

temp

acpi_3

acpi_2

acpi_1

acpi_0

Feature computation

temp

acpi_3

acpi_2

acpi_1

acpi_0
How it works at search time

Feature computation

Neuron Constraints

Feature computation

acpi_3
acpi_2
acpi_1
acpi_0

acpi_3
acpi_2
acpi_1
acpi_0

acpi_3
acpi_2
acpi_1
acpi_0

acpi_3
acpi_2
acpi_1
acpi_0

acpi_3
acpi_2
acpi_1
acpi_0

acpi_3
acpi_2
acpi_1
acpi_0

acpi_3
acpi_2
acpi_1
acpi_0
Forward Propagation

Neuron Constraints

Feature computation

Feature computation

temp
acpi_3
acpi_2
acpi_1
acpi_0
temp
acpi_3
acpi_2
acpi_1
acpi_0
temp
acpi_3
acpi_2
acpi_1
acpi_0
temp
acpi_3
acpi_2
acpi_1
acpi_0
temp
acpi_3
acpi_2
acpi_1
acpi_0
Backward propagation
EML: CP model

Problem data:
- n tasks (with CPI values)
- Each task is assumed to run indefinitely
- m = 48 cores
- Room temperature

Decision variables:
\[ \text{CORE}_i \in \{0..m - 1\} \quad \forall t_i \]

Balancing constraints:
Roughly the same number of tasks per core
\[ gcc([\text{CORE}_i], \{0..m - 1\}, \lfloor n/m \rfloor, \lceil n/m \rceil) \]
Equivalent allocation variables: 
\[ X_{i,j} = 1 \Leftrightarrow \text{CORE}_i = j \]

Average CPI per core (FEATURE COMPUTATION): 
\[ \text{average}(\text{ACPI}_j, [\text{CPI}_i], [X_{i,j} | j=k]) \quad \forall \text{ core } k \]

Neural Network: collection of act_function constraints
\[ \text{ann}_j(ctemp_j, \text{ACPI}_j, \text{ACPI}_{n\text{gbr}} \ldots, rtemp) \]

Objective:
\[ \min_{j \in \text{cores}} \ \max_{j} ctemp_j \]

Bartolini, Lombardi, Milano, Benini, Optimization and Controlled Systems: A Case Study on Thermal Aware Workload Dispatching, AAAI 2012
Results

Solution Approach:
- Model solved via Large Neighborhood Search (LNS)
- First solution via a CPI balancing heuristic

Heat generated on the cores by MILP using an expert design model  
Heat generated on the cores by the EML approach
How to cast a decision tree in a combinatorial model

Three encodings:
- Meta-constraints
- Table constraint
- Other encodings

Extension: Random Forests

Bonfietti, Lombardi, Milano: Embedding Decision Trees and Random Forests in CP, CPAIOR 2011
Decision Tree

- Input: tuple of attribute values
- Output: a class

```
+-------------------------+    +-------------------------+
| attribute values        | →  | class labels            |
| +------------------------+    | +------------------------+    +------------------------+
| A                       |    | numeric attribute       |    | symbolic attribute     |
| B ≤ 20                  |    | branch labels           |    | branch labels           |
| C                       |    | class labels            |    | class labels            |
| A ≤ 10                  |    |                            |    |                          |
| B > 10                  |    |                            |    |                          |
| C 0                     |    |                            |    |                          |
| A ≤ 10                  |    |                            |    |                          |
| B > 10                  |    |                            |    |                          |
| C 0                     |    |                            |    |                          |
| A ≤ 10                  |    |                            |    |                          |
| B > 10                  |    |                            |    |                          |
| C 0                     |    |                            |    |                          |
```
Classification example

A: 30, B: 20, C: 1
Classification example

A: 30, B: 20, C: 1
Classification example

A: 30, B: 20, C: 1
class = 1
How do we embed a DT in CP?

- A decision variable for each attribute
  
  \[ A \in \{-\infty, \infty\} \]
  
  \[ B \in \{-\infty, \infty\} \]
  
  \[ C \in \{0, 1\} \]

- A decision variable for the class

  \[ Y \in \{0, 1\} \]

- Enforce consistency on:

  \[ Y = DT(A, B, C) \]

  This is the tricky part
Encoding #1

[A > 20] \land [C = 0] \land [B > 20] \Rightarrow [Y = 1]

A path is an implication
A path is an implication

The tree defines all the paths

\[( [A > 20] \land [C = 0] \land [B > 20] ) \lor

[Y = 1] \Leftrightarrow ( [A > 20] \land [C = 1] ) \lor

( [A \leq 20] \land [B \leq 10] \land [C = 1] \land [A > 10] )\]
A path is an implication.
The tree defines all the paths.

