Stochastic optimization : Independent distributions

Consider the scenario based formulation for two stage stochastic linear programs with fixed recourse:

$$\min \quad c'x + \sum_{r=1}^{K} p_r q_r' y_r \qquad (*)$$

$$\text{s.t.} \quad Ax = b, \quad x \geq 0$$

$$T_r x + W y_r = h_r \quad \forall r = 1, \dots K$$

$$y_r \geq 0 \quad \forall r = 1, \dots K$$

For simplicity, assume you have l random terms in the second stage, each taking 2 possible values independently. Then, the number of scenarios $K = 2^l$. For example if $l = 10$, $K = 1024$, $l = 20$, $K = 1048576$. This is exponential in l. Here the size of the formulation in $(*)$ grows exponentially in l and hence it is apriori not clear if $(*)$ is efficiently solvable in the size of the input representation.

We will discuss some basics of computational complexity and how it might be used to study the complexity of optimization problems under uncertainty.

Basics of computational complexity

Linear programming is solvable in polynomial time in the input size while integer programming is NP-hard. This basically means that we have an efficient algorithm that given any arbitrary instance of a LP can solve it while it is unlikely that we will have an efficient algorithm that given any arbitrary instance of a IP will solve it.

We will concretize these ideas by describing some of the key ideas of computational complexity

For example, suppose we consider a binary integer program, we can solve it in finite time by enumerating all solutions, but this is not efficient.

Size

Let Σ be a finite set (typically $\{0,1\}$) called the alphabet where the elements of Σ are the letters.

Σ^* = Set of all finite strings (words) of letters

$\Sigma = \{0,1\}$ $\quad \Sigma^* = \{\phi, 0, 1, 00, 01, 10, 11, 000, \ldots\}$

The size of a word is the number of letters in the word counting multiplicities.

Say $\alpha = p/q$ is a rational number where $p, q \in \mathbb{Z}$ (integers) are relatively prime, $q \geq 1$. Then,

$$\text{Size}(\alpha) = \underbrace{1}_{\text{bit to store sign}} + \underbrace{\lceil \log_2(|p|+1) \rceil}_{\text{bits to store } p} + \underbrace{\lceil \log_2(|q|+1) \rceil}_{\text{bits to store } q}$$

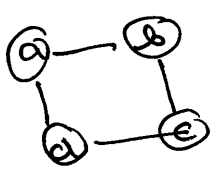
$p = 7$ $\quad \underbrace{111}_{3 \text{ bits}}$ $\qquad p = 3$ $\quad \underbrace{11}_{2 \text{ bits}}$

To represent a linear program $\min\{c'x \mid Ax = b, x \geq 0\}$ where $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ are rational

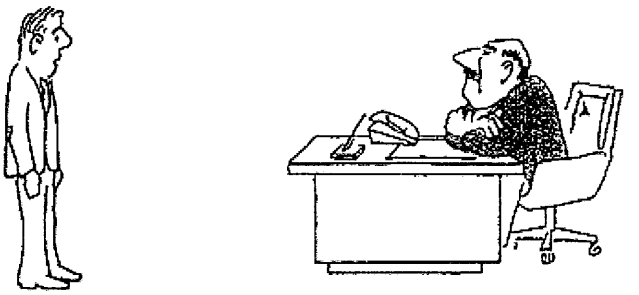
$$\text{Size}(LP) = O((mn+m+n)\log_2 U)$$

where U is the largest absolute entry in the input representation of A, b, c. Note this is polynomial in n, $m \leq \log_2 U$.

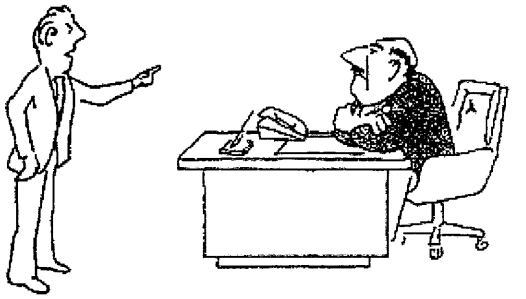
Similar approaches can be used to represents graphs.



