CSI 5166

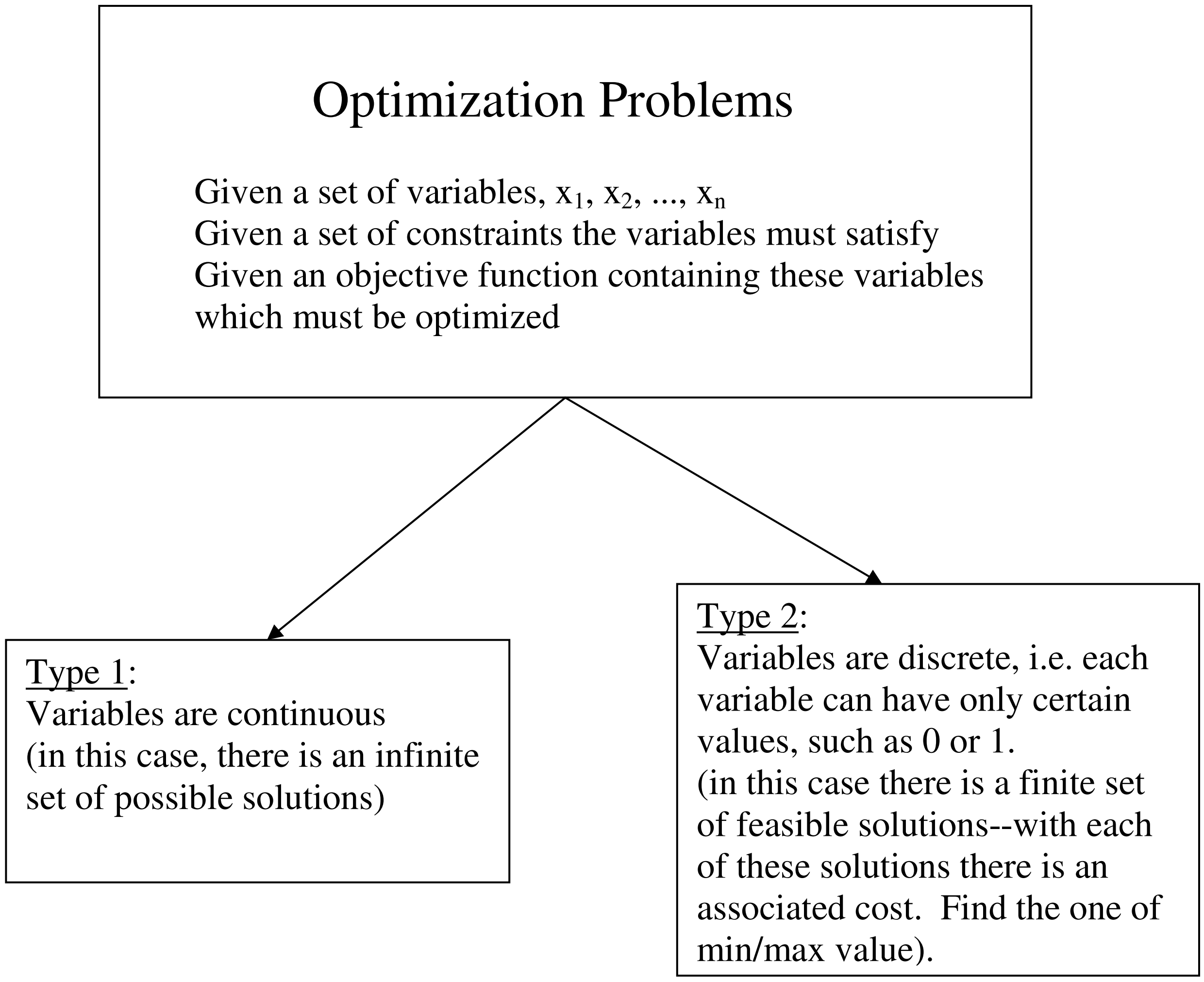
APPLICATIONS OF COMBINATORIAL OPTIMIZATION

Class Notes
Winter 2017

By Dr. Sylvia Boyd

1. Introduction and Background

What is Combinatorial Optimization?



Optimization Problems

Given a set of variables, x_1, x_2, ..., x_n
Given a set of constraints the variables must satisfy
Given an objective function containing these variables which must be optimized

Type 1:
Variables are continuous
(in this case, there is an infinite set of possible solutions)

Type 2:
Variables are discrete, i.e. each variable can have only certain values, such as 0 or 1.
(in this case there is a finite set of feasible solutions--with each of these solutions there is an associated cost. Find the one of min/max value).

