Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

6.438 Algorithms For Inference
Fall 2014

Recitation-6: Hardness of Inference

Contents

1 NP-Hardness Part-II **3**

1.1 NP - A class of decision problems 3

1.2 Reductions between decision problems and optimization problems 3

2 The complexity of Inference-doing procedures **6**

2.1 List of Hardness results in Inference 6

2.2 Complexity class #P (Sharp-P) 6

2.3 Hardness of MAP 7

2.3.1 Another Hard problem: Maximum Independent Set (MIS) 7

2.3.2 Reduction of MIS to MAP: 7

2.4 Hardness of Marginal Probability Queries 8

2.4.1 Reduction from #Independent Sets to Marginal Probability Queries 8

2.5 Hardness of converting U. graphical model description to D. graphical model description 9

2.5.1 Reduction from Partition function to graphical model conversion . . 10

Preface

Most of the material presented in this recitation can be regarded as additional learning. The material is not directly related to things we will do in the course (though it may help to improve understanding), and it will not be tested on exams.

From the point of view of understanding in the course, it is important for students to be aware of the hardness results mentioned in section 2.1. However, for also understanding the proofs of the same, reading the entire material is recommended.

1 NP-Hardness Part-II

Note: It is recommended to read the introductory section on NP-hardness in recitation-4 notes, before reading this section.

1.1 NP - A class of decision problems

Last time, we introduced NP informally as a class of problems. We will make our definitions slightly more precise, by looking at NP as a class of decision problems, since that is how they are defined in the CS literature.

What is a decision problem? It is a question with a single Yes/No answer. Examples - "Is the number 5 prime?" or "Does there exist a chordal graph containing a 5 cycle?". The class NP is then defined as the class of *decision problems* which can be solved in polynomial time by a non-deterministic Turing machine. As before, we use an alternate characterization which does not involve Turing machines - NP is the class of decision problems for which Yes answers are easy to verify, given some 'additional information'. This additional information is often called a *certificate* or *witness* for the problem. We will understand it through some examples.

Consider the following problem which is in NP:

P1: ***Compositeness testing:*** Given a number N, is the number composite?

This problem is in NP, because if the correct answer to the problem is Yes (i.e. if N is really composite), there is a certificate which allows us to verify this quickly. This certificate is simply a (non-trivial) factor of N; because given such a factor, say f, we can easily check by division whether N/f is an integer, and thus verify that N is composite. Note that this verification can be done in polynomial time.

1.2 Reductions between decision problems and optimization problems

Consider the following 2 problems:

P2: ***Clique decision problem:*** Given a graph G, and a positive integer k, does G have a clique of size $\geq$k?.

P3: ***Maximum clique problem:*** Given a graph G, find the largest clique in G. (i.e. give the list of vertices which form the largest clique)

Note: if G has multiple largest cliques, an algorithm for P3 only needs to output one of them to be considered correct.

We make some observations about these 2 problems. First, note that P2 is a decision problem, while P3 is not. We further observe that P2 is in NP; since YES answers to the question are easy to verify (any clique of size $\geq$k serves as a witness). Thirdly, we note that P2 is an easier problem than P3, in the sense that if we could solve P3, then we would immediately have the answer to P2 for all positive integers k.

We now ask the reverse question - if we can solve P2 for all graphs G and positive integers k, can we also solve P3?

The answer, perhaps surprisingly, is yes, although we need to do some extra work. The method is described below:

Reduction from P3 to P2 (optional): Suppose we have an oracle[1] for solving problem P2. We first query the oracle on our graph, for each positive integer from 1 to $|V|$. Thus, we learn the size of the maximum clique in G, call it k^*. Now, remove a single vertex of G, say vertex 1, and call the resulting graph G'. Use the oracle for P2 to answer the following query on G': "Does G' have a clique of size $\geq k$?" If the answer is No, it means that vertex 1 was an essential part of our largest clique. We leave vertex 1 untouched and pick some other vertex in G, remembering not to go back to vertex 1. On the other hand, if G' still has a clique of size $\geq k^*$, this means we do not lose anything by discarding vertex 1. We remove it and continue to work with the smaller graph G'. By repeating this operation for each vertex of G, we can remove all non-essential vertices, until we are left with only the clique of size k^*.

In the above method, we made at most $2N$ queries to the oracle, where $N = |V|$. In practice, this means that if we had some algorithm for solving P2, we would have to run the algorithm $2N$ times. Thus, it appears that having an algorithm for P2 would give us a 'slower' algorithm for P3. However, if we consider any algorithm that runs in polynomial time to be *efficient*, then having an 'efficient' algorithm for P2 also gives us an 'efficient' algorithm for P3. In fact, the method we have seen above is basically a polynomial-time reduction from P3 to P2. The above argument shows us that P2 and P3 are *polynomial-time equivalent* [2]; since P3 is polynomial-time reducible to P2, and P2 is also (trivially) reducible to P3. Informally, we may say that P2 and P3 are equally hard problems.

