Constraint Learning

Stefano Teso (with some slides by Luc De Raedt and Andrea Passerini)

KU Leuven

@ Reasoning Web Summer School '19, Bolzano

Why constraints?

Constraints are **ubiquitous** in AI and OR

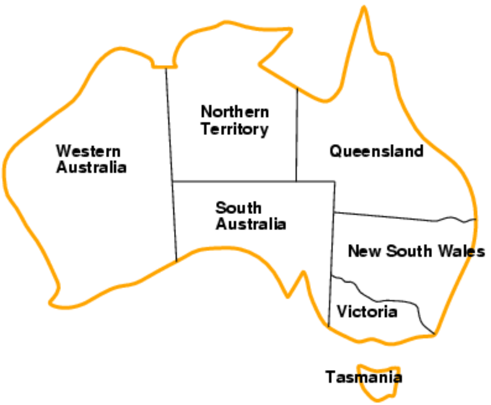
Perhaps the two most common formalisms are:

- constraint satisfaction (**CSP**)
- linear programming (**LP**)

... and all their extensions

Especially common in declarative approaches to problem solving:
define **specification** *of the problem, let solver do the heavy lifting*

Example: Map Coloring

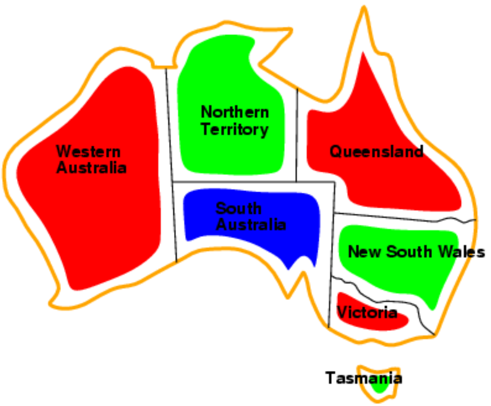


- Vars: WA, NT, Q, NSW, V, SA, T
- Domains: {red, green, blue}

- Constraints: *adjacent regions must have different colors*, e.g., WA $\neq$ NT

(Credit: Marriot & Stuckey)

Example: Map Coloring



A **solution** is a **complete** and **consistent** assignment, e.g., WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green

Notice that it may not be unique!

Example: Sudoku

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

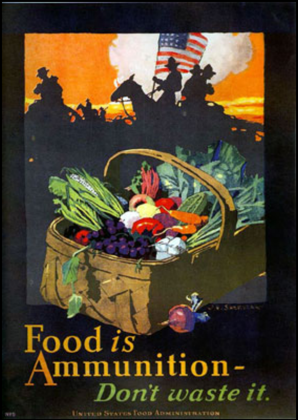
no repeated numbers in any row, column, or 3×3 *square*

Example: Sudoku

```
array[1..N,1..N] of var PuzzleRange: puzzle;

% All different in rows
constraint forall (i in PuzzleRange) (
    alldifferent( [ puzzle[i,j] | j in PuzzleRange ]) );

% All different in columns.
constraint forall (j in PuzzleRange) (
    alldifferent( [ puzzle[i,j] | i in PuzzleRange ]) );

% All different in sub-squares:
constraint forall (a, o in SubSquareRange)(
    alldifferent( [ puzzle[(a-1) * S + a1, (o-1) * S + o1] |
                    a1, o1 in SubSquareRange ] ) );

solve satisfy;
```

Using **MiniZinc**: `https://www.minizinc.org/`

Example: Stigler's Diet problem



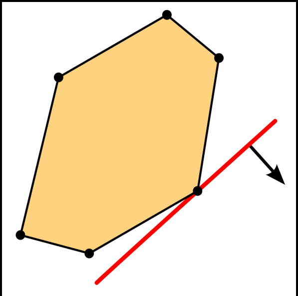
x_i = amount of food i in dietc_i

a_{ij} = **amount** of nutrient j in food i (20g protein / burger)

$$
\begin{aligned}
\min_{x} \quad & \sum_{i \in \mathcal{F}} c_i x_i & \\
\text{s.t.} \quad & \sum_{i \in \mathcal{F}} a_{ij} x_i \geq \text{minnutr}_j & \forall j \in \mathcal{N} \\
& \sum_{i \in \mathcal{F}} a_{ij} x_i \leq \text{maxnutr}_j & \forall j \in \mathcal{N} \\
& \text{minserve}_i \leq x_i \leq \text{maxserve}_i & \forall i \in \mathcal{F}
\end{aligned}
$$

given nutrient information and cost per serving, select the number of servings of each food so as to (1) minimize the total cost, while (2) meeting nutritional requirements, i.e. min / max level of nutritional component [Sti45] (actively studied [vD18])

Example: Stigler's Diet problem



$$\boldsymbol{x}, \boldsymbol{c} \in \mathbb{R}^{|\mathcal{F}|}, \boldsymbol{b} \in \mathbb{R}^{|\mathcal{N}|}, A \in \mathbb{R}^{|\mathcal{F}| \times |\mathcal{N}|}$$

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) = \sum_{i \in \mathcal{F}} c_i x_i$$

$$\text{s.t. } A\boldsymbol{x} \leq \boldsymbol{b}$$

A **linear program** in standard form: the constraints $A\boldsymbol{x} \leq \boldsymbol{b}$ implicitly define a (possibly unbounded) **feasible polytope**, while $\boldsymbol{c}$ defines a linear **objective function** f over it

The polytope can be viewed as the intersection of $|\mathcal{N}|$ hyperplanes

$$\mathrm{Sol}(A, \boldsymbol{b}) = \{\boldsymbol{a}_j \cdot \boldsymbol{x} \leq b_j : j = 1, \ldots, |\mathcal{N}|\}$$

Why constraint learning?

Constraints are **ubiquitous** in AI and OR: *define problem specification, feed it to a solver*

Why constraint learning?

Constraints are **ubiquitous** in AI and OR: *define problem specification, feed it to a solver*

…but formalizing the problem is **hard**!!!

- Most users are *not* modelling experts
- Often requires interaction between domain and modelling experts (going back & forth, plenty of debugging)
- Experts do *not* work for free

This **hinders adoption** of smart and efficient solution techniques, makes decision making **harder than it needs to be**

Constraint learning

If past working and non-working solutions are available, **acquire a model from them**!

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

```
 array[1..N,1..N] of var PuzzleRange: puzzle;

 % All different in rows
 constraint forall (i in PuzzleRange) (
     alldifferent( [ puzzle[i,j] | j in PuzzleRange ]) );

 % All different in columns.
 constraint forall (j in PuzzleRange) (
     alldifferent( [ puzzle[i,j] | i in PuzzleRange ]) );

 % All different in sub-squares:
 constraint forall (a, o in SubSquareRange)(
     alldifferent( [ puzzle[(a-1) * S + a1, (o-1) * S + o1] |
                     a1, o1 in SubSquareRange ] ) );

solve satisfy;
```

Note: CL is an form of **machine learning**, link discussed later on

Learning Bundesliga scheduling rules [BS16]

2	-1	4	-3	6	-5	8	-7	10	-9	12	-11	14	-13	16	-15	18	-17
-8	15	-10	7	-18	9	-4	1	-6	3	-14	13	-12	11	-2	17	-16	5
4	-17	14	-1	12	-13	10	-15	16	-7	18	-5	6	-3	8	-9	2	-11
7	11	-8	15	-16	-18	-1	3	-12	13	-2	9	-10	17	-4	5	-14	6
-3	-13	1	-11	10	16	-15	-17	14	-5	4	-18	2	-9	7	-6	8	12
15	9	-7	13	-14	-12	3	11	-2	17	-8	6	-4	5	-1	18	-10	-16
-17	-5	-15	-18	2	14	-9	-13	7	-11	10	-16	8	-6	3	12	1	4
11	18	5	9	-3	-10	17	12	-4	6	-1	-8	-15	16	13	-14	-7	-2
-13	-6	-17	-5	4	2	-11	-9	8	-16	7	14	1	-12	-18	10	3	15
9	16	11	6	-8	-4	13	5	-1	12	-3	-10	-7	18	17	-2	-15	-14
-5	-12	-13	-16	1	7	-6	-18	15	-14	17	2	3	10	-9	4	-11	8
6	14	9	12	-7	-1	5	16	-3	18	-15	-4	-17	-2	11	-8	13	-10
-18	-10	-12	-14	15	8	-16	-6	17	2	13	3	-11	4	-5	7	-9	1
12	-7	18	10	-17	-15	2	14	-11	-4	9	-1	16	-8	6	-13	5	-3
-14	4	-16	-2	11	17	-18	-10	13	8	-5	15	-9	1	-12	3	-6	7
10	-8	6	-17	-9	-3	12	2	5	-1	16	-7	18	-15	14	-11	4	-13
-16	3	-2	8	13	11	-14	-4	-18	15	-6	17	-5	7	-10	1	-12	9
-2	1	-4	3	-6	5	-8	7	-10	9	-12	11	-14	13	-16	15	-18	17
8	-15	10	-7	18	-9	4	-1	6	-3	14	-13	12	-11	2	-17	16	-5
-4	17	-14	1	-12	13	-10	15	-16	7	-18	5	-6	3	-8	9	-2	11
-7	-11	8	-15	16	18	1	-3	12	-13	2	-9	10	-17	4	-5	14	-6
3	13	-1	11	-10	-16	15	17	-14	5	-4	18	-2	9	-7	6	-8	-12
-15	-9	7	-13	14	12	-3	-11	2	-17	8	-6	4	-5	1	-18	10	16
17	5	15	18	-2	-14	9	13	-7	11	-10	16	-8	6	-3	-12	-1	-4
-11	-18	-5	-9	3	10	-17	-12	4	-6	1	8	15	-16	-13	14	7	2
13	6	17	5	-4	-2	11	9	-8	16	-7	-14	-1	12	18	-10	-3	-15
-9	-16	-11	-6	8	4	-13	-5	1	-12	3	10	7	-18	-17	2	15	14
5	12	13	16	-1	-7	6	18	-15	14	-17	-2	-3	-10	9	-4	11	-8
-6	-14	-9	-12	7	1	-5	-16	3	-18	15	4	17	2	-11	8	-13	10
18	10	12	14	-15	-8	16	6	-17	-2	-13	-3	11	-4	5	-7	9	-1
-12	7	-18	-10	17	15	-2	-14	11	4	-9	1	-16	8	-6	13	-5	3
14	-4	16	2	-11	-17	18	10	-13	-8	5	-15	9	-1	12	-3	6	-7
-10	8	-6	17	9	3	-12	-2	-5	1	-16	7	-18	15	-14	11	-4	13
16	-3	2	-8	-13	-11	14	4	18	-15	6	-17	5	-7	10	-1	12	-9

J	Scheme	Ref	Trans	Constraint
1	scheme(612,34,18,1,18)	284	absolute_value	symmetric_alldifferent([1..18])*34
2	vector(612)	289	id	global_cardinality([-18.. -1-17,0-0,1..18-17])*1
3	scheme(612,34,18,34,1)	288	id	alldifferent*18
4	repart(612,34,18,17,18)	282	id	alldifferent*306
5	scheme(612,34,18,2,2)	286	id	alldifferent*153
6	scheme(612,34,18,1,18)	284	id	alldifferent*34
7	repart(612,34,18,34,9)	283	sign	alldifferent*306
8	scheme(612,34,18,17,1)	287	absolute_value	alldifferent*36
9	scheme(612,34,18,2,1)	285	absolute_value	alldifferent*306
10	repart(612,34,18,34,9)	283	id	sum_ctr(0)*306
11	repart(612,34,18,34,9)	283	id	sum_cubes_ctr(0)*306
12	scheme(612,34,18,1,18)	284	id	sum_squares_ctr(2109)*34
13	repart(612,34,18,34,9)	283	id	twin*1
14	repart(612,34,18,34,9)	283	id	elements([i,-i])*1
15	modulo(612,4)	281	id	all_differ_from_at_least_k_pos(152)*1
16	first(9,[1,3,5,7,9,11,13,15,17])	280	id	strictly_increasing*1
17	repart(612,34,18,34,9)	283	id	alldifferent_interval(2)*306
18	scheme(612,34,18,2,1)	285	id	alldifferent_interval(2)*306
19	repart(612,34,18,34,9)	283	sign	sum_ctr(0)*306
20	scheme(612,34,18,1,18)	284	sign	sum_ctr(0)*34
21	repart(612,34,18,34,9)	283	sign	twin*1
22	repart(612,34,18,34,9)	283	absolute_value	twin*1
23	repart(612,34,18,34,9)	283	sign	elements([i,-i])*1
24	repart(612,34,18,34,9)	283	absolute_value	elements([i,i])*1
25	first(9,[1,3,5,7,9,11,13,15,17])	280	absolute_value	strictly_increasing*1
26	first(6,[1,4,7,10,13,16])	279	absolute_value	strictly_increasing*1
27	repart(612,34,18,34,9)	283	sign	alldifferent_interval(2)*306
28	scheme(612,34,18,34,1)	288	sign	among_seq(3,[-1])*18

learn a constraint satisfaction model for Bundesliga team scheduling from the data of a **single** *season*