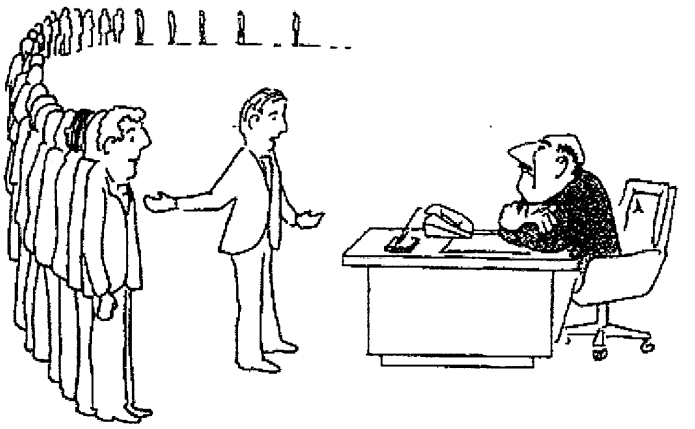
$$\underbrace{(\{a,b,c,d\}, \{\{a,b\}, \{b,c\}, \{c,d\}, \{a,d\}\})}_{\text{polynomial size in number of nodes \& edges}}$$



"I can't find an efficient algorithm, I guess I'm just too dumb."



"I can't find an efficient algorithm, because no such algorithm is possible!"



"I can't find an efficient algorithm, but neither can all these famous people."

Problem (We focus on decision problems with an answer YES or NO)

A problem is any subset Π of Σ^*.

(*) Given a word $x \in \Sigma^*$, does x belong to Π?

Example:

1) Given a system $Ax \leq b$ of linear inequalities, does it have a solution? (Here Π would be all (A,b) with a solution)

2) Given a system $Ax \leq b$ of linear inequalities, does it have an integral solution?

3) Given an undirected graph, does it have a perfect matching? (Here Π would be undirected graphs with perfect matching)

4) Given an undirected graph, does it have an Hamiltonian cycle (visit each node once exactly & form a cycle)?

An instance of a problem is a concrete input x.
Algorithm solves problem Π if for any instance x it returns the correct output YES or NO. The running time is the number of operations (steps) it takes to solve it for the input. and we can compute the maximum running time over all inputs of a size n.

Class P : Collection of all decision problems that are solvable in polynomial time in the input size. Namely, there is an algorithm with running time $T(n) = O(n^k)$ for some integer k for every instance of size n that decides whether or not $X \in \Sigma^*$ belongs to Π

Class NP : Collection of all decision problems for which each input $X \in \Sigma^*$ that belongs to Π (YES answer) has a polynomial time checkable certificate of the correctness of the answer.

Problems 1), 2), 3) and 4) are in NP.

Only problems 1) and 3) are in P.

For example, consider:

Given an undirected graph, is it non-Hamiltonian?

No such polynomial time checkable certificate is known for a YES answer to this question.

Here polynomial time checkable certificate means checking in time polynomial in the input size.

Clearly $P \subseteq NP$ (simply solve the problem & the solution provides a certificate verifiable

NP (nondeterministic polynomial time): The proof need not be found in polynomial time in input size

Open question $P = NP$? $P \neq NP$?

Optimization problem

Optimization problems can be transformed to decision version of problems as follows:

Consider $\min\{f(x) : x \in \overline{X}\}$ where $f(\cdot)$ is a rational function over $\overline{X}$.

Decision problem:

Given a rational number r, is there an $x \in \overline{X}$ with $f(x) \leq r$? (*)

Suppose we have an upper bound B on the size of the minimum value (being proportional to sums of logarithms of numerator & denominator, then we can find the optimum value by solving (*) for $O(B)$ choices of r by binary search.

Example: Find the minimum length TSP tour on a graph with nonnegative integer lengths c_{ij} $\forall (i,j) \in E$. Clearly the minimum value is bounded by $\sum_{(i,j)\in E} c_{ij}$ & we can do bisection search in $O\left(\log_2\left(\sum c_{ij}\right)\right)$ steps.

