Approximation Algorithms
Load Balancing and k-Center

CMSC 451, Summer 2009

---

Homework 5

- Exercises 11.1, 11.4, 11.10 from the textbook. Additional problem.
- For 11.4: Give an LP and accompanying algorithm to approximately solve this problem. You do not need to prove that the LP is correct, but should give some explanation as to why your algorithm gives a \( b \)-approximation of the optimal solution.
- Due next Monday
Load Balancing: LPT Rule

Longest processing time (LPT). Sort n jobs in descending order of processing time, and then run list scheduling algorithm.

LPT-List-Scheduling\( (m, n, t_1, t_2, \ldots, t_n) \) {
    \textbf{Sort} jobs so that } \text{ } \text{ } t_1 \geq t_2 \geq \ldots \geq t_n \text{.}
    \textbf{for} i = 1 \text{ } \text{ } \text{to} \text{ } m \{ \}
    \quad L_i \leftarrow 0 \text{ } \text{ } \text{load on machine } i \text{.}
    \quad J(i) \leftarrow \emptyset \text{ } \text{ } \text{jobs assigned to machine } i \text{.}
    \}
    \textbf{for} j = 1 \text{ } \text{ } \text{to} \text{ } n \{ \}
    \quad i = \text{argmin}_k \text{} \text{ } \text{L}_k \text{.} \text{ } \text{machine } i \text{ has smallest load}
    \quad J(i) \leftarrow J(i) \cup \{j\} \text{.} \text{ } \text{assign job } j \text{ to machine } i \text{.}
    \quad L_i \leftarrow L_i + t_j \text{.} \text{ } \text{update load of machine } i \text{.}
    \}
    \text{return } J(1), \ldots, J(m) \text{.}
\}

Practice Proof: Load Balancing: LPT Rule

\textbf{Observation.} If at most m jobs, then list-scheduling is optimal.
\textbf{Pf.} Each job put on its own machine. \textbullet

\textbf{Lemma 3.} If there are more than m jobs, \( L^* \geq 2t_{m+1} \). \textbf{Pf.} \texttt{YOU DO.}

\textbf{Theorem.} LPT rule is a 3/2 approximation algorithm.
\textbf{Pf.} Same basic approach as for list scheduling.

\textbf{List Scheduling Proof:}
\textbf{Theorem.} Greedy algorithm is a 2-approximation.
\textbf{Pf.} Consider load \( L_i \) of bottleneck machine \( i \).
\begin{itemize}
    \item Let \( j \) be last job scheduled on machine \( i \).
    \item When job \( j \) assigned to machine \( i \), i had smallest load. Its load before assignment is \( L_i - t_j \) \( \Rightarrow \) \( L_i - t_j \leq L_k \) for all \( 1 \leq k \leq m \).
    \item Sum inequalities over all \( k \) and divide by \( m \) by m:
    \begin{align*}
        L_i - t_j &\leq \frac{1}{m} \sum_k L_k = \frac{1}{m} \sum_k t_k \leq L^* \\
        L_i &\leq \frac{(L_i - t_j)}{L^*} + \frac{t_j}{L^*} \leq 2L^*
    \end{align*}
\end{itemize}
Observation. If at most \( m \) jobs, then list-scheduling is optimal.

\[ \text{Pf. Each job put on its own machine.} \]

Lemma 3. If there are more than \( m \) jobs, \( L^* \geq 2t_{m+1} \).

\[ \text{Pf.} \]
- Consider first \( m+1 \) jobs \( t_1, \ldots, t_{m+1} \).
- Since the \( t_i \)'s are in descending order, each takes at least \( t_{m+1} \) time.
- There are \( m+1 \) jobs and \( m \) machines, so by pigeonhole principle, at least one machine gets two jobs.

Theorem. LPT rule is a \( 3/2 \) approximation algorithm.

\[ \text{Pf. Same basic approach as for list scheduling.} \]

\[ L_j = \frac{(L_j - t_j)}{\sum_j t_j} + \frac{t_j}{\sum_j t_j} \leq \frac{3}{2} L^*. \]

( by observation, can assume number of jobs \( > m \) )

Q. Is our \( 3/2 \) analysis tight?

A. No.

Theorem. [Graham, 1969] LPT rule is a \( 4/3 \)-approximation.

\[ \text{Pf. More sophisticated analysis of same algorithm.} \]

Q. Is Graham's \( 4/3 \) analysis tight?

A. Essentially yes.

Ex: \( m \) machines, \( n = 2m+1 \) jobs, 2 jobs of length \( m+1 \), \( m+2 \), \( \ldots \), \( 2m-1 \) and one job of length \( m \).
11.2 Center Selection

**k-Center Selection Problem**

*Input.* Set of n sites $s_1, ..., s_n$ and integer $k > 0$.

*Center selection problem.* Select $k$ centers $C$ so that maximum distance from a site to nearest center is minimized.
Input. Set of n sites \( s_1, \ldots, s_n \) and integer \( k > 0 \).

Center selection problem. Select \( k \) centers \( C \) so that maximum distance from a site to nearest center is minimized.

Notation.
- \( \text{dist}(x, y) = \) distance between \( x \) and \( y \).
- \( \text{dist}(s_i, C) = \min_{c \in C} \text{dist}(s_i, c) \) = distance from \( s_i \) to closest center.
- \( r(C) = \max_i \text{dist}(s_i, C) \) = smallest covering radius.

Goal. Find set of centers \( C \) that minimizes \( r(C) \), subject to \( |C| = k \).

Distance function properties.
- \( \text{dist}(x, x) = 0 \) \hspace{1cm} \text{(identity)}
- \( \text{dist}(x, y) = \text{dist}(y, x) \) \hspace{1cm} \text{(symmetry)}
- \( \text{dist}(x, y) \leq \text{dist}(x, z) + \text{dist}(z, y) \) \hspace{1cm} \text{(triangle inequality)}
k-Center Selection Problem

Input. Set of n sites \( s_1, ..., s_n \) and integer \( k > 0 \).

Center selection problem. Select \( k \) centers \( C \) so that maximum distance from a site to nearest center is minimized.

Goal. Find set of centers \( C \) that minimizes \( r(C) \), subject to \( |C| = k \).

Distance function properties.
- \( \text{dist}(x, x) = 0 \) (identity)
- \( \text{dist}(x, y) = \text{dist}(y, x) \) (symmetry)
- \( \text{dist}(x, y) \leq \text{dist}(x, z) + \text{dist}(z, y) \) (triangle inequality)

Some distance function possibilities.
- Euclidean distance
- Travel time
- Edit distance

k-Center Selection Example

Ex: each site is a point in the plane, a center can be any point in the plane, \( \text{dist}(x, y) = \) Euclidean distance.

Remark: search can be infinite!
**k-Center Selection: Greedy Algorithm Ideas**

**Greedy selection outline:**
- Repeatedly add a point to the set of centers based on some selection criteria
- Recalculate the distances from each point to its closest center

**Greedy Algorithm: A False Start**

**Greedy algorithm.** Put the first center at the best possible location for a single center, and then keep adding centers so as to reduce the covering radius each time by as much as possible.

**Remark:** arbitrarily bad!
**k-Center Selection: Greedy Algorithm**

**Greedy algorithm [Gonzalez 1985].** Repeatedly choose the next center to be the site *farthest* from any existing center.

```plaintext
Greedy-Center-Selection(k, n, s1, s2, ..., sn) {
    C = ∅
    foreach s_i: dist(s_i, C) = infinity
    repeat k times {
        Select a site s_i with maximum dist(s_i, C)
        Add s_i to C
    }
    foreach s_i: dist(s_i, C) = distance to closest site in C
    return C
}
```

**Observation.** Upon termination all centers in C are pairwise at least r(C) apart.

**Pf.** By construction of algorithm.

---

**k-Center Selection: Analysis of Greedy Algorithm**

**Lemma.** Let s be some site covered by center c* in the optimal solution with optimal radius r, then if s is chosen as a center it covers everything c* covered within radius 2r.

**Pf.** (by picture)
Center Selection: Analysis of Greedy Algorithm

**Theorem.** Let $C^*$ be an optimal set of centers. Then $r(C) \leq 2r(C^*)$.

*Pf. (by contradiction)* Assume $r(C^*) < \frac{1}{2} r(C)$.

- For each site $c_i$ in $C$, consider ball of radius $\frac{1}{2} r(C)$ around it.
- By Lemma, exactly one $c_i^*$ in each ball since otherwise not optimal; let $c_i$ be the site paired with $c_i^*$.
- Consider any site $s$ and its closest center $c_i^*$ in $C^*$.
- $\text{dist}(s, C) \leq \text{dist}(s, c_i) \leq \text{dist}(s, c_i^*) + \text{dist}(c_i^*, c_i) \leq 2r(C^*)$.
- Thus $r(C) \leq 2r(C^*)$. □

---

Center Selection

**Theorem.** Let $C^*$ be an optimal set of centers. Then $r(C) \leq 2r(C^*)$.

**Theorem.** Greedy algorithm is a 2-approximation for center selection problem.

**Remark.** Greedy algorithm always places centers at sites, but is still within a factor of 2 of best solution that is allowed to place centers anywhere. e.g., points in the plane.

**Question.** Is there hope of a 3/2-approximation? 4/3?

**Theorem.** Unless $P = \text{NP}$, there no $\rho$-approximation for center-selection problem for any $\rho < 2$. 
Is our algorithm analysis tight?

Tight example for $k=2$:

Greedy Distance = $n/2$

Optimal Distance = $n/4$