From Branching to Cutting (and back again)

Daniel Espinoza

Universidad de Chile

Newcastle, NSW, Australia
1 What’s wrong with branching?
   - Why we use it, why we don’t like it
   - What has been proposed?

2 A Local Cuts Approach
   - The main idea:

3 The Experiments
   - The instances
   - The results
   - Comparing with split closure
   - Final Remarks
Outline

1. What’s wrong with branching?
   - Why we use it, why we don’t like it
   - What has been proposed?

2. A Local Cuts Approach
   - The main idea:

3. The Experiments
   - The instances
   - The results
   - Comparing with split closure
   - Final Remarks
Introduction

- Become an indispensable tool in [DFJ 54].
- Branching (and bound and cut) it’s the best that we have.
  - Unless we actually know the polyhedral description of the set.
- But it tends to tail off.
- Is it because we generate too many sub-problems?
- Is it because we (don’t) know how to branch?
- Does it work only in problems where we have a reasonable good LP representation?
- Can we represent as cuts all (most) of what we learn in branching back in the root node?
  - For sure!
- Can we compute it in reasonable time?
- Should we ask for little less?
Introduction

- Become an indispensable tool in [DFJ 54].
- Branching (and bound and cut) it’s the best that we have.
  - Unless we actually know the polyhedral description of the set.
- But it tends to tail off.
  - Is it because it generate too many sub-problems?
  - Is it because we (don’t) know how to branch?
  - Does it work only in problems where we have a reasonable good LP representation?
- Can we represent as *cuts* all (most) of what we learn in branching back in the root node?
  - For sure!
- Can we compute it in reasonable time?
- Should we ask for little less?
Introduction

- Become an indispensable tool in [DFJ 54].
- Branching (and bound and cut) it’s the best that we have.
  - Unless we actually know the polyhedral description of the set.
- But it tends to tail off.
  - Is it because it generate too many sub-problems?
  - Is it because we (don’t) know how to branch?
  - Does it work only in problems where we have a reasonable good LP representation?
- Can we represent as cuts all (most) of what we learn in branching back in the root node?
  - For sure!
  - Should we ask for little less?
Why we use it, why we don’t like it

Introduction

- Become an indispensable tool in [DFJ 54].
- Branching (and bound and cut) it’s the best that we have.
  - Unless we actually know the polyhedral description of the set.
- But it tends to tail off.
  - Is it because it generate too many sub-problems?
  - Is it because we (don’t) know how to branch?
  - Does it work only in problems where we have a reasonable good LP representation?
  - Can we represent as cuts all (most) of what we learn in branching back in the root node?
  - Forward?
  - Can we compute it in reasonable time?
  - Should we ask for little less?
Why we use it, why we don’t like it

Introduction

- Become an indispensable tool in [DFJ 54].
- Branching (and bound and cut) it’s the best that we have.
  - Unless we actually know the polyhedral description of the set.
- But it tends to tail off.
  - Is it because it generate too many sub-problems?
  - Is it because we (don’t) know how to branch?
  - Does it work only in problems where we have a reasonable good LP representation?
  - Can we represent as *cuts* all (most) of what we learn in branching back in the root node?
    - For real?
    - Can we compute in reasonable time?
    - Should we ask for little less?
Introduction

- Become an indispensable tool in [DFJ 54].
- Branching (and bound and cut) it’s the best that we have.
  - Unless we actually know the polyhedral description of the set.
- But it tends to tail off.
  - Is it because it generate too many sub-problems?
  - Is it because we (don’t) know how to branch?
    - Does it work only in problems where we have a reasonable good LP representation?
- Can we represent as cuts all (most) of what we learn in branching back in the root node?
  - Can we compute it in reasonable time?
  - Should we ask for little less?
Introduction

- Become an indispensable tool in [DFJ 54].
- Branching (and bound and cut) it’s the best that we have.
  - Unless we actually know the polyhedral description of the set.
- But it tends to tail off.
  - Is it because it generate too many sub-problems?
  - Is it because we (don’t) know how to branch?
  - Does it work only in problems where we have a reasonable good LP representation?
- Can we represent as cuts all (most) of what we learn in branching back in the root node?
  - For sure!
  - Can we compute it in reasonable time?
  - Should we ask for little more?
Introduction

- Become an indispensable tool in [DFJ 54].
- Branching (and bound and cut) it's the best that we have.
  - Unless we actually know the polyhedral description of the set.
- But it tends to tail off.
  - Is it because it generate too many sub-problems?
  - Is it because we (don’t) know how to branch?
  - Does it work only in problems where we have a reasonable good LP representation?
- Can we represent as cuts all (most) of what we learn in branching back in the root node?
  - For sure!
  - Can we compute it in reasonable time?
  - Should we ask for little less?
Introduction

- Become an indispensable tool in [DFJ 54].
- Branching (and bound and cut) it’s the best that we have.
  - Unless we actually know the polyhedral description of the set.
- But it tends to tail off.
  - Is it because it generate too many sub-problems?
  - Is it because we (don’t) know how to branch?
  - Does it work only in problems where we have a reasonable good LP representation?
- Can we represent as cuts all (most) of what we learn in branching back in the root node?
  - For sure!
  - Can we compute it in reasonable time?
  - Should we ask for little less?
Introduction

- Become an indispensable tool in [DFJ 54].
- Branching (and bound and cut) it’s the best that we have.
  - Unless we actually know the polyhedral description of the set.
- But it tends to tail off.
  - Is it because it generate too many sub-problems?
  - Is it because we (don’t) know how to branch?
  - Does it work only in problems where we have a reasonable good LP representation?
- Can we represent as cuts all (most) of what we learn in branching back in the root node?
  - For sure!
  - Can we compute it in reasonable time?
  - Should we ask for little less?
Introduction

- Become an indispensable tool in [DFJ 54].
- Branching (and bound and cut) it’s the best that we have.
  - Unless we actually know the polyhedral description of the set.
- But it tends to tail off.
  - Is it because it generate too many sub-problems?
  - Is it because we (don’t) know how to branch?
  - Does it work only in problems where we have a reasonable good LP representation?
- Can we represent as *cuts* all (most) of what we learn in branching back in the root node?
  - For sure!
  - Can we compute it in reasonable time?
  - Should we ask for little less?
An Example: sw24978 (STSP)
To solve pla85900, [ABCCEGH 09] one technique was to iterate B&B&C:

**Require:** C original formulation for STSP

1. **loop**
2. Iterate B&B&C until tailing off.
3. Collect all globally valid cuts for STSP into C.
4. **C ← C ∩ {x : ax ≤ b, ∀ (a, b) e C}**

- Infeasibility propagation in SAT solvers (Achtberg).
- Cuts derived from infeasibility branches (Kilinç et al.).
- Pure branching: Disjunctive Programming (Balas et al.).
- CPLEX 11.0 onwards (sometimes) repeat the branching after a few nodes.
- They all fail to completely capture general partial B&B&C trees.
What has been proposed?

Some alternative approaches

To solve pla85900, [ABCCEGH 09] one technique was to iterate B&B&C:

Require: \( C \) original formulation for STSP

1: loop
2: Iterate B&B&C until tailing off.
3: Collect all globally valid cuts for STSP into \( C \).
4: \( C \leftarrow C \cap \{ x : ax \leq b, \forall (a, b) \in C \} \).

- Infeasibility propagation in SAT solvers (Achtberg).
- Cuts derived from infeasibility branches (Kilinç et al.).
- Pure branching: Disjunctive Programming (Balas et al.).
- CPLEX 11.0 onwards (sometimes) repeat the branching after a few nodes.
- They all fail to completely capture general partial B&B&C trees.
Some alternative approaches

To solve pla85900, [ABCCEGH 09] one technique was to iterate B&B&C:

Require: $C$ original formulation for STSP

1. **loop**
2. Iterate B&B&C until tailing off.
3. Collect all *globally valid* cuts for STSP into $C$.
4. $C \leftarrow C \cap \{ x : ax \leq b, \forall (a, b) \in C \}$.

- Infeasibility propagation in SAT solvers (Achtberg).
- Cuts derived from infeasibility branches (Kilinç et al.).
- Pure branching: Disjunctive Programming (Balas et al.).
- CPLEX 11.0 onwards (sometimes) repeat the branching after a few nodes.
- They all fail to completely capture general partial B&B&C trees.
Some alternative approaches

To solve pla85900, [ABCCEGH 09] one technique was to iterate B&B&C:

**Require:** $C$ original formulation for STSP

1. **loop**
2. Iterate B&B&C until tailing off.
3. Collect all *globally valid* cuts for STSP into $C$.
4. $C \leftarrow C \cap \{x : ax \leq b, \forall (a, b) \in C\}$.

- Infeasibility propagation in SAT solvers (Achtberg).
- Cuts derived from infeasibility branches (Kilinç et al.).
- Pure branching: Disjunctive Programming (Balas et al.).
- CPLEX 11.0 onwards (sometimes) repeat the branching after a few nodes.

They all fail to completely capture general partial B&B&C trees.
What has been proposed?

**Some alternative approaches**

To solve pla85900, [ABCCEGH 09] one technique was to *iterate* B&B&C:

**Require:** $C$ original formulation for STSP

1. **loop**
2. Iterate B&B&C until tailing off.
3. Collect all *globally valid* cuts for STSP into $C$.
4. $C \leftarrow C \cap \{x : ax \leq b, \forall (a, b) \in C\}$.

