Decision Problems

Given Some Universal Set X, Let R be a subset of X. The decision problem for R is: Given an arbitrary element a of X, does a belong to R? Note: X is usually assumed to be a set of strings, but this can be interpreted loosely.
The class P

Let R be a set of strings. If there exists a Polynomial time algorithm: $O(n)$, $O(n^2)$, ... that solves the decision problem for R, then R is in the class P.

Note the use of the Big O notation.

Sorting, $O(n \lg n)$, is in P.

Binary Search, $O(\lg n)$, is in P.
What is Nondeterminism?

This is a deterministic Finite State Machine. Every state has exactly two output arcs, one labeled A and one labeled B.

This machine can be implemented easily, but may be difficult to design.
A Nondeterministic Machine

This machine is *nondeterministic*

There may be two output arcs with the same label.

There may be no output arc for some inputs.

This machine may be easy to draw, but it cannot be implemented.
Three types of State Machines

Simple: No Extra Storage

PDM: An auxiliary Stack

Turing Machine: An auxiliary Read/Write Tape

Anything that can be written in a conventional programming language can be implemented as a Turing Machine.
Deterministic Conversions

- Any Non-Deterministic FSM (no aux. storage) Can be converted to a deterministic machine in quickly. (All FSMs run in O(n) time.)
- Non-Deterministic PDM’s are more powerful than deterministic PDM’s. All PDM’s run in O(n) time, but converting from Non-Det. PDM to a real algorithm might produce an O(n³) algorithm.
TM Deterministic Conversion

- Deterministic and Non-Deterministic TM’s are equally Powerful.
- Any Non-Deterministic TM can be converted to a deterministic TM
- The conversion may cause an exponential slow-down in running time.
  (We don’t know if this is necessary, but no one has proven that it isn’t.)
Non-Deterministic Algorithms

- Working with Turing Machines is too difficult to be practical.
- Since TM’s and programming languages are equivalent, TM’s are always represented as HLL programs.
- NonDeterminism is introduced using the statement: \( V := \text{SELECT}(A,B,C,...); \)
The SELECT Statement

- The SELECT statement cannot be implemented.
- SELECT is equivalent to assigning a CONSTANT to a variable.
- SELECT represents several potential assignment statements that COULD be coded in a deterministic program.
Accepting and Rejecting

- Since we are only concerned with decision problems, we insist that a program accept a string by executing a special ACCEPT statement.
- Deterministic programs must execute a REJECT statement to reject a string.
- A Deterministic program must execute either ACCEPT or REJECT for any string.
NonDeterministic Acceptance

- A NonDeterministic algorithm accepts a string, if it is possible to replace each execution of the SELECT function with a constant assignment, so that the ACCEPT statement will be executed.
- The transformation is permitted to take the specific input into account. (And usually must do so.)
- NonDeterministic algorithms never contain REJECT statements. (Acceptance is based on program transformation, not just on program execution.)
Why NonDeterminism?

- NP is the set of problems that can be solved in Polynomial time by nondeterministic algorithms.
- Many interesting problems are easy to formulate as polynomial time nondeterministic algorithms.
- No known polynomial time algorithms exist for these problems.
- In general we don’t know if P=NP.
Completeness

- Given a class of problems \( K \), (remember that \( K \) must be a set of sets of strings)
- A Problem \( R \) is \( K \)-Hard, if a solution to \( R \) would allow us to solve every problem in \( K \).
- A problem \( R \) is \( K \)-Complete, if it is \( K \)-Hard, and a member of \( K \).
A problem $R$ is NP complete if:

- $R$ is in NP (i.e. there exists a nondeterministic polynomial time algorithm that recognizes the elements of $R$)
- $R$ is NP-Hard (i.e. finding a deterministic polynomial-time algorithm that recognizes $R$, would allow us to recognize any problem in NP in polynomial time.)
To Show NP-Completeness

- To show that R is NP-Complete:
- First construct a Nondeterministic Polynomial time algorithm for R.
- Then show that if $X \in \text{NP}$ then $X$ can be transformed into an instance of R in polynomial time.
The “Easy” Way

- To prove the NP-Hardness of R:
  - Select a known NP-Complete Problem K.
  - Construct a transformation T that will convert any instance of K into an instance of R in polynomial time.
  - We must show that for any string s:
    - if s is in K then T(s) is in R
    - if s is not in K then T(s) is not in R
    - T runs in polynomial time
Cook’s Theorem (Outline)

Given a string $S$ and a Non-Deterministic Turing machine $M$, create a CNF expression $E(S,M)$ which is satisfiable if and only if $M$ accepts $S$ in polynomial time.

Demonstrate an algorithm for generating $E(S,M)$ for any pair $S,M$.

Prove that the algorithm for generating $E(S,M)$ runs in polynomial time.
SAT is in NP

\[
\text{SAT}(e) \\
\quad k = \text{the number of distinct variables in } e; \\
\quad \text{Allocate a boolean array } v \text{ of size } k; \\
\quad \text{for } i = 1 \text{ to } k \text{ do} \\
\quad \quad v[i] := \text{select}\{\text{TRUE, FALSE}\}; \\
\quad \text{endfor} \\
\quad \text{evaluate } e \text{ on } v \text{ and assign the result to } R; \\
\quad \text{if } (R = \text{TRUE}) \text{ then} \\
\quad \quad \text{accept;} \\
\quad \text{endif} \\
\text{end SAT}
\]
Some Basic Problems 1

- **3-SAT (3-Satisfiability)**
  
  Given a CNF boolean expression $C = c[1] \& c[2] \& \ldots \& c[m]$ such that every clause $c[i]$ has exactly 3 literals, is $C$ satisfiable?

- **3DM (3-Dimensional Matching)**
  
  Given $W, X,$ and $Y$, three sets, each with $Q$ elements, and a set $M \subseteq W \times X \times Y$, is there a subset $M_1 \subseteq M$ such that $|M_1| = Q$ and no two elements of $M_1$ agree in any coordinate?
Some Basic Problems 2

- **VC (Vertex Cover)**
  Given a graph $G=(V,E)$ and a positive integer $K \leq |V|$, is there a set $V' \subseteq V$ such that $|V'| < K$ and for each $\{u,v\} \in E$, at least one of $u$ or $v$ is in $V'$?

- **CLIQUE**
  Given a graph $G=(V,E)$ and a positive integer $J \leq |V|$, does $G$ have a subgraph which is a complete graph with $J$ vertices?
Some Basic Problems 3

- **HC (Hamiltonian Circuit)**
  Given a graph $G=(V,E)$ is there a simple cycle in $G$ that contains all vertices of $G$.

- **PARTITION**
  Given a set of positive integers $A$, is there a subset $A' \in A$ such that the sum of the elements of $A'$ is exactly half the sum of the elements of $A$?
Some Basic Problems 4

- **X3C (Exact cover by 3-Sets)**
  
  Give a finite set $X$ with $|X|=3q$ for some integer $q$, and a collection $C$ of 3-element sets of $X$, is there a set $C' \subseteq C$ such that every element of $X$ occurs in exactly one element of $C'$

- **MINIMUM COVER**
  
  Given a collection $C$ of subsets of a set $S$, and a positive integer $K$, is there a set $C' \subseteq C$ such that $|C'| \leq K$ and every element of $S$ is contained in at least one element of $C'$. 
Some Basic Problems 5

- **HITTING SET**
  Given a collection $C$ of subsets of a set $S$ and a positive integer $K$, is there a set $S' \subseteq S$ such that $|S'| \leq K$ and $S'$ contains at least one element from every set in $C$?

- **SUBGRAPH ISOMORPHISM**
  Given two graphs $G=(V,E)$ and $H=(V',H')$, does $G$ contain an exact copy of $H$ as a subgraph?
Some Basic Problems 6

- **BOUNDED DEGREE SPANNING TREE**
  Given a graph $G=(V,E)$ and an integer $J \leq |V| - 1$, is there a spanning tree $T=(V,E')$ of $G$ such that no vertex has degree more than $K$ in $T$?

- **MINIMUM EQUIVALENT DIGRAPH**
  Given a directed graph $G=(V,A)$, and a positive integer $K \leq |A|$, is there a directed graph $G'=(V,A')$ such that $A' \subseteq A$, $|A'| \leq K$, and there is a path from $u$ to $v$ in $G'$ if and only if there is a path from $u$ to $v$ in $G$?
Some Basic Problems 7

**KNAPSACK**

Given a finite set $U$ such that every element $u \in U$ has a size $s(u)$ and a value $v(u)$, both of which are positive integers, and given two positive integers $B$ and $K$, is there a subset $U'$ of $U$ such that the total size of the elements of $U'$ is less than or equal to $B$ and the total value for the elements of $U'$ is greater than or equal to $K$?
MULTIPROCESSOR SCHEDULING

Given a set $A$ of tasks, such that each $a \in A$ has a length $l(a)$ which is a positive integer, and given a number of processors $m$, and a deadline $D$, both of which are positive integers, is there a partition of $A$ into disjoint subsets $A = A_1 \cup A_2 \cup \ldots \cup A_m$ such that for any subset $A_i$, the total length of all tasks in $A_i$ is less than or equal to $D$?
3-SAT (from CNF SAT) - 1

if \( c[j] = A \) \( \implies (A \lor S[j,1] \lor S[j,2]) \land (A \lor \overline{S[j,1]} \lor S[j,2]) \land (A \lor S[j,1] \lor \overline{S[j,2]}) \land (A \lor \overline{S[j,1]} \lor \overline{S[j,2]}) \)

if \( c[j] = (A \lor B) \) \( \implies (A \lor B \lor S[j,1]) \land (A \lor B \lor \overline{S[j,1]}) \)

if \( c[j] = (A \lor B \lor C) \) \( \implies (A \lor B \lor C) \)
3-SAT - 2

Example Only
4 or more is similar

\[
\text{if } c[j] = (A \lor B \lor C \lor D \lor E \lor F) \Rightarrow \\
(A \lor B \lor S[j,1]) \land \\
(S[j,1] \lor C \lor S[j,2]) \land \\
(S[j,2] \lor D \lor S[j,3]) \land \\
(S[j,3] \lor D \lor F)
\]
3-Sat: Proof

- Left as an exercise
- For each of the four different transformations, show that the generated set of clauses can be set to TRUE if and only if the original clause can be set to TRUE
3D Match (from SAT)

- One pair per clause
- One structure per Variable

Modeling TRUE and FALSE
3DM: Notes

- One Star is constructed for each variable.
- There are 2 points for each clause.
- A different set of $a_x$ and $b_x$ variables are used for each star.
- To form a complete matching AT LEAST one triangle must be selected from each star.
To cover all the $a_x$ and $b_x$ variables, it is necessary to select every other point.

Either the $u_x$ or the $\overline{u}_x$ points must be selected. *All* of one and *none* of the other.

