An introduction to energy optimization in SMS++
Part III: a quick recap of decomposition techniques

Antonio Frangioni

Dipartimento di Informatica, Università di Pisa

EdF Labs — May 25-26, 2023
Meta–Outline

- **Part I: SMS++ basics & energy-related components**
- **Part II: hands-on with SMS++ for energy optimization**
- **Part III: a quick recap of decomposition techniques**
- **Part IV: decomposition & energy optimization in SMS++**
Outline – Part III

1. Dual decomposition (Dantzig-Wolfe/Lagrangian/Column Generation)
2. Primal decomposition (Benders’/Resource)
3. All Are One, One Is All
4. Dual Decomposition: the Nonlinear and Integer Cases
5. Alternative Good Formulations for $\text{conv}(X)$
6. Primal Decomposition: the Nonlinear and Integer Cases
7. Decomposition-aware modelling systems
8. Conclusions (for now)
Dual decomposition, a.k.a. Inner Approximation
Dantzig-Wolfe decomposition
Lagrangian Relaxation
Column Generation
Block-diagonal Convex (Linear) Program

- Block-diagonal program: convex $X$, $n$ “complicating” constraints

\[
(\Pi) \quad \max \{ cx : Ax = b, \ x \in X \}
\]

e.g, $X = \{ x : Ex \leq d \} = \bigotimes_{k \in K} \left( X^k = \{ x^k : E^k x^k \leq d^k \} \right)$

($|K|$ large $\Rightarrow$ $(\Pi)$ very large), $Ax = b$ linking constraints

- We can efficiently optimize upon $X$ (much more so than solving the whole of $(\Pi)$, anyway) for different reasons:
  - a bunch of (many, much) smaller problems instead of a large one
  - $X$ has (the $X^k$ have) structure (shortest path, …)

- We could efficiently solve $(\Pi)$ if linking constraints were not there

- But they are (there): how to exploit it?
Dantzig-Wolfe reformulation

Dantzig-Wolfe reformulation\textsuperscript{1}: $X$ convex $\iff$ represent it by points

$$X = \{ x = \sum_{\bar{x} \in X} \bar{x} \theta_{\bar{x}} : \sum_{\bar{x} \in X} \theta_{\bar{x}} = 1, \; \theta_{\bar{x}} \geq 0 \; \bar{x} \in X \}$$

then reformulate $(\Pi)$ in terms of the convex multipliers $\theta$

\[
\begin{align*}
\text{(\Pi)} & \quad \begin{cases} 
\max & c \left( \sum_{\bar{x} \in X} \bar{x} \theta_{\bar{x}} \right) \\
A \left( \sum_{\bar{x} \in X} \bar{x} \theta_{\bar{x}} \right) & = b \\
\sum_{\bar{x} \in X} \theta_{\bar{x}} & = 1, \; \theta_{\bar{x}} \geq 0 \; \bar{x} \in X 
\end{cases}
\end{align*}
\]

\begin{itemize}
\item only $n + 1$ rows, but $\infty$-ly many columns
\item note that “$\bar{x} \in X$” is an index, not a constraint ($\theta$ is the variable)
\end{itemize}

A rather semi-infinite program, but “only” $\bar{x} \in \text{ext } X$ needed

Not that this makes it any less infinite, unless $X$ is a polytope (compact polyhedron) $\iff$ finite set of vertices

Could this ever be a good idea? Actually, it could: polyhedra may have few faces and many vertices . . . or vice-versa

\[ \begin{align*}
\text{n-cube} & \quad |x_i| \leq 1 \quad \forall i \quad 2n \text{ faces} \quad 2^n \text{ vertices} \\
\text{n-co-cube} & \quad \sum_i |x_i| \leq 1 \quad 2^n \text{ faces} \quad 2n \text{ vertices}
\end{align*} \]

Except, most often the number of vertices is too large

\[ \begin{align*}
Ax = b \\
Ex \leq d
\end{align*} \]

\[ \begin{align*}
AX\theta = b \\
e\theta = 1
\end{align*} \]

a (linear) program with (exponentially/infinitely) many columns

But, efficiently optimize over \( X \Longrightarrow \) generate vertices (≡ columns)
Dantzig-Wolfe decomposition $\equiv$ Column Generation

- $B \subset X$ (small), solve restriction of $(\Pi)$ with $X \rightarrow B$, i.e.,

$$
(\Pi_B) \quad \begin{cases} 
\max \sum_{\bar{x} \in B} (c\bar{x}) \theta_{\bar{x}} \\
\sum_{\bar{x} \in B} (A\bar{x}) \theta_{\bar{x}} = b \\
\sum_{\bar{x} \in B} \theta_{\bar{x}} = 1, \quad \theta_{\bar{x}} \geq 0 
\end{cases} \quad \bar{x} \in B
$$

- "master problem" ($B$ small, not too costly)
- note how the parentheses have moved: linearity is needed (for now)

- If $B$ contains the "right" columns, $x^* = \sum_{\bar{x} \in B} \bar{x} \theta_{\bar{x}}^*$ optimal for $(\Pi)$

- How do I tell if $B$ contains the "right" columns? Use duality

$$
(\Delta_B) \quad \min \{ yb + \nu : \nu \geq c\bar{x} - y(A\bar{x}) \quad \bar{x} \in B \} 
$$

$$
= \min \{ f_B(y) = \max \{ c\bar{x} + y(b - A\bar{x}) : \bar{x} \in B \} \} 
$$

one constraint for each $\bar{x} \in B$
Dantzig-Wolfe decomposition $\equiv$ Lagrangian relaxation

- Dual of $(\Pi)$: $(\Delta) \equiv (\Delta_X)$ (many constraints)
- $f_B = \text{lower approximation of Lagrangian function}$
  \[
  (\Pi_y) \quad f(y) = \max \left\{ c x + y (b - Ax) : x \in X \right\} \geq f_B(y)
  \]
- Assumption: optimizing over $X$ is “easy” for each objective $\implies$ obtaining $\bar{x}$ s.t. $f(y) = c\bar{x} + y(b - A\bar{x})$ is “easy”
- Important: $(\Pi_y)$ Lagrangian relaxation
  \[
  f(y) \geq v(\Pi) = v(\Delta) \quad \forall y
  \]
  provided $(\Pi_y)$ is solved exactly, or at least a $\bar{f} \geq f(y)$ is used
- Thus, $(\Delta_B)$ outer approximation of the Lagrangian Dual
  \[
  (\Delta) \quad \min \left\{ f(y) = \max \left\{ c x + y (b - Ax) : x \in X \right\} \right\}
  \]

---

Lagrangian duality vs. Linear duality

- Note about the LP case \( X = \{ x : Ex \leq d \} \):
  
  \[
  \begin{align*}
  (\Delta) \quad & \min \left\{ yb + \max \left\{ (c - yA)x : Ex \leq d \right\} \right\} \\
  \quad & \equiv \min \left\{ yb + \min \left\{ wd : wE = c - yA , \ w \geq 0 \right\} \right\} \\
  \quad & \equiv \min \left\{ yb + wd : wE + yA = c , \ w \geq 0 \right\} \\
  \quad & \equiv \text{exactly the linear dual of } (\Pi)
  \end{align*}
  \]

- **y “partial” duals**: duals \( w \) of \( Ex \leq d \) “hidden” in the subproblem

- There is only one duality

- Will repeatedly come in handy
Dantzig-Wolfe decomposition \( \equiv \) Dual row generation

- Primal/dual optimal solution \( x^*/(v^*, y^*) \) out of \((\Pi_B)/(\Delta_B)\)

\( x^* \) feasible to \((\Pi)\), so optimal \(\iff\) \((v^*, y^*)\) feasible to \((\Delta)\)

\[ \iff v^* \geq (c - y^*A)x \quad \forall x \in X \]

\[ \iff v^* \geq \max \{ (c - y^*A)x : x \in X \} \]

In fact: \( v^* \geq (c - y^*A)x^* \equiv y^*b + v^* \geq f(y^*) \implies \)

\[ v(\Pi) \geq cx^* = y^*b + v^* \geq f(y^*) \geq v(\Delta) \geq v(\Pi) \implies x^*/(v^*, y^*) \) optimal

Otherwise, \( B = B \cup \{x^*\} \): add new column to \((\Pi_B)\) / row to \((\Delta_B)\), rinse & repeat