- Infeasibility propagation in SAT solvers (Achtberg).
- Cuts derived from infeasibility branches (Kilinç et al.).
- Pure branching: Disjunctive Programming (Balas et al.).
- CPLEX 11.0 onwards (sometimes) repeat the branching after a few nodes.
- They all fail to completely capture general partial B&B&C trees.

Daniel Espinoza  
From Branching to Cutting (and back again)
What's wrong with branching?

A Local Cuts Approach

The Experiments

What has been proposed?

Some alternative approaches

To solve pla85900, [ABCCEGH 09] one technique was to iterate B&B&C:

**Require:** C original formulation for STSP

1: **loop**
2: Iterate B&B&C until tailing off.
3: Collect all *globally valid* cuts for STSP into $C$.
4: $C \leftarrow C \cap \{x : ax \leq b, \forall (a, b) \in C\}$.

- Infeasibility propagation in SAT solvers (Achtberg).
- Cuts derived from infeasibility branches (Kilinç et al.).
- Pure branching: Disjunctive Programming (Balas et al.).
- CPLEX 11.0 onwards (sometimes) repeat the branching after a few nodes.
- They all fail to completely capture general partial B&B&C trees.
Some alternative approaches

To solve pla85900, [ABCCEGH 09] one technique was to iterate B&B&C:

Require: $C$ original formulation for STSP

1: **loop**
2: Iterate B&B&C until tailing off.
3: Collect all *globally valid* cuts for STSP into $C$.
4: $C \leftarrow C \cap \{x : ax \leq b, \forall (a, b) \in C\}$.

Infeasibility propagation in SAT solvers (Achtberg).

- Cuts derived from infeasibility branches (Kilinç et al.).
- Pure branching: Disjunctive Programming (Balas et al.).
- CPLEX 11.0 onwards (sometimes) repeat the branching after a few nodes.
- They all fail to completely capture general partial B&B&C trees.
What has been proposed?

Some alternative approaches

- To solve pla85900, [ABCCEGH 09] one technique was to iterate B&B&C:

  **Require:**  $C$ original formulation for STSP

  1. **loop**
  2. Iterate B&B&C until tailing off.
  3. Collect all *globally valid* cuts for STSP into $C$.
  4. $C \leftarrow C \cap \{x : ax \leq b, \forall (a, b) \in C\}$.

- Infeasibility propagation in SAT solvers (Achtberg).
- Cuts derived from infeasibility branches (Kilinç et al.).
- Pure branching: Disjunctive Programming (Balas et al.).
- CPLEX 11.0 onwards (sometimes) repeat the branching after a few nodes.
- They all fail to completely capture general partial B&B&C trees.
What has been proposed?

Some alternative approaches

- To solve pla85900, [ABCCEGH 09] one technique was to iterate B&B&C:
  
  **Require:** $C$ original formulation for STSP
  
  1: **loop**
  2: Iterate B&B&C until tailing off.
  3: Collect all *globally valid* cuts for STSP into $C$.
  4: $C \leftarrow C \cap \{x : ax \leq b, \forall (a, b) \in C\}$.

- Infeasibility propagation in SAT solvers (Achtberg).
- Cuts derived from infeasibility branches (Kilinç et al.).
- Pure branching: Disjunctive Programming (Balas et al.).
- CPLEX 11.0 onwards (sometimes) repeat the branching after a few nodes.
- They all fail to completely capture general partial B&B&C trees.
What has been proposed?

**Some alternative approaches**

- To solve pla85900, [ABCCEGH 09] one technique was to **iterate** B&B&C:
  
  **Require:** $C$ original formulation for STSP
  1. **loop**
  2. Iterate B&B&C until tailing off.
  3. Collect all *globally valid* cuts for STSP into $C$.
  4. $C \leftarrow C \cap \{x : ax \leq b, \forall (a, b) \in C\}$.

- Infeasibility propagation in SAT solvers (Achtberg).
- Cuts derived from infeasibility branches (Kilinç et al.).
- Pure branching: Disjunctive Programming (Balas et al.).
- CPLEX 11.0 onwards (sometimes) repeat the branching after a few nodes.

- They all fail to completely capture general partial B&B&C trees.
Some alternative approaches

To solve pla85900, [ABCCEGH 09] one technique was to iterate B&B&C:

Require: C original formulation for STSP

1: loop
2: Iterate B&B&C until tailing off.
3: Collect all *globally valid* cuts for STSP into $C$.
4: $C \leftarrow C \cap \{x : ax \leq b, \forall (a, b) \in C\}$.

- Infeasibility propagation in SAT solvers (Achtberg).
- Cuts derived from infeasibility branches (Kilinç et al.).
- Pure branching: Disjunctive Programming (Balas et al.).
- CPLEX 11.0 onwards (sometimes) repeat the branching after a few nodes.
- They all fail to completely capture general partial B&B&C trees.
Outline

1. What’s wrong with branching?
   - Why we use it, why we don’t like it
   - What has been proposed?

2. A Local Cuts Approach
   - The main idea:

3. The Experiments
   - The instances
   - The results
   - Comparing with split closure
   - Final Remarks
Another point of view
What is a partial B&B&C tree?

- Given an objective function $c$, and a feasible set $P := \{ x \in \mathbb{R}^n : Ax = b, \ l \leq x \leq u, \ x_i \in \mathbb{Z} \forall i \in I \}$.
- If $F \subseteq P$ and $\mathcal{L} = \{ P^i \}_{i=1}^N$ satisfy that $\forall x \in P$ either:
  - $\exists f \in F : cx \leq cf$, or
  - $\exists i \in \{1, \ldots, N\} : x \in P^i$.

Defining $LB_F = \max cf : f \in F$, we have that:

$$LB_f \leq z_P \leq \max_{i=1,\ldots,N} \{ LB_f, \max z_{P^i} \}$$

- In B&B&C, $F$ can be viewed as the set of feasible solutions found so far, and $P^i$ as the LP relaxations on the active leaves of the tree.
- Our goal is to prove $P^i \cap P = \emptyset$ or find $x \in P^i \cap P$ with $cx > LB_f$. 

Daniel Espinoza
From Branching to Cutting (and back again)
The main idea:

Another point of view
What is a partial B&B&C tree?

- Given an objective function $c$, and a feasible set $P := \{ x \in \mathbb{R}^n : Ax = b, \ l \leq x \leq u, \ x_i \in \mathbb{Z} \ \forall i \in I \}$.
- If $F \subseteq P$ and $\mathcal{L} = \{ P^i \}_{i=1}^N$ satisfy that $\forall x \in P$ either:
  1. $\exists f \in F : cx \leq cf$, or
  2. $\exists i \in \{ 1, \ldots, N \} : x \in P^i$.

Defining $LB_F = \max cf : f \in F$, we have that:

$$LB_f \leq z_P \leq \max \{ LB_f, \ \max_{i=1,\ldots,N} z_{P^i} \}$$

- In B&B&C, $F$ can be viewed as the set of feasible solutions found so far, and $P^i$ as the LP relaxations on the active leaves of the tree.
- Our goal is to prove $P^i \cap P = \emptyset$ or find $x \in P^i \cap P$ with $cx > LB_f$. 
The main idea:

Another point of view
What is a partial B&B&C tree?

- Given an objective function \( c \), and a feasible set \( P := \{ x \in \mathbb{R}^n : Ax = b, \ l \leq x \leq u, \ x_i \in \mathbb{Z} \forall i \in I \} \).

- If \( F \subseteq P \) and \( \mathcal{L} = \{ P^i \}_{i=1}^{N} \) satisfy that \( \forall x \in P \) either:
  1. \( \exists f \in F : cx \leq cf \), or
  2. \( \exists i \in \{1, \ldots, N\} : x \in P^i \).

Defining \( LB_F = \max cf : f \in F \), we have that:

\[
LB_f \leq z_P \leq \max \{ LB_f, \ \max_{i=1,\ldots,N} z_{P^i} \}
\]

- In B&B&C, \( F \) can be viewed as the set of feasible solutions found so far, and \( P^i \) as the LP relaxations on the active leaves of the tree.

- Our goal is to prove \( P^i \cap P = \emptyset \) or find \( x \in P^i \cap P \) with \( cx > LB_f \).
Another point of view
What is a partial B&B&C tree?

- Given an objective function $c$, and a feasible set $P := \{x \in \mathbb{R}^n : Ax = b, \ l \leq x \leq u, \ x_i \in \mathbb{Z} \forall i \in I\}$.
- If $F \subseteq P$ and $\mathcal{L} = \{P^i\}_{i=1}^N$ satisfy that $\forall x \in P$ either:
  1. $\exists f \in F : cx \leq cf$, or
  2. $\exists i \in \{1, \ldots, N\} : x \in P^i$.

Defining $\text{LB}_F = \max cf : f \in F$, we have that:

$$\text{LB}_f \leq z_P \leq \max \{\text{LB}_f, \max_{i=1,\ldots,N} z_{P^i}\}$$

- In B&B&C, $F$ can be viewed as the set of feasible solutions found so far, and $P^i$ as the LP relaxations on the active leaves of the tree.
- Our goal is to prove $P^i \cap P = \emptyset$ or find $x \in P^i \cap P$ with $cx > \text{LB}_f$. 
Another point of view
What is a partial B&B&C tree?