We now state an important fact without proof – **P2 is NP-Complete**. Since P2 is polynomial-time reducible to P3, **P3 is NP-hard**. From a practical standpoint, this means that we have almost no hope of finding an efficient algorithm for P3.

Remark 1: Recall the factorization problem: Given a number N, list down all its prime factors as well as the exponents of these factors. In the recitation-4 notes, we had mentioned that factorization problem is in NP, since it is easy to verify whether a given prime factorization is correct. However, this is not strictly correct, since the class NP is formally defined as a class of decision problems. Hence, for inclusion in NP, we need to consider a decision version of factorization:

P4: ***Decision version of Factorization problem*** Given a natural number N and

[1]An oracle is a machine which instantly answers queries to a certain problem

[2]Two problems A and B are said to be polynomial-time equivalent, if solving either one in PT gives a method for solving the other in PT.

another natural number k, does N have any non-trivial[3] factor larger than k?

Exercise: Verify that the decision version of factorization is in NP. What is the certificate for a Yes answer?

Challenge: Show that P4 is polynomial-time equivalent to the factorization problem.

Remark 2: The above discussion helps us understand why we don't lose anything by defining NP as a class of just decision problems, since they are often equivalent to the corresponding optimization problem. However, for the rest of our discussion, we will focus only on optimization problems. The distinction between decision problems and optimization problems is important from a computer science perspective, but it will not make any difference to what we do.

[3] A factor is considered trivial if it equals either 1 or the number itself

2 The complexity of Inference-doing procedures

2.1 List of Hardness results in Inference

We will be interested in the following propostions:

(a) Finding MAP assignment is:

  (i) NP-Hard on undirected graphs.
  (ii) NP-Hard on factor graphs.
  (iii) NP-Hard on directed graphs.

(b) Finding marginal probability of a node is:

  (i) #P-Hard on undirected graphs.
  (ii) #P-Hard on factor graphs.
  (iii) #P-Hard on directed graphs.

(c) Computing the partition function of an undirected graphical model is #P-Hard.

(d) Converting an undirected graphical model description to a directed graphical model description is #P-Hard.

We will discuss the proofs of most of these statements in detail. We will not prove the hardness results for directed graphs, although we will give some idea of how it can be done. Since many of these results involve the complexity class #P, we will first look at the definition of this class.

2.2 Complexity class #P (Sharp-P)

The class #P is a class of problems which count the number of certificates to a problem in NP. Thus, for the decision version of the clique problem (which is in NP), the corresponding problem in #P would be:

P5: ***#Clique Problem:*** Given a graph G and a positive integer k, how many cliques of size $\geq$k does G contain?

Clearly, knowing the answer to this would allow us to answer the question: "Does G contain a clique of size $\geq$k?"

In general, #P problems are at least as hard as the corresponding NP problems. Similar to the notion of NP-Hardness, a problem is said to be #P-Hard if every problem in #P is reducible to it in PT. Thus, solving a #P-Hard problem in PT would imply #P = P. Since the counting versions of *all* NP-problems are in #P, and are harder than the corresponding decision problems, #P=P would automatically imply P=NP. Thus knowing that a problem is #P-Hard gives us a very strong reason to believe that it is not solvable in polynomial time - an even stronger reason than knowing it is NP-Hard!

2.3 Hardness of MAP

We will prove that finding the MAP assignment on an undirected graph is NP-Hard, by giving an (informal) reduction to it from a well-known NP-Hard problem viz. the maximum independent set problem. Our reduction shall make use of distrubutions which factorize over maximal cliques, hence our hardness proofs will also imply the same hardness results for factor graphs. First, let us formally define the MAP problem for undirected graphs. :

P6: ***MAP Estimation over Undirected Graphs:*** Given a set of discrete random variables $\{X_1, X_2, \ldots, X_N\}$, with with $X_i \in \mathcal{X}$ for each $i \in [1..N]$; an undirected graph $G = (V, E)$, and functions $\phi_{X_C}(X_C)$ for each maximal clique C in G; such that the probability distribution of these random variables satisfies the condition: $\mathbb{P}_{X_1,..X_N}(x_1, ..x_N) \propto \prod_{C \in (\mathcal{C})} \phi_C(x_C)$. Find an assignment $(x_1^*, x_2^*, \ldots, x_N^*)$ which maximizes $\mathbb{P}_{X_1,..X_N}(x_1^*, ..x_N^*)$ among all possible assignments in $\mathcal{X}^{\mathcal{N}}$.