Clearly finite if \text{ext} \(X\) is, globally convergent anyway:

the Cutting-Plane algorithm for convex programs\(^3\) (applied to \((\Delta)\))

\(^3\) Kelley “The Cutting-Plane Method for Solving Convex Programs” J. of the SIAM, 1960
Geometry of the Lagrangian dual

\[ f_B \leq f \] (CP model),
Geometry of the Lagrangian dual

\[ f_B \leq f \text{ (CP model), } v^* = f_B(y^*) \text{ lower bound on } v(\Pi_B) \]
Geometry of the Lagrangian dual

- $f_B \leq f$ (CP model), $\nu^* = f_B(y^*)$ lower bound on $\nu(\Pi_B)$
- Optimal solution $\bar{x}$ gives separator between $(\nu^*, y^*)$ and $epi f \equiv (c\bar{x}, A\bar{x}) =$ new row in $(\Delta_B)$ (subgradient of $f$ at $y^*$)
Geometry of the Lagrangian dual

- $f_B \leq f$ (CP model), $\nu^* = f_B(y^*)$ lower bound on $\nu(\Pi_B)$

- Optimal solution $\bar{x}$ gives separator between $(\nu^*, y^*)$ and $\text{epi } f \equiv (c\bar{x}, A\bar{x}) =$ new row in $(\Delta_B)$ (subgradient of $f$ at $y^*$)

- Improve CP model,
Geometry of the Lagrangian dual

- $f_B \leq f$ (CP model), $v^* = f_B(y^*)$ lower bound on $v(\Pi_B)$

- Optimal solution $\bar{x}$ gives separator between $(v^*, y^*)$ and $\text{epi } f \equiv (c\bar{x}, A\bar{x}) = \text{new row in } (\Delta_B)$ (subgradient of $f$ at $y^*$)

- Improve CP model, re-solve the master problem, rinse & repeat
Dantzig-Wolfe decomposition $\equiv$ Inner Approximation

- “Abstract” view of $(\Pi_B)$: $\text{conv}(B)$ inner approximation of $X$

$$
(\Pi_B) \quad \max \left\{ cx : Ax = b , \ x \in \text{conv}(B) \right\}
$$

- $x^*$ solves the Lagrangian relaxation of $(\Pi_B)$ with $y^*$, i.e.,

$$
x^* \in \text{argmax} \left\{ (c - y^*A)x : x \in \text{conv}(B) \right\}
$$

$$
\implies (c - y^*A)x \leq (c - y^*A)x^* \text{ for each } x \in \text{conv}(B) \subseteq X
$$

- $(c - y^*A)\bar{x} = \max\{(c - y^*A)x : x \in X\} \geq (c - y^*A)x^*$

- Column $\bar{x}$ has positive reduced cost

$$
(c - y^*A)(\bar{x} - x^*) = (c - y^*A)\bar{x} - cx^* + y^*b = (c - y^*A)\bar{x} - v^* > 0
$$

$$
\implies \bar{x} \notin \text{conv}(B) \implies \text{makes sense to add } \bar{x} \text{ to } B
$$
Geometry of Dantzig-Wolfe/Column Generation

- $X_B = \text{conv}(B)$ inner approximation of $X$

A. Frangioni  (DI — UniPi)  SMS++ @ EdF  May 25-26, 2023  12 / 49
Geometry of Dantzig-Wolfe/Column Generation

\[ Ax = b \]

\[ c \prec y^* A \]

\( c - y^* A \) separates \( X_B \cap Ax = b \) from all \( x \in X \) better than \( x^* \)

Increase \( X_B \), re-solve master problem, rinse & repeat

Issue:

\( X_B \cap Ax = b \) must be nonempty
Geometry of Dantzig-Wolfe/Column Generation

- \( c - y^* A \) separates \( X_B \cap Ax = b \) from all \( x \in X \) better than \( x^* \)

\[ \implies \text{optimizing } c - y^* A \text{ finds new } \bar{x} \in X \text{ (if any)} \]
Geometry of Dantzig-Wolfe/Column Generation

- \( Ax = b \)
- \( c - y^* A \) separates \( X_B \cap Ax = b \) from all \( x \in X \) better than \( x^* \)
- \( \implies \) optimizing \( c - y^* A \) finds new \( \bar{x} \in X \) (if any)
- Increase \( X_B \),

\[ Ax = b \]
Geometry of Dantzig-Wolfe/Column Generation

- $c - y^*A$ separates $X_B \cap Ax = b$ from all $x \in X$ better than $x^*$.
- $\implies$ optimizing $c - y^*A$ finds new $\bar{x} \in X$ (if any).
- Increase $X_B$, re-solve master problem, rinse & repeat.
\[ c - y^* A \] separates \( X_B \cap Ax = b \) from all \( x \in X \) better than \( x^* \)

\[ \implies \text{optimizing } c - y^* A \text{ finds new } \bar{x} \in X \text{ (if any)} \]

- Increase \( X_B \), re-solve master problem, rinse & repeat

- Issue: \( X_B \cap Ax = b \) must be nonempty
The Unbounded Case

- $X$ unbounded $\iff$ $\text{rec } X \supset \{ 0 \} \implies f(y) = \nu(\Pi_y) = \infty$ happens

- $X = \text{conv}(\text{ext } X = X_0) + \text{cone}(\text{ext rec } X = X_\infty)$

- $B = (B_0 \subset X_0) \cup (B_\infty \subset X_\infty) = \{ \text{ points } \bar{x} \} \cup \{ \text{ rays } \bar{\chi} \} \implies$
  \[\max \left\{ \begin{array}{l}
c \left( \sum_{\bar{x} \in B_0} \bar{x} \theta_{\bar{x}} + \sum_{\bar{\chi} \in B_\infty} \bar{\chi} \theta_{\bar{\chi}} \right) \\
A \left( \sum_{\bar{x} \in B_0} \bar{x} \theta_{\bar{x}} + \sum_{\bar{\chi} \in B_\infty} \bar{\chi} \theta_{\bar{\chi}} \right) = b \\
\sum_{\bar{x} \in B_0} \theta_{\bar{x}} = 1 \\
\theta_{\bar{x}} \geq 0 \quad \bar{x} \in B_0, \quad \theta_{\bar{\chi}} \geq 0 \quad \bar{\chi} \in B_\infty\end{array} \right.\]

- $\text{In } (\Delta_B)$, constraints $y(A\bar{\chi}) \geq c\bar{\chi}$ (a.k.a. “feasibility cuts”)

- $\left(\Pi_{y^*}\right)$ unbounded $\iff (c - y^*A)\bar{\chi} > 0$ for some $\bar{\chi} \in \text{rec } X$
  (violated constraint) $\implies B_\infty = B_\infty \cup \{ \bar{\chi} \}$

- $(\Delta) = \min \{ f(y) : y \in Y \}$, $(\Pi_{y^*})$ provides either subgradients of $f$
  (a.k.a. “optimality cuts”), or violated valid inequalities for $Y^3$
Primal decomposition, a.k.a. Outer Approximation Benders’ decomposition Resource decomposition
Staircase-structured Convex (Linear) Program

- Staircase-structured program: convex $X$, “complicating” variables

  \[
  (\Pi) \quad \max \{ \ cx + ez : \ Dx + Ez \leq d , \ x \in X \} \]

  e.g, $Dx + Ez \leq d \equiv D_k x + E_k z_k \leq d_k \quad k \in K \ (|K| \text{ large}) \implies$

  \[
  Z(x) = \{ z : Ez \leq d - Dx \} = \bigotimes_{k \in K} (Z_k(x) = \{ z_k : E_k z_k \leq d_k - D_k x \})
  \]

- We can efficiently optimize upon $Z(x)$ (much more so than solving the whole of $(\Pi)$, anyway) for different reasons:
  - a bunch of (many, much) smaller problems instead of a large one
  - $Z(x)$ has (the $Z_k(x)$ have) structure (shortest path, ...)