Learning spreadsheet formulas with TaCLe [KPGDR17]

T1

ID	Salesperson	1st Quarter	2nd Quarter	3rd Quarter	4th Quarter	Total	Rank	Label	Items sold total	Max items sold
1	Diana Coolen	353	378	396	387	1514	2	Great	34	20
2	Marc Desmet	370	408	387	386	1551	1	Great	29	10
3	Kris Goossens	175	146	167	203	691	3	Low	19	19
4	Birgit Kenis	93	98	96	105	392	4	Low	17	15

Block 1 = T1[:, 1]; Block 2 = T1[:, 2]; Block 3 = T1[:, 3:8]; Block 4 = T1[:, 9]; Block 5 = T1[:, 10:11]

T2

Total	991	1030	1046	1081	4148
Average	247.75	257.5	261.5	270.25	1037
Max	370	408	396	387	1551
Min	93	98	96	105	392

Block 6 = T2[1:4, :]

T3

Salesperson	Items sold
Diana Coolen	5
Marc Desmet	10
Marc Desmet	8
Diana Coolen	9
Birgit Kenis	15
Marc Desmet	8
Birgit Kenis	2
Diana Coolen	20
Marc Desmet	3
Kris Goossens	19

Block 8 = T3[:, 1]; Block 9 = T3[:, 2]

T4

Quarter	Income	Expenses	Total
Q1	991	212	779
Q2	1030	710	1099
Q3	1046	137	2008
Q4	1081	240	2849

Block 10 = T4[:, 1]; Block 11 = T4[:, 2:4]

T5

Customer	Contact	Contact Name
Frank	1	Diana Coolen
Sarah	3	Kris Goossens
George	3	Kris Goossens
Mary	2	Diana Coolen
Tim	4	Birgit Kenis

Block 12 = T5[:, 1]; Block 13 = T5[:, 2]; Block 14 = T5[:, 3]

$SERIES(T_1[:, 1])$

$T_1[:, 1] = RANK(T_1[:, 5])^*$

$T_1[:, 1] = RANK(T_1[:, 6])^*$

$T_1[:, 1] = RANK(T_1[:, 10])^*$

$T_1[:, 8] = RANK(T_1[:, 7])$

$T_1[:, 8] = RANK(T_1[:, 3])^*$

$T_1[:, 8] = RANK(T_1[:, 4])^*$

$T_1[:, 7] = SUM_{row}(T_1[:, 3{:}6])$

$T_1[:, 10] = SUMIF(T_3[:, 1], T_1[:, 2], T_3[:, 2])$

$T_2[1, :] = SUM_{col}(T_1[:, 3{:}7])$

$T_2[2, :] = AVERAGE_{col}(T_1[:, 3{:}7])$

$T_2[3, :] = MAX_{col}(T_1[:, 3{:}7])$

$T_2[4, :] = MIN_{col}(T_1[:, 3{:}7])$

$T_4[:, 2] = SUM_{col}(T_1[:, 3{:}6])$

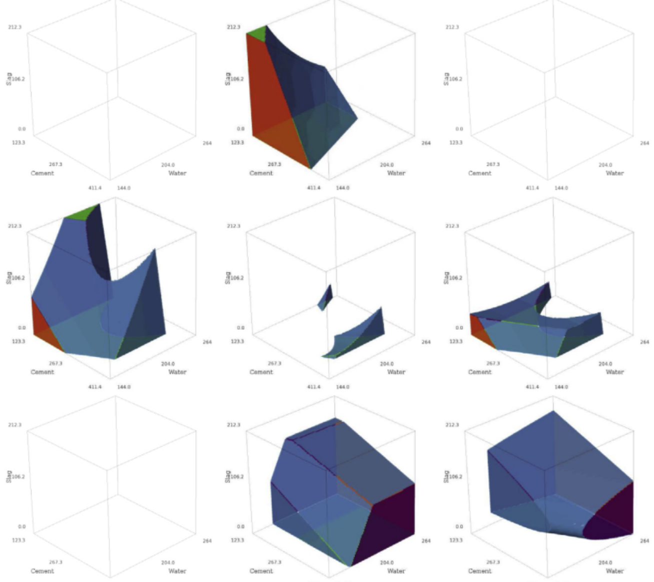
$T_4[:, 4] = PREV(T_4[:, 4]) + T_4[:, 2] - T_4[:, 3]$

$T_5[:, 2] = LOOKUP(T_5[:, 3], T_1[:, 2], T_1[:, 1])^*$

$T_5[:, 3] = LOOKUP(T_5[:, 2], T_1[:, 1], T_1[:, 2])$

$T_1[:, 11] = MAXIF(T_3[:, 1], T_1[:, 2], T_3[:, 2])$

Learning Concrete Mixing from Positive-only data [PK17]



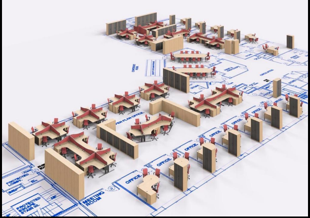
Learning to Synthesize [DTP18]

A user wishes to buy a custom PC. The PC is assembled from individual components: CPU, HDD, RAM, etc. Valid PC configurations must satisfy constraints, e.g. CPUs only work with compatible motherboards [TDP17]

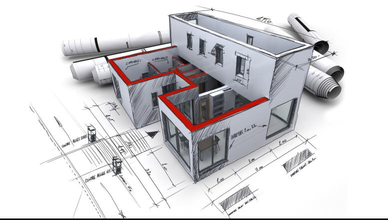
Hard: "Intel CPUs are incompatible with AMD motherboards"

Soft: "The user prefers one CPU over another"

Learning to Synthesize [DTP18]



Interior design



Building design

Urban planning

Dimensions of Constraint Learning

- Types of constraints:
  - **hard** constraints define the set of valid assignments; used in SAT, LP, CP, answer set programming, ...
  - **soft** constraints define preferences among valid assignments; used in LP, all of operations research

Dimensions of Constraint Learning

- Types of constraints:
  - **hard** constraints define the set of valid assignments; used in SAT, LP, CP, answer set programming, . . .
  - **soft** constraints define preferences among valid assignments; used in LP, all of operations research
- Learning techniques:
  - **search-based**: smartly enumerate the candidate theories and pick one that best matches the data
  - **solver-based**: encode the learning problem as a satisfaction or optimization problem and feed it to a solver

Dimensions of Constraint Learning

- Types of constraints:
  - **hard** constraints define the set of valid assignments; used in SAT, LP, CP, answer set programming, ...
  - **soft** constraints define preferences among valid assignments; used in LP, all of operations research
- Learning techniques:
  - **search-based**: smartly enumerate the candidate theories and pick one that best matches the data
  - **solver-based**: encode the learning problem as a satisfaction or optimization problem and feed it to a solver
- Are the examples available from the get-go?
  - Yes: use passive / offline / batch learning
  - No: use **interactive learning**

Overview

Gallia est omnis divisa in partes tres:

- Learning **hard** constraints
  - That is, learning the requirements themselves (e.g. the *rules of sudoku* or the *nutritional requirements of diets*)

Overview

Gallia est omnis divisa in partes tres:

- Learning **hard** constraints
  - That is, learning the requirements themselves (e.g. the *rules of sudoku* or the *nutritional requirements of diets*)
- Learning **soft** constraints
  - That is, learning preferences among feasible alternatives (e.g. *cheaper diets that satisfy all requirements should be preferred*)

Overview

Gallia est omnis divisa in partes tres:

- Learning **hard** constraints
  - That is, learning the requirements themselves (e.g. the *rules of sudoku* or the *nutritional requirements of diets*)
- Learning **soft** constraints
  - That is, learning preferences among feasible alternatives (e.g. *cheaper diets that satisfy all requirements should be preferred*)
- Learning hard & soft constraints **interactively**
  - Useful when examples are not readily available or usage of supervision is expensive and should be minimized

Learning Hard Constraints

Overview

- The simplest case: Boolean formulas
  - Learning conjunctions (monomials)
  - Learning k-CNF
- Search techniques
  - General-to-specific, Specific-to-general, Version spaces
  - Syntax-guided synthesis
- Applications / implementations

The simplest case: Boolean formulas

$$\mathcal{X} = \{0,1\}^n,\ \underbrace{\boldsymbol{X} = (X_1, \ldots, X_n)}_{\textit{variables}},\ \underbrace{\boldsymbol{x} = (x_1, \ldots, x_n)}_{\textit{assignment}} \qquad \#\ \textbf{domain}$$

$$\mathcal{Y} = \{0,1\} \qquad \#\ \textbf{labels}$$

$$\mathcal{H} = \{\text{candidate formulas } \phi \text{ on } \boldsymbol{X}\} \qquad \#\ \textbf{hypotheses}$$

Examples:

- conjunctions / disjunctions of up to k literals

$$\mathcal{H} = \{L_{i_1} \vee \ldots \vee L_{i_k}\ :\ \text{all } L\text{'s are literals}\}$$

- conjunctive / disjunctive normal form (k-CNF, k-term DNF)

$$\mathcal{H} = \left\{ \bigwedge_c (L_{i_1} \vee \ldots \vee L_{i_k}) : \text{all } L\text{'s are literals} \right\}$$

for instance $(\textit{Saturday} \vee \textit{Sunday}) \wedge \textit{Sunny} \wedge \neg\textit{Bored} \wedge \neg\textit{Sick}$

The simplest case: Boolean formulas

Let $\phi^* \in \mathcal{H}$ be a hidden Boolean concept and

$$\mathcal{D} = \{(\boldsymbol{x}_k, y_k)\}_{k=1,\dots,s} \subseteq \mathcal{X} \times \mathcal{Y} \qquad \# \text{ dataset}$$

$$\text{where } y_k = \mathbb{1}\{\boldsymbol{x} \models \phi^*\}$$

$$\ell(\phi, \mathcal{D}) = |\{k \,:\, \mathbb{1}\{\boldsymbol{x}_k \models \phi\} \neq y_k\}| \qquad \# \text{ 0–1 loss}$$

Learning amounts to finding $\phi \in \mathcal{H}$ with minimal (zero) loss

$$\text{find } \phi \in \mathcal{H}$$
$$\text{s.t. } \ell(\phi, \mathcal{D}) = 0$$

This is an **search** problem

Assumptions

Assumption 1: there is a ground-truth hypothesis ϕ^* and it **belongs** to $\mathcal{H}$ (read: $\mathcal{H}$ is "expressive enough")

Assumption 2: example labels **match** ϕ^*, i.e., $y_k = \phi^*(\boldsymbol{x}_k)$ for all $k = 1, \ldots, s$ (read: there is no annotation noise)

This is the **realizable setting**: a candidate ϕ with zero loss exists and can be found by minimizing the loss [Mit81]

A bit of theory

Learning amounts to finding $\phi \in \mathcal{H}$ with minimal (zero) loss

$$\text{find } \phi \in \mathcal{H}$$
$$\text{s.t. } \ell(\phi, \mathcal{D}) = 0$$

This also **Empirical Risk Minimization** (ERM) [Vap13].

This is **good**!

