

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 $F = \{x \in \mathbf{Z}^n \mid Ax \leq b\}$

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

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,  $\infty$ );  
   $(x^*, \lambda^*) :=$  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, ∞);

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

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

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

Branch and Bound 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^*) ;

if $\lambda^* \geq$ currentbest **then return** (null, ∞);

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 $\lambda^l <$ currentbest **then** currentbest := λ^l ;

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

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, ∞);

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

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

if $\lambda^* \geq \text{currentbest}$ **then return** (null, ∞);

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 := λ^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, λ^l) **else return** (x^r, λ^r)

Termination

If $u_i = l_i$ for all i we terminate immediately.

If we do not terminate, then in each recursive call, either one l_i has increased or one u_i has decreased.

Since u_i, l_i are integers, we terminate.

Unit of time = One run of pseudocode (except recursive calls)

Unit of time = One run of pseudocode (except recursive calls)

We can show by induction: $T \leq \prod_{i=1}^n (u_i - l_i + 1)$.

Unit of time = One run of pseudocode (except recursive calls)

We can show by induction: $T \leq \prod_{i=1}^n (u_i - l_i + 1)$.

In practice, it can be hoped that many subcalls finish early.

However, all known algorithms for solving integer linear programs are worst case exponential time.

Non-depth first Branch and Bound

Instead of letting the recursion stack represent the branch and bound tree, we can make it an explicit data structure

This allows us to do **best first search**: Choose the expand the subproblem with the lowest relaxed solution value.

Hope: This also contains very good valid solutions which will allow the currentbest value to decrease significantly.

activeset := {**root**}

currentbest := ∞ ;

while activeset is not empty **do**

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

 remove node k from activeset;

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

 and the corresponding relaxed optimal values z_i ;

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

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

else if child i optimal solution is integral **then**

 currentbest := z_i ;

else add child i to activeset

Ingredients:

- 1 Relaxation method. Expands space of feasible solution. Relaxed instance must be *efficiently solvable*.
- 2 Search space division.
 - The optimal solution of the relaxed problem can not be feasible in any of the relaxed subproblems (it should be “killed”)
 - Any non-relaxed feasible solution should be feasible in one of the subproblems.

Given a weighted complete graph $G = (V, w)$, $V = \{0, \dots, n - 1\}$, $w : V \times V \rightarrow \mathbf{R}^+$, $w(i, j) = w(j, i)$, find the lightest Hamiltonian path in graph, i.e., find a permutation π of V minimizing

$$\sum_{i=0}^{n-1} w(\pi(i), \pi((i + 1) \bmod n)).$$

Let $V' = V$ and $w : V \times V' \rightarrow \mathbf{R}^+$, $w'(u, v) = w(u, v)$.

If π a TSP tour, $\{(\pi(i), \pi(i + 1) \bmod n))\}$ is a *perfect matching* between V and V'

The weight of the perfect matching is the length of the tour.

Let $V' = V$ and $w : V \times V' \rightarrow \mathbf{R}^+$, $w'(u, v) = w(u, v)$.

If π a TSP tour, $\{(\pi(i), \pi(i + 1) \bmod n)\}$ is a *perfect matching* between V and V'

The weight of the perfect matching is the length of the tour.

A minimum weight perfect matching can be found efficiently using min cost flow.

The minimum weight perfect matching corresponds to a *cycle cover* of F .

If this is not a TSP tour, it has a cycle of length $< n$.

Branch on which edge of the cycle to *not* include in the tour (put their weights to ∞ in subproblems).

Alternative relaxation

Given a TSP tour, by removing the two edges adjacent to node 1, we get a *spanning tree* of $V - \{1\}$.

The weight of the spanning tree + weight of removed edges is the length of the tour.

Alternative relaxation

Given a TSP tour, by removing the two edges adjacent to node 1, we get a *spanning tree* of $V - \{1\}$.

The weight of the spanning tree + weight of removed edges is the length of the tour.

Def 1-tree = spanning tree on $\{2,3,\dots, n\}$ + two edges out of 1.

The minimum weight 1-tree can be found efficiently.

If the optimal relaxed solution (the minimum weight 1-tree) is not a cycle, it must have a node v of degree ≥ 3 .

Branch on not including each edge adjacent to v in the tour (put their weights to infinity).

Only one lower bound to be obtained using 1-tree on original weights.

Only one lower bound to be obtained using 1-tree on original weights.

Given values $\alpha_i, i \in V$, add $\alpha_i + \alpha_j$ to w_{ij} .

Only one lower bound to be obtained using 1-tree on original weights.

Given values $\alpha_i, i \in V$, add $\alpha_i + \alpha_j$ to w_{ij} .

Any TSP tour increases with the amount $2 \sum \alpha_i$ so optimal tour remains the same.

Only one lower bound to be obtained using 1-tree on original weights.

Given values $\alpha_i, i \in V$, add $\alpha_i + \alpha_j$ to w_{ij} .

Any TSP tour increases with the amount $2 \sum \alpha_i$ so optimal tour remains the same.

Value of optimal 1-tree may change. Thus, we can try to optimize the choice of (α_i) . The best value can be found efficiently (Held and Karp)