CS1200: Intro. to Algorithms and their Limitations Prof. Salil Vadhan

Sender–Receiver Exercise 5: Reading for Senders

Harvard SEAS - Fall 2025 *2025-10-21*

1 Long Path $\leq$ SAT

Section 2 is also in the reading for receivers. Your goal will be to communicate the *proof* of Theorem 2.1 (i.e. the content of Section 3) to the receivers.

2 The Result

In Section 17.3 of the textbook, we saw how the (seemingly hard) problem of Graph 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 allows all the effort put into SAT Solvers to solve many large instances of Graph Coloring in practice.

In this exercise, you'll see a similar reduction for the *Longest Path* problem. Recall that a *path* is a walk with no repeated vertices.

Input: A digraph $G = (V, E)$ and two vertices $s, t \in V$

Output: A *longest path* from s to t in G, if one exists

Computational Problem Longest Path

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, if one exists

Computational Problem Long Path

Since a path has no repeated vertices, it suffices to consider $k \leq n$. If we have an efficient algorithm for Long Path, then we can solve Longest Path by trying $k = n, n-1, \ldots, 0$ until we succeed in finding a path. The $k = n$ case is essentially the same as the Hamiltonian Path problem,[1] which is a special case of the notorious Travelling Salesperson Problem (TSP): In 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 (if $(u, v) \in E$) or a very large travel time (if $(u, v) \notin E$). In such a case, the only way to visit all cities in travel time at most $n - 1$ is via a Hamiltonian path.[2]

[1] In the Hamiltonian Path problem, we don't specify the start and end vertex; any path of length n suffices. But the two problems can be efficiently reduced to each other (exercise).

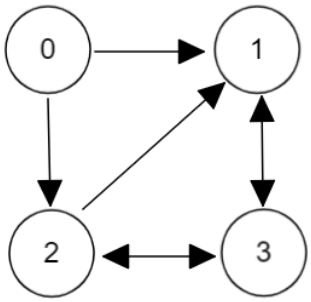
[2] Often in TSP, it is also required that the salesperson return to their starting city s. If we add edges of travel time 1 from all cities to s, then we see that Hamiltonian Path is also a special case of this variant of TSP.

The reduction from LONG PATH to SAT is given as follows.

Theorem 2.1. LONG PATH *on a digraph with n vertices, m edges, and a path length k reduces to* SAT *in time $O(n^2k)$.*

3 The Proof

Let $G = (V, E)$, s, t, k be our instance of LONG PATH. We will use the following graph G with $s = 0$, $t = 1$, and $k = 3$ as a running example:



The proof will be divided into two steps: constructing a SAT instance and then using the SAT instance in the reduction algorithm.

3.1 Constructing the SAT instance

The construction of the SAT instance φ from the LONG PATH instance will proceed by introducing a set of boolean 'indicator' variables (similar in spirit to what we did in Theorem **??**). We will have $(k + 1) \cdot n$ variables named $x_{i,v}$ for each $i \in [k + 1]$ and vertex $v \in V$. We'll seek to have $x_{i,v} = 1$ mean that the i^{th} vertex in the path is v.

For example, the path $0 \to 2 \to 3 \to 1$ in the example graph would be represented as the following assignment:

$$\begin{bmatrix} x_{0,0} & x_{0,1} & x_{0,2} & x_{0,3} \\ x_{1,0} & x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,0} & x_{2,1} & x_{2,2} & x_{2,3} \\ x_{3,0} & x_{3,1} & x_{3,2} & x_{3,3} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

To implement our goal to have $x_{i,v} = 1$ mean that the i^{th} vertex in the path is v, we need to enforce some constraints among the variables. For example, the path should start at s and end at t. Thinking about these constraints is a crucial step in the reduction and the clauses help to enforce these constraints. The output of the reduction, φ, is the AND of all the clauses introduced below, each of which enforces a different kind of constraint.