This models a variable being TRUE or FALSE.
Clause 1: \((U \lor V \lor W)\)

Satisfaction Tester New Triple Specification
Satisfaction is modeled by selecting all $T_x$ and $S_x$ variables.

If a 3-CNF expression is satisfiable, there must be (at least) one true literal in every clause.

A truth assignment can be modeled by selecting the star points that correspond to the FALSE literals.
3DM Notes

- If the original expression is satisfiable, enough points will be left over to cover all $T_x$ and $S_x$ variables.
- If the original expression is not satisfiable, there will be some pair of $T_x$ and $S_x$ variables that cannot be selected, because all the required star points will be used up.
Satisfying the Formula
Now, What’s Left?

- There are \( m \) variables and \( n \) clauses
- There are \( m \) “stars” and \( n \) “propellers”
- Each star has \( 2n \) points, \((2n \times m) \) total.
- Half of the points are used up by the truth setting. \((\text{Leaving } n \times m)\)
- One blade on each propeller is used up by satisfaction. This uses up \( n \) points. \((\text{Leaving } (n-1) \times m)\)
There must be one stack for each unused star point.

$2n \times m \times m \times (n-1)$ stacks.

There must be one blade for each point in each star.

$2n \times m$ blades in each stack.

Elements
Vertex Cover

- A Vertex Cover of a Graph G=(V,E) is a set V’⊆V such that for every edge (a,b)∈E, either a∈V’ or b∈V’.
- That is, V’ contains at least one endpoint of every edge.
- Optimization: Find the smallest vertex cover of G.
- Decision: Does G have a vertex cover of size K?
Vertex Cover: Relations

- Independent Set of $G=(V,E)$: $V' \subseteq V$ such that if $u \in V'$ and $v \in V'$, then $\{u,v\} \notin E$.

- INDEPENDENT SET PROBLEM: Given $G=(V,E)$ and $J$ an integer, is there an independent set $V'$ of $G$ such that $|V'| \geq J$?

- Relations:
  - $V'$ is a vertex cover for $G$ iff $V-V'$ is an independent set for $G$.
  - $V'$ is an independent set for $G$ iff $V'$ is a clique in the complement of $G$. 
The complement of $G$

Complete Graph on $N$ Vertices

Complement of $G$

Delete All Edges from $G$
Vertex Cover

- Transformation from 3-Sat.
- Transform Each variable into a pair of vertices labeled with the variable and its complement.
- Transform each clause into a ring of 3 vertices labeled with the literals.
- Connect identically labeled vertices with edges. (See Next Slide.)
Vertex Cover

N Variables
M Clauses
K=2M+N

One Pair Per Variable
Join Clause to Literal
One Per Clause
Vertices Labeled with Literals from Clause
Vertex Cover: Proof

- Structurally, every vertex cover of the transformed graph must have at least $2M+N$ Vertices, choose $N$ vertices from the top, one from each pair, and two from each triangle on the bottom.

- Every choice of $N$ vertices from the top corresponds to a truth assignment for the original expression, and vice versa.
Vertex Cover: Proof

- Suppose the original expression is satisfiable.
- Choose N vertices from the top corresponding with the satisfying assignment.
- There must be one true literal in each clause. Identify such, and choose the two other vertices from each ring at the bottom.
Vertex Cover: Proof

- The only issue is coverage of the edges between top and bottom.
- There is exactly one such edge attached to each bottom vertex.
- For each bottom triple, the chosen vertices cover the top-to-bottom edges.
- Because the unchosen vertex corresponds to a true literal, the other end of the edge has been chosen for the truth assignment.
Vertex Cover: Proof

- Suppose the original expression is not satisfiable.
- Attempt to form a vertex cover by choosing one vertex from each top pair and two vertices from each bottom ring. (This is necessary.)
- The choice of top vertices corresponds to a truth assignment for the expression.
Vertex Cover: Proof

- Because the original expression is not satisfiable, the truth assignment must produce one clause whose literals are all false.
- Examine the corresponding triple. (red vertices are chosen.)
- Neither end-point of the edge attached to the unchosen vertex has been chosen.
Vertex Cover: Proof

- From the previous, we conclude that if the original expression is not satisfiable, then every vertex cover must have at least $2M+N+1$ vertices.
Hamiltonian Circuit

Transformation from Vertex Cover

Map each edge to a “RR-Tracks” Structure, and identify the sides with the vertices touched by the edge.
Hamiltonian Circuit

The Vertex Cover contains V but not U.

The Vertex Cover contains U but not V.

The Vertex Cover contains both U and V.
Hamiltonian Circuit

Join all the U-Sides together into a loop, (and all the W-Sides ...)
Let the ends dangle for the moment.
Modeling the Integer $K$

Replicate each dangling edge $K$ times

Attach one dangling edge to each of the new vertices.

Create $K$ new vertices
Hamitonian Circuit: Proof

- Suppose the original graph has a vertex cover $V'$ of size $K$.
- Start with Vertex $A_1$, and choose a vertex $v$ in $V'$.
- Traverse the path corresponding to $v$.
- When traversing an RR-Tracks structure, follow the double-Z path if the other vertex is not in $V'$, otherwise go straight through.
Hamiltonian Circuit: Proof

- After finishing the traverse of the v path, go to vertex A₂.
- Choose another vertex w of V’, and traverse the path for w.
- Continue until all vertices of V’ have been exhausted. Then return to A₁.
Hamiltonian Circuit: Proof

- Because $V'$ is a vertex cover, we must have traversed at least one edge of every RR-Tracks structure.
- For those where we would not traverse the other side directly, we took the double-Z path to get those vertices.
- The result is a Hamiltonian Circuit.
Hamiltonian Circuit: Proof

- Suppose the transformed graph has a Hamiltonian circuit. Since we can begin anywhere, we shall begin on $A_1$.
- Leaving $A_1$, we have no choice but to begin a path corresponding to some vertex $v$.
- We must begin and end on the path for $v$. 
Hamiltonian Circuit: Proof

- We must traverse exactly K paths.
- Every path corresponds to a vertex.
- We cannot traverse a vertex path more than once.
- We must visit every RR-Tracks structure.
- Every Hamiltonian circuit corresponds to the selection of K vertices from the original graph.
Hamiltonian Circuit: Proof

- This selection of vertices must be a vertex cover, because one side of every RR-Tracks structure is traversed, and because every edge corresponds to a RR-Tracks structure.
Hamiltonian Path

- Transformation from vertex cover is identical.
- Break $A_1$ into two vertices $A_{1a}$ and $A_{1b}$.
- For every edge $(A_1, v)$, create two new edges $(A_{1a}, v)$ and $(A_{1b}, v)$.
- Create two new vertices $S$, and $E$.
- Add an edge between $S$ and $A_{1a}$, and an edge between $E$ and $A_{1b}$. 
Traveling Salesman

- Given a complete graph G with weighted edges, What is the Hamiltonian Cycle of least weight? (Every permutation of the vertices is a Hamiltonian Cycle.)
- Decision Problem: Does G have a Hamiltonian Cycle of weight K?
Traveling Salesman

- Conversion from Hamiltonian Cycle.
- Given an arbitrary graph G, assign the weight 1 to each edge.
- Add additional edges to G making a complete graph.
- Assign the weight 2 to each new edge.
- Set $K=n$ where $n$ is the number of vertices in G.
Partition

- Partition is the key to a number of numeric problems.
- An instance of Partition is a set of numbers A.
- The question is “Is it possible to divide A into two disjoint sets A=B ∪ C such that the sum of the elements of B is equal to the sum of the elements of C?”
Partition: Proof

- Start with 3DM
- Given Four Sets:
  \[ W = \{w_1, w_2, \ldots, w_n\} \]
  \[ X = \{x_1, x_2, \ldots, x_n\} \]
  \[ Y = \{y_1, y_2, \ldots, y_n\} \]
  \[ M = \{m_1, m_2, \ldots, m_k\} \subseteq W \times X \times Y \]

We must construct a set of numbers from these four sets
Binary Number Format

Segment: Sufficient bits to hold the number $k (= \text{size of } M)$

One Segment For Each Element Of $W$

One Segment For Each Element Of $X$

One Segment For Each Element Of $Y$
Transforming M

- We add one segmented number in A for each ordered triple in M.
- If \((w_i, x_j, y_h) \in M\) then we set the three segments corresponding to \(w_i\), \(x_j\), and \(y_h\) equal to 1.
- All other segments are set equal to 0.
- We use \(a_x\) to denote the number associated with \(m_x \in M\)
Transforming M 2

\[ m_x = (w_i, x_j, y_h) \]

\[ = a_x \]
The Other Numbers

- Let B be the segmented number that has each segment set to 1.
- Let C be the sum of all segmented numbers that were created by transforming elements of M.
- Let $P = 2C - B$ and let $Q = C + B$
- We add P and Q to A (but not C or B)
The Other Numbers 2

Note: $C$ has the value of at most $k$ in each segment.
Verification

- The total of all numbers in A is
  \[ C + P + Q = C + 2C - B + C + B = 4C \]
- If A has a partition, each set must add up to 2C
- If A has a partition, then P and Q must be in different sets. \( P + Q = 3C \)
- A has a partition if and only if there is a subset A’ of A whose elements sum to B.
Verification 2

- Consider the set containing $P=2C-B$. To reach the target size of $2C$, we must add elements totaling $B$ to this set.
- Suppose $A$ has such a set $A'$. Let $M'$ be the subset of $M$ (in 3DM) that corresponds to $A'$. $M'$ is a complete matching for $M$. 
Verification 3

- If any element of W, X, or Y were missing, a segment of the sum of A’ would be zero.
- If any element of W, X, or Y appears twice in M’ then the sum of A’ would not have a 1 in the position corresponding to that element. (Segments cannot overflow into one another.)
Verification 4

- If M has a complete matching M’ then the subset A’ of A corresponding to M’ has the sum B.
- Each element of W, X, and Y appears exactly once in M’, so each segment of the sum must equal one.
From 3-Sat

For each clause, \{A,B,C\}, convert it into the following graph.

Note: A, B and C can’t all be the same color.

