Lecture 12

Graph Matching & Linear Programming

Part 1: Matchings in Graphs

Machine Learning for Structured Data
Vlad Niculae · LTL, UvA · https://vene.ro/mlsd

Graph Matching & Linear Programming

1. Matchings in Graphs
2. Finding a Maximum-Weight Matching
3. Linear Programming
4. Bipartite graphs and the assignment problem

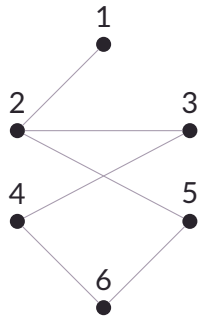
Today's agenda

- So far, we've seen two examples of structure prediction using dynamic programming: I wouldn't be too surprised if you thought DP is all you need.
- DP is widely applicable and, once you get the principle, you can come up with your own DP algorithms (e.g.: alignments with transitions, rewarding `MM` more)
- Today we see a problem that is:
  - relatively easy to state
  - widely applicable in practice
  - not tractable by DP: in fact, logsumexp is intractable!
- There are specialized algorithms for argmax for this problem. But you need to know them; sometimes they are hard to implement, etc.
- So, I will teach you a magic trick: formulate and solve (almost) any structured argmax problem using linear programming.

Matchings in Graphs

Given an undirected graph $G = (V, E)$, a **matching** (sometimes: coupling) is a subset of edges $M \subseteq E$ such that no two edges in M touch.

Examples:

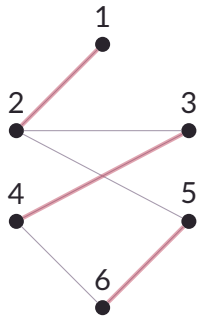


Matchings in Graphs

Given an undirected graph $G = (V, E)$, a **matching** (sometimes: coupling) is a subset of edges $M \subseteq E$ such that no two edges in M touch.

Examples:

- $M = \{(1, 2), (3, 4), (5, 6)\}$

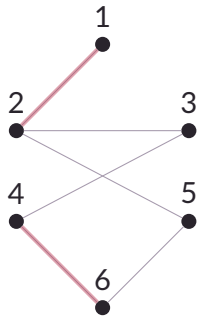


Matchings in Graphs

Given an undirected graph $G = (V, E)$, a **matching** (sometimes: coupling) is a subset of edges $M \subseteq E$ such that no two edges in M touch.

Examples:

- $M = \{(1, 2), (3, 4), (5, 6)\}$
- $M = \{(1, 2), (4, 6)\}$

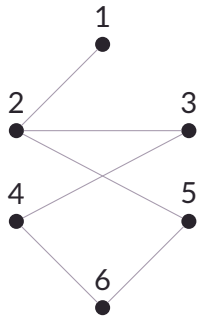


Matchings in Graphs

Given an undirected graph $G = (V, E)$, a **matching** (sometimes: coupling) is a subset of edges $M \subseteq E$ such that no two edges in M touch.

Examples:

- $M = \{(1, 2), (3, 4), (5, 6)\}$
- $M = \{(1, 2), (4, 6)\}$
- $M = \{\}$

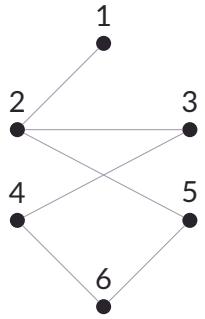


Weighted Matchings in Graphs

Given an undirected graph $G = (V, E)$, a **matching** (sometimes: coupling) is a subset of edges $M \subseteq E$ such that no two edges in M touch.

Examples:

- $M = \{(1, 2), (3, 4), (5, 6)\}$
- $M = \{(1, 2), (4, 6)\}$
- $M = \{\}$



If the graph is **weighted** $G = (V, E, w)$, where $w(e)$ is the weight of edge $e \in E$, we define the weight of a matching:

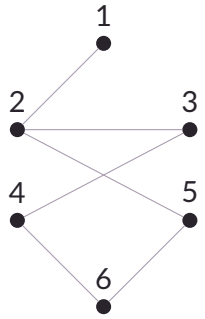
$$w(M) = \sum_{e \in M} w(e)$$

Weighted Matchings in Graphs

Given an undirected graph $G = (V, E)$, a **matching** (sometimes: coupling) is a subset of edges $M \subseteq E$ such that no two edges in M touch.

