Probabilistic Graphical Models

David Sontag

New York University

Lecture 7, March 8, 2012

Today's lecture

1. MAP inference as an integer linear program
2. Linear programming relaxations for MAP inference
3. Efficiently solving the dual

MAP as an integer linear program (ILP)

- MAP as a discrete optimization problem is

$$\arg\max_{\mathbf{x}} \sum_{i \in V} \theta_i(x_i) + \sum_{ij \in E} \theta_{ij}(x_i, x_j).$$

- To turn this into an integer linear program, we introduce variables

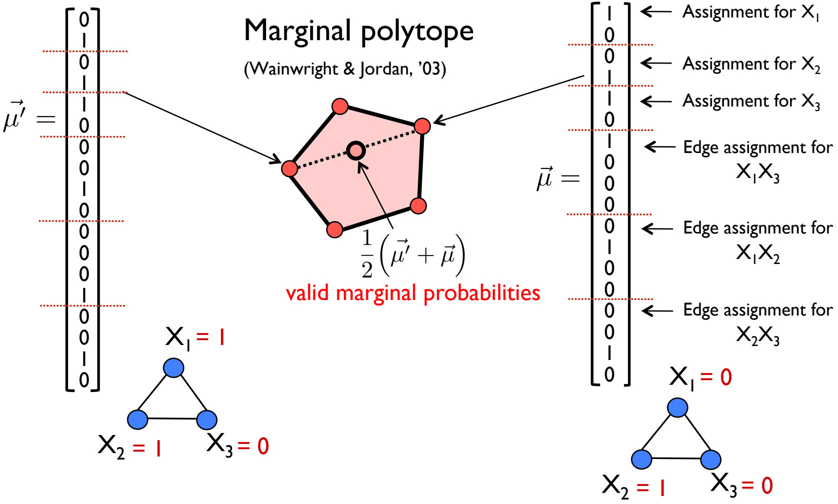
  1. $\mu_i(x_i)$, one for each $i \in V$ and state x_i
  2. $\mu_{ij}(x_i, x_j)$, one for each edge $ij \in E$ and pair of states x_i, x_j

- The objective function is then

$$\max_{\mu} \sum_{i \in V} \sum_{x_i} \theta_i(x_i)\mu_i(x_i) + \sum_{ij \in E} \sum_{x_i, x_j} \theta_{ij}(x_i, x_j)\mu_{ij}(x_i, x_j)$$

- What is the dimension of μ, if binary variables?

Visualization of feasible μ vectors



What are the constraints?

- Force every "cluster" of variables to choose a local assignment:

$$
\begin{aligned}
\mu_i(x_i) &\in \{0,1\} \quad \forall i \in V, x_i \\
\sum_{x_i} \mu_i(x_i) &= 1 \quad \forall i \in V \\
\mu_{ij}(x_i, x_j) &\in \{0,1\} \quad \forall ij \in E, x_i, x_j \\
\sum_{x_i, x_j} \mu_{ij}(x_i, x_j) &= 1 \quad \forall ij \in E
\end{aligned}
$$

- Enforce that these local assignments are globally consistent:

$$
\begin{aligned}
\mu_i(x_i) &= \sum_{x_j} \mu_{ij}(x_i, x_j) \quad \forall ij \in E, x_i \\
\mu_j(x_j) &= \sum_{x_i} \mu_{ij}(x_i, x_j) \quad \forall ij \in E, x_j
\end{aligned}
$$

MAP as an integer linear program (ILP)

$$\mathrm{MAP}(\theta) = \max_{\mu} \sum_{i \in V} \sum_{x_i} \theta_i(x_i)\mu_i(x_i) + \sum_{ij \in E} \sum_{x_i, x_j} \theta_{ij}(x_i, x_j)\mu_{ij}(x_i, x_j)$$

subject to:

$$\begin{aligned}
\mu_i(x_i) &\in \{0,1\} \quad \forall i \in V, x_i \\
\sum_{x_i} \mu_i(x_i) &= 1 \quad \forall i \in V \\
\mu_i(x_i) &= \sum_{x_j} \mu_{ij}(x_i, x_j) \quad \forall ij \in E, x_i \\
\mu_j(x_j) &= \sum_{x_i} \mu_{ij}(x_i, x_j) \quad \forall ij \in E, x_j
\end{aligned}$$