- Polynomial size
- Polynomial time propagation
- Does not enforce GAC
- Very suitable for SMT

Can we do better?
We can use a table constraint!

Bessiere, Regin, IJCAI 97
Continuous attributes can be discretised using the splitting thresholds appearing in the tree.

Path ↔ set of feasible assignments
Solution quality

- 20 instances, 48 cores, 288 jobs
- CP with LNS against Localsolver, 90 sec time limit
More compact encodings:
• use an MDD
• use compressed tuples
• convert to a sNNF - sdNNF
• decomposition approaches

Variants
• Random forests
• Regression trees
• Multi-output trees
In a nutshell:

**EML enables combinatorial optimization over complex real world systems**

- It’s not the only solution…
- …and it does not need to be about CP

**But CP is very well suited for EML!**

- Ease of embedding (a EM in a global constraint)
- Flexibility and modularity (the EM is “just a constraint”)
In a nutshell:

**EML enables combinatorial optimization over complex real world systems**

- It’s not the only solution…
- …and it does not need to be about CP

**But CP is very well suited for EML!**

- Ease of embedding (a Neural Network in a global constraint)
- Flexibility and modularity (the EM is “just a constraint”)

Which issues shall we address to make it work?
Issue #1

- You can always embed a ML in (e.g.) Local Search
- Can CP work better?

Yes (we have data)! Mainly thanks to propagation
Issue #1

- You can always embed a ML in (e.g.) Local Search
- Can CP work better?

Yes (we have data)! Mainly thanks to propagation

We need **inference methods** for ML models
(e.g. bounds on the I/O of a Neural Network)
- Inference should be **effective** (tight bounds)
- Inference should be **efficient**
<table>
<thead>
<tr>
<th>Issue #2</th>
<th>Complex ML models</th>
<th>Simple ML models</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>more accurate</td>
<td>less accurate</td>
</tr>
<tr>
<td></td>
<td>slow, weak propagation</td>
<td>effective propagation</td>
</tr>
<tr>
<td>Risk:</td>
<td>poor quality solutions</td>
<td>apparently good solutions</td>
</tr>
</tbody>
</table>
Issue #2

Complex ML models
- more accurate
- slow, weak propagation

Simple ML models
- less accurate
- effective propagation

Risk: poor quality solutions
Risk: apparently good solutions

There is an **accuracy/optimization trade-off**
- How to characterize it?
- Can we find design rules?

We are currently working on Deep NN with a Google Faculty Award
• Very interesting approach by Fischetti and Jo:
• They model a DNN with fixed weights in MIP
• They define upper bound tightening procedures as follows
  – They consider units layer by layer
  – For the current unit, remove from the model all the constraints and variables related to all other units in the same layer or in subsequent layers
  – Solve twice the resulting model for min and max the output

M. Fischetti, Jason Jo, Deep neural networks and mixed integer linear optimization, Constraints 23, 2018
Issue #3

A typical training set in ML looks like this:
Issue #3

The ML model provides a prediction for every input value.
Issue #3

The optimizer will **search for the best one** (HP: prediction = cost)
If this is far from known examples, there may be a large error.
If this is far from known examples, there may be a large error.

How do we choose representative observations?

Borrow concepts from black-box optimization, surrogate model construction by adaptive sampling.
Issue #4

Many ML models provide estimated confidence levels.

Can we take them into account when optimizing?
Issue #4

Many ML model provide estimated confidence levels

Can we take them into account when optimizing?

Some ML models can deal with uncertain inputs

Can we employ EML in stochastic optimization?

Some errors are more important than other
Using ML to solve the problem

Aimed at learning models in the general form:

\[
\begin{align*}
\min \ z &= x_0 \\
\text{subject to: } \nu_m(\vec{x}_{m,in}, \vec{x}_{m,\text{out}}) &\quad \forall m \in M \\
\vec{x} &\in D_{\vec{x}}
\end{align*}
\] (P3)

- where \( \vec{x} \) are problem variables, \( D_{\vec{x}} \) their domain and \( x_0 \) is the variable representing the objective value,
- and \( \nu_m(\vec{x}_{m,in}, \vec{x}_{m,\text{out}}) \) is a proper encoding of a machine learning model in the hosting language.
Aimed at learning models in the general form:
Aimed at learning models in the general form:
ML for Search

A. Lodi and G. Zarpellon On learning and Branching: a survey, TOP 2017
Search strategies

• In tree-based search strategies: two fundamental steps
  – Variable selection: on which variable to branch
  – Node selection: which value to assign to the variable
    Some work on ML applied to MIP to guide these choices