1. **At least one vertex is assigned to each step of the path:** This constraint is enforced by the clause $\left(\bigvee_{v \in V} x_{i,v}\right)$, for each $i = 0, \ldots, k$. In our example, this would yield the following clauses:

$$(x_{0,0} \vee x_{0,1} \vee x_{0,2} \vee x_{0,3}),$$
$$(x_{1,0} \vee x_{1,1} \vee x_{1,2} \vee x_{1,3}),$$
$$(x_{2,0} \vee x_{2,1} \vee x_{2,2} \vee x_{2,3}),$$
$$(x_{3,0} \vee x_{3,1} \vee x_{3,2} \vee x_{3,3})$$

2. **Two distinct vertices are not assigned to the same step of the path:** For each pair $v, w \in V$ such that $v \neq w$ and $i \in [k+1]$, we introduce the clause $(\neg x_{i,v} \vee \neg x_{i,w})$. In our example, this would yield the following clauses:

$$(\neg x_{0,0} \vee \neg x_{0,1}), (\neg x_{0,0} \vee \neg x_{0,2}), (\neg x_{0,0} \vee \neg x_{0,3}), (\neg x_{0,1} \vee \neg x_{0,2}), (\neg x_{0,1} \vee \neg x_{0,3}), (\neg x_{0,2} \vee \neg x_{0,3}),$$
$$(\neg x_{1,0} \vee \neg x_{1,1}), (\neg x_{1,0} \vee \neg x_{1,2}), (\neg x_{1,0} \vee \neg x_{1,3}), (\neg x_{1,1} \vee \neg x_{1,2}), (\neg x_{1,1} \vee \neg x_{1,3}), (\neg x_{1,2} \vee \neg x_{1,3}),$$
$$(\neg x_{2,0} \vee \neg x_{2,1}), (\neg x_{2,0} \vee \neg x_{2,2}), (\neg x_{2,0} \vee \neg x_{2,3}), (\neg x_{2,1} \vee \neg x_{2,2}), (\neg x_{2,1} \vee \neg x_{2,3}), (\neg x_{2,2} \vee \neg x_{2,3}),$$
$$(\neg x_{3,0} \vee \neg x_{3,1}), (\neg x_{3,0} \vee \neg x_{3,2}), (\neg x_{3,0} \vee \neg x_{3,3}), (\neg x_{3,1} \vee \neg x_{3,2}), (\neg x_{3,1} \vee \neg x_{3,3}), (\neg x_{3,2} \vee \neg x_{3,3})$$

3. **The path starts with s:** This is enforced by the simple clause $(x_{0,s})$ In our example, this would yield the clause $(x_{0,0})$.

4. **The path ends with t:** This is enforced by the clause $(x_{k,t})$. In our example, this would yield the clause $(x_{3,1})$.

5. **The same vertex is not assigned to two different steps in the path:** We enforce this by introducing this clause for each $v \in V$ and $i \neq j \in [k+1]$: $(\neg x_{i,v} \vee \neg x_{j,v})$.

   In our example, the clauses would look as the follows. For vertex 0, we would have the following clauses:

   $$(\neg x_{0,0} \vee \neg x_{1,0}), (\neg x_{0,0} \vee \neg x_{2,0}), (\neg x_{0,0} \vee \neg x_{3,0}), (\neg x_{1,0} \vee \neg x_{2,0}), (\neg x_{1,0} \vee \neg x_{3,0}), (\neg x_{2,0} \vee \neg x_{3,0})$$

   We can do the same for vertices 1,2, and 3:

   $$(\neg x_{0,1} \vee \neg x_{1,1}), (\neg x_{0,1} \vee \neg x_{2,1}), (\neg x_{0,1} \vee \neg x_{3,1}), (\neg x_{1,1} \vee \neg x_{2,1}), (\neg x_{1,1} \vee \neg x_{3,1}), (\neg x_{2,1} \vee \neg x_{3,1}),$$
   $$(\neg x_{0,2} \vee \neg x_{1,2}), (\neg x_{0,2} \vee \neg x_{2,2}), (\neg x_{0,2} \vee \neg x_{3,2}), (\neg x_{1,2} \vee \neg x_{2,2}), (\neg x_{1,2} \vee \neg x_{3,2}), (\neg x_{2,2} \vee \neg x_{3,2}),$$
   $$(\neg x_{0,3} \vee \neg x_{1,3}), (\neg x_{0,3} \vee \neg x_{2,3}), (\neg x_{0,3} \vee \neg x_{3,3}), (\neg x_{1,3} \vee \neg x_{2,3}), (\neg x_{1,3} \vee \neg x_{3,3}), (\neg x_{2,3} \vee \neg x_{3,3}),$$

6. **For any two adjacent steps in the path, the vertices should form an edge in the graph:** We enforce this using the following clauses. For each step $i \in [k]$, and any pair of vertices $(u, v) \in (V \times V) \setminus E$ (that is, (u, v) *do not* form an edge): $(\neg x_{i,u} \vee \neg x_{i+1,v})$. Thus, we enforce the constraint that consecutive vertices are edges by excluding non-edges.