- Many extremely good off-the-shelf solvers, such as CPLEX and Gurobi

Linear programming relaxation for MAP

Integer linear program was:

$$\mathrm{MAP}(\theta) = \max_{\mu} \sum_{i \in V} \sum_{x_i} \theta_i(x_i)\mu_i(x_i) + \sum_{ij \in E} \sum_{x_i, x_j} \theta_{ij}(x_i, x_j)\mu_{ij}(x_i, x_j)$$

subject to

$$\begin{aligned}
\mu_i(x_i) &\in \{0, 1\} \quad \forall i \in V, x_i \\
\sum_{x_i} \mu_i(x_i) &= 1 \quad \forall i \in V \\
\mu_i(x_i) &= \sum_{x_j} \mu_{ij}(x_i, x_j) \quad \forall ij \in E, x_i \\
\mu_j(x_j) &= \sum_{x_i} \mu_{ij}(x_i, x_j) \quad \forall ij \in E, x_j
\end{aligned}$$

Relax integrality constraints, allowing the variables to be **between** 0 and 1:

$$\mu_i(x_i) \in [0, 1] \quad \forall i \in V, x_i$$

Linear programming relaxation for MAP

Linear programming relaxation is:

$$\mathrm{LP}(\theta) = \max_{\mu} \sum_{i \in V} \sum_{x_i} \theta_i(x_i)\mu_i(x_i) + \sum_{ij \in E} \sum_{x_i, x_j} \theta_{ij}(x_i, x_j)\mu_{ij}(x_i, x_j)$$

$$\begin{aligned}
\mu_i(x_i) &\in [0,1] \quad \forall i \in V, x_i \\
\sum_{x_i} \mu_i(x_i) &= 1 \quad \forall i \in V \\
\mu_i(x_i) &= \sum_{x_j} \mu_{ij}(x_i, x_j) \quad \forall ij \in E, x_i \\
\mu_j(x_j) &= \sum_{x_i} \mu_{ij}(x_i, x_j) \quad \forall ij \in E, x_j
\end{aligned}$$

- Linear programs can be solved **efficiently**!
- Since the LP relaxation maximizes over a **larger** set of solutions, its value can only be larger!

$$\mathrm{MAP}(\theta) \leq \mathrm{LP}(\theta)$$

Dual decomposition

- Consider the original discrete optimization problem:

$$\mathrm{MAP}(\theta) = \max_{\mathbf{x}} \sum_{i \in V} \theta_i(x_i) + \sum_{ij \in E} \theta_{ij}(x_i, x_j).$$

- If we push the maximizations *inside* the sums, the value can only *increase*:

$$\mathrm{MAP}(\theta) \leq \sum_{i \in V} \max_{x_i} \theta_i(x_i) + \sum_{ij \in E} \max_{x_i, x_j} \theta_{ij}(x_i, x_j)$$

- Recall from your homework that you can always *reparameterize* a distribution by operations like

$$\begin{aligned} \theta_i^{\mathrm{new}}(x_i) &= \theta_i^{\mathrm{old}}(x_i) + f(x_i) \\ \theta_{ij}^{\mathrm{new}}(x_i, x_j) &= \theta_{ij}^{\mathrm{old}}(x_i, x_j) - f(x_i) \end{aligned}$$

for **any** function $f(x_i)$, without changing the distribution

Dual decomposition

- Define:

$$\begin{aligned}\tilde{\theta}_i(x_i) &= \theta_i(x_i) + \sum_{ij \in E} \delta_{j \to i}(x_i) \\ \tilde{\theta}_{ij}(x_i, x_j) &= \theta_{ij}(x_i, x_j) - \delta_{j \to i}(x_i) - \delta_{i \to j}(x_j)\end{aligned}$$

- It is easy to verify that

$$\sum_i \theta_i(x_i) + \sum_{ij \in E} \theta_{ij}(x_i, x_j) = \sum_i \tilde{\theta}_i(x_i) + \sum_{ij \in E} \tilde{\theta}_{ij}(x_i, x_j) \quad \forall \mathbf{x}$$

- Thus, we have that:

$$\mathrm{MAP}(\theta) = \mathrm{MAP}(\tilde{\theta}) \le \sum_{i \in V} \max_{x_i} \tilde{\theta}_i(x_i) + \sum_{ij \in E} \max_{x_i, x_j} \tilde{\theta}_{ij}(x_i, x_j)$$

- Every value of δ gives a different upper bound on the value of the MAP!
- The **tightest** upper bound can be obtained by minimizing the r.h.s. with respect to δ!