A, B, and C are the “Literal Vertices”

This is the “Clause Component”
Bin Packing

- Input: a set of objects $B$ along with a set of associated sizes, $S$, such that every $b_i \in B$ there is a size $s_i \in S$. (Sizes not unique)
- For all $s_i \in S$, $0 \leq s_i \leq 1$.
- Minimization problem: What is the minimum number of bins of size 1 that will hold all elements?
Bin Packing

- Decision Problem: Will all objects fit in K bins?
- Transformation from partition.
- Given A, let X be the sum of all elements of A.
- Multiply each element by 2/X, and add to S.
- Ask the question, will the elements of S fit in 2 bins?
Subset Sum

- Given a set of numbers $S$ (with possible duplicates) and an integer $K$, is there a subset of $S$ whose sum is equal to $K$?
- Optimization problem: What is the subset of $S$ with the maximum sum not exceeding $K$?
- Transformation from partition. Use the same base set. Let $X$ be sum of all elements of $A$. $K=X/2$. 
Knapsack

- Given a set of objects \( C = \{x_1, x_2, \ldots, x_n\} \) with an associated set of sizes \( \{s_1, s_2, \ldots, s_n\} \) and an associated set of values \( \{v_1, v_2, \ldots, v_n\} \), and two numbers \( k \) and \( m \) is there a subset \( A \subseteq C \) such that the sum of the sizes of the elements of \( A \) is less than or equal to \( k \), and the sum of the values of the elements of \( A \) is greater than or equal to \( m \)?
Knapsack

- From Partition:
  - Let the objects be the numbers from the partition problem. Set both the size and the value of the number to be equal to its value.
  - Set $m=k=\text{half the total size of all elements.}$
3-Colorability 2

- Create the following graph segment
- Each variable appears in both complemented and uncomplemented form.
3-Colorability 3

- The graph segment on the previous slide is the truth-setting component.
- The color assigned to the T vertex will represent True, the color assigned to the F vertex will represent False, and the color assigned to the U vertex will represent “other.”
3-Colorability 4

- Go back to the Clause Components, and connect each Literal Vertex to the T vertex of the Truth-Setting component.
- If a Literal Vertex corresponds the variable x, then connect the literal vertex to the x vertex of the Truth-Setting Component.
- If it represents x’, then connect it to the x’ vertex.
3-Colorability: Proof

- The resultant graph is 3-Colorable, if and only if the original expression is satisfiable.
- Assign colors in the truth setting component to be consistent with the truth assignment.
- Because the assignment is satisfying, at least one literal in each clause must be assigned the “True” color.
3-Colorability: Proof 2

- Each literal vertex has two neighbors, one of which has the “True” color, the other of which may have either the “True” color or the “False” color.
- Since the Truth-Setting component is colored consistently with a satisfying assignment, each clause component will have a Literal Vertex with two “True” colored neighbors.
3-Colorability: Proof 3

- Use the “False” color to color the vertex with two “True” neighbors.
- Complete the coloring as follows, (Red=False, Blue=True, Yellow=Other)
Now assume the graph is 3-colorable.

No Literal Vertex can be colored “True.”

In a Clause component, it is impossible to color all Literal Vertices “Other.”
3-Colorability: Proof 5

- A least one vertex in every Clause Component must be colored “False”. (The corresponding Vertex In Truth-Setting Component is colored “True”)
- Every coloring of the Truth-Setting Component corresponds to a truth-assignment of the original expression.
- A three coloring corresponds to a satisfying assignment.
Exercises (Easy) - 1

- **LONGEST PATH**
  Given a graph $G=(V,E)$, and a positive integer $K \leq |V|$, does $G$ contain a simple path with $K$ or more edges?

- **SET PACKING**
  Given a collection $C$ of finite sets, and a positive integer $K \leq |C|$, Does $C$ contain $K$ disjoint sets?

- **Partition Into Hamiltonian Subgraphs**
  Given a graph $G=(V,E)$ and a positive integer $K \leq |V|$, can the vertices of $G$ be partitioned into $k \leq K$ disjoint sets $V_1, ..., V_k$ such that $1 \leq i \leq k$, the subgraph induced by $V_i$ contains a Hamiltonian circuit?
Exercises (Easy) - 2

- **Largest Common Subgraph**
  Given two graphs $G_1=(V_1, E_1)$, and $G_2=(V_2, E_2)$, and a positive integer $K$, do there exist subsets $E_1' \subseteq E_1$ and $E_2' \subseteq E_2$ such that $|E_1'|=|E_2'| \geq K$ and such that the two subgraphs $G_1'=(V_1, E_1')$ and $G_2'=(V_2, E_2')$ are isomorphic?

- **Minimum Sum of Squares**
  Given a finite set $A$, and an integer size $s(a)$ for all $a \in A$ and positive integers $K$ and $J$, can the elements of $A$ be partitioned into $K$ disjoint sets $A_1$ through $A_K$, such that
  \[
  \sum_{i=1}^{k} \left( \sum_{a \in A_i} s(a) \right)^2 \leq J
  \]
Exercises (Medium) - 1

- Feedback Vertex Set
  Given a directed graph $G=(V,E)$, and a positive integer $K \leq |V|$ is there a subset $V' \subseteq V$ such that $|V'| \leq K$ and $V'$ contains a vertex from every directed cycle in $G$?

- Exact Cover by 4-Sets
  Given a finite set $X$, with $|X|=4q$, $q$ an integer, and a collection $C$ of 4-element subsets of $X$, is there a subcollection $C' \subseteq C$ such that every element of $X$ occurs in exactly one element of $C'$?

- Dominating Set
  Given a graph $G=(V,E)$, and a positive integer $K \leq |V|$, is there a subset $V' \subseteq V$, such that $|V'| \leq K$, and every vertex $v \in V-V'$ is joined to one element of $V'$ by an edge in $E$?
Exercises (Medium) - 2

- **Steiner Trees in Graphs**
  Given a graph $G=(V,E)$ and a subset $R \subseteq V$, and a positive integer $K \leq |V|-1$ is there a subtree of $G$ that contains all vertices of $R$, and no more than $K$ edges?

- **Star-Free Regular Expression Equivalence**
  Given two star-free regular expressions $E_1$ and $E_2$, do $E_1$ and $E_2$ represent different sets of strings?
Exercises (Hard)

- Set Splitting (3-Sat)
  Given a collection $C$ of subsets of a finite set $S$, is there a partition of $S$ into two subsets $S_1$ and $S_2$ such that no element of $C$ is completely contained in either $S_1$ or $S_2$?

- Partition into Paths of Length 2 (3DM)
  Given a graph $G=(V,E)$ with $|V|=3q$, for some positive integer $q$, is there a partition of $V$ into $q$ disjoint subsets $V_1, V_2, \ldots, V_q$, of three elements each, such that for each $V_i=\{u,v,w\}$ at least two of the edges $\{u,v\}, \{v,w\},$ and $\{u,w\}$ are contained in $E$?

- Graph Grundy Numbering (3-Sat)
  Given a directed graph $G=(V,E)$, is there a way to label the vertices with positive integers (duplicates are allowed), such that for each $v\in V$, the label on $v$ is the least non-negative integer which is not in the set of labels assigned to the successors of $v$?
Approximation Theory

- Used For Optimization Problems
- Feasible solution: A not-necessarily optimal solution to the problem
  - A valid, but not necessarily minimal graph coloring
  - A bin-packing into some number of bins, not necessarily minimal
Approximation Theory

Given a problem $P$ and an input $I$, $\text{opt}(I)$ is the size of the optimal solution, sometimes denoted $\text{opt}_P(I)$.

- The minimum number of colors needed to color a graph
- The minimum number of bins needed to hold a set of elements
Approximation Theory

- Given an approximation algorithm \( A \), and an Input \( I \), \( A(I) \) is the approximate solution, and \( \text{Size}(A(I)) \) is its size.
- The quality ratio of a solution \( A(I) \) is defined as follows:

\[
 r_A(I) = \begin{cases} 
 \frac{\text{size}(A(I))}{\text{opt}(I)} & \text{Minimization} \\
 \frac{\text{opt}(I)}{\text{size}(A(I))} & \text{Maximization}
\end{cases}
\]
The quality measures of an approximation algorithm are:

\[ R_A(m) = \text{lub}\{r_A(I) \mid \forall I \text{ with } \text{opt}(I) = m\} \]

\[ S_A(m) = \text{lub}\{r_A(I) \mid \forall I \text{ with size } m\} \]

Replace Least Upper Bound with Maximum For finite sets.
Approximation Theory

- \( R_A(m) \) is a measure of how close to the optimal value one can get, regardless of input size.
- \( R_A(m) \) is infinite for some problems.
- \( S_A(m) \) is a measure of how close to the optimal value one can get, taking input size into account.
- \( S_A(m) \) is finite.
Approximation Theory

- $R_A = \text{lub} \{R_A(m) \mid m > 0\}$
- $S_A = \text{lub} \{S_A(m) \mid m > 0\}$
- For some bin-packing approximations, $R_A \leq 4/3$
- For graph coloring, approximation quality depends on graph size. For existing algorithms, there are families of 3-colorable graphs that require an arbitrarily large number of colors. $R_A$ is infinite.
Approximations

- Bin Packing
- Subset Sum
- Vertex Cover
- Graph Coloring
- Euclidean Traveling Salesman
- General Traveling Salesman
BIN PACKING Approximation

- **Real-Time First Fit:**
  - Add elements to Bin 1.
  - When Bin 1 is full go to Bin 2, and so forth.
  - Never go back to a previous bin.

- **First Fit**
  - Try each element in each bin, starting with Bin 1.
  - Add element to new bin if it won’t fit in any existing bin.
  - Elements are not sorted in any way.
BIN PACKING Approximation

- Non-Increasing First-Fit (Niff)
  - Sort elements into descending (non-decreasing) order
  - Then, same as First-Fit
- Niff is a good approximation
  - $R_A$ is finite, and small
  - Niff Runs quickly
Bin Packing Approximation

- In the approximation produced by Niff, there are $X \geq \text{Opt}(I)$ bins. The $X - \text{Opt}(I)$ bins are extra.
- The first element placed in an extra bin must be of size $\leq 1/3$.
- Suppose this were not the case. Because elements are placed in descending order, all placed objects must have size $> 1/3$. 

Bin Packing Approximation

- No bin can have more than two objects, because if one did, its total size would exceed 1.
- Some bins must have two objects, because if all have just 1, the extra-bin object would have to be placed with one of these objects in the optimal solution, but the algorithm tried to do this and it didn’t fit.
Bin Packing Approximation

- If some bins have only one object, they must precede the bins with two objects, because the algorithm tried to fit the extra-bin object into all of the 1-object bins, and it didn’t fit. Therefore none of the 2-object-bin objects will fit either, because they must be the same size or larger than the extra bin object. Since they are smaller than the 1-bin objects, they must have been placed later.
Bin Packing Approximation

- Assume there are $k \leq \text{Opt}(I)$ 2-object bins.
- The $2k$ objects in these bins plus the object placed in the extra bin must fit in $k$ bins in the optimal solution.
- Since there are $2k+1$ objects, at least one bin must have three objects.
- Since all objects have size $> 1/3$, this bin must have size $> 1$ which is impossible.
Bin Packing Approximation

- The number of objects placed in extra bins must be less than \( \text{Opt}(I) \).
- Suppose that \( \text{Opt}(I) \) objects are placed in extra bins. Denote these objects as \( e_1, e_2, \ldots, e_{\text{Opt}(I)} \).
- Object \( e_i \) will not fit in bin \( i \). The algorithm tried to put it there, and it wouldn’t fit.
Bin Packing Approximation

Let the total size of all objects in bin $i$ be designated as $B_i$.