Combinatorial optimization problems are discrete optimization problems. The problems are often modelled by a weighted graph, and the constraints and variables then represent something in the graph.

Example of a graph G:

Some Examples of Combinatorial Optimization Problems

1. **Optimal drilling of circuit boards**
   (TRAVELLING SALESMAN PROBLEM)

Given a circuit board and a set of holes to be drilled by a laser drill which is moved mechanically from hole to hole.

We wish to decide an order to drill these holes which requires the laser drill to be moved the least distance.

This problem can be modelled using a complete weighted graph G:

The circuit board drilling problem is an application of the what is called the *travelling salesman problem (TSP)*: Given a complete weighted graph G, find a cycle which visits all the nodes exactly once and ends up where it started, and has minimum weight.

2. Optimal assignment of T.A.'s and courses (MAXIMUM BIPARTITE MATCHING)

Given a set of courses and a set of possible TAs, connect a TA to the courses which he/she is willing to TA. Find a matching of TAs to courses which maximizes the number of courses that have a TA assigned.

Example:

This is an application of *maximum bipartite matching*: Given a bipartite graph with partition X, Y of the nodes, find the largest set of edges which do not share an end node.

3. Construction of reliable communication networks (MINIMUM WEIGHT K-EDGE CONNECTED SUBGRAPH PROBLEM)

A communication network consists of centers and links. You can talk about the reliability of the network in terms of the number of links that it can withstand losing (and still have communication).

Examples:

In designing a network with a certain reliability, you are given the cost of building each link. Now figure out which links to choose so that the specified reliability is met, and the network has least cost to build.

This is an application of the *minimum weight k-edge connected subgraph problem*: Given a weighted graph G, find the subgraph of minimum weight which is k-edge connected.

Example:

4. Logic circuit plotter
(MINIMUM WEIGHT MATCHING PROBLEM, CHINESE POSTMAN PROBLEM)

A plotter is used to draw proposed logic circuits. It operates by moving a pen back and forth and, at the same time, rolling a sheet of paper (which is on a roller) forwards and backwards underneath the pen.

Problem: Minimize the time to draw the circuit.

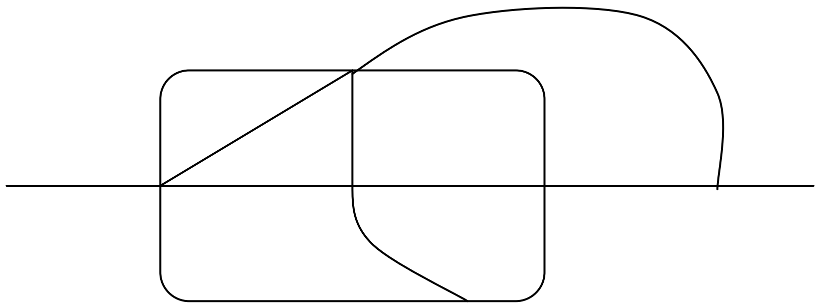
"pen-down" time + "pen-up" time

(actual drawing)

(pen is not in contact with paper, but is moving from the end of one line to where it will start the next line)

(Note: Pen must end up where it started)

Example:



We can represent this as a graph by placing a node where 2 or more lines meet or cross, or a line ends.

Can this circuit be drawn with no pen-up motion?

Theorem (Euler): The figure can be traced with no pen-up motion if and only if it is connected and all nodes have even degree.

To minimize pen-up motion (this is similar to the Chinese Postman Problem): Find a new set of lines we can add to the figure which turn every odd node into an even node, such that the total traversal time of the new lines is as small as possible. These new lines will "pair up" the odd nodes).

Let t(p,q) be the time required for the pen to move from p to q.

Create a new complete weighted graph consisting of the odd nodes, using t(p,q) as the weights.

Now solve the minimum weight perfect matching problem: Find a set of n/2 node-disjoint edges such that the sum of the weights on the edges is minimized.

The result for our problem:

How will we draw the figure?

BACKGROUND INFORMATION

1. GRAPH THEORY

A *graph* G = (V,E) is defined as a finite set of *nodes* (also sometimes called *vertices*) V which are interconnected by a finite set of *edges* E. We usually use m to denote |E| and n to denote |V|. A graph is a very useful tool which can be used to model many different problems.

Each edge e in E corresponds to two nodes in V, called the *ends* of e. An edge e in E with ends u and v can be denoted by uv. Two nodes u and v are said to be *adjacent* if uv is in E, and e is said to be *incident* with u and v.

