A Note on Terminology

Knuth. [SIGACT News 6, January 1974, p. 12 – 18]

Find an adjective X that sounds good in sentences like,

- EUCLIDEAN-TSP is X.
- It is X to decide whether a given graph has a Hamiltonian cycle.
- It is unknown whether FACTOR is an X problem.

Note: X does not necessarily imply that a problem is in NP, just that every problem in NP polynomial reduces to X.
A Note on Terminology

Knuth’s original suggestions.
- Hard.
- Tough.
- Herculean.
- Formidable.
- Arduous.

Some English word write-ins.
- Impractical.
- Bad.
- Heavy.
- Tricky.
- Intricate.
- Prodigious.
- Difficult.
- Intractable.
- Costly.
- Obdurate.
- Obstinate.
- Exorbitant.
- Interminable.

Hard-boiled. [Ken Steiglitz] In honor of Cook.

Sisyphean. [Bob Floyd] Problem of Sisyphus was time-consuming.

Ulyssen. [Don Knuth] Ulysses was known for his persistence.

Supersat. [Al Meyer] Greater than or equal to satisfiability.

like today’s lecture.

PET. [Shen Lin] Probably exponential time.

GNP. [Al Meyer] Greater than or equal to NP in difficulty.

A Note on Terminology: Consensus

**NP-complete.** A problem in NP such that every problem in NP polynomial reduces to it.

**NP-hard.** A problem such that every problem in NP reduces to it. (may or may not be a yes/no question)

From now on. We will be talking about NP-hard problems. (Usually optimization versions of NP-complete problems.)
Coping With NP-Completeness

Q. Suppose I need to solve an NP-hard problem. What should I do?
A. Theory says you're unlikely to find poly-time algorithm.

Must sacrifice one of three desired features.
- Solve problem in polynomial time.
- Solve arbitrary instances of the problem.
- Solve problem to optimality.

Chapters 10 & 11. Deal with some NP-complete problems that arise in practice.
10.1 Solving Small NP-Hard Problems

In General: We have an arbitrary graph $G$ and arbitrary $k$.

Specific Case: What if we only want to find vertex covers for small constant $k$ (say $k = 2$).

Algorithm: Check all pairs, and see if any two cover all the edges.

Runtime: number of pairs - $O(n \text{ choose } 2) = O(n^2)$
           checking if vertex cover - $O(m)$
           total - $O(mn^2)$

Polynomial for all constant $k$!
10.2 Solving NP-Hard Problems on Trees

**Independent Set on Trees**

**IS on trees:** Given a tree, find a maximum independent set.

**Recall:** an independent set $S$ is a set of nodes such that no two share an edge.

**On a tree:** for every $v$ in $S$, its parent and children are not in $S$.

**Algorithm:** Choose nodes greedily (bottom-up).

**Proof:** (exchange argument)
- Consider a max cardinality independent set $S$.
- If $v \in S$, we’re done.
- If $u \notin S$ and $v \notin S$, then $S \cup \{v\}$ is independent $\Rightarrow S$ not maximum.
- IF $u \in S$ and $v \notin S$, then $S \cup \{v\} - \{u\}$ is independent.
Coping With NP-Completeness

Q. Suppose I need to solve an NP-hard problem. What should I do?
A. Theory says you’re unlikely to find poly-time algorithm.

Must sacrifice one of three desired features.
- Solve problem in polynomial time.
- Solve arbitrary instances of the problem.
- Solve problem to optimality.

Chapters 10 & 11. Deal with some NP-complete problems that arise in practice.
Approximation Algorithms

Q. Suppose I need to solve an NP-hard problem. What should I do?
A. Theory says you’re unlikely to find poly-time algorithm.

Must sacrifice one of three desired features.
- Solve problem in polynomial time.
- Solve arbitrary instances of the problem.
- Solve problem to optimality.

α-approximation algorithm.
- Guaranteed to run in poly-time.
- Guaranteed to solve arbitrary instance of the problem
- Guaranteed to find solution within ratio \( \alpha \) of true optimum.

Challenge. Need to prove a solution’s value is close to optimum, without even knowing what optimum value is!

11.4 Vertex Cover
Small Vertex Covers

**NP-hard**: Optimization version of Vertex-Cover

**Greedy Approach**:

```plaintext
Vertex-Cover-Approx (V, E) {
    let S = emptyset
    while some edge uv in E
        add u and v to S
        remove edges adjacent to u or v from E
    return S
}
```

Runtime: \(O(m)\)

Optimal?

No: Opt = 1, Alg = 2

---

**Vertex Cover: Approximation Analysis**

**Theorem.** Greedy algorithm is a 2-approximation.
- Need to compare our answer with some optimal cover \(S^*\) and show our solution is no more than twice as big.