Reduction

Π_0 is polynomial time reducible to Π_1

$\Downarrow$

We solve an arbitrary instance of Π_0 by using a polynomial number of calls to a blackbox that solves Π_1 plus a polynomial number of standard additional computational operations

Denote it as, $\Pi_0 \leq_p \Pi_1$

Important implications

1) Suppose Π_1 is solvable in polynomial time.
   Then, Π_0 can be solved in polynomial time

2) Suppose Π_0 cannot be solved in polynomial time.
   Then, Π_1 cannot be solved in polynomial time.

You can find other types of reductions in the literature. For example there is a polynomial time algorithm which given any instance X_0 of Π_0 outputs an instance X_1 of Π_1 such that X_0 is a YES instance of Π_0 if and only if X_1 is a YES instance of Π_1 (one call).

NP - complete

The hardest problems in NP are NP-complete

A problem is NP-complete if:

(1) Problem is in NP

(2) Every problem in NP is reducible to it

To show a problem Π_1 is NP-complete

1) Show Π_1 belongs to NP.

2) Select a problem Π_0 that is NP-complete (Sometimes straightforward, sometimes creative)

3) Reduce Π_0 to Π_1 in polynomial time.

This work since $\Pi_0^* \leq_p \Pi_0 \leq_p \Pi_1$ for all $\Pi_0^* \in NP$.

We need reduction from only one problem. There are many choices - satisfiability, partition, vertex cover, ...

When a problem satisfies (2) but not necessarily (1), the problem is referred to as NP hard.

These problems are atleast as hard as NP complete problems.

For example: Testing whether a matrix is completely positive, namely $\exists\ v_i \in R_+^n$ such that $A = \sum v_i v_i'$ is known to be NP-hard but it is not known if it is NP complete (namely lies in NP) as of today.

Example

One problem that is known to be NP- complete is the PARTITION problem.

PARTITION : Given a set of positive integers $\{a_1, \ldots, a_n\}$, is there a subset $S \subseteq \underbrace{\{1, \ldots n\}}_{N}$ such that $\sum_{i \in S} a_i = \sum_{i \in N \setminus S} a_i$?

We now consider the 0/1 knapsack problem.

KNAPSACK : Given a set of positive integers $\{w_1, \ldots, w_n\}$ and $\{p_1, \ldots, p_n\}$, positive integers W & P, is there a subset $S \subseteq N = \{1, \ldots n\}$ such that $\sum_{i \in S} w_i \leq W$ and $\sum_{i \in S} p_i \geq P$

Theorem : KNAPSACK is NP- complete

Proof : Verifying the problem is in NP is trivial. We do a reduction from the PARTITION problem. as follows. Given an instance of the PARTITION problem, set $w_i = a_i$, $p_i = a_i$, $W = \sum_i a_i/2$, $P = \sum_i a_i/2$

Then the KNAPSACK problem corresponds to checking if there is $S \subseteq N$ such that $\sum_{i \in S} a_i \leq \sum_{i \in N} a_i/2$, $\sum_{i \in S} a_i \geq \sum_{i \in N} a_i/2$

or equivalently if $\sum_{i \in S} a_i = \sum_{i \in N} a_i/2$? or equivalently if

$\sum_{i \in S} a_i = \sum_{i \in S} \frac{a_i}{2} + \sum_{i \in N \setminus S} \frac{a_i}{2}$ or equivalently $\sum_{i \in S} a_i = \sum_{i \in N \setminus S} a_i$?

Hence KNAPSACK is NP- complete.

a

Counting (enumeration) problems

In comparison to decision problems which asks if there exists a solution satisfying some property, the counting (or enumeration) problem asks for the number of solutions that satisfy a property.

Clearly such problems are atleast as hard as the decision version, since we can count the number of solutions to verify if it is > 0 or $= 0$ to answer the decision problem.

The reason that such a class was introduced was since there were problems where the decision version is efficiently solvable but the counting version is hard.

Example

1) Given an undirected graph (even bipartite graphs), count the number of perfect matchings.

2) Given a graph, count the number of Hamiltonian cycles.

3) Given a graph, count the number of spanning trees.

Note this class of problems, does not require us to list out all the solutions (which might be exponential in number) but merely determine how many are there (this can be represented in a polynomially bounded number of digits).