- Given an objective function $c$, and a feasible set $P := \{ x \in \mathbb{R}^n : Ax = b, l \leq x \leq u, x_i \in \mathbb{Z} \forall i \in I \}$.
- If $F \subseteq P$ and $\mathcal{L} = \{ P^i \}_{i=1}^N$ satisfy that $\forall x \in P$ either:
  1. $\exists f \in F : cx \leq cf$, or
  2. $\exists i \in \{1, \ldots, N\} : x \in P^i$.

  Defining $LB_F = \max cf : f \in F$, we have that:
  \[
  LB_f \leq z_P \leq \max \{ LB_f, \max_{i=1,\ldots,N} z_{P^i} \}
  \]

- In B&B&C, $F$ can be viewed as the set of feasible solutions found so far, and $P^i$ as the LP relaxations on the active leaves of the tree.
- Our goal is to prove $P^i \cap P = \emptyset$ or find $x \in P^i \cap P$ with $cx > LB_f$. 

Daniel Espinoza From Branching to Cutting (and back again)
Another point of view

What is a partial B&B&C tree?

- Given an objective function $c$, and a feasible set
  $P := \{x \in \mathbb{R}^n : Ax = b, \ l \leq x \leq u, \ x_i \in \mathbb{Z} \forall i \in I\}$.

- If $F \subseteq P$ and $\mathcal{L} = \{P^i\}_{i=1}^N$ satisfy that $\forall x \in P$ either:
  1. $\exists f \in F : cx \leq cf$, or
  2. $\exists i \in \{1, \ldots, N\} : x \in P^i$.

Defining $LB_F = \max cf : f \in F$, we have that:

$$LB_f \leq z_P \leq \max\{LB_f, \ \max_{i=1,\ldots,N} z_{P^i}\}$$

- In B&B&C, $F$ can be viewed as the set of feasible solutions found so far, and $P^i$ as the LP relaxations on the active leaves of the tree.

- Our goal is to prove $P^i \cap P = \emptyset$ or find $x \in P^i \cap P$ with $cx > LB_f$. 
The main idea:

Transforming $\mathcal{L}$ back into cuts

- If we want to *summarize* into cuts
  \[ \overline{\mathcal{L}} := \text{conv}_\text{hull} \left( \bigcup_{i=1, \ldots, N} P_i \right) \]
  is enough to be able to *optimize* over $\overline{\mathcal{L}}$.

- But, since $P_i$ is a polyhedron, and our objective is linear, we have that
  \[ \max cx : x \in \overline{\mathcal{L}} = \max_{i=1, \ldots, N} \max cx : x \in P_i. \]

- The trick is to solve a feasibility LP using all feasible rays/vertices of $\overline{\mathcal{L}}$ by column generation.

- How many optimization calls? \[ \Omega(n) \]

- If we work on a projected space of dimension $k$, \[ \Omega(k) \]

- Related to force some coefficients of the resulting cut to be zero a priori.
The main idea:

Transforming $\mathcal{L}$ back into cuts

- If we want to *summarize* into cuts
  
  $\overline{\mathcal{L}} := \text{conv}_\text{hull} \left( \bigcup_{i=1,...,N} P^i \right)$ is enough to be able to *optimize* over $\overline{\mathcal{L}}$.

- But, since $P^i$ is a polyhedron, and our objective is linear, we have that
  
  $\max cx : x \in \overline{\mathcal{L}} = \max_{i=1,...,N} cx : x \in P^i$.

- The trick is to solve a feasibility LP using all feasible rays/vertices of $\overline{\mathcal{L}}$ by column generation.

  - how many optimization calls? $\Omega(n)$
  - if the working space is projected to some dimension $k$, then $\Omega(k)$
The main idea:

Transforming $\mathcal{L}$ back into cuts

- If we want to *summarize* into cuts
  \[ \overline{\mathcal{L}} := \text{conv}_\text{hull} \left( \bigcup_{i=1}^{N} P^i \right) \]
  is enough to be able to *optimize* over $\overline{\mathcal{L}}$.

- But, since $P^i$ is a polyhedron, and our objective is linear,
  we have that $\max cx : x \in \overline{\mathcal{L}} = \max_{i=1,...,N} cx : x \in P^i$.

- The trick is to solve a feasibility LP using all feasible rays/vertices of $\overline{\mathcal{L}}$ by column generation.
  - how many optimization calls? $\Omega(n)$.
  - If we work on a projected space of dimension $k$? $\Omega(k)$.
  - Related to force some coefficients of the resulting cut to be zero a priori.
The main idea:

**Transforming $\mathcal{L}$ back into cuts**

- If we want to *summarize* into cuts
  $$\overline{\mathcal{L}} := \text{conv}_\text{hull} \left( \bigcup_{i=1,...,N} P^i \right)$$
  is enough to be able to *optimize* over $\overline{\mathcal{L}}$.

- But, since $P^i$ is a polyhedron, and our objective is linear, we have that
  $$\max cx : x \in \overline{\mathcal{L}} = \max_{i=1,...,N} \max cx : x \in P^i.$$

- The trick is to solve a feasibility LP using all feasible rays/vertices of $\overline{\mathcal{L}}$ by column generation.
  - how many optimization calls? $\Omega(n)$.
  - If we work on a projected space of dimension $k$? $\Omega(k)$.
  - Related to force *some* coefficients of the resulting cut to be zero a priori.
The main idea:

Transforming $\mathcal{L}$ back into cuts

- If we want to *summarize* into cuts
  
  $\overline{\mathcal{L}} := \text{conv}_hull \left( \bigcup_{i=1,...,N} P^i \right)$ is enough to be able to *optimize* over $\overline{\mathcal{L}}$.

- But, since $P^i$ is a polyhedron, and our objective is linear, we have that
  
  $\max cx : x \in \overline{\mathcal{L}} = \max_{i=1,...,N} \max cx : x \in P^i$.

- The trick is to solve a feasibility LP using all feasible rays/vertices of $\overline{\mathcal{L}}$ by column generation.
  
  - how many optimization calls? $\Omega(n)$.
  - If we work on a projected space of dimension $k$? $\Omega(k)$.
  - Related to force *some* coefficients of the resulting cut to be zero a priori.
The main idea:

Transforming $\mathcal{L}$ back into cuts

- If we want to summarize into cuts
  \[ \overline{\mathcal{L}} := \text{conv}_{\text{hull}} \left( \bigcup_{i=1,\ldots,N} P^i \right) \]
  is enough to be able to optimize over $\overline{\mathcal{L}}$.

- But, since $P^i$ is a polyhedron, and our objective is linear, we have that
  \[ \max cx : x \in \overline{\mathcal{L}} = \max_{i=1,\ldots,N} cx : x \in P^i. \]

- The trick is to solve a feasibility LP using all feasible rays/vertices of $\overline{\mathcal{L}}$ by column generation.
  - how many optimization calls? $\Omega(n)$.
  - If we work on a projected space of dimension $k$? $\Omega(k)$.
  - Related to force some coefficients of the resulting cut to be zero a priori.
The main idea:

The separation problem:

**Primal Version**

\[
\begin{align*}
\min_{\lambda, \mu} \quad & \sum_{i \in V} \lambda_i x^i + \sum_{i \in R} \mu_i r^i + e \\
\text{s.t.} \quad & \sum_{i \in V} \lambda_i = 1 \\
& \lambda, \mu \geq 0
\end{align*}
\]

\[
(P_{\|err\|})
\]

**Dual Version**

\[
\begin{align*}
\max_{\lambda_o} \quad & \left\| \sum_{i \in V} \lambda_i x^i + \sum_{i \in R} \mu_i r^i - \lambda_o x \right\| \\
\text{s.t.} \quad & \sum_{i \in V} \lambda_i - \lambda_o = 0 \\
& \lambda_o \leq 1, \lambda, \mu \geq 0
\end{align*}
\]

\[
(P_{\|w\|})
\]
The main idea:

The separation problem:

**Primal Version**

\[ \min \quad \|e\| \]
\[ s.t. \quad \sum_{i \in V} \lambda_i x^i + \sum_{i \in R} \mu_i r^i + e = x \]
\[ \sum_{i \in V} \lambda_i = 1 \]
\[ \lambda, \mu \geq 0 \]

**Dual Version**

\[ \max \quad \lambda_o \]
\[ s.t. \quad \left\| \sum_{i \in V} \lambda_i x^i + \sum_{i \in R} \mu_i r^i - \lambda_o x \right\| \leq \delta \]
\[ \sum_{i \in V} \lambda_i - \lambda_o = 0 \]
\[ \lambda_o \leq 1, \lambda, \mu \geq 0 \]
The main idea:

Some notes on separation and *mapping*:

- Which norm in the normalization constraints? $L_1, L_2, L_\infty$?
  - Should the resulting cuts be any different?
  - Should the iterated procedure be any different?
  - Should we only care about the size of the resulting problem?