- If $\mathcal{H}$ is "not too expressive" (e.g. finite or bounded VC dimension), ERM is PAC learnable
- This means that if enough examples s are given, the hypothesis found by ERM **behaves** like the true one:

$$\Pr((\boldsymbol{x} \models \phi^*) \Leftrightarrow (\boldsymbol{x} \models \phi^{\text{ERM}})) = 1$$

- ... but it doesn't say *anything* about $\phi^{\text{ERM}} = \phi^*$

Learning conjunctions (monomials)

Start from $\phi = \boldsymbol{x}_1$, then check each literal in turn. Example:

- Current hypothesis

$$\phi = \neg X_1 \wedge X_2 \wedge \neg X_3 \wedge X_4 \wedge \neg X_5$$

- Is $\neg X_1$ necessary? Generate

$$\boldsymbol{x}' = \{X_1, X_2, \neg X_3, X_4, \neg X_5\}$$

If $\boldsymbol{x}'$ is an example, check that $y = 1$:

- if positive, $\neg X_1$ is not necessary, delete it from ϕ
- if negative, $\neg X_1$ is necessary, keep it

Only $n + 1$ questions needed to recover ϕ^* (**Find-S**)

Learning conjunctions (monomials)

Consider a hidden concept $\phi^* = X_2 \land X_4$

Example	X_1	X_2	X_3	X_4	X_5	y	
$\boldsymbol{x}_1$	0	1	1	1	1	1	$\phi = \neg X_1 \land X_2 \land X_3 \land X_4 \land X_5$

Learning conjunctions (monomials)

Consider a hidden concept $\phi^* = X_2 \wedge X_4$

Example	X_1	X_2	X_3	X_4	X_5	y	
$\boldsymbol{x}_1$	0	1	1	1	1	1	$\phi = \neg X_1 \wedge X_2 \wedge X_3 \wedge X_4 \wedge X_5$
$\boldsymbol{x}_2$	1	1	1	1	1	1	$\phi = X_2 \wedge X_3 \wedge X_4 \wedge X_5$

Learning conjunctions (monomials)

Consider a hidden concept $\phi^* = X_2 \wedge X_4$

Example	X_1	X_2	X_3	X_4	X_5	y	
$\boldsymbol{x}_1$	0	1	1	1	1	1	$\phi = \neg X_1 \wedge X_2 \wedge X_3 \wedge X_4 \wedge X_5$
$\boldsymbol{x}_2$	1	1	1	1	1	1	$\phi = X_2 \wedge X_3 \wedge X_4 \wedge X_5$
$\boldsymbol{x}_3$	0	1	1	0	0	1	$\phi = X_2 \wedge X_3 \wedge X_4 \wedge X_5$

Learning conjunctions (monomials)

Consider a hidden concept $\phi^* = X_2 \wedge X_4$

Example	X_1	X_2	X_3	X_4	X_5	y	
$\boldsymbol{x}_1$	0	1	1	1	1	1	$\phi = \neg X_1 \wedge X_2 \wedge X_3 \wedge X_4 \wedge X_5$
$\boldsymbol{x}_2$	1	1	1	1	1	1	$\phi = X_2 \wedge X_3 \wedge X_4 \wedge X_5$
$\boldsymbol{x}_3$	0	1	1	0	0	1	$\phi = X_2 \wedge X_3 \wedge X_4 \wedge X_5$
$\boldsymbol{x}_4$	1	0	0	1	0	1	$\phi = X_2 \wedge X_3 \wedge X_4$

The "generalization" of all **positive** examples is:

$$\phi = X_2 \wedge X_3 \wedge X_4$$

Intuition about PAC: if $\boldsymbol{x} \sim \Pr(\boldsymbol{X})$ is "diffuse" enough and if there is no noise, eventually we will some $\boldsymbol{x}' = (\cdot, \cdot, 0, \cdot, \cdot)$ with $y' = 1$, which allows us to find $\phi' = X_2 \wedge X_4 = \phi^*$.

Learning conjunctions (monomials)

Pros:

- Discovers a hypothesis $\phi \in VC(\mathcal{D})$
- **Only needs positive examples**

Cons:

- Discovers a most specific hypothesis only – unclear why we should focus on that (read: it may be too cautious)

Learning CSPs

Constraint satisfaction problems (CSPs) are like concepts but:

- **Variables** can be non-Boolean, usually $\mathcal{X} \subseteq \mathbb{Z}^n$ (although continuous variables have been considered for LP)
- **Constraints** can be non-Boolean, e.g.

$$X_1 \geq X_2, \quad X_1 \neq X_2, \quad \mathrm{alldiff}(\{X_i : i \in \mathcal{I}\})$$

(We used $\mathrm{alldiff}$ in sudoku)

Propositionalization can encode any CSP to Bool vars only:

(X_1, X_2, X_3)	$X_1 < X_2$	$X_1 > X_2$	$X_1 = X_2$	$X_1 < X_3$	...	y
$(1, 2, 3)$	1	0	0	1	...	1
$(2, 3, 1)$	1	0	0	0	...	0

Constraint satisfaction problems (CSPs) are like concepts but:

- **Variables** can be non-Boolean, usually $\mathcal{X} \subseteq \mathbb{Z}^n$ (although continuous variables have been considered for LP)
- **Constraints** can be non-Boolean, e.g.

$$X_1 \geq X_2, \quad X_1 \neq X_2, \quad \mathrm{alldiff}(\{X_i : i \in \mathcal{I}\})$$

(We used $\mathrm{alldiff}$ in sudoku)

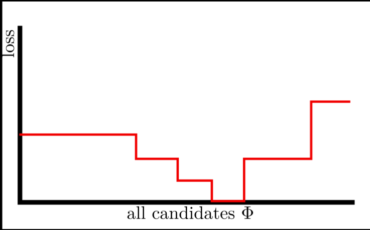
This makes **identifiability** harder: $X_1 = X_2$ and

IOW: syntactically different theories are semantically equivalent

Learning as search

Learning amounts to finding $\phi \in \mathcal{H}$ with minimal (zero) loss

$$\text{find } \phi \in \mathcal{H}$$
$$\text{s.t. } \ell(\phi, \mathcal{D}) = 0$$



This is a **large, hard problem**:

1. **large** because $\mathcal{H}$ is exponential in the number of variables,
2. **hard** because combinatorial: all variables are discrete

Generate-and-test

Generate-and-test is the simplest possible algorithm:

enumerate all $\phi \in \mathcal{H}$ and keep the ones with zero loss

Generate-and-test is the simplest possible algorithm:

enumerate all $\phi \in \mathcal{H}$ and keep the ones with zero loss

- Obviously correct :-)

Generate-and-test

Generate-and-test is the simplest possible algorithm:

enumerate all $\phi \in \mathcal{H}$ and keep the ones with zero loss

- Obviously correct :-)
- Obviously inefficient :-)

Not viable if $\mathcal{H}$ is very large, e.g. $n \geq 20$, but one can avoid to enumerate trivially invalid candidates – **used it in practice**

Also make sure not to enumerate twice

- Use **lexicographic ordering**: impose $S \succ M \succ C \succ B$

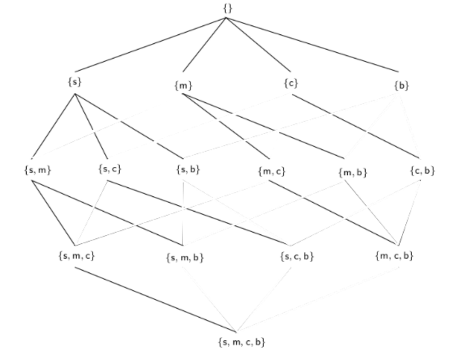


Fig. 3.5. The partial order over the item-sets

This avoids enumerating the same theory twice

These rules can become pretty tricky: [KTDR19] learns non-linear mathematical programs from **tensor** data → plenty of indices → four-level hierarchical lexicographic ordering!

Learning spreadsheet formulas with TaCLe [KPGDR17]

T1

ID	Salesperson	1st Quarter	2nd Quarter	3rd Quarter	4th Quarter	Total	Rank	Label	Items sold total	Max items sold
1	Diana Coolen	353	378	396	387	1514	2	Great	34	20
2	Marc Desmet	370	408	387	386	1551	1	Great	29	10
3	Kris Goossens	175	146	167	203	691	3	Low	19	19
4	Birgit Kenis	93	98	96	105	392	4	Low	17	15

Block 1 = T1[:, 1]; Block 2 = T1[:, 2]; Block 3 = T1[:, 3:8]; Block 4 = T1[:, 9]; Block 5 = T1[:, 10:11]

T2

Total	991	1030	1046	1081	4148
Average	247.75	257.5	261.5	270.25	1037
Max	370	408	396	387	1551
Min	93	98	96	105	392

Block 6 = T2[1:4, :]

T3

Salesperson	Items sold
Diana Coolen	5
Marc Desmet	10
Marc Desmet	8
Diana Coolen	9
Birgit Kenis	15
Marc Desmet	8
Birgit Kenis	2
Diana Coolen	20
Marc Desmet	3
Kris Goossens	19

Block 8 = T3[:, 1]; Block 9 = T3[:, 2]

T4

Quarter	Income	Expenses	Total
Q1	991	212	779
Q2	1030	710	1099
Q3	1046	137	2008
Q4	1081	240	2849

Block 10 = T4[:, 1]; Block 11 = T4[:, 2:4]

T5

Customer	Contact	Contact Name
Frank	1	Diana Coolen
Sarah	3	Kris Goossens
George	3	Kris Goossens
Mary	2	Diana Coolen
Tim	4	Birgit Kenis

Block 12 = T5[:, 1]; Block 13 = T5[:, 2]; Block 14 = T5[:, 3]

$SERIES(T_1[:,1])$

$T_1[:,1] = RANK(T_1[:,5])^*$

$T_1[:,1] = RANK(T_1[:,6])^*$

$T_1[:,1] = RANK(T_1[:,10])^*$

$T_1[:,8] = RANK(T_1[:,7])$

$T_1[:,8] = RANK(T_1[:,3])^*$

$T_1[:,8] = RANK(T_1[:,4])^*$

$T_1[:,7] = SUM_{row}(T_1[:,3{:}6])$

$T_1[:,10] = SUMIF(T_3[:,1], T_1[:,2], T_3[:,2])$

$T_2[1,:] = SUM_{col}(T_1[:,3{:}7])$

$T_2[2,:] = AVERAGE_{col}(T_1[:,3{:}7])$

$T_2[3,:] = MAX_{col}(T_1[:,3{:}7])$

$T_2[4,:] = MIN_{col}(T_1[:,3{:}7])$

$T_4[:,2] = SUM_{col}(T_1[:,3{:}6])$

$T_4[:,4] = PREV(T_4[:,4]) + T_4[:,2] - T_4[:,3]$

$T_5[:,2] = LOOKUP(T_5[:,3], T_1[:,2], T_1[:,1])^*$

$T_5[:,3] = LOOKUP(T_5[:,2], T_1[:,1], T_1[:,2])$

$T_1[:,11] = MAXIF(T_3[:,1], T_1[:,2], T_3[:,2])$

- vector = row or column
- block = type-consistent contiguous vectors
- only constraints compatible with observed blocks are enumerated – using MiniZinc!

ModelSeeker

- **General-to-specific** (or top-down): start from most general hypothesis $\phi \in \mathcal{H}$, e.g., $\phi = \top$, and gradually specialize it to **exclude negative examples**

  i.e. add constraints as we go
- **Specific-to-general** (or bottom-up): start from most specific hypothesis ϕ and gradually generalize it to **cover positive examples**.

  i.e. remove constraints as we go

Using generalization $\succeq_g$

Lattice of concepts over $\{S, M, C, B\}$ w.r.t. generalization relation

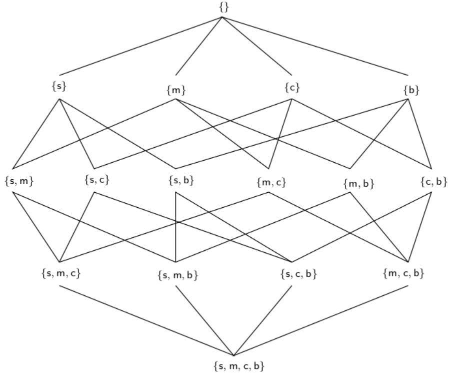


Fig. 3.5. The partial order over the item-sets

where the **generalization relation** $\phi \succeq_g \phi'$ iff ϕ covers (labels as positive) all instances covered by ϕ' and possibly some more

Using generalization $\succeq_g$

Lattice of concepts over $\{S, M, C, B\}$ w.r.t. generalization relation

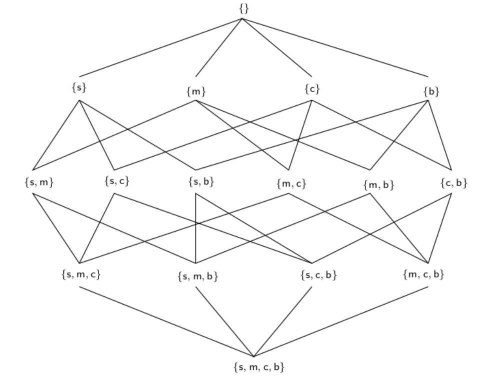


Fig. 3.5. The partial order over the item-sets

The **version space** is the set of candidates in $\mathcal{H}$ consistent with all examples: $VS(\mathcal{D}) = \{h \in \mathcal{H} \,:\, \ell(h, \mathcal{D}) = 0\}$

Using generalization $\succeq_g$

Lattice of concepts over $\{S, M, C, B\}$ w.r.t. generalization relation

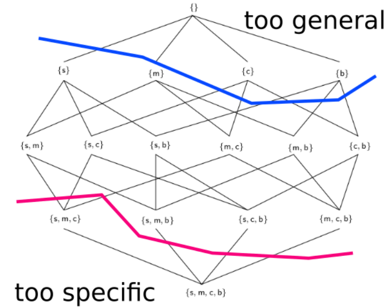


Fig. 3.5. The partial order over the item-sets

Consider examples:

$$\{1, 1, 0, 0\}, \text{negative}$$
$$\{1, 1, 1, 0\}, \text{positive}$$

The **version space** is the set of candidates in $\mathcal{H}$ consistent with all examples: $VS(\mathcal{D}) = \{h \in \mathcal{H} : \ell(h, \mathcal{D}) = 0\}$

Bi-directional search

Lattice of concepts over $\{S, M, C, B\}$ w.r.t. generalization relation

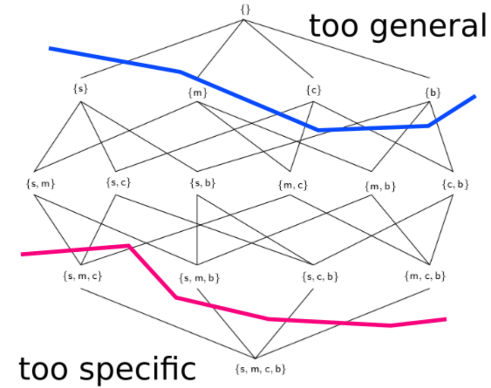


Fig. 3.5. The partial order over the item-sets

Bi-directional search iteratively shrinks the version space by observing more and more examples (more later)

Learning amounts to finding $\phi \in \mathcal{H}$ with minimal (zero) loss

$$\text{find } \phi \in \mathcal{H}$$
$$\text{s.t. } \ell(\phi, \mathcal{D}) = 0$$

Just encode this as propositional **satisfiability** (SAT)!