Dual decomposition

- We obtain the following **dual** linear program: $L(\delta) =$

$$\sum_{i \in V} \max_{x_i} \Big(\theta_i(x_i) + \sum_{ij \in E} \delta_{j \to i}(x_i) \Big) + \sum_{ij \in E} \max_{x_i, x_j} \Big(\theta_{ij}(x_i, x_j) - \delta_{j \to i}(x_i) - \delta_{i \to j}(x_j) \Big),$$

$$\mathrm{DUAL\text{-}LP}(\theta) = \min_{\delta} L(\delta)$$

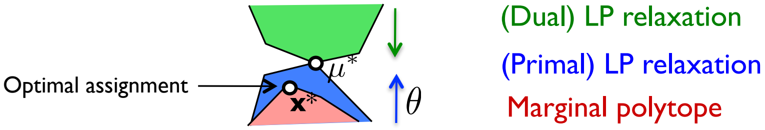
- We showed two ways of upper bounding the value of the MAP assignment:

$$\begin{aligned} \mathrm{MAP}(\theta) &\leq \mathrm{LP}(\theta) & (1) \\ \mathrm{MAP}(\theta) &\leq \mathrm{DUAL\text{-}LP}(\theta) \leq L(\delta) & (2) \end{aligned}$$

- The dual LP allows us to upper bound the value of the MAP assignment without solving a LP to optimality
- Although we derived these linear programs in seemingly very different ways, in turns out that:

$$\mathrm{LP}(\theta) = \mathrm{DUAL\text{-}LP}(\theta)$$

Linear programming duality



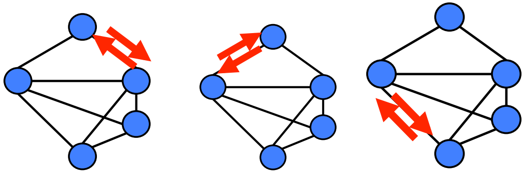
$$\mathrm{MAP}(\theta) \leq \mathrm{LP}(\theta) = \mathrm{DUAL\text{-}LP}(\theta) \leq L(\delta)$$

Solving the dual efficiently

- Many ways to solve the dual linear program, i.e. minimize with respect to δ:

$$\sum_{i \in V} \max_{x_i} \Big(\theta_i(x_i) + \sum_{ij \in E} \delta_{j \to i}(x_i) \Big) + \sum_{ij \in E} \max_{x_i, x_j} \Big(\theta_{ij}(x_i, x_j) - \delta_{j \to i}(x_i) - \delta_{i \to j}(x_j) \Big),$$

- One option is to use the subgradient method, as you saw in Lecture 3
- Can also solve using **block coordinate-descent**, which gives algorithms that look very much like max-sum belief propagation:



Max-product linear programming (MPLP) algorithm

Input: A set of factors $\theta_i(x_i), \theta_{ij}(x_i, x_j)$

Output: An assignment $x_1, \ldots, x_n$ that approximates the MAP

Algorithm:

- Initialize $\delta_{i\to j}(x_j) = 0, \quad \delta_{j\to i}(x_i) = 0, \quad \forall ij \in E, x_i, x_j$
- Iterate until small enough change in $L(\delta)$:

  For each edge $ij \in E$ (sequentially), perform the updates:

$$\begin{aligned} \delta_{j\to i}(x_i) &= -\frac{1}{2}\delta_i^{-j}(x_i) + \frac{1}{2}\max_{x_j}\left[\theta_{ij}(x_i,x_j) + \delta_j^{-i}(x_j)\right] \quad \forall x_i \\ \delta_{i\to j}(x_j) &= -\frac{1}{2}\delta_j^{-i}(x_j) + \frac{1}{2}\max_{x_i}\left[\theta_{ij}(x_i,x_j) + \delta_i^{-j}(x_i)\right] \quad \forall x_j \end{aligned}$$

  where $\delta_i^{-j}(x_i) = \theta_i(x_i) + \sum_{ik\in E, k\neq j} \delta_{k\to i}(x_i)$

- Return $x_i \in \arg\max_{\hat{x}_i} \tilde{\theta}_i^{\delta}(\hat{x}_i)$

Other approaches to solve MAP

- Local search
  - Greedily search over the space of assignments
  - Start from an arbitrary assignment (e.g., random). Iterate:
  - Choose a variable. Change a new state for this variable to maximize the value of the resulting assignment
- Branch-and-bound
  - Exhaustive search over space of assignments, pruning branches that can be provably shown not to contain a MAP assignment
  - Can use the LP relaxation or its dual to obtain upper bounds
  - Lower bound obtained from value of any assignment found along the way
- Branch-and-cut (most powerful method; used by CPLEX)
  - Same as branch-and-bound, except spend more time getting tighter bounds
  - Adds *cutting-planes* to cut off fractional solutions of the LP relaxation, making the upper bound tighter