- We could efficiently solve $(\Pi)$ if linking variables were fixed

- But they are not (fixed): how to exploit it?
**Benders’ reformulation**

- Benders' reformulation: define the concave value function

\[
\text{(B)} \quad \max \{ \, cx + v(x)\, \} = \max \{ \, ez : Ez \leq d - Dx \, \} : x \in X
\]

(note: clearly \(v(x) = -\infty\) may happen)

- Clever trick\(^4\): use duality to reformulate the inner problem

\[
v(x) = \min \{ \, w(d - Dx) \, : \, w \in W = \{ \, w : wE = e, w \geq 0 \, \} \}
\]

so that \(W\) does not depend on \(x\)

- As before, \(W = \text{conv}( \, \text{ext } W = W_0 \, ) + \text{cone}( \, \text{ext rec } W = W_\infty \, ) \Rightarrow \)

\[
\text{(B)} \quad \max cx + v \\
\quad v \leq \bar{w}(d - Dx) \quad \bar{w} \in W_0 \\
\quad 0 \leq \bar{\omega}(d - Dx) \quad \bar{\omega} \in W_\infty \\
\quad x \in X
\]

still very large, but we can generate \(\bar{w} / \bar{\omega}\) by computing \(v(x)\)

Benders’ decomposition

- Select (small) $\mathcal{B} = (\mathcal{B}_0 \subset \mathcal{W}_0) \cup (\mathcal{B}_\infty \subset \mathcal{W}_\infty)$, solve master problem

$$(\mathcal{B}_\mathcal{B}) \quad \max cx + v$$

A. Frangioni (DI — UniPi)
SMS++ @ EdF
May 25-26, 2023 16 / 49
All Are One, One Is All
Benders is Lagrange . . .

Block-diagonal case

\[(\Pi) \quad \max \{cx : Ax = b, Ex \leq d\}\]
\[(\Delta) \quad \min \{yb + wd : wE + yA = c, w \geq 0\}\]

Think of \(y\) as complicating variables in \((\Delta)\), you get

\[(\Pi) \quad \max \{cx : Ax = b, Ey \leq d\}\]
\[(\Delta) \quad \min \{yb + \min\{wd : wE = c - yA, w \geq 0\}\}\]
\[= \min \{yb + \max\{(c - yA)x : Ex \leq d\}\}\]

i.e., the Lagrangian dual of \((\Pi)\)

The value function of \((\Delta)\) is the Lagrangian function of \((\Pi)\)
...Lagrange is Benders ...

- Dual of $\Pi$ (linear case $X = \{ x : Ax = b \}$)

  $\Pi$ \( \max \{ cx + ez : Dx + Ez \leq d, Ax = b \} \)

  $\Delta$ \( \min \{ yb + wd : yA + wD = c, wE = e, w \geq 0 \} \)

  Lagrangian dual of the dual constraints $yA + wD = c$ (multiplier $x$):

  $\Delta$ \( \max \{ \min \{ yb + wd + (c - yA + wD)x : wE = e, w \geq 0 \} \} \)

  \[ = \max \{ cx + \min \{ y(b - Ax) + w(d - Dx) : wE = e, w \geq 0 \} \} \]

  \[ = \max \{ cx + \min \{ y(b - Ax) \} + \min \{ w(d - Dx) : wE = e, w \geq 0 \} \} \]

  \[ = \max \{ cx + \max \{ ez : Dx + Ez \leq e \} : Ax = b \} \]

  i.e., Benders’ reformulation of $\Pi$

- The Lagrangian function of $\Delta$ is the value function of $\Pi$
Both Lagrange and Benders boil down (changing sign if necessary) to
\[
\min \left\{ \phi(\lambda) : \lambda \in \Lambda \right\}
\]
with \( \Lambda \) and \( \phi \) convex, \( \phi \) nondifferentiable

Both \( \Lambda \) and \( \phi \) only implicitly known via a (costly) oracle: \( \tilde{\lambda} \mapsto \)
- either \( \phi(\tilde{\lambda}) < \infty \) and \( \tilde{g} \in \partial \phi(\tilde{\lambda}) \equiv \phi(\lambda) \geq \phi(\tilde{\lambda}) + \tilde{g}(\lambda - \tilde{\lambda}) \forall \lambda \)
- or \( \phi(\tilde{\lambda}) = \infty \) and a valid inequality for \( \Lambda \) violated by \( \tilde{\lambda} \)

“Natural” algorithm: the Cutting-Plane method\(^3\) \(\equiv\)
revised simplex method with mechanized pricing in the discrete case

Natural \(\iff\) fast: convex nondifferentiable optimization \(\Omega(1/\varepsilon^2)\), Cutting-Plane method much worse than that (will see soon)

Many variants/other algorithms possible, another story (course)
You can apply Lagrange to a Staircase-structured program

- Reformulate a staircase-structured program

  $$\text{max } cx + e' z' + e'' z''$$

  $$Dx + E' z' \leq d', \quad Dx + E'' z'' \leq d''$$

  $$x \in X$$
You can apply Lagrange to a Staircase-structured program

- Reformulate a staircase-structured program
  \[ \max cx + e'z' + e''z'' \]
  \[ Dx + E'z' \leq d', \ Dx + E''z'' \leq d'' \]
  \[ x \in X \]

  ...as a block-diagonal one
  \[ \max c(x' + x'')/2 + e'z' + e''z'' \]
  \[ Dx' + E'z' \leq d', \ x' \in X \]
  \[ Dx'' + E''z'' \leq d'', \ x'' \in X \]
  \[ x' = x'' \]

- Issue: \( Dx + Ez \leq d \) must have structure, not \( Ez \leq d - Dx \)

- Classical approach in stochastic programs
  (but beware the multi-stage case)
You can apply Benders’ to a Block-diagonal program

Reformulate a block-diagonal program

\[
\begin{align*}
\text{max } & \quad c'x' + c''x'' \\
E'x' & \leq d' \quad , \quad E''x'' \leq d'' \\
A'x' + A''x'' & = b
\end{align*}
\]

\[\text{Resource decomposition}^{5}\]

\[\text{in multicommodity parlance}\]

---

You can apply Benders’ to a Block-diagonal program

- Reformulate a block-diagonal program
  \[
  \max c' x' + c'' x'' \\
  E' x' \leq d', \ E'' x'' \leq d'' \\
  A' x' + A'' x'' = b
  \]

  \[. . . \text{as a staircase-structured one}\]
  \[
  \max c' z' + c'' z'' \\
  E' z' \leq d', \ A' z' = x' \\
  E'' z'' \leq d'', \ A'' z'' = x'' \\
  x' + x'' = b
  \]

- Issue: \(Ez \leq d, \ Az = x\) must have structure, not \(Ez \leq d\)

- Resource decomposition\(^5\) in multicommodity parlance

Dual Decomposition: 
the Nonlinear and Integer Cases
Block-diagonal Convex Nonlinear Programs

- Nonlinear $c(\cdot)$ concave, $A(\cdot)$ component-wise convex, $X$ convex

  \[
  (\Pi) \quad \max \ \{ \ c( x ) : A( x ) \leq b , \ x \in X \ \}
  \]

  \[
  (\Delta) \quad \max \ \{ \ f( y ) = yb + \max \ \{ \ c( x ) - yA( x ) : x \in X \ \} : y \geq 0 \ \}
  \]

- Any $\bar{x} \in X$ still gives $f( y ) \geq c( \bar{x} ) + y( b - A( \bar{x} ) )$, same $(\Delta_B) / (\Pi_B)$

- $yA( \bar{x} )$ still linear in $y$ even if nonlinear in $x$

\[
\begin{align*}
  c( \sum_{\bar{x} \in B} \bar{x} \theta_{\bar{x}} ) & \geq \sum_{\bar{x} \in B} c(\bar{x}) \theta_{\bar{x}} \quad (c(\cdot) \text{ concave}) + \\
  A( \sum_{\bar{x} \in B} \bar{x} \theta_{\bar{x}} ) & \leq \sum_{\bar{x} \in B} A(\bar{x}) \theta_{\bar{x}} \leq b \quad (A(\cdot) \text{ convex}) \implies (\Pi_B) \text{ safe inner approximation } (v(\Pi_B) \leq v(\Pi))
\end{align*}
\]

- Basically everything keeps working, but you may need constraint qualification\(^6\) (usually easy to get)

---

\(^6\) LeMaréchal, Hiriart-Urruty “Convex Analysis and Minimization Algorithms” Springer, 1993
Block-diagonal Nonconvex Nonlinear Programs

