3.1 Algorithm complexity

Consider two alternative algorithms A and B for solving a given problem. Suppose A is $O(n^2)$ and B is $O(2^n)$, where n is the size of the instance. Let n_0^A be the size of the largest instance that can be solved in one hour with algorithm A on a given computer, and n_0^B be the corresponding size for algorithm B. Denote by n^A e n^B the size of the largest instances that can be solved in one hour on a computer that is 100 times faster. How large is n^A with respect to n_0^A and n^B with respect to n_0^B?

3.2 Size of a problem instance

Determine the size of an instance of the minimum cost spanning tree problem in terms of the number of nodes n and the number of edges m.

3.3 Complexity of longest path problem and shortest simple path problem

A *simple path* in a directed graph is a path that visits each intermediate node at most once. Consider the MAX-SIMPLEPATH problem: Given a directed graph $G = (N, A)$ with a rational length associated to each arc, and a pair of nodes s and t, find a simple path of maximum total length from s to t. Show that this problem is $\mathcal{NP}$-hard.

To do so, show that the following recognition version of MAX-SIMPLEPATH is $\mathcal{NP}$-complete.

> MAX-SIMPLEPATH-r: Given a directed graph $G = (N, A)$ with a rational length associated to each arc, a pair of nodes s, t, and an integer K, does there exist a *simple path* from s to t of length at least K?

[*Hint*: Consider the HAMILTONIAN-CIRCUIT-r problem where, given a directed graph, we have to decide whether it contains a Hamiltonian circuit. Describe a polynomial-time transformation from HAMILTONIAN-CIRCUIT-r to MAX-SIMPLEPATH-r.]

Now consider the MIN-SIMPLEPATH problem: Given a directed graph with rational arc costs, and a pair of nodes s, t, we look for a shortest simple path from s to t. Why is it also $\mathcal{NP}$-hard?

3.4 Complexity of a discrete optimization problem and ILP formulation size

Give an Integer Linear Programming formulation for the problem of finding a minimum cost spanning tree in an undirected graph $G = (N, E)$. How does the number of constraints grow with the number of the nodes $n = |N|$? Is there a direct relationship between the size of an ILP formulation (number of constraints and variables) and the difficulty of the corresponding problem?

SOLUTION

3.1 Algorithm complexity

Let n^A and n^B denote the size of the largest instances that can be solved in one hour with algorithms A and, respectively, B on a computer 100 times faster that the original one. Since, by definition of n_0^A, $(n_0^A)^2$ elementary operations are performed when using algorithm A on the original computer, $100(n_0^A)^2$ operations can be executed on the faster machine. Therefore, $(n^A)^2 = 100(n_0^A)^2$ and hence $n^A = 10n_0^A$. Similarly, for algorithm B, we have that $2^{n_0^B}$ elementary operations are performed on the original computer, $100\ 2^{n_0^B}$ are performed on the faster one. Then $2^{n^B} = 100\ 2^{n_0^B}$ implies that $n^B = n_0^B + log_2(100) < n_0^B + 7$.
Note that the above considerations about the number of elementary operations performed in one hour are true up to any multiplicative constant c.

To summarize, with the quadratic algorithm A we can solve instances that are 10 times larger, while with the exponential algorithm B we can solve instances that are larger by a small number of bits.

3.2 Size of a problem instance

An instance of the minimum cost spanning tree problem consists of a graph $G = (N, E)$ with an integer weight c_{ij} assigned to each edge $\{i, j\} \in E$. Without loss of generality we can assume that $m = |E| > n = |N|$, otherwise the graph is not connected and it admits no spanning tree.

Recall that, given any integer $i \in \mathbb{Z}$, $\lceil \log_2 i \rceil + 1$ bits are needed to code it in memory. The "+1" addendum takes into account the sign bit. For nonnegative integers, $\lceil \log_2 i \rceil$ bits suffice.

To describe any instance I of the minimum cost spanning tree problem, we need to store

- the values of n and m using $\lceil \log_2 n \rceil + \lceil \log_2 m \rceil$ bits,
- for each edge $\{i, j\} \in E$, the indices of the two nodes i and j, using $2\lceil \log_2 n \rceil$ bits,
- for each edge $\{i, j\} \in E$, the weight c_{ij} using at most $\lceil \log_2 c_{\max} \rceil + 1$ bits, where $c_{max} := \max_{\{i,j\} \in E} c_{ij}$.

In total we need

$$\lceil \log_2 n \rceil + \lceil \log_2 m \rceil + m(2\lceil \log_2 n \rceil + \lceil \log_2 c_{\max} \rceil + 1)$$

bits. Since $m > n$, the size of an instance I, denoted by $|I|$, is thus $O(m(\log_2 n + \log_2 c_{\max}))$.

It is worth pointing out that usually the number of bits needed to code any numerical value is considered as a constant. Indeed, if we use a 64 bit machine and we assume that all instances to be dealt with in practice satisfy $n \leq 2^{64}$, $m \leq 2^{64}$ and $c_{max} \leq 2^{64}$, the values of n, m and of each cost c_{ij} can be stored in a single memory work. Under this assumption, the size of an instance $|I|$ is $O(m)$. Recall that m is $O(n^2)$ for dense graphs and m is $O(n)$ for sparse graphs.