- We want *sparse* cuts, and few optimization calls... how do we force it?
  - If we force some coefficients to be zero, and to separate, we win!
  - Which variables should we look at? (e.g. branching variables).
  - Best $k$-pseudo costs?
  - How much do we loose by this approximation?
The main idea:

Some notes on separation and mapping:

- Which norm in the normalization constraints? $L_1, L_2, L_\infty$?
  - Should the resulting cuts be any different?
  - Should the iterated procedure be any different?
  - Should we only care about the size of the resulting problem?
- We want sparse cuts, and few optimization calls... how do we force it?
- $k$-pseudo costs?
- How much do we lose by this approximation?
The main idea:

Some notes on separation and *mapping*:

- Which norm in the normalization constraints? $L_1, L_2, L_\infty$?
  - Should the resulting cuts be any different?
  - Should the iterated procedure be any different?
  - Should we only care about the size of the resulting problem?

- We want *sparse* cuts, and few optimization calls... how do we force it?
  - If we force some coefficients to be zero, and to separate:
    - Which variables should we look at? (e.g. branching variables)
    - Should we only focus on these?
    - Can we make this process even more efficient?
    - How much do we lose by this approximation?
The main idea:

Some notes on separation and mapping:

- Which norm in the normalization constraints? $L_1$, $L_2$, $L_\infty$?
  - Should the resulting cuts be any different?
  - Should the iterated procedure be any different?
  - Should we only care about the size of the resulting problem?

- We want sparse cuts, and few optimization calls... how do we force it?
  - If we force some coefficients to be zero, and to separate, we win!

Which variables should we look at (e.g. branching variables)?

- And how do we know if we have enough of them?
The main idea:

Some notes on separation and *mapping*:

- Which norm in the normalization constraints? $L_1, L_2, L_\infty$?
  - Should the resulting cuts be any different?
  - Should the iterated procedure be any different?
  - Should we only care about the size of the resulting problem?

- We want *sparse* cuts, and few optimization calls... how do we force it?
  - If we force some coefficients to be zero, and to separate, we win!
  - Which variables should we look at? (e.g. branching variables).
  - Best $k$-pseudo costs?
  - How much do we loose by this *approximation*?
The main idea:

Some notes on separation and *mapping*:

- Which norm in the normalization constraints? $L_1$, $L_2$, $L_\infty$?  
  - Should the resulting cuts be any different?  
  - Should the iterated procedure be any different?  
  - Should we only care about the size of the resulting problem?

- We want *sparse* cuts, and few optimization calls... how do we force it?  
  - If we force some coefficients to be zero, and to separate, we win!  
  - Which variables should we look at? (e.g. branching variables).  
  - Best $k$-pseudo costs?  
  - How much do we loose by this *approximation*?
Some notes on separation and *mapping*:

- Which norm in the normalization constraints? $L_1$, $L_2$, $L_\infty$? 
  - Should the resulting cuts be any different?
  - Should the iterated procedure be any different?
  - Should we only care about the size of the resulting problem?

- We want *sparse* cuts, and few optimization calls... how do we force it?
  - If we force some coefficients to be zero, and to separate, we win!
  - Which variables should we look at? (e.g. branching variables).
    - Best $k$-pseudo costs?
    - How much do we loose by this *approximation*?
Some notes on separation and *mapping*:

- Which norm in the normalization constraints? $L_1, L_2, L_\infty$?
  - Should the resulting cuts be any different?
  - Should the iterated procedure be any different?
  - Should we only care about the size of the resulting problem?

- We want *sparse* cuts, and few optimization calls... how do we force it?
  - If we force some coefficients to be zero, and to separate, we win!
  - Which variables should we look at? (e.g. branching variables).
  - Best $k$-pseudo costs?
  - How much do we loose by this *approximation*?
The main idea:

Some notes on separation and *mapping*:

- Which norm in the normalization constraints? $L_1$, $L_2$, $L_{\infty}$?
  - Should the resulting cuts be any different?
  - Should the iterated procedure be any different?
  - Should we only care about the size of the resulting problem?
- We want *sparse* cuts, and few optimization calls... how do we force it?
  - If we force some coefficients to be zero, and to separate, we win!
  - Which variables should we look at? (e.g. branching variables).
  - Best $k$-pseudo costs?
  - How much do we loose by this *approximation*?
The main idea:

Some notes on guarantees and on facets

- Can we guarantee that we will be able to cut?
  - If partial B&B improve bound, we should!
  - Assume $z = cx$ is in the problem, use $z$ as one variable (like Gomory).
  - If partial B&B does not improve....
  - Use lexicographic simplex solution to root LP as starting point (like Gomory).

- Should we care about facets of $\overline{L}$?
  - No, since $\overline{L}$ is just an approximation of $\text{conv}_h (P)$.
  - Should we just $\delta$-separate?
The main idea:

Some notes on guarantees and on facets

- Can we guarantee that we will be able to cut?
  - If partial B&B improve bound, we should!
    - Assume $z = cx$ is in the problem, use $z$ as one variable (like Gomory).
    - If partial B&B does not improve....
    - Use lexicographic simplex solution to root LP as starting point (like Gomory).
  - Should we care about facets of $\mathcal{L}$?
    - Since $\mathcal{L}$ is just an approximation of $\text{conv}_h(P)$, should we just $\delta$-separate?
Some notes on guarantees and on facets

- Can we guarantee that we will be able to cut?
  - If partial B&B improve bound, we should!
  - Assume $z = cx$ is in the problem, use $z$ as one variable (like Gomory).
  - If partial B&B does not improve....
  - Use lexicographic simplex solution to root LP as starting point (like Gomory).

- Should we care about facets of $\overline{\mathcal{C}}$?
  - Procedure produces faces of $\overline{\mathcal{L}}$.
  - By tilting we can produce facets of $\overline{\mathcal{L}}$...
  - Should we?
  - Since $\overline{\mathcal{L}}$ is just an approximation of $\text{conv}_h(P)$, should we $\delta$-separate?
Some notes on guarantees and on facets

Can we guarantee that we will be able to cut?
- If partial B&B improve bound, we should!
- Assume $z = cx$ is in the problem, use $z$ as one variable (like Gomory).
- If partial B&B does not improve....
  - Use lexicographic simplex solution to root LP as starting point (like Gomory).

Should we care about facets of $\mathcal{L}$?
- Procedure produces faces of $\mathcal{L}$
- By tilting we can produce facets of $\mathcal{L}$
- Should we?
- Since $\mathcal{L}$ is just an approximation of conv $\operatorname{conv}(P)$, should we just $\delta$-separate?
Some notes on guarantees and on facets

- Can we guarantee that we will be able to cut?
  - If partial B&B improve bound, we should!
  - Assume $z = cx$ is in the problem, use $z$ as one variable (like Gomory).
  - If partial B&B does not improve....
  - Use lexicographic simplex solution to root LP as starting point (like Gomory).

- Should we care about facets of $\mathcal{L}$?
  - Procedure produces faces of $\mathcal{L}$.
  - By tilting we can produce facets of $\mathcal{L}$.
  - We should also consider some combination of doing both or should we just separate...
The main idea:

**Some notes on guarantees and on facets**

- Can we guarantee that we will be able to cut?
  - If partial B&B improve bound, we should!
  - Assume $z = cx$ is in the problem, use $z$ as one variable (like Gomory).
  - If partial B&B does not improve....
  - Use lexicographic simplex solution to root LP as starting point (like Gomory).

- Should we care about facets of $\overline{L}$?
  - Procedure produces *faces* of $\overline{L}$.
  - By tilting we can produce *facets* of $\overline{L}$...
  - Should we?
  - Since $\overline{L}$ is just an approximation of $\text{conv}_hull(P)$, should we just $\delta$-separate?
The main idea:

Some notes on guarantees and on facets

- Can we guarantee that we will be able to cut?
  - If partial B&B improve bound, we should!
  - Assume $z = cx$ is in the problem, use $z$ as one variable (like Gomory).
  - If partial B&B does not improve....
  - Use lexicographic simplex solution to root LP as starting point (like Gomory).

- Should we care about facets of $\overline{L}$?
  - Procedure produces *faces* of $\overline{L}$.
    - By tilting we can produce *facets* of $\overline{L}$...
    - Should we?
    - Since $\overline{L}$ is just an approximation of $\text{conv}_\text{hull}(P)$, should we just $\delta$-separate?
The main idea:

Some notes on guarantees and on facets

- Can we guarantee that we will be able to cut?
  - If partial B&B improve bound, we should!
  - Assume $z = cx$ is in the problem, use $z$ as one variable (like Gomory).
  - If partial B&B does not improve....
  - Use lexicographic simplex solution to root LP as starting point (like Gomory).

- Should we care about facets of $\overline{L}$?
  - Procedure produces *faces* of $\overline{L}$.
  - By tilting we can produce *facets* of $\overline{L}$...
  - Should we?
  - Since $\overline{L}$ is just an approximation of $\text{conv}_hull(P)$, should we just $\delta$-separate?
Some notes on guarantees and on facets

- Can we guarantee that we will be able to cut?
  - If partial B&B improve bound, we should!
  - Assume $z = cx$ is in the problem, use $z$ as one variable (like Gomory).
  - If partial B&B does not improve....
  - Use lexicographic simplex solution to root LP as starting point (like Gomory).

