Information from a dual LP solution can sometimes be used to derive good approximation algorithms.

Let \( y^* \) be an optimal solution to the SC dual LP, and consider the solution in which we choose all subsets for which the corresponding dual inequality is **tight**; i.e., the inequality is met with equality for subset \( S_j \), and \( \sum_{i: e_i \in S_j} y_i^* = w_j \).

Let \( I' \) denote the indices of the subsets in this solution.

This algorithm also is an \( f \)-approximation algorithm for the set cover problem.

**Lemma 1.7:** The collection of subsets \( S_j, j \in I' \), is a set cover.

**Proof:** Suppose that there exists some uncovered element \( e_k \). Then for each subset \( S_j \) containing \( e_k \), it must be the case that

\[
\sum_{i: e_i \in S_j} y_i^* < w_j.
\]

Let \( \epsilon \) be the smallest difference between the RHS and LHS of all constraints involving \( e_k \); i.e.,

\[
\epsilon = \min_{j: e_k \in S_j} \left( w_j - \sum_{i: e_i \in S_j} y_i^* \right).
\]

By the above inequality, we know that \( \epsilon > 0 \).
Consider now a new dual solution $y'$ in which $y'_k = y^*_k + \epsilon$ and every other component of $y'$ is the same as in $y^*$. Then $y'$ is a dual feasible solution since for each $j$ such that $e_k \in S_j$,

$$\sum_{i: e_i \in S_j} y'_i = \sum_{i: e_i \in S_j} y^*_i + \epsilon \leq w_j$$

by the definition of $\epsilon$. For each $j$ such that $e_k \notin S_j$,

$$\sum_{i: e_i \in S_j} y'_i = \sum_{i: e_i \in S_j} y^*_i \leq w_j,$$

as before. Furthermore, $\sum_{i=1}^n y'_i > \sum_{i=1}^n y^*_i$, which contradicts the optimality of $y^*$. Thus, all elements are covered and $I'$ is a set cover.

---

**Theorem 1.8:** The dual rounding algorithm described above is an $f$-approximation algorithm for the set cover problem.

**Proof:** The central idea is the following “charging” argument: when we choose a set $S_j$ to be in the cover, we “pay” for it by charging $y'_i$ to each of its elements $e_i$; each element is charged at most once for each set that contains it (and hence at most $f$ times), and so the total cost is at most $f \sum_{i=1}^m y^*_i$, or $f$ times the dual objective function.

More formally, since $j \in I'$ only if $w_j = \sum_{i: e_i \in S_j} y^*_i$, we have that the cost of the set cover $I'$ is
\[ \sum_{j \in I'} w_j = \sum_{j \in I'} \sum_{i : e_i \in S_j} y_i^* = \sum_{i=1}^n \left\{ j \in I' : e_i \in S_j \right\} \cdot y_i^* \leq \sum_{i=1}^n f_i y_i^* \leq f \sum_{i=1}^n y_i^* \leq f \cdot \text{OPT} \]

The second eq. follows from the fact that when we interchange the order of summation, the coefficient of \( y_i^* \) is equal to the number of times that this term occurs overall. The final ineq. follows from the weak duality property.

• This algorithm can do no better than the algorithm of the previous section
  – We can show that if \( I \) indexes the solution returned by the primal rounding algorithm of the previous section, then \( I \subseteq I' \)
• This follows from a property of optimal LP solutions called **complementary slackness**
• We showed earlier the following for any feasible solution \( x \) to the set cover LP relaxation, and any feasible solution \( y \) to the dual LP:

\[ \sum_{i=1}^n y_i \leq \sum_{i=1}^n y_i \sum_{j:e_i \in S_j} x_j = \sum_{j=1}^m x_j \sum_{i:e_i \in S_j} y_i \leq \sum_{j=1}^m x_jw_j \]
Furthermore, we claimed that strong duality implies that for optimal solutions $x^*$ and $y^*$,

$$\sum_{i=1}^{n} y_i^* = \sum_{j=1}^{m} w_j x_j^*$$

Thus, for any optimal solutions $x^*$ and $y^*$ the two inequalities in the chain of inequalities above must in fact be equalities.