   In our example, this would yield the following clauses. For the non-edges leaving vertex 0:

$$(\neg x_{0,0} \vee \neg x_{1,3}),$$
$$(\neg x_{1,0} \vee \neg x_{2,3}),$$
$$(\neg x_{2,0} \vee \neg x_{3,3})$$

   Looking at the non-edges of vertex 1:

$$(\neg x_{0,1} \vee \neg x_{1,0}), (\neg x_{0,1} \vee \neg x_{1,2}),$$
$$(\neg x_{1,1} \vee \neg x_{2,0}), (\neg x_{1,1} \vee \neg x_{2,2}),$$
$$(\neg x_{2,1} \vee \neg x_{3,0}), (\neg x_{2,1} \vee \neg x_{3,2})$$

   Looking at the non-edges leaving vertex 2:

$$(\neg x_{0,2} \vee \neg x_{1,0}),$$
$$(\neg x_{1,2} \vee \neg x_{2,0}),$$
$$(\neg x_{2,2} \vee \neg x_{3,0})$$

   Looking at the non-edges leaving vertex 3:

$$(\neg x_{0,3} \vee \neg x_{1,0}),$$
$$(\neg x_{1,3} \vee \neg x_{2,0}),$$
$$(\neg x_{2,3} \vee \neg x_{3,0})$$

Number of clauses in φ: Overall, 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 $k \leq n$.)

3.2 Reduction algorithm

The reduction from Long Path to SAT is as follows.

`LongPathViaSAT`(G, s, t, k):
Input : A digraph $G = (V, E)$, vertices $s, t \in V$, a path-length k
Output : A *longest path* from s to t in G with no repeated vertices, or $\perp$ if no path from s to t exists

0 Construct φ from G, s, t, k as described in Section 3.1;
1 Call the SAT Oracle on φ, obtaining α, which is either a satisfying assignment to φ or $\perp$ if φ is unsatisfiable;
2 **if** $\alpha = \perp$ **then return** $\perp$;
3 **else**
4 | Convert α to a Long Path solution P via Claim 3.2;
5 | **return** P;

Algorithm 3.2: Long Path by reduction to SAT

In Line 1, we call a SAT oracle on φ to find out whether φ is satisfiable. The following claim shows that (crucially) the satisfiability of φ is directly related to the existence of a path of length k from s to t in G.

Claim 3.1. *If there a* Long Path *solution on instance* (G, s, t, k)*, then* φ *is satisfiable.*

Further, if φ is satisfiable, we also obtain a satisfying assignment α. The following Claim, used in Line 6 of the reduction algorithm, shows how to efficiently construct a path of length k from s to t in G from α.

Claim 3.2. *We can transform any satisfying assignment* α *to* φ *into a solution to* Long Path *on instance* (G, s, t, k) *in time* $O(nk)$*.*

Before proving the claims, let's see why Theorem 2.1 follows. First, the reduction algorithm indeed runs in time $O(n^2k)$. The construction of φ in Line 0, which has $O(n^2k)$ clauses, takes time $O(n^2k)$ using the adjacency list representation of G. The oracle call in Line 1 is—by definition—1 step. By Claim 3.2, Line 4 takes time $O(nk)$. To prove correctness, Claim 3.1 implies that we only return $\perp$ when there is no solution to the Long Path instance, and Claim 3.2 implies that whenever we return a path P, it is a valid solution to the Long Path instance.

3.3 Proofs of the claims

Proof of Claim 3.1. Suppose there is a Long Path solution on instance (G, s, t, k). That is, there is a path P of length k, starting at s and ending at t. 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 clauses of Type 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 $i = 0, \ldots, k$). □

Proof of Claim 3.2. 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 Long Path 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 all $i \in \{0, \ldots, k-1\}$. Those are all the conditions for P to be a path of length k from s to t, so it is, 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 3.1 would still hold even if we accidentally forgot to include some of the clause types in constructing φ; it is only by proving Claim 3.2 that we are sure that we haven't missed anything essential for ensuring that we get a solution to Long Path.