- Should we care about facets of $\overline{L}$?
  - Procedure produces *faces* of $\overline{L}$.
  - By tilting we can produce *facets* of $\overline{L}$...
  - Should we?
    - Since $\overline{L}$ is just an approximation of $\text{conv}_\text{hull}(P)$, should we just $\delta$-separate?
Some notes on guarantees and on facets

- Can we guarantee that we will be able to cut?
  - If partial B&B improve bound, we should!
  - Assume $z = cx$ is in the problem, use $z$ as one variable (like Gomory).
  - If partial B&B does not improve....
  - Use lexicographic simplex solution to root LP as starting point (like Gomory).

- Should we care about facets of $\overline{L}$?
  - Procedure produces *faces* of $\overline{L}$.
  - By tilting we can produce *facets* of $\overline{L}$...
  - Should we?
  - Since $\overline{L}$ is just an approximation of $\text{conv}_hull(P)$, should we just $\delta$-separate?
The main idea:

What is our goal?
Finding out whether this procedure can work

The procedure:

1. Solve root LP for $P$, and let $x^*$ be its solution.
2. Generate a limited branch and bound tree (how?).
3. Select a subset of variables $V \subseteq \{1, \ldots, n\}$ and work on $P' = \{x \in \mathbb{R}^V : \exists y \in \mathbb{R}^{V^c}, (x, y) \in P\}$.
4. Select a separation formulation, tolerances, and Separate $x^*_P$ from $P'$.
5. If found a cut, add it to $P$, let $x^*$ be its solution, and go to 4.
6. If no improvements, stop! otherwise, go to 2

Note that we have A LOT of parameters.
Want a proof of concept.

Daniel Espinoza
From Branching to Cutting (and back again)
The main idea:

What is our goal?
Finding out whether this procedure can work

The procedure:

1. Solve root LP for $P$, and let $x^*$ be its solution.
2. Generate a limited branch and bound tree (how?).
3. Select a subset of variables $V \subseteq \{1, \ldots, n\}$ and work on $P' = \{x \in \mathbb{R}^V : \exists y \in \mathbb{R}^{V^c}, (x, y) \in P\}$.
4. Select a separation formulation, tolerances, and Separate $x^*_P$ from $P'$.
5. If found a cut, add it to $P$, let $x^*$ be its solution, and go to 4.
6. If no improvements, stop! otherwise, go to 2.

Note that we have A LOT of parameters.

Want a proof of concept.
The main idea:

What is our goal?
Finding out whether this procedure can work

The procedure:

1. Solve root LP for $P$, and let $x^*$ be its solution.
2. Generate a limited branch and bound tree (how?).
3. Select a subset of variables $V \subseteq \{1, \ldots, n\}$ and work on $P' = \{x \in \mathbb{R}^V : \exists y \in \mathbb{R}^{V^c}, (x, y) \in P\}$.
4. Select a separation formulation, tolerances, and Separate $x^*_P$ from $P'$.
5. If found a cut, add it to $P$, let $x^*$ be its solution, and go to 4.
6. If no improvements, stop! otherwise, go to 2

Note that we have A LOT of parameters.
Want a proof of concept.
What’s wrong with branching?

A Local Cuts Approach

The Experiments

The main idea:

What is our goal?
Finding out whether this procedure can work

The procedure:

1. Solve root LP for $P$, and let $x^*$ be its solution.
2. Generate a limited branch and bound tree (how?).
3. Select a subset of variables $V \subseteq \{1, \ldots, n\}$ and work on $P' = \{x \in \mathbb{R}^V : \exists y \in \mathbb{R}^{V^c}, (x, y) \in P\}$.
4. Select a separation formulation, tolerances, and Separate $x^*_P$ from $P'$.
5. If found a cut, add it to $P$, let $x^*$ be its solution, and go to 4.
6. If no improvements, stop! otherwise, go to 2

Note that we have A LOT of parameters.

Want a proof of concept.
The main idea:

**What is our goal?**
Finding out whether this procedure can work

The procedure:

1. Solve root LP for $P$, and let $x^*$ be its solution.
2. Generate a limited branch and bound tree (how?).
3. Select a subset of variables $V \subseteq \{1, \ldots, n\}$ and work on $P' = \{x \in \mathbb{R}^V : \exists y \in \mathbb{R}^{V^c}, (x, y) \in P\}$.
4. Select a separation formulation, tolerances, and Separate $x^*_P$ from $P'$.
5. If found a cut, add it to $P$, let $x^*$ be its solution, and go to 4.
6. If no improvements, stop! otherwise, go to 2

- Note that we have A LOT of parameters.
- Want a proof of concept.
The main idea:

What is our goal?
Finding out whether this procedure can work

The procedure:

1. Solve root LP for $P$, and let $x^*$ be its solution.
2. Generate a limited branch and bound tree (how?).
3. Select a subset of variables $V \subseteq \{1, \ldots, n\}$ and work on $P' = \{x \in \mathbb{R}^V : \exists y \in \mathbb{R}^{V^c}, (x, y) \in P\}$.
4. Select a separation formulation, tolerances, and Separate $x_p^*$ from $P'$.
5. If found a cut, add it to $P$, let $x^*$ be its solution, and go to 4.
6. If no improvements, stop! otherwise, go to 2.

Note that we have A LOT of parameters.
Want a proof of concept.
The main idea:

What is our goal?
Finding out whether this procedure can work

The procedure:

1. Solve root LP for $P$, and let $x^*$ be its solution.
2. Generate a limited branch and bound tree (how?).
3. Select a subset of variables $V \subseteq \{1, \ldots, n\}$ and work on $P' = \{x \in \mathbb{R}^V : \exists y \in \mathbb{R}^{V^c}, (x, y) \in P\}$.
4. Select a separation formulation, tolerances, and Separate $x^*_P$ from $P'$.
5. If found a cut, add it to $P$, let $x^*$ be its solution, and go to 4.
6. If no improvements, stop! otherwise, go to 2

Note that we have A LOT of parameters.

Want a proof of concept.
What’s wrong with branching?

A Local Cuts Approach

The Experiments

The main idea:

What is our goal?
Finding out whether this procedure can work

The procedure:

1. Solve root LP for $P$, and let $x^*$ be its solution.
2. Generate a limited branch and bound tree (how?).
3. Select a subset of variables $V \subseteq \{1, \ldots, n\}$ and work on $P' = \{x \in \mathbb{R}^V : \exists y \in \mathbb{R}^V, (x, y) \in P\}$.
4. Select a separation formulation, tolerances, and Separate $x^*_P$ from $P'$.
5. If found a cut, add it to $P$, let $x^*$ be its solution, and go to 4.
6. If no improvements, stop! otherwise, go to 2

- Note that we have A LOT of parameters.
- Want a proof of concept.

Daniel Espinoza  From Branching to Cutting (and back again)
The main idea:

What is our goal?
Finding out whether this procedure can work

The procedure:

1. Solve root LP for $P$, and let $x^*$ be its solution.
2. Generate a limited branch and bound tree (how?).
3. Select a subset of variables $V \subseteq \{1, \ldots, n\}$ and work on $P' = \{x \in \mathbb{R}^V : \exists y \in \mathbb{R}^{V^c}, (x, y) \in P\}$.
4. Select a separation formulation, tolerances, and Separate $x_P^*$ from $P'$.
5. If found a cut, add it to $P$, let $x^*$ be its solution, and go to 4.
6. If no improvements, stop! otherwise, go to 2

- Note that we have A LOT of parameters.
- Want a proof of concept.
1. What's wrong with branching?
   - Why we use it, why we don't like it
   - What has been proposed?

2. A Local Cuts Approach
   - The main idea:

3. The Experiments
   - The instances
   - The results
   - Comparing with split closure
   - Final Remarks
Where do we test?

- Anuret’s collection.
- Atamtürk collection.
- Cor@al, Deis, MIPLIB3, MIPLIB 2003, Mittleman, others.
- Total 1,539 problems.
- We do expect the scheme to be expensive.
- Thus we should not look into easy problems...
- Filter out those problems that CPLEX 11.2 can solve within 20 minutes on an Intel Xeon E5345.
- Only 303 problems remain.
The instances

Where do we test?

- Anuret’s collection.
- Atamtürk collection.
- Cor@al, Deis, MIPLIB3, MIPLIB 2003, Mittleman, others.
- Total 1,539 problems....
- We do expect the scheme to be expensive
- Thus we should not look into easy problems...
- Filter out those problems that CPLEX 11.2 can solve within 20 minutes on an Intel Xeon E5345.
- Only 303 problems remain.
Where do we test?

- Anuret’s collection.
- Atamtürk collection.
- Cor@al, Deis, MIPLIB3, MIPLIB 2003, Mittleman, others.

Total 1,539 problems....

We do expect the scheme to be expensive

Thus we should not look into easy problems...

Filter out those problems that CPLEX 11.2 can solve within 20 minutes on an Intel Xeon E5345.

Only 303 problems remain.
The instances

Where do we test?

- Anuret’s collection.
- Atamtürk collection.
- Cor@al, Deis, MIPLIB3, MIPLIB 2003, Mittleman, others.
- Total 1,539 problems....
  - We do expect the scheme to be expensive
  - Thus we should not look into easy problems...
  - Filter out those problems that CPLEX 11.2 can solve within 20 minutes on an Intel Xeon E5345.
  - Only 303 problems remain.
Where do we test?

