CS120: Intro. to Algorithms and their Limitations Prof. Salil Vadhan

Active Learning Exercise 6: Reading for Receivers

Harvard SEAS - Fall 2021 *Nov. 4, 2021*

The goals of this exercise are:

- to develop your skills at understanding, distilling, and communicating proofs and the conceptual ideas in them,
- to practice reductions to SAT, and in particular how logic is useful for modelling problems

To prepare for this exercise as a receiver, you should try to understand the theorem statement and definition in Section 1 below, and review the material on Logic and Satisifiability covered in class on October 28 and November 2. Your partner sender will communicate the proof of Theorem 1.1.

1 The Result

In class, we saw how the (seemingly hard) problem of Graph k-Coloring can be efficiently reduced to CNF-Satisfiability (SAT). Although SAT also seems to be a hard problem (as we'll formalize in the last part of the course), this allowed all the effort put into SAT Solvers to solve many large k-coloring instances in practice.

In this exercise, you'll see a similar reduction for the *Longest Path* problem.

Input	: A digraph $G = (V, E)$ and two vertices $s, t \in V$
Output	: A *longest path* from s to t in G with no repeated vertices, or $\perp$ if no path from s to t exists

Computational Problem LongestPath

Actually, it will be more convenient to consider a version where the desired path length is specified in the input.

Input	: A digraph $G = (V, E)$, two vertices $s, t \in V$, and a path-length $k \in \mathbb{N}$
Output	: A path from s to t in G of length k and no repeated vertices, or $\perp$ if no such path from s to t exists

Computational Problem LongPath

If we have an efficient algorithm for LongPath, then we can solve LongestPath by trying $k = n, n-1, \ldots, 0$ until we succeed in finding a path. The $k = n$ case is known as the *Hamiltonian Path* problem, which is a special case of the notorious *Travelling Salesperson Problem (TSP)*. In the TSP, we have a salesperson who wishes to visit n cities (to sell their goods) in the shortest travel time possible. If we model the possible travel between cities as a directed graph, then Hamiltonian Path corresponds to the special case where all pairs u, v of cities either have a travel time of 1 (edge

(u, v) present) or a very large travel time (edge (u, v) not present). In such a case, the only way to visit all cities in travel time at most $n - 1$ is via a Hamiltonian Path.[1]

The reduction from LongPath to SAT is given as follows.

Theorem 1.1. *LongPath on a digraph with n vertices, m edges, and a path length k reduces to SAT in time $O(n^2k)$.*

2 The Proof

Constructing a SAT instance φ from a LongPath instance (G, s, t, k).

Converting a satisfying assignment α to φ into a LongPath solution P.

[1] Often in the TSP, it is also required that the salesperson return back to their starting city s. If we add edges of travel time 1 from all cities to s, then we see that Hamiltonian Path is also a special case of this variant of the TSP.

Correctness of the Reduction.

Runtime of the Reduction.

CS120: Intro. to Algorithms and their Limitations Prof. Salil Vadhan

Active Learning Exercise 6: Reading for Senders

Harvard SEAS - Fall 2021 *Nov. 4, 2021*

The goals of this exercise are:

- to develop your skills at understanding, distilling, and communicating proofs and the conceptual ideas in them,
- to practice reductions to SAT, and in particular how logic is useful for modelling problems

Section 1 is also in the reading for receivers. Your goal will be to communicate the *proof* of Theorem 1.1 (i.e. the content of Section 2) to the receivers.

1 The Result

In class, we saw how the (seemingly hard) problem of Graph k-Coloring can be efficiently reduced to CNF-Satisfiability (SAT). Although SAT also seems to be a hard problem (as we'll formalize in the last part of the course), this allowed all the effort put into SAT Solvers to solve many large k-coloring instances in practice.

In this exercise, you'll see a similar reduction for the *Longest Path* problem.

Input	: A digraph $G = (V, E)$ and two vertices $s, t \in V$
Output	: A *longest path* from s to t in G with no repeated vertices, or $\perp$ if no path from s to t exists

Computational Problem LongestPath

Actually, it will be more convenient to consider a version where the desired path length is specified in the input.

Input	: A digraph $G = (V, E)$, two vertices $s, t \in V$, and a path-length $k \in \mathbb{N}$
Output	: A path from s to t in G of length k and no repeated vertices, or $\perp$ if no such path from s to t exists

Computational Problem LongPath

If we have an efficient algorithm for LongPath, then we can solve LongestPath by trying $k = n, n-1, \ldots, 0$ until we succeed in finding a path. The $k = n$ case is known as the *Hamiltonian Path* problem, which is a special case of the notorious *Travelling Salesperson Problem (TSP)*. In the TSP, we have a salesperson who wishes to visit n cities (to sell their goods) in the shortest travel time possible. If we model the possible travel between cities as a directed graph, then Hamiltonian Path corresponds to the special case where all pairs u, v of cities either have a travel time of 1 (edge (u, v) present) or a very large travel time (edge (u, v) not present). In such a case, the only way to visit all cities in travel time at most $n - 1$ is via a Hamiltonian Path.[1]

[1] Often in the TSP, it is also required that the salesperson return back to their starting city s. If we add edges of travel time 1 from all cities to the starting city, then we see that Hamiltonian Path is also a special case of this variant of the TSP.