The only way this can happen is that whenever $y_i^* > 0$ then $\sum_{j: e_i \in S_j} x_j^* = 1$, and whenever $x_j^* > 0$, then $\sum_{i: e_i \in S_j} y_i^* = w_j$.

That is, whenever a LP variable (primal or dual) is nonzero, the corresponding constraint in the dual or primal is tight.

These conditions are known as the **complementary slackness** conditions.

Thus, if $x^*$ and $y^*$ are optimal solutions, the complementary slackness conditions must hold.

The converse is also true:

- if $x^*$ and $y^*$ are feasible primal and dual solutions, respectively, then
- if the complementary slackness conditions hold, the values of the two objective functions are equal and therefore
- the solutions must be optimal.
• In the case of the set cover program, if $x_j^* > 0$ for any primal optimal solution $x^*$, then the corresponding dual inequality for $S_j$ must be tight for any dual optimal solution $y^*$

• Recall that in the algorithm of the previous section, we put $j \in I$ when $x_j^* \geq 1/f$

• Thus, $j \in I$ implies that $j \in I'$, so that $I' \supseteq I$

1.5 The primal-dual method

• The basic idea of the algorithm in this section is that the dual rounding algorithm of the previous section uses relatively few properties of an optimal dual solution

• Instead of actually solving the dual LP, we can construct a feasible dual solution with the same properties

• In this case, constructing the dual solution is much faster than solving the dual LP, and hence leads to a much faster algorithm
The algorithm of the previous section used the following properties:

- First, we used the fact that $\sum_{i=1}^{n} y_i \leq \text{OPT}$, which is true for any feasible dual solution $y$.
- Second, we include $j \in I'$ precisely when $\sum_{i: e_i \in S_j} y_i = w_j$, and $I'$ is a set cover.
- These two facts together gave the proof that the cost of $I'$ is no more than $f$ times optimal.
- The proof of Lemma 1.7 (that we have constructed a feasible cover) shows how to obtain an algorithm that constructs a dual solution.

Consider any feasible dual solution $y$, and let $T$ be the set of the indices of all tight dual constraints; i.e., $T = \{j: \sum_{i: e_i \in S_j} y_i = w_j\}$.

- If $T$ is a set cover, then we are done.
- If $T$ is not a set cover, then some item $e_i$ is uncovered, and as shown in the proof of Lemma 1.7 it is possible to improve the dual objective function by increasing $y_i$ by some $\epsilon > 0$.
- More specifically, we can increase $y_i$ by $\min_{j: e_i \in S_j} (w_j - \sum_{k: e_k \in S_j} y_k)$, so that the constraint becomes tight for the subset $S_j$ that attains the minimum.
Additionally, the modified dual solution remains feasible
Thus, we can add $j$ to $T$, and element $e_i$ is now covered by the sets in $T$
We repeat this process until $T$ is a set cover
Since an additional element $e_i$ is covered each time, the process is repeated at most $n$ times
For a complete algorithm, we need to give only an initial dual feasible solution
We can use the solution $y_i = 0$ for each $i = 1, \ldots, n$; this is feasible since each $w_j$, $j = 1, \ldots, m$, is nonnegative

**Algorithm 1.1:**
Primal-dual algorithm for the set cover problem

1. $y \leftarrow 0$
2. $I \leftarrow \emptyset$
3. while there exists $e_i \notin \bigcup_{j \in I} S_j$ do
   i. Increase the dual variable $y_i$ until there is some $\ell$ with $e_i \in S_\ell$ such that $\sum_{j : e_j \in S_\ell} y_j = w_\ell$
   ii. $I \leftarrow I \cup \{\ell\}$
Theorem 1.9: Algorithm 1.1 is an \( f \)-approximation algorithm for the set cover problem.

