Decomposition methods

Imagine we have to solve

\[
\min \ f(x_1, y) + f_2(x_2, y)
\]

If \( f_1, f_2 \) convex, \( x_1, x_2 \) big

not essential

We could attack this by

\[
\min \ \left[ \min_{x_1} f(x_1, y) \right] + \left[ \min_{x_2} f_2(x_2, y) \right]
\]

Which has the advantage that the inner problems can be done parallel.

Process:

Fix some \( y_0 \)

\[
\phi_1 = \min_{x_1} f(x_1, y_0)
\]

\[
\phi_2 = \min_{x_2} f_2(x_2, y_0)
\]

generate \( y_{n+1} \)
Usually thought of as a master problem and two (some) slave problems

slaves: \[ \phi_1(y) = \min_{x_1} f_1(x_1, y) \]
\[ \phi_2(y) = \min_{x_2} f_2(x_2, y) \]

master: \[ \min_Y \phi_1(y) + \phi_2(y) = \phi(y) \]

Now depending on \( f_1, f_2 \), master could have a variety of forms. Assume \( f_1, f_2 \) convex, smooth -- then \( \phi \): smooth, convex

so we could do gradient descent on \( \phi \)

Procedure:

choose \( y_0 \)

- solve slaves, return \[ \frac{\partial f_i}{\partial y} \bigg|_{x_i, y_n} \]

\[ y = y - \mu \nabla \phi = y - \mu \left[ \frac{\partial f_1}{\partial y} + \frac{\partial f_2}{\partial y} \right] \bigg|_{x_1, y_n} \]

Clearly works for more than one slave.
Notice because this is a form of coordinate ascent, there can be trouble

\[ f_1(x_1, y) = (x+y)^2 + \varepsilon(x-y)^2 \]

\[ f_2(x_2, y) = (x-y)^2 + \varepsilon(x+y)^2 \]

\[ y_0 = 1 \]

then

\[ x_1 = -\left(\frac{1-\varepsilon}{1+\varepsilon}\right) \]

\[ x_2 = \left(\frac{1-\varepsilon}{1+\varepsilon}\right) \]

So

\[ \phi(y) = \left(1+\varepsilon\right)\left(2\left(y - \frac{1-\varepsilon}{1+\varepsilon}\right)^2 + 2\varepsilon\left(y + \left(\frac{1-\varepsilon}{1+\varepsilon}\right)^2\right) \right) \]

and min is at \( y \sim 1 - 4\varepsilon + O(\varepsilon^2) \).

\[ \rightarrow \text{ this means slow steps } f_2 \]
Alternative:

\[
\min f_1(x_1, y_1) + f_2(x_2, y_2)
\]

\[s.t. \quad y_1 = y_2\]

Consider Dual

Lagrangian is 
\[L(\lambda, x_1, y_1, x_2, y_2)\]
\[= f_1 + f_2 + \lambda (y_1 - y_2)\]

Dual is 
\[
g(\lambda) = \inf_{x_1, y_1, x_2, y_2} L(\lambda, x_1, y_1, x_2, y_2)
\]

and dual is a lower bound on

\[
V(x_1, y_1, x_2, y_2) \geq g(\lambda)
\]

So now we can do subgradient ascent on dual to max dual.

Note dual is envelope of linear functions if may be is convex, but may not be smooth.
\[ -x^*, x^0, y^0, \text{ etc.} \]

**Slaves:**

\[
\min_{x_i, y_i} f_i(x_i^*, y_i^*) + \lambda^n y_i
\]

**Master:**

\[
\triangledown g(\lambda) = y_i^n - y_j^{n+1}
\]

\[ x^{n+1} = x^n + \mu^n g(\lambda) \]

Notice. When we have evaluated slaves, we get a value for the primal.

- Master gives us a value for dual.
- If we happen to encounter \( x_1, y_1, x_2, y_2 \), s.t.
  \[ v(x_1, y_1, x_2, y_2) = g(\lambda) \]
  this point is a soln.:
Notice we can deal with probs coupled by constraints

\[
\begin{align*}
\min & \quad f_1(x_1) + f_2(x_2) \\
\text{s.t.} & \quad h_1(x_1) + h_2(x_2) \leq 0
\end{align*}
\]

Write Lagrangian

\[
\mathcal{L}(x_1, x_2, \lambda) = f_1(x_1) + \lambda^T h_1(x_1) + f_2(x_2) + \lambda^T h_2(x_2)
\]

That is:

\[
\mathcal{L}(\lambda) = \max_{x_1, x_2} \mathcal{L}(x_1, x_2, \lambda).
\]

So are max dual

notice for some value of \( \lambda \), then

\[
\frac{dg}{\lambda^k} \quad \text{argmin} \quad f_1(x_1) + \lambda^* h_1(x_1)
\]

write \( x_1^*, x_2^* \) for

\[
\text{argmin} \quad f_2(x_2) + \lambda^* h_2(x_2)
\]

then

\[
\frac{dg}{\lambda^*} = h_1(x_1^*) + h_2(x_2^*)
\]
So our procedure becomes

- **Start w/ some $\lambda_0$**

  - **Slaves**
    \[
    \arg \min_{x_i} f(x_i) + \lambda \ h_i(x_i)
    \]