A graph is *simple* if it has no loops or parallel edges. A graph is *complete* if it is simple and has all possible edges uv, for all nodes u, v in V, $u \neq v$. The complete graph on n nodes is denoted by K_n.

The *neighbours* of a node v are all the nodes in V which are adjacent to v in G, and the *degree* of a node v is the number of neighbours of v, and is often denoted by *d(v)*. A graph with all node degree k is called *k-regular.*

Important Fact 1: Σ (d(v) : $v \in V$) = 2*|E|
Proof:

Important Fact 2: The number of nodes with odd degree is even in any graph.
Proof:

Important Fact 3: The number of edges in K_n is:
Proof:

A *weighted graph* is one in which every edge is assigned a number called its *weight* or *cost*.

A *subgraph* of a graph G = (V, E) is a graph G' = (V', E') such that V' is a subset of V, and E' is a subset of E such that both ends of every edge in E' are in V'. A subgraph is called *spanning* if V' = V.

A *path* from node v_0 to node v_k in a graph is a sequence of edges $v_0v_1, v_1v_2, \ldots, v_{k-1}v_k$, where $v_0, v_1, \ldots, v_k$ are all distinct. Such a path can also be represented by $v_0, v_1, v_2, \ldots, v_k$. A single node is a path of length 0. A *cycle* in a graph is like a path except $v_0 = v_k$. (i.e. we can represent a cycle by a node sequence $v_0, v_1, v_2, \ldots, v_k$, where the nodes are distinct except that $v_0 = v_k$ and there is an edge v_iv_{i+1} for i = 0, 1, 2,...,k-1, and an edge v_kv_0 in G). The *length* of a path or cycle is the number of edges contained in it.

A graph is *k-edge connected* if there are k edge-disjoint paths between every pair of nodes. A 1-edge connected graph is simply called *connected*. A *component* of a graph is a maximal connected subgraph.

A *cut* in a graph is defined by a set of nodes $Q \subseteq V$ and is the set of edges with one end in Q, one end not in Q (this set of edges is denoted by $\delta(Q)$). Note that removing the edges of a cut from G disconnects that component.

Important Fact: A graph is k-edge connected if and only if the size of a smallest cut in G is k.

A graph *is k-node connected* if there are k paths between every pair of nodes such that these paths are internally node-disjoint. A *k-node cut* is a set of nodes such that removing these nodes from the graph (along with their incident edges) disconnects the graph.

Important Fact: G is k-node connected if and only if the size of a smallest node cut is greater than or equal to k.

A graph is called *acyclic* if it contains no cycles. A *forest* is an acyclic graph. A *tree* is a connected forest. The nodes of degree 1 in a tree are called the *leaves*.

A graph is called *bipartite* if there is a partition of the nodes into two sets X, Y such that all edges have one end in X and one end in Y. A *complete bipartite graph* is one for which all possible edges between X and Y are included, and is denoted by $K_{a,b}$, where $a=|X|$ and $b=|Y|$.

Important Fact: A bipartite graph contains no odd length cycle.

A *directed graph*, or *digraph*, is a graph in which all of the edges have been given a direction. In a digraph, an edge is represented by an ordered pair <u, v> where the edge (also called an *arc*) is directed from u to v. Node u is called the *tail* of the arc, and v is called the *head*. The *indegree* of a node v (denoted by $d^{in}(v)$) is the number of edges directed into v, and the *outdegree* of v (denoted by $d^{out}(v)$) is the number of edges directed out of v.

Important Fact: $\Sigma\ (d^{in}(v) : v \in V) = \Sigma\ (d^{out}(v) : v \in V) = |E|$

A directed graph is called *symmetric* if $d^{in}(v) = d^{out}(v)$ for all nodes v. A directed graph is *strongly connected* if there is a directed path from u to v and from v to u for all node pairs u and v.

2. INTEGER AND LINEAR PROGRAMMING PROBLEMS

Given a set of variables $x_1, x_2, ..., x_n$, a *linear programming problem (LP)* has the form:

Minimize (or maximize) $z = c_1x_1 + c_2x_2 + c_3x_3 + ... + c_nx_n$

Subject to:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots a_{1n}x_n \geq b_1$$
$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \ldots a_{2n}x_n \leq b_2$$
$$\vdots$$
$$a_{m1}x_1 + a_{m2}x_2 + a_{m3}x_3 + \ldots a_{mn}x_n = b_m$$

(may also have $x_i \geq 0$ for some or all i values)

Notes:

If there is an added restriction that the variables must have integer values, then it is called an *integer linear programming problem (ILP)*, and if the variables all must be 0 or 1, then it is called a *binary ILP*. There are no known efficient methods for solving ILPs exactly in general (although sometimes there are methods that can be applied for particular ILPs). Later we will look at some strategies for solving difficult combinatorial optimization problems using both LP and ILP models.