• Heuristic search strategies are more diverse
  – Large neighborhood search
  – Heuristic search
ML in MIP search

• In MIP the heuristics for selecting a variable is sometimes based on Strong Branching and variants
  – ML can be used to approximate SB by means of supervised learning technique
    – Feature extraction to describe the branching variable
    – A regressor is learned to predict the estimated SB value
    – The regressor is used as a branching heuristics

• Features should be
  – Size independent
  – Invariant to changes within the instance
  – Scale-independent (if parameter change)
ML in MIP search

- Experimented on MIPLIB
  - $10^5$ observations
  - On MIPLIB the new search strategy imitates full Strong Branching—modest improvement
- Learning within the same instance
- SB scores are only used to select the single best variable
- Learning to imitate the correct ranking of the candidate variables at each node

Deep networks assisted heuristic tree search

• DLTS is an incomplete tree search in which decisions about which branches to explore and how to bound nodes are made by a DNN
  – DFS, LDS, weighted Beem search
  – Plus aggressive pruning

• The DNN is problem specific

• Two networks:
  – a policy network to make predictions on the best branch
  – a value network to predict the cost of completing the solution

A.Hottung, S.Tanaka, K.Tierney
Deep Learning Assisted Heuristic Tree Search for the Container Pre-marshalling Problem
arXiv:1709.09972
Deep networks assisted heuristic tree search

A.Hottung, S.Tanaka, K.Tierney
Deep Learning Assisted Heuristic Tree Search for the Container Pre-marshalling Problem
arXiv:1709.09972
ML in MIP search

• Instead of an off-line ML method, an online learning strategy could be employed
  – Data generated on-the-fly within the search process itself
  – Reliability: depending on the number of times a real SB score was computed for a given var one could deem the candidate reliable
  – Drawback: not adapting over time (after all var are reliable)
  – Perpetual version of the method

ML in MIP search

• Again on learning during search in MIP

• Two general branching strategies:
  – Strong Branching (SB): simulate branching on multiple variables; select var. with max product of LP increase large computation time, small number of nodes
  – Pseudocost Branching (PC): choose based on running averages of observed branching effect for each variable a lot of manual tuning, small time, reasonable number of nodes

• Can we get the best of both worlds (SB and PC)?
  – small number of nodes
  – small time
  – as little manual tuning as possible

ML in MIP search

• Relaxed Binary Labels:
  – $y_{ij} = 1$; if SB score of $x_j$ is within $\alpha$ of max SB score at $N_i$
  – $y_{ij} = 0$; otherwise
  where $\alpha$ in $(0,1)$ is small

• Goal: Learn a function of the features that ranks variables with better labels higher than other variables.

• Note that for data collection: run SB for some nodes, and use SB scores as labels for variables within each node; compute features describing each variable (dataset D)

• Also decide if run the primal heuristics or not.

ML in MIP search

• Reinforcement learning:
  – Within a node selection stage the tree is traversed from root to leaf (best first)
  – Once the leaf is reached the tree is updated: computed for the leaf and propagated upward

  \[
  \text{Score}(N) = \text{estimate}(N) + \Gamma \frac{\text{visits}(P)/100}{\text{visits}(N)}
  \]

  Measure of node quality\hspace{1cm} \text{Exploration/exploration balance}

  P: parent node\hspace{1cm} N: current node

A. Sabharwal, H. Samulowitz, H. Reddy, "Guiding combinatorial optimization with UCT". CPAIOR 2012
ML in MIP search

• The essential part of the reward measure consists of LP objective values combined with visit counters
• Good tradeoff between exploration and exploitation.
Design of heuristic algorithms on graphs

- Research question: Given a graph optimization problem G and a distribution of problem instances, can we learn better heuristics that generalise to unseen instances?
- Approach:
  - adopt a greedy meta-algorithm design
  - Use a graph embedding network (deep network) to represent the policy in the greedy algorithm
    - The network features nodes capturing their properties
    - Parametrises the evaluation function
  - Q-learning to learn a greedy policy

Design of heuristic algorithms on graphs

- To estimate the quality of the partial solution $S'$ resulting from adding a node to $S$, we use an evaluation function $Q$ which is learned using a collection of problem instances

  $$\hat{Q}(h(S), v; \Theta)$$

- $Q$ is based on an estimation of the state $S$, adding node $v$ and based on parameters $\Theta$

- Parameters are tuned via a deep network

---

Graph embedding

• The graph embedding network provides a p-dimensional feature embedding for each node, given the current partial solution $S$
• Updated at each node of the search tree
• After few steps of recursion the network will produce a new embedding for each node considering
  – Node characteristics
  – Long range interactions between node features
• Use node embedding to update the evaluation function $Q$ of the partial solution