- Anuret’s collection.
- Atamtürk collection.
- Cor@al, Deis, MIPLIB3, MIPLIB 2003, Mittleman, others.
- Total 1,539 problems....
- We do expect the scheme to be expensive
  - Thus we should not look into easy problems...
  - Filter out those problems that CPLEX 11.2 can solve within 20 minutes on an Intel Xeon E5345.
  - Only 303 problems remain.
Where do we test?

- Anuret’s collection.
- Atamtürk collection.
- Cor@al, Deis, MIPLIB3, MIPLIB 2003, Mittleman, others.
- Total 1,539 problems....
- We do expect the scheme to be expensive
- Thus we should not look into easy problems...
  - Filter out those problems that CPLEX 11.2 can solve within 20 minutes on an Intel Xeon E5345.
  - Only 303 problems remain.
Where do we test?

- Anuret’s collection.
- Atamtürk collection.
- Cor@al, Deis, MIPLIB3, MIPLIB 2003, Mittleman, others.
- Total 1,539 problems....
- We do expect the scheme to be expensive
- Thus we should not look into easy problems...
- Filter out those problems that CPLEX 11.2 can solve within 20 minutes on an Intel Xeon E5345.

- Only 303 problems remain.
The instances

Where do we test?

- Anuret’s collection.
- Atamtürk collection.
- Cor@al, Deis, MIPLIB3, MIPLIB 2003, Mittleman, others.
- Total 1,539 problems....
- We do expect the scheme to be expensive
- Thus we should not look into easy problems...
- Filter out those problems that CPLEX 11.2 can solve within 20 minutes on an Intel Xeon E5345.
- Only 303 problems remain.
Strictly speaking, we propose a cutting machinery.

... but we are branching.

... in the end we want to solve!

Use as base comparisons both:

1. Cplex default branch and bound and our branching
   machinery.
2. Cplex aggressive cut and preprocessed at the root node.

Different settings on mapping:

1. Simple mapping
2. Branching variables mapping

Different branching schemes to construct partial B&B&C tree:

1. Defaults, but no cutting (1)
2. Defaults (2)
3. Best bound and Strong branching (3)
Strictly speaking, we propose a cutting machinery. 
... but we are branching.
... in the end we want to solve!
Use as base comparisons both:
- Cplex default branch and bound and cut (without heuristics).
- Cplex aggressive cut and preprocessed at the root node.
Different settings on mapping:
- All variables (200).
- Branching variables (100).
Different branching schemes to construct partial B&B&C tree:
- Defaults, but no cutting (1).
- Defaults (2).
- Best bound and Strong branching (3).
Strictly speaking, we propose a cutting machinery.
... but we are branching.
... in the end we want to solve!

Use as base comparisons both:
- Cplex default branch and bound and cut (without heuristics).
- Cplex aggressive cut and preprocessed at the root node.

Different settings on mapping:
- All variables (200).
- Branching variables (100).

Different branching schemes to construct partial B&B&C tree:
- Defaults, but no cutting (1).
- Defaults (2).
- Best bound and Strong branching (3).
Strictly speaking, we propose a cutting machinery.
... but we are branching.
... in the end we want to solve!
Use as base comparisons both:

1. Cplex default branch and bound and cut (without heuristics).
2. Cplex aggressive cut and preprocessed at the root node.
Different settings on mapping:

Different branching schemes to construct partial B&B&C tree:

- Defaults, but no cutting (1).
- Defaults (2).
- Best bound and Strong branching (3).
Against what do we compare?

- Strictly speaking, we propose a cutting machinery.
- ... but we are branching.
- ... in the end we want to solve!
- Use as base comparisons both:
  1. Cplex default branch and bound and cut (without heuristics).
  2. Cplex aggressive cut and preprocessed at the root node.
- Different settings on mapping:
  1. All variables (100).
  2. Branching variables (100).
- Different branching schemes to construct partial B&B&C tree:
  1. Defaults but no cutting (1).
  2. Defaults (2).
  3. Best bound and Strong branching (3).
Strictly speaking, we propose a cutting machinery.

... but we are branching.

... in the end we want to solve!

Use as base comparisons both:

1. Cplex default branch and bound and cut (without heuristics).
2. Cplex aggressive cut and preprocessed at the root node.

Different settings on mapping:

- All variables (200).
- Branching variables (100).

Different branching schemes to construct partial B&B&C tree:

- Defaults, but no cutting (1).
- Defaults (2).
- Best bound and Strong branching (3).
Strictly speaking, we propose a cutting machinery.
... but we are branching.
... in the end we want to solve!
Use as base comparisons both:
1. Cplex default branch and bound and cut (without heuristics).
2. Cplex aggressive cut and preprocessed at the root node.
Different settings on *mapping*:
- All variables (200).
- Branching variables (100).
Different branching schemes to construct partial B&B&C tree:
- Defaults, but no cutting (1).
- Defaults (2).
- Best bound and Strong branching (3).
Strictly speaking, we propose a cutting machinery.
... but we are branching.
... in the end we want to solve!
Use as base comparisons both:
  1 Cplex default branch and bound and cut (without heuristics).
  2 Cplex aggressive cut and preprocessed at the root node.
Different settings on mapping:
  All variables (200).
  Branching variables (100).
Different branching schemes to construct partial B&B&C tree:
What’s wrong with branching?

A Local Cuts Approach

The Experiments

The instances

Against what do we compare?

- Strictly speaking, we propose a cutting machinery.
- ... but we are branching.
- ... in the end we want to solve!
- Use as base comparisons both:
  1. Cplex default branch and bound and cut (without heuristics).
  2. Cplex aggressive cut and preprocessed at the root node.
- Different settings on *mapping*:
  - All variables (200).
  - Branching variables (100).
- Different branching schemes to construct partial B&B&C tree:
  - Defaults, but no cutting (1).
  - Defaults (2).
  - Best bound and Strong branching (3).
Strictly speaking, we propose a cutting machinery.

... but we are branching.

... in the end we want to solve!

Use as base comparisons both:

1. Cplex default branch and bound and cut (without heuristics).
2. Cplex aggressive cut and preprocessed at the root node.

Different settings on *mapping*:

- All variables (200).
- Branching variables (100).

Different branching schemes to construct partial B&B&C tree:

- Defaults, but no cutting (1).
- Defaults (2).
- Best bound and Strong branching (3).
Against what do we compare?

- Strictly speaking, we propose a cutting machinery.
- ... but we are branching.
- ... in the end we want to solve!
- Use as base comparisons both:
  1. Cplex default branch and bound and cut (without heuristics).
  2. Cplex aggressive cut and preprocessed at the root node.
- Different settings on *mapping*:
  - All variables (200).
  - Branching variables (100).
- Different branching schemes to construct partial B&B&C tree:
  - Defaults, but no cutting (1).
  - Defaults (2).
  - Best bound and Strong branching (3).
Strictly speaking, we propose a cutting machinery.
... but we are branching.
... in the end we want to solve!
Use as base comparisons both:
1. Cplex default branch and bound and cut (without heuristics).
2. Cplex aggressive cut and preprocessed at the root node.
Different settings on mapping:
   - All variables (200).
   - Branching variables (100).
Different branching schemes to construct partial B&B&C tree:
   - Defaults, but no cutting (1).
   - Defaults (2).
   - Best bound and Strong branching (3).
Strictly speaking, we propose a cutting machinery.
... but we are branching.
... in the end we want to solve!
Use as base comparisons both:
1. Cplex default branch and bound and cut (without heuristics).
2. Cplex aggressive cut and preprocessed at the root node.
Different settings on mapping:
- All variables (200).
- Branching variables (100).
Different branching schemes to construct partial B&B&C tree:
- Defaults, but no cutting (1).
- Defaults (2).
- Best bound and Strong branching (3).
Against what do we compare?

- Different separation problem formulations:
  - Minimize $L_1$ norm of error (1000).
  - Minimize $L_1$ norm of resulting inequality (2000).
  - Minimize $L_2$ norm of resulting inequality (3000, 4000).
  - Minimize $L_2$ norm of error (5000).
- We DO NOT test lexicographic solutions to root node.
- We DO NOT test the effect of complete separation against approximate separation.
- We DO NOT test tilting/facet procedures, but use resulting face from the separation LP.
Different separation problem formulations:

- Minimize $L_1$ norm of error (1000).
- Minimize $L_1$ norm of resulting inequality (2000).
- Minimize $L_2$ norm of resulting inequality (3000, 4000).
- Minimize $L_2$ norm of error (5000).

- We DO NOT test lexicographic solutions to root node.
- We DO NOT test the effect of complete separation against approximate separation.
- We DO NOT test tilting/facet procedures, but use resulting face from the separation LP.
Against what do we compare?

- Different separation problem formulations:
  - Minimize $L_1$ norm of error (1000).
  - Minimize $L_1$ norm of resulting inequality (2000).
  - Minimize $L_2$ norm of resulting inequality (3000, 4000).
  - Minimize $L_2$ norm of error (5000).

- We DO NOT test lexicographic solutions to root node.
- We DO NOT test the effect of complete separation against approximate separation.
- We DO NOT test tilting/facet procedures, but use resulting face from the separation LP.
Against what do we compare?

- Different separation problem formulations:
  - Minimize $L_1$ norm of error (1000).
  - Minimize $L_1$ norm of resulting inequality (2000).
  - Minimize $L_2$ norm of resulting inequality (3000,4000).
  - Minimize $L_2$ norm of error (5000).

