

Branching Algorithms

Divide and Conquer - Partition search space and recurse.

Use domain specific method for cutting off parts of search space.

Integer Linear Program

P :

Input: $A \in \mathbf{R}^{m \times n}$, $b \in \mathbf{R}^m$, $c \in \mathbf{R}^n$,

Output: $x \in F$ minimizing $\langle c, x \rangle$ where $F = \{x \in \mathbf{Z}^n | Ax \leq b\}$

Bounded Integer Linear Program

P :

Input: $A \in \mathbf{R}^{m \times n}$, $b \in \mathbf{R}^m$, $c \in \mathbf{R}^n$, $l \in \mathbf{Z}^n$, $u \in \mathbf{Z}^n$, $l_i < u_i$.

Output: $x \in F$ minimizing $\langle c, x \rangle$ where
 $F = \{x \in \mathbf{Z}^n \mid Ax \leq b, \forall i : l_i \leq x_i \leq u_i\}$

Relaxed Program

P^* :

Input: $A \in \mathbf{R}^{m \times n}$, $b \in \mathbf{R}^m$, $c \in \mathbf{R}^n$, $l \in \mathbf{Z}^n$, $u \in \mathbf{Z}^n$, $l_i < u_i$.

Output: $x \in F^*$ minimizing $\langle c, x \rangle$ where
 $F^* = \{x \in \mathbf{R}^n \mid Ax \leq b, \forall i : l_i \leq x_i \leq u_i\}$

Branch and Bound for ILP

```
ILPSolve( ILPinstance  $P$  )  
   $P^* := \text{Relax}(P)$ ; Run simplex algorithm on  $P^*$ ;
```

Branch and Bound for ILP

```
ILPSolve( ILPinstance  $P$  )  
   $P^* := \text{Relax}(P)$ ; Run simplex algorithm on  $P^*$ ;  
  if  $P^*$  is infeasible
```

Branch and Bound for ILP

```
ILPSolve( ILPinstance  $P$  )  
   $P^* := \text{Relax}(P)$ ; Run simplex algorithm on  $P^*$ ;  
  if  $P^*$  is infeasible then return (null,  $\infty$ );
```

Branch and Bound for ILP

```
ILPSolve( ILPinstance  $P$  )  
   $P^* := \text{Relax}(P)$ ; Run simplex algorithm on  $P^*$ ;  
  if  $P^*$  is infeasible then return (null,  $\infty$ );  
   $(x^*, \lambda^*) := \text{Optimal solution/value to } P^*$ ;
```

Branch and Bound for ILP

```
ILPSolve( ILPinstance  $P$  )  
   $P^* := \text{Relax}(P)$ ; Run simplex algorithm on  $P^*$ ;  
  if  $P^*$  is infeasible then return (null,  $\infty$ );  
   $(x^*, \lambda^*) := \text{Optimal solution/value to } P^*$ ;  
  if  $x^* \in \mathbf{Z}^n$ 
```

Branch and Bound for ILP

```
ILPSolve( ILPinstance  $P$  )  
   $P^* := \text{Relax}(P)$ ; Run simplex algorithm on  $P^*$ ;  
  if  $P^*$  is infeasible then return (null,  $\infty$ );  
   $(x^*, \lambda^*) := \text{Optimal solution/value to } P^*$ ;  
  if  $x^* \in \mathbf{Z}^n$  then return  $(x^*, \lambda^*)$ ;
```

Branch and Bound for ILP

```
ILPSolve( ILPinstance  $P$  )  
   $P^* := \text{Relax}(P)$ ; Run simplex algorithm on  $P^*$ ;  
  if  $P^*$  is infeasible then return (null,  $\infty$ );  
   $(x^*, \lambda^*) := \text{Optimal solution/value to } P^*$ ;  
  if  $x^* \in \mathbf{Z}^n$  then return  $(x^*, \lambda^*)$ ;
```