- LP problems, network flow problems, and shortest path problems (among others) all have primal-dual optimization algorithms
- Primal-dual algorithms start with a dual feasible solution, and use dual information to infer a primal, possibly infeasible, solution
- If the primal solution is indeed infeasible, the dual solution is modified to increase the value of the dual objective function

1.6 A greedy algorithm

- We show that a greedy algorithm gives an approximation algorithm with a performance guarantee that is often significantly better than \( f \)
- Greedy algorithms optimize each particular decision, even if this sequence of locally optimal decisions may not lead to a globally opt. solution
- Greedy algorithms are very easy to implement, and hence greedy algorithms are a commonly used heuristic, even when they have no performance guarantee
ALGORITHM 1.2:
A greedy algorithm for the set cover problem

1. $I \leftarrow \emptyset$
2. $\hat{S}_j \leftarrow S_j \quad \forall j$
3. while $I$ is not a set cover do
4. $\ell \leftarrow \arg \min_{j: \hat{S}_j \neq \emptyset} (w_j/|\hat{S}_j|)$
5. $I \leftarrow I \cup \{\ell\}$
6. $\hat{S}_j \leftarrow \hat{S}_j - S_\ell \quad \forall j$

- In each round, we choose the set that minimizes the ratio of its weight to the number of currently uncovered elements it contains.
- In the event of a tie, we pick an arbitrary set that achieves the minimum ratio.
- We continue choosing sets until all elements are covered.
- Obviously, this will yield a polynomial-time algorithm, since there can be no more than $m$ rounds, and in each we compute $O(m)$ ratios, each in constant time.
Let $H_k$ denote the $k$th harmonic number: i.e.,
\[ H_k = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{k} \]

Note that $H_k \approx \ln k$

The following fact is one that we will use many times in the course

It can be proven with simple algebraic manipulations

**Fact 1.10:** Given positive numbers $a_1, \ldots, a_k$ and $b_1, \ldots, b_k$, then
\[
\min_{i=1, \ldots, k} \frac{a_i}{b_i} \leq \frac{\sum_{i=1}^{k} a_i}{\sum_{i=1}^{k} b_i} \leq \max_{i=1, \ldots, k} \frac{a_i}{b_i}
\]

**Theorem 1.11:** Algorithm 1.2 is an $H_n$-approximation algorithm for the set cover problem.

The basic intuition for the analysis of the algorithm is as follows

Let $OPT$ denote the value of an optimal solution to the set cover problem

An optimal solution covers all $n$ elements with a solution of weight $OPT$

Therefore, there must be some subset that covers its elements with an average weight of at most $OPT/n$
Similarly, after $k$ elements have been covered, the optimal solution can cover the remaining $n - k$ elements with a solution of weight $OPT$

This implies that there is some subset that covers its remaining uncovered elements with an average weight of at most $OPT/(n - k)$

So in general the greedy algorithm pays about $OPT/(n - k + 1)$ to cover the $k$th uncovered element, giving a performance guarantee of

$$\sum_{k=1}^{n} \frac{1}{n - k + 1} = H_n$$

We can improve the performance guarantee of the algorithm slightly by using the dual of the LP relaxation in the analysis

Let $g$ be the maximum size of any subset $S_j$; i.e.,

$$g = \max_j |S_j|$$

Recall that $Z^*_{LP}$ is the optimum value of the LP relaxation for the set cover problem

The following theorem immediately implies that the greedy algorithm is an $H_g$-approximation algorithm, since $Z^*_{LP} \leq OPT$

**Theorem 1.12:** Algorithm 1.2 returns a solution indexed by $I$ such that $\sum_{j \in I} w_j \leq H_g \cdot Z^*_{LP}$.
Proof: We will construct an infeasible dual solution $y$ s.t. $\sum_{j \in I} w_j = \sum_{i=1}^n y_i$. We will then show that $y' = \frac{1}{H_g} y$ is a feasible dual solution. By the weak duality theorem, $\sum_{i=1}^n y_i' \leq Z_{LP}^*$, so that $\sum_{j \in I} w_j = \sum_{i=1}^n y_i = H_g \sum_{i=1}^n y_i' \leq H_g \cdot \text{OPT}$. We will eventually see the reason we choose to divide the infeasible dual solution $y$ by $H_g$.

Dual fitting: construct an infeasible dual solution whose value is equal to the value of the primal solution constructed, scaling the dual solution by a single value makes it feasible.