- We DO NOT test lexicographic solutions to root node.
- We DO NOT test the effect of complete separation against approximate separation.
- We DO NOT test tilting/facet procedures, but use resulting face from the separation LP.
Different separation problem formulations:
  - Minimize $L_1$ norm of error (1000).
  - Minimize $L_1$ norm of resulting inequality (2000).
  - Minimize $L_2$ norm of resulting inequality (3000,4000).
  - Minimize $L_2$ norm of error (5000).

We DO NOT test lexicographic solutions to root node.
We DO NOT test the effect of complete separation against approximate separation.
We DO NOT test tilting/facet procedures, but use resulting face from the separation LP.
Against what do we compare?

- Different separation problem formulations:
  - Minimize $L_1$ norm of error (1000).
  - Minimize $L_1$ norm of resulting inequality (2000).
  - Minimize $L_2$ norm of resulting inequality (3000,4000).
  - Minimize $L_2$ norm of error (5000).

- We DO NOT test lexicographic solutions to root node.

- We DO NOT test the effect of complete separation against approximate separation.

- We DO NOT test tilting/facet procedures, but use resulting face from the separation LP.
Against what do we compare?

- Different separation problem formulations:
  - Minimize $L_1$ norm of error (1000).
  - Minimize $L_1$ norm of resulting inequality (2000).
  - Minimize $L_2$ norm of resulting inequality (3000,4000).
  - Minimize $L_2$ norm of error (5000).

- We DO NOT test lexicographic solutions to root node.

- We DO NOT test the effect of complete separation against approximate separation.

- We DO NOT test tilting/facet procedures, but use resulting face from the separation LP.
Against what do we compare?

- Different separation problem formulations:
  - Minimize $L_1$ norm of error (1000).
  - Minimize $L_1$ norm of resulting inequality (2000).
  - Minimize $L_2$ norm of resulting inequality (3000, 4000).
  - Minimize $L_2$ norm of error (5000).

- We DO NOT test lexicographic solutions to root node.

- We DO NOT test the effect of complete separation against approximate separation.

- We DO NOT test tilting/facet procedures, but use resulting face from the separation LP.
What are the quality measures?

- **Number of cuts added by the scheme.**
- % of best bound achieved by a particular configuration
  - Defined as
    \[
    100 \cdot \left( 1 - \frac{Z_{LP} - LB}{UB^2} \right)
    \]
  - Where $LB$ is the worst bound seen across all the configurations.
  - $UB^2 = \max(|UB|, UB - LB, 0.1)$.
  - UB best known solution or optimal value.
  - Overall indicator taken as arithmetic average.
- Number of times slower than the fastest configuration.
  - Overall indicator taken as geometric average.
- Limit running time to 7,200 seconds.
- Total 9,696 runs, about 800 days computing time.
- Use performance profiles.
What are the quality measures?

- Number of cuts added by the scheme.
- % of best bound achieved by a particular configuration

  Defined as

  \[ 100 \cdot \left( 1 - \frac{Z_{LP} - LB}{UB^2} \right) \]

  Where \( LB \) is the worst bound seen across all the configurations.

  \( UB^2 = \max\{||UB||, UB - LB, 0.1\} \).

  \( UB \) best known solution or optimal value.

  Overall indicator taken as arithmetic average.

- Number of times slower than the fastest configuration.

  Overall indicator taken as geometric average.

- Limit running time to 7,200 seconds.

- Total 9,696 runs, about 800 days computing time.

- Use performance profiles.
What are the quality measures?

- Number of cuts added by the scheme.
- % of best bound achieved by a particular configuration
  - Defined as
    \[ 100 \cdot \left(1 - \frac{Z_{LP} - LB}{UB^2}\right) \]
    - Where \( LB \) is the worst bound seen across all the configurations.
    - \( UB^2 = \max\{|UB|, UB - LB, 0.1\} \).
    - \( UB \) best known solution or optimal value.
    - Overall indicator taken as arithmetic average.
- Number of times slower than the fastest configuration.
  - Overall indicator taken as geometric average
- Limit running time to 7,200 seconds.
- Total 9,696 runs, about 800 days computing time.
- Use performance profiles.
What are the quality measures?

- Number of cuts added by the scheme.
- % of best bound achieved by a particular configuration
  - Defined as
    \[ 100 \cdot \left(1 - \frac{Z_{LP} - LB}{UB2}\right) \]
  - Where \( LB \) is the worst bound seen across all the configurations.
  - \( UB2 = \max\{|UB|, UB - LB, 0.1\} \).
  - \( UB \) best known solution or optimal value.
  - Overall indicator taken as arithmetic average.
- Number of times slower than the fastest configuration.
  - Overall indicator taken as geometric average.
- Limit running time to 7,200 seconds.
- Total 9,696 runs, about 800 days computing time.
- Use performance profiles.
What are the quality measures?

- Number of cuts added by the scheme.
- % of best bound achieved by a particular configuration
  - Defined as
    \[
    100 \cdot \left( 1 - \frac{Z_{LP} - LB}{UB2} \right)
    \]
  - Where \( LB \) is the worst bound seen across all the configurations.
  - \( UB2 = \max\{|UB|, UB - LB, 0.1\} \).
- Overall indicator taken as arithmetic average.
- Number of times slower than the fastest configuration.
  - Overall indicator taken as geometric average.
- Limit running time to 7,200 seconds.
- Total 9,696 runs, about 800 days computing time.
- Use performance profiles.
What are the quality measures?

- Number of cuts added by the scheme.
- % of best bound achieved by a particular configuration
  - Defined as
    \[ 100 \cdot \left( 1 - \frac{Z_{LP} - LB}{UB2} \right) \]
  - Where \( LB \) is the worst bound seen across all the configurations.
  - \( UB2 = \max\{|UB|, UB - LB, 0.1\} \).
  - \( UB \) best known solution or optimal value.
- Overall indicator taken as arithmetic average.
- Number of times slower than the fastest configuration.
- Overall indicator taken as geometric average.
- Total running time is 7,200 seconds.
- Overall best running time is about 800 days.
- Use performance profiles.
What's wrong with branching?

A Local Cuts Approach

The Experiments

The instances

What are the quality measures?

- Number of cuts added by the scheme.
- % of best bound achieved by a particular configuration
  - Defined as
    \[ 100 \cdot \left( 1 - \frac{Z_{LP} - LB}{UB^2} \right) \]
  - Where \( LB \) is the worst bound seen across all the configurations.
  - \( UB^2 = \max\{|UB|, UB - LB, 0.1\} \).
  - \( UB \) best known solution or optimal value.
  - Overall indicator taken as arithmetic average.

- Number of times slower than the fastest configuration.
  - Overall indicator taken as geometric average.

- Limit running time to 7,200 seconds.
- Total 9,696 runs, about 800 days computing time.
- Use performance profiles.
What's wrong with branching?

A Local Cuts Approach

The Experiments

The instances

What are the quality measures?

- Number of cuts added by the scheme.
- % of best bound achieved by a particular configuration
  - Defined as
    \[
    100 \cdot \left(1 - \frac{Z_{LP} - LB}{UB2}\right)
    \]
  - Where \( LB \) is the worst bound seen across all the configurations.
  - \( UB2 = \max\{|UB|, UB - LB, 0.1\} \).
  - \( UB \) best known solution or optimal value.
  - Overall indicator taken as arithmetic average.

- Number of times slower than the fastest configuration.
  - Overall indicator taken as geometric average.
  - Limit running time to 7,200 seconds.
  - Total 9,696 runs, about 800 days computing time.
  - Use performance profiles.
The instances

What are the quality measures?

- Number of cuts added by the scheme.
- % of best bound achieved by a particular configuration
  - Defined as
    \[
    100 \cdot \left( 1 - \frac{Z_{LP} - LB}{UB^2} \right)
    \]
  - Where \( LB \) is the worst bound seen across all the configurations.
  - \( UB^2 = \max\{ |UB|, UB - LB, 0.1 \} \).
  - \( UB \) best known solution or optimal value.
  - Overall indicator taken as arithmetic average.
- Number of times slower than the fastest configuration.
  - Overall indicator taken as geometric average.
- Limit running time to 7,200 seconds.
- Total 9,696 runs, about 800 days computing time.
- Use performance profiles.
The instances

What are the quality measures?

- Number of cuts added by the scheme.
- % of best bound achieved by a particular configuration
  - Defined as
    \[
    100 \cdot \left(1 - \frac{Z_{LP} - LB}{UB^2}\right)
    \]
  - Where \( LB \) is the worst bound seen across all the configurations.
  - \( UB^2 = \max\{|UB|, UB - LB, 0.1\} \).
  - \( UB \) best known solution or optimal value.
  - Overall indicator taken as arithmetic average.
- Number of times slower than the fastest configuration.
  - Overall indicator taken as geometric average.
  - Limit running time to 7,200 seconds.
- Total 9,696 runs, about 800 days computing time.
- Use performance profiles.

Daniel Espinoza

From Branching to Cutting (and back again)
What are the quality measures?

- Number of cuts added by the scheme.
- % of best bound achieved by a particular configuration
  - Defined as
    \[
    100 \cdot \left(1 - \frac{Z_{LP} - LB}{UB2}\right)
    \]
  - Where \( LB \) is the worst bound seen across all the configurations.
  - \( UB2 = \max\{|UB|, UB - LB, 0.1\} \).
  - \( UB \) best known solution or optimal value.
  - Overall indicator taken as arithmetic average.

