CS 161: Design and Analysis of Algorithms
Linear Programming II: Duality/Reductions

- Recap
- Example
- Duality
- Reductions
Recap

Objective Function

\[
\max \sum_i c_i x_i
\]

Constraints

\[
\sum_i A_{j,i} x_i \leq b_j \forall j
\]

\[
x_i \geq 0 \forall i
\]
Profit Maximization

• Suppose a candy company can make two types of candy.
• The company can produce up to 500 boxes a day of the first type, each box making the company $5.
• They can produce up to 300 boxes of the second type, which box making them $10.
• The company can only produce 600 boxes of candy per day.
Profit Maximization

- Variables $x_1$ and $x_2$ represent the number of boxes of candy 1 and 2 produced
- $0 \leq x_1, x_2$
- $x_1 \leq 500$
- $x_2 \leq 300$
- $x_1 + x_2 \leq 600$
- Maximize: $5x_1 + 10x_2$
Simplex Method
Simplex Method

- $x_1 \leq 500$
Simplex Method

• $x_2 \leq 300$
Simplex Method

- $x_1 + x_2 \leq 600$
Feasible Region
Simplex Method

• Recall:
  – Start at any vertex of the feasible region
  – Repeatedly move to neighboring vertex with more optimal solution
Simplex Method

Profit: $0

Diagram showing a constant profit of $0.
Simplex Method

Profit: $0

Profit: $2500

Profit: $3000
Simplex Method

Profit: $0

Profit: $2500

Profit: $3000
Simplex Method

Profit: $0 

Profit: $3000 

Profit: $4500 

Profit: $2500
Simplex Method

Profit: $0
Profit: $2500
Profit: $3000
Profit: $4500
Simplex Method

Profit: $0

Profit: $2500

Profit: $3000

Profit: $3500

Profit: $4500
Simplex Method

Profit: $4500

$x_1 = 300$
$x_2 = 300$
Why is a vertex always optimal?

- Set of points where objective function is equal to a constant c forms a hyperplane
- Change value of objective function by shifting this hyperplane
- Hyperplane must intersect feasible region
- Objective function maximized when intersection is at an extreme
Objective Function

Profit: $1000
Profit: $2000
Profit: $3000
Profit: $4000
Profit: $4500
Proof of Optimality?

• We wish to prove that our solution is optimal be showing that there is no way to make more than $4500
Proof of Optimality?

• Recall the LP:
  – Maximize: $5 \, x_1 + 10 \, x_2$ subject to the constraints
  – $0 \leq x_1, x_2$
  – $x_1 \leq 500$
  – $x_2 \leq 300$
  – $x_1 + x_2 \leq 600$
Proof of Optimality

• Let’s try combining constraints
• $x_1 + x_2 \leq 600 \rightarrow 5x_1 + 5x_2 \leq 3000$
• $x_2 \leq 300 \rightarrow 5x_2 \leq 1500$
• Add together: $5x_1 + 10x_2 \leq 4500$
• But $5x_1 + 10x_2$ is just the objective function!
• Therefore, the objective function is always at most $4500$, so our solution is optimal
Proof of Optimality

• Goal: combine constraints together to get a bound on the objective function

\[
\max \sum_i c_i x_i \\
\sum_i A_{j,i} x_i \leq b_j \forall j \\
x_i \geq 0 \forall i
\]
Proof of Optimality

\[ \sum_j y_j \left( \sum_i A_{j,i} x_i \right) \leq \sum_j b_j y_j \]
Proof of Optimality

• Want bound on objective function, so want

\[ \sum_{i} c_i x_i \leq \sum_{i} \left( \sum_{j} A_{j,i} y_j \right) x_i \]
Proof of Optimality

• Sufficient condition:

\[ c_i \leq \sum_j A_{j,i} y_j \]
Proof of Optimality

• Thus, as long as

\[ c_i \leq \sum_j A_{j,i} y_j \]

• We have that

\[ \sum_i c_i x_i \leq \sum_j b_j y_j \]
Proof of Optimality

• Goal:

\[
\min \sum_{j} b_j y_j \\
\sum_{j} A_{j,i} y_i \geq c_i \forall i \\
y_i \geq 0 \forall i
\]
Duality

• Our goal then is to solve another linear program!
• This alternate linear program is known as the **dual** of the original program
• By construction, optimal solution of dual is at least optimal solution of primal
• **Duality Theorem**: optimums coincide
Matrix Notation

\[
\begin{align*}
\text{max } c^T x \\
A x & \leq b \\
x & \geq 0
\end{align*}
\]

\[
\begin{align*}
\text{min } y^T b \\
y^T A & \geq c^T \\
y & \geq 0
\end{align*}
\]
Reductions

• We already saw that linear programming can be used to solve the max flow problem
• What we showed was a reduction: Given an instance of the max flow problem, we:
  – Construct a linear program
  – Solve the linear program
  – Convert solution of linear program into solution for max flow
Reduction

• In general, solving problem A reduces to solving problem B if:
  – Given an instance of problem A, we can efficiently compute an instance of problem B***
  – Given a solution to the instance of problem B, we can efficiently construct a solution to the instance of problem A***