- SAT is NP-complete in general,
- but SAT (and related) solvers can be very efficient in practice
- also avoid encoding all examples / constraints from the get go [KPGDR17]

The advantage is that **learning is certifiably exact**!

Recall linear programs in **canonical form**:

$$\max_{\boldsymbol{x}} \ \boldsymbol{c} \cdot \boldsymbol{x} \tag{1}$$

$$\text{s.t. } \boldsymbol{a}_j \cdot \boldsymbol{x} \leq b_j \qquad j = 1, \ldots, m \tag{2}$$

and learn A and $\boldsymbol{b}$ from positive–negative examples labelled by a hidden, ground-truth polytope $A^*, \boldsymbol{b}^*$.

Notation

Name	Constant
$i = 1, \dots, n$	Index over variables
$j = 1, \dots, m$	Index over constraints
$k = 1, \dots, s$	Index over examples
$(\boldsymbol{x}^k, y^k)$	The kth example: instance $\boldsymbol{x}^k$ and label y^k
$a_{max} \in \mathbb{R}$	Maximum value for $a_{j,i}$
$b_{max} \in \mathbb{R}$	Maximum value for b_j
	Decision variable
$a_{j,i} \in \mathbb{R}$	Learned coefficients
$b_j \in \mathbb{R}$	Learned biases
	Auxiliary variable
$v_{k,j} \in \{0,1\}$	Whether example k violates constraint j
$z^a_{j,i} \in \{0,1\}$	Whether coefficient $a_{j,i}$ is non-zero
$z^b_j \in \{0,1\}$	Whether coefficient b_j is non-zero

Learning LPs with IncaLP

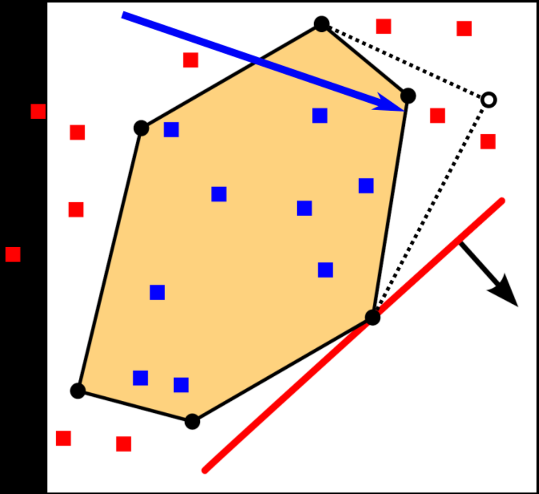
$$
\begin{aligned}
\min_{A,\boldsymbol{b}} \; & \textstyle\sum_{i,j} z^a_{j,i} + \sum_j z^b_j & & \quad (3)\\
\text{s.t. } & \boldsymbol{a}_j \cdot \boldsymbol{x}^k \leq b_j & \forall j,k \; : \; y^k = 1 & \quad (4)\\
& \textstyle\sum_j v_{k,j} \geq 1 & \forall k \; : \; y^k = 0 & \quad (5)\\
& \boldsymbol{a}_j \cdot \boldsymbol{x}^k \geq M v_{k,j} - M + b_j + \epsilon & \forall j,k \; : \; y^k = 0 & \quad (6)\\
& \textstyle\sum_i z^a_{j,i} \geq z^b_j & \forall j & \quad (7)\\
& -a_{max} z^a_{j,i} \leq a_{j,i} \leq a_{max} z^a_{j,i} & \forall i,j & \quad (8)\\
& -b_{max} z^b_j \leq b_j \leq b_{max} z^b_j & \forall j & \quad (9)
\end{aligned}
$$

IncaLP

The IncaLP algorithm: m is the number of constraints, $\mathcal{D}$ are the examples, and θ is the decision tree.

procedure $\textsc{LearnIncremental}(m, \mathcal{D}, \theta)$
  $i \leftarrow 1$
  $\mathcal{D}_i \leftarrow \mathrm{Choose}(\mathcal{D}, \theta, 20)$
  $\mathcal{V}_i \leftarrow$ all misclassified examples in $\mathcal{D} \setminus \mathcal{D}_i$
  while $\mathcal{V}_i$ is not empty **do**
    $A_i, \boldsymbol{b}_i \leftarrow \textsc{Solve}(\textsc{Encode}(m, \mathcal{D}_i))$ ▷ Eq. 3–9
    if could not find A_i, $\boldsymbol{b}_i$ consistent with $\mathcal{D}_i$ **then**
      return *infeasible*
    $\mathcal{V}_i \leftarrow$ all misclassified examples in $\mathcal{D} \setminus \mathcal{D}_i$
    $\mathcal{D}_{i+1} \leftarrow \mathcal{D}_i \cup \mathrm{Choose}(\mathcal{V}_i, \theta, 1)$
    $i \leftarrow i + 1$
  return $A_i, \boldsymbol{b}_i$

IncaLP



Choice of **which examples to add** is driven by a decision tree heuristic: not strictly necessary, but it does speed things up

IncaLP

The non-parametric IncaLP algorithm: $\mathcal{D}$ are the examples.

procedure $\textsc{LearnNoParams}(\mathcal{D})$
    $m \leftarrow 1$
    $\theta \leftarrow \mathrm{LearnDT}(\mathcal{D})$
    while *true* **do**
        $A_i, \boldsymbol{b}_i \leftarrow \textsc{LearnIncremental}(m, \mathcal{D}, \theta)$
        if could not find A_i, $\boldsymbol{b}_i$ **then**
            $m \leftarrow m + 1$
        else
            return $A_i, \boldsymbol{b}_i$

Guaranteed to **terminate** in the realizable setting

Search is search!

Learning amounts to finding $\phi \in \mathcal{H}$ with minimal (zero) loss

$$\begin{aligned} &\text{find } \phi \in \mathcal{H} \\ &\text{s.t. } \ell(\phi, \mathcal{D}) = 0 \end{aligned}$$

In principle, **any** search algorithm can be used:

- genetic algorithms (see e.g. [PK17]), tabu search, simulated annealing, ant colony optimization...
- any form of **stochastic local search** suitable for combinatorial optimization, and there are plenty [HS04]

Many of these procedures can be tuned for **anytime** solving – i.e. if you stop them at any time, they give you *some* solution

(However, they may **never** find a perfect solution)

Links to machine learning

- Constraint learning from positive–negative examples is to some extent equivalent to **binary classification**: *learn a hypothesis* $h : \mathcal{X} \to \{0, 1\}$ *with low loss*
  - Indeed, concept learning **is** classification
  - Learning constraints of linear programs is equivalent to learning a convex polytope [GKKN18]
- This means that in principle standard machine learning tools can be used here too (yeah, also neural nets)
- **However** the learned model is not used only for prediction! e.g. learned CSP can be analyzed, debugged, explained, adapted by some expert, *etc.*

Links to machine learning

- Most theory in machine learning focuses on guaranteeing **generalization**, i.e., finding conditions under which the learned classifier generalizes to instances *not* in the training set
- In the realizable case, this
- No though is given to **identifiability**: the learned model must *behave* like the gold standard, but it may be *different*!
  (Same issue in Bayesian networks *etc.*)
  The theory says nothing about this.

Learning Soft Constraints

Why soft constraints?

- Deal with conflicting requirements (e.g. multi-objective optimization)
- Combine knowledge and uncertainty (probabilistic relational models, fuzzy logic)
- Combine statistical and relational approaches to learning (statistical relational learning)

E.g: Markov Logic networks

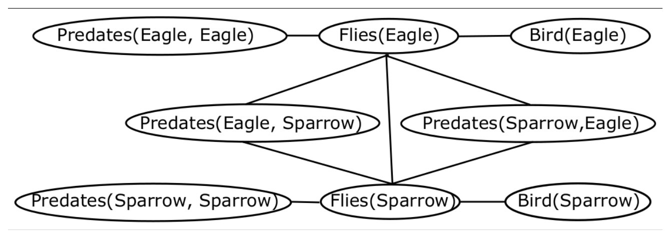
Definition

- A Markov Logic Network (MLN) L is a set of pairs (F_i, w_i) where:
  - F_i is a formula in first-order logic
  - w_i is a real number (the weight of the formula)
- Applied to a finite set of constants $C = \{c_1, \ldots, c_{|C|}\}$ it defines a Markov network $M_{L,C}$:
  - $M_{L,C}$ has one binary node for each possible grounding of each atom in L. The value of the node is 1 if the ground atom is true, 0 otherwise.
  - $M_{L,C}$ has one feature for each possible grounding of each formula F_i in L. The value of the feature is 1 if the ground formula is true, 0 otherwise. The weight of the feature is the weight w_i of the corresponding formula

Intuition

- A MLN is a *template* for Markov Networks, based on logical descriptions
- Single atoms in the template will generate nodes in the network
- Formulas in the template will be generate cliques in the network
- There is an edge between two nodes iff the corresponding ground atoms appear together in at least one grounding of a formula in L

Markov Logic networks: example



Ground network

- A MLN with two (weighted) formulas:

$$w_1 \quad \forall x \left(\texttt{Bird(x)} \Rightarrow \texttt{Flies(x)}\right)$$

$$w_2 \quad \forall x, y \left(\texttt{Predates(x,y)} \land \texttt{Bird(y)} \Rightarrow \texttt{Bird(x)}\right)$$

- applied to a set of two constants $\{\texttt{Sparrow}, \texttt{Eagle}\}$
- generates the Markov Network shown in figure

Joint probability

- A ground MLN specifies a joint probability distribution over possible worlds (i.e. truth value assignments to all ground atoms)
- The probability of a possible world x is:

$$p(x) = \frac{1}{Z} \exp \left(\sum_{i=1}^{F} w_i n_i(x) \right)$$

where:

  - the sum ranges over formulas in the MLN (i.e. clique templates in the Markov Network)
  - $n_i(x)$ is the number of true groundings of formula F_i in x
  - The partition function Z sums over all possible worlds (i.e. all possible combination of truth assignments to ground atoms)

Inference

- Compute value of x with maximal probability

$$x^* = \operatorname*{argmax}_x \frac{1}{Z} \exp\left(\sum_{i=1}^{F} w_i n_i(x)\right) = \operatorname*{argmax}_x \sum_{i=1}^{F} w_i n_i(x)$$

$\Rightarrow$ boils down to weighted MAX-SAT.

- Compute value of x with max. probability given evidence e

$$x^* = argmax_x p(x|e)$$

where evidence fixes the value of some of the variables in x

Learning

- Learn weights of (given) formulas
  - parameter learning
- Learn both formulas and weights
  - structure learning

Weighted Constraint Satisfaction Problems (wCSP)

Definition

Given

- A set of pairs $\{(c_i, w_i)\}_{i=1}^n$ where:
  - c_i is a (soft) constraint
  - $w_i \in \mathbb{R}$ is a weight
- An indicator function $\mathbb{1}\{x \models c\}$ evaluating to one if c is satisfied by x, and zero otherwise

Find

$$x^* = \operatorname*{argmax}_{x \in \mathcal{X}} f(x) = \operatorname*{argmax}_{x \in \mathcal{X}} \sum_{i=1}^n w_i \cdot \mathbb{1}\{x \models c_i\}$$

Note
Hard constraints can be incorporated in $\mathcal{X}$

Preference learning

Given

- A set of (soft) constraints $\{(c_i)\}_{i=1}^{n}$
- A set of examples pairs $\mathcal{D} = \{(x_j, x_j')\}_{j=1}^{m}$ such that for all j it should hold that

$$f(x_j) > f(x_j')$$

- An **objective** function $J(w, \mathcal{D})$

Find

- A set of weights $\hat{w}$ minimizing the objective:

$$\hat{w} = \operatorname*{argmin}_{w} J(w, \mathcal{D})$$

E.g. SVM ranking

$$
\begin{array}{cc}
\min\limits_{w} & ||w||^2 + \lambda \sum_{j=1}^{m} \xi_j \\
s.t & f(x_j) - f(x'_j) \geq 1 - \xi_j \quad \forall j \in [1, n]
\end{array}
$$

where:

- ξ_j is a penalty for not ranking x_j higher than x'_j with a large enough margin
- $||w||^2$ is a regularization term (margin is $2/||w|| \rightarrow$ large margin separation)
- $\lambda \in R^+$ is a parameter trading off margin and correct rankings

Learning weights for wCSP

Structured-output learning

Given

- A set of (soft) constraints $\{(c_i)\}_{i=1}^{n}$
- A set of input-output pairs $\mathcal{D} = \{(x_j, y_j)\}_{j=1}^{m}$ such that for all j it should hold that

$$y_j = \operatorname*{argmax}_{y \in \mathcal{Y}} f(x_j, y_j)$$

- An **objective** function $J(w, \mathcal{D})$

Find

- A set of weights $\hat{w}$ minimizing the objective:

$$\hat{w} = \operatorname*{argmin}_{w} J(w, \mathcal{D})$$

Structured-output learning for wCSP

E.g. Structured-output SVM

$$\min_{w} \quad ||w||^2 + \lambda \sum_{j=1}^{m} \xi_j$$
$$s.t \quad f(x_j, y_j) - f(x_j, y_j') \geq 1 - \xi_j \quad \forall j \in [1, n] \quad \forall y_j' \neq y_j$$

where:

- ξ_j is a penalty for not ranking y_j higher than any alternative output y_j' with a large enough margin

Problem
The number of constraints is equal to $m \times (|\mathcal{Y}| - 1)$ and is typically exponential in the number of output variables

Cutting plane algorithm

1. Initialize weights $w = 0$ and set of constraints $\mathcal{S}_j = \varnothing$ for each example j
2. While constraint added, for each example (x_j, y_j)
   2.1 Check penalty using current S_j

   $$\xi_j = \max_{y_j' \in S_j} 1 + f(x_j, y_j') - f(x_j, y_j) \quad \text{[weighted CSP problem!!]}$$

   2.2 Check penalty in full space $\mathcal{Y}$

   $$\xi_j^{new} = \max_{y_j' \neq y_j} 1 + f(x_j, y_j') - f(x_j, y_j) \quad \text{[weighted CSP problem!!]}$$

   2.3 If $\xi_j^{new} - \xi_j > \epsilon$
      2.3.1 Add constraint and update S_j
      2.3.2 Retrain

Weighted Constraint Optimization Problems (wCOP)

(possible) Definition

Given

- A set of triplets $\{(c_i, w_i, \phi_i)\}_{i=1}^{n}$ where:
  - c_i is a (soft) constraint
  - $w_i \in \mathbb{R}$ is a weight
  - ϕ_i is a cost function mapping c_i and x to a real value (e.g. a measure of how far x is from satisfying c_i)

Find

$$x^* = \underset{x \in \mathcal{X}}{\operatorname{argmin}}\, f(x) = \underset{x \in \mathcal{X}}{\operatorname{argmin}} \sum_{i=1}^{n} w_i \cdot \phi(x, c_i)$$

Note
It is more natural to model the problem as cost minimization

Learning weights for wCOP

Preference learning

Given

- A set of (soft) constraints and cost functions $\{(c_i, \phi_i)\}_{i=1}^n$
- A set of examples pairs $\mathcal{D} = \{(x_j, x_j')\}_{j=1}^m$ such that for all j it should hold that

$$f(x_j) > f(x_j')$$

- An **objective** function $J(w, \mathcal{D})$

Find

- A set of weights $\hat{w}$ minimizing the objective:

$$\hat{w} = \operatorname*{argmin}_w J(w, \mathcal{D})$$

E.g. SVM ranking

$$\begin{array}{ll} \min\limits_{w} & ||w||^2 + \lambda \sum_{j=1}^{m} \xi_j \\ s.t & f(x_j) - f(x'_j) \geq 1 - \xi_j \quad \forall j \in [1, n] \end{array}$$

where:

- ξ_j is a penalty for not ranking x_j higher than x'_j with a large enough margin
- $||w||^2$ is a regularization term (margin is $2/||w|| \rightarrow$ large margin separation)
- $\lambda \in R^+$ is a parameter trading off margin and correct rankings

Learning weights for wCOP

Structured-output learning

Given

- A set of (soft) constraints and cost functions $\{(c_i, \phi_i)\}_{i=1}^{n}$
- A set of input-output pairs $\mathcal{D} = \{(x_j, y_j)\}_{j=1}^{m}$ such that for all j it should hold that

$$y_j = \operatorname*{argmin}_{y \in \mathcal{Y}} f(x_j, y_j)$$

- An **objective** function $J(w, \mathcal{D})$

Find

- A set of weights $\hat{w}$ minimizing the objective:

$$\hat{w} = \operatorname*{argmin}_{w} J(w, \mathcal{D})$$

Structured-output learning for wCOP

E.g. Structured-output SVM

$$\min_{w} \quad ||w||^2 + \lambda \sum_{j=1}^{m} \xi_j$$
$$s.t \quad f(x_j, y_j') - f(x_j, y_j) \geq 1 - \xi_j \quad \forall j \in [1, n] \quad \forall y_j' \neq y_j$$

where:

- ξ_j is a penalty for not giving y_j a lower cost than to any alternative output y_j' with a large enough margin

Problem
The number of constraints is equal to $m \times (|\mathcal{Y}| - 1)$ and is typically exponential in the number of output variables

Structured-output SVM for wCOP

Cutting plane algorithm

1. Initialize weights $w = 0$ and set of constraints $\mathcal{S}_j = \varnothing$ for each example j
2. While constraint added, for each example (x_j, y_j)

   2.1 Check penalty using current S_j

   $$\xi_j = \max_{y_j' \in S_j} 1 + f(x_j, y_j) - f(x_j, y_j') \quad \text{[weighted COP problem!!]}$$

   2.2 Check penalty in full space $\mathcal{Y}$

   $$\xi_j^{new} = \max_{y_j' \neq y_j} 1 + f(x_j, y_j) - f(x_j, y_j') \quad \text{[weighted COP problem!!]}$$

   2.3 If $\xi_j^{new} - \xi_j > \epsilon$

      2.3.1 Add constraint and update S_j

      2.3.2 Retrain

Constraint Selection and Preference learning

Given

- A set of candidate (soft) constraints and cost functions $\mathcal{C} = \{(c_i, \phi_i)\}_{i=1}^n$
- A set of examples pairs $\mathcal{D} = \{(x_j, x_j')\}_{j=1}^m$ such that for all j it should hold that

$$f(x_j) > f(x_j')$$

- An **objective** function $J(w, \mathcal{D})$

Find

- A subset of the constraints $\hat{\mathcal{C}} \subseteq \mathcal{C}$ and a set of weights $\hat{w}$ s.t.:

$$(\hat{\mathcal{C}}, \hat{w}) = \underset{\mathcal{C}' \subseteq \mathcal{C}}{\operatorname{argmin}} \underset{w \in \mathbb{R}^{|\mathcal{C}'|}}{\operatorname{argmin}} J(w, \mathcal{D})$$

Regularization for Constraint Selection

2-norm regularization

$$J(w) = ||w||^2 + \lambda E(w)$$

- Penalizes weights by (squared) Euclidean norm
- Weights with less influence on error get smaller values
- No explicit bias towards exactly zero weights

Regularization for Constraint Selection

1-norm regularization

$$J(w) = |w| + \lambda E(w)$$

- Penalizes weights by sum of absolute values
- Encourages less relevant weights to be exactly zero (sparsity inducing norm)

Constraint Selection and Preference learning for wCOP

E.g. SVM ranking with 1-norm regularization

$$\begin{array}{ll} \min\limits_{w} & |w| + \lambda \sum_{j=1}^{m} \xi_j \\ s.t & f(x_j) - f(x'_j) \geq 1 - \xi_j \quad \forall j \in [1, n] \end{array}$$

where:

- ξ_j is a penalty for not ranking x_j higher than x'_j
- $|w|$ is a sparsity inducing regularization term
- $\lambda \in R^+$ trades off sparsity and correct rankings

Note
Structured-output learning can also be adapted by replacing two-norm with one-norm

Constraint Learning and Preference learning

Given

- A language $\mathcal{L}$ defining valid constraints
- A set of examples pairs $\mathcal{D} = \{(x_j, x'_j)\}_{j=1}^{m}$ such that for all j it should hold that
$$f(x_j) > f(x'_j)$$
- An **objective** function $J(w, \mathcal{D})$

Find

- A set of constraints valid according to the language $\mathcal{L}$ and a set of weights $\hat{w}$ s.t.:

$$(\hat{\mathcal{C}}, \hat{w}) = \underset{\mathcal{C}' \in \mathcal{L}}{\text{argmin}} \underset{w \in \mathbb{R}^{|\mathcal{C}'|}}{\text{argmin}} J(w, \mathcal{D})$$

Pyconstruct

Pyconstruct is a Python library for soft constraint learning (both wCSPs and wCOPs) using structured-output prediction. The wCOP is encoded in **MiniZinc**.

URL: `github.com/unitn-sml/pyconstruct`

Constraint Learning and Preference learning for wCSP (hints)

Two step approach

1. Run hard constraint learning algorithm to get set of candidate constraints $\mathcal{C}$
2. Run constraint selection and preference learning on $\mathcal{C}$

Combined approach

1. Start with empty set of constraints $\mathcal{C} = \varnothing$
2. While no improvement
   2.1 Use language bias from hard constraint learning to generate candidate extensions $\mathcal{C}'$ of constraints in $\mathcal{C}$
   2.2 Replace $\mathcal{C}$ with $\mathcal{C}'$
   2.3 Run constraint selection and preference learning on $\mathcal{C}$
   2.4 Discard zero weight constraints from $\mathcal{C}$

Interactive Learning

What is interactive learning good for?

1. To extract knowledge from an expert
2. To elicit the preferences of a customer by asking simple questions about alternative products
3. To ask the labels of the most informative instances only, esp. if supervision is expensive
4. To speed learning up by asking smart queries

Note: related to *active learning* and *preference elicitation*, but not quite the same

From offline to interactive

There is a hidden, **target theory** C^* over domain $\mathcal{X}$.

Offline:

- **Given** instances x_i labelled by $y_i = \mathbb{1}\{x_i \models C^*\}$
- **Find** a theory C s.t. $y_i = 1 \Leftrightarrow (x_i \models C)$ for $i = 1, \ldots, n$

From offline to interactive

There is a hidden, **target theory** C^* over domain $\mathcal{X}$.

Offline:

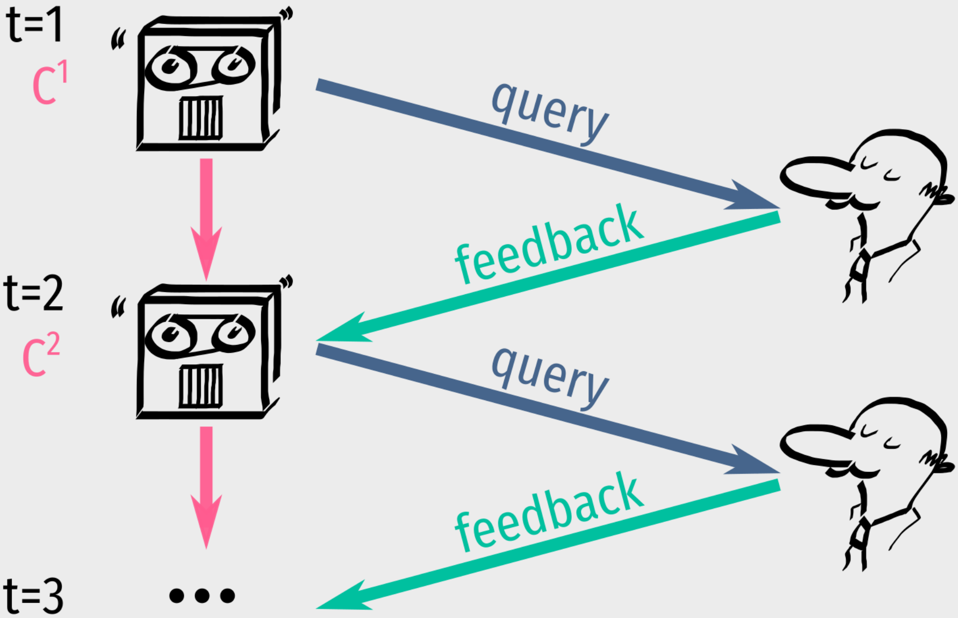
- **Given** instances x_i labelled by $y_i = \mathbb{1}\{x_i \models C^*\}$
- **Find** a theory C s.t. $y_i = 1 \Leftrightarrow (x_i \models C)$ for $i = 1, \ldots, n$

$$\Downarrow$$

Interactive:

- **Given** an **oracle** that answers queries by consulting C^*
- **Find** a theory C consistent with all answers

(The soft constraints case changes analogously.)


t=1
C^1
query
feedback
t=2
C^2
query
feedback
t=3
•••

Assumption 1: there is a ground-truth hypothesis h^* and it is **contained** in $\mathcal{H}$ (read: $\mathcal{H}$ is "expressive enough")

Assumption 2: example labels **match** h^*, i.e., $y_k = h^*(\boldsymbol{x}_k)$ for all $k = 1, \ldots, s$ (read: there is no annotation noise)

Assumption 3: the oracle is a **domain expert**

- Does *always* interpret/understand the queries
- Very dedicated, so *always* provides correct feedback

(Could also be a robot or a measurement apparatus)

An algorithm template

```
procedure LEARN (max. iterations T)
    C^1 ← initial theory
    for t = 1, ..., T do
        Choose a query q (e.g. an instance x ∈ 𝒳)
        Ask q to the oracle
        Receive feedback (e.g. whether x is a model of C*)
        C^{t+1} ← update C^t according to feedback
    return C^T
```

Questions:

- what **kind** of queries should be asked?
- how to pick an **informative** query?

What kind of queries?

For **hard** constraints

- membership: does x satisfy C^*?
- partial membership: does $x[V]$ satisfy C^*?
- equivalence: are C^t and C^* logically equivalent? If not, provide a counter-example.

For **soft** constraints[1]

- scoring: what is the score $f^*(x)$ of x?
- ranking: is $f^*(x) \geq f^*(x')$?
- improvement: give me a configuration x' s.t. $f^*(x') > f^*(x)$

[1] E.g., for wCSP the real score is $f^*(x) = \sum_i w_i^* \mathbb{1}\{x \models c_i\}$

Interactive Learning of Hard Constraints

Membership Queries: Monomials [BDH+16]

- Current hypothesis (conjunction)

$$C^t = \{\neg X_1, X_2, \neg X_3, X_4, \neg X_5\}$$

- To check whether $\neg X_1$ is really necessary, generate instance

$$x = \{X_1, X_2, \neg X_3, X_4, \neg X_5\}$$

- Ask membership query "$x \models C^*$?"
  - if positive, $\neg X_1$ is not necessary, delete it from C^t
  - if negative, $\neg X_1$ is necessary, keep it

Only $\#\text{vars} + 1$ questions needed to recover C^*

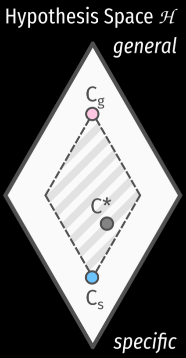
Membership Queries: ConAcq [BDH+16]

Bi-directional search

version space identified by

- most general candidate C_g
- most specific candidate C_s

C^* always within version space

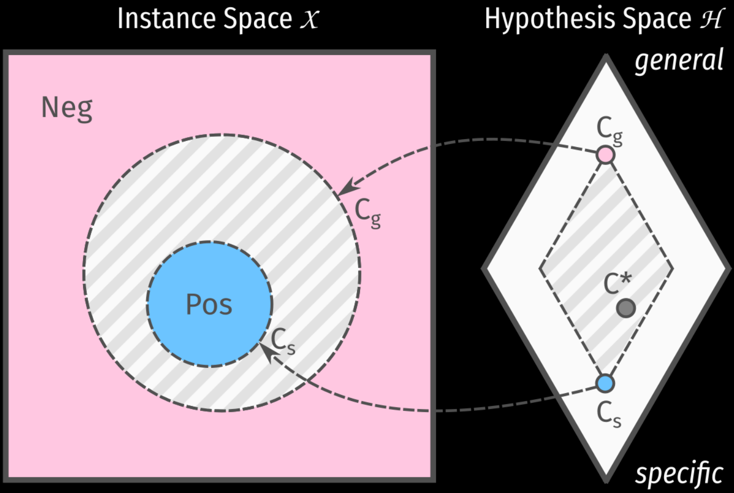


Idea: pick $x \in \mathrm{Sol}(C_g) \setminus \mathrm{Sol}(C_s)$

- If x is positive, generalize most specific candidate
- If x is negative, specialize most generic candidate

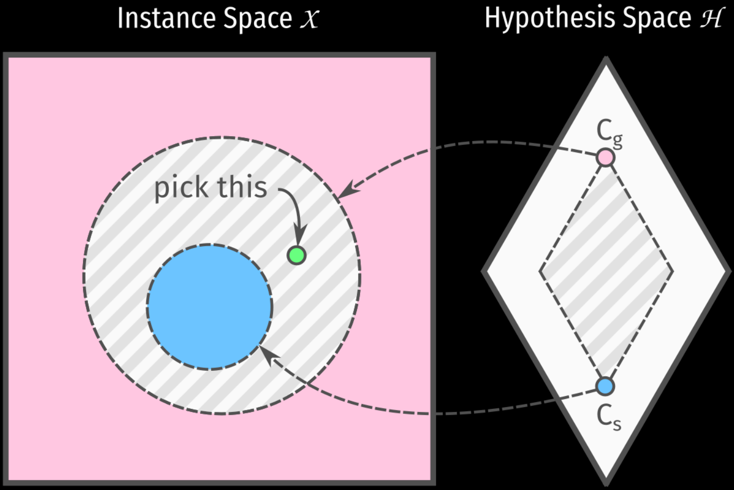
where $\mathrm{Sol}(C) = \{x : x \models C\}$

Version space and Instances



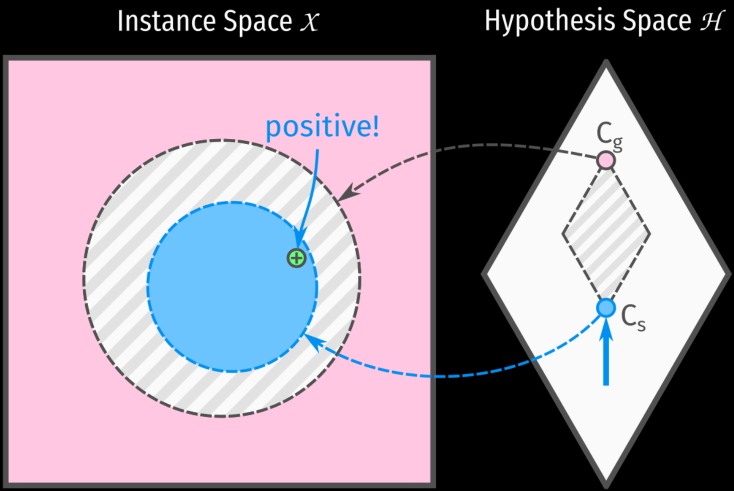
Note: $\mathrm{Sol}(C)$ is *inside* the circle

Query Selection



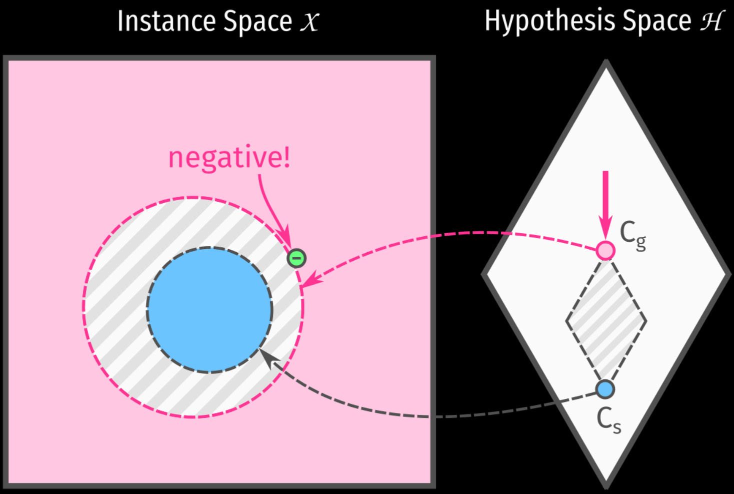
Select instance $x \in \mathrm{Sol}(C_g) \setminus \mathrm{Sol}(C_s)$

Positive $\Rightarrow$ generalize C_s



Generalizing C_s = removing constraints from it

Negative $\Rightarrow$ **specialize** C_g



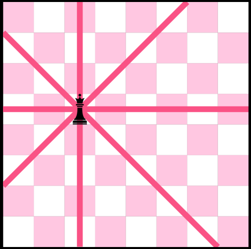
Specializing C_g = adding constraints to it

Partial Queries: Quacq [BDH+16]

Consider learning the **Eight Queens Problem**

Membership: does the board x satisfy *all* constraints?

Partial membership: does the *partial* board $x[V]$ violate at least *one* constraint?

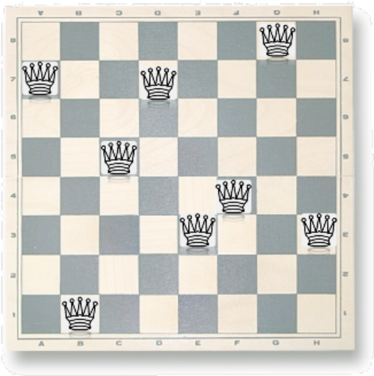


Partial membership is more informative: **all completions of the partial configuration are also negative!**

(It is also easier to answer from the oracle's perspective.)

Example (C. Bessiere)

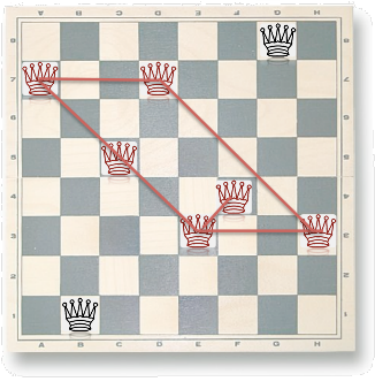
Partial Queries



ask(2, 8, 4, 2, 6, 5, 1, 6)

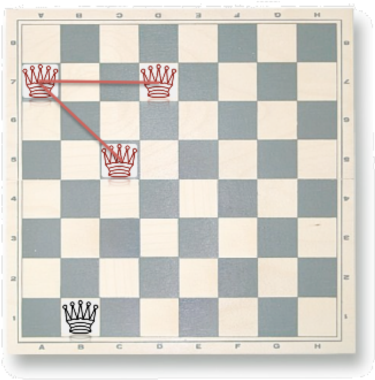
Example (C. Bessiere)

Partial Queries



ask(2, 8, 4, 2, 6, 5, 1, 6) = No

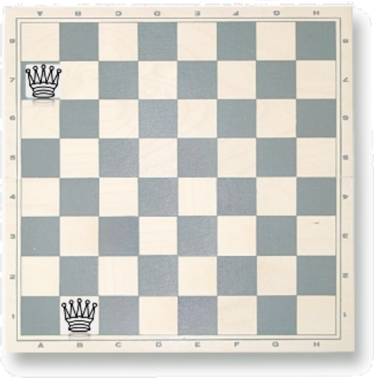
Partial Queries



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

Example (C. Bessiere)

Partial Queries



ask(2, 8, -, -, -, -, -, -) = Yes

Equivalence Queries [Ang88]

Ask whether $C^t = C^*$. If not, provide a **counter-example** x s.t.

$$x \models C^* \wedge x \not\models C^t$$

or vice-versa.

More **powerful** than membership queries[2].

But **impractical**, even domain experts may have trouble answering them.

[2] Equivalence queries can be simulated by polynomially many membership queries [BKLO17].

Soft Constraints

A user wishes to buy a custom PC. The PC is assembled from individual components: CPU, HDD, RAM, etc. Valid PC configurations must satisfy constraints, e.g. CPUs only work with compatible motherboards [TDP17]

Hard: "Intel CPUs are incompatible with AMD motherboards"

Soft: "The user prefers one CPU over another"

Weighted Constraint Satisfaction Problems (wCSP)

Definition (same as before!)