**Lemma 1.** The algorithm finds a vertex cover.
**Proof.** by contradiction.
**Vertex Cover: Approximation Analysis**

**Theorem.** Greedy algorithm is a 2-approximation.
- Need to compare our answer with some optimal cover $S^*$ and show our solution is no more than twice as big.

**Lemma 2.** The vertex cover is no more than twice the size of an optimal vertex cover.

**Proof.**
- All edges have at least one endpoint in $S^*$.
- Let $M$ be the set of edges we choose: all the endpoints are distinct.
- Thus, any $S$ (including $S^*$) must contain at least one (distinct) endpoint for each.
- Hence, $\text{Alg} = |S_{\text{alg}}| \leq 2|M| \leq 2|S^*| = 2 \text{Opt}.$

This analysis is tight: i.e. the algorithm can get this bad.

---

**Vertex-Cover: In-Class Exercise**

**Recall:** We showed that $\text{Independent-Set} \leq_p \text{Vertex-Cover}$

Does this mean we have an approximation for Independent-Set?

If so, what is the approximation factor?

If not, give the worst example you can find.
Chapter 11
Approximation Algorithms

11.1 Load Balancing
Load Balancing

**Input.** m identical machines; n jobs, job j has processing time $t_j$.
- Job j must run contiguously on one machine.
- A machine can process at most one job at a time.

**Def.** Let $J(i)$ be the subset of jobs assigned to machine i. The **load** of machine i is $L_i = \sum_{j \in J(i)} t_j$.

**Def.** The **makespan** is the maximum load on any machine $L = \max_i L_i$.

**Load balancing.** Assign each job to a machine to minimize makespan.

---

Is Load-Balancing NP-Hard?

**Yes.** Reduce from Subset Sum
- create 2 machines, and a job time $s_i$ for each element of size $s_i$.
- let $S$ be the sum of all the sizes, and create a job time $S - 2W$.
- the makespan is $\leq S - W$ iff there is a subset which sums to $W$. 

---
Load Balancing: List Scheduling

Greedy algorithm.
- Consider \( n \) jobs in some fixed order.
- Assign job \( j \) to machine whose load is smallest so far.

\[
\text{List-Scheduling}(m, n, t_1, t_2, \ldots, t_n) \{
\text{for } i = 1 \text{ to } m \{ \\
\quad L_i \leftarrow 0 \quad \text{load on machine } i \\
\quad J(i) \leftarrow \emptyset \quad \text{jobs assigned to machine } i \\
\}
\text{for } j = 1 \text{ to } n \{ \\
\quad i = \arg\min_k L_k \quad \text{machine } i \text{ has smallest load} \\
\quad J(i) \leftarrow J(i) \cup \{j\} \quad \text{assign job } j \text{ to machine } i \\
\quad L_i \leftarrow L_i + t_j \quad \text{update load of machine } i \\
\}
\text{return } J(1), \ldots, J(m)
\}
\]

Runtime. \( O(nm) \)
Optimal?
Load Balancing: List Scheduling

Machine 1
Machine 2
Machine 3

Time

0

A
B
C
D
E
F
G
H
I
J

Load Balancing: List Scheduling

Machine 1
Machine 2
Machine 3

Time

0

A
B
C
D
E
F
G
H
I
J
Load Balancing: List Scheduling

<table>
<thead>
<tr>
<th>Time</th>
<th>Machine 1</th>
<th>Machine 2</th>
<th>Machine 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>30</td>
<td>D</td>
<td>E</td>
<td></td>
</tr>
</tbody>
</table>

Load Balancing: List Scheduling

<table>
<thead>
<tr>
<th>Time</th>
<th>Machine 1</th>
<th>Machine 2</th>
<th>Machine 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>30</td>
<td>D</td>
<td>E</td>
<td></td>
</tr>
</tbody>
</table>
Load Balancing: List Scheduling

Machine 1: A, E
Machine 2: B, D
Machine 3: C, F, G

Tasks: A, B, C, D, E, F, G, H, I, J

Time progression:
- Time 0: A, B, C, D, E
- Time 1: A, B, C, D, E, F
- Time 2: A, B, C, D, E, F, G

Scheduling order: A, B, C, D, E, F, G

Load balance: Machines 1, 2, 3 are utilized efficiently.
Load Balancing: List Scheduling

Machine 1

A | E | Machine 1
B | D | G
C | F|

Time

Load Balancing: List Scheduling

I | J

A | E | H | I
B | D | G
C | F|

Time
Load Balancing: List Scheduling

Machine 1
Machine 2
Machine 3

A
B
C
D
E
F
G
H
I
J

Time

Load Balancing: List Scheduling

Machine 1
Machine 2
Machine 3

A
B
C
D
E
F
G
H
I
J

Time
Load Balancing: List Scheduling

- First worst-case analysis of an approximation algorithm.
- Need to compare resulting solution with optimal makespan $L^*$.