Pick i so that $x_i^* \notin \mathbf{Z}$

Branch and Bound for ILP

```
ILPSolve( ILPinstance  $P$  )  
   $P^* := \text{Relax}(P)$ ; Run simplex algorithm on  $P^*$ ;  
  if  $P^*$  is infeasible then return (null,  $\infty$ );  
   $(x^*, \lambda^*) :=$  Optimal solution/value to  $P^*$ ;  
  if  $x^* \in \mathbf{Z}^n$  then return  $(x^*, \lambda^*)$ ;  
  
  Pick  $i$  so that  $x_i^* \notin \mathbf{Z}$   
   $(x^l, \lambda^l) := \text{ILPSolve}( P[u_i \leftarrow \lfloor x_i^* \rfloor] )$   
  
   $(x^r, \lambda^r) := \text{ILPSolve}( P[l_i \leftarrow \lceil x_i^* \rceil] )$ 
```

Branch and Bound for ILP

```
ILPSolve( ILPinstance  $P$  )  
   $P^* := \text{Relax}(P)$ ; Run simplex algorithm on  $P^*$ ;  
  if  $P^*$  is infeasible then return (null,  $\infty$ );  
   $(x^*, \lambda^*) :=$  Optimal solution/value to  $P^*$ ;  
  if  $x^* \in \mathbf{Z}^n$  then return  $(x^*, \lambda^*)$ ;  
  
  Pick  $i$  so that  $x_i^* \notin \mathbf{Z}$   
   $(x^l, \lambda^l) := \text{ILPSolve}( P[u_i \leftarrow \lfloor x_i^* \rfloor] )$   
  
   $(x^r, \lambda^r) := \text{ILPSolve}( P[l_i \leftarrow \lceil x_i^* \rceil] )$   
  if  $\lambda^l < \lambda^r$  then return  $(x^l, \lambda^l)$  else return  $(x^r, \lambda^r)$ 
```

Branch and Bound for ILP

```
ILPSolve( ILPinstance  $P$  )  
   $P^* := \text{Relax}(P)$ ; Run simplex algorithm on  $P^*$ ;  
  if  $P^*$  is infeasible then return (null,  $\infty$ );  
   $(x^*, \lambda^*) := \text{Optimal solution/value to } P^*$ ;  
  if  $x^* \in \mathbf{Z}^n$  then return  $(x^*, \lambda^*)$ ;  
  
  Pick  $i$  so that  $x_i^* \notin \mathbf{Z}$   
   $(x^l, \lambda^l) := \text{ILPSolve}( P[u_i \leftarrow \lfloor x_i^* \rfloor] )$   
  
   $(x^r, \lambda^r) := \text{ILPSolve}( P[l_i \leftarrow \lceil x_i^* \rceil] )$   
  if  $\lambda^l < \lambda^r$  then return  $(x^l, \lambda^l)$  else return  $(x^r, \lambda^r)$ 
```

Branch and Bound for ILP

ILPSolve(ILPinstance P , Real currentbest)

$P^* := \text{Relax}(P)$; Run simplex algorithm on P^* ;

if P^* is infeasible **then return** (null, ∞);

$(x^*, \lambda^*) :=$ Optimal solution/value to P^* ;

if $x^* \in \mathbf{Z}^n$ **then return** (x^*, λ^*) ;

Pick i so that $x_i^* \notin \mathbf{Z}$

$(x^l, \lambda^l) := \text{ILPSolve}(P[u_i \leftarrow \lfloor x_i^* \rfloor] \quad)$

$(x^r, \lambda^r) := \text{ILPSolve}(P[l_i \leftarrow \lceil x_i^* \rceil] \quad)$

if $\lambda^l < \lambda^r$ **then return** (x^l, λ^l) **else return** (x^r, λ^r)

Branch and Bound for ILP

```
ILPSolve( ILPinstance  $P$ , Real currentbest)
   $P^* := \text{Relax}(P)$ ; Run simplex algorithm on  $P^*$ ;
  if  $P^*$  is infeasible then return (null,  $\infty$ );
   $(x^*, \lambda^*) := \text{Optimal solution/value to } P^*$ ;
  if  $x^* \in \mathbf{Z}^n$  then return  $(x^*, \lambda^*)$ ;
  if  $\lambda^* \geq \text{currentbest}$  then return (null,  $\infty$ );
  Pick  $i$  so that  $x_i^* \notin \mathbf{Z}$ 
   $(x^l, \lambda^l) := \text{ILPSolve}( P[u_i \leftarrow \lfloor x_i^* \rfloor] )$ 
   $(x^r, \lambda^r) := \text{ILPSolve}( P[l_i \leftarrow \lceil x_i^* \rceil] )$ 
  if  $\lambda^l < \lambda^r$  then return  $(x^l, \lambda^l)$  else return  $(x^r, \lambda^r)$ 
```