#P : Counting version of NP problems

#P ~~complete~~ complete : Every problem in #P can be reduced to it

Problem 1) is #P-complete though the decision version is easy.

Problem 2) is #P-Complete and the decision version is NP-complete.

Problem 3) is easy as is the decision version.

Again we perform reductions from #P complete problem that is known to a problem of interest.

Note that beyond NP-hardness, #P-hardness that is based on worst case analysis, it might be possible to approximate such problems using deterministic or randomized algorithms.

Note that for #P-hard & #P-complete problems it is unlikely to expect an efficient algorithm to solve it

Intractability issues from uncertainty (independence)

We start by focusing on sums of random variables that are independent and discrete.

Input: Independent two point random variables

(P_1) $\quad \bar{c}_i = \begin{cases} 0 & \text{w.p } 1/2 \\ a_i & \text{w.p. } 1/2 \end{cases}$ $\quad a_i$ the integer

b a positive integer

Output: Compute $P\left(\sum_{i=1}^{n} \bar{c}_i \leq b\right)$

Proposition: Problem (P_1) is # P-hard.

Proof: To show (P_1) is # P-hard, we use a polynomial time transformation to the problem of computing the number of feasible solutions to a knapsack problem which is # P-hard.

Knapsack

Input: Positive integers $a_1, \ldots a_n, b$

Output: Number of solutions $x \in \{0,1\}^n$ such that $\sum_{i=1}^{n} a_i x_i \leq b$

Let the output to # knapsack be $K(a,b)$.

Then $P\left(\sum_{i=1}^{n} \bar{c}_i \leq b\right) = \dfrac{K(a,b)}{2^n}$.

Hence (P_1) is # P-hard.

Note that one might think the hardness stems from the non-convexity of probability. However this is not the only reason as we see next (independence plays a role).

Input: Independent 2 point random variables

(P_2) $c_i = \begin{cases} 0 & \text{w.p } 1/2 \\ a_i & \text{w.p } 1/2 \end{cases}$ a_i +ve integers

b positive integer

Output: Compute $E(\sum_i \bar{c}_i - b)^+ = E(\max(0, \sum_i c_i - b))$

Proof:

$$E(\sum_i \bar{c}_i - b)^+ = E(\max(\sum_i \bar{c}_i, b)) - b$$
$$= \sum_{t=1}^{\infty} P(\max(\sum_i \bar{c}_i, b) \geq t) - b$$
$$= \sum_{t=b+1}^{\infty} P(\sum_i \bar{c}_i \geq t)$$

Hence, $E(\sum_i \bar{c}_i - b)^+ - E(\sum_i \bar{c}_i - (b+1))^+$

$$= \sum_{t=b+1}^{\infty} P(\sum_i \bar{c}_i \geq t) - \sum_{t=b+2}^{\infty} P(\sum_i \bar{c}_i \geq t)$$
$$= P(\sum_i \bar{c}_i \geq b+1)$$
$$= 1 - P(\sum_i \bar{c}_i \leq b) = 1 - \frac{K(a,b)}{2^n}$$

Hence, a polynomial number of calls to (P_2) solves #Knapsack. This implies (P_2) is #P-hard.

We use the fact that for $\bar{c} \in \{0,1,2,\ldots\}$, $E(\bar{c}) = \sum_{t=1}^{\infty} P(\bar{c} \geq t)$

We next discuss why 2 stage stochastic linear programming is #P-hard with independent discrete distributions.

This is based on a reduction from a network reliability problem.

S-t reliability

Input: Directed $G(V, E)$, $s, t \in V$, $s \neq t$, rational probability $p \in [0, 1]$ of each arc failing

Output: Probability that the graph has a directed path operational from s to t. (Say $R(p)$)

This problem is #P-hard even for $p = 1/2$ where it is equivalent to counting the number of subgraphs with a directed path from s to t.