- Number of times slower than the fastest configuration.
  - Overall indicator taken as geometric average.
  - Limit running time to 7,200 seconds.
  - Total 9,696 runs, about 800 days computing time.

Use performance profiles.
What are the quality measures?

- Number of cuts added by the scheme.
- % of best bound achieved by a particular configuration
  - Defined as
    \[ 100 \cdot \left( 1 - \frac{Z_{LP} - LB}{UB2} \right) \]
  - Where \( LB \) is the worst bound seen across all the configurations.
  - \( UB2 = \max\{|UB|, UB - LB, 0.1\} \).
  - \( UB \) best known solution or optimal value.
  - Overall indicator taken as arithmetic average.
- Number of times slower than the fastest configuration.
  - Overall indicator taken as geometric average.
  - Limit running time to 7,200 seconds.
  - Total 9,696 runs, about 800 days computing time.
- Use performance profiles.
### The overall table:

<table>
<thead>
<tr>
<th></th>
<th>Conf</th>
<th>Ninst</th>
<th>Operf</th>
<th>Tperf</th>
<th></th>
<th>Conf</th>
<th>Ninst</th>
<th>Operf</th>
<th>Tperf</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPX-root</td>
<td>1101</td>
<td>199</td>
<td>61.68</td>
<td>99.21</td>
<td>CPX-BBC</td>
<td>1102</td>
<td>198</td>
<td>81.07</td>
<td>252.87</td>
</tr>
<tr>
<td></td>
<td>1103</td>
<td>190</td>
<td>88.45</td>
<td>1310.11</td>
<td></td>
<td>1201</td>
<td>145</td>
<td>64.55</td>
<td>327.21</td>
</tr>
<tr>
<td></td>
<td>1202</td>
<td>146</td>
<td>76.77</td>
<td>329.42</td>
<td></td>
<td>1203</td>
<td>148</td>
<td>84.37</td>
<td>891.13</td>
</tr>
<tr>
<td></td>
<td>2101</td>
<td>198</td>
<td>62.13</td>
<td>107.89</td>
<td></td>
<td>2102</td>
<td>197</td>
<td>83.79</td>
<td>274.03</td>
</tr>
<tr>
<td></td>
<td>2103</td>
<td>192</td>
<td>89.39</td>
<td>1472.81</td>
<td></td>
<td>2201</td>
<td>199</td>
<td>62.70</td>
<td>609.38</td>
</tr>
<tr>
<td></td>
<td>2202</td>
<td>192</td>
<td>84.30</td>
<td>918.22</td>
<td></td>
<td>2203</td>
<td>189</td>
<td>88.34</td>
<td>1563.92</td>
</tr>
<tr>
<td></td>
<td>3101</td>
<td>197</td>
<td>62.22</td>
<td>110.85</td>
<td></td>
<td>3102</td>
<td>197</td>
<td>83.90</td>
<td>290.23</td>
</tr>
<tr>
<td></td>
<td>3103</td>
<td>192</td>
<td>89.35</td>
<td>1301.44</td>
<td></td>
<td>3201</td>
<td>195</td>
<td>62.74</td>
<td>518.09</td>
</tr>
<tr>
<td></td>
<td>3202</td>
<td>193</td>
<td>81.43</td>
<td>672.53</td>
<td></td>
<td>3203</td>
<td>192</td>
<td>87.60</td>
<td>1390.96</td>
</tr>
<tr>
<td></td>
<td>4101</td>
<td>199</td>
<td>61.54</td>
<td>2.06</td>
<td></td>
<td>4102</td>
<td>189</td>
<td>81.68</td>
<td>9.72</td>
</tr>
<tr>
<td></td>
<td>4103</td>
<td>187</td>
<td>87.92</td>
<td>277.32</td>
<td></td>
<td>4201</td>
<td>196</td>
<td>61.68</td>
<td>368.25</td>
</tr>
<tr>
<td></td>
<td>4202</td>
<td>192</td>
<td>79.28</td>
<td>510.20</td>
<td></td>
<td>4203</td>
<td>182</td>
<td>87.61</td>
<td>1205.63</td>
</tr>
<tr>
<td></td>
<td>5101</td>
<td>199</td>
<td>61.67</td>
<td>2.33</td>
<td></td>
<td>5102</td>
<td>191</td>
<td>81.72</td>
<td>10.21</td>
</tr>
<tr>
<td></td>
<td>5103</td>
<td>186</td>
<td><strong>87.97</strong></td>
<td><strong>302.47</strong></td>
<td></td>
<td>5201</td>
<td>187</td>
<td>60.60</td>
<td>396.51</td>
</tr>
<tr>
<td></td>
<td>5202</td>
<td>183</td>
<td>78.20</td>
<td>530.16</td>
<td></td>
<td>5203</td>
<td>188</td>
<td>86.71</td>
<td>1223.96</td>
</tr>
</tbody>
</table>

- Some instances fail due to memory/time limit (and are discarded).
What's wrong with branching?
A Local Cuts Approach
The Experiments

The results

Time/Quality trade-off:

![Graph showing time vs. quality trade-off with points labeled CPX-root, CPX-B&B, 2101, 2102, 2103, 3201, 3202, 3203, 5103, 5201, 5202, 5203, 1101, 1102, 1103, 1201, 1202, 1203.](image_url)
The results

2103 Performance profile
The results

5103 Performance profile

![Graph showing performance profile comparison between CPX-bbc, CPX-root, 5103, and 5103 cuts across different instances and performance percentages.]

Daniel Espinoza  From Branching to Cutting (and back again)
## Comparing with split closure

### The overall table:

<table>
<thead>
<tr>
<th>Conf</th>
<th>Ninst</th>
<th>Operf</th>
<th>Tperf</th>
</tr>
</thead>
<tbody>
<tr>
<td>5120</td>
<td>22</td>
<td>89.5878</td>
<td>2.0563</td>
</tr>
<tr>
<td>3120</td>
<td>22</td>
<td>89.1957</td>
<td>0.9993</td>
</tr>
<tr>
<td>CPX-bbc</td>
<td>22</td>
<td>87.1556</td>
<td>1.0599</td>
</tr>
<tr>
<td>SC</td>
<td>22</td>
<td>82.6373</td>
<td>1.0599</td>
</tr>
<tr>
<td>1120</td>
<td>22</td>
<td>89.4441</td>
<td>1.5949</td>
</tr>
<tr>
<td>4120</td>
<td>22</td>
<td>89.5552</td>
<td>1.7320</td>
</tr>
<tr>
<td>2120</td>
<td>22</td>
<td>89.5599</td>
<td>1.2355</td>
</tr>
</tbody>
</table>
Comparing with split closure

Time/Quality trade-off:

- Split Closure
- CPX-bbc

Daniel Espinoza From Branching to Cutting (and back again)
Comparing with split closure

5120 Performance profile

![5120 Performance profile graph]

- Split Closure
- CPX-bbc
- 5120
- 5120 cuts

Daniel Espinoza
From Branching to Cutting (and back again)
It seems this approach might work in practice.

- There are several *practical* speed-ups implemented but not explained here.
- Current *public* interface in CPLEX does not allow to directly access (at will) leaves of the B&B&C tree.
- Can *mapped* cuts converge to full description?
- What about precision?
- Lots of related questions...
Final Remarks

- It seems this approach might work in practice.
- There are several *practical* speed-ups implemented but not explained here.
- Current *public* interface in CPLEX does not allow to directly access (at will) leaves of the B&B&C tree.
- Can *mapped* cuts converge to full description?
  - Probably not... but how far?
- What about precision?
- Lots of related questions...
It seems this approach might work in practice.

There are several *practical* speed-ups implemented but not explained here.

Current *public* interface in CPLEX does not allow to directly access (at will) leaves of the B&B&C tree.

Can *mapped* cuts converge to full description?
- Probably not... but how far?

What about precision?

Lots of related questions...
Final Remarks

- It seems this approach might work in practice.
- There are several *practical* speed-ups implemented but not explained here.
- Current *public* interface in CPLEX does not allow to directly access (at will) leaves of the B&B&C tree.
- Can *mapped* cuts converge to full description?
  - Probably not... but how far?
- What about precision?
- Lots of related questions...
Final Remarks

- It seems this approach might work in practice.
- There are several *practical* speed-ups implemented but not explained here.
- Current *public* interface in CPLEX does not allow to directly access (at will) leaves of the B&B&C tree.
- Can *mapped* cuts converge to full description?
  - Probably not... but how far?
- What about precision?
- Lots of related questions...
It seems this approach might work in practice.

There are several *practical* speed-ups implemented but not explained here.

Current *public* interface in CPLEX does not allow to directly access (at will) leaves of the B&B&C tree.

Can *mapped* cuts converge to full description?
  - Probably not... but how far?

What about precision?

Lots of related questions...
Final Remarks

- It seems this approach might work in practice.
- There are several *practical* speed-ups implemented but not explained here.
- Current *public* interface in CPLEX does not allow to directly access (at will) leaves of the B&B&C tree.
- Can *mapped* cuts converge to full description?
  - Probably not... but how far?
- What about precision?
- Lots of related questions...