Modelling Example: An Integer Linear Programming Formulation (ILP) For the Perfect Matching Problem

General Perfect Matching Problem**:**
Given a complete weighted graph G = (V,E) on n nodes where n is even, find a minimum weight (cost) perfect matching in G (a *perfect matching* is a set of n/2 edges such that every node in V is an endpoint of exactly 1 of these edges).

Example:

We can represent any perfect matching M by a 0-1 vector $x \in R^E$ as follows:

$$x_e = \begin{cases} 1 & \text{if edge } e \text{ is in } M \\ 0 & \text{otherwise} \end{cases}$$

For our example:

For an ILP description:

Matching Degree Constraints: The idea behind these constraints is that there is exactly one edge in M incident with each node.

Form of Constraints: $\sum (x_{ij} : \ i \in V, i \neq j) = 1$ for all nodes j in V.
For our example:

<u>ILP Formulation</u>:

If you solve the ILP, for all $x \in R^E$,

$$\begin{aligned} &\text{minimize } cx \\ &\text{subject to } \ 1) \ \text{matching degree constraints,} \\ &\qquad\qquad\quad 2) \ x_e \in \{0,1\} \end{aligned}$$

then you will have a vector x whose 1's represent an optimal perfect matching M for our problem.

<u>Linear Programming (LP) Formulation for Perfect Matchings</u>:

If we relax the restriction of x being a 0-1 vector in the ILP formulation and instead allow the values of x to lie anywhere between 0 and 1 (i.e. $0 \leq x_e \leq 1$), we may get fractional optimal solutions to the corresponding LP.

Example:

However, it turns out that for the perfect matching problem, there is another set of constraints we can add to our LP which will ensure that there IS a 0-1 optimal solution.

Odd Cut Constraints: Suppose we have a subset S of the nodes such that the size of S is odd. What must be true about any perfect matching M of our graph?

How do we represent this as a set of constraints?

Final LP representation for perfect matchings:
If you solve the LP, for all $x \in R^E$,

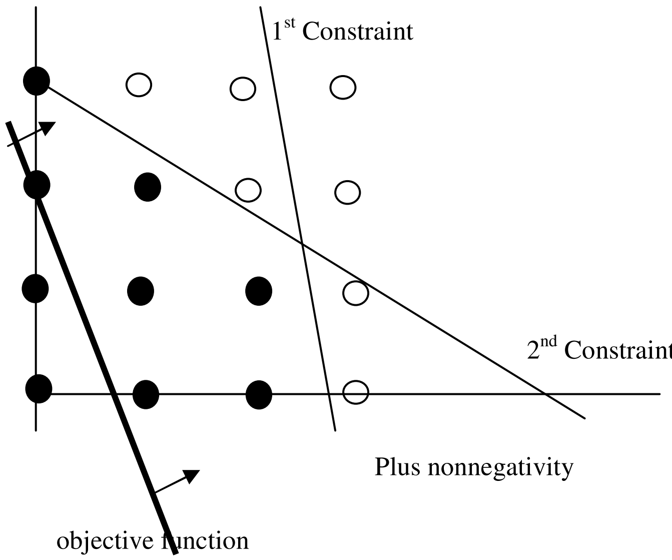
minimize cx
subject to 1) matching degree constraints,
2) odd cut constraints
3) $0 \leq x_e \leq 1$ for all edges e in E

then there will exist a 0-1 optimal solution x where the 1's correspond to a perfect matching.

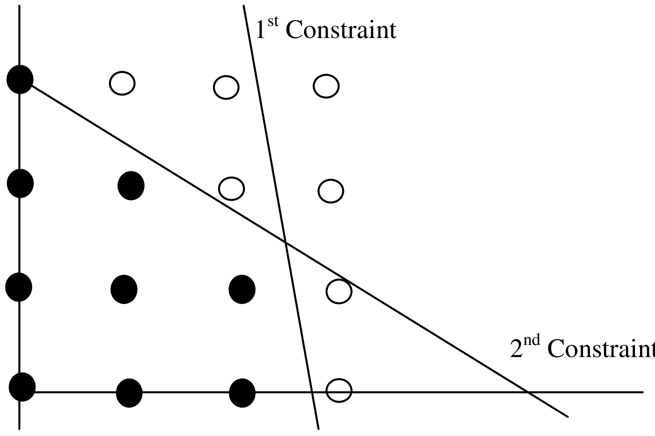
ILP's Geometrically

For any ILP we have a linear set of constraints defining a feasible region (called the *LP relaxation feasible region*), and we are interested in the integer solutions in that region.