3.3 Complexity of longest path problem and shortest simple path problem

To prove that MAX-SIMPLEPATH-r is $\mathcal{NP}$-complete, we need to (i) verify that MAX-SIMPLEPATH-r belongs to $\mathcal{NP}$ and (ii) show that every other problem in $\mathcal{NP}$ can be reduced to MAX-SIMPLEPATH-r in polynomial time.

MAX-SIMPLEPATH-r clearly belongs to $\mathcal{NP}$. Indeed, (a) it is a recognition problem and (b) given any solution to it (sequence of nodes), we can verify in polynomial time (linear time w.r.t. to the number of nodes in G) that it is a path from s to t, it is simple, and it has a total cost $\geq K$.

To verify (ii), we show that the problem HAMILTONIAN-CIRCUIT-r, which is known to be $\mathcal{NP}$-complete, can be reduced in polynomial time to MAX-SIMPLEPATH-r.

HAMILTONIAN-CIRCUIT-r: Given a directed graph $G' = (N', A')$, does it contain a Hamiltonian circuit, i.e., a circuit visiting each node of G' exactly once?

Given any instance of HAMILTONIAN-CIRCUIT-r, it is easy to construct in polynomial time a special instance of MAX-SIMPLEPATH-r such that the answer to the HAMILTONIAN-CIRCUIT-r instance is yes if and only if the answer to the corresponding MAX-SIMPLEPATH-r instance is yes.

Let $G' = (N', A')$ be the directed graph of the given instance of HAMILTONIAN-CIRCUIT-r. Consider the particular instance of MAX-SIMPLEPATH-r defined by the directed graph $G = (N, A)$ where $N = N'$, $A = A'$, and each arc has a unit length. Moreover, we take $s = t$ as any node in N and $K = |N|$. Clearly, G' contains a Hamiltonian circuit if and only if G contains a simple path from s to t of length at least $|N|$. Indeed, any Hamiltonian circuit is a simple path from a node to itself with exactly $|N|$ arcs. Note that the polynomial-time transformation is very simple here because the graph in the HAMILTONIAN-CIRCUIT-r can be directly used in the MAX-SIMPLEPATH-r instance and we just need to appropriately select the nodes s, t and the value of K.

Let MAX-SIMPLEPATH be the problem of, given a directed graph and a pair of nodes s and t, finding a longest simple path from s to t. Since MAX-SIMPLEPATH-r is $\mathcal{NP}$-complete, MAX-SIMPLEPATH is $\mathcal{NP}$-hard. Indeed, MAX-SIMPLEPATH is at least as hard as MAX-SIMPLEPATH-r (an algorithm for the former problem could be used to solve the latter one) and is not in $\mathcal{NP}$ (it isn't a recognition problem).

To verify that the problem MIN-SIMPLEPATH is also $\mathcal{NP}$-hard, we just need to reduce in polynomial time MAX-SIMPLEPATH-r to the recognition version MIN-SIMPLEPATH-r. Given any instance of MAX-SIMPLEPATH-r defined by $G = (N, A)$, costs c_{ij}, nodes s, t and value K, it suffices to consider the instance of MIN-SIMPLEPATH-r with $G' = (N', A')$, c'_{ij}, K' where $N' = N$, $A' = A$, $c'_{ij} = -c_{ij}$ for all arcs $(i, j) \in A'$ and $K' = -K$.

3.4 Complexity of a discrete optimization problem and ILP formulation size

Consider any instance of the minimum spanning tree problem, which consists of an undirected graph $G = (N, E)$ with a cost c_{ij} assigned to each edge $\{i, j\} \in E$.

For each edge $\{i, j\} \in E$, we define a binary variable x_{ij} such that $x_{ij} = 1$ if $\{i, j\}$ is in the spanning tree and $x_{ij} = 0$ otherwise. Then the problem can be formulated as the following Integer Linear Program:

$$\begin{aligned}
\min \quad & \sum_{\{i,j\} \in E} c_{ij} x_{ij} & & \\
s.t. \quad & \sum_{\{i,j\} \in E} x_{ij} = |N| - 1 & & \text{(cardinality)} \\
& \sum_{\{i,j\} \in E : i,j \in S} x_{ij} \leq |S| - 1 & \forall S \subseteq N : 2 \leq |S| < |N| \quad & \text{(subtour elimination)} \\
& x_{ij} \in \{0,1\} & \forall i,j \in N. &
\end{aligned}$$

which contains a subtour elimination constraint for each proper subset $S \subset N$ of cardinality at least 2. For $S = N$ the corresponding subtour elimination constraint amounts to the cardinality constraint.

Note that, since there are $2^n - 2$ subtour elimination constraints, the size of the above ILP formulation grows exponentially with the number of nodes $n = |N|$ of G. Therefore, even if a polynomial-time algorithm for solving these ILPs would exist, it would require exponential time with respect to the size of the graph G.

This example shows that the size of an ILP formulation of a discrete optimization problem is not directly related to its inherent computational complexity. What really matters is the problem structure. Indeed, Prim's and Kruskal's simple greedy algorithms exploit the (matroidal) structure of the minimum spanning tree problem to find, for every instance, an optimal solution in polynomial time.