Examples:

- $M = \{(1, 2), (3, 4), (5, 6)\}$
- $M = \{(1, 2), (4, 6)\}$
- $M = \{\}$



If the graph is **weighted** $G = (V, E, w)$, where $w(e)$ is the weight of edge $e \in E$, we define the weight of a matching:

$$w(M) = \sum_{e \in M} w(e)$$

A **maximum weight matching** is a matching M maximizing $w(M)$.

Applications in Biology: Protein Structure

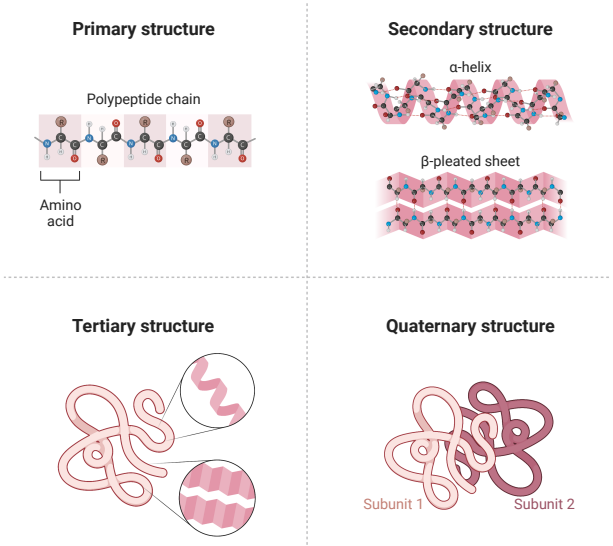


Image courtesy of dr. Laura Carroll, created with BioRender.com

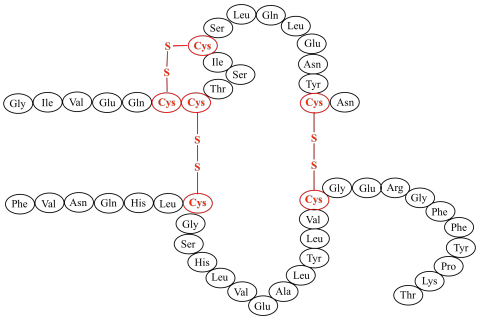
Applications in Biology: Protein Structure

One of the many forms of *protein structure*: covalent bonds between parts of the chain.

Ex: disulfide bridges between Cysteine residues.

These grant stability, determine protein folding patterns, etc.

But which pairs will form bonds and which won't?



Disulfide bridges in human insulin. Source: `https://www.chem.ucla.edu/~harding/IGOC/D/disulfide_bridge.html`, public domain image.

Applications in Biology: Protein Structure

One of the many forms of *protein structure*: covalent bonds between parts of the chain.

Ex: disulfide bridges between Cysteine residues.

These grant stability, determine protein folding patterns, etc.

But which pairs will form bonds and which won't?

jmb
Journal of Molecular Biology
Volume 434, Issue 2, 30 January 2022, 167357

Research Article

Disulfide Bonds Play a Critical Role in the Structure and Function of the Receptor-binding Domain of the SARS-CoV-2 Spike Antigen

Andrey M. Grishin [1], Nataliya V. Dolgova [1, 2], Shelby Landreth [4], Olivier Fisette [5], Ingrid J. Pickering [2, 3], Graham N. George [2, 3], Darryl Falzarano [4], Miroslaw Cygler [1]

Show more

+ Add to Mendeley Share Cite

https://doi.org/10.1016/j.jmb.2021.167357

Get rights and content

Highlights

- The Receptor-Binding Domain of the Spike protein is dependent on its disulfide bonds.
- Reduction of the disulfide bonds in the RBD render it unstable.
- Reduction of the disulfide bonds of the RBD decreases its affinity to ACE2.
- The highly-reducing environment can stop SARS-CoV-2 replication in cell-based assays.