Because object $e_i$ won’t fit in bin $i$, the following two inequalities must be true.

$$e_i + B_i > 1$$

$$\sum_{i=1}^{Opt(I)} (e_i + B_i) > Opt(I)$$
Bin Packing Approximation

- However, because Opt(I) is the size of the optimal solution, the total size of all objects must be less than or equal to Opt(I)
- Taken together, the total number of extra bins cannot exceed Opt(I)/3
- \( R_{\text{Niff}} \leq \frac{4}{3} \)
- The above computation assumes that Opt(I) is a multiple of 3. Exercise: consider the other two cases using Opt(I)-1 instead of Opt(I).
Bin Packing Approximation

- The largest difference occurs when the optimum is 2 bins, but the algorithm uses 3.
- \(0.5, 0.4, 0.3, 0.3, 0.3, 0.2\)

Niff Solution

Optimal Solution
Bin Packing Approximation

- $S_{\text{Niff}} \leq 3/2$
- Exercise: Find a family of sets of objects with arbitrarily large sets, such that the optimal bin packing has 2 bins, but Niff uses 3 bins.
- Solution: $S_1 = \{0.5, 0.4, 0.3, 0.3, 0.3, 0.15, 0.05\}$
- $S_k = S_{k-1}$ but divide the smallest element in half. $S_2 = \{0.5, 0.4, 0.3, 0.3, 0.3, 0.15, 0.025, 0.025\}$
- $S_3 = \{0.5, 0.4, 0.3, 0.3, 0.3, 0.15, 0.025, 0.0125, 0.0125\}$
Subset Sum Approximation

- Subset Sum: given a set of $n$ objects of sizes $s_1$ through $s_n$, and an integer $K > 0$, find the subset with the largest total size not exceeding $C$.

- Greedy algorithm: consider objects in order 1-$n$. Add each object $s_i$ to the set unless the object would make the total exceed $C$. If the object $s_i$ does cause the limit to be exceeded, but $s_i$ is larger than the current total, throw everything away, and put $s_i$ in the set. (and continue)
Better Greedy Method: for every subset, $S$, of objects containing at most $k$ objects, where $k$ is a constant, start the greedy algorithm with the elements of $S$ already selected.

This is a family of approximation algorithms, one algorithm for each $k$.

Denote these algorithms $A_k$.

$A_k$ is of order $n^{k+1}$ and gives an approximation with a quality ratio of $1+1/k$ or smaller.
Subset Sum Approximation

- Because we start the greedy method with all subsets of size \( k \), we must start with the set that contains the \( k \) largest items in the optimal solution.
- There must be at least one element \( e_x \) of the optimal solution that is not in the approximate solution.
Subset Sum Approximation

- The element $e_x$ is not one of the $k$ largest elements of the optimal solution, therefore its size must be less than or equal to $\text{Opt}(I)/(k+1)$.
- The algorithm attempted to include $e_x$ in the solution, but it wouldn’t fit.
- The amount of slack ($\text{slack} = C$ minus Solution size) must be less than $\text{Opt}(I)/(k+1)$.
- Since $\text{Opt}(I) < C$, the difference between $\text{Opt}(I)$ and the approximate solution must be less than $\text{Opt}(I)/(k+1)$ as well.
Subset Sum Approximation

\[ R_{A_k} (Opt(I)) = \frac{Opt(I)}{Opt(I) - Opt(I) / (k + 1)} \]

\[ = \frac{1}{1 - 1/(k+1)} = \frac{1}{((k+1) - 1) / (k+1)} \]

\[ = \frac{k + 1}{k} = \frac{k}{k} + \frac{1}{k} = 1 + \frac{1}{k} \]
Subset Sum Approximation

- For each subset, the algorithm does $\Theta(n)$ work, looking at each of $n-k$ elements using constant time for each.
- There are $\Theta(n^k)$ subsets of size $k$.
- Each subset can be generated in $\Theta(k)=\Theta(1)$ time.
- (Note that $k$ is a constant.)
Subset Sum Approximation

- Assume all element sizes are stored in a 1-based array.
- Use an array of size k to generate the subset.
- Initialize the array as follows:

  1 2 ... k
Subset Sum Approximation

- Each element of the array has a limiting value. These values are illustrated below.

\[
\begin{array}{cccc}
\text{n-k} & \text{n-k+1} & \cdots & \text{n} \\
\end{array}
\]
Subset Sum Approximation

- To generate a new set, increment the $k^{th}$ element of the array.
- If the $k^{th}$ element exceeds its limiting value, go to the $k-1^{st}$ element and increment that.
- Continue until we encounter an element that does not exceed its limiting value after incrementing.
Subset Sum Approximation

- Suppose the $i^{th}$ element was incremented to the value $x$.
- Now move forward through the array, setting each value to one larger than the previous. The $(i+1)^{st}$ element is set to $x+1$, the $(i+2)^{nd}$ is set to $x+2$, etc.
- If all elements exceed their limiting values, we have generated all subsets, so stop.
Subset Sum Approximation

- Subsets of size 3 from a set of 6 elements:
  - \{1,2,3\} \ {1,2,4\} \ {1,2,5\} \ {1,2,6\} \ {1,3,4\}
  - \{1,3,5\} \ {1,3,6\} \ {1,4,5\} \ {1,4,6\} \ {1,5,6\}
  - \{2,3,4\} \ {2,3,5\} \ {2,3,6\} \ {2,4,5\} \ {2,4,6\}
  - \{2,5,6\} \ {3,4,5\} \ {3,4,6\} \ {3,5,6\} \ {4,5,6\}
Vertex Cover Approximation

- Create a matching set by starting with the empty set $M$.
- Choose an arbitrary edge $e$ from $G$.
- Add $e$ to our matching set $M$.
- Delete $e$ and the vertices incident to it from $G$.
- Repeat the previous 3 steps until $G$ has no edges.
Vertex Cover Approximation

- The vertices incident on the edges of $M$ form a vertex cover $V'$.
- $V'$ is no larger than twice the minimal cover.
- One endpoint of each edge in $M$ must be in every vertex cover, so it is not possible to delete more than $|M|/2$ vertices from $V'$ and still have it cover all vertices.
Graph Coloring Approximation

- Given $G=(V,E)$ with $n$ vertices.
- Use the integers $\{1, 2, 3, \ldots, n\}$ to represent colors.
- Start by assigning 0 to every vertex.
- Process the vertices one at a time.
- For each vertex, $V_i$, start by coloring $V_i$ with the color 1.
Graph Coloring Approximation

- Check the neighbors of $V_i$ to see if any is colored 1. If not then go to the next vertex, $V_{i+1}$.
- If there is a neighbor colored 1, recolor $V_i$ with color 2, and repeat the neighbor search.
- Repeat the previous step incrementing the color until we find a color $c$ that has not been used to color any of $V_i$’s neighbors.
Graph Coloring Approximation

- This algorithm is called Sequential Graph coloring, or SC.
- Let $K$ be the maximum degree of any vertex in $G$. Then SC uses no more than $K+1$ colors.
- Proof: The color-assignment and testing procedure will test no more than $K+1$ colors. The procedure always starts with 1 and increments.
Graph Coloring Approximation

- There are bipartite (2-colorable) graphs for which SC uses an arbitrarily large number of colors.

K vertices on the top, K on the bottom

Processing order of \(a_1, b_1, a_2, b_2, \ldots, a_k, b_k\) uses \(k\) colors.

Every vertex on the top connected to every vertex on the bottom except the one directly below it.
Graph Coloring Approximation

- Approximate Graph Coloring is hard
- Suppose we have an approximation algorithm which is guaranteed to produce a coloring with less than 4/3 the optimal number of colors.
- This algorithms colors 3-colorable graphs with $n < 3 \times 4/3 = 4$ colors. I.E., 3 colors. Four-colored and higher graphs need 4 colors.
- Thus the approximation algorithm gives us a way to solve the 3-colorability problem in polynomial time.
Graph Coloring Approximation

- Even if the approximation works only for graphs that require a large number of colors, the result is the same.
- Suppose the graph works only for graphs that require \( k \) or more colors.
- (The minimum number of colors needed to color a graph is called its Chromatic Number, and is designated \( \chi(G) \))
Graph Coloring Approximation

- Graph Composition: Given G and H, replace every vertex of G with a copy of H.
- Denote the replacement of vertex v as $H_v$.
- If (v,w) is an edge in G, connect every vertex of $H_v$ to every vertex of $H_w$. 
Graph Coloring Approximation

G

H

Composition
Graph Coloring Approximation

- If we have a $<4/3$ optimal graph coloring algorithm that works for graphs with chromatic numbers of $k$ or larger, compose the original graph with a complete graph on $k$ vertices.
- If the original chromatic number of $G$ was $\chi(G)$, the new graph has chromatic number $k\chi(G)$. 
Graph Coloring Approximation

- If the original graph was three-colorable, the approximation algorithm will use less than $4/3 \times 3k = 4k$ colors.
- If the original graph requires more than three colors, then the approximation algorithm must use at least $4k$ colors to color it. (Chromatic number is at least $4k$)
Graph Coloring Approximation

- Suppose we have an approximation algorithm that guarantees to use no more than $M^* \chi(G)$ colors, $M$ a constant.
- If we compose a graph with itself, the new chromatic number is $\chi(G)^2$. If we do it twice, the new chromatic number is $\chi(G)^3$. 
Graph Coloring Approximation

- For every constant $M$, there is a constant $K$ such that $3^K < M4^K$.
- Thus we can use an approximation with an $M\chi(G)$ guarantee to solve the 3-colorability problem in polynomial time by composing a graph with itself $K$ times.
- (The composition is huge, but polynomial in size.)
Traveling Salesman Approx.

- Assume that the triangle inequality holds. In other words, $w(a,b)+w(b,c) \leq w(a,c)$
- Obtain the minimum spanning tree of the complete weighted graph.
- The weight of the minimum spanning tree must be less than the weight of the minimum Hamiltonian Path.
Traveling Salesman Approx.

- Form a non-simple cycle by traversing the MST. When a leaf is encountered, reverse direction and go back. This cycle will have weight twice that of the MST.
- Convert the MST to a simple cycle by shortcutting vertices.
- The result will have no more than twice the weight of the minimum Hamiltonian path.
Traveling Salesman Approx.
Traveling Salesman Approx.

- The general problem is much harder to approximate.
- Suppose we have an approximation that is guaranteed to find a Hamiltonian cycle with less than $K$ times the minimum weight.
- We can use this algorithm to solve the general Hamiltonian cycle problem in polynomial time.
Traveling Salesman Approx.

- Given an arbitrary graph G, assign a weight of 1 to each edge.
- Add all other edges to G to make it a complete graph.
- Assign a weight of n*K+1 to each new edge.
- If the original graph has a Hamiltonian cycle, the approximation algorithm must find it, otherwise the weight of the found cycle would be at least n*K+1, more than K optimal.
Decision Problems

Given Some Universal Set X,
Let R be a subset of X.
The decision problem for R is:
Given an arbitrary element a of X, does a belong to R?
Note: X is usually assumed to be a set of strings, but this can be interpreted loosely.
The class P

Let R be a set of strings. If there exists a Polynomial time algorithm: $O(n), O(n^2), ...$ that solves the decision problem for R, then R is in the class P.