Note: If a distribution has multiple MAP assignments, an algorithm is considered correct if it returns any one of the MAP assignments.

2.3.1 Another Hard problem: Maximum Independent Set (MIS)

A set of vertices in a graph is called an Independent Set, if no two of the vertices in this set are connected in the graph. The Maximum Independent Set problem asks us to find the largest independent set in a graph. We define it formally below:

P7: ***Maximum Independent Set problem:*** Given a graph $G = (V, E)$, find a set of vertices $U \subseteq V$, such that:

(i) $\forall i, j \in U, (i, j) \notin E$.

(ii) U has maximum cardinality among all sets satisfying (i).

The independent set problem is closely related to the maximum clique problem. In fact, observe that a subset of vertices is an independent set if and only if it is a clique in the *complement graph.* (The 'complement graph' of G is a graph G' over the same set of vertices, having exactly those edges which are missing in G). Since building a complement graph is a straightforward operation, an efficient algorithm for solving the max-clique problem would allow one to efficiently solve the max-independent set problem, and vice versa. Hence, the **maximum-independent set problem (P7) is NP-Hard**.

2.3.2 Reduction of MIS to MAP:

Assume that we have an efficient algorithm to solve the MAP problem (P6). How can se use this to solve the Maximum Independent Set problem (P7)? In other words, given a graph G, we wish to construct a set of random variables and an undirected graphical model description over these variables (i.e. graph as well as potential functions), such that finding the MAP assignment of this distribution allows us to find the largest clique in G. In fact, we have already solved this problem in one of the homeworks! (Well, almost)

We saw in Hw Problem 2.1, given a graph G, how to build a graphical model description which assigns non-zero values only to subsets which are independent sets. In fact, we only used clique potentials upto size 2 (viz. nodes and edges) for solving this problem. We also had node potentials assigned to each node in the graph, such the probability of choosing an independent set was proportional to the product of potentials of the selected nodes. We now fix a constant positive value >1 for every node potential. Then, the independent set with the largest number of nodes has the highest probability in the distribution. Thus, finding the MAP assignment of this distribution directly gives us the largest independent set in graph G ! QED.

Remark 3: We have proved the hardness of computing MAP for undirected graphs and factor graphs. We might be tempted to use the same proof for showing hardness on directed graphs. Unfortunately, the proof does not work. (*Challenge*: Why?)

Remark 4: The easiest way to prove hardness of MAP (as well as marginal probability queries) for directed graphs is to model another NP-hard problem with them viz. 3-SAT. We won't go into it here since that would be covering too much material. However, those who have prior background in complexity theory and know about the 3-SAT problem are encouraged to try it.

2.4 Hardness of Marginal Probability Queries

First, let us define the marginal problem formally. This definition is very similar to the definition of MAP query on undirected graphical model.

P8: ***Marginal Probability Computation over Undirected Graphs:*** Given a set of discrete random variables $\{X_1, X_2, \ldots, X_N\}$, with $X_i \in \mathcal{X}$ for each $i \in [1..N]$; an undirected graph $G = (V, E)$, and functions $\phi_{X_C}(X_C)$ for each maximal clique C in G; such that the probability distribution of these random variables satisfies the condition: $\mathbb{P}_{X_1,..X_N}(x_1, ..x_N) \propto \prod_{C \in (\mathcal{C})} \phi_C(x_C)$.
Given an integer $j \in \{1, ..|V|\}$, find the marginal probability of node j viz. $\mathbb{P}_{X_j}(x_j)\, \forall x_j \in \mathcal{X}$.

We will prove that the task of finding marginal probability of an arbitrary node in a graph is #P-Hard, by reducing another #P-Hard problem to it - viz.the problem of counting the number of Independent Sets in a graph.

2.4.1 Reduction from #Independent Sets to Marginal Probability Queries

The reduction uses the same undirected graphical model that we used above. Given a graph $G = (V, E)$, we build an undirected graphical model with the same edge potentials as Hw problem 2.1. Instead of setting the node potentials to some positive value, we set them all to 1. Notice what this does to the probabilities of the sets - every independent set S now has equal probability. In fact, the unnormalized probability (i.e. the product of all the potentials) for every independent set equals 1, and equals 0 if the set is not independent.

