Integer Programming and Combinatorial Optimization
Slide set 2: Computational Complexity

Katta G. Murty Lecture slides

Aim: To study efficiency of various algo. for solving problems, and to classify algos. as **good** abd **bad**. Also, to classify problems into **easy** and **hard**.

Origin of subject: Goes back to 1960's.

Size of a problem instance: Bigger instances take more effort to solve in general. Need to develop a measure of how large an instance is.

Consider 2 instances of a system of 2 eqs. in 2 unknowns.

$$\begin{aligned} x_1 + x_2 &= 2 \qquad & 2937560086x_1 - 97635004389x_2 &= 1132003058 \\ -2x_1 + x_2 &= -1 \qquad & 8790204137x_1 - 713255436859x_2 &= 950786341 \end{aligned}$$

Right instance has lerger nos. than left, hence takes more work. So, should consider right instance to be larger than the left.

All problems involve data (we assume integral), stored in binary form in computer. Commonly used measure of largeness, **Size** $=$ amount of memory space to store data; i.e., **total no. binary digits in data** $\approx \Sigma(1 + \log_2(1 + |a|)$: over data elements a).

Running time of algo. proportional to **computational effort** (measured in terms of additions, multiplications, divisons & comparisons). This expected to $\uparrow$ with size, but problems with same size take different times because of differences in data. How to develop a measure of running time as a function of size? 3 approaches.

Empirical analysis: Solve large no. of representative instances of various sizes, & fit a function for running time in terms of size.

Used very much in practice, but measure depends on programming skill, & patterns of data in examples solved. Not suitable for developing a mathematical theory.

Avarage case analysis: Assumes a probability distribution for data, & derives asymptotic expected running time in terms of size by statistical arguments.

Analysis highly complicated even under simple distributions. Also, not clear which distributions are representative.

Worst case analysis: Determines an upper bound on running time in terms of size, for all possible data. So, gives guaranteed max. running time.

Basis for computational complexity theory. Drawback: depends on pathological instances which may be very rare.

Worst case complexity of an algo. is $O(n^\beta)$ where $n =$ size $\Rightarrow$ running time $\leq \alpha n^\beta$ where α, β are constants. In this case algo. said to be **polynomially bounded**.

Decision problems: Those for which answer is **yes** or **no**. **NP-completeness theory** deals with these.

Each opt. problem has a **decision problem version** (also called **recognition version, or feasibility version**).

TSP Decision problem: Given n, integer cost matrix $C_{n\times n}$, integer α; is there a tour with cost $\leq \alpha$?.

An opt. problem, & its decision problem version are equivalent in terms of being poly. bounded.

Problem reductions & Problem transformations

Problem X_1 **polynomially reducible** to problem X_2 if $\exists$ an algo. for X_1 using as a subroutine an algo. for X_2, calling it a polynomial no. of times (poly. in size of X_1). **S. Cook (1971)** introduced it.

X_1 **polynomially transforms** to X_2 (a special type of poly. reduction) if X_1 can be transformed with at most a poly. blow-up in size into X_2.

If X_1 poly. reducible (transforms) to X_2 & X_2 poly. bounded, so is X_1 (X_2 is at least as hard as X_1 in terms of polynomial solvability).

Example: Set packing: Given $A_{m \times n} = (a_{ij}), e = (1, \ldots, 1)^T \in R^n$, integer $c \in R^n$ and α $\exists x \in R^n$ feasible to: $Ax \leq e \quad cx \geq \alpha \quad x_j$ binary $\forall j$.

Node packing: Same as above, but each row of A has exactly 2 nonzero entries of 1.

Theorem: Set packing polynomially transforms to node packing.

$P, NP, NP-$complete, $NP-$hard classes

P = class of all problems solvable by a poly. bounded algo. Criterion of **poly. time** as characterization of a **good algo.** proposed by J. Edmonds, A. Cobham in early 1960's. Examples: LP, linear constraints, etc.

Certificate for a decision problem: For a decision problem with yes-answer, it is info. that can be used (by a certificate checking algo.) to verify that "yes" is correct answer in poly. time.

For TSP-decision problem, a tour with cost $\leq \alpha$ is a certificate.

NP: class of decision problems whose "yes-instances" have a certificate & a certificate checking algo. Examples: all problems in P, 0-1 IP, pure & MIP, TSP, etc.

Certificate of optimality or **good characterization**: Information that can be used to check optimality in poly. time. Example: LP, a certificate is a primal & dual pair of opt. sols.

NP-hard: Class of problems (not necessarily in NP) s. th.

every problem in NP polynomially rduces to it.

NP-complete: $NP \cap NP-$hard.

S. Cook (1971) introduced concepts of NP, NP-complete, NP-hard & poly. reducibility by proving: **Theorem:** Satisfiability problem is NP-complete.

0-1 IP version of satisfiability: Data: $N = \{1, \dots, n\}$. Given m pairs of subsets of N, (S_i^+, S_i^-), $i = 1$ to m. Is there a feasible sol. to: $\Sigma_{j \in S_i^+} x_j + \Sigma_{j \in S_i^-}(1 - x_j) \geq 1,$ for $i = 1$ to m; x_j binary $\forall j$.

How to show a problem X_2 is NP-complete: If $X_2 \in NP$ and a known NP-complete problem X_1 polynomially reduces (or transforms) to X_2, then X_2 is NP-complete.

Example: Consider **Subset sum:** Data: positive integers $d_0; d_1, \ldots, d_n$. Is there a sol. to: $\Sigma_{j=1}^{n} d_j x_j = d_0$ all x_j binary ?

Set partitioning feasibility: Data: $A_{m \times n} = (a_{ij})$, $e = (1, \ldots, 1)^T \in R^n$. Is there a sol. to: $Ax = e,\quad x$ binary?

Theorem: Given that set partitioning feasibility is NP-complete, so is subset sum.

Reference: M R Garey & D S Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, 1979.

Differences Between LP and IP Models:

LP	IP
1. Theoretically proven nec. and suff. optimality conditions exist. Useful to check whether a given feasible solution optimal	No known opt. conds. to check whether a given feasible sol. is opt., other than to compare it with every other feasible solution implicitly or explicitly.
2. Algos. are algebriac methods based on opt. conds.	All existing methods are enumerative methods based on partial enumeration.
3. Excellent software packages available. Very large models can be solved within reasonable times using them.	Performance of algorithms is very highly dependent on problem data. For most models, only moderate sized problems can be solved within reasonable times.