Note the use of the Big O notation. Sorting, $O(n \log n)$, is in P. Binary Search, $O(\log n)$, is in P.
What is Nondeterminism?

This is a deterministic Finite State Machine.

Every state has exactly two output arcs, one labeled A and one labeled B.

This machine can be implemented easily, but may be difficult to design.
A Nondeterministic Machine

This machine is *nondeterministic*

There may be two output arcs with the same label.

There may be no output arc for some inputs.

This machine may be easy to draw, but it cannot be implemented.
Three types of State Machines

Simple: No Extra Storage

PDM: An auxiliary Stack

Turing Machine: An auxiliary Read/Write Tape

Anything that can be written in a conventional programming language can be implemented as a Turing Machine.
Deterministic Conversions

- Any Non-Deterministic FSM (no aux. storage) Can be converted to a deterministic machine in quickly. (All FSMs run in O(n) time.)
- Non-Deterministic PDM’s are more powerful than deterministic PDM’s. All PDM’s run in O(n) time, but converting from Non-Det. PDM to a real algorithm might produce an O(n³) algorithm.
TM Deterministic Conversion

- Deterministic and Non-Deterministic TM’s are equally Powerful.
- Any Non-Deterministic TM can be converted to a deterministic TM
- The conversion may cause an exponential slow-down in running time. (We don’t know if this is necessary, but no one has proven that it isn’t.)
Non-Deterministic Algorithms

- Working with Turing Machines is too difficult to be practical.
- Since TM’s and programming languages are equivalent, TM’s are always represented as HLL programs.
- NonDeterminism is introduced using the statement: \( V := \text{SELECT}(A,B,C,...); \)
The SELECT Statement

- The SELECT statement cannot be implemented.
- SELECT is equivalent to assigning a CONSTANT to a variable.
- SELECT represents several potential assignment statements that COULD be coded in a deterministic program.
Accepting and Rejecting

- Since we are only concerned with decision problems, we insist that a program accept a string by executing a special ACCEPT statement.
- Deterministic programs must execute a REJECT statement to reject a string.
- A Deterministic program must execute either ACCEPT or REJECT for any string.
NonDeterministic Acceptance

- A NonDeterminisic algorithm accepts a string, if it is possible to replace each execution of the SELECT function with a constant assignment, so that the ACCEPT statement will be executed.
- The transformation is permitted to take the specific input into account. (And usually must do so.)
- NonDeterministic algorithms never contain REJECT statements. (Acceptance is based on program transformation, not just on program execution.)
Why NonDeterminism?

- NP is the set of problems that can be solved in Polynomial time by nondeterministic algorithms.
- Many interesting problems are easy to formulate as polynomial time nondeterministic algorithms.
- No known polynomial time algorithms exist for these problems.
- In general we don’t know if P=NP.
Completeness

- Given a class of problems $K$, (remember that $K$ must be a set of sets of strings)
- A Problem $R$ is $K$-Hard, if a solution to $R$ would allow us to solve every problem in $K$.
- A problem $R$ is $K$-Complete, if it is $K$-Hard, and a member of $K$. 
NP-Completeness

- A problem $R$ is NP complete if:
  - $R$ is in NP (i.e. there exists a nondeterministic polynomial time algorithm that recognizes the elements of $R$)
  - $R$ is NP-Hard (i.e. finding a deterministic polynomial-time algorithm that recognizes $R$, would allow us to recognize any problem in NP in polynomial time.)
To Show NP-Completeness

- To show that R is NP-Complete:
  - First construct a Nondeterministic Polynomial time algorithm for R.
  - Then show that if $X \in \text{NP}$ then $X$ can be transformed into an instance of R in polynomial time.
The “Easy” Way

- To prove the NP-Hardness of R:
  - Select a known NP-Complete Problem K.
  - Construct a transformation $T$ that will convert any instance of K into an instance of R in polynomial time.
- We must show that for any string $s$:
  - if $s$ is in K then $T(s)$ is in R
  - if $s$ is not in K then $T(s)$ is not in R
- $T$ runs in polynomial time
Cook’s Theorem (Outline)

Given a string $S$ and a Non-Deterministic Turing machine $M$, create a CNF expression $E(S,M)$ which is satisfiable if and only if $M$ accepts $S$ in polynomial time.

Demonstrate an algorithm for generating $E(S,M)$ for any pair $S,M$.

Prove that the algorithm for generating $E(S,M)$ runs in polynomial time.
SAT is in NP

SAT(e)

\[ k = \text{the number of distinct variables in } e; \]
Allocate a boolean array \( v \) of size \( k \);
for \( i=1 \) to \( k \) do
    \[ v[i] := \text{select}\{\text{TRUE, FALSE}\}; \]
endfor
evaluate \( e \) on \( v \) and assign the result to \( R \);
if \( (R=\text{TRUE}) \) then
    accept;
endif
end SAT
Some Basic Problems 1

✧ 3-SAT (3-Satisfiability)
Given a CNF boolean expression $C = c[1] \& c[2] \& \ldots \& c[m]$ such that every clause $c[i]$ has exactly 3 literals, is $C$ satisfiable?

✧ 3DM (3-Dimensional Matching)
Given $W$, $X$, and $Y$, three sets, each with $Q$ elements, and a set $M \subseteq W \times X \times Y$, is there a subset $M_1 \subseteq M$ such that $|M_1| = Q$ and no two elements of $M_1$ agree in any coordinate?
Some Basic Problems 2

- **VC (Vertex Cover)**
  Given a graph $G=(V,E)$ and a positive integer $K \leq |V|$, is there a set $V' \subseteq V$ such that $|V'| < K$ and for each $\{u,v\} \in E$, at least one of $u$ or $v$ is in $V'$?

- **CLIQUE**
  Given a graph $G=(V,E)$ and a positive integer $J \leq |V|$, does $G$ have a subgraph which is a complete graph with $J$ vertices?
Some Basic Problems 3

✧ HC (Hamiltonian Circuit)
   Given a graph $G=(V,E)$ is there a simple cycle in $G$ that contains all vertices of $G$.

✧ PARTITION
   Given a set of positive integers $A$, is there a subset $A' \in A$ such that the sum of the elements of $A'$ is exactly half the sum of the elements of $A$?
Some Basic Problems 4

✧ X3C (Exact cover by 3-Sets)
Give a finite set \(X\) with \(|X|=3q\) for some integer \(q\), and a collection \(C\) of 3-element sets of \(X\), is there a set \(C' \subseteq C\) such that every element of \(X\) occurs in exactly one element of \(C'\)

✧ MINIMUM COVER
Given a collection \(C\) of subsets of a set \(S\), and a positive integer \(K\), is there a set \(C' \subseteq C\) such that \(|C'| \leq K\) and every element of \(S\) is contained in at least one element of \(C'\).
Some Basic Problems 5

- **HITTING SET**
  Given a collection $C$ of subsets of a set $S$ and a positive integer $K$, is there a set $S' \subseteq S$ such that $|S'| \leq K$ and $S'$ contains at least one element from every set in $C$?

- **SUBGRAPH ISOMORPHISM**
  Given two graphs $G=(V,E)$ and $H=(V',H')$, does $G$ contain an exact copy of $H$ as a subgraph?
Some Basic Problems 6

✧ **BOUNDED DEGREE SPANNING TREE**

Given a graph $G=(V,E)$ and an integer $J \leq |V|-1$, is there a spanning tree $T=(V,E')$ of $G$ such that no vertex has degree more than $K$ in $T$?

✧ **MINIMUM EQUIVALENT DIGRAPH**

Given a directed graph $G=(V,A)$, and a positive integer $K \leq |A|$, is there a directed graph $G'=(V,A')$ such that $A' \subseteq A$, $|A'| \leq K$, and there is a path from $u$ to $v$ in $G'$ if and only if there is a path from $u$ to $v$ in $G$?
Some Basic Problems 7

KNAPSACK

Given a finite set $U$ such that every element $u \in U$ has a size $s(u)$ and a value $v(u)$, both of which are positive integers, and given two positive integers $B$ and $K$, is there a subset $U'$ of $U$ such that the total size of the elements of $U'$ is less than or equal to $B$ and the total value for the elements of $U'$ is greater than or equal to $K$?
MULTIPROCESSOR SCHEDULING

Given a set $A$ of tasks, such that each $a \in A$ has a length $l(a)$ which is a positive integer, and given a number of processors $m$, and a deadline $D$, both of which are positive integers, is there a partition of $A$ into disjoint subsets $A=A_1 \cup A_2 \cup ... \cup A_m$ such that for any subset $A_i$, the total length of all tasks in $A_i$ is less than or equal to $D$?
3-SAT (from CNF SAT) - 1

if \( c[j] = A \) \( \Rightarrow \) \((A \lor S[j,1] \lor S[j,2]) \land (A \lor \overline{S[j,1]} \lor S[j,2]) \land (A \lor S[j,1] \lor \overline{S[j,2]}) \land (A \lor S[j,1] \lor S[j,2])\)

if \( c[j] = (A \lor B) \) \( \Rightarrow \) \((A \lor B \lor S[j,1]) \land (A \lor B \lor \overline{S[j,1]})\)

if \( c[j] = (A \lor B \lor C) \) \( \Rightarrow \) \((A \lor B \lor C)\)
if \( c[j] = (A \lor B \lor C \lor D \lor E \lor F) \Rightarrow (A \lor B \lor S[j,1]) \land (S[j,1] \lor C \lor S[j,2]) \land (S[j,2] \lor D \lor S[j,3]) \land (S[j,3] \lor D \lor F) \)
3-Sat: Proof

- Left as an exercise
- For each of the four different transformations, show that the generated set of clauses can be set to TRUE if and only if the original clause can be set to TRUE
3D Match (from SAT)

One structure per Variable

One pair per clause

Modeling TRUE and FALSE
3DM: Notes

- One Star is constructed for each variable.
- There are 2 points for each clause.
- A different set of $a_x$ and $b_x$ variables are used for each star.
- To form a complete matching AT LEAST one triangle must be selected from each star.
3DM Notes

- To cover all the $a_x$ and $b_x$ variables, it is necessary to select every other point.
- Either the $u_x$ or the $\overline{u}_x$ points must be selected. *All* of one and *none* of the other.
- This models a variable being TRUE or FALSE.
3D Match

Clause 1: $(U \lor V \lor \overline{W})$

Satisfaction Tester

New Triple Specification
Satisfaction is modeled by selecting all $T_x$ and $S_x$ variables.

If a 3-CNF expression is satisfiable, there must be (at least) one true literal in every clause.

