INTEGER PROGRAMMING

PREREQUISITES: ME 366M (Introduction to Operations Research) or equivalent; a working knowledge of at least one computer programming language.


OBJECTIVE: Many problems that arise in manufacturing and socio-economic systems, such as machine scheduling, vehicle routing, resource management, and telecommunications network design, can be modeled as integer or mixed-integer programs. Generic models that make up the field of combinatorial optimization also fit the integer programming (IP) format. The aim of this course is to present the theory and exact techniques that have been developed to solve related models. These techniques include branch and bound, cutting planes, Lagrangian relaxation, and column generation (Dantzig-Wolfe). However, it is rare that any one technique can be applied successfully in solving realistically sized IPs. In most cases, it will be necessary to identify and exploit a familiar underlying structure in the model. Polyhedral theory will play an active role in this regard and will be discussed at some length. In addition, students will be required to program a number of the algorithms presented in class.

GRADING:

<table>
<thead>
<tr>
<th>Component</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Homework</td>
<td>8%</td>
</tr>
<tr>
<td>Programs</td>
<td>7%</td>
</tr>
<tr>
<td>Midterm Exam 1</td>
<td>20% (Tuesday, February 24)</td>
</tr>
<tr>
<td>Midterm Exam 2</td>
<td>25% (Tuesday, March 31)</td>
</tr>
<tr>
<td>Final Exam</td>
<td>40% (Friday, May 15, 2 - 5 PM)</td>
</tr>
</tbody>
</table>

NOTE: Homework that is one class late will be penalized 10%; it will not be accepted after that date. All students must take the exams when scheduled. There will be NO make-up exams and there will be NO incompletes in the course; every student will get a grade.
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Pages</th>
<th>Topic</th>
<th>HW#</th>
<th>Problems</th>
</tr>
</thead>
<tbody>
<tr>
<td>I.1</td>
<td>3-20</td>
<td>Introduction</td>
<td>1</td>
<td>I.1-2,3,6,7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modeling</td>
<td>2</td>
<td>I.1-10,11,17,19</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Commercial codes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Preprocessing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I.5</td>
<td>114-139</td>
<td>Computational Complexity</td>
<td>3</td>
<td>I.5-11, handout</td>
</tr>
<tr>
<td>II.4</td>
<td>349-367</td>
<td>Branch and Bound</td>
<td>4</td>
<td>II.4-1,2,3,8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>General relaxation</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Search trees</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Implementation issues</td>
<td></td>
<td></td>
</tr>
<tr>
<td>notes</td>
<td>205-229</td>
<td>0-1 Implicit Enumeration</td>
<td>5</td>
<td>handout</td>
</tr>
<tr>
<td>II.1</td>
<td></td>
<td>Traditional Cutting Planans</td>
<td>6</td>
<td>II.4-9,12</td>
</tr>
<tr>
<td>notes</td>
<td>337-341</td>
<td>Method of integer forms</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>412-417</td>
<td>Primal all-integer cuts</td>
<td></td>
<td></td>
</tr>
<tr>
<td>II.3</td>
<td></td>
<td>Cuts with unit coefficients</td>
<td></td>
<td></td>
</tr>
<tr>
<td>II.5</td>
<td></td>
<td>Dual all-integer cuts</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I.2</td>
<td>41-49</td>
<td>Lagrangian Relaxation</td>
<td>8</td>
<td>II.3-13,14(i-iii),16,17</td>
</tr>
<tr>
<td>II.3</td>
<td>323-337</td>
<td>Relation to duality</td>
<td></td>
<td></td>
</tr>
<tr>
<td>II.5</td>
<td>409-412</td>
<td>Subgradients</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Implementation</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>notes</td>
<td>Lagrangian decomposition</td>
<td></td>
<td></td>
</tr>
<tr>
<td>II.1</td>
<td>83-109</td>
<td>Polyhedral Theory</td>
<td>9</td>
<td>I.4-1,2,4,11, II.1-5</td>
</tr>
<tr>
<td>II.1</td>
<td>205-217</td>
<td>Linear algebra</td>
<td>10</td>
<td>II.2-1,3,4</td>
</tr>
<tr>
<td>II.2</td>
<td>259-290</td>
<td>Valid inequalities</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Structured problems</td>
<td></td>
<td></td>
</tr>
<tr>
<td>III.1</td>
<td>535-561</td>
<td>Unimodularity and Graphs</td>
<td>11</td>
<td>III.3-4,5,6</td>
</tr>
<tr>
<td>III.2</td>
<td>608-626</td>
<td>Matching and Covering Problems</td>
<td></td>
<td></td>
</tr>
<tr>
<td>II.5</td>
<td>420-422</td>
<td>Knapsack Problem</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Notes</td>
<td></td>
<td>Heuristic Methods</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*a* Notes from *Practical Bilevel Programming* by Bard, Chapter 3 — Integer Programming.