Consider a 2 stage stochastic LP where we set $V' = V$, $E' = E \cup \{(t, s)\}$ and

$$\min_{x \in [0,1]} cx + E[Q(x, \tilde{q})] \qquad (*)$$

where $Q(x, q) = \min \sum_{(i,j) \in E} q_{ij} y_{ij} - y_{ts}$

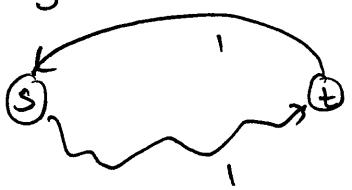
$$\text{s.t.} \sum_{j \in O(i)} y_{ij} - \sum_{j \in I(i)} y_{ji} = 0 \quad \forall i \in V'$$

$$0 \leq y_{ij} \leq x \quad \forall E \cup \{(t, s)\}$$

$$\tilde{q}_{ij} = \begin{cases} 0 & \text{w.p } 1/2 \quad \text{(works)} \\ 2 & \text{w.p } 1/2 \quad \text{(fails)} \end{cases}$$

In the 2nd stage, for a particular realization $\bar{q}$ if there is a directed path from s to t working, the optimal 2nd stage cost $= -X$

(Set $Y_{ij} = X \ \forall (i,j)$ on the path from s to t, $Y_{ts} = X$ and $Y_{ij} = 0$ for all other arcs)



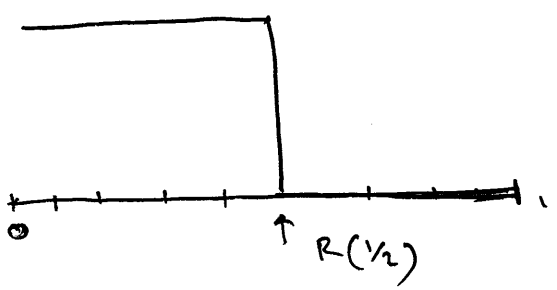
On the other hand if there is no directed path from s to t, the optimal 2nd stage cost $= 0$

(Set $Y_{ij} = 0 \ \forall (i,j) \in E'$, any flow $\delta > 0$ will have cost atleast $2\delta - \delta > 0$)

$\therefore \ E[Q(X,\bar{q})] = -X \overbrace{R(1/2)}^{\text{reliability with probability of } 1/2}$

For a fixed $X > 0$, evaluating the expected 2nd stage cost is #P-hard.

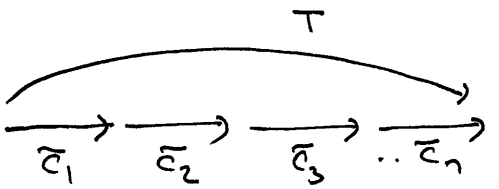
Also, the optimal solution $x^* = \begin{cases} 1 & \text{if } C < R(1/2) \\ 0 & \text{if } C > R(1/2) \\ [0,1] & \text{if } C = R(1/2) \end{cases}$



$R(1/2)$ can take up to $2^{|E|}$ values $\left\{0, \frac{1}{2^{|E|}}, \frac{2}{2^{|E|}}, \cdots \frac{2^{|E|}-1}{2^{|E|}}\right\}$

We can solve $O(|E|)$ problems of the type(*) to evaluate $R(1/2)$ using bisection search. This implies solving the problem is #P-hard.

Similarly, we can get hardness results for computing the expected project makespan in PERT networks & the probability of the makespan being less than or equal to T.



Makespan = $\max(T, \sum_i c_i)$

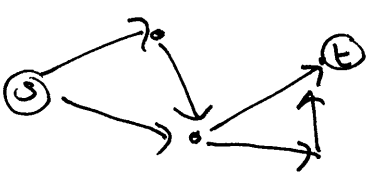
We now consider the special case of series parallel graphs (such as the one above) where this problem (computing mean or cdf) can be efficiently solved under restricted support of durations.

Definition, A PERT network is 2-terminal series-parallel if it can be reduced by a finite sequence of series & parallel reductions to a single activity.

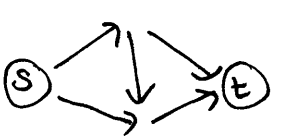
1) Series reduction : Head of activity is the tail of another activity with outdegree = indegree = 1

2) Parallel reduction : Two activities or more are in parallel with same head & tail.