A truth assignment can be modeled by selecting the star points that correspond to the FALSE literals.
3DM Notes

- If the original expression is satisfiable, enough points will be left over to cover all $T_x$ and $S_x$ variables.
- If the original expression is not satisfiable, there will be some pair of $T_x$ and $S_x$ variables that cannot be selected, because all the required star points will be used up.
Satisfying the Formula
Now, What’s Left?

- There are $m$ variables and $n$ clauses
- There are $m$ “stars” and $n$ “propellers”
- Each star has $2n$ points, $(2n \times m$ total).
- Half of the points are used up by the truth setting. (Leaving $n \times m$)
- One blade on each propeller is used up by satisfaction. This uses up $n$ points. (Leaving $(n-1) \times m$)
There much be one blade for each point in each star.

2n×m blades in each stack.

There must be one stack for each unused star point.

2n×m×m×(n-1) stacks.

Garbage Collection

Elements
Vertex Cover

- A Vertex Cover of a Graph $G=(V,E)$ is a set $V' \subseteq V$ such that for every edge $(a,b) \in E$, either $a \in V'$ or $b \in V'$.
- That is, $V'$ contains at least one endpoint of every edge.
- Optimization: Find the smallest vertex cover of $G$.
- Decision: Does $G$ have a vertex cover of size $K$?
Vertex Cover: Relations

- Independent Set of $G=(V,E)$: $V' \subseteq V$ such that if $u \in V'$ and $v \in V'$, then $\{u,v\} \notin E$.
- INDEPENDENT SET PROBLEM: Given $G=(V,E)$ and $J$ an integer, is there an independent set $V'$ of $G$ such that $|V'| \geq J$?
- Relations:
  - $V'$ is a vertex cover for $G$ iff $V-V'$ is an independent set for $G$.
  - $V'$ is an independent set for $G$ iff $V'$ is a clique in the complement of $G$. 
The complement of $G$

Complete Graph on N Vertices

Delete All Edges from $G$

Complement of $G$
Vertex Cover

- Transformation from 3-Sat.
- Transform Each variable into a pair of vertices labeled with the variable and its complement.
- Transform each clause into a ring of 3 vertices labeled with the literals.
- Connect identically labeled vertices with edges. (See Next Slide.)
Vertex Cover

N Variables
M Clauses
K=2M+N

One Pair Per Variable

One Per Clause

Vertices Labeled with Literals from Clause

Join Clause to Literal
Vertex Cover: Proof

- Structurally, every vertex cover of the transformed graph must have at least 2M+N Vertices, choose N vertices from the top, one from each pair, and two from each triangle on the bottom.
- Every choice of N vertices from the top corresponds to a truth assignment for the original expression, and vice versa.
Vertex Cover: Proof

- Suppose the original expression is satisfiable.
- Choose N vertices from the top corresponding with the satisfying assignment.
- There must be one true literal in each clause. Identify such, and choose the two other vertices from each ring at the bottom.
Vertex Cover: Proof

- The only issue is coverage of the edges between top and bottom.
- There is exactly one such edge attached to each bottom vertex.
- For each bottom triple, the chosen vertices cover the top-to-bottom edges.
- Because the unchosen vertex corresponds to a true literal, the other end of the edge has been chosen for the truth assignment.
Vertex Cover: Proof

- Suppose the original expression is not satisfiable.
- Attempt to form a vertex cover by choosing one vertex from each top pair and two vertices from each bottom ring. (This is necessary.)
- The choice of top vertices corresponds to a truth assignment for the expression.
Vertex Cover: Proof

- Because the original expression is not satisfiable, the truth assignment must produce one clause whose literals are all false.
- Examine the corresponding triple. (red vertices are chosen.)
- Neither end-point of the edge attached to the unchosen vertex has been chosen.
Vertex Cover: Proof

- From the previous, we conclude that if the original expression is not satisfiable, then every vertex cover must have at least $2M+N+1$ vertices.
Hamiltonian Circuit

Transformation from Vertex Cover

Map each edge to a “RR-Tracks” Structure, and identify the sides with the vertices touched by the edge.

+ K an Integer
Hamiltonian Circuit

The U-Side

The Vertex Cover contains V but not U.

The V-Side

The Vertex Cover contains U but not V.

The U-Side

The Vertex Cover contains both U and V.

The V-Side
Hamiltonian Circuit

Join all the U-Sides together into a loop, (and all the W-Sides ...) Let the ends dangle for the moment.
Modeling the Integer $K$

Replicate each dangling edge $K$ times

Attach one dangling edge to each of the new vertices.

Create $K$ new vertices

The V-Side

The U-Side

$A_1$, $A_2$, $A_3$, ..., $A_K$
Hamiltonian Circuit: Proof

- Suppose the original graph has a vertex cover $V'$ of size $K$.
- Start with Vertex $A_1$, and choose a vertex $v$ in $V'$.
- Traverse the path corresponding to $v$.
- When traversing an RR-Tracks structure, follow the double-Z path if the other vertex is not in $V'$, otherwise go straight through.
Hamiltonian Circuit: Proof

- After finishing the traverse of the v path, go to vertex $A_2$.
- Choose another vertex w of $V'$, and traverse the path for w.
- Continue until all vertices of $V'$ have been exhausted. Then return to $A_1$. 
Hamiltonian Circuit: Proof

- Because V’ is a vertex cover, we must have traversed at least one edge of every RR-Tracks structure.
- For those where we would not traverse the other side directly, we took the double-Z path to get those vertices.
- The result is a Hamiltonian Circuit.
Hamiltonian Circuit: Proof

- Suppose the transformed graph has a Hamiltonian circuit. Since we can begin anywhere, we shall begin on $A_1$.
- Leaving $A_1$, we have no choice but to begin a path corresponding to some vertex $v$.
- We must begin and end on the path for $v$. 
Hamiltonian Circuit: Proof

- We must traverse exactly $K$ paths.
- Every path corresponds to a vertex.
- We cannot traverse a vertex path more than once.
- We must visit every RR-Tracks structure.
- Every Hamiltonian circuit corresponds to the selection of $K$ vertices from the original graph.
Hamiltonian Circuit: Proof

- This selection of vertices must be a vertex cover, because one side of every RR-Tracks structure is traversed, and because every edge corresponds to a RR-Tracks structure.
Hamiltonian Path

- Transformation from vertex cover is identical.
- Break $A_1$ into two vertices $A_{1a}$ and $A_{1b}$.
- For every edge $(A_1, v)$, create two new edges $(A_{1a}, v)$ and $(A_{1b}, v)$.
- Create two new vertices $S$, and $E$.
- Add an edge between $S$ and $A_{1a}$, and an edge between $E$ and $A_{1b}$. 
Traveling Salesman

- Given a complete graph $G$ with weighted edges, What is the Hamiltonian Cycle of least weight? (Every permutation of the vertices is a Hamiltonian Cycle.)
- Decision Problem: Does $G$ have a Hamiltonian Cycle of weight $K$?
Traveling Salesman

- Conversion from Hamiltonian Cycle.
- Given an arbitrary graph $G$, assign the weight 1 to each edge.
- Add additional edges to $G$ making a complete graph.
- Assign the weight 2 to each new edge.
- Set $K=n$ where $n$ is the number of vertices in $G$. 
Partition

- Partition is the key to a number of numeric problems.
- An instance of Partition is a set of numbers $A$.
- The question is "Is it possible to divide $A$ into two disjoint sets $A = B \cup C$ such that the sum of the elements of $B$ is equal to the sum of the elements of $C"
Partition: Proof

- Start with 3DM
- Given Four Sets:
  \[ W = \{ w_1, w_2, \ldots, w_n \} \]
  \[ X = \{ x_1, x_2, \ldots, x_n \} \]
  \[ Y = \{ y_1, y_2, \ldots, y_n \} \]
  \[ M = \{ m_1, m_2, \ldots, m_k \} \subseteq W \times X \times Y \]

We must construct a set of numbers from these four sets.
Binary Number Format

Segment: Sufficient bits to hold the number \( k(=\text{size of M}) \)

One Segment For Each Element Of W
One Segment For Each Element Of X
One Segment For Each Element Of Y
Transforming M

- We add one segmented number in A for each ordered triple in M.
- If \((w_i, x_j, y_h) \in M\) then we set the three segments corresponding to \(w_i\), \(x_j\), and \(y_h\) equal to 1.
- All other segments are set equal to 0.
- We use \(a_x\) to denote the number associated with \(m_x \in M\)
Transforming $M_2$

$m_x = (w_i, x_j, y_h)$

$0 0 \ldots 1 \ldots 0 0 \ldots 1 \ldots 0 0 \ldots 1 \ldots$

$= a_x$
The Other Numbers

- Let $B$ be the segmented number that has each segment set to 1.
- Let $C$ be the sum of all segmented numbers that were created by transforming elements of $M$.
- Let $P = 2C - B$ and let $Q = C + B$
- We add $P$ and $Q$ to $A$ (but not $C$ or $B$)
The Other Numbers 2

\[ B = \begin{array}{ccccccc}
1 & 1 & \ldots & 1 & \ldots & 1 & 1 & \ldots & 1 & \ldots & 1 & \ldots \\
\end{array} \]

\[ m_1 \rightarrow a_1 \]
\[ m_2 \rightarrow +a_2 \]
\[ \ldots \]
\[ m_k \rightarrow +a_k \]
\[ C \]

Note: C has the value of at most k in each segment.
Verification

- The total of all numbers in A is $C+P+Q=C+2C-B+C+B=4C$
- If A has a partition, each set must add up to $2C$
- If A has a partition, then P and Q must be in different sets. ($P+Q=3C$)
- A has a partition if and only if there is a subset A’ of A whose elements sum to B.
Consider the set containing $P=2C-B$. To reach the target size of $2C$, we must add elements totaling $B$ to this set.

Suppose $A$ has such a set $A'$. Let $M'$ be the subset of $M$ (in 3DM) that corresponds to $A'$. $M'$ is a complete matching for $M$. 
Verification 3

- If any element of \( W, X, \) or \( Y \) were missing, a segment of the sum of \( A' \) would be zero.