- $c(\cdot)$ and/or $A(\cdot)$ and/or $X$ not concave / convex: not much changes

---

Block-diagonal Nonconvex Nonlinear Programs

- \( c(\cdot) \) and/or \( A(\cdot) \) and/or \( X \) not concave / convex: not much changes except \((\Pi_y)\) is hard and you are not really solving \((\Pi)\)
- \( yA(\bar{x}) \) still linear in \( y \), \((\Delta)\) still convex \(\equiv\) “convexified” \((\Pi)\):
  \[
  c(x) = cx, \quad A(x) = Ax \implies (\Delta) \equiv \max \left\{ cx : Ax \leq b, \ x \in X^{**} \right\}
  \]
  (“**” \(\equiv\) biconjugate \(\equiv\) closed convex envelope / hull)

---

Block-diagonal Nonconvex Nonlinear Programs

- $c(\cdot)$ and/or $A(\cdot)$ and/or $X$ not concave / convex: not much changes except $(\Pi_Y)$ is hard and you are not really solving $(\Pi)$

- $yA(\bar{x})$ still linear in $y$, $(\Delta)$ still convex $\equiv$ “convexified” $(\Pi)$:

  \[ c(x) = cx, \quad A(x) = Ax \implies (\Delta) \equiv \max \left\{ cx : Ax \leq b, \ x \in X^{**} \right\} \]

  ("**" $\equiv$ biconjugate $\equiv$ closed convex envelope / hull)

  \[ A(x) = Ax \implies (\Delta) \equiv \max \left\{ c^{**}(x) : Ax \leq b \right\} \]

  ($c_X(\cdot) = c(\cdot) + \mathbb{1}_X(\cdot), \mathbb{1}_X \equiv$ indicator function $\equiv 0$ in $X$, $\infty$ outside)

  better than $\max \left\{ c^{**}(x) : A^{**}(x) \leq b, \ x \in X^{**} \right\}$

- General formula ugly to write\(^7\), but better than

  \[ \max \left\{ c^{**}(x) : A^{**}(x) \leq b, \ x \in X^{**} \right\} \]

- “A Lagrangian Dual does not distinguish a set from its convex hull”

  for better (efficiency) and for worse (not the same problem)

---

\(^7\) Lemaréchal, Renaud “A Geometric Study of Duality Gaps, with Applications” *Math. Prog.*, 2001
Block-diagonal Integer Programs

- **Special case: X combinatorial** (e.g., $X = \{ x \in \mathbb{Z}^n : Ex \leq d \}$)

  $$(\Pi) \quad \max \{ cx : Ax = b, \ x \in X \}$$

  $$(\Delta) \quad \min \{ yb + \max \{ (c - yA)x : x \in X \} \}$$

  nothing changes if we can still efficiently optimize over $X$ due to size (decomposition) and/or structure (integrality)

- Except we are solving a (potentially good) relaxation of $(\Pi)$

  $$(\bar{\Pi}) \begin{cases} \max \ c \left( \sum_{\bar{x} \in X} \bar{x} \theta_{\bar{x}} \right) \\ A \left( \sum_{\bar{x} \in X} \bar{x} \theta_{\bar{x}} \right) = b \\ \sum_{\bar{x} \in X} \theta_{\bar{x}} = 1 , \ \theta_{\bar{x}} \geq 0 \ \bar{x} \in X \end{cases}$$

  $\equiv \max \{ cx : Ax = b , \ x \in X^{**} = conv( X ) \}$

- $\theta_{\bar{x}} \in \mathbb{Z}$ gives a reformulation of $(\Pi)$; could branch on $\theta_{\bar{x}}$, but usually better doing it on $x$, easier to integrate in the relaxation computation
Block-diagonal Integer Programs (cont.d)

- **Good news:** $(\Pi)$ better (not worse) than continuous relaxation
  \[
  \text{conv}(X) \subseteq \{ x \in \mathbb{R}^n : Ex \leq d \}
  \]

- **Bad news:** $(\Pi_y)$ “too easy” \( \text{conv}(X) = \{ x \in \mathbb{R}^n : Ex \leq d \} \)
  \(\equiv\) integrality property) \(\implies\) $(\Pi)$ same as continuous relaxation

$(\Pi_y)$ must be easy, but not too easy (no free lunch)

Anyway, at best gives good bounds \(\implies\)
Branch & Bound with DW/Lagrangian/CG \(\equiv\) Branch & Price

Although it can be used to drive good heuristics\(^8,9\)

Branching nontrivial: may destroy subproblem structure
\(\implies\) branch on $x$ (but $(\Pi_B)$ is on $\theta$)

Little support from off-the-shelf tools, only SCIP / GCG\(^{10}\) (for now)

---

\(^8\) Daniilidis, Lemaréchal “On a Primal-Proximal Heuristic in Discrete Optimization” *Math. Prog.*, 2005

\(^9\) Scuzziato, Finardi, F. “Solving Stochastic […] Unit Commitment with a New Primal Recovery […]” *IJEPES*, 2021

\(^{10}\) [https://scipopt.org](https://scipopt.org), [https://gcg.or.rwth-aachen.de](https://gcg.or.rwth-aachen.de)
Digression: How to Choose your Lagrangian relaxation

- There may be many choices

  \((\Pi) \max \left\{ cx : Ax = b, Ex \leq d, x \in \mathbb{Z}^n \right\}\)

  \((\Pi'_y) \max \left\{ cx + y(b - Ax) : x \in X' = \{ x \in \mathbb{Z}^n : Ex \leq d \} \right\}\)

  \((\Pi''_w) \max \left\{ cx + w(d - Ex) : x \in X'' = \{ x \in \mathbb{Z}^n : Ax = b \} \right\}\)

- The best between \((\Delta')\) and \((\Delta'')\) depends on integrality of \(X', X'':\)
  - if both have it, both \((\Delta')\) and \((\Delta'')\) \(\equiv\) continuous relaxation
  - if only one has it, the one that does not, but if both don’t have it?

- Here comes Lagrangian decomposition\(^{11}\) (looks familiar?)

  \((\Pi) \equiv \max \left\{ (cx' + cx'')/2 : x' \in X', x'' \in X'', x' = x'' \right\}\)

  \((\Pi_\lambda) \max \left\{ (c/2 + \lambda)x' : x' \in X' \right\} + \max \left\{ (c/2 - \lambda)x'' : x'' \in X'' \right\}\)

  \((\bar{\Delta}) \equiv (\bar{\Pi}) \max \left\{ cx : x \in \text{conv}(X') \cap \text{conv}(X'') \right\}\)

- better than both (but need to solve two hard subproblems)

---

Intersection between red and blue ≡ grey ≡ continuous relaxation
Intersection between red and blue \equiv grey \equiv continuous relaxation

Lagrangian relaxation of blue constraints shrinks the red (\Rightarrow grey) part

Lagrangian relaxation of red constraints shrinks the blue (\Rightarrow grey) part

Lagrangian decomposition (both red and blue constraints) shrinks both \Rightarrow the grey part more

But the intersection of convex hulls is larger (bad) than \Rightarrow the convex hull of the intersection = still a relaxation in general
Geometry of Lagrangian Decomposition

- Intersection between red and blue $\equiv$ grey $\equiv$ continuous relaxation
- Lagrangian relaxation of blue constraints shrinks the red ($\Rightarrow$ grey) part

Intersection between red and blue $\equiv$ grey $\equiv$ continuous relaxation
Lagrangian relaxation of blue constraints shrinks the red ($\Rightarrow$ grey) part
Intersection between red and blue ≡ grey ≡ continuous relaxation

Lagrangian relaxation of blue constraints shrinks the red (⇒ grey) part

Lagrangian relaxation of red constraints shrinks the blue (⇒ grey) part

Lagrangian decomposition (both red and blue constraints) shrinks both =⇒ the grey part more

But the intersection of convex hulls is larger (bad) than the convex hull of the intersection =⇒ still a relaxation in general
Geometry of Lagrangian Decomposition

- Intersection between red and blue ≡ grey ≡ continuous relaxation
- Lagrangian relaxation of **blue** constraints shrinks the red (⇒ grey) part
- Lagrangian relaxation of **red** constraints shrinks the blue (⇒ grey) part

But the intersection of convex hulls is larger (bad) than the convex hull of the intersection =⇒ still a relaxation in general
Geometry of Lagrangian Decomposition

- Intersection between red and blue ≡ grey ≡ continuous relaxation
- Lagrangian relaxation of blue constraints shrinks the red (⇒ grey) part
- Lagrangian relaxation of red constraints shrinks the blue (⇒ grey) part
- Lagrangian decomposition (both red and blue constraints)
Intersection between red and blue \(\equiv\) grey \(\equiv\) continuous relaxation

