Computational Intractability	2010/4/8, rev. 2011/4/14
Lecture 1	
Professor: David Avis	*Scribe:Shohei Nishida*

1 Integer programming

There are many important practical problems that can be formulated as integer programs. In fact, the whole class of NP-complete problems have such a formulation. This means that we cannot expect polynomially fast algorithms to exist for integer programming. Nevertheless, important theoretical advances, coupled with fast computer processors, have opened up new possibilities for finding optimum solutions via integer programming.

1.1 Formulating integer programs

Integer programs are just linear programs for which some or all variables are constrained to be integers. The simplest has a single constraint and binary variables and is called the *knapsack problem*: The input data are two integer sequences, $a_1, a_2, \cdots, a_n$ and $c_1, c_2, \cdots, c_n$ and and an integer b.

$$\begin{aligned} max \;\; z &= \sum_{j=1}^{n} c_j x_j \\ \sum_{j=1}^{n} a_j x_j &\leq b \\ x_j &= 0 \; or \; 1, \quad j = 1, 2, ..., n \end{aligned} \tag{1}$$

The general form is given by an m by n matrix A, an n-vector c and an m-vector b. Here m is the number of constraints, and n is the number of decision variables.

$$\begin{aligned} max \;\; z &= c^T x \\ Ax &\leq b \\ x &\geq 0 \;\; integer \end{aligned} \tag{2}$$

2 Examples

2.1 Sushi problem

There are n pieces of sushi. Each of them has cost, a_i yen, and enjoyment c_i. Unfortunately we have only b yen.We want to maximize total enjoyment with total cost at most b yen. Which pieces of sushi should we buy? Indeed, this is exactly the knapsack problem formulated as (1).

2.2 Two machine scheduling problem

Given n jobs that have processing time $t_1, t_2, \cdots, t_n$,and finish time T. We have two computers and a job can be scheduled to either computer. we want to find a schedule such that all jobs are completed by time T, if such a schedule exists. One way to do this is to find the maximum assignment of work to computer 1 that can be done by time T. This can be achieved by the following integer program, which is a special case of the knapsack problem where each $a_j = c_j$.

$$\begin{aligned} max \;\; z &= \sum_{j=1}^{n} t_j x_j \\ \sum_{j=1}^{n} t_j x_j &\leq T \\ x_j &= \begin{cases} 1 & \text{if job } j \text{ is scheduled on machine 1} \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

2.3 Travelling salesman problem

There are n cities that a salesman has to visit. We are given a_{ij}, the distance between point i and point j, for each pair of cities i, j. Note that a_{ij} is not necessarily equal to a_{ji}. For example, the flying time from Tokyo to Vancouver is not the same as from Vancouver to Tokyo. The problem is to find the minimum length tour visiting each city once and returning to the starting point. Here is an integer programming formulation. We can model the problem as a complete graph with edges in each direction between each pair of cities.

$$\begin{aligned} min \; z &= \sum_{1 \leq i \neq j \leq n} a_{ij} x_{ij} \\ \sum_{i \neq j} x_{ij} &= 1 \quad j = 1, 2, ..., n \qquad (3) \\ \sum_{j \neq i} x_{ij} &= 1, \quad i = 1, 2, ..., n \qquad (4) \\ \sum_{i,j \in S} x_{ij} &\leq |S| - 1 \; for \; all \; S \subset \{1, 2, ..., n\}, 2 \leq |S| \leq n - 1 \qquad (5) \\ x_{ij} &= \begin{cases} 1 \text{ use edge from i to j} \\ 0 \text{ otherwise} \end{cases} \end{aligned}$$

To understand the meaning of the constraints (5) first consider the problem without these constraints. The meaning of the constraints (3) is that there must be some incoming edge to each city j. Similarly, the meaning of the constraints (4) is that there must be some outgoing edge for each city i.

Without the constraints (5) we may have short cycles. For example, with $n = 6$ cities, we could use edges 1-2-3-1 and 4-5-6-4. This route satisfies all other constraints, but is not a travelling salesman tour. Indeed, it violates (5) with $S = \{1, 2, 3\}$ and $S = \{4, 5, 6\}$. For this reason the constraints (5) are called subtour elimination constraints.