Suppose we choose to add subset $S_j$ to our solution in iteration $k$. Then for each $e_i \in S_j$, we set $y_i = w_j / |S_j|$. Since each $e_i \in S_j$ is uncovered in iteration $k$, and is then covered for the remaining iterations of the algorithm (because we added $S_j$), the dual variable $y_i$ is set to a value exactly once; in particular, it is set in the iteration in which element $e_i$ is covered. Furthermore, $w_j = \sum_{i: e_i \in S_j} y_i$; that is, the weight of the subset $S_j$ chosen in the $k$th iteration is equal to the sum of the duals $y_i$ of the uncovered elements that are covered in the $k$th iteration. This immediately implies that $\sum_{j \in I} w_j = \sum_{i=1}^n y_i$. 
It remains to prove that the dual solution \( y' = \frac{1}{H_g} y \) is feasible. We must show that for each subset \( S_j \), \( \sum_{i:e_i \in S_j} y_i' \leq w_j \). Pick an arbitrary subset \( S_j \). Let \( a_k \) be the number of elements in this subset that are still uncovered at the beginning of the \( k \)th iteration, so that \( a_1 = |S_j| \), and \( a_{k+1} = 0 \). Let \( A_k \) be the uncovered elements of \( S_j \) covered in the \( k \)th iteration, so that \( |A_k| = a_k - a_{k+1} \). If subset \( S_p \) is chosen in the \( k \)th iteration, then for each element \( e_i \in A_k \) covered in the \( k \)th iteration,

\[
y_i' = \frac{w_p}{H_S|S_p|} \leq \frac{w_j}{H_g a_k}
\]

- Above \( S_p \) is the set of uncovered elements of \( S_p \) at the beginning of the \( k \)th iteration.
- The inequality follows because if \( S_p \) is chosen in the \( k \)th iteration, it must minimize the ratio of its weight to the number of uncovered elements it contains.

\[
\sum_{i:e_i \in S_j} y_i' = \sum_{k=1}^{\ell} \sum_{i:e_i \in A_k} y_i' \leq \sum_{k=1}^{\ell} (a_k - a_{k+1}) \frac{w_j}{H_g a_k} \\
\leq \frac{w_j}{H_g} \sum_{k=1}^{\ell} \frac{a_k - a_{k+1}}{a_k}
\]
\[
\leq \frac{w_j}{H_g} \sum_{k=1}^{\ell} \left( \frac{1}{a_k} + \frac{1}{a_k - 1} + \cdots + \frac{1}{a_k+1 + 1} \right)
\]

\[
\leq \frac{w_j}{H_g} \sum_{i=1}^{s_j} \frac{1}{i}
\]

\[
= \frac{w_j}{H_g} H |s_j|
\]

\[
\leq w_j
\]

where the final inequality follows because $|s_j| \leq g$.

The reason for scaling the dual solution by $H_g$: we know that $H |s_j| \leq H_g$ for all sets $j$. 

- No approximation algorithm for the set cover problem with guarantee better than $H_n$ is possible, under an assumption slightly stronger than $P = NP$.

**Theorem 1.13**: If there is a $c \ln n$-approximation algorithm for the unweighted set cover problem for some constant $c < 1$, then there is an $O(n^{O(\log \log n)})$-time deterministic algorithm for each NP-complete problem.

**Theorem 1.14**: There exists some constant $c > 0$ s.t. if there exists a $c \ln n$-approximation algorithm for the unweighted set cover problem, then $P = NP$. 

2. Greedy algorithms and local search

- A local search algorithm starts with an arbitrary feasible solution to the problem, and checks if some small, local change to the solution results in an improved objective function.
- If so, the change is made.
- When no further change can be made, we have a locally optimal solution, and it is sometimes possible to prove that such locally optimal solutions have value close to that of the optimal solution.

2.1 Scheduling jobs with deadlines on a single machine

- We are given some type of work that must be done, and some resources to do the work, and from this we must create a schedule to complete the work that optimizes some objective:
  - perhaps we want to finish all the work as soon as possible, or
  - perhaps we want to make sure that the average time at which we complete the various pieces of work is as small as possible.
• We schedule \( n \) jobs on a single machine
  – it can process at most one job at a time, and must
    process a job until its completion
  – each job \( j \) must be processed for \( p_j \) units of time
  – the processing of job \( j \) may begin no earlier than
    a specified release date \( r_j, j = 1, \ldots, n \)
• The schedule starts at time 0, and each \( r_j \geq 0 \)
• Each job \( j \) has a specified due date \( d_j \), and if we
  complete its processing at time \( C_j \), then its
  lateness \( L_j = C_j - d_j \)
• We want to schedule the jobs so as to minimize
  the maximum lateness \( L_{\text{max}} = \max_{j=1}^{n} L_j \).