Lagrangian relaxation of blue constraints shrinks the red (\(\Rightarrow\) grey) part

Lagrangian relaxation of red constraints shrinks the blue (\(\Rightarrow\) grey) part

Lagrangian decomposition (both red and blue constraints) shrinks both

\(\Rightarrow\) the grey part more
Geometry of Lagrangian Decomposition

- Intersection between red and blue = grey = continuous relaxation
- Lagrangian relaxation of blue constraints shrinks the red (⇒ grey) part
- Lagrangian relaxation of red constraints shrinks the blue (⇒ grey) part
- Lagrangian decomposition (both red and blue constraints) shrinks both
  ⇒ the grey part more
- But the intersection of convex hulls is larger (bad) than
  the convex hull of the intersection ⇒ still a relaxation in general
A Computational Example: Capacitated Facility Location

- Set $O$ of facilities to be installed, cost $f_i$ and capacity $u_i$ for $i \in O$
- Set $D$ of customers to be served, demand $d_j$ (unique product) for $j \in D$
- Unitary transport cost $c_{ij}$ on arc $(i,j) \in A$ (facility $i \rightarrow$ customer $j$)

\[
\begin{align*}
\min & \sum_{(i,j) \in A} c_{ij} d_j x_{ij} + \sum_{i \in O} f_i z_i \\
\text{s.t.} & \sum_{i : (i,j) \in A} x_{ij} = 1 & j \in D \\
& \sum_{j : (i,j) \in A} d_j x_{ij} \leq u_i z_i & i \in O \\
& x_{ij} \in [0,1] / \{0,1\} & (i,j) \in A \\
& z_i \in \{0,1\} & i \in O
\end{align*}
\]

- Splittable / unsplittable: customers can/not be served by $> 1$ facility
- $> 1$ products $\rightarrow$ multicommodity network design with very simple paths
Lagrangian Relaxations of Capacitated Facility Location

- **Relax (2):** \(|O|\) (mixed-integer) knapsacks

\[
\sum_{j \in D} y_j + \min \sum_{i \in O} \left[ \sum_{(i,j) \in A} (c_{ij}d_j - y_j)x_{ij} + f_i z_i \right] \\
\sum_{(i,j) \in A} d_j x_{ij} \leq u_i z_i \\
x_{ij} \in [0, 1] / \{0, 1\} \\
z_i \in \{0, 1\}
\]

- **Relax (3):** \(|O|\) 1-variable problems + \(|D|\) simple choice problems

\[
\min \sum_{j \in D} \sum_{(i,j) \in A} d_j(c_{ij} + w_i)x_{ij} + \sum_{i \in O}(f_i - w_i u_i)z_i \\
\sum_{(i,j) \in A} x_{ij} = 1 \\
x_{ij} \in [0, 1] / \{0, 1\} \\
z_i \in \{0, 1\}
\]

**Exercise:** which relaxation gives the best bound in the splittable / unsplittable case?
Dantzig-Wolfe Reformulation $\equiv$ Column Generation

- Column-generation view of the problem: patterns for facility $i$
  \[ \mathcal{P}^i = \{ p \in [0, 1]^{\mathcal{D}} : \sum_{j \in \mathcal{D}} d_j p_j \leq u_i , (i, j) \notin A \implies p_j = 0 \} \]
  except $p = 0$, + integrality if needed

- $p \in \mathcal{P}^i \implies c_p = f_i + \sum_{j : (i, j) \in A} c_{ij} d_j p_j, \mathcal{P} = \bigcup_{i \in O} \mathcal{P}^i$

- (disaggregated) Dantzig-Wolfe reformulation $\equiv$
  \[
  \begin{align*}
  \min & \sum_{i \in O} \sum_{p \in \mathcal{P}^i} c_p \theta_p \\
  \text{subject to} & \quad \sum_{p \in \mathcal{P}} p_j \theta_p = 1 \quad j \in \mathcal{D} \\
  \quad & \quad \sum_{p \in \mathcal{P}^i} \theta_p \leq 1 \quad i \in O \\
  \quad & \quad \theta_p \geq 0 \quad p \in \mathcal{P}
  \end{align*}
  \]

- D-W/CP: start with (small) $\mathcal{B}^i \subset \mathcal{P}^i$, solve (8)–(11) restricted to $\mathcal{B}$, take $y_i$ duals of (9), solve Lagrangian relaxations, rinse & repeat

- Eventually yields good bounds . . .
Column-generation view of the problem: patterns for facility $i$

$$\mathcal{P}^i = \{ p \in [0, 1]^{|D|} : \sum_{j \in D} d_j p_j \leq u_i, (i, j) \notin A \implies p_j = 0 \}$$

except $p = 0$, + integrality if needed

$$p \in \mathcal{P}^i \implies c_p = f_i + \sum_{j: (i, j) \in A} c_{ij} d_j p_j, \mathcal{P} = \bigcup_{i \in O} \mathcal{P}^i$$

(disaggregated) Dantzig-Wolfe reformulation $\equiv$

$$\min \sum_{i \in O} \sum_{p \in \mathcal{P}^i} c_p \theta_p$$

$$\sum_{p \in \mathcal{P}} p_j \theta_p = 1 \quad j \in D$$

$$\sum_{p \in \mathcal{P}^i} \theta_p \leq 1 \quad i \in O$$

$$\theta_p \geq 0 \quad p \in \mathcal{P}$$

D-W/CP: start with (small) $\mathcal{B}^i \subset \mathcal{P}^i$, solve (8)–(11) restricted to $\mathcal{B}$, take $y_i$ duals of (9), solve Lagrangian relaxations, rinse & repeat

Eventually yields good bounds . . . if the master problem is nonempty
Algorithmic Issues

- **Issue:** the master problem can be (primal) empty (≡ dual unbounded)

- **Phase 0 approach:** seek for feasible solution first

\[
\begin{align*}
\min & \sum_{j \in D} v_j \\
\sum_{p \in B} p_j \theta_p + v_j &= 1 & j \in D \\
\sum_{p \in B^i} \theta_p &\leq 1 & i \in O \\
\theta_p &\geq 0 & p \in B \\
v_j &\geq 0 & j \in D
\end{align*}
\]

- Minimise cost of slack variables \(v_j\), disregard true costs

- Ends with some \(v_j^* > 0\) ≡ DW reformulation \(\implies\) original problem empty

- Otherwise master problem feasible with \(B\), start “true” optimization
Algorithmic Issues

- Issue: the master problem can be (primal) empty (≡ dual unbounded)

- Phase 0 approach: seek for feasible solution first

\[
\begin{align*}
\min & \quad \sum_{j \in D} v_j \\
\text{s.t.} & \quad \sum_{p \in B} p_j \theta_p + v_j = 1 & j \in D \\
& \quad \sum_{p \in B^i} \theta_p \leq 1 & i \in O \\
& \quad \theta_p \geq 0 & p \in B \\
& \quad v_j \geq 0 & j \in D
\end{align*}
\]

- Minimise cost of slack variables \(v_j\), disregard true costs

- Ends with some \(v_j^* > 0\) \(\equiv\) DW reformulation \(\implies\) original problem empty

- Otherwise master problem feasible with \(B\), start “true” optimization

- Real issue: can take forever because D-W/CP inefficient

- And you have to do branching (Branch & Price) on top of that
Don’t try this at home, by-the book

How a by-the-book implementations behave:

- $y^*$ immediately shoots much farther from optimum than initial point
  - ≡ having good initial point not much useful
- No apparent improvement for a long time as information slowly accrues
- A mysterious threshold is hit and “real” convergence begins
Don’t try this at home, by-the-book

- How a by-the-book implementations behave:

- $y^*$ immediately shoots much farther from optimum than initial point ≡ having good initial point not much useful
- No apparent improvement for a long time as information slowly accrues
- A mysterious threshold is hit and “real” convergence begins
- Can be improved (stabilised), but that’s another story (course)
Alternative Good Formulations for \(\text{conv}(X)\)
Alternative Good Formulations for $conv(X)$

- (Under mild assumptions) $conv(X)$ is a polyhedron $\iff$
  
  \[
  conv(X) = \{ x \in \mathbb{R}^n : \tilde{E}x \leq \tilde{d} \} 
  \]

- There are (at least as) good (as DW) formulations for the problem in the natural variable space, which is an advantage

- Except, practically all good formulations are too large

- Very few exceptions (integrality property $\approx$ networks)