Given

- A set of pairs $\{(c_i, w_i)\}_{i=1}^n$ where:
  - c_i is a (soft) constraint
  - $w_i \in \mathbb{R}$ is a weight
- An indicator function $\mathbb{1}\{x \models c\}$ evaluating to one if c is satisfied by x and to zero otherwise

Find

$$x^* = \operatorname*{argmax}_{x \in \mathcal{X}} f(x) = \operatorname*{argmax}_{x \in \mathcal{X}} \sum_{i=1}^{n} w_i \cdot \mathbb{1}\{x \models c_i\}$$

Note hard constraints can be incorporated in $\mathcal{X}$

Assumption: hypothesis space $\mathcal{H}$ **contains** (C^*, w^*)

- For weight learning, we can reconstruct $\boldsymbol{w}^*$ perfectly

Depending on application: the oracle is **not a domain expert**

- May **not interpret/understand** the queries
- May provide **noisy** feedback

For instance, a customer on an e-commerce website.

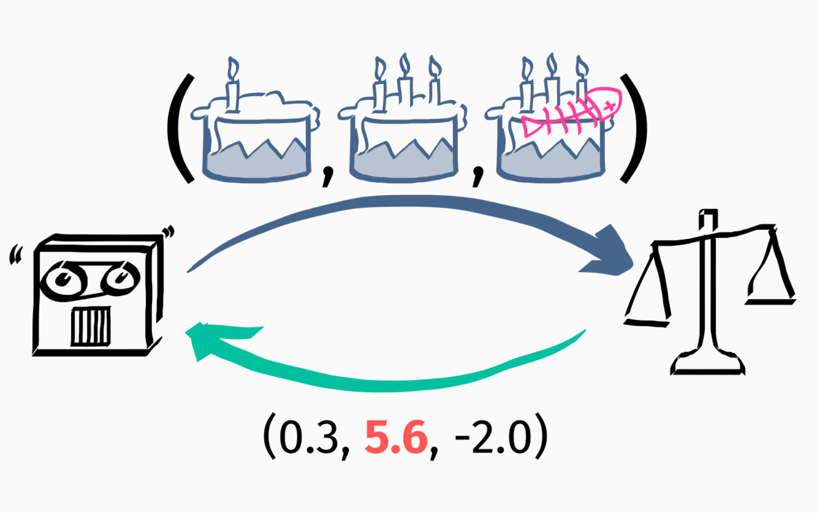
As a consequence, the **version space** may be empty!

The generic **weight learning** loop

procedure $\textsc{LearnWeights}$ (C, max iterations T)
  $\boldsymbol{w}^1 \leftarrow$ initial weights
  for $t = 1, \ldots, T$ **do**
    Choose a **query** q (e.g. an instance x)
    Ask q to the oracle
    Receive **feedback** (e.g. the actual score $f^*(x)$)
    $\boldsymbol{w}^{t+1} \leftarrow$ update $\boldsymbol{w}^t$ according to feedback
  return $\boldsymbol{w}^T$

Different instantiations for different types of queries / feedback

Scoring Queries (for a wCSP about cakes!)



A pretty ideal setup—can observe $f^*(x)$ directly!

Weight learning of wCSP via **regression** [RS04]

Same as offline case, except the dataset is built interactively

procedure LEARN (max iterations T, sample set size k)
  $\mathcal{D} \leftarrow \varnothing$
  $\boldsymbol{w} \leftarrow$ initial weights
  for $t = 1, \ldots, T$ **do**
    Sample $x_1, \ldots, x_k \in \mathrm{argmax}_{x \in \mathcal{X}} f(x; \boldsymbol{w})$
    Present $\{x_1, \ldots, x_k\}$ to the oracle
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(x_j, y_j)\}_{j=1}^{k} \qquad (y_j = f^*(x_j) + \text{noise})$
    $\boldsymbol{w}^{t+1} \leftarrow$ solve **regression** over $\mathcal{D}$
  return $\boldsymbol{w}^T$

Regression amounts to solving

$$\boldsymbol{w}^{t+1} \leftarrow \underset{\boldsymbol{w}}{\mathrm{argmin}} \sum_{(x,y) \in \mathcal{D}} (f(x; \boldsymbol{w}) - y)^2$$

A scoring oracle may not be available

$f^*($  $) = ?$

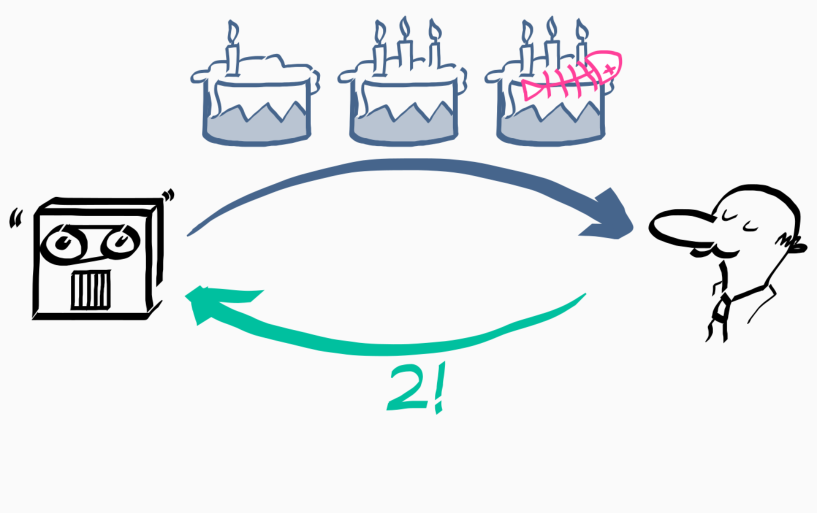
$f^*($

	7-8	17-19	19-20
Mon	running	--	swimming
Tue	--	--	gym
Wed	running	--	--
Thur	swimming	--	gym
Fri	--	soccer	soccer

$) = ?$

even experts may not be able to provide absolute scores reliably

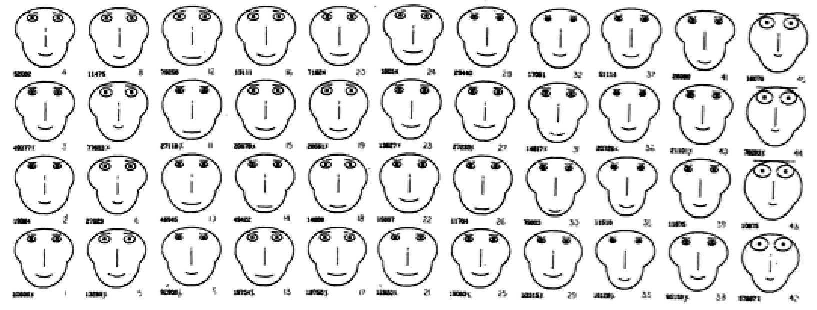
Ranking Queries



relative judgements only — no more pesky absolute scores

Ranking does not solve everything

What if it is hard to compare alternatives[3], e.g. they are too similar or too diverse? What if there are just too many?



[3] From https://eagereyes.org/criticism/chernoff-faces

Preference learning for wCSP via **ranking**[4]

Offline case: SVM ranking

$$\begin{aligned} \min_{w} \quad & ||w||^2 + \lambda \textstyle\sum_{j=1}^{m} \xi_j \\ s.t \quad & f(x_j) - f(x'_j) \geq 1 - \xi_j \quad \forall j \in [1, n] \end{aligned}$$

where:

- $f(x) = \sum_{i=1}^{n} w_i \cdot \mathbb{1}\{x \models c_i\}$
- ξ_j is a **penalty** for not ranking x_j higher than x'_j with a large enough margin
- $||w||^2$ is a **regularization** term (margin is $2/||w|| \rightarrow$ large margin separation)
- **parameter** $\lambda \in R^+$ trades off margin and penalty

[4]Slide: Andrea Passerini

Ranking for weight learning of wCSP

Simple extension of offline ranking SVM

procedure LEARN (max iterations T)
    $C^1, \boldsymbol{w}^1 \leftarrow$ initial theory, initial weights
    for $t = 1, \ldots, T$ **do**
        Choose x, x' to be high scoring and reasonably diverse
        Present (x, x') to the oracle
        Add oracle ranking $x \succcurlyeq x'$ to $\mathcal{D}$
        $\boldsymbol{w}^{t+1} \leftarrow$ learn ranking from $\mathcal{D}$
    return $\boldsymbol{w}^T$

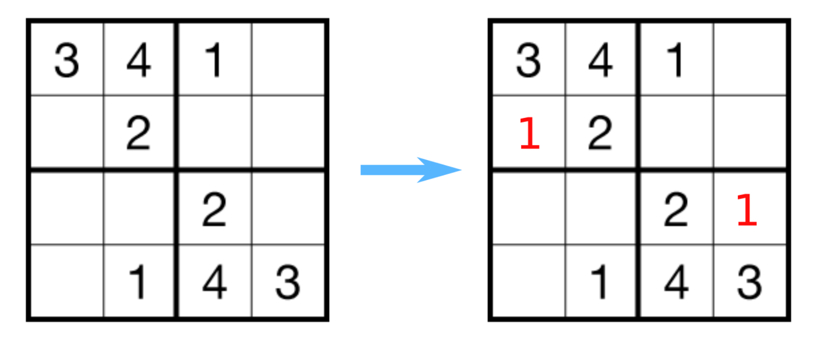
To get high score / diverse $\boldsymbol{x}$'s solve, e.g.

$$\operatorname*{argmax}_{x,x'} f(x; w) + f(x'; w) + \alpha \cdot d(x, x')$$

$$\text{s.t. } d(x, x') \leq d_{\max}$$

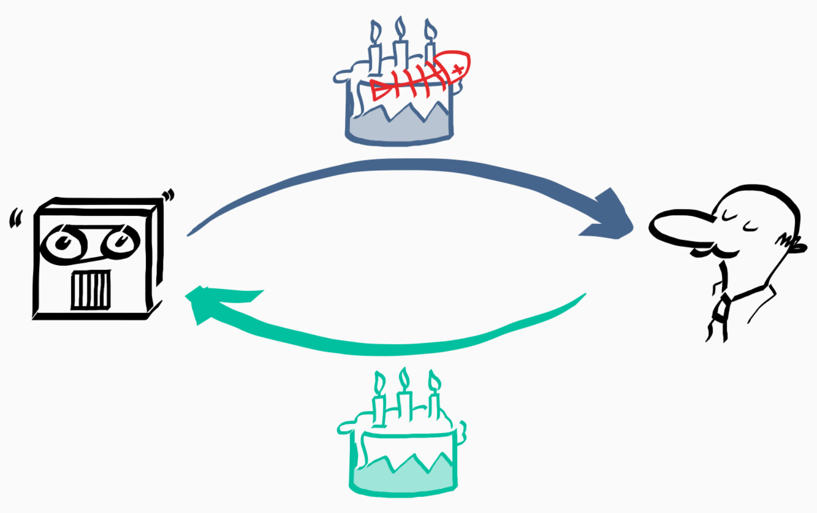
k-way inference can be slow

What if it is easy to manipulate the configurations?



It's possible to avoid "k-way inference" by asking the user to improve the current best configuration

Improvement Queries



boils down to a pairwise preference $f^*(\bar{x}^t) \geq f^*(x^t)$

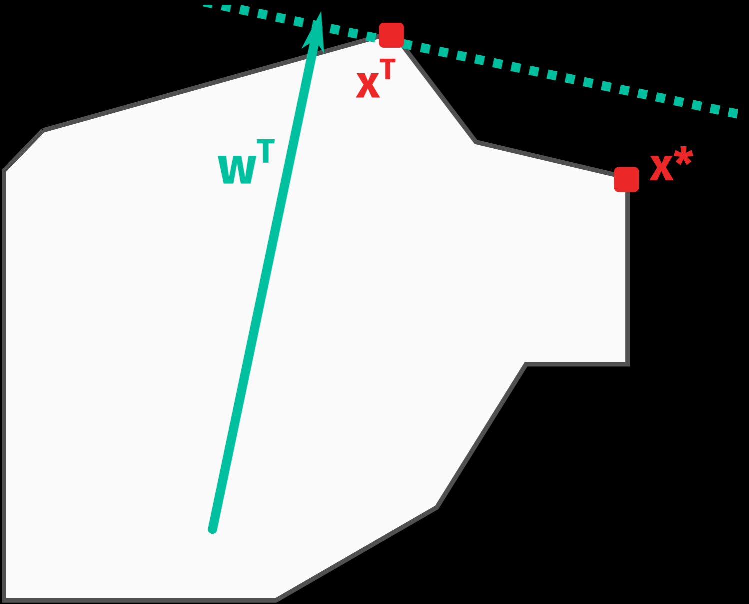
Coactive Learning for weight learning of wCOP [SJ15]

