Domain decomposition (or dual decomposition) methods

1) Apply to more than MRF problems.

2) General idea
   - Decompose problem into independent parts, by duplicating variables
   - Impose constraints to make the duplicated variables agree
   - Set up Lagrangian
   - Iterate:
     - Go down (up) primal vars
     - Go up (down) dual vars. Using subgradient descent
   - Now break ties.
Example:

\[
\begin{align*}
\min & \sum_i m_i(x_i) + \sum_{ij} m_{ij}(x_i, x_j) \\
\text{becomes} & \min m_1(u_1) + m_2(u_2) + m_3(u_3) + m_{12}(u_1, u_2) + m_{13}(u_1, u_3) \\
& + m_{23}(v_2, v_3) + m_{23}(v_2, v_3)
\end{align*}
\]

\[
\begin{align*}
\text{s.t.} & \quad u_2 = v_2, \quad u_3 = v_3 \\
\text{write} & \quad \min h_u(u_1, u_2, u_3) + h_v(v_2, v_3) \\
\text{s.t.} & \quad u_2 = v_2, \quad u_3 = v_3
\end{align*}
\]
Lagrangian:

\[ L(u, v, \lambda) = h_u(u_1, u_2, u_3) + h_v(v_2, v_3) + \lambda_1(u_2 - v_2) + \lambda_2(u_3 - v_3) \]

Dual:

\[ G(\lambda) = \inf_{(u, v)} L(u, v, \lambda) \leq \text{lower bound} \]

So we'd like \( \lambda^* \) to \( \max \ G(\lambda) \).

Strategy:
- For fixed \( \lambda^n \), evaluate \( G(\lambda) \)
- Now get subgradient, update \( \lambda^{n+1} \)

Evaluating \( G(\lambda) \):

- Find \( \min_{u, v} L(u, v, \lambda^n) \)
- By choice of \( h_u, h_v \), this is easy.

Update \( \lambda \):

\[ \nabla G(\lambda^n) = \text{subgradient} = \begin{pmatrix} u_2 - v_2 \\ u_3 - v_3 \end{pmatrix} \quad \rightarrow \quad \text{how take a small step} \]
Notice this is good for integer problems, because we have integer points.

But they may not agree at overlap.

- random allocation
- voting.

Thm: if they do agree at overlap, at convergence, we have exact soln.

(Fairly obvious)

Notice also: this is rather good for other problems.

**Important cases:**

- big problem, parallelism (origins)
- reliable local experts.
1) Parser, which is discriminative works by parses.

\[ \text{parse} = \underset{y}{\operatorname{argmax}} \ w^\top \phi(x, y) \]

2) Part of speech tagger, disc, Game mechanism.

\[ \text{tag} = \underset{z}{\operatorname{argmax}} \ w^\top \phi(x, z) \]

Now, notice parsing implies some sort of tagging.
- Each problem individually has quite a good solution.
- Each allows exact inference.
Idea:

a) \( y^* = (\hat{y}, z_p) \)

\[ \text{expose implicit pos into in parse.} \]

b) \[ \max \quad w_p \Phi(x, (\hat{y}, z_p)) + w_t \Psi(x, z_t) \]

\[ \text{s.t.} \quad z_p = z_t \]

c) this fits into previous recipe, but notice

- through constraints, expertise of tagger biases parser
- ditto, parser \ldots tagger.

d) improves performance.