*b* Notes from *Integer Programming* by Garfinkel and Nemhauser.
COMPUTER ASSIGNMENTS

During the semester, you will be asked to write a number of computer programs implementing some of the methods discussed in class. These programs can be written in any language and run on any machine you choose, circumstances permitting. The code should be well-documented with comments so someone reviewing it can understand the logic even if he or she is not familiar with the language. In addition, input and output should be well thought out. Interactive codes are fine if the user is not overburdened with input requests. Output should be tabularized and easy to read. Often you will be asked to provide the solution to a sample problem.

Computer accounts for the ME RS/6000 workstation laboratory in ETC 3.140 can be obtained from the lab administrator Sue Ponder in ETC 3.138. Accounts on University computers can be obtained from the Computation Center.

In general, the assignment that you turn in should include:

1) Name, date, class, title of assignment.
2) Well-documented code.
3) Input files if not an interactive problem; if interactive, provide a printout of the session.
4) Statement of model. If solving an IP, for example, provide the model in algebraic form and the data.
5) Results in tabularized, or other easy to read form.

Assignments:

(Data structures and sorting)
1a. Let \( \bar{N} \) be a fixed positive integer. Generate \( N \) random integers \( \{Y_1, \ldots, Y_N\} \) between 0 and \( \bar{N} \). Write a routine that sorts these numbers in three different ways: (i) descending order, (ii) ascending order, (iii) ascending order of \( D_i \), where \( D_i = |Y_i - C| \) is the deviation from a given constant \( C \).

An easy way to do this is with a bubble sort. That is, compare the numbers in the first two positions, \( Y_1 \) and \( Y_2 \). If \( Y_2 > Y_1 \), interchange their positions. Now compare the numbers in the second and third positions continuing with the same logic. After executing all pairwise comparisons in sequence, repeat the entire process until no more interchanges are required. Demonstrate your routine for \( N = 24 \).

1b. (Heaps) Assume that \( N \) is odd (if not, let the \( N+1 \)st number be \(-\infty\) and put \( N \leftarrow N+1 \)). The numbers \( Y_1, \ldots, Y_N \) are said to form a heap if

\[
Y_i \geq \max\{Y_{2i}, Y_{2i+1}\} \quad \text{for} \quad 1 \leq i \leq N/2 \tag{1}
\]
A heap of 11 numbers is shown in Fig. 1, with $Y_i$ adjacent to node $i$; e.g., $Y_2$ is 35 and is adjacent to node 2. Nodes $2i$ and $2i+1$ are called the children of node $i$. Node $i$ is the parent. The following routine places $N$ numbers in a heap.

**Heaping routine (with $N$ odd).**

1. Set $k = (N+1)/2$.
2. Let $k \leftarrow (k-1)$. If $k = 0$, stop. Otherwise put $j \leftarrow k$.
3. Set $s \leftarrow 2j$. If $Y_s < Y_{s+1}$, put $s \leftarrow (s+1)$. If $Y_s > Y_j$, go to Step 4; otherwise, go to Step 2.
4. Interchange $Y_s$ with $Y_j$. If $2s > N$, go to Step 2. Otherwise, put $j \leftarrow s$ and go to Step 3.

Remark: If it is desirable to for a heap with the smallest number at the top (i.e., with $Y_i \leq \min\{Y_{2i}, Y_{2i+1}\}$ in eq. (1)), reverse the inequalities in Step 3 of the above routine.

After putting the $N$ numbers in a heap, to output them in descending order, at most $N$ interchanges are required. This is achieved by executing the following routine: Output the number at node 1, replace it with $-\infty$, and restart the heaping routing at Step 2 with $k = 2$.

Each iteration restores the heap by filtering the $-\infty$ downward. After $N$ iterations, the heap will have been emptied out. Each iteration takes at most $\log_2 N$ interchanges so the total number of interchanges to output a heap in sequence is approximately $N \log_2 N$. Demonstrate your routine with the same numbers used in part (1a).

**Due:** 29 January
(Finding cycles in a graph)

2a. A graph, G = (V,E), is comprised of a set of n = |V| vertices and a set of m = |E| edges. It is possible to represent an undirected graph with an upper triangular n(n–1)/2 matrix, where the rows are numbered 1,...,n–1 and the columns 2,...,n. An entry of say, c_{ij} in cell (i,j), indicates that node i is connected to node j (j>i) and the edge has weight c_{ij}. When c_{ij} = 0, there is no edge connecting nodes i and j. The maximum number of edges in a graph is n(n–1)/2 which occurs when every node is connected to every other node. The density of a graph is \( r = \frac{m}{n(n–1)/2} \), where 0< \( r \leq 1 \). That is, for any i and j, there is a \( r \times 100\% \) chance that an edge exists between i and j. (Note, for a directed graph with no loops, an (n–1) \times (n–1) matrix can be used to store the graph.)

Write a program to generate a graph with n nodes and density \( r \) where each edge has weight 1. Store the graph in an upper triangular matrix. To construct the graph, draw a random number x uniformly distributed between 0 and 1 for each pair (i,j). If x \( \in (0, r) \) place an edge between i and j. Once you are finished, check to see that each node is connected to at least two other nodes. The graph should be connected (no isolated subgraphs) and each node should have degree 2 or greater, where the degree of a node is the number of edges incident to it. If necessary, add edges randomly from select nodes to assure these conditions are met. The resultant graph will have a density of approximately \( r \).