Perceptron-based preference learning from improvement queries

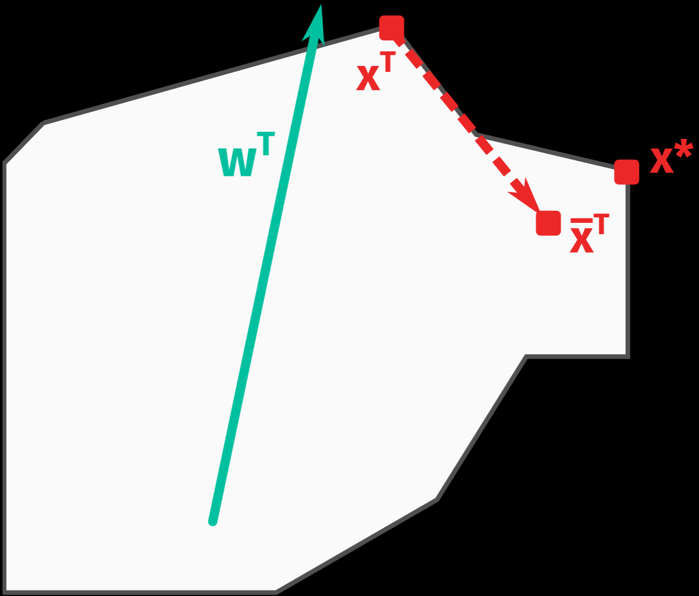
procedure LEARN (max iterations T)
  $C^1, \boldsymbol{w}^1 \leftarrow$ initial theory, initial weights
  for $t = 1, \ldots, T$ **do**
    $x^t \leftarrow \text{argmax}_{x \in \mathcal{X}} \sum_i w_i \mathbb{1}_i(x)$
    Present x^t to the oracle
    Obtain improved configuration $\bar{x}^t$
    $\boldsymbol{w}^{t+1} \leftarrow \boldsymbol{w}^t + \phi(\bar{x}^t) - \phi(x^t)$
  return C^T

Note quality of configurations approaches optimum as $O(1/\sqrt{T})$ under assumptions on the improvements

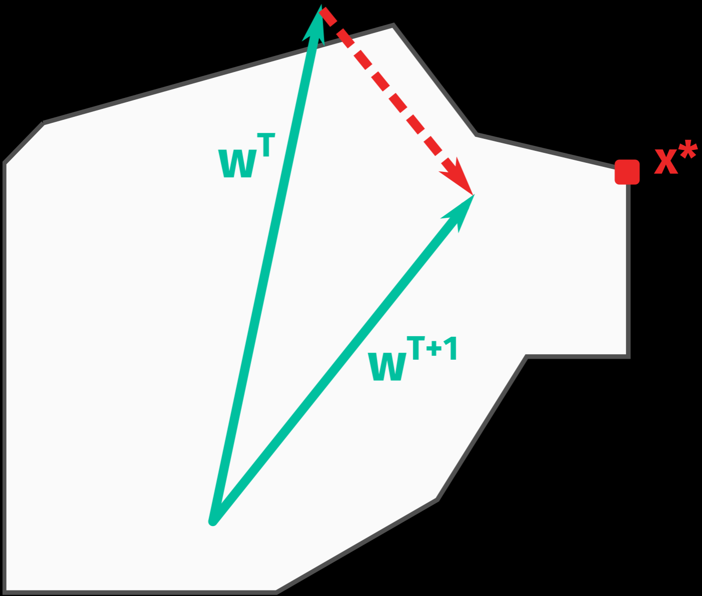
Coactive learning: iteration T



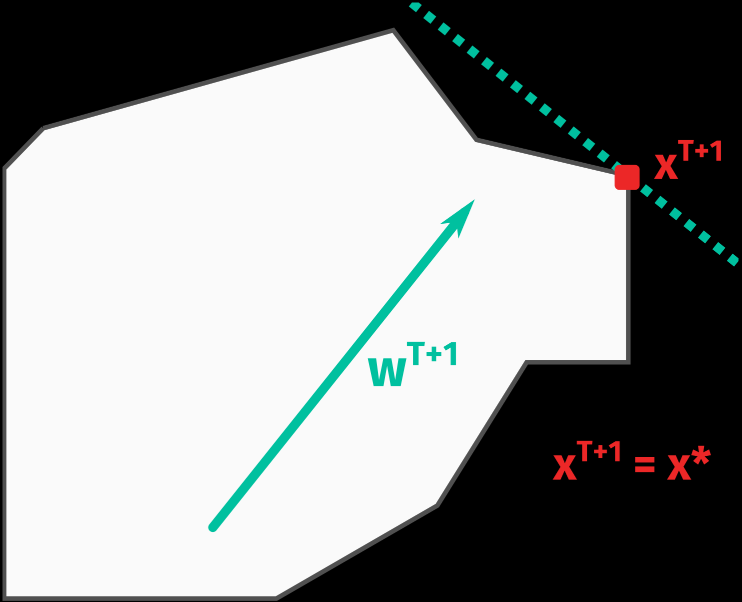
Coactive learning: iteration T



Coactive learning: iteration T



Coactive learning: iteration $T+1$



Coactive Critiquing [TDP17]

Recall that constraints $\approx$ features in wCSP

$$\mathbb{1}_i(x) = \mathbb{1}\{x \models c_i\} \qquad \forall i = 1, \ldots, n$$

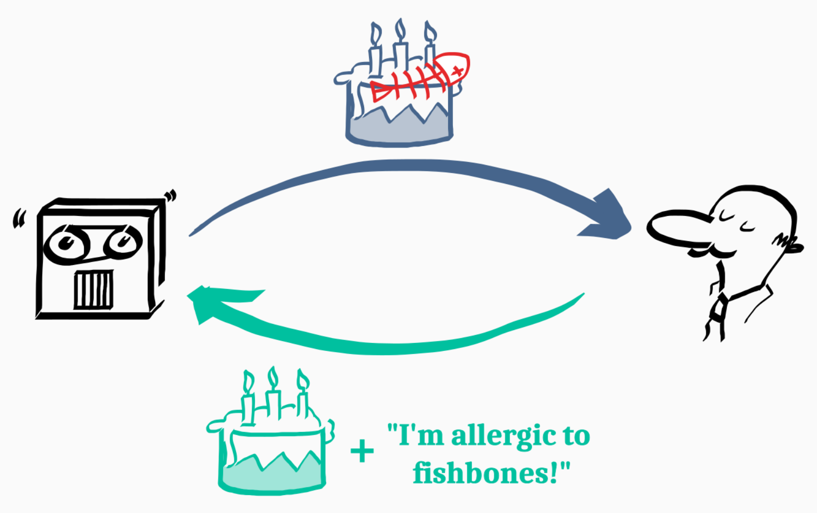
What if we don't have all of the constraints/features?

Idea

- If oracle improvement can't be explained by the learner (e.g. by linear spearability), a constraint is missing
- Ask for the missing constraint, acquire c_{n+1}

Add $\mathbb{1}_{n+1} = \mathbb{1}\{x \models c_{n+1}\}$ to the pool of features, proceed as usual with Coactive Learning

Critiquing Queries



add the critique to the constraints, update the feature space

Critiquing Queries

A few remarks

- Can be proven to converge under assumptions, even if most constraints are acquired on-the-fly
- Not really “learning”
  - Critiquing queries provide the missing constraints

Once more: **powerful oracles make learning easier**

Still much work to do!

- How to combine learning of hard **and** soft constraints?
- What are the **"best" queries** in the soft setting?
- How to properly deal with **rationally bounded** oracles?
  e.g. how to combine technology and human interaction?
- How far can we **push** and/or **guide** the oracle?
  e.g. how to best exploit and control human abilities?

(and much more)

Wrapping up

Take-away

- **Interaction** is fundamental when specifications and preferences are hard to specify upfront, can cut labeling cost and speed up learning
- **CSPs** can be learned via version space approaches (in the realizable setting)
- **wCSP/wCOP weight** learning can be cast as interactive ranking + smart query selection
- Different **query types** have different:
  - ability to learn from human non-experts
  - theoretical efficiency [Ang88, BDH+16]

Thank you!

Related frameworks

Many topics related to interactive constraint learning

- Pool-based **Active Learning**
- **Preference Elicitation** (for interactive recommendation)
- Programming by Feedback [ASSS14]
- Inverse Combinatorial Optimization [Heu04]
- ...

Pool-based Active Learning [Set12, Han14]

Given hidden decision function $f^* : \mathcal{X} \to \{\pm 1\}$, instances $x_1, \ldots, x_n \in \mathcal{X}$, and an oracle that labels instances with f^*

Find a good estimate f of f^* with *as few queries as possible*

Remarks

- Like CP, focuses on quality of learned model $\text{loss}(f^*, f)$
- ... but geometrical flavor: SVMs, Gaussian Processes
- **Many strategies in common** (e.g. version spaces)

Query types: labeling queries ($\approx$ membership), search queries ($\approx$ equivalence), rationales and explanations, ...

Preference Elicitation [Bou02, PTV16]

Given products $x_1, \dots, x_n \in \mathcal{X}$ and a user who ranks alternatives by relative preferrability

Find a good item $x \in \mathcal{X}$ with *the least cognitive effort*

If we knew the true user' scoring function $f(x)$, it would be easy!
But we don't, so we estimate it iteratively by asking queries

Remarks

- Must model **preferences**, similar to wCSP/wCOP
- wCSP/wCOP useful for recommending combinatorial items
- Unlike CP, **only quality of recommendation** x **matters**
- Learns approximation f of f^* only as byproduct

Methods: Bayesian, minimax regret, online learning

Dana Angluin.
Queries and concept learning.
Machine learning, 1988.

Riad Akrour, Marc Schoenauer, Michèle Sebag, and Jean-Christophe Souplet.
Programming by feedback.
In *International Conference on Machine Learning*, number 32, pages 1503–1511. JMLR. org, 2014.

Christian Bessiere, Abderrazak Daoudi, Emmanuel Hebrard, George Katsirelos, Nadjib Lazaar, Younes Mechqrane, Nina Narodytska, Claude-Guy Quimper, and Toby Walsh.
New approaches to constraint acquisition.
In *Data Mining and Constraint Programming*. 2016.

Christian Bessiere, Frédéric Koriche, Nadjib Lazaar, and Barry O'Sullivan.
Constraint acquisition.
Artificial Intelligence, 244:315–342, 2017.

Craig Boutilier.
A POMDP formulation of preference elicitation problems.
In *AAAI/IAAI*, 2002.

Nicolas Beldiceanu and Helmut Simonis.
Modelseeker: Extracting global constraint models from positive examples.
In *Data Mining and Constraint Programming*, pages 77–95. Springer, 2016.

Paolo Dragone, Stefano Teso, and Andrea Passerini.

Constructive preference elicitation.
Frontiers in Robotics and AI, 4:71, 2018.

Lee-Ad Gottlieb, Eran Kaufman, Aryeh Kontorovich, and Gabriel Nivasch.
Learning convex polytopes with margin.
In *Advances in Neural Information Processing Systems*, pages 5706–5716, 2018.

Steve Hanneke.
Theory of disagreement-based active learning.
Foundations and Trends® in Machine Learning, 2014.

Clemens Heuberger.
Inverse combinatorial optimization: A survey on problems, methods, and results.
Journal of combinatorial optimization, 2004.

Holger H Hoos and Thomas Stützle.

Stochastic local search: Foundations and applications.
Elsevier, 2004.

Samuel Kolb, Sergey Paramonov, Tias Guns, and Luc De Raedt.
Learning constraints in spreadsheets and tabular data.
Machine Learning, 106(9-10):1441–1468, 2017.

Mohit Kumar, Stefano Teso, and Luc De Raedt.
Acquiring integer programs from data.
2019.

Tom M Mitchell.
Generalization as search.
In *Readings in artificial intelligence*, pages 517–542. Elsevier, 1981.

Tomasz P Pawlak and Krzysztof Krawiec.

Automatic synthesis of constraints from examples using mixed integer linear programming.
European Journal of Operational Research, 261(3):1141–1157, 2017.

Gabriella Pigozzi, Alexis Tsoukias, and Paolo Viappiani.
Preferences in artificial intelligence.
Annals of Mathematics and Artificial Intelligence, 2016.

Francesca Rossi and Allesandro Sperduti.
Acquiring both constraint and solution preferences in interactive constraint systems.
Constraints, 9(4):311–332, 2004.

Burr Settles.
Active learning.
Synthesis Lectures on Artificial Intelligence and Machine Learning, 2012.

Pannaga Shivaswamy and Thorsten Joachims.
Coactive learning.
JAIR, 2015.

George J Stigler.
The cost of subsistence.
Journal of farm economics, 27(2):303–314, 1945.

Stefano Teso, Paolo Dragone, and Andrea Passerini.
Coactive critiquing: Elicitation of preferences and features.
In *AAAI*, pages 2639–2645, 2017.

Vladimir Vapnik.
The nature of statistical learning theory.
Springer science & business media, 2013.

Corné van Dooren.

A review of the use of linear programming to optimize diets, nutritiously, economically and environmentally.
Frontiers in nutrition, 5:48, 2018.