  - **Master**
    \[
    \lambda_{n+1} \rightarrow \lambda_n + \alpha \ [h_1(x_1^*) + h_2(x_2^*)]
    \]

  if we encounter $x_1^*$, $x_2^*$, $\lambda^*$ s.t.

  \[
  f_1(x_1^*) + f_2(x_2^*) = g(\lambda^*)
  \]

  we are done.
We can apply this strategy to discrete problems, too (main issue is notation)

- Simple example

```
MLF:  
```

```
\begin{align*}
&\sum_i \Theta_i(x_i) \\
&\quad + \Theta_{12}(x_1, x_2) \\
&\quad + \Theta_{13}(x_1, x_3) \\
&\quad + \Theta_{23}(x_2, x_3) \\
&\quad + \Theta_{34}(x_3, x_4) \\
&\quad + \Theta_{24}(x_2, x_4)
\end{align*}
```

\( x_i \in \{\text{labels}\} \cup \{0, 1, 3\} \)

Now for decomposed problem

\[
L = C_{T_1} + C_{T_2} + \lambda_3 (x_3 - x_3') + \lambda_4 (x_4 - x_4')
\]

Notice: Each component is a tree

- So slaves are easy

Notice Ew's change the unary forms
For more interesting cases, we need to construct some notation.

- For consistency w/ Soataj et al. review, I will max

- Divide problem into unary terms and (k>1)-ary terms, called factors

- Each factor gets its own copy of each variable that appears in the factor

- vars: \( x_i \) \( \leftarrow \) \( i \)th variable, version for unary term

- \( x_i^f \) \( \leftarrow \) \( i \)th variable, copy that likes \( ai \) factor \( f \)

- \( x_f \) \( \leftarrow \) all the vars for factor \( f \), (that factor's copies)
With that notation, we can write

$$\max_{x_i, x_f} \sum_i \Theta_i(x_i) + \sum_f \Theta_f(x_f)$$

s.t. \( \mathbb{1}(x_i = l) - \mathbb{1}(x_f^i = l) = 0 \)

\( \forall i, f, l \)

(Notice how this deals w/ non 0-1 labels)

(eq)

\begin{tikzpicture}

% Drawing the graph

% Nodes

\node (1) at (0,0) {1};
\node (2) at (1,1) {2};
\node (3) at (0,1) {3};
\node (4) at (1,0) {4};

% Edges

\draw (1) -- (2);
\draw (1) -- (3);
\draw (2) -- (4);
\draw (3) -- (4);

% Constraints

\node at (2.5, 0.5) {\textbf{constraints}};
\end{tikzpicture}
This means we have rather a lot of Lagrange multipliers (one per pair of states for each constraint link)

This justifies seeing the L.M.'s as messages

You can see a link to D.P. fairly clearly here.

Issues

When to stop?

If you encounter a solution (Primal = Dual)

No progress on dual
- which $x_i$ values?
  - at random

In some cases, fair approx is obvious (follows)

How to decompose?

Per problem.
Example: making parse trees and P.O.S tags consistent (Rush & coll/

. Parsing

United flies some large jets

\[
\begin{array}{c}
N \\
\text{NP} \\
\text{NP} \\
\text{VP} \\
S
\end{array}
\]

. There are methods to construct such trees by

\[
y^* = \text{argmax}_{y \in \text{Trees}} f(y)
\]
Tagging:

United flies some large jets

\[ N \quad V \quad D \quad A \quad N \]

\[ \text{noun} \quad \text{verb} \quad \text{determiner} \quad \text{adjective} \quad \text{noun} \]

there are methods to produce a set of tags by

\[ z^* = \operatorname{argmax}_{z \in \text{Tags}} q(z) \]

Now imagine we want to make the tags consistent with the parse.

define \( l(y) = \{ \text{strip tree from } y, \text{ produce tag sequence} \} \)

then we are interested in

\[ \operatorname{argmax}_{y \in \text{Trees}, \ z \in \text{Tags}} f(y) + g(z) \]

s.t. \( z = l(y) \)
We know how to deal with this

\[ L(s, y, z) = f(y) + g(z) + S^T [z - l(y)] \]

\[ \Gamma(s) = \sup_{y, z, l} L(s, y, z). \]

and \( \Gamma(s) \) \( \geq \) Value of primal

so \( \min \Gamma(s) \)

\( \Rightarrow \) this gives us slaves

\[ \arg\max_y f(y) - S^T l(y) \]

\[ \arg\max_z g(z) + S^T z \]

\( \Rightarrow \) master \( \min \Gamma(s) \) - subgradient is easy.

IDEA: force agreement between multiple parsing models