This is illustrated in the figure below.



Theorem: For any ILP P, there is a corresponding LP Q such that, for every objective function there exists an integer optimal solution for Q (so solving the LP Q solves the ILP P).



Question: How easy is it to find the corresponding LP Q for an ILP in general ?

Another example: An Integer Linear Programming Formulation (ILP) of the Travelling Salesman Problem (TSP)

General TSP: Given a complete weighted graph G = (V,E) on n nodes, find a cycle which visits every node exactly once, and has minimum weight (or cost). We call the edge set of such a cycle a **tour**.

We can represent any tour T by a 0-1 vector $x \in R^E$ as follows:

$$x_e = \begin{cases} 1 & \text{if edge } e \text{ is in } T \\ 0 & \text{otherwise} \end{cases}$$

Example: For n = 4

G:

We can represent T for our example by

$x =$

Cost vector $c \in R^E$ for our example:

$c =$

So the cost of T is $cx =$

An Integer Linear Programming (ILP) formulation for the TSP is a set of linear constraints which, if satisfied by any 0,1 vector x, imply that x represents a tour.

For this ILP formulation we have the following types of constraints:

<u>Degree Constraints</u>: The idea behind these constraints is that our tour must come to and leave every node j <u>exactly</u> once.

Form of constraints: $\sum (x_{ij} : i \in V, i \neq j) = 2$ for all nodes j in V.

For our example:

Subtour Elimination Constraints

Note that 0-1 vectors that satisfy all degree constraints may not represent a tour, as there may be subtours present. The subtour elimination constraints make sure that for any node subset S, our solution comes to S and leaves it at least once.

Form of constraints: $\sum (x_e$: e has exactly one end in S$) \geq 2$ for all $S \subset V$, $|S| \geq 2$, $|S| \leq n-2$.

For our example:

ILP Formulation:

If you solve the ILP, for all $x \in R^E$,

minimize cx
subject to 1) degree constraints,
2) subtour elimination constraints
3) $x_e \in \{0,1\}$

then you will have a vector x whose 1's represent an optimal TSP tour for our problem.

Problems:

Question: How hard is it to find the necessary constraints to add to the ILP for TSP so that we can drop the integer requirement?

Answer:

There have been a huge number of families of constraints which have all been shown to be necessary for this.

For n=8: The complete corresponding LP for the TSP has been found (by computer generation) and there are______________ constraints in total.

No one knows the complete LP description for n>= 10 at this time.

3. BRIEF BACKGROUND: ALGORITHM ANALYSIS AND COMPLEXITY THEORY

Algorithm Analysis

When we are analyzing algorithms to predict their run time in this course, we will be considering what happens in the worst case. The analysis will be an approximation of the function f(k) (ignoring constants) that represents the number of basic operations performed by the algorithm when the size k of the problem gets large. The complexity of the algorithm will be written as $\Theta(f(k))$ or $O(f(k))$.

Notes:

Complexity of Problems

Classes of problems: P, NP, NP-complete

Class P: The class of problems solvable in polynomial time.
i.e. There is an algorithm known for the problem which has complexity bounded above by a polynomial in k.

Class NP (nondeterministic polynomial): (just the idea)
(Note: It deals only with problems in their decision, i.e. yes/no form)

The problems for which, for any input that has a yes answer, there is a certificate from which the correctness of this answer can be derived in polynomial time.

Example:

Class NP-complete: (hardest problems in NP)

A problem Q (in decision form) that is in NP is called NP-complete if <u>all</u> problems in NP can be reduced to the problem.

If a problem is NP-complete, it implies that if one problem can be solved in polynomial time, then they all can.

Relationship of problem sets P, NP, NP-complete:

$P \subseteq NP$, NP-complete $\subseteq NP$

A very famous problem is to find the answer to the following question: Does P = NP?

Note: If you could find a polynomial-time algorithm for any one NP-complete problem, then you would have shown that P = NP.

Status of problems we have looked at so far: