# CSCC63 Assignment 3

## Polytime Reductions, Self-Reductions, and $\mathcal{PS}$

## Due 11:59pm, August 7

**Warning:** *For this assignment you may work either alone or in pairs. Your electronic submission of a PDF to Crowdmark affirms that this assignment is your own work and that of your partner, and no one else's, and is also in accordance with the University of Toronto Code of Behaviour on Academic Matters, the Code of Student Conduct, and the guidelines for avoiding plagiarism in CSCC63. Note that using Google or any other online resource is considered plagiarism.*

1. (10 marks) Consider the language MIN-BLOCKS defined as:

   **Instance:** A matrix $M$ and an integer $k$.

   **Question:** We're allowed to permute the rows and columns of $M$ as we like, and we want to do so in a way that minimizes the number of single-number rectangular "blocks" needed to describe the matrix. E.g., is we were to write $(a, b, c, d; 137)$, where $1 \leqslant a \leqslant c \leqslant m$ and $1 \leqslant b \leqslant d \leqslant n$ , then we would know that every entry $M_{i,j}$ for $a \leqslant i \leqslant c$ and $b \leqslant j \leqslant d$ would have the number 137.

   Our question is, can we permute the rows and columns of $M$ in such a way that we need no more than $k$ blocks to store it?

   As an example, if $M$ is

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

   then we can permute the second and third row and the first and second columns to get

   $$\begin{bmatrix} 0 & 1 & 1 \\ 2 & 1 & 1 \\ 2 & 2 & 2 \end{bmatrix}.$$

   We need four blocks to describe the matri we get from this permutation:

   $$\begin{bmatrix} \mathbf{0} & 1 & 1 \\ 2 & 1 & 1 \\ 2 & 2 & 2 \end{bmatrix},$$

   $$\begin{bmatrix} 0 & \mathbf{1} & \mathbf{1} \\ 2 & \mathbf{1} & \mathbf{1} \\ 2 & 2 & 2 \end{bmatrix},$$

   $$\begin{bmatrix} 0 & 1 & 1 \\ 2 & 1 & 1 \\ \mathbf{2} & \mathbf{2} & \mathbf{2} \end{bmatrix},$$

   and

   $$\begin{bmatrix} 0 & 1 & 1 \\ \mathbf{2} & 1 & 1 \\ \mathbf{2} & 2 & 2 \end{bmatrix}.$$

   Notice that we allow the third and fourth blocks to overlap, since they store the same number.

   You will also find that we can't use fewer than four blocks to store this matrix, regardless of the permutation. So if I give you $M$ and $k$ for $k \geqslant 4$, it will be *yes*-instance. If I give you a $k < 4$ it will be a *no*-instance.

You can assume this language is in $\mathcal{NP}$, but you have to show that it is $\mathcal{NP}$-complete (so you have to show that it's $\mathcal{NP}$-hard).

With this in mind, give a reduction that shows MIN-BLOCKS is $\mathcal{NP}$-hard. Reduce from some language in the Polytime_Reduction_Examples_1 or the Polytime_Reduction_Examples_2 handout.

**Solution:**

We reduce from UHAM-PATH:

Let $G$ be an undirected graph with $n$ vertices and $m$ edges, and let $s$ and $t$ be two vertices in $G$.

Let $G'$ be an undirected graph with two new vertices added: $s'$ and $t'$. We will add one edge from $s'$ to $s$ and one edge from $t$ to $t'$. Let $n' = n + 2$ and $m' = m + 2$, and let $M'$ be the $n' \times m'$ incidence matrix of $G'$.

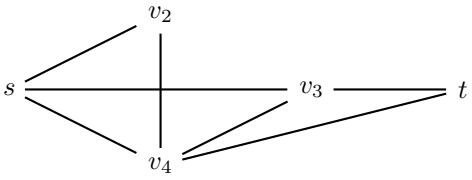
*Note that this means that the row $i$ of $M'$ corresponds to the vertex $v_i$ in $G'$, and the column $j$ in $M'$ corresponds to the edge $e_j$ in $G'$. The entry $M'_{ij}$ is 1 if $v_i$ is an endpoint of $e_j$, and 0 otherwise. Note that each column contains exactly 2 "1"s.*

We will modify $M'$ to form a new $n' \times m'$ matrix $M''$ such that:

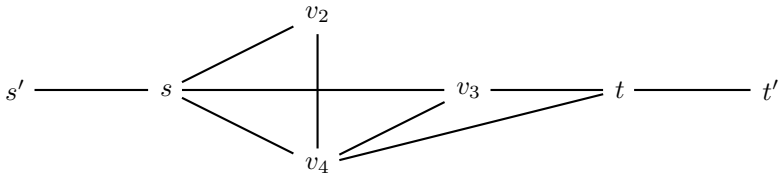
$$M''_{ij} = \begin{cases} j, & M'_{ij} = 1, \\ n'm'j + i, & M'_{ij} = 0. \end{cases}$$

*We'll count the indices here from 1, not 0. If we started at 0 we'd add a 1 to the $i$ and $j$.*

As an example, suppose we were given the following graph $G$:




We'd build from this $G$ the following $G'$:




The incidence matrix of $G'$ would be

$$M' = \left[\begin{array}{ccc:ccc:ccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right]$$

We'd use this $M'$ to build the following matrix $M''$:

$$M'' = \left[\begin{array}{ccc:ccc:ccc} 1 & 127 & 190 & 253 & 316 & 379 & 442 & 605 & 668 \\ 1 & 2 & 3 & 4 & 317 & 380 & 443 & 606 & 669 \\ 66 & 2 & 192 & 255 & 5 & 381 & 444 & 607 & 670 \\ \hdashline 67 & 130 & 3 & 256 & 319 & 6 & 7 & 608 & 670 \\ 68 & 131 & 194 & 4 & 5 & 6 & 446 & 8 & 672 \\ 69 & 132 & 195 & 258 & 321 & 384 & 7 & 8 & 9 \\ \hdashline 70 & 133 & 196 & 259 & 322 & 385 & 448 & 611 & 9 \end{array}\right]$$

We note that $0 < i, j \leqslant m'n'$ for all $i$ and $j$, and so:

- If $j \neq j'$, then $M''_{ij} \neq M''_{i'j'}$ for any $i$, $i'$.
- If $M'_{ij} = 1$, then $M''_{ij} = M''_{i'j'}$ iff $j = j'$ and $M''_{i',j'} = 1$.
- If $M'_{ij} = 0$ and $i' \neq i$, then $M''_{ij} \neq M''_{i'j}$.

In particular, the blocks that we get for $M''$ after any permutation of rows and columns will all be single-column – no matter how we permute the columns, we will never have a block that spans more than one column. We can therefore focus on permutations of only the rows of $M''$.

In addition, we see that once we permute the rows of $M''$, each column contains either $n$ or $n-1$ blocks: if $e_j = \{v_i v_{i'}\}$ then column $j$ of $M''$ contains $n-1$ blocks iff $i$ and $i'$ are adjacent in our row ordering. Otherwise column $j$ contains $n$ blocks. As an example, we can permute the fourth and fifth rows of the above $M''$ as follows:

$$M'' = \left[\begin{array}{ccc:ccc:ccc} \mathbf{1} & 127 & 190 & 253 & 316 & 379 & 442 & 605 & 668 \\ \mathbf{1} & \mathbf{2} & 3 & 4 & 317 & 380 & 443 & 606 & 669 \\ 66 & \mathbf{2} & 192 & 255 & \mathbf{5} & 381 & 444 & 607 & 670 \\ \hdashline 68 & 131 & 194 & 4 & \mathbf{5} & \mathbf{6} & 446 & 8 & 672 \\ 67 & 130 & 3 & 256 & 319 & \mathbf{6} & \mathbf{7} & 608 & 670 \\ 69 & 132 & 195 & 258 & 321 & 384 & \mathbf{7} & 8 & \mathbf{9} \\ \hdashline 70 & 133 & 196 & 259 & 322 & 385 & 448 & 611 & \mathbf{9} \end{array}\right]$$

We've highlighted the only numbers in this permutation that do *not* have to be placed in a $1 \times 1$ block. Note how each double block represents an edge between adjacent vertices.

Recall that each row corresponds to a vertex, and so any permutation of rows corresponds to a permutation of vertices. The order of the above permutation is $s', s, v_2, v_4, v_3, t, t'$. The highlighted blocks tell us that there are edges beteween the adjacent nodes in this list, and so this order represents a path from $s'$ to $t'$. In fact, the restriction of this list to $G$ gives us a Hamiltonian path from $s$ to $t$. We'll formalize this in our iff proof.

> **Note:** *We observe that no block in $M''$ can span more than one column, regardless of how we permute the rows and edges. In addition, every column requires either $n'$ or $N'-1$ blocks: if we look at a column $e_j = \{v_i, v_{i'}\}$, then we can see that column $j$ of $M''$ can be covered by $n'-1$ blocks iff the rows $i$ and $i'$ are permuted be adjacent. Otherwise the column will require $n'$ blocks.*

Once we've constructed $M''$, we return $\langle M'', k' = n'm' - n' + 1\rangle$.

**Proof that $\langle G, s, t\rangle \in$ UHAM-PATH iff $\langle M'', k'\rangle \in$ MIN-BLOCKS:**

> Suppose that $\langle G, s, t\rangle \in$ UHAM-PATH.
> Then $G$ has a Hamiltonian path $P = [s = v_{i_1}, v_{i_2}, \ldots, t = v_{i_n}]$.
> Then $G'$ has a Hamiltonian path $P' = [s', s, v_{i_2}, \ldots, t, t']$.
> Suppose we permute the rows of $M''$ to match this Hamiltonian path ($s'$ on the top row, $s$ on the second, $v_{i_2}$ on the third, and so on), and consider the columns corresponding to the edges $e_j$ in $P'$.
> As we have noted, these columns can be covered by $n'-1$ blocks, since we've permuted the rows of vertices adjacent in $P'$ to be adjacent in $M''$.

The other columns of $M''$ will need $n'$ blocks.
So in total we'll need $(n'-1) \times (n'-1)$ blocks to cover the columns representing edges in $P'$, and $(m'-n'+1) \times n'$ locks to cover the remaining columns.
When we expand this number, we find that the total number of blocks we need is

$$([n']^2 - 2n' + 1) + (n'm' - [n']^2 + n') = n'm' - n' + 1 = k'.$$

So $M''$ can be covered by $k'$ blocks.
$\Rightarrow \langle M'', k' \rangle \in$ MIN-BLOCKS.
Suppose that $\langle M'', k' \rangle \in$ MIN-BLOCKS.
Then we can permute the rows and columns of $M''$ so that ir can be covered by at most $k'$ blocks.
Now, we have observed that no block covering $M''$ can span more than one column, regardless of permutation. So we can focus on the permutations of the rows of $M''$.
Furthermore, we know that every column of $Mpprm$ must be covered by either $n'$ or $n'-1$ blocks.
This means that there must be at least $n-1$ columns of $M''$ that are permuted so that they require $n'-1$ blocks: otherwise the number of blocks we would need would be at least

$$\begin{aligned}(n'-2) \times (n'-1) + (m'-n'+2) \times n' &= ([n']^2 - 3n' + 2) + (n'm' - [n']^2 + 2n') \\ &= n'm' - n' + 2 \\ &> n'm' - n' + 1 \\ &= k'.\end{aligned}$$

We observe that each of these $n'-1$ columns must be unique, since they each represent an edge in $G'$, and since $G'$ cannot have repeated edges.
Now, the permutation that we have for the rows of $M''$ induces an ordering $v_{i'_1}, v_{i'_2}, \ldots, v_{i'_{n'}}$ of the vertices in $G'$, since the rows of $M''$ represent the vertices of $G'$.
As we have noted, the columns of $M''$ covered by $n'-1$ blocks must correspond to a pair of rows in which $v_{i'_\ell}$ and $v_{i'_{\ell+1}}$ are connected by an edge.
Moreover, each of these edges must be unique, and there must be $n'-1$ of them. So $v_{i'_\ell}$ and $v_{i'_{\ell+1}}$ are connected by an edge for all $1 \leqslant \ell < n'$.
But then, $P'' = [v_{i'_1}, v_{i'_2}, \ldots, v_{i'_{n'}}]$ must be a Hamiltonian path in $G'$.
Since $s'$ and $t'$ have only one neighbour each in $G'$, $P''$ must begin and end in $s'$ and $t'$.
But this means that the remainder of $P''$ is a simple path $P = [s = v_{i'_2}, v_{i'_3}, \ldots, t = v_{i'_{n'-1}}]$ in $G$.
We can see that $P$ is a simple path of length $n'-2 = n$, and so it must be a Hamiltonian path in $G$ from $s$ to $t$.
$\Rightarrow \langle G, s, t \rangle \in$ UHAM-PATH.

Finally, we see that $M''$ is a matrix of size $\mathcal{O}(mn)$, and that each entry has a size of $\mathcal{O}(mn)$. So each entry needs $\mathcal{O}(\log(mn))$ to write, and the total size of $M''$ is $\mathcal{O}(mn \log(mn))$. Each entry can be calculated in polytime, so the entire matrix can be found in polytime. Similarly, we can calculate $k' = n'm' - n' + 1$ in a polynomial amount of time.

So MIN-BLOCKS is $\mathcal{NP}$-complete, as required.

■

2. (10 marks) Consider the language CLOSE-SOLUTION-3SAT defined as:

**Instance:** A 3CNF $\phi$ and a satisfying truth assignment $\tau$ for $\phi$.

**Question:** Does $\phi$ have a second satisfying truth assignment $\tau' \neq \tau$ such that $\tau$ and $\tau'$ agree on at least 7/8 of their variable assignments?

You can assume this language is in $\mathcal{NP}$, but you have to show that it is $\mathcal{NP}$-complete (so you have to show that it's $\mathcal{NP}$-hard). With this in mind, give a proof that CLOSE-SOLUTION-3SAT is $\mathcal{NP}$-hard.

**Solution:**

Let $L$ be any language in $\mathcal{NP}$. Then $L$ has a polytime verifier $V$ such that

$$L = \{\langle x \rangle \mid \exists c, V \text{ accepts } \langle x, c \rangle\}.$$

Let $N_0$ be the NTM

Let $N_0$ = "On input $x$:
1. Non-deterministically choose a bit $b \in \{0, 1\}$.
2. If $b == 0$, *accept*.
3. Otherwise, non-deterministically choose a certificate $c$.
4. Run $V$ on $\langle x, c \rangle$.
5. If it accepts, *accept*. Otherwise, *reject*."

Now, let $\langle x \rangle$ be an arbitrary instance of $L$.

Let $\phi_0$ be the 3CNF that we get by running the parsimonious Cook-Levin reduction (with the 3CNF modification from Week 11) on the NTM $N_0$ and the input $x$, and let $\tau_0$ be the truth assignment induced by the computation history of $N_0$ in which $b$ is set to 0.

**Note:** On the top of page 14 and on the bottom of slide 30 for Week 10 we state that you can't use the Cook-Levin reduction for self-reduction questions. Its use was deliberately left available for other types of questions, and so you can use it here.

Since $N_0$ accepts if $b = 0$, we see that $\tau_0$ satisfies $\phi_0$.
Now, let $k$ be the number of variables in $\phi_0$. For $0 \leqslant i < 7k$ we'll introduce the variable $y_i$ to $\phi_0$, along with the clause $(y_i \vee y_i \vee y_i)$. Let $\phi$ be the resulting 3CNF, and let $\tau$ be the extension to $\tau_0$ in which every $y_i$ is set to *true*. Note that $\tau$ satisfies $\phi$, and that any $\tau' \neq \tau$ that also satisfies $\phi$ must also set every $y_i$ to *true* – that is, any satisfying $\tau'$ must agree with $\tau$ on at least 7/8 of its variables. We can see that $\langle \phi, \tau \rangle$ is a valid instance of CLOSE-SOLUTION-3SAT.

**Proof that $x \in L$ iff $\langle \phi, \tau \rangle \in$ CLOSE-SOLUTION-3SAT:**

Suppose that $x \in L$.
Then there is some certificate $c'$ such that $V$ accepts $\langle x, c' \rangle$.
This means that there is a accepting run of $N_0$ in which $b$ is set to 1 and $c'$ is the chosen certificate.
The computation history from this run will induce a truth assignment $\tau_0'$.
Note that $\tau_0'$ satisfies $\phi_0$, and that furthermore $\tau_0' \neq \tau_0$, since we're using the parsimonious Cook-Levin reduction.
We can extend $\tau_0'$ to a truth assignment for $\phi$ by setting $y_i$ to *true* for all $i$.
We note that $\tau'$ satsifies $\phi$, and that $\tau' \neq \tau$.
Finally, we also note that $\tau$ and $\tau'$ must agree on the $y_i$, which comprise 7/8 of the variables in $\phi$.
$\Rightarrow \langle \phi, \tau \rangle \in$ CLOSE-SOLUTION-3SAT.
Suppose that $\langle \phi, \tau \rangle \in$ CLOSE-SOLUTION-3SAT.
Then there is some second truth assignment $\tau'$ that also satisfies $\phi$, and which agrees with $\tau$ on at least 7/8 of its variables.
Since there is only one way of satisfying the $y_i$ variables, and since the rest of $\phi$ is the $\phi_0$ from the parsimonious Cook-Levin reduction, the restriction of $\tau'$ to $\phi_0$ gives us a truth assignment $\tau_0' \neq \tau_0$ that satifies $\phi_0$.

Since we're using the parsimonious Cook-Levin reduction, there is one satsfying truth assignment for $\phi_0$ for every accepting computation history of $N_0$ on $x$. So $\tau_0$ and $\tau_0'$ represent different accepting computation histories of $N_0$ on $x$.
But there is only one computation history for $N_0$ on $x$ in which $b = 0$, and that computation history is the one inducing the truth assignment $\tau_0$. So $\tau_0'$ represents a computation history in which $b = 1$.
Now, the only way that $N_0$ can accept if $b = 1$ is if it chooses a certificate $c'$ such that $\langle x, c' \rangle$ is accepted by $V$.
So there is some certificate $c'$ proving that $x \in L$.
$\Rightarrow x \in L$.

Finally, we argue:

- Running the parsimonious Cook-Levin reduction is polytime, as we have discussed in class.
- $N_0$ is a polytime NTM, and so if we simulate it and force it to take the $b = 0$ choice the process will also take a polynomial amount of time (note that we're forcing a known accepting computation history, not searching for an accepting computation hitory).
- We can build the truth assignment $\tau_0$ by reading off the values of this computation history, and this process will also take polynomial amount of time.
- Since $\phi_0$ is polynomial in the size of $x$, and since we build $\phi$ from $\phi_0$ by adding a number of clauses that is linear in the size of $\phi_0$, the total size (and computation required for) $\phi$ is polynomial in the size of $x$.
- Similarly, the time required to build $\tau$ is polynomial in the size of $x$.

So the total time needed to build $\langle \phi, \tau \rangle$ is polynomial in $x$, and so we have a polytime reduction $L \leqslant_p$ CLOSE-SOLUTION-3SAT.

So CLOSE-SOLUTION-3SAT is $\mathcal{NP}$-complete, as required.

■

3. Consider the language PATH-CONSTRAINT defined as:

**Instance:** A directed graph $G = (V, E)$, two vertices $s$ and $t$, and two numbers $r$ and $k$.

**Question:** Can we find a size-$r$ subset $R$ of vertices in $V$ such that if $P_R$ is the longest simple path from $s$ to $t$ using only vertices in $R$, then the number of vertices in $P_R$ is somewhere between 2 and $k - 1$?

(a) (10 marks) Show that PATH-CONSTRAINT is $\mathcal{NP}$-hard (and therefore not likely to be in co-$\mathcal{NP}$).

**Solution:**

We reduce from INDEPEDENT-SET.
Suppose we have an independent set instance $\langle G, k \rangle$. We build our PATH-CONSTRAINT language as follows:

We set $G'$ to be a copy of $G$ in which every edge $\{u, v\}$ is replaced with two directed edges $(u, v)$ and $(v, u)$. We add two new vertices $s$ and $t$ to $G'$, and for every vertex $v \neq s$ in $G'$ we add the edge $(s, v)$. We also add the edge $(v, t)$ to $G'$ for every vertex in $G'$ other than $s$ and $t$.

We return the PATH-CONSTRAINT instance $\langle G', r = k + 2, k' = 4 \rangle$

**Proof that** $\langle G, k \rangle \in$ INDEPEDENT-SET iff $\langle G', k + 2, 4 \rangle \in$ PATH-CONSTRAINT**:**

$\Rightarrow$) Suppose that $\langle G, k \rangle \in$ INDEPEDENT-SET.
Then $G$ has a size-$k$ independent set $S \subseteq V$.
Now, let $S'$ be $\{s, t\} cup \{v \mid v \in S\}$. Then $S'$ is a set of $k + 2 = r$ vertices in $G'$.
Moreover, there is a length-2 path from $s$ to $t$ in $S'$, and for every $v \in S$ there is a length-three path $[s, v, t]$.

However, there cannot be a simple length-4 path from $s$ to $t$, in $S'$, since that woud require two vertices $u, v \in S$ to have an edge $\{u, v\}$ in $G$.
So the maximum length of any $s$ to $t$ in $S'$ is between 2 and $3 = 4 - 4 = k' - 1$.
$\Rightarrow \langle G', k+2, 4\rangle \in$ PATH-CONSTRAINT.

$\Leftarrow$) Suppose that $\langle G', k+2, 4\rangle \in$ PATH-CONSTRAINT.
Then there is some size-$r$ subset $S'$ of vertices in $G'$ such that the longest $s$-to-$t$ path in $S'$ has a length between 2 and $k' - 1 = 4 - 1 = 3$.
Since the longest $s$-to-$t$ path in $S'$ has length at least 2, we see that $s$ and $t$ must be in $S'$.
Let $S$ be the remainder of $S'$. We see that $S$ is a size $r - 2 = (k+2) - 2 = k$ vertices in $G$.
Now, if we take any two vertices $u, v \in S$, we see that $\{u, v\}$ is not an edge in $G$, since otherwise $[s, u, v, t]$ would be a length-4 path in $S'$.
So $S$ is a size-$k$ independent set in $G$.
$\Rightarrow \langle G, k\rangle \in$ INDEPEDENT-SET

Finally, we see that this reduction is polytime: it takes $\mathcal{O}(|V|^2)$ time to copy the graph $G$ and $s$ and $t$, and polynomial time to copy out $|V|$ twice. So the entire reduction is polytime.

So PATH-CONSTRAINT is $\mathcal{NP}$-hard, as required.

■

(b) (10 marks) Show that PATH-CONSTRAINT is co-$\mathcal{NP}$-hard (and therefore not likely to be in $\mathcal{NP}$). Reduce from some language in the Polytime_Reduction_Examples_1 handout.

**Solution:**

Note that $\overline{PATH - CONSTRAINT}$ is the set of tuples $\langle G, s, t, r, k\rangle$ where $G = (V, E)$ is a directed graph such that every size-$r$ suubset of vertices of $G$ induces a subgraph in which the longest path from $s$ to $t$ has a length either less than 2 or at least $k$.

We reduce from HAM-PATH.
Suppose we have a HAM-PATH instance $\langle G, s, t\rangle$. We build our PATH-CONSTRAINT language as follows:

We set $G'$ to be $G$, and we return the PATH-CONSTRAINT instance $\langle G', s, t, r = |V|, k' = |V|\rangle$.

**Proof that** $\langle G, s, t\rangle \in$ HAM-PATH iff $\langle G', s, t, r, k'\rangle \in$ PATH-CONSTRAINT**:**

$\Rightarrow$) Suppose that $\langle G, k\rangle \in$ HAM-PATH.
Then there is a Hamiltonian path $P = [s = v_{i_1}, v_{i_2}, \ldots, t = v_{i_{|V|}}]$ in $G$.
Now, there is only one set $R$ of vertices in $G'$ containing $r = |V|$ vertices, and that is the set $V$ of all vertices in $G'$.
In this set $R$, we see that $P$ is a simple $s$-to-$t$ path of length $|V| = k'$.
So the longest path in $R$ has length at least $k'$.
$\Rightarrow \langle G', s, t, |V|, |V|\rangle \in$ PATH-CONSTRAINT.

$\Leftarrow$) Suppose that $\langle G', s, t, |V|, |V|\rangle \in$ PATH-CONSTRAINT.
Then every size-$r$ subset $R$ of vertices in $G'$ contains either a simle path of length $k'$ or longer from $s$ to $t$, or a longest $s$-to-$t$ path of length at most 1.
But every $s$-to-$t$ path contains both $s$ and $t$, and so must have length at least 2. So the longest simple path from $s$ to $t$ has length at least $k' = |V|$.
Now, we can see that there's only one set $R$ of $r = |V|$, and that set contains every vertex in $G$. Let $P$ be the length-$k'$ simple path from $s$ to $t$ in $R$.
We can see that $P$ is a simple $s$-to-$t$ path in $G$ that must cover all of the $|V|$ vertices in $G$, and so it is a Hamiltonian path from $s$ to $t$ in $G$.
$\Rightarrow \langle G, s, t\rangle \in$ HAM-PATH

Finally, we see that this reduction is polytime: it takes $\mathcal{O}(n^2)$ time to modify the graph $G$ and add $s$ and $t$, and polynomial time to calculate $r$ and $k'$. So the entire reduction is polytime.

So PATH-CONSTRAINT is co-$\mathcal{NP}$-hard, as required.

■

(c) (10 marks) Show that PATH-CONSTRAINT $\in \mathcal{PS}$.

**Solution:**

We'll use the following program:

Let $M$ = "On input $\langle G = (V, E), s, t, r, k\rangle$:

1. For every subset $R$ of vertices in $V$:
2. If $|R| == r$:
3. $longest = 0$.
4. For every simple $s$-to-$t$ path $P$ using only vertices in $R$:
5. $longest = \max(longest, len(P))$.
6. If $2 \leqslant longest \leqslant k - 1$, *accept*.
7. *Reject*.

Now, we can see that lines 2, 3, 5, 6, and 7 can all be run in polynomial time, and so they take polynomial space as well. We have to consider the space required to loop over all of the subsets of $V$ and to loop over all of the simple $s$-to-$t$ path in $R$.

- To loop over the subsets of vertices in $V$, we simply use an $n$-bit register $C$ and use it to count from 0 to $2^n - 1$, where $n = |V|$. For any number $0 \leqslant i < 2^n$, we set $R = \{v_j \,|\, C[j] = 1\}$.

  We can see that the space needed to loop over these subsets is just the space we need for the counter, which will be $\mathcal{O}(n)$. So this can be done in polynomial space.

- To loop over the simple $s$-to-$t$ paths, we can simply loop over the length-$n$ strings from $\Sigma = \{1, 2, \ldots, n\}$ (or the strings of length at most $n$). Following the procedure from page 10 of week 11 in the lecture notes, we identify, e.g., $12524\ldots$ with the sequence $v_1, v_2, v_5$. Note that we stop reading once we see a repeated vertex.

  Once we have this sequence of vertices, we can check whether it represents an $s$-to-$t$ path $P$ in $G$ that uses vertices in $R$. If it is, we proceed to step 5, otherwise we move to the next string.

  We can see that we need only $\mathcal{O}(nlogn)$ space to loop over the length-$n$ strings in $\Sigma$, and that generating a vertex sequence and checking it using these strings takes polynomial time (and therefore polynomial space, as well). So the entire process takes polynomial space.

Once wqe put all of these steps together we see that $M$ requires a polynomial amount of space.

Finally, we see that if $\langle G, s, t, r, k\rangle \in$ PATH-CONSTRAINT, then there is some size-$R_0$ subset $R_0$ of its vertices for which the longest simple $s$-to-$t$ path $P_{R_0}$ has a length between 2 and $k - 1$. When run on $\langle G, s, t, r, k\rangle$ $M$ will either reach the outer loop iteration $R = R_0$, or it will accept before reaching this point. In this loop iteration the variable *longest* will be set to $len(P_{R_0})$ after the inner loop reaches the iteration $P = P_{R_0}$, and since this is the longest simple $s$-to-$t$ path in $R$, the value of *longest* when $M$ reaches line 6 will be $len(P_{R_0})$, at which point it will accept.

On the other hand, if $M$ accepts $\langle G, s, t, r, k\rangle$ it must do so in line 6. It can only reach line 6 after looping through every simple $s$-to-$t$ path in some size $r$ subset $R$ of vertices, at which point *longest* holds the length of the longest simnple $s$-to-$t$ path in $R$. Since $M$ accepts when *longest* is between 2 and $k - 1$, we know that $R$ is a size-$r$ subset of vertices in $G$ for which the longest $s$-to-$t$ path has a length between 2 and $k - 1$. So $\langle G, s, t, r, k\rangle \in$ PATH-CONSTRAINT.

So PATH-CONSTRAINT $\in \mathcal{PS}$, as required.

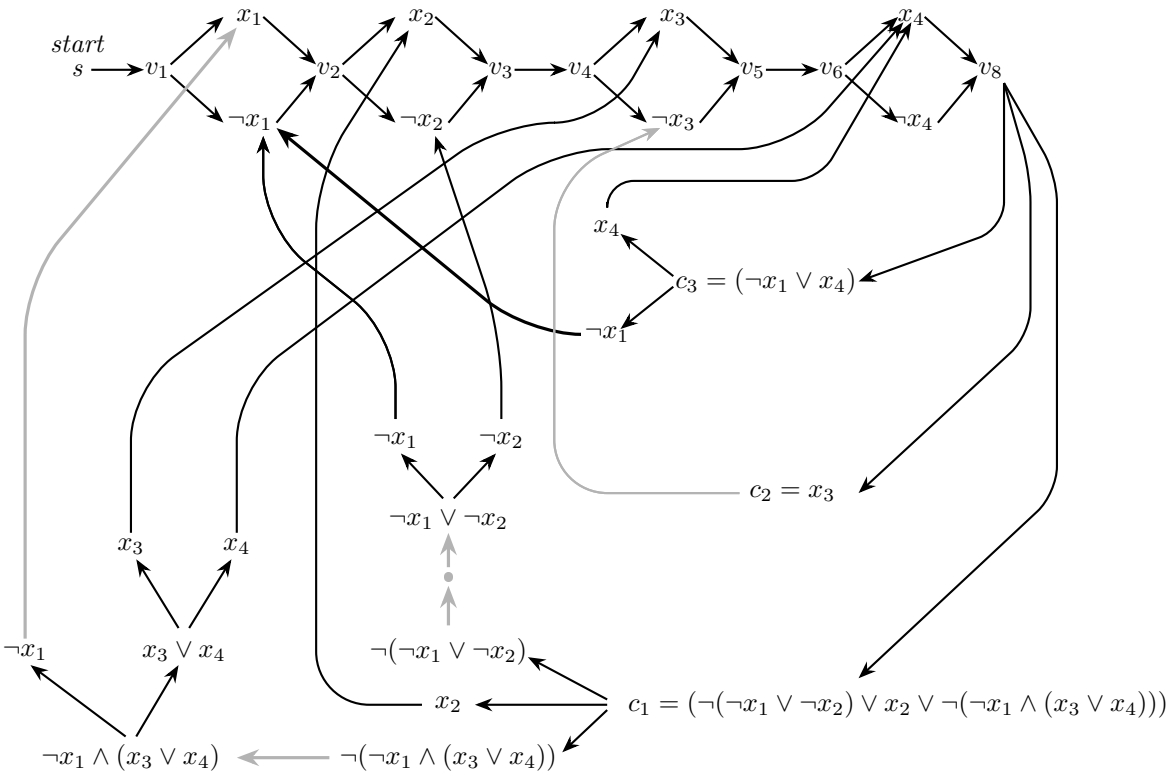
■

4. (10 marks) Let $f$ be the polytime reduction from TQBF to GG (generalized geography) as described in class.

   Let $\phi = \forall x_1, x_2 \exists x_3, \forall x_4 (\neg(\neg x_1 \vee \neg x_2) \vee x_2 \vee \neg(\neg x_1 \wedge (x_3 \vee x_4))) \wedge x_3 \wedge (\neg x_1 \vee x_4)$.

   Draw $f(\phi)$ (do not simplify the formula before applying $f$). Label your graph appropriately and indicate the starting node.

   **Solution:**




   **Note:** *The arrows and node in grey will likely be tricky for students to get, since those connections aren't obvious given the class example. We'll be lenient with the end connections (the ones at the literal nodes) if there are mistakes there, but you should get the internal grey arrows (the ones from the negated clauses). Make sure you understand why they're connected the way they are for the final.*

5. (10 marks) Recall the STORAGE-BOX language, as given in assignment 2.

   You've already given a proof that this language is $\mathcal{NP}$-complete, but we can also ask how, given an oracle for STORAGE-BOX, to find a solution for a particular STORAGE-BOX instance.

   Show how you can use an oracle for STORAGE-BOX to build a polytime oracle program **FIND-STORAGE-BOXES** such that for any instance of STORAGE-BOX:

   - If any solution exists for that instance, **FIND-STORAGE-BOXES** will return some such solution.
   - If no such solution exists, **FIND-STORAGE-BOXES** will return *null*.

   Justify why your program is correct and runs in polytime.

**Solution:**

Suppose that our STORAGE-BOX oracle is referred to as **SB**. Here is our algorithm:

**FIND-STORAGE-BOX** = "On input $\langle U, b, k\rangle$:

1. Query **SB** on $\langle U, b, k\rangle$.
   $\mathcal{O}(|\langle U, b, k\rangle|)$ *time.*
2. If it returns *false*, return *null*.
   $\mathcal{O}(1)$ *time.*
3. Else:
   $\mathcal{O}(1)$ *time.*
4. Let $U' = U$ and $S = []$.
   $\mathcal{O}(|\langle U, b, k\rangle|)$ *time.*
5. For $i = 1$ to $\min(k, |U|)$:
   $\mathcal{O}(|U|) = \mathcal{O}(|\langle U, b, k\rangle|)$ *iterations.*
6. If $|U'| > i$, set $S[i] = \{U[i]\}$.
   $\mathcal{O}(|\langle U, b, k\rangle|)$ *time per iteration.*
7. For $j = |U'|$ to $i + 1$:
   $\mathcal{O}(|U|)$ *iterations.*
8. Let $U'' = U' \setminus \{U'[j]\} \cup \{x_{ij}\}$, where $s(x_{ij}) = U'[i] + U'[j]$.
   $\mathcal{O}(|\langle U, b, k\rangle|)$ *time per iteration.*
9. Query **SB** on $\langle U'', b, k\rangle$.
   $\mathcal{O}(|\langle U, b, k\rangle|)$ *time per iteration.*
10. If it accepts:
    $\mathcal{O}(1)$ *time per iteration.*
11. $S[i] = S[i] \cup U'[j]$.
    $\mathcal{O}(|\langle U, b, k\rangle|)$ *time per iteration.*
12. $U' = U''$.
    $\mathcal{O}(|\langle U, b, k\rangle|)$ *time per iteration.*
13. Return $\langle S\rangle$.
    $\mathcal{O}(|\langle U, b, k\rangle|)$ *time.*

We can see that if $n = |\langle U, b, k\rangle|)$ time, then this whole program takes $\mathcal{O}(n^3)$ time. So this is a polytime program.

Moreover, we can see that our program returns *null* iff $\langle U, b, k\rangle \notin$ STORAGE-BOX, as required. Otherwise, suppose that we set the predicate $P(i)$ to be true iff, after iteration $i$ of the outer loop in **FIND-STORAGE-BOX**, we have that:

i) There is a valid box assignment for $\{U', b, k\}$ in which $U'[i']$ is assigned to box $i'$ for all $1 \leqslant i' \leqslant i$ and such that $U'[i'']$ is not assigned to box $i'$ for any $i'' > i$ and $i' \leqslant i$.

ii) $\sum_{u \in S[i']} s(u) = s\big(U'[i']\big)$ for all $i' \leqslant i$.

iii) $U = \Big( \bigcup_{i'=1}^{i} S[i'] \Big) \cup \{U'[i''] \mid i'' > i\}$.

Now, we can prove this statement using induction on $i$, and we can see that once the loop finishes there will be no element of $U$ that is not in some $S[i']$:

- If $i = k$ upon the completion of the loop, then we know from point i) of $P$ that there are no elements $i'' > i$ that are assigned to any boxes from 1 to $i = k$. But every element must be assigned to some box, and so there are no elements $i'' > i$ left to consider.
- If $i = |U|$ upon the completion of the loop, then there never were any $i'' > |U|$ to consider in the first place.

So by point iii) we know that every element of $U$ is in some $S[i']$. But we also know that the elements of each $S[i']$ can be assigned to box $i'$ for $1 \leqslant i' \leqslant i$, and so $S$ represents a box assignment for all of $U$. So we return a valid box assignment, as required.

□

6. (10 marks) Recall the FEEDBACK-ARC-SET language, as given in the Polytime_Reduction_Examples_1 on the course website.

   You've already been given a proof that this language is $\mathcal{NP}$-complete, but we can also ask how, given an oracle for FEEDBACK-ARC-SET, to find an optimal feedback arc set for a particular directed graph $G$.

   Show how you can use an oracle for FEEDBACK-ARC-SET to build a polytime oracle program **MINIMIZE-FEEDBACK-ARC** such that takes as input any directed graph $G$ and returns a smallest feedback arc set for $G$.

   Justify why your program is correct and runs in polytime.

   **Solution:**

   Suppose that our FEEDBACK-ARC-SET oracle is referred to as **FAS**. Here is our algorithm:

   **FIND-MIN-FAS** = "On input $\langle G = (V, E)\rangle$:

   1. $max = |V|$
      $\mathcal{O}(n)$ *time, where* $n = |V|$.
   2. For $i = |V|$ to 0:
      $\mathcal{O}(n)$ *iterations.*
   3. Query **FAS** on $\langle G, i\rangle$.
      $\mathcal{O}(|\langle G, n\rangle|)$ *time per iteration.*
   4. $S = []$.
      $\mathcal{O}(1)$ *time.*
   5. For each edge $e$ in $G$:
      $\mathcal{O}(m)$ *iterations, where* $m = |E|$.
   6. Let $G'$ be a copy of $G$ with $e$ removed.
      $\mathcal{O}(|G|) = \mathcal{O}(n^2)$ *time per iteration.*
   7. Run **FAS** on $\langle G', max - 1\rangle$.
      $\mathcal{O}(|\langle G, n\rangle|)$ *time per iteration.*
   8. If it accepts:
      $\mathcal{O}(1)$ *time per iteration.*
   9. $max -= 1$.
      *Amortizes to* $\mathcal{O}(1)$ *time per iteration.*
   10. $S$.append($e$).
       $\mathcal{O}(\log n)$ *time per iteration.*
       *(We'll accept* $\mathcal{O}(1)$ *time per iteration.)*
   11. $G = G'$.
       $\mathcal{O}(n^2)$ *time per iteration.*
   12. Return $S$.
       $\mathcal{O}(m)$ *time.*

   Now, if we make use of the fact that $m \leqslant \binom{n}{2} = \mathcal{O}(n^2)$, we find that this entire algorithm takes at most $\mathcal{O}(n^4)$ time. So this is a polytime algorithm.

   Moreover, if $\hat{k}$ is the size of the smallest feedback arc set for $G$, then we can prove by induction on the first loop that after every iteration, $\hat{k} \leqslant max$, with equality iff $i \leqslant \hat{k}$. Since this loop terminates when $i = 0$, we can see that $max = \hat{k}$ in line 4.

Furthermore, we can form a predicate $P$ that is true after every iteration of the loop from lines 5 to 11 iff:

i) $|S|+max= \hat{k}$.

ii) There is a $\hat{k}$-sized feedback arc set for $G$ that contains all of $S$ and exactly $max$ edges from $G'$.

iii) If $E'$ is the set of edges in $G'$, then $E = S \sqcup G'$, and no edge in $E'$ has ever been an element of $S$.

We can again prove this predicate using induction on the number of loop iterations, and we can see that once the loop terminates that $max= 0$. Otherwise, there would be some feedback arc set of size $\hat{k}$ containing all of $S$ and $max$ edges from $G'$, by point ii). If $max\neq 0$, thie feedback arc set must contain some edge $e \notin S$. But we see from point iii) that no element can ever be removed from $S$, and so in the loop iteration for the edge $e$ the oracle call to **FAS** shoulld have accepted. So we have a contradiction.

But this means that once the loop terminates $S$ must contain $\hat{k}$ elements by point i), and by point ii) we see that this must be a feedback arc set of $G$. So we return a minium-sized feedback arc set, as required.

□

7. (10 marks) Consider the 3DM language, as given in the Polytime_Reduction_Examples_1 handout on the course website.

   Modify the $\mathcal{NP}$-hardness proof for 3DM to make it a parsimonious reduction from #3SAT.

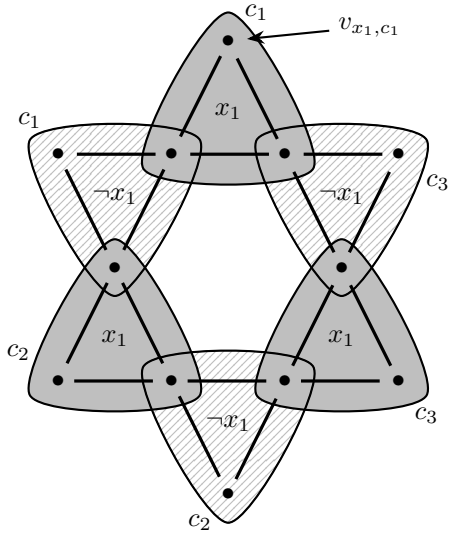
   **Solution:**

   Recall that the reduction for 3DM is a modification of the in-class reduction from 3SAT into PARTITION-INTO-TRIANGLES. So let's start by modifying that reduction.

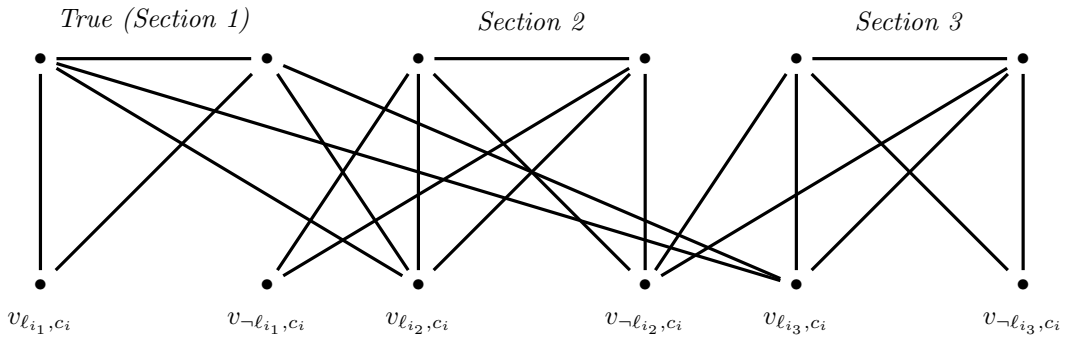
   Suppose we have an input 3CNF $\phi$. We'll assume using our helpful 3SAT lemma from the first Polytime Reductions handout that that this $\phi$ has three distinct variables per clause (i.e., no $(x \vee x \vee y)$ type clauses). Note that this lemma is parsimonious.

   Now, consider the PARTITION-INTO-TRIANGLES reduction from the polytime reductions chapter, our graph is made out of variable widgets, clause widgets, and garbage collection widgets. Our new graph $G'$ will be made of the same types of components.

   Firstly, the variable widgets will be built in exactly the same way. There will be one widget per variable, and each widget will be a star with $2n$ points, where $n$ is the number of clauses in $\phi$. So for a 3-clause $\phi$, the widget for $x_1$ would be of the form:



Recall that the variable $v_{x_1,c_1}$ is a variable that remains free if we choose $x_1 = T$, while the other variable is taken. The same applies for the other clauses and variables. So we now have to form the clause widgets. For the clause $c_i = (\ell_{i_1} \vee \ell_{i_2} \vee \ell_{i_3})$, where $\ell_{i_1}, \ell_{i_2}$, and $\ell_{i_3}$ are the literals in $c_i$, we produce the following widget:




*Remember that when we choose a variable assignment, half of the nodes in the bottom row are used up and so are unavailable for the clause widget!*

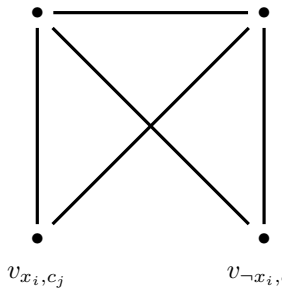
Note that the variable $v_{\ell_{i_1},c_i}$ connects to the appropriate outer node in the corresponding variable widget: so if $\ell_{i_1} = \neg x_2$, for example, the $v_{\ell_{i_1},c_i}$ above would actually be $v_{\neg x_2,c_i}$.

Now, We can see that there is only one way to partition this widget when a clause assignment is satisfying, and no way to partition it for a non-satsifying assignment:

- If $\ell_{i_1} = \ell_{i_2} = \ell_{i_3} = F$ (the clause is not satisfying), we'd have to connect $v_{\neg\ell_{i_1},c_i}$ to *Section 2*. But then, we'd have to connect $v_{\neg\ell_{i_2},c_i}$ to *Section 3*, and we'd have no way to connect $v_{\neg\ell_{i_1},c_i}$. So there would be no partition of the widget.
- If $\ell_{i_1} = \ell_{i_2} = F$ and $\ell_{i_3} = T$, then as before, we'd have to connect $v_{\neg\ell_{i_1},c_i}$ to *Section 2* and $v_{\neg\ell_{i_2},c_i}$ to *Section 3*. But this time we'd have to connect $v_{\ell_{i_3},c_i}$ to *Section 1 (True)*. So there would be exactly one partition of the widget.
- If $\ell_{i_1} = \ell_{i_3} = F$ and $\ell_{i_2} = T$, we'd have to connect $v_{\neg\ell_{i_1},c_i}$ to *Section 2*. But then, we'd have to connect $v_{\ell_{i_2},c_i}$ to *Section 1 (True)*, and then $v_{\neg\ell_{i_3},c_i}$ to *Section 3*.
- If $\ell_{i_1} = F$ and $\ell_{i_2} = \ell_{i_2} = T$, then we'd have to connect $v_{\neg\ell_{i_1},c_i}$ to *Section 2*. We'd then have to connect $v_{\ell_{i_2},c_i}$ to *Section 1 (True)*. Finally, we'd have to connect $v_{\ell_{i_3},c_i}$ to *Section 3*.
- If $\ell_{i_1} = T$ and $\ell_{i_2} = \ell_{i_2} = F$, then we'd have to connect $v_{\ell_{i_1},c_i}$ to *Section 1 (True)*. We'd then have to connect $v_{\neg\ell_{i_3},c_i}$ to *Section 3*. Finally, we'd have to connect $v_{\neg\ell_{i_2},c_i}$ to *Section 2*.
- If $\ell_{i_1} = \ell_{i_3} = T$ and $\ell_{i_2} = F$, then we'd have to connect $v_{\ell_{i_1},c_i}$ to *Section 1 (True)*. We'd then have to connect $v_{\ell_{i_3,c_i}}$ to *Section 3*. Finally, we'd have to connect $v_{\neg\ell_{i_2},c_i}$ to *Section 2*.
- $\ell_{i_1} = \ell_{i_2} = T$ and $\ell_{i_3} = F$, then we'd have to connect $v_{\ell_{i_1},c_i}$ to *Section 1 (True)*. We'd then have to connect $v_{\ell_{i_2,c_i}}$ to *Section 2*. Finally, we'd have to connect $v_{\neg\ell_{i_3},c_i}$ to *Section 3*.
- $\ell_{i_1} = \ell_{i_2} = \ell_{i_3} = T$, then we'd have to connect $v_{\ell_{i_1},c_i}$ to *Section 1 (True)*. We'd then have to connect $v_{\ell_{i_2,c_i}}$ to *Section 2*. Finally, we'd have to connect $v_{\ell_{i_3},c_i}$ to *Section 3*.

In all of these cases, we can see that a satisfying assignment to $c_i$ induces exactly one partition of the clause widget.

Finally, we see that if we have assigned the nodes from a satisfying truth assignment, the only remaining unpartitioned nodes are of the form $v_{x_i,c_j}$ or $v_{\neg x_i,c_j}$, where neither $x_i$ nor $\neg x_i$ occurs in clause $c_j$. These pairs will never be used by any clause widget, so we can add exactly one garbage collection widget of the form




to each of these pairs. We can see that for any truth assignment one of the bottom nodes will be taken, and so there will be exactly one way to partition the remaining three nodes.

So all told:

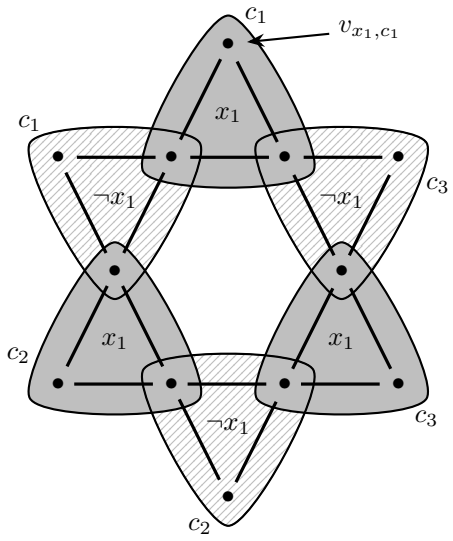
- If we are given any satisfying truth assignment there will be exactly one corresponding triangle partition, while for any non-satisfying assignment there will be none.
- If $\tau_1$ and $\tau_2$ are satisfying truth assignments for $\phi$ that differ on some variable $x_i$, then the triangle partitions they contribute differ on the clause widget for $x_i$. So they contribute different triangle partitions.
- Every partition $P$ must give us a satisfying truth assignment $\tau$ (because of the variable widgets). Since each such $\tau$ can contribute only one triangle partition for $G$, the partiton it contributes must be the $P$.

So there is a one-to-one relationship between the satisfying truth assignments for $\phi$ and the partitions of $G$.

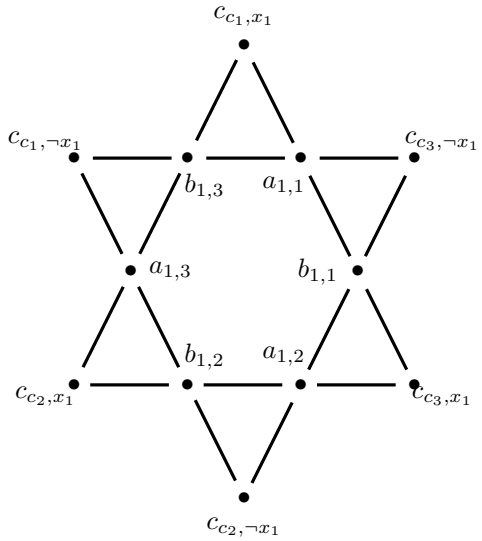
Finally, we can see that this reduction is polytime: if $\phi$ has $n$ clauses and $k$ variables, then $G$ has $4kn$ nodes for the variable widgets, $6n$ extra nodes for the clause widgets, and $2(k-3)n$ nodes for the garbage collectors. These nodes, and their edges, can be found in polynomial time, and so the whole construction is polytime, as required.

In our original 3DM reduction we gave a 3-colour of $G$ in which every colour had the same number of vertices. We used these colours to define the sets $A$, $B$, and $C$. Let's repeat that step here.

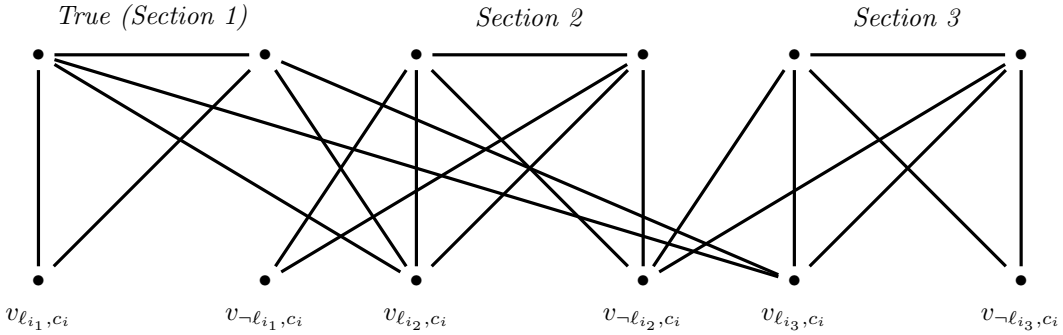
Recall that the variable widgets will be built with one widget per variable, and each widget will be a star with $2n$ points, where $n$ is the number of clauses in $\phi$. So for a 3-clause $\phi$, the widget for $x_1$ would be of the form:




What we want to do is label each of the vertices. So we'll give the vertices these labels:

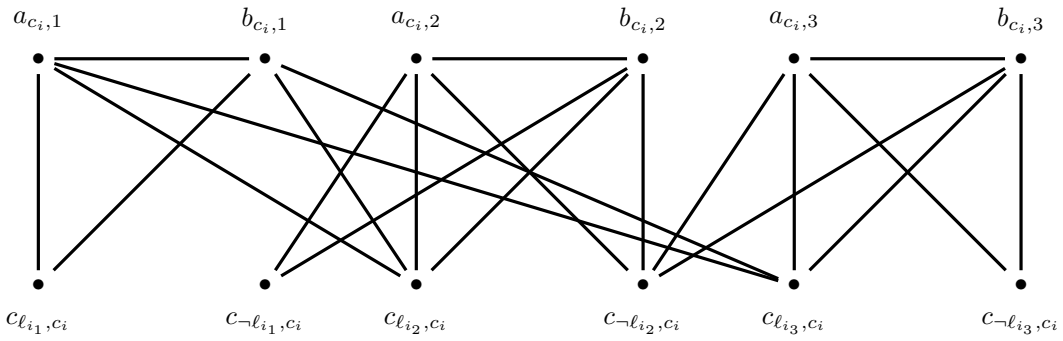



Recall that the variable $v_{x_1,c_1}$ is a variable that remains free if we choose $x_1 = T$, while the other variable is taken. The same applies for the other clauses and variables. With this in mind, recall that in the #PARTITION-INTO-TRIANGLES section of the reduction, for the clause $c_i = (\ell_{i_1} \vee \ell_{i_2} \vee \ell_{i_3})$, where $\ell_{i_1}, \ell_{i_2}$, and $\ell_{i_3}$ are the literals in $c_i$, we produced the following widget:

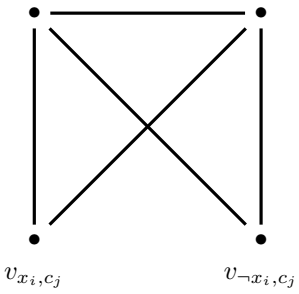


So now we want to label these nodes. The labels we give are as follows:

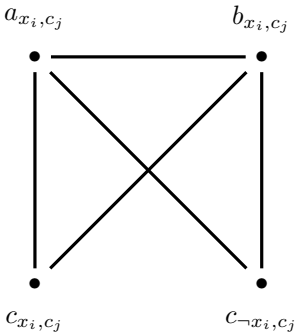
*Remember that the vertices on the bottom row are also part of the variable widgets, so they're already labelled!*




Finally, we have to label the nodes from the garbage collection widgets. Recall that these widgets connect to nodes of the form $v_{x_i,c_j}$ or $v_{\neg x_i,c_j}$, where neither $x_i$ nor $\neg x_i$ occurs in clause $c_j$. These pairs will never be used by any clause widget, and for each such pair we add exactly one garbage collection widget of the form




We label these vertices as follows:




If we assign all of the $a$ labels to the set $A$, the $b$ labels to the set $B$, and the $c$ labels to the set $C$, then we can see that $|A| = |B| = |C| = 2nk$, where $n$ is the number of clauses in $\phi$ and $k$ is the number of variables. Furthermore, every triangle in $G$ contains exactly one of each type of label (has a triple of labels in $A \times B \times C$). So we can set $T$ to be a list of all of the triangles in $G$.

*Note that we can find $T$ in polynomial time by looping over all triples of vertices!*

Clearly, any partitioning of $G$ into disjoint triangles will induce a subset $M \subseteq T$ that satisfies properties i) and ii) in the 3DM problem defintion, and vice-versa. So there is a one-to-one relation between the ways of partitioning the graph $G$ into triangles and the number of certificates $M \subseteq T$. Thus, there is a one-to-one relation between the certificates $M \subseteq T$ and the satisfying truth assignments for $\phi$.