\[ p_1 = 2, \ r_1 = 0, \ p_2 = 1, \ r_2 = 2, \ p_3 = 4, \]
\[ r_3 = 1. \] In this schedule, \( C_1 = 2, \ C_2 = 3, \)
and \( C_3 = 7 \)
This problem is NP-hard, and in fact, even deciding if there is a schedule for which 
\( L_{\text{max}} \leq 0 \) (can all jobs be completed by their due date) is strongly NP-hard

This is a problem that we often encounter in everyday life, and many of us schedule our lives with the following simple greedy heuristic:
– focus on the task with the earliest due date

We will show that in certain circumstances this is a provably good thing to do

This optimization problem is not particularly amenable to obtaining near-optimal solution

We first provide a good lower bound

Let \( S \) denote a subset of jobs, and let 
\[
\begin{align*}
    r(S) &= \min_{j \in S} r_j, \\
    p(S) &= \sum_{j \in S} p_j, \\
    d(S) &= \max_{j \in S} d_j
\end{align*}
\]

Let \( L^*_{\text{max}} \) denote the optimal value

**Lemma 2.1:** For each subset \( S \) of jobs,
\[
L^*_{\text{max}} \geq r(S) + p(S) - d(S)
\]

**Proof:** Consider the optimal schedule as one for \( S \). Let \( j \) be the last job in \( S \) to be processed. Since none of the jobs in \( S \) can be processed before \( r(S) \), and in total they require \( p(S) \) time units, it follows that job \( j \) cannot complete any earlier than time \( r(S) + p(S) \). The due date of job \( j \) is \( d(S) \) or earlier, and so the lateness of job \( j \) in this schedule is at least \( r(S) + p(S) - d(S) \); hence, the claim follows. ■
• A job \( j \) is available at time \( t \) if its release date \( r_j \leq t \)

• We consider the following natural algorithm: at each moment that the machine is idle, start processing next an available job with the earliest due date

• This is the earliest due date (EDD) rule

**Theorem 2.2:** The EDD rule is a 2-approximation algorithm for the problem of minimizing the maximum lateness on a single machine subject to release dates with negative due dates.

**Proof:** Consider the schedule produced by the EDD rule, and let job \( j \) be a job of maximum lateness in this schedule; that is, \( L_{\text{max}} = C_j - d_j \). Focus on the time \( C_j \) in this schedule; find the earliest point in time \( t \leq C_j \) such that the machine was processing without any idle time for the entire period \( [t, C_j) \). Several jobs may be processed in this time interval; we require only that the machine not be idle for some interval of positive length within it. Let \( S \) be the set of jobs that are processed in the interval \( [t, C_j) \). We know that just prior to \( t \), none of these jobs were available (and at least one job in \( S \) is available at time \( t \)); hence, \( r(S) = t \).
Furthermore, since only jobs in $S$ are processed throughout this time interval, $p(S) = C_j - t = C_j - r(S)$. Thus, $C_j \leq r(S) + p(S)$; since $d(S) < 0$, we can apply Lemma 2.1 to get that

$$L_{max}^* \geq r(S) + p(S) - d(S) \geq r(S) + p(S) \geq C_j.$$  

On the other hand, by applying Lemma 2.1 with $S = \{j\}$,

$$L_{max}^* \geq r_j + p_j - d_j \geq -d_j.$$  

Adding the two inequalities, we see that the maximum lateness of the schedule computed is

$$L_{max} = C_j - d_j \leq 2L_{max}^*,$$

which completes the proof of the theorem.  

**2.2 The $k$-center problem**

- Consider a particular variant of clustering, the $k$-center problem
- We are given as input an undirected, complete graph $G = (V, E)$, with a distance $d_{ij} \geq 0$ between each pair of vertices $i, j \in V$
- We assume $d_{ii} = 0$, $d_{ij} = d_{ji}$ for each $i, j \in V$
- The distances obey the triangle inequality: for each triple $i, j, l \in V$, it is the case that

$$d_{ij} + d_{ji} \geq d_{ii}$$
Distances model similarity: vertices that are closer to each other are more similar, whereas those farther apart are less similar.