- If any element of \( W, X, \) or \( Y \) appears twice in \( M' \) the the sum of \( A' \) would not have a 1 in the position corresponding to that element. (Segments cannot overflow into one another.)
Verification 4

- If M has a complete matching M’ then the subset A’ of A corresponding to M’ has the sum B.
- Each element of W, X, and Y appears exactly once in M’, so each segment of the sum must equal one.
Graph 3-Colorability 1

- From 3-Sat
- For each clause, \{A,B,C\}, convert it into the following graph.
- Note: A, B and C can’t all be the same color.
- A, B, and C are the “Literal Vertices”
- This is the “Clause Component”
Bin Packing

- Input: a set of objects $B$ along with a set of associated sizes, $S$, such that every $b_i \in B$ there is a size $s_i \in S$. (Sizes not unique)
- For all $s_i \in S$, $0 \leq s_i \leq 1$.
- Minimization problem: What is the minimum number of bins of size 1 that will hold all elements?
Bin Packing

- Decision Problem: Will all objects fit in K bins?
- Transformation from partition.
- Given A, let X be the sum of all elements of A.
- Multiply each element by 2/X, and add to S.
- Ask the question, will the elements of S fit in 2 bins?
Subset Sum

- Given a set of numbers $S$ (with possible duplicates) and an integer $K$, is there a subset of $S$ whose sum is equal to $K$?
- Optimization problem: What is the subset of $S$ with the maximum sum not exceeding $K$.
- Transformation from partition. Use the same base set. Let $X$ be sum of all elements of $A$. $K=X/2$. 
Knapsack

Given a set of objects \( C = \{x_1, x_2, \ldots, x_n\} \) with an associated set of sizes \( \{s_1, s_2, \ldots, s_n\} \) and an associated set of values \( \{v_1, v_2, \ldots, v_n\} \), and two numbers \( k \) and \( m \) is there a subset \( A \subseteq C \) such that the sum of the sizes of the elements of \( A \) is less than or equal to \( k \), and the sum of the values of the elements of \( A \) is greater than or equal to \( m \)?
Knapsack

- From Partition:
- Let the objects be the numbers from the partition problem. Set both the size and the value of the number to be equal to its value.
- Set $m=k=$ half the total size of all elements.
3-Colorability 2

- Create the following graph segment
- Each variable appears in both complemented and uncomplemented form.
3-Colorability 3

- The graph segment on the previous slide is the truth-setting component.
- The color assigned to the T vertex will represent True, the color assigned to the F vertex will represent False, and the color assigned to the U vertex will represent “other.”
3-Colorability 4

- Go back to the Clause Components, and connect each Literal Vertex to the T vertex of the Truth-Setting component.
- If a Literal Vertex corresponds to the variable $x$, then connect the literal vertex to the $x$ vertex of the Truth-Setting Component.
- If it represents $x'$, then connect it to the $x'$ vertex.
3-Colorability: Proof

- The resultant graph is 3-Colorable, if and only if the original expression is satisfiable.
- Assign colors in the truth setting component to be consistent with the truth assignment.
- Because the assignment is satisfying, at least one literal in each clause must be assigned the “True” color.
3-Colorability: Proof 2

- Each literal vertex has two neighbors, one of which has the “True” color, the other of which may have either the “True” color or the “False” color.
- Since the Truth-Setting component is colored consistently with a satisfying assignment, each clause component will have a Literal Vertex with two “True” colored neighbors.
Use the “False” color to color the vertex with two “True” neighbors.

Complete the coloring as follows, (Red=False, Blue=True, Yellow=Other)
Now assume the graph is 3-colorable.

No Literal Vertex can be colored “True.”

In a Clause component, it is impossible to color all Literal Vertices “Other.”
3-Colorability: Proof 5

- A least one vertex in every Clause Component must be colored “False”. (The corresponding Vertex In Truth-Setting Component is colored “True”)
- Every coloring of the Truth-Setting Component corresponds to a truth-assignment of the original expression.
- A three coloring corresponds to a satisfying assignment.
LONGEST PATH
Given a graph $G=(V,E)$, and a positive integer $K \leq |V|$, does $G$ contain a simple path with $K$ or more edges?

SET PACKING
Given a collection $C$ of finite sets, and a positive integer $K \leq |C|$, Does $C$ contain $K$ disjoint sets?

Partition Into Hamiltonian Subgraphs
Given a graph $G=(V,E)$ and a positive integer $K \leq |V|$, can the vertices of $G$ be partitioned into $k \leq K$ disjoint sets $V_1, \ldots, V_k$ such that $1 \leq i \leq k$, the subgraph induced by $V_i$ contains a Hamiltonian circuit?
Exercises (Easy) - 2

- Largest Common Subgraph
  Given two graphs $G_1=(V_1,E_1)$, and $G_2=(V_2,E_2)$, and a positive integer $K$, do there exist subsets $E_1'\subseteq E_1$ and $E_2'\subseteq E_2$ such that $|E_1'|=|E_2'| \geq K$ and such that the two subgraphs $G_1'=(V_1,E_1')$ and $G_2'=(V_2,E_2')$ are isomorphic?

- Minimum Sum of Squares
  Given a finite set $A$, and an integer size $s(a)$ for all $a \in A$ and positive integers $K$ and $J$, can the elements of $A$ be partitioned into $K$ disjoint sets $A_1$ through $A_K$, such that

\[
\sum_{i=1}^{k} \left( \sum_{a \in A_i} s(a) \right)^2 \leq J
\]
Exercises (Medium) - 1

- Feedback Vertex Set
  Given a directed graph $G=(V,E)$, and a positive integer $K \leq |V|$ is there a subset $V' \subseteq V$ such that $|V'| \leq K$ and $V'$ contains a vertex from every directed cycle in $G$?

- Exact Cover by 4-Sets
  Given a finite set $X$, with $|X|=4q$, $q$ an integer, and a collection $C$ of 4-element subsets of $X$, is there a subcollection $C' \subseteq C$ such that every element of $X$ occurs in exactly one element of $C'$?

- Dominating Set
  Given a graph $G=(V,E)$, and a positive integer $K \leq |V|$, is there a subset $V' \subseteq V$, such that $|V'| \leq K$, and every vertex $v \in V-V'$ is joined to one element of $V'$ by an edge in $E$?
Exercises (Medium) - 2

- **Steiner Trees in Graphs**
  Given a graph $G=(V,E)$ and a subset $R \subseteq V$, and a positive integer $K \leq |V| - 1$ is there a subtree of $G$ that contains all vertices of $R$, and no more than $K$ edges?

- **Star-Free Regular Expression Equivalence**
  Given two star-free regular expressions $E_1$ and $E_2$, do $E_1$ and $E_2$ represent different sets of strings?
Exercises (Hard)

- **Set Splitting (3-Sat)**
  Given a collection $C$ of subsets of a finite set $S$, is there a partition of $S$ into two subsets $S_1$ and $S_2$ such that no element of $C$ is completely contained in either $S_1$ or $S_2$?

- **Partition into Paths of Length 2 (3DM)**
  Given a graph $G=(V,E)$ with $|V|=3q$, for some positive integer $q$, is there a partition of $V$ into $q$ disjoint subsets $V_1, V_2, \ldots, V_q$, of three elements each, such that for each $V_i=\{u,v,w\}$ at least two of the edges $\{u,v\}, \{v,w\}, \text{ and } \{u,w\}$ are contained in $E$?

- **Graph Grundy Numbering (3-Sat)**
  Given a directed graph $G=(V,E)$, is there a way to label the vertices with positive integers (duplicates are allowed), such that for each $v \in V$, the label on $v$ is the least non-negative integer which is not in the set of labels assigned to the successors of $v$?
Approximation Theory

- Used For Optimization Problems
- Feasible solution: A not-necessarily optimal solution to the problem
  - A valid, but not necessarily minimal graph coloring
  - A bin-packing into some number of bins, not necessarily minimal
Approximation Theory

- Given a problem $P$ and an input $I$, $\text{opt}(I)$ is the size of the optimal solution, sometimes denoted $\text{opt}_P(I)$.
  - The minimum number of colors needed to color a graph
  - The minimum number of bins needed to hold a set of elements
Approximation Theory

- Given an approximation algorithm $A$, and an Input $I$, $A(I)$ is the approximate solution, and $\text{Size}(A(I))$ is its size.
- The quality ratio of a solution $A(I)$ is defined as follows:

$$r_A(I) = \frac{\text{size}(A(I))}{\text{opt}(I)}$$

Minimization

$$r_A(I) = \frac{\text{opt}(I)}{\text{size}(A(I))}$$

Maximization
Approximation Theory

The quality measures of an approximation algorithm are:

\[ R_A(m) = \text{lub}\{r_A(I) \mid \forall I \text{ with } \text{opt}(I) = m\} \]

\[ S_A(m) = \text{lub}\{r_A(I) \mid \forall I \text{ with size } m\} \]

Replace Least Upper Bound with Maximum
For finite sets.
Approximation Theory

- $R_A(m)$ is a measure of how close to the optimal value I can get, regardless of input size.
- $R_A(m)$ is infinite for some problems
- $S_A(m)$ is a measure of how close to the optimal value one can get, taking input size into account.
- $S_A(m)$ is finite.
Approximation Theory

- $R_A = \text{lub} \{R_A(m) \mid m > 0\}$
- $S_A = \text{lub} \{S_A(m) \mid m > 0\}$
- For some bin-packing approximations, $R_A \leq 4/3$
- For graph coloring, approximation quality depends on graph size. For existing algorithms, there are families of 3-colorable graphs that require an arbitrarily large number of colors. $R_A$ is infinite.
Approximations

- Bin Packing
- Subset Sum
- Vertex Cover
- Graph Coloring
- Euclidean Traveling Salesman
- General Traveling Salesman
BIN PACKING Approximation

- **Real-Time First Fit:**
  - Add elements to Bin 1.
  - When Bin 1 is full go to Bin 2, and so forth.
  - Never go back to a previous bin.

- **First Fit**
  - Try each element in each bin, starting with Bin 1.
  - Add element to new bin if it won’t fit in any existing bin.
  - Elements are not sorted in any way.
BIN PACKING Approximation

- Non-Increasing First-Fit (Niff)
  - Sort elements into descending (non-decreasing) order
  - Then, same as First-Fit

- Niff is a good approximation
  - $R_A$ is finite, and small
  - Niff Runs quickly
Bin Packing Approximation

- In the approximation produced by Niff, there are $X \geq \text{Opt}(l)$ bins. The $X - \text{Opt}(l)$ bins are extra.
- The first element placed in an extra bin must be of size $\leq 1/3$.
- Suppose this were not the case. Because elements are placed in descending order, all placed objects must have size $> 1/3$. 
Bin Packing Approximation

- No bin can have more than two objects, because if one did, its total size would exceed 1.
- Some bins must have two objects, because if all have just 1, the extra-bin object would have to be placed with one of these objects in the optimal solution, but the algorithm tried to do this and it didn’t fit.
Bin Packing Approximation

- If some bins have only one object, they must precede the bins with two objects, because the algorithm tried to fit the extra-bin object into all of the 1-object bins, and it didn’t fit. Therefore none of the 2-object-bin objects will fit either, because they must be the same size or larger than the extra bin object. Since they are smaller than the 1-bin objects, they must have been placed later.
Bin Packing Approximation

- Assume there are $k \leq \text{Opt}(I)$ 2-object bins.
- The $2k$ objects in these bins plus the object placed in the extra bin must fit in $k$ bins in the optimal solution.
- Since there are $2k+1$ objects, at least one bin must have three objects.
- Since all objects have size $> 1/3$, this bin must have size $> 1$ which is impossible.
Bin Packing Approximation

- The number of objects placed in extra bins must be less than Opt(I).
- Suppose that Opt(I) objects are placed in extra bins. Denote these objects as $e_1, e_2, \ldots, e_{Opt(I)}$
- Object $e_i$ will not fit in bin $i$. The algorithm tried to put it there, and it wouldn’t fit.
Bin Packing Approximation