- Good news: rows can be generated incrementally

- But a few more variables do as a lot more constraints
Row generation/polyhedral approaches

\[ Ax = b \]

\[ Ax = b \]

\[ Ex \leq d \]

\[ Ax \leq b \cap Ex \leq d \text{ outer approximation of feasible region} \]
Ax = b

Ax = b

Ex ≤ d

Ax ≤ b \cap Ex ≤ d \text{ outer approximation} of feasible region

Optimal solution of continuous relaxation gives bound,
Row generation/polyhedral approaches

\[ Ax = b \]

\[ Ex \leq d \]

- \( Ax \leq b \cap Ex \leq d \) outer approximation of feasible region
- Optimal solution of continuous relaxation gives bound, valid inequality \( \equiv \) separator from \( \text{conv}(X) \)
Row generation/polyhedral approaches

- $Ax \leq b \cap Ex \leq d$ outer approximation of feasible region
- Optimal solution of continuous relaxation gives bound, valid inequality $\equiv$ separator from $\text{conv}(X)$
- Smaller feasible region,
Row generation/polyhedral approaches

- $Ax \leq b \cap Ex \leq d$ outer approximation of feasible region
- Optimal solution of continuous relaxation gives bound, valid inequality $\equiv$ separator from $\text{conv}(X)$
- Smaller feasible region, re-solve continuous relaxation, rinse & repeat
Example: Capacitated Facility Location

- **Strong forcing constraints** for Capacitated Facility Location

\[
\begin{align*}
\min \ (1) \\
(2), \ (3), \ (4), \ (5) \\
\sum_{i,j} x_{ij} & \leq d_j z_i \quad (i, j) \in A 
\end{align*}
\]

- Obviously valid, “only” \(\#A\) many \(\implies\) trivially separable

- \(\#A\) more constraints can make continuous relaxation unbearably slower

  \(\implies\) much better to separate them on-the-fly

- Just **lazy constraints** for solvers that support the notion

- Theoretical result: (15) \(\implies\) same bound as DW (in the splittable case)

- Many different ways to skin a cat (don’t do this at home!)
A picture is worth 100 words
Let’s see some code running
Branch & Cut

- \( R = \) (small) subset of row indices, \( E_R x \leq d_R \) reduced set

- Solve outer approximation to (\( \bar{\Pi} \))

\[
\begin{align*}
(\bar{\Pi}_R) & \quad \text{max} \{ c x : A x = b, E_R x \leq d_R \}
\end{align*}
\]

feed the separator with primal optimal solution \( x^* \)

- Separator for (several sub-families of) facets of \( \text{conv}(X) \)

- Several general approaches, countless specialized ones

- Most often separators are \textbf{hard combinatorial problems} themselves
  (though using general-purpose MIP solvers \textit{is} an option\(^{12}\))

- May \textbf{tail off}, \textbf{branching} useful far before having solved (\( \bar{\Pi}_X \))

\(^{12}\) Fischetti, Lodi, Salvagnin “Just MIP It!” \textit{MATHEURISTICS, Ann. Inf. Syst.}, 2009
Branch & Cut vs. Branch & Price

- Which is best?

- Row generation naturally allows multiple separators

- Very well integrated in general-purpose solvers
  (but harder to exploit “complex” structures)

- Column generation naturally allows very unstructured separators

- Simpler to exploit “complex” structures
  (but much less developed software tools)

- Column generation is row generation in the dual

- Then, of course, Branch & Cut & Price
  (nice, but software issues remain and possibly worsen)
Primal Decomposition: the Nonlinear and Integer Cases
Staircase-structured $z$-convex Nonlinear Programs

- $f(x, \cdot)$ and $G(x, \cdot)$ concave, $Z$ convex:
  
  $$(\Pi) \quad \max \{ f(x, z) : G(x, z) \geq 0, \ x \in X, \ z \in Z \}$$

  $$(B) \quad \max \{ v(x) : x \in X \}$$

  where $v(x) = \max \{ f(x, z) : G(x, z) \geq 0, \ z \in Z \}$

  $= \text{value function of a convex program} \implies \text{convex}$

- $(B) \equiv (\Pi)$ without assumptions on $f(\cdot, z)$, $G(\cdot, z)$ and $X$, i.e., if $(\Pi)$ is hard, then $(B)$ is just as hard as $(\Pi)$

- $(B)$ may still be more efficient (e.g., $x$ “very few” but $z$ “very many”)

- Standard example: $X = \{ x \in \mathbb{Z}^n : Ex \leq d \}$ combinatorial:
  
  $$(\Pi) \quad \max \{ cx + ez : Ax + Bz \leq b, \ x \in X \}$$

  nothing changes . . . except $(B_B)$ now is combinatorial $\implies$ hard

- However $(B_W)$ now is equivalent to $(\Pi) \implies$ no branching needed unless for solving $(B_B)$
Still need duality: which one? Lagrangian\(^{13}\), of course

\[ v(x) = \min \{ \max \{ f(x, z) + \lambda G(x, z) : z \in Z \} : \lambda \geq 0 \} \]

Under appropriate constraint qualification, two cases occur:

- either \( \exists \bar{\lambda} \geq 0, \bar{z} \in Z \) s.t. \( v(x^*) = f(x^*, \bar{z}) + \bar{\lambda} G(x^*, \bar{z}) > -\infty \)
- or \( v(x^*) = -\infty \implies \{ z \in Z : G(x^*, z) \geq 0 \} = \emptyset \implies \exists \bar{\nu} \geq 0, \bar{z} \in Z \) s.t. \( \max \{ \bar{\nu} G(x^*, z) : z \in Z \} = \bar{\nu} G(x^*, \bar{z}) < 0 \)

\(^{13}\)Geoffrion “Generalized Benders Decomposition” JOTA, 1972
Still need duality: which one? Lagrangian\textsuperscript{13}, of course

\[ v(x) = \min \left\{ \max \{ f(x, z) + \lambda G(x, z) : z \in Z \} : \lambda \geq 0 \right\} \]

Under appropriate constraint qualification, two cases occur:

either \( \exists \bar{\lambda} \geq 0, \bar{z} \in Z \) s.t. \( v(x^*) = f(x^*, \bar{z}) + \bar{\lambda}G(x^*, \bar{z}) > -\infty \)

or \( v(x^*) = -\infty \implies \{ z \in Z : G(x^*, z) \geq 0 \} = \emptyset \implies \exists \bar{\nu} \geq 0, \bar{z} \in Z \) s.t. \( \max \{ \bar{\nu}G(x^*, z) : z \in Z \} = \bar{\nu}G(x^*, \bar{z}) < 0 \)

General form of the master problem

\[
\begin{align*}
(B) \quad \max & \quad v \\
\text{s.t.} & \quad v \leq \max \{ f(x, z) + \bar{\lambda}G(x, z) : z \in Z \} \quad \bar{\lambda} \in \Lambda_0 \\
& \quad 0 \leq \max \{ \bar{\nu}G(x, z) : z \in Z \} \quad \bar{\nu} \in \Lambda_\infty \\
& \quad x \in X
\end{align*}
\]

\textsuperscript{13} Geoffrion “Generalized Benders Decomposition” \textit{JOTA}, 1972
Staircase-structured $z$-convex Nonlinear Programs (cont.d)

- Still need duality: which one? Lagrangian$^{13}$, of course
  \[ \nu(x) = \min \left\{ \max \{ f(x, z) + \lambda G(x, z) : z \in Z \} : \lambda \geq 0 \right\} \]

- Under appropriate constraint qualification, two cases occur:
  - either $\exists \bar{\lambda} \geq 0, \bar{z} \in Z$ s.t. $\nu(x^*) = f(x^*, \bar{z}) + \bar{\lambda} G(x^*, \bar{z}) > -\infty$
  - or $\nu(x^*) = -\infty \implies \{ z \in Z : G(x^*, z) \geq 0 \} = \emptyset \implies \exists \bar{\nu} \geq 0, \bar{z} \in Z$ s.t. $\max \{ \bar{\nu} G(x^*, z) : z \in Z \} = \bar{\nu} G(x^*, \bar{z}) < 0$

- General form of the master problem
  \[
  (B) \quad \max \nu \\
  \nu \leq \max \{ f(x, z) + \bar{\lambda} G(x, z) : z \in Z \} \quad \bar{\lambda} \in \Lambda_0 \\
  0 \leq \max \{ \bar{\nu} G(x, z) : z \in Z \} \quad \bar{\nu} \in \Lambda_\infty \\
  x \in X
  \]

- Beware those nasty “max”: must be that the “max” is independent of $x$!