Lecture 12

Graph Matching & Linear Programming

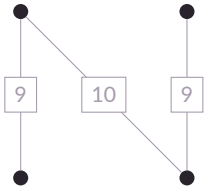
Part 2: Finding a Maximum-Weight Matching

Machine Learning for Structured Data
Vlad Niculae · LTL, UvA · https://vene.ro/mlsd

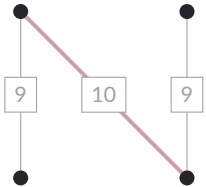
Graph Matching & Linear Programming

1. Matchings in Graphs
2. **Finding a Maximum-Weight Matching**
3. Linear Programming
4. Bipartite graphs and the assignment problem

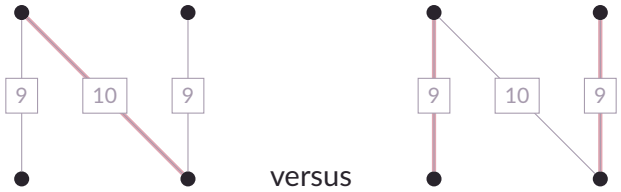
Greedy Approaches Fail



Greedy Approaches Fail



Greedy Approaches Fail



Finding a maximum weight matching

Spoiler: a good algorithm exists. But what if:

- you can't find it / don't know the right keyword to search for
- no good implementation is available?
- you want to modify the problem slightly?
- you just want to prototype something to make sure the rest of your model makes sense?

Lecture 12

Graph Matching & Linear Programming

Part 3: Linear Programming

Machine Learning for Structured Data
Vlad Niculae · LTL, UvA · https://vene.ro/mlsd

Graph Matching & Linear Programming

1. Matchings in Graphs
2. Finding a Maximum-Weight Matching
3. **Linear Programming**
4. Bipartite graphs and the assignment problem

A crash course in linear optimization ("programming")

Here once again "programming" means "optimization".

Optimization problem (in general):

A problem of the form:

$$\begin{aligned} \text{minimize} \quad & F(\boldsymbol{y}) \\ \text{subject to} \quad & G_i(\boldsymbol{y}) \le 0, \quad i = 1, \dots, p \\ & \boldsymbol{y} \in \mathbb{R}^d \end{aligned}$$

Notation/terminology:

- y_j are called the variables,
- F is called the "objective",
- G_i are constraints.

Here, $\boldsymbol{y}$ is just a variable.

Example: machine learning model training:

$$\text{minimize} \quad L(\boldsymbol{\theta})$$

where L is a total loss over a training set, and $\boldsymbol{\theta}$ are the model weights, is an optimization problem (but not a linear one usually).

A crash course in linear optimization

An optimization problem is **linear** if the objective F and all constraints G_i are linear:

Linear program (LP):

A problem of the form:

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{a} \cdot \boldsymbol{y} \\ \text{subject to} \quad & \boldsymbol{g}_i \cdot \boldsymbol{y} + b_j \leq 0, \quad i = 1, \ldots, p \\ & \boldsymbol{y} \in \mathbb{R}^d \end{aligned}$$

General-purpose algorithms can solve this problem in polynomial time in expectation.

A crash course in linear optimization

An optimization problem is **linear** if the objective F and all constraints G_i are linear:

Linear program (LP):

A problem of the form:

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{a} \cdot \boldsymbol{y} \\ \text{subject to} \quad & \boldsymbol{g}_i \cdot \boldsymbol{y} + b_j \leq 0, \quad i = 1, \ldots, p \\ & \boldsymbol{y} \in \mathbb{R}^d \end{aligned}$$

General-purpose algorithms can solve this problem in polynomial time in expectation.

Integer linear program (ILP):

A problem of the form:

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{a} \cdot \boldsymbol{y} \\ \text{subject to} \quad & \boldsymbol{g}_i \cdot \boldsymbol{y} + b_j \leq 0, \quad i = 1, \ldots, p \\ & \boldsymbol{y} \in \mathbb{Z}^d \end{aligned}$$

This is discrete optimization!