2b. A (cordless) cycle in a graph is a series of nodes (\( i_1, i_2, ..., i_r, i_{r+1} \)) such that each \( i_k \) (k = 1,...,r) is unique and \( i_1 = i_{r+1} \). Write a routine that identifies all cycles in the graph. This can be done with a depth-first search through the matrix that represents the graph. Start with row 1, cell (1,2). If there is a 1 in this cell, go to row 2 and check cell (2,3) (then cell (2,3), etc.). If there is a 1 in this cell, go back to row 1 and check whether there is a 1 in cell (1,3). If so, there is a cycle 1→2→3→1. In any case, you must also check cell (2,4) and if there is a 1 in this cell, check (1,4) for the cycle 1→2→4→1. Continue this logic identifying all cycles; then go back to row 1 and begin again with cell (1,3). After all checks are made, go to row 2 and begin again with cell (2,3). In fact, it is only necessary to examine a row if all the entries in the column above the first nonzero entry in the row are zero. For row k, for example, this would be the case if there were no edges from node i to node k (i < k) in the graph. Repeat procedure until all unique cycles have been identified.

For parts (a) and (b), provide output for two cases, the first where \( n_1 = 10 \) and \( r_1 = 0.1 \) and the second where \( n_2 = 20 \) and \( r_2 = 0.05 \).

Due: 12 February
References

Preprocessing


Lagrangian Relaxation


Column Generation Techniques -- Branch and Price


Polyhedral Theory


Branch and Cut Techniques


*Separation Algorithms*


1) Write a depth-first branch and bound code for a zero-one mixed integer linear program (MILP) using OSL as the LP solver. Assume that the input is an MPS file. Your code should be able to branch in either direction (i.e., $x_j = 1$ or $x_j = 0$) and start at any initial point. Therefore, a second input file should include data defining an initial point as well as upper and lower bounds on the objective function value (these might be $+\infty$ and $-\infty$).

Solve problem EXIMDL2.f in the OSL library and the sample problem provided in class. Compare your results and solution time with the MIP provided by OSL.

Due: January 28

2) Write an algorithm using Lagrangian relaxation to solve the flow problem with budget constraints on page 332. Investigate the different relaxations mentioned. Use OSL or CPLEX to solve the subproblems. Generate your own instances (say 10 for each case) for $n = 10$, $b = 100$, $c_{ij}$ a random number in the interval [1,10], and $t_{ij}$ in the interval $\left[0.05 \frac{b}{n} - 0.2 \frac{b}{n}\right]$.

Provide a table that lists the dual solution, the iteration on which the best solution was found, the optimal solution, and the percent gap between the two. Indicate which stopping rule you used and how the multipliers and step size were adjusted.

3) Write a program that randomly generates zero-one knapsack problems. Input should include the number of variables, $n$, and the range of the $a$ and $b$ parameters. The program should write out in algebraic form the corresponding inequality: $a_1 x_1 + a_2 x_2 + \cdots + a_n x_n \leq b$.

3.1) Write a subroutine that identifies all minimal dependent sets, $C$, and generates the corresponding valid inequalities.

3.2) Now find the extensions of $C$, denoted by $E(C)$, and update the inequalities.

3.3) As an alternative to part (3.2), use the lifting procedure to strengthen the inequalities found in part (3.1).

3.4) As an alternative to part (3.3), use Proposition 2.6 on page 268 of Nemhauser and Wolsey to find all valid inequalities of the form

$$\sum_{j \in N-C} \alpha_j x_j + \sum_{j \in C} x_j \leq |C| - 1$$

where $N = \{1, \ldots, n\}$.
RESEARCH PROJECT

Each of you will select a topic related to integer programming or combinatorial optimization, and conduct an in-depth study covering its origins, developments, and current status. This will involve extensive library research; your findings should be documented in a report with the basic references cited (see Bibliography for standard format). If appropriate, a representative computer program should be written to demonstrate the methodology being investigated.

In selecting a topic, it is important not to make it too broad or too narrow. For instance, decomposition techniques would be inappropriate because too much has been written on it to possibly distill for a class report. Narrowing the scope to Benders' decomposition method for nonlinear problems might be more appropriate. Before beginning, please see me for approval of the topic.

Background reading should include at least five (5) articles. In writing your report, you should think about what you have read, and provide your personal opinions about the presentation and usefulness of the work. You should not simply repeat what the authors have said. The article below by Eilon is a good model to follow.

Here are some general points to consider when reading about a particular study:

1) What is the general purpose of the article, who it is intended for, and why is the topic important?

2) What are the main results?

3) Indicate the modeling technique used (linear programming, simulation, etc.), and the experimental methodology followed. Was the analysis sufficient?

4) What do you think is the main contribution of the article? How is the work unique? Who might benefit from the results? Practitioners, researchers, managers, etc.?

5) What are the weaknesses and strengths of the work? How might it have been improved? What are your recommendations for future work in this area?

Reference