- Possible in a few cases, complicated in general

13 Geoffrion “Generalized Benders Decomposition” JOTA, 1972
Case I, separability: $f(x, z) = f(x) + h(z)$, $G(x, z) = G(x) + H(z)$

\[(B) \quad \max f(x) + \nu \]

\[
\nu \leq \bar{\lambda}G(x) + \max \{ h(z) + \bar{\lambda}H(z) : z \in Z \} \quad \bar{\lambda} \in \Lambda_0
\]

\[
0 \leq \bar{\nu}G(x) + \max \{ \bar{\nu}G(z) : z \in Z \} \quad \bar{\nu} \in \Lambda_\infty
\]

$x \in X$

convex $\iff$ $f(\cdot)$ convex and $G(\cdot)$ concave $(\bar{\lambda} \geq 0, \bar{\nu} \geq 0)$, otherwise nonlinear nonconvex cuts, $(B)$ “hard” (but $(\Pi)$ was)

Case II, special forms: $f(z_i)$ concave, univariate

\[
\max \{ \sum_i x_i f(z_i) : \sum_i x_i z_i \leq c \ , \ z_i \geq 0 \ , \ Ax \leq b \ , \ x \geq 0 \}
\]

\[
\nu(x) = \min_{\lambda} \sum_i \max \{ x_i (f(z_i) - \lambda z_i) : z_i \geq 0 \} + \lambda c
\]

\[
\nu(x) \leq \sum_i x_i \max \{ (f(z_i) - \bar{\lambda} z_i) : z_i \geq 0 \} + \bar{\lambda}c
\]

can optimize on the $z$ independently from the $x$ $\implies$ “normal” linear cuts
Staircase-structured non convex Nonlinear Programs

\[ f(x, \cdot) \text{ and/or } G(x, \cdot) \text{ not concave and/or } Z \text{ not convex:} \]

Some workarounds possible:

- Use exact duality for nonconvex / integer problems
- Approximate the convex hull by some hierarchy (RLT, . . .)
- Give up duality and use combinatorial Benders’ (feasibility) cuts

In general much harder / less efficient

Alternative route: use Benders’ to solve continuous relaxation: Benders’ subproblem as yet another (strong)

14 Guzelsoy, Ralphs “Duality for Mixed-Integer Linear Programs” *ITOR*, 2007
Staircase-structured non convex Nonlinear Programs

- $f(x, \cdot)$ and/or $G(x, \cdot)$ not concave and/or $Z$ not convex:
  though luck: you basically cannot do anything

- Benders’ requires duality, duality requires convexity: no Benders’ for
  
  $$(\Pi) \quad \max \{ cx + ez : Ax + Bz \leq b , \ x \in X , \ z \in \mathbb{Z}^m \}$$

---

14 Guzelsoy, Ralphs “Duality for Mixed-Integer Linear Programs” ITOR, 2007
17 Costa, Cordeau, Gendron “Benders, Metric and Cutset Inequalities for Multicommodity […] Network Design” COAP, 2009
Staircase-structured non convex Nonlinear Programs

- \( f(x, \cdot) \) and/or \( G(x, \cdot) \) not concave and/or \( Z \) not convex:
  though luck: you basically cannot do anything

- Benders’ requires duality, duality requires convexity: no Benders’ for
  \[
  \max \{ cx + ez : Ax + Bz \leq b, \ x \in X, \ z \in \mathbb{Z}^m \}
  \]

Some workarounds possible:
- Use exact duality for nonconvex / integer problems\(^{14}\) (though!)
- Approximate the convex hull by some hierarchy\(^{15}\) (RLT, . . .)
- Give up duality and use combinatorial Benders’ (feasibility) cuts\(^{16}\)

In general much harder / less efficient

Alternative route: use Benders’ to solve continuous relaxation:
Benders’ subproblem as yet another (strong\(^{17}\)) cut generator

Often more efficient and supported by some off-the-shelf solver

---

\(^{14}\) Guzelsoy, Ralphs “Duality for Mixed-Integer Linear Programs” \textit{ITOR}, 2007

\(^{15}\) Sen, Sherali “Decomposition [. . .] for Two-Stage Stochastic Mixed-Integer Programming” \textit{Math. Prog.}, 2006


\(^{17}\) Costa, Cordeau, Gendron “Benders, Metric and Cutset Inequalities for Multicommodity [. . .] Network Design” \textit{COAP}, 2009
Outline

1. Dual decomposition (Dantzig-Wolfe/Lagrangian/Column Generation)
2. Primal decomposition (Benders’/Resource)
3. All Are One, One Is All
4. Dual Decomposition: the Nonlinear and Integer Cases
5. Alternative Good Formulations for $\text{conv}(X)$
6. Primal Decomposition: the Nonlinear and Integer Cases
7. Decomposition-aware modelling systems
8. Conclusions (for now)
Modelling languages, and what they are for

- Most interactions with optimization solvers via Algebraic Modelling Languages (AML): commercial AMPL or GAMS\textsuperscript{18}, AIMMS\textsuperscript{19} and OPL\textsuperscript{20}, or open-source Coliop or ZIMPL\textsuperscript{21}

- Interfaced with a varying set (few/many) of general-purpose solvers for large problem classes (MILP, MINLP, conic, \ldots )

- AML is a separate language, typically interpreted (not efficient)

- Mostly “flat” languages (no OOP), modularity an issue

- Focus on “model once, solve once”; some offer some support for iterative procedures but clearly an afterthought

- Hide the complexities of the model/solution process to inexperienced users

\textsuperscript{18}https://ampl.com, \; \textsuperscript{19}https://www.gams.com
\textsuperscript{19}https://www.aimms.com/platform/aimms-development
\textsuperscript{20}https://www.ibm.com/docs/en/icos/12.8.0.0?topic=opl-optimization-programming-language
\textsuperscript{21}http://www.coliop.org, \; https://zimpl.zib.de
Modelling systems, and what they are for

- Modelling systems: libraries written in general-purpose languages providing similar functionalities to AML

- Often open-source: FLOPCpp, COIN Rehearse and Gravity\(^{22}\) (C++), PuLP and Pyomo\(^{23}\) (Python), JuMP (Julia) and YALMIP\(^{24}\) (Matlab)

- May not fully replicate AML constructs, sometimes more limited

- Solver interfacing and overhead lower with efficient languages (C++)

- Multiple models and iterative procedures more natural

- Can exploit OOP features of host language for better modularity

- Mostly focus on general-purpose solvers and “model once, solve once”

- Tailored for end-users, not algorithms developers

---

\(^{22}\) https://github.com/coin-or/FlopCpp, https://github.com/coin-or/Gravity

\(^{23}\) https://github.com/coin-or/Rehearse

Decomposition / structure-aware solvers

- Some solvers provide decomposition capabilities:
  - Cplex does Benders’, structure automatic or user hints
  - SCIP\textsuperscript{10} does B&C&P (one-level D-W), pricing & reformulation up to the user (plugins)
  - GCG\textsuperscript{10} extends SCIP with automatic and user-defined (one-level) D-W and recently also a generic (one-level) Benders’ approach
  - DDSIP\textsuperscript{25} and PIPS\textsuperscript{26} implement D-W for two-stage stochastic programs
  - The BaPCoD B&C&P code has been used to develop Coluna.jl\textsuperscript{27}, doing one-level D-W and (alpha) Benders’, multi-level planned

- Other solvers use structure in different ways: BlockIP\textsuperscript{28}, OOPS\textsuperscript{29}

---

25 https://github.com/RalfGollmer/ddsip
26 https://github.com/Argonne-National-Laboratory/PIPS
27 https://github.com/atoptima/Coluna.jl
28 http://www-eio.upc.edu/~jcastro/BlockIP.html
29 https://www.maths.ed.ac.uk/~gondzio/parallel/solver.html
In a word?

- Decomposition-aware modelling systems: are there any?

- OOPS is interfaced with SML, providing some parallel capabilities.
- PIPS is interfaced with StructJuMP, using BlockDecomposition.

No modelling system is focused on multi-level structure, non-general-purpose solvers, parallel, and modularity/extendability.

Although JuMP is doing a good job at promoting some of these.