Lemma 1. $L^* \geq \max_j t_j$.
Proof. Some machine must process the most time-consuming job.

Lemma 2. $L^* \geq (1/m) \sum_j t_j$.
Proof.
- The total processing time is $\sum_j t_j$.
- Some machine must do at least $1/m$ fraction of the total work.
Theorem. Greedy algorithm is a 2-approximation.

Pf. Consider load $L_i$ of bottleneck machine $i$.

- Let $j$ be last job scheduled on machine $i$.
- 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 $1 \leq k \leq m$.

\[ L_i \leq L^* \leq 2L^* \]

Lemma 1
Load Balancing: In-Class Exercise

Q. Is our analysis tight?

i.e. is there an instance of the load balancing problem where our algorithm's load is exactly twice optimum?

Load Balancing: List Scheduling Analysis

Q. Is our analysis tight?
A. Essentially yes.

Ex: 10 machines, 90 length 1 jobs, 1 job of length 10

\[ m = 10 \]

List scheduling makespan = 19
Load Balancing: List Scheduling Analysis

Q. Is our analysis tight?
A. Essentially yes.

Ex: 10 machines, 90 length 1 jobs, 1 job of length 10

$\text{Alg} = 2m - 1$
$\text{Opt} = m$

Is arbitrarily close to 2!

Load Balancing: List Scheduling Analysis

Q. Is our analysis tight?
A. Essentially yes.

Ex: $m$ machines, $m(m-1)$ length 1 jobs, 1 job of length $m$

In General: Alg = $2m - 1$
Opt = $m$

Is arbitrarily close to 2!
Load Balancing: List Scheduling Analysis

Q. Is our analysis tight?
A. Essentially yes.

Can we find a better approximation algorithm?

Load Balancing: Longest Processing Time (LPT)

**Improved Greedy Algorithm.** Sort $n$ jobs in descending order of processing time, and then run list scheduling algorithm.

```plaintext
LPT-List-Scheduling(m, n, t_1, t_2, \ldots, t_n) {
    Sort jobs so that $t_1 \geq t_2 \geq \ldots \geq t_n$

    for $i = 1$ to $m$ {
        $L_i \leftarrow 0$ \quad \text{load on machine } i
        $J(i) \leftarrow \emptyset$ \quad \text{jobs assigned to machine } i
    }

    for $j = 1$ to $n$ {
        $i = \text{argmin}_k L_k$ \quad \text{machine } i \text{ has smallest load}
        $J(i) \leftarrow J(i) \cup \{j\}$ \quad \text{assign job } j \text{ to machine } i
        $L_i \leftarrow L_i + t_j$ \quad \text{update load of machine } i
    }

    return $J(1), \ldots, J(m)$
}
```
Load Balancing: LPT Rule

Observation. If at most $m$ jobs, then list-scheduling is optimal.
Pf. Each job put on its own machine. •

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

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

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

\[\text{Lemma 3}\]

Q. Is our 3/2 analysis tight?
A. No. (but how would you prove this?)

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 $m+1$, $m+2$, ..., $2m-1$ and one job of length $m$. 
11.2 Center Selection
Center Selection Problem

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

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)
Center Selection Problem

**Input.** Set of n sites \( s_1, \ldots, s_n \) in some space \( S \), and integer \( k > 0 \).

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

**Is Center-Selection NP-Hard?**

**Yes.** Reduce from Vertex Cover.
- let the sites be the set of vertices, and let \( k \) be the max cover size
- let \( dist(u,v) = \) length of shortest path from \( u \) to \( v \).
- solve the center selection problem
- \( r(C) \leq 1 \) if and only if there is a vertex cover of size \( k \)

Center Selection Example

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

**Remark:** search can be infinite!
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!

Center Selection: Greedy Algorithm

Greedy algorithm. Repeatedly choose the next center to be the site farthest from any existing center.

```plaintext
Greedy-Center-Selection(k, n, s_1, s_2, ..., s_n) {
    C = ∅
    repeat k times {
        Select a site s_i with maximum dist(s_i, C)
        site farthest from any center
        Add s_i to C
    }
    return C
}
```

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

Pf. By construction of algorithm.
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.
- Exactly one $c_i^*$ in each ball; 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 $r(C) \leq 2r(C^*)$. □

Finally, more diagrams.