The reduction from LongPath to SAT is given as follows.

Theorem 1.1. *LongPath on a digraph with n vertices, m edges, and a path length k reduces to SAT in time $O(n^2k)$.*

2 The Proof

Let $G = (V, E)$, s, t, k be our instance of LongPath. Without loss of generality $k < n$, as any path of length larger than $n - 1$ must have repeated vertices. From this instance, we construct a SAT instance φ as follows.

Constructing φ. We will have $(k + 1)n$ variables $x_{i,v}$ for each $i \in [k + 1]$ and vertex $v \in V$. Intuitively, $x_{i,v} = 1$ will mean that the i'th vertex in the path is v.

For clauses, we include the following constraints:

1. At least one vertex is assigned to the i'th step in the path for each $i = 0, \ldots, k$: $\left(\bigvee_{v \in V} x_{i,v}\right)$
2. $v \neq w$ are not both assigned to the i'th step in the path for each $v \neq w \in V$ and $i = 0, \ldots, k$: $(\neg x_{i,v} \vee \neg x_{i,w})$
3. The path starts with s: $(x_{0,s})$
4. The path ends with t: $(x_{k+1,t})$
5. v is not assigned to both the i'th and j'th step in the path for each $v \in V$ and $i \neq j \in [k+1]$: $(\neg x_{i,v} \vee \neg x_{j,v})$
6. For each non-edge $(u, v) \in V \times V - E$ and $i \in [k]$, the i'th pair of vertices in the path is not equal to (u, v): $(\neg x_{i,u} \vee \neg x_{i+1,v})$

Above we have 2 clauses of size 1, $k+1$ clauses of size n, and $O(n^2k+nk^2+(n^2-m)k) = O(n^2k)$ clauses of size 2. (Recall that $k < n$.) By inspection, this collection of clauses can be constructed in time $O(n^2k)$.

Now, if we call a SAT oracle on φ, we will find out whether φ is satisfiable, and if so, we will also obtain a satisfying assignment α. So to complete the reduction, we will prove the following two claims:

Claim 2.1. *If there a LongPath solution on instance (G, s, t, k), then φ is satisfiable.*

Claim 2.2. *We can transform any satisfying assignment α to φ into a solution to LongPath on instance (G, s, t, k) in time $O(nk)$.*

Then, the following will be a valid reduction from LongPath to SAT:

```
LongPathViaSAT(G, s, t, k)
  Input  : A digraph G = (V, E), vertices s, t ∈ V, a path-length k
  Output : A longest path from s to t in G with no repeated vertices, or ⊥ if no path from
           s to t exists
Construct φ from G, s, t, k as described above;
Call the SAT Oracle on φ, obtaining α, which is either a satisfying assignment to φ or ⊥ if
  φ is unsatisfiable;
if α = ⊥ then return ⊥;
else
    Convert α to a LongPath solution P via Claim 2.3;
    return P;
```

Algorithm 1: LongPath by reduction to SAT

Claim 2.1 implies that we only return $\perp$ when there is no solution to the LongPath instance. Claim 2.3 implies that whenever we return a path P, it is a valid solution for the LongPath instance.

So let's prove the two claims.

Proof of Claim 2.1. Suppose there is a LongPath solution on instance (G, s, t, k). That is, there is a path P of length k, starting at s, ending at t, and with no repeated vertices. Let $(v_0, v_1, \ldots, v_k)$ be the vertices on the path P. Consider the following assignment α to the variables of φ:

$$\alpha_{i,v} = \begin{cases} 1 & \text{if } v_i = v \\ 0 & \text{otherwise.} \end{cases}$$

We verify by inspection that α satisfies all of the clauses of φ. It satisfies all clauses of Types 1 and 2 because for each $i = 0, \ldots, k$, we set $\alpha_{i,v} = 1$ for exactly one value of v (namely $v = v_i$). It satisfies Clause 3 and 4 and because P starts with s and ends with t. It satisfies all clauses of Type 5 because P has no repeated vertices. It satisfies all clauses of Type 6 because P only uses edges of G (i.e. $(v_i, v_{i+1}) \in E$ for $i = 0, \ldots, k$). □

Claim 2.3. *We can transform any satisfying assignment α to φ into a solution to LongPath on instance (G, s, t, k) in time $O(nk)$.*

Proof of Claim 2.3. Let α be any satisfying assignment to φ. Because α satisfies the clauses of Types 1 and 2, for each $i = 0, \ldots, k$, we have that $\alpha_{i,v} = 1$ for exactly one value of v, call this value v_i. Our LongPath solution will be $P = (v_0, v_1, \ldots, v_k)$, which can be constructed from α in time $O(nk)$.

Since α satisfies Clauses 3 and 4, we have $v_0 = s$ and $v_k = t$. Since α satisfies all clauses of Type 5, P has no repeated vertices. Since α satisfies all clauses of Type 6, we have that $(v_i, v_{i+1}) \in E$ for $i = 0, \ldots, k$. Thus, P is a path of length k from s to t with no repeated vertices, as desired. □

While the proofs of the two claims are very similar, it is important to note that they are proving different directions. In particular, Claim 2.1 would still hold even if we accidentally forgot to include some of the clause types in constructing φ; it is only by proving Claim 2.3 that we are sure that we haven't missed anything essential for ensuring that we get a solution to LongPath.