Useful heuristics exist, but very hard in general. (NP-complete!)

ILP for structure prediction

- ILPs with boolean variables $y_i \in \{0, 1\}$ can express many structured problems we care about.
- Think of this like a "domain-specific language":
- Just like pytorch can be used to express and train ML models, the language of LP can be used to express and solve argmax problems in structure prediction.
- Whenever possible, specialized algorithms are much better.
- But LP is great for prototyping, testing, exploring small changes, etc...

ILP for max-weight matchings

Finding a max-weight matching in a weighted graph $G = (V, E, w)$:

- A variable $y_e \in \{0, 1\}$ for each $e \in E$.
- In a matching, for each node, at most one incident edge can be selected. For any $u \in V$ define $E(u) = \{e \in E : e = (u, \cdot) \text{ or } e = (\cdot, u)\}$. Then we need:

$$\left(\sum_{e \in E(u)} y_e \right) \leq 1, \text{ for every node } u \in V$$

- Maximize the sum of weights of the selected edges:

$$\sum_{e \in E} w(e) y_e$$

Putting it all together,

Max-weight matching ILP

$$\text{maximize} \quad \sum_{e \in E} w(e) y_e$$

$$\text{subject to} \quad \left(\sum_{e \in E(u)} y_e \right) \leq 1, \text{ for all } u \in V,$$

$$y_e \in \{0, 1\} \text{ for all } e \in E.$$

Plug this into a general-purpose solver and you have exact solutions for small problems. (In general, solving ILP is exponential in the problem size.)

Relaxing the integer constraints

ILP is hard, but LP (without I) is much easier.

(Continuous optimization generally easier than discrete optimization.)

This leads to the fractional relaxation that can be solved in polynomial time:

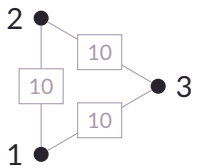
Max-weight matching relaxed LP

$$\begin{aligned} \text{maximize} \quad & \sum_{e \in E} w(e) y_e \\ \text{subject to} \quad & \left(\sum_{e \in E(u)} y_e \right) \leq 1, \text{ for all } u \in V, \\ & 0 \leq y_e \leq 1 \text{ for all } e \in E. \end{aligned}$$

In fact, since every edge participates in at least one sum constraint, the constraints $x_e \leq 1$ are redundant and can be removed. But leaving them in is not a problem.

Relaxations generally introduce approximation

The ILP is exact, the relaxed LP is not:



The ILP constraints give:

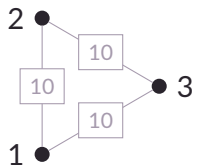
$$\begin{cases} y_{12} + y_{13} \leq 1 \\ y_{12} + y_{23} \leq 1 \\ y_{13} + y_{23} \leq 1 \\ y_{ij} \in \{0, 1\} \end{cases}$$

Write $\boldsymbol{y} = (y_{12}, y_{23}, y_{13})$. Trying all 2^3 combinations shows that the only allowed configurations are:

- $\boldsymbol{y} = (0, 0, 0)$ obj= 0
- $\boldsymbol{y} = (1, 0, 0)$ obj= 10
- $\boldsymbol{y} = (0, 1, 0)$ obj= 10
- $\boldsymbol{y} = (0, 0, 1)$ obj= 10

Relaxations generally introduce approximation

The ILP is exact, the relaxed LP is not:



The ILP constraints give:

$$\begin{cases} y_{12} + y_{13} \leq 1 \\ y_{12} + y_{23} \leq 1 \\ y_{13} + y_{23} \leq 1 \\ y_{ij} \in \{0, 1\} \end{cases}$$

Write $\boldsymbol{y} = (y_{12}, y_{23}, y_{13})$. Trying all 2^3 combinations shows that the only allowed configurations are:

- $\boldsymbol{y} = (0, 0, 0)$ obj= 0
- $\boldsymbol{y} = (1, 0, 0)$ obj= 10
- $\boldsymbol{y} = (0, 1, 0)$ obj= 10
- $\boldsymbol{y} = (0, 0, 1)$ obj= 10

Relaxing the integer constraints still allows these configurations but also allows the fractional one

$$\boldsymbol{y} = \left(\frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right)$$

But this "configuration" has higher objective:

$$\frac{10}{2} + \frac{10}{2} + \frac{10}{2} = 15 > 10$$

In general, fractional relaxations can have spurious solutions.