*** Need to define “efficient”
The Power of Reductions

• If solving problem A reduces to solving problem B, then we can reuse an algorithm to solve problem B in order to solve A
  – Convert instance of A into instance of B
  – Solve B using our algorithm
  – Convert solution to solution for A
The Power of Reductions

• What makes linear programming so powerful is that many problems can be reduced to linear programs
Reductions so Far

- Max Flow
- Profit Maximization
Bipartite Matching

• Suppose we have a list of n boys and n girls, and set of pairs (i,j) that mean boy i and girl j like each other

• Can we pair every boy with a girl so that each pair likes each other?
Bipartite Matching
Bipartite Matching
Bipartite Matching as Maximum Flow
Bipartite Matching as Maximum Flow

• To see if there is a perfect matching, direct all edges from boy to girl
• Add a source node $s$ with edges to each boy
• Add a sink node $t$ with edges from each girl
• Compute the max flow
• If there is a flow equal to $n$, then there is a perfect matching
Bipartite Matching as Maximum Flow
Problem?
Bipartite Matching as Maximum Flow

• Our Max Flow algorithm always produces an integer flow if the edge weights are integers
  – Always increments flow by integer value
• Therefore, the maximum flow in the bipartite matching problem has integer flow on each edge
• To get matching, take edges with flow =1
Bipartite Matching as Maximum Flow
Bipartite Matching as Maximum Flow
Linear Programming for Shortest Path

• Can think of a path from s to t as a flow of size 1 from s to t

• Shortest path problem: find flow that minimizes total weight of edges
Linear Programming for Shortest Path

\[
\begin{align*}
\min & \quad \sum_{e} f_{e} w(e) \\
\sum_{(u,v)} f_{(u,v)} & = \sum_{(v,w)} f_{(v,w)} \forall v \neq s, t \\
\sum_{(s,v)} f_{(s,v)} & = 1 \\
f_{e} & \geq 0
\end{align*}
\]
Linear Programming for Shortest Path

• Any path from s to t represents an integer solution to this problem
• Objective function evaluated on such a path is just equal to the weight of the path
• Will the linear program give integer solution?
Linear Programming for Shortest Path
Linear Programming for Shortest Path
Linear Programming for Shortest Path

- In general, solution to linear program does not give us a path
- Solution: take any path from s to t using only edges with non-zero flow
Linear Programming for Shortest Path
Linear Programming for Shortest Path

• Proof of correctness:
  – Suppose we have optimal flow $F$. Let $C_F$ be the cost
  – Let $P$ be some path from $s$ to $t$ using only edges with non-zero flow.
  – Let $C_P$ be the cost of the flow obtained by sending 1 unit of flow along the edges of $P$
  – Claim: $C_P \leq C_F$
Linear Programming for Shortest Path

• Claim: $C_p \leq C_F$
  – Proof: Let $r$ be the minimum amount of flow in $F$ along any of the edges in $P$
  – Subtract $r$ from the flow along each edge in $P$
  – $C_{F'} = C_F - r C_p$
  – The size of the flow $F'$ is $1-r$. Multiply the flow in each edge by $1/(1-r)$
  – $C_{F''} = (C_F - r C_p)/(1-r)$
Linear Programming for Shortest Path

• Claim: \( C_p \leq C_F \)
  - \( F'' \) is a flow of size 1 with cost \( C_{F''} = (C_F - r C_p)/(1-r) \)
  - Therefore, \( C_{F''} \geq C_F \)
Linear Programming for Shortest Path

\[
C_F \leq \frac{C_F - rC_P}{1 - r}
\]

\[
\frac{r}{1 - r} C_F \geq \frac{r}{1 - r} C_P
\]

\[
C_F \geq C_P
\]
Linear Programming for Shortest Path

• Proof of correctness:
  – $C_p \leq C_f$, so $C_p = C_f$
  – Therefore, the path $P$ is also an optimal solution to the linear program
  – Therefore, it must be the shortest path
Reductions

• Reductions allow us to solve one problem using algorithm for another problem
• Reductions can also be used to show the impossibility of a good algorithm
Reductions

• Suppose we have some really hard problem A, and we don’t think we can solve A efficiently
• Suppose further that we have a reduction from solving A to solving some other problem B
• What if we had an efficient algorithm to solve B?
Applications

• Complexity Theory:
  – To prove that we can’t solve some particular problem A efficiently, we often come up with some contrived problem B
  – B is defined in a way that allows us to prove that B cannot be solved efficiently
  – We then show a reductions from solving B to solving A, thus showing that A cannot be solved efficiently
Applications

• Cryptography:
  – Prior to the 1970s, cryptographers typically created schemes that resisted known attacks
  – What about attacks that we haven’t though of yet?
  – Goal of modern cryptography: prove no such attacks exist
Applications

• Cryptography:
  – Unfortunately, we have not been able to prove that any useful scheme is secure against all attacks
  – Instead, we start with some hard problem (e.g. factoring integers)
  – We show that if an adversary can break our scheme, they can solve the hard problem
  – Thus, if we assume the problem cannot be solved efficiently, no adversary can break our scheme efficiently