Branch and Bound for ILP

```
ILPSolve( ILPinstance  $P$ , Real currentbest)
   $P^* := \text{Relax}(P)$ ; Run simplex algorithm on  $P^*$ ;
  if  $P^*$  is infeasible then return (null,  $\infty$ );
   $(x^*, \lambda^*) :=$  Optimal solution/value to  $P^*$ ;
  if  $x^* \in \mathbf{Z}^n$  then return  $(x^*, \lambda^*)$ ;
  if  $\lambda^* \geq$  currentbest then return (null,  $\infty$ );
  Pick  $i$  so that  $x_i^* \notin \mathbf{Z}$ 
   $(x^l, \lambda^l) := \text{ILPSolve}( P[u_i \leftarrow \lfloor x_i^* \rfloor], \text{currentbest} )$ 

   $(x^r, \lambda^r) := \text{ILPSolve}( P[l_i \leftarrow \lceil x_i^* \rceil] )$ 
  if  $\lambda^l < \lambda^r$  then return  $(x^l, \lambda^l)$  else return  $(x^r, \lambda^r)$ 
```

Branch and Bound for ILP

```
ILPSolve( ILPinstance  $P$ , Real currentbest)
   $P^* := \text{Relax}(P)$ ; Run simplex algorithm on  $P^*$ ;
  if  $P^*$  is infeasible then return (null,  $\infty$ );
   $(x^*, \lambda^*) := \text{Optimal solution/value to } P^*$ ;
  if  $x^* \in \mathbf{Z}^n$  then return  $(x^*, \lambda^*)$ ;
  if  $\lambda^* \geq \text{currentbest}$  then return (null,  $\infty$ );
  Pick  $i$  so that  $x_i^* \notin \mathbf{Z}$ 
   $(x^l, \lambda^l) := \text{ILPSolve}( P[u_i \leftarrow \lfloor x_i^* \rfloor], \text{currentbest} )$ 
  if  $\lambda^l < \text{currentbest}$  then currentbest :=  $\lambda^l$ ;
   $(x^r, \lambda^r) := \text{ILPSolve}( P[l_i \leftarrow \lceil x_i^* \rceil] )$ 
  if  $\lambda^l < \lambda^r$  then return  $(x^l, \lambda^l)$  else return  $(x^r, \lambda^r)$ 
```

Branch and Bound for ILP

```
ILPSolve( ILPinstance  $P$ , Real currentbest)
   $P^* := \text{Relax}(P)$ ; Run simplex algorithm on  $P^*$ ;
  if  $P^*$  is infeasible then return (null,  $\infty$ );
   $(x^*, \lambda^*) := \text{Optimal solution/value to } P^*$ ;
  if  $x^* \in \mathbf{Z}^n$  then return  $(x^*, \lambda^*)$ ;
  if  $\lambda^* \geq \text{currentbest}$  then return (null,  $\infty$ );
  Pick  $i$  so that  $x_i^* \notin \mathbf{Z}$ 
   $(x^l, \lambda^l) := \text{ILPSolve}( P[u_i \leftarrow \lfloor x_i^* \rfloor], \text{currentbest} )$ 
  if  $\lambda^l < \text{currentbest}$  then currentbest :=  $\lambda^l$ ;
   $(x^r, \lambda^r) := \text{ILPSolve}( P[l_i \leftarrow \lceil x_i^* \rceil], \text{currentbest} )$ 
  if  $\lambda^l < \lambda^r$  then return  $(x^l, \lambda^l)$  else return  $(x^r, \lambda^r)$ 
```

Branch and Bound

activeset := {**root**}

currentbest := ∞ ;

while activeset is not empty **do**

 choose a branching node, node $k \in$ activeset

 remove node k from activeset;

 generate the children of node k , child i , $i = 1, \dots, n_k$,

 and the corresponding lower bounds z_i ;

for $i = 1, \dots, n_k$ **do**

if $z_i \geq$ currentbest **then** kill child i ;

else if child i is a complete solution **then**

 currentbest := z_i ;

else add child i to activeset