Formally, let X_S denote the N-length binary vector which is the indicator vector for choosing the subset S of V, for every $S \subseteq V$ (where $N = |V|$). Then, we have:

$\bar{\mathbb{P}}(X_S) = \mathbb{1}(\text{S is an independent set in G})$

Let us write down the partition function for this distribution, viz.:

$$\begin{aligned}\mathbb{Z} &= \sum_{X_S \in \{0,1\}^N} \bar{\mathbb{P}}(X_S) \\ &= \sum_{S \subseteq V} \mathbb{1}(\text{S is an independent set in G}) \\ &= \text{Number of independent sets in G}\end{aligned}$$

This shows that computing the Partition function allows us to solve the Independent set counting problem, and thus computing partition function is #P-Hard! However, we have yet to prove that marginal queries are #P-Hard. We will make use of the following lemma to complete the proof:

Lemma 1 *The problem of computing the partition function of an undirected graph is polynomial-time (i.e. efficiently) reducible to the problem of computing the marginal probability of an arbitrary node.*

We will not prove this lemma here. (For a proof, see Hw problem 5.3). The lemma implies that if we had a PT algorithm to solve marginal queries, we could use it to compute the partition function of the graphical model in polynomial time. But as we saw above, computing the partition function is #P-Hard! Hence, marginal queries are also #P-Hard in general. QED.

Remark 5: For the above reductions, we used graphical models where the underlying distribution is not positive. One could therefore, conjecture that the MAP problem or marginal query problem might be easier on graphical models restricted to positive distributions. Unfortunately, this is not the case, as is easily seen by replacing '0' with a small positive quantity 'epsilon' in the potential functions used in the above reductions.

2.5 Hardness of converting U. graphical model description to D. graphical model description

We have seen in class how to convert an undirected graphical model to a directed graph. We did not, however, see a way to transform the potentials for undirected graph into potentials[4] for the directed graph. In fact, we will show that this omission was no accident - the problem is computationally intractable!

[4]By potentials for a directed graph, we mean the conditional probability function of each node given its parents.

As usual, we begin by stating our problem formally:

P9: ***Converting Undirected graphical model description to Directed graphical model description:*** Given an undirected graph $G = (V, E)$ and a set of potential functions $\phi_C(\mathcal{X}_C)$ over the maximal cliques of G; give a directed graph $G' = (V, E')$ over the same set of vertices, and a set of conditional probability functions $\mathbb{P}_{X_i|X_{\Pi_i}}(x_i|x_{\Pi_i})$ for every $i \in V$, such that the 2 graphical models represent the same probability distribution over $\mathcal{X}_V$.

2.5.1 Reduction from Partition function to graphical model conversion

The reduction is straightforward. Suppose, for a particular probability distribution over random variables $\mathcal{X}_1, ..\mathcal{X}_N$, we had both an undirected description (in terms of potentials), and a directed description in terms of conditional probabilities. Pick any assignment over $\mathcal{X}_1, ..\mathcal{X}_N$. Calculate the following 2 quantities:

$$\prod_{i\in[1..N]} \mathbb{P}_{\mathcal{X}_i|\mathcal{X}_{\Pi_i}}(x_i|x_{Pi_i}) = \mathbb{P}_{\mathcal{X}1,..\mathcal{X}_N}(x_1, ..x_N) \tag{1}$$

$$\prod_{C\in\mathcal{C}(\mathcal{G})} \phi_{\mathcal{X}_C}(x_C) = \frac{1}{\mathbb{Z}}\mathbb{P}_{\mathcal{X}1,..\mathcal{X}_N}(x_1, ..x_N) \tag{2}$$

Note that the LHS of both equations are easy to compute. If we divide the first equation by the second, we get the partition function on the RHS! Thus, solving P9 would give us an efficient procedure for computing the partition function of an arbitrary undirected graphical model, which we know is #P-Hard! Hence, P9 itself must be a #P-Hard problem. QED.

Remark 6: The reason why this conversion is hard is that the directed graphical model has much more information than the undirected model. In an undirected model, we can multply any of the potentials by a constant without changing the distribution. On the other hand, directed graphical model parameters are conditional probabilities, and they are unique for a given graph and a given probability distribution.

Remark 7: The other direction of this result is not true. That is, while converting undirected model parameters to directed is a #P-Hard problem, converting from directed to undirected is trivial. Think about how you can do this.

Remark 8: "Alternative proof": Here is another (simpler) proof that conversion is #P-Hard: Suppose we wish to compute the marginal probability of node j. Then we simply choose an ordering for conversion where j appears as the first node. Since j does not have any parents, $\mathbb{P}(\mathcal{X}_j)$ appears as one of the parameters in the directed model. Thus, doing this conversion would immediately give us the marginal of node j, solving a #P-Hard problem. QED.
Challenge: Is there any difference between this proof and the previous one?