We are also given a positive integer $k$ as input.

The goal is to find $k$ clusters, grouping together the vertices that are most similar into clusters together.

We will choose a set $S \subseteq V$, $|S| = k$, of $k$ cluster centers.

Each vertex will assign itself to its closest cluster center, grouping the vertices into $k$ different clusters.

The objective is to minimize the maximum distance of a vertex to its cluster center.

Geometrically speaking, the goal is to find the centers of $k$ different balls of the same radius that cover all points so that the radius is as small as possible.

More formally, we define the distance of a vertex $i$ from a set $S \subseteq V$ of vertices to be $d(i,S) = \min_{j \in S} d_{ij}$.

Then the corresponding radius for $S$ is equal to $\max_{i \in V} d(i,S)$, and the goal of the $k$-center problem is to find a set of size $k$ of minimum radius.
We give a greedy 2-approximation algorithm for the $k$-center problem that is simple and intuitive. Our algorithm first picks a vertex $i \in V$ arbitrarily, and puts it in our set $S$ of cluster centers. Then it makes sense for the next cluster center to be as far away as possible from all the other cluster centers. Hence, while $|S| < k$, we repeatedly find a vertex $j \in V$ that determines the current radius (or in other words, for which the distance $d(j, S)$ is maximized) and add it to $S$. Once $|S| = k$, we stop and return $S$.

**Algorithm 2.1:**
A greedy 2-approximation algorithm for the $k$-center problem

1. Pick arbitrary $i \in V$
2. $S \leftarrow \{i\}$
3. while $|S| < k$ do
4. $j \leftarrow \arg\max_{j \in V} d(j, S)$
5. $S \leftarrow S \cup \{j\}$
• $k = 3$ and the distances are given by the Euclidean distances between points.
• The nodes 1, 2, 3 are the nodes selected by the greedy algorithm, whereas the nodes $1^*, 2^*, 3^*$ are the three nodes in an optimal solution.

**Theorem 2.3:** Algorithm 2.1 is a 2-approximation algorithm for the $k$-center problem.

Proof: Let $S^* = \{j_1, \ldots, j_k\}$ be the optimal solution, and let $r^*$ denote its radius. This solution partitions $V$ into clusters $V_1, \ldots, V_k$, where each point $j \in V$ is placed in $V_i$ if it is closest to $j_i$ among all of the points in $S^*$ (ties are broken arbitrarily). Each pair of points $j$ and $j'$ in the same cluster $V_i$ are at most $2r^*$ apart: by the triangle inequality, the distance $d_{jj'}$ between them is $\leq d_{jj_i}$, the distance from $j$ to the center $j_i$, $+ d_{jj_i}'$, the distance from the center $j_i$ to $j'$ (i.e., $d_{jj'} \leq d_{jj_i} + d_{jj_i}'$); since $d_{jj_i}'$ and $d_{jj_i}'$ are each at most $r^*$, we see that $d_{jj_i} \leq 2r^*$. 

$\Box$
Consider \( S \subseteq V \) selected by the greedy algorithm. If one center in \( S \) is selected from each cluster of the optimal solution \( S^* \), then every point in \( V \) is clearly within \( 2r^* \) of some selected point in \( S \).

Suppose then that we select two points within the same cluster. I.e., in some iteration, the algorithm selects \( j \in V_i \), even though it had already selected \( j' \in V_i \) in an earlier iteration. Again, the distance between these two points is \( \leq 2r^* \). We select \( j \) because it is currently the furthest from the points already in \( S \). Hence, all points are within a distance of at most \( 2r^* \) of some center already selected for \( S \). This remains true in subsequent iterations. ■

- This result is the best possible; if there exists a \( \rho \)-approximation algorithm with \( \rho < 2 \), then \( P = NP \).

**Theorem 2.4:** There is no \( \alpha \)-approximation algorithm for the \( k \)-center problem for \( \alpha < 2 \) unless \( P = NP \).