Lecture 12

Graph Matching & Linear Programming

Part 4: Bipartite graphs and the assignment problem

Machine Learning for Structured Data
Vlad Niculae · LTL, UvA · https://vene.ro/mlsd

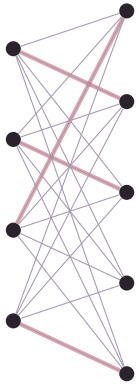
Graph Matching & Linear Programming

1. Matchings in Graphs
2. Finding a Maximum-Weight Matching
3. Linear Programming
4. **Bipartite graphs and the assignment problem**

Bipartite weighted matching: the assignment problem

An undirected graph $G = (V, E)$ is called bipartite if

- $V = V_A \cup V_B$ with $V_A \cap V_B = \varnothing$
- every edge is from some node in V_A to some node in V_B, never within.



Write $n = |V_A|$ and $m = |V_B|$ and assume $n \leq m$ (otherwise, swap them.)

The **assignment problem**, e.g., optimally assigning tasks V_A to workers V_B: max-weight matchings in a bipartite graphs, with the additional constraint that every node in V_A gets assigned.

The weights and variables can be organized into $n \times m$ matrices $\boldsymbol{A}$ and $\boldsymbol{Y}$. The node-degree constraints are row and column sums:

$$\begin{cases} \sum_{j=1}^{m} y_{ij} = 1 & \text{for } i = 1, \ldots, n \\ \sum_{i=1}^{n} y_{ij} \leq 1 & \text{for } j = 1, \ldots, m \end{cases}$$

In this bipartite case, it can be shown that the LP relaxation is exact!

Bipartite weighted matching LP

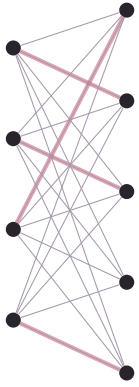
Bipartite weighted matching LP

$$\text{maximize} \quad \sum_{i=1}^{n} \sum_{j=1}^{m} a_{ij} y_{ij}$$

$$\text{subject to} \quad \sum_{j=1}^{m} y_{ij} = 1 \quad \text{for } i = 1, \ldots, n$$

$$\sum_{i=1}^{n} y_{ij} \leq 1 \quad \text{for } j = 1, \ldots, m$$

$$y_{ij} \geq 0 \quad \text{for all } i, j.$$



Learning To Match

Matchings can be structured outputs in ML tasks: we train a model that predicts the edge weights. For example:

- Protein structure: which pairs of Cysteine residues are more compatible?
- Learning to match students with thesis projects (recommender system)
- Matching words in foreign language sentence translations (reordering)

Example architecture: parametrize edge weights $a_{ij} = \boldsymbol{u}_i \cdot \boldsymbol{v}_j$ where $\boldsymbol{u}_i$ is an embedding of a student i and $\boldsymbol{v}_j$ embedding of the project j (each possibly passed through multiple hidden layers).

Learning To Match

Matchings can be structured outputs in ML tasks: we train a model that predicts the edge weights. For example:

- Protein structure: which pairs of Cysteine residues are more compatible?
- Learning to match students with thesis projects (recommender system)
- Matching words in foreign language sentence translations (reordering)

Example architecture: parametrize edge weights $a_{ij} = \boldsymbol{u}_i \cdot \boldsymbol{v}_j$ where $\boldsymbol{u}_i$ is an embedding of a student i and $\boldsymbol{v}_j$ embedding of the project j (each possibly passed through multiple hidden layers).

For predicting assignment, can solve the LP.

However, for learning, logsumexp over all matchings is intractable!