- Let the total size of all objects in bin $i$ be designated as $B_i$.
- Because object $e_i$ won’t fit in bin $i$, the following two inequalities must be true.

\[ e_i + B_i > 1 \]

\[ \sum_{i=1}^{Opt(I)} (e_i + B_i) > Opt(I) \]
Bin Packing Approximation

- However, because Opt(I) is the size of the optimal solution, the total size of all objects must be less than or equal to Opt(I).
- Taken together, the total number of extra bins cannot exceed Opt(I)/3.
- $R_{\text{Niff}} \leq 4/3$
- The above computation assumes that Opt(I) is a multiple of 3. Exercise: consider the other two cases using Opt(I)-1 instead of Opt(I).
Bin Packing Approximation

- The largest difference occurs when the optimum is 2 bins, but the algorithm uses 3.
- .5, .4, .3, .3, .3, .2

Niff Solution

Optimal Solution

.2
.3
.5
.4
.3
.3
Bin Packing Approximation

- \( S_{\text{Niff}} \leq 3/2 \)
- Exercise: Find a family of sets of objects with arbitrarily large sets, such that the optimal bin packing has 2 bins, but Niff uses 3 bins.
- Solution: \( S_1 = \{0.5, 0.4, 0.3, 0.3, 0.3, 0.15, 0.05\} \)
- \( S_k = S_{k-1} \) but divide the smallest element in half. \( S_2 = \{0.5, 0.4, 0.3, 0.3, 0.3, 0.15, 0.025, 0.025\} \)
- \( S_3 = \{0.5, 0.4, 0.3, 0.3, 0.3, 0.15, 0.025, 0.0125, 0.0125\} \)
Subset Sum Approximation

- Subset Sum: given a set of n objects of sizes $s_1$ through $s_n$, and an integer $K>0$, find the subset with the largest total size not exceeding C.

- Greedy algorithm: consider objects in order 1-n. Add each object $s_i$ to the set unless the object would make the total exceed C. If the object $s_i$ does cause the limit to be exceeded, but $s_i$ is larger than the current total, throw everything away, and put $s_i$ in the set. (and continue)
Subset Sum Approximation

- Better Greedy Method: for every subset, $S$, of objects containing at most $k$ objects, where $k$ is a constant, start the greedy algorithm with the elements of $S$ already selected.
- This is a family of approximation algorithms, one algorithm for each $k$.
- Denote these algorithms $A_k$.
- $A_k$ is of order $n^{k+1}$ and gives an approximation with a quality ratio of $1+1/k$ or smaller.
Because we start the greedy method with all subsets of size $k$, we must start with the set that contains the $k$ largest items in the optimal solution.

There must be at least one element $e_x$ of the optimal solution that is not in the approximate solution.
Subset Sum Approximation

- The element $e_x$ is not one of the $k$ largest elements of the optimal solution, therefore its size must be less than or equal to $\text{Opt}(I)/(k+1)$.
- The algorithm attempted to include $e_x$ in the solution, but it wouldn’t fit.
- The amount of slack ($\text{slack} = C - \text{Solution size}$) must be less than $\text{Opt}(I)/(k+1)$.
- Since $\text{Opt}(I)<C$, the difference between $\text{Opt}(I)$ and the approximate solution must be less than $\text{Opt}(I)/(k+1)$ as well.
Subset Sum Approximation

\[ R_{A_k}(Opt(I)) = \frac{Opt(I)}{Opt(I) - Opt(I)/(k+1)} \]

\[ = \frac{1}{1 - 1/(k+1)} = \frac{1}{((k+1) - 1)/(k+1)} \]

\[ = \frac{k+1}{k} = \frac{k}{k} + \frac{1}{k} = 1 + \frac{1}{k} \]
Subset Sum Approximation

- For each subset, the algorithm does $\Theta(n)$ work, looking at each of $n-k$ elements using constant time for each.
- There are $\Theta(n^k)$ subsets of size $k$.
- Each subset can be generated in $\Theta(k)=\Theta(1)$ time.
- (Note that $k$ is a constant.)
Subset Sum Approximation

- Assume all element sizes are stored in a 1-based array.
- Use an array of size $k$ to generate the subset.
- Initialize the array as follows:

```
1 2 ... k
```
Subset Sum Approximation

- Each element of the array has a limiting value. These values are illustrated below.

\[
\begin{array}{cccc}
  n-k & n-k+1 & \ldots & n \\
\end{array}
\]
Subset Sum Approximation

- To generate a new set, increment the $k^{th}$ element of the array.
- If the $k^{th}$ element exceeds its limiting value, go to the $k-1^{st}$ element and increment that.
- Continue until we encounter an element that does not exceed its limiting value after incrementing.
Subset Sum Approximation

- Suppose the $i^{th}$ element was incremented to the value $x$.
- Now move forward through the array, setting each value to one larger than the previous. The $i+1^{st}$ element is set to $x+1$, the $i+2^{nd}$ is set to $x+2$, etc.
- If all elements exceed their limiting values, we have generated all subsets, so stop.
Subset Sum Approximation

- Subsets of size 3 from a set of 6 elements:
  - {1,2,3}  {1,2,4}  {1,2,5}  {1,2,6}  {1,3,4}
  - {1,3,5}  {1,3,6}  {1,4,5}  {1,4,6}  {1,5,6}
  - {2,3,4}  {2,3,5}  {2,3,6}  {2,4,5}  {2,4,6}
  - {2,5,6}  {3,4,5}  {3,4,6}  {3,5,6}  {4,5,6}
Vertex Cover Approximation

- Create a matching set by starting with the empty set $M$.
- Choose an arbitrary edge $e$ from $G$.
- Add $e$ to our matching set $M$.
- Delete $e$ and the vertices incident to it from $G$.
- Repeat the previous 3 steps until $G$ has no edges.
Vertex Cover Approximation

- The vertices incident on the edges of M form a vertex cover $V'$.
- $V'$ is no larger than twice the minimal cover.
- One endpoint of each edge in M must be in every vertex cover, so it is not possible to delete more than $|M|/2$ vertices from $V'$ and still have it cover all vertices.
Graph Coloring Approximation

- Given $G=(V,E)$ with $n$ vertices.
- Use the integers $\{1,2,3,\ldots,n\}$ to represent colors.
- Start by assigning 0 to every vertex.
- Process the vertices one at a time
- For each vertex, $V_i$, start by coloring $V_i$ with the color 1.
Graph Coloring Approximation

- Check the neighbors of $V_i$ to see if any is colored 1. If not then go to the next vertex, $V_{i+1}$.
- If there is a neighbor colored 1, recolor $V_i$ with color 2, and repeat the neighbor search.
- Repeat the previous step incrementing the color until we find a color $c$ that has not been used to color any of $V_i$’s neighbors.
Graph Coloring Approximation

- This algorithm is called Sequential Graph coloring, or SC.
- Let K be the maximum degree of any vertex in G. Then SC uses no more than K+1 colors.
- Proof: The color-assignment and testing procedure will test no more than K+1 colors. The procedure always starts with 1 and increments.
Graph Coloring Approximation

- There are bipartite (2-colorable) graphs for which SC uses an arbitrarily large number of colors.

Processing order of $a_1, b_1, a_2, b_2, \ldots, a_k, b_k$ uses $k$ colors.

Every vertex on the top connected to every vertex on the bottom except the one directly below it.
Graph Coloring Approximation

- Approximate Graph Coloring is hard
- Suppose we have an approximation algorithm which is guaranteed to produce a coloring with less than $\frac{4}{3}$ the optimal number of colors.
- This algorithms colors 3-colorable graphs with $n<3\times\frac{4}{3}=4$ colors. I.E., 3 colors. Four-colored and higher graphs need 4 colors.
- Thus the approximation algorithm gives us a way to solve the 3-colorability problem in polynomial time.
Graph Coloring Approximation

- Even if the approximation works only for graphs that require a large number of colors, the result is the same.
- Suppose the graph works only for graphs that require \( k \) or more colors.
- (The minimum number of colors needed to color a graph is called its Chromatic Number, and is designated \( \chi(G) \))
Graph Coloring Approximation

- **Graph Composition**: Given $G$ and $H$, replace every vertex of $G$ with a copy of $H$.
- Denote the replacement of vertex $v$ as $H_v$.
- If $(v,w)$ is an edge in $G$, connect every vertex of $H_v$ to every vertex of $H_w$. 
Graph Coloring Approximation

Composition
Graph Coloring Approximation

- If we have a <4/3 optimal graph coloring algorithm that works for graphs with chromatic numbers of k or larger, compose the original graph with a complete graph on k vertices.
- If the original chromatic number of G was $\chi(G)$, the new graph has chromatic number $k\chi(G)$. 
Graph Coloring Approximation

- If the original graph was three-colorable, the approximation algorithm will use less than $\frac{4}{3} \times 3k = 4k$ colors.
- If the original graph requires more than three colors, then the approximation algorithm must use at least $4k$ colors to color it. (Chromatic number is at least $4k$)
Graph Coloring Approximation

- Suppose we have an approximation algorithm that guarantees to use no more than $M^* \chi(G)$ colors, $M$ a constant.
- If we compose a graph with itself, the new chromatic number is $\chi(G)^2$. If we do it twice, the new chromatic number is $\chi(G)^3$. 
Graph Coloring Approximation

- For every constant $M$, there is a constant $K$ such that $3^K < M4^K$.
- Thus we can use an approximation with an $M\chi(G)$ guarantee to solve the 3-colorability problem in polynomial time by composing a graph with itself $K$ times.
- (The composition is huge, but polynomial in size.)
Traveling Salesman Approx.

- Assume that the triangle inequality holds. In other words, 
  \[ w(a,b) + w(b,c) \leq w(a,c) \]
- Obtain the minimum spanning tree of the complete weighted graph.
- The weight of the minimum spanning tree must be less than the weight of the minimum Hamiltonian Path.
Traveling Salesman Approx.

- Form a non-simple cycle by traversing the MST. When a leaf is encountered, reverse direction and go back. This cycle will have weight twice that of the MST.
- Convert the MST to a simple cycle by shortcutting vertices.
- The result will have no more than twice the weight of the minimum Hamiltonian path.
Traveling Salesman Approx.
Traveling Salesman Approx.

- The general problem is much harder to approximate.
- Suppose we have an approximation that is guaranteed to find a Hamiltonian cycle with less than K times the minimum weight.
- We can use this algorithm to solve the general Hamiltonian cycle problem in polynomial time.
Traveling Salesman Approx.

- Given an arbitrary graph $G$, assign a weight of 1 to each edge.
- Add all other edges to $G$ to make it a complete graph.
- Assign a weight of $n*K+1$ to each new edge.
- If the original graph has a Hamiltonian cycle, the approximation algorithm must find it, otherwise the weight of the found cycle would be at least $n*K+1$, more than $K$ optimal.