We tried working with Julia, but most solvers are in C/C++, and the full circle Julia → C++ → Julia did not work well.

So we choose no-performance-compromise C++, accepting the drawbacks.

---

30 https://www.maths.ed.ac.uk/ERGO/sml
31 https://github.com/StructJuMP/StructJuMP.jl
32 https://github.com/atoptima/BlockDecomposition.jl
Decomposition-aware modelling systems: are there any?

- In a word? Very few (that’s two words . . . )
- OOPS is interfaced with SML\(^{30}\), providing some parallel capabilities
- PIPS is interfaced with StructJuMP\(^{31}\), using BlockDecomposition\(^{32}\)

---

\(^{30}\) https://www.maths.ed.ac.uk/ERGO/sml
\(^{31}\) https://github.com/StructJuMP/StructJuMP.jl
\(^{32}\) https://github.com/atoptima/BlockDecomposition.jl
Decomposition-aware modelling systems: are there any?

- In a word? Very few (that’s two words . . . )
- OOPS is interfaced with SML\textsuperscript{30}, providing some parallel capabilities
- PIPS is interfaced with StructJuMP\textsuperscript{31}, using BlockDecomposition\textsuperscript{32}
- No modelling system is focused on multi-level structure, non-general-purpose solvers, parallel, and modularity/extendability
- Although JuMP is doing a good job at promoting some of these

\textsuperscript{30} https://www.maths.ed.ac.uk/ERGO/sml
\textsuperscript{31} https://github.com/StructJuMP/StructJuMP.jl
\textsuperscript{32} https://github.com/atoptima/BlockDecomposition.jl
In a word? Very few (that’s two words . . .)

OOPs is interfaced with SML\textsuperscript{30}, providing some parallel capabilities

PIPS is interfaced with StructJuMP\textsuperscript{31}, using BlockDecomposition\textsuperscript{32}

No modelling system is focused on multi-level structure, non-general-purpose solvers, parallel, and modularity/extendability

Although JuMP is doing a good job at promoting some of these

We tried working with Julia, but most solvers are in C / C++, and the full circle Julia $\rightarrow$ C++ $\rightarrow$ Julia did not work well

So we choose no-performance-compromise C++, accepting the drawbacks

\footnotesize{\textsuperscript{30}https://www.maths.ed.ac.uk/ERGO/sml\textsuperscript{31}https://github.com/StructJuMP/StructJuMP.jl\textsuperscript{32}https://github.com/atoptima/BlockDecomposition.jl}
Conclusions
(Part III)
Conclusions (Part III)

- General block structure can (and must in some cases) be exploited
Conclusions (Part III)

- General block structure can (and must in some cases) be exploited
- Well-understood main tools: reformulation + duality
Conclusions (Part III)

- General block structure can (and must in some cases) be exploited
- Well-understood main tools: reformulation + duality
- Two different approaches, “primal” and “dual”: for linear programs
  Lagrange is Benders’ in the dual, and vice-versa
Conclusions (Part III)

- General block structure can (and must in some cases) be exploited
- Well-understood main tools: reformulation + duality
- Two different approaches, “primal” and “dual”: for linear programs
  - Lagrange is Benders’ in the dual, and vice-versa

- Both boil down to the 60+-years old Cutting-Plane algorithm\(^3\)
  - “plus some branching” to deal with nonconvexity
Conclusions (Part III)

- General block structure can (and must in some cases) be exploited
- Well-understood main tools: reformulation + duality
- Two different approaches, “primal” and “dual”: for linear programs
  Lagrange is Benders’ in the dual, and vice-versa
- Both boil down to the 60+-years old Cutting-Plane algorithm\(^3\)
  “plus some branching” to deal with nonconvexity
- Different twists, different conditions to work:
Conclusions (Part III)

- General block structure can (and must in some cases) be exploited.
- Well-understood main tools: reformulation + duality.
- Two different approaches, “primal” and “dual”: for linear programs.
- Lagrange is Benders’ in the dual, and vice-versa.
- Both boil down to the 60+-years old Cutting-Plane algorithm\(^3\)
  “plus some branching” to deal with nonconvexity.
- Different twists, different conditions to work:
  - who is complicating (constraints vs. variables), but tricks (≡ other reformulations) can be used to create the desired structure.
Conclusions (Part III)

- General block structure can (and must in some cases) be exploited.
- Well-understood main tools: reformulation + duality.
- Two different approaches, "primal" and "dual": for linear programs. Lagrange is Benders’ in the dual, and vice-versa.
- Both boil down to the 60+-years old Cutting-Plane algorithm\(^3\) “plus some branching” to deal with nonconvexity.
- Different twists, different conditions to work:
  - who is complicating (constraints vs. variables), but tricks (≡ other reformulations) can be used to create the desired structure.
  - who is reformulated (subproblem vs. master problem).
Conclusions (Part III)

- General block structure can (and must in some cases) be exploited.
- Well-understood main tools: reformulation + duality.
- Two different approaches, “primal” and “dual”: for linear programs. Lagrange is Benders’ in the dual, and vice-versa.
- Both boil down to the 60+-years old Cutting-Plane algorithm \(^3\) “plus some branching” to deal with nonconvexity.
- Different twists, different conditions to work:
  - who is complicating (constraints vs. variables), but tricks (≡ other reformulations) can be used to create the desired structure.
  - who is reformulated (subproblem vs. master problem).
  - where integer / nonconvexity can be (subproblem vs. master problem).
Conclusions (Part III)

- General block structure can (and must in some cases) be exploited
- Well-understood main tools: reformulation + duality
- Two different approaches, “primal” and “dual”: for linear programs
  Lagrange is Benders’ in the dual, and vice-versa
- Both boil down to the 60+-years old Cutting-Plane algorithm^{3}
  “plus some branching” to deal with nonconvexity
- Different twists, different conditions to work:
  - who is complicating (constraints vs. variables), but tricks (≡ other reformulations) can be used to create the desired structure
  - who is reformulated (subproblem vs. master problem)
  - where integer / nonconvexity can be (subproblem vs. master problem)
  - where branching / cutting is done (subproblem vs. master problem)
Conclusions (Part III)

- General block structure can (and must in some cases) be exploited
- Well-understood main tools: reformulation + duality
- Two different approaches, “primal” and “dual”: for linear programs
  Lagrange is Benders’ in the dual, and vice-versa
- Both boil down to the 60+-years old Cutting-Plane algorithm\(^3\)
  “plus some branching” to deal with nonconvexity
- Different twists, different conditions to work:
  - who is complicating (constraints vs. variables), but tricks (≡ other
    reformulations) can be used to create the desired structure
  - who is reformulated (subproblem vs. master problem)
  - where integer / nonconvexity can be (subproblem vs. master problem)
  - where branching / cutting is done (subproblem vs. master problem)
  - where/which nonlinearities can be easily dealt with
Conclusions (Part III)

- General block structure can (and must in some cases) be exploited
- Well-understood main tools: reformulation + duality
- Two different approaches, “primal” and “dual”: for linear programs
- Lagrange is Benders’ in the dual, and vice-versa
- Both boil down to the 60+-years old Cutting-Plane algorithm\(^3\)
  “plus some branching” to deal with nonconvexity

Different twists, different conditions to work:
- who is complicating (constraints vs. variables), but tricks (≡ other reformulations) can be used to create the desired structure
- who is reformulated (subproblem vs. master problem)
- where integer / nonconvexity can be (subproblem vs. master problem)
- where branching / cutting is done (subproblem vs. master problem)
- where/which nonlinearities can be easily dealt with

But from theory to practice there is a large gulf to be crossed
Conclusions (Part III)

- General block structure can (and must in some cases) be exploited
- Well-understood main tools: reformulation + duality
- Two different approaches, “primal” and “dual”: for linear programs
  Lagrange is Benders’ in the dual, and vice-versa
- Both boil down to the 60+-years old Cutting-Plane algorithm\(^3\)
  “plus some branching” to deal with nonconvexity
- Different twists, different conditions to work:
  - who is complicating (constraints vs. variables), but tricks (≡ other reformulations) can be used to create the desired structure
  - who is reformulated (subproblem vs. master problem)
  - where integer / nonconvexity can be (subproblem vs. master problem)
  - where branching / cutting is done (subproblem vs. master problem)
  - where/which nonlinearities can be easily dealt with
- But from theory to practice there is a large gulf to be crossed
- Assume this is done for you (another story – course)