Example



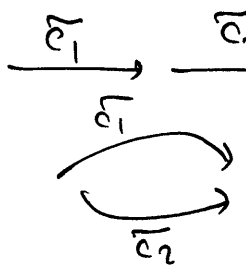
Series parallel



Not series parallel

In this case with restricted support (for simplicity say $\{0,1\}$), we can efficiently compute the expected makespan & cdf.

Note that with n activities, the makespan takes values only in the range $\{0,1,\ldots,n\}$.



$\bar{c}_1 + \bar{c}_2$ (Convolution)

$\max(\bar{c}_1, \bar{c}_2)$ (Product)

$$P(\bar{c}_1+\bar{c}_2=0) = P(\bar{c}_1=0)\,P(\bar{c}_2=0)$$
$$P(\bar{c}_1+\bar{c}_2=1) = P(\bar{c}_1=0)\,P(\bar{c}_2=1) + P(\bar{c}_1=1)\,P(\bar{c}_2=0)$$
$$P(\bar{c}_1+\bar{c}_2=2) = P(\bar{c}_1=1)\,P(\bar{c}_2=1)$$

$$P(\max(\bar{c}_1,\bar{c}_2)=0) = P(\bar{c}_1=0)\,P(\bar{c}_2=0)$$
$$P(\max(\bar{c}_1,\bar{c}_2)=1) = P(\bar{c}_1=1)\,P(\bar{c}_2=0) + P(\bar{c}_1=0)\,P(\bar{c}_2=1) + P(\bar{c}_1=1)\,P(\bar{c}_2=1)$$

$\vdots$

more generally

$$P\left(\sum_{i=1}^{\ell}\bar{c}_i = t\right) = P\left(\sum_{i=1}^{\ell-1}\bar{c}_i = t\right)P(\bar{c}_\ell=0) + P\left(\sum_{i=1}^{\ell-1}\bar{c}_i = t-1\right)P(\bar{c}_\ell=1)$$

$$P\left(\max_{i=1..\ell}\bar{c}_i = 0\right) = P\left(\max_{i=1..\ell-1}\bar{c}_i=0\right)P(\bar{c}_\ell=0)$$
$$P\left(\max_{i=1..\ell}\bar{c}_i = 1\right) = P\left(\max_{i=1..\ell-1}\bar{c}_i=1\right)P(\bar{c}_\ell=0) + P\left(\max_{i=1..\ell-1}\bar{c}_i=0\right)P(\bar{c}_\ell=1) + P\left(\max_{i=1..\ell-1}\bar{c}_i=1\right)P(\bar{c}_\ell=1)$$

At each step the range always lies in $\{0,\ldots,n\}$ & we reduce the number of arcs by a polynomial number of operations. Hence computing the distribution is efficiently possible.

Suppose you show that a problem is NP-hard. You should then give low priority to try and find an exact, efficient algorithm to solve the problem. Rather, it is wiser to focus on:

1) Find efficient algorithms to solve "useful" special cases of the problem.
2) Find practical algorithms (not theoretically efficient) to solve the problem.
3) Find efficient algorithms to "approximately solve" the problem

Remarks

A nice video on the Theory of Computation by Christos Papadimitriou at the SIMONS Institute provides a good philosophical level understanding of the need of computational complexity theory.

The #P-hardness for cdf of sums of independent random variables can be found in papers such as 'Allocating bandwidth for bursty connections' by Kleinberg, Rabani & Tardos (2000) The hardness results for stochastic linear programs is discussed in 'Computational complexity of stochastic programming problems' by Dyer & Stougie (2006) & 'A comment on computational complexity of stochastic programming problems' by Hanasusanto, Kuhn & Wiesemann (2016). The results on PERT networks can be found in 'Computational complexity of PERT problems' by Hagstrom (1988) and 'Scheduling under uncertainty: bounding the makespan distribution' by Möhring (2001).

The classical reference for complexity is Computers & intractability: A guide to theory of NP-completeness by Garey & Johnson (1979). Cook proved NP-completeness of the first problem in this class - the satisfiability problem (SA in the paper 'The complexity of theorem proving procedures' in 1971. The complexity class #P was introduced by Valiant in 1979 in his paper 'The Complexity of enumeration & reliability problems'. Applications of this in network problem is discussed in 'The complexity of counting cuts & probability that graph is