Non-Probabilistic Structure Learning

What can we do when we cannot compute the probabilistic loss:

$$-\log P(\boldsymbol{y}) = L_{\text{NLL}}(\boldsymbol{y}) = -\operatorname{score}(\boldsymbol{y}) + \log \sum_{\boldsymbol{y}' \in \mathcal{Y}} \exp \operatorname{score}(\boldsymbol{y}')$$

Non-Probabilistic Structure Learning

What can we do when we cannot compute the probabilistic loss:

$$-\log P(\boldsymbol{y}) = L_{\text{NLL}}(\boldsymbol{y}) = -\operatorname{score}(\boldsymbol{y}) + \log \sum_{\boldsymbol{y}' \in \mathcal{Y}} \exp \operatorname{score}(\boldsymbol{y}')$$

Recall: An alternative training objective is the perceptron loss:

$$L_{\text{Perc}}(\boldsymbol{y}) = -\operatorname{score}(\boldsymbol{y}) + \max_{\boldsymbol{y}' \in \mathcal{Y}} \operatorname{score}(\boldsymbol{y}')$$

The Perceptron loss can sometimes be computed using (I)LP!

Structured Perceptron Learning

$$L_{\text{Perc}}(\boldsymbol{y}) = -\,\text{score}(\boldsymbol{y}) + \max_{\boldsymbol{y}' \in \mathcal{Y}} \text{score}(\boldsymbol{y}')$$

When we can

- represent the structures $\boldsymbol{y}$ as vectors and $\mathcal{Y}$ as linear constraints,
- write $\text{score}(\boldsymbol{y}) = \boldsymbol{a} \cdot \boldsymbol{y}$,

(e.g., our graph matching applications). Then:

- solve the ILP $\boldsymbol{y}^* = \arg\max_{\boldsymbol{y} \in \mathcal{Y}} \text{score}(\boldsymbol{y})$ or the LP $\boldsymbol{y}^* = \arg\max_{\boldsymbol{y} \in \tilde{\mathcal{Y}}} \text{score}(\boldsymbol{y})$
- $L_{\text{Perc}}(\boldsymbol{y}) = -\boldsymbol{a} \cdot \boldsymbol{y} + \boldsymbol{a} \cdot \boldsymbol{y}^*$

In the relaxed perceptron, fractional solutions are allowed: there is theory that shows this is ok and can even work better.

In both cases, as long as you have any code (non-pytorch) to find $\boldsymbol{y}^*$, plug it in and pytorch gives the correct gradient $\nabla_{\boldsymbol{a}} L_{\text{Perc}}(\boldsymbol{y}) = \boldsymbol{y}^\star - \boldsymbol{y}$.

Practical Example: Knapsack problems

Say we have a knapsack that can hold B liters, and we are packing for a trip.

We have n objects, each with a volume v_i liters, and with a score (value) a_i. We want to pack the most valuable items.

Weighted Knapsack (I)LP

$$\text{maximize} \quad \sum_{i=1}^{n} a_i y_i$$

$$\text{subject to} \quad \sum_{i=1}^{n} v_i y_i \leq B$$

(if ILP:) $y_i \in \{0, 1\}$ for all i

(if LP:) $y_i \in [0, 1]$ for all i

```python
import numpy as np
import cvxpy as cp

v = np.array([ .5,  .1, .9, .2, 3.0])
a = np.array([-.2, -.5, .3, .2, 1.0])
B = 4.0

n = v.shape[0]
y = cp.Variable(n, integer=False)  # integer=True for ILP

objective = a @ y
constraints = [v @ y <= B,
               y >= 0,  # applied elementwise
               y <= 1]  # applied elementwise

problem = cp.Problem(cp.Maximize(objective), constraints)
problem.solve()

print(y.value.round(2))
# if LP:  [0.   0.   0.95   1.   0.98]
# if ILP: [0.   0.   1.0    0.   1.0]
```

Ease of Prototyping Extensions

Say you are now asked to solve the Knapsack problem with an additional requirement: the first two items are mutually exclusive.

Can right away add a constraint $y_1 + y_2 \leq 1$.

A dedicated algorithm can be much faster, exact, but not as easy to modify.

And if you come up with a better specialized algorithm, you should still test it against the ILP.