Graph embedding

Q-learning

- We would like to learn an evaluation function $Q$ across a set of $m$ graphs from distribution $D$ (different size)
- Reinforcement learning formulation
  - State: sequence of nodes
  - Transition: tagging a node selected by the last action
  - Action: an action is a node that is not part of the current state
  - Rewards: $r(S,v)$ is the change in the cost function after taking action $v$ and transitioning to $S'$
    \[
    r(S,v) = c(h(S'), G) - c(h(S), G),
    \]
  - Policy: deterministic greedy policy
    \[
    \pi(v|S) := \arg\max_{v' \in \mathcal{S}} \hat{Q}(h(S), v')
    \]
Q-learning

• The update of the Q function is in general performed when a full solution has been built (delayed reward)
• Updating it at every step too miopic
• Tradeoff: update after n steps

• Overall very interesting performances.

Goal: Generality of MIP + Flexibility of Feature Learning
ML for Portfolio Selection and Algorithm configuration

Material extracted by the presentation of Lars Kotthoff at the ACP Summer School, Jackson, 05 June 2018

Algorithm selection

- Different algorithms have different performances on different problems
- .....even on different instances of the same problem
- often no clear winner among different approaches

we need an algorithm performance estimator
Algorithm selection

- Different algorithms have different performances on different problems
- …..even on different instances of the same problem
- Often no clear winner among different approaches

We need an algorithm performance estimator
Performance models

• Performance models of black-box processes

• also called surrogate models

• Replace expensive underlying process with cheap approximate model

• Build the approximation on the basis of real evaluation via machine learning techniques

• No knowledge about the underlying process
Algorithm selection
Algorithm portfolios

• Instead of a single algorithm use several complementary algorithms

• Idea from Economics – minimize the risk by spreading it out across several securities

• For computational problems – minimize the risk of algorithm performing poorly

• Algorithm used in a loose sense (constraint solvers, search strategies, modeling choices, different propagation algos)
Algorithm portfolios

• Most approaches rely on machine learning

• Train with representative data (training set) – performance for all algorithms in a portfolio

• Evaluate performance on a test set (unseen instances)

• Need to balance easy/hard instances on both sets

• May need hundreds/thousands of instances
Algorithm selection ingredients

- Feature extraction
- Performance evaluator
- Prediction based selector/scheduler
Feature extraction

- Syntactic features: analyse instance description
  - Number of vars/constraints/clauses
  - Ratios
  - Graph properties (node degree, connectivity)
- Probing features: run algorithms for short time
  - Number of nodes/propagation within time limit
  - Estimated search space size
- Dynamic features: instance changes while algo running
  - Number of constraint propagation
  - Change in var domains
Feature extraction

– Syntactic features: analyse instance description
  • Number of vars/constraints/clauses
  • Ratios

– Probing features: run algorithms for short time
  • Number of nodes/propagation within time limit
  • Estimated search space size

– Dynamic features: instance changes while algo running
  • Number of constraint propagation
  • Change in var domains

**Which features?**
Mix of complex/simple features
In the end.....whatever works best
Performance evaluator

- Model for entire portfolios
  - Predict the best algorithm in the portfolio (classification)
  - Clustering and assign algorithms to clusters
  - Important to assign a weight during learning (difference best-worst time) to focus on important instances


Performance evaluator

– Model for individual algorithms
  • Predict the performance of each algorithm
  • Compare predictions and chose the best


– Hybrid methods, e.g. pairwise comparisons

Xu, Lin, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. “Hydra-MIP: Automated Algorithm Configuration and Selection for Mixed Integer Programming.” In RCRA@IJCAI11

Prediction selector

– Best algo
– N best algorithms ranked
– Allocation of resources to algorithms
– Change the current one?


Algorithm configuration

– Given a set of problems, find the best algorithm parameter configuration

– Parameters:
  • Anything you can change
    – search heuristic, variable ordering ….
  • Some will affect performances
Algorithm configuration

Model based algorithm configuration

- Evaluate small number of configurations
- Build model of parameter-performance surface based on the results
- Use model to predict where to evaluate next
- Repeat
- Allows targeted exploration of new configurations
- Can take instance features into account like algorithm selection
- Risk of overtuning (similar to overfitting)

To wrap up
Thanks for listening!
Questions?
Alamo – Sahinidis et al.

Start

Initial sampling

Build surrogate model

Update training data set

Adaptive sampling

Model converged?

true

Stop

false
Alamo – Sahinidis et al. adaptive sampling

**Model i**
- Surrogate model
- Black-box function
- Data points

**Sample Points**
- Model error
- New sample point

**Model i+1**
- New surrogate model

Error maximization

Rebuild model