So there is a one-to-one relationship between the satisfying truth assignments for $\phi$ and the partitions of $G$.

Finally, we can see that this reduction is polytime: the construction of $G$ is polytime in the size of $\phi$, as described in the #PARTITION-INTO-TRIANGLES section of the reduction. The label sets $A$, $B$, and $C$ are simply labels of the vertices of $G$, and can be efficiently computed. Finally, the set $T$ can be found in polytime by looping over the triples of vertices in $G$, and so can also be found in polytime. So the entire construction is polytime, as required.

■

8. (10 marks) Consider again the FEEDBACK-ARC-SET language, as given in the Polytime_Reduction_Examples_1 handout on the course website.

   Modify the $\mathcal{NP}$-hardness proof for FEEDBACK-ARC-SET to make it a parsimonious reduction.

   **Solution:**

   To start, let's look at the biggest problem that keeps the initial reduction from being parsimonious: when we look at the poible feedback arc sets of $G'$ in our iff proof, we note that we might have a feedback arc set $S$ containing an edge of the form $(v_1, u_0)$, where $\{v, u\}$ is an edge in $G$. For the purpoes of the regular $\mathcal{NP}$-completeness reduction we argue that any such set $S$ can be transformed into an equivalent feedback arc set $S'$ of $'$, where $|S'| \leqslant |S|$, and where the edge $(v_1, u_0)$ is replaced with the edge $(u_0, u_1)$. An investigation of this process shows that there's no eay one-to-one (or even a one-to-$m$ or $n$-to-$m$) relationship beteween these $S$ and $S'$. To get one-to-one relationship between our certificates we'll need to ensure that all feedback arc sets of the ie $k$ contain only edges of the form $(u_0, u_1)$: none of them contin any edge of the form $(v_1, u_0)$.

   The reason this is difficult is that we might not have a minimal feedback arc set (or verte cover). If a graph $G$ has a minimum sized feedback arc set of size $\hat{k}$, then our initial resuction will give us a graph $G'$ whose minimal feedback arc set $\hat{S}$ is also of size $\hat{k}$. If we run our reduction on a VERTEX-COVER instance $\langle G, k = \hat{k} + 1\rangle$, we could add any one edge to $\hat{S}$ to get a size $k$ feedback arc set $S$. So we need to enure that we only use VERTEX-COVER instances $\langle G, k\rangle$ where $k$ is the size of the samllest vertex cover in $G$. Finding this particular $k$ for a general graph $G$ is both $\mathcal{NP}$-hard and co-$\mathcal{NP}$-hard, though, and so we can't just try to find the minimum $k$. Nor can we directly limit ourselves to instances with a minimum $k$.

   Fortunately, there is a type of graph that will be useful to us where we know the minimum value of $k$: If we run the parsimonious 3SAT $\leqslant_p$ CLIQUE reduction on a $k'$-clause 3CNF $\phi$, we get a graph $G$ that has a $k'$-clique iff $\phi$ is satisfiable. Note that this particular $G$ never has a clique of size $k'+1$. Similarly, we note that if we run $G$ through the reduction CLIQUE $\leqslant_p$ INDEPENDENT-SET gives us a graph $G'$ with an independent set of size $k'$ iff $G$ has a clique of size $k'$. Note that this $G'$ never has an independent set of size $k' + 1$, and note furthermore that the reduction (from the Polytime Reductions handout) is parsimonious. Finally, suppose that we run $G'$ through the reduction INDEPENDENT-SET $\leqslant_p$ VERTEX-COVER (and so get the same graph $G'$). We note that $G'$ has a vertex cover of size $k = n - k'$ iff $G'$ has an independent set of size $k'$, and tht furthermore it never has a vertex cover of size $k - 1$. We can also see that this reduction is parsimonious.

   So, given a $k'$-clause 3CNF $\phi$, let $f(\phi) = \langle G, k\rangle$ be the instance of VERTEX-COVER that we obtain by running $\phi$ through the composition of the parsimonious reductions from 3SAT to CLIQUE to INDEPENDENT-SET to VERTEX-COVER. Note that this $f$ is polytime, and that $G$ has exactly one size-$k$ vertex cover for every one satisfying truth assignment to $\phi$.

   Once we've got the VERTEX-COVER instance $\langle G = (V, E), k\rangle$ we'll run it through a modification of the VERTEX-COVER $\leqslant_p$ FEEDBACK-ARC-SET reduction from the Polytime Reductions handout. As before, we'll build a $G'$ using an initial vertex set of $V_0 \cup V_1$, where $V_0$ and $V_1$ are copies of $V$. For every $v \in V$, we add the edge $(v_0, v_1)$ to $G'$.

Now, for every edge $e_j = \{u, v\} \in E$, we add $2(k+1)$ new vertices to $G'$: $v_{e_j,1}, v_{e_j,2}, \ldots, v_{e_j,k+1}$ and $u_{e_j,1}, u_{e_j,2}, \ldots, u_{e_j,k+1}$. Using these vertices we add the new edges $(u_1, u_{e_j,i})$, $(u_{e_j,i}, v_0)$, $(v_1, v_{e_j,i})$, and $(v_{e_j,i}, u_0)$ for all $1 \leqslant i \leqslant k+1$.

*This is effectively like adding $k+1$ copies of the edges $(u_1, v_0)$ and $(v_1, u_0)$ in the initial reduction. We're not allowed to use multigraphs, though, so we can't add any one edge more than once. So we split the repeated edges with a vertex in the middle to make it valid graph.*

Since we know that there is no $k-1$ vertex cover for $G$, we can make the following claim:

**Claim:** Every size-feedback arc set of $G'$ of size at most $k$ must have size exactly $k$, and must contain only edges of the form $(u_0, u_1)$.

**Proof:** Suppose that we have a set $S'$ of at most $k$ edges for $G'$, or which contains exactly $k$ edges, at least one of which is *not* of the form $(u_0, u_1)$. Let $S$ be the set of vertices in $G$ defined as $S = \{v \in V \mid (v_0, v_1) \in S'\}$. Note that $|S| < k$.
Since $|S| < k$, it cannot be a vertex cover for $G$. So there are at least two vertices $u, v \in V \setminus S$ such that there is an edge $e_{u,v} = \{u, v\} \in E$.
Since $u, v \in V \setminus S$, the edges $(u_0, u_1)$ and $(v_0, v_1)$ are not in $S'$.
Now, there are $k+1$ disjoint paths $u_1, u_{e_{u,v},i}, v_0$ in $G'$, where $i \leqslant i \leqslant k+1$. Since $S'$ contains at most $k$ edges, at least one of these paths must contain no edges in $S'$. Call this path $P_{u,v}$.
Similarly, there must be a path $P_{v,u}$ from $v_1$ to $u_0$ that contains no edge in $S'$.
But then the sequence $(u_0, u_1), P_{u,v}, (v_0, v_1), P_{v,u}$ defines a cycle that has no edges in $S'$.
So $S'$ is not a feedback arc set.
So any feedback arc set of $G'$ that has at most $k$ edges must be made up of exactly $k$ edges of the form $(u_0, u_1)$, as required.

**Proof that this reduction is parsimonious:**

$\Rightarrow$) Suppose that we have a size-$k$ vertex cover $S$ of $G$. Then the set of edges $S'f(S) = \{(u_0, u_1) \mid u \in S\}$ is a size-$k$ feedback arc set of $G'$, as seen in the FEEDBACK-ARC-SET reduction. eginitemize

$\Leftarrow$) Suppose that we have a size-$k$ feedback arc set $S'$ of $G'$ that only uses edges of the form $(v_0, v_1)$. Then the set $S = g(S') = \{v \in G \mid (v_0, v_1) \in S'\}$ is a size-$k$ vertex cover of $G$.

- We note that $g = f^{-1}$, and so this relation is one-to-one. Moreover, we see by our claim that every size-$k$ feedback arc set of $G'$ uses only edges of the form $(v_0, v_1)$, and so $f$ gives a bijection between the size-$k$ vertex covers of $G$ and the size-$k$ feedback arc sets of $G'$, as required.

Finally, we note that $\langle G, k\rangle = f(\phi)$ is polynomial in the size of $\phi$, and in fact can be calculated from $\phi$ in polynomial time. Moreover, if $G$ has $n$ vertices and $M$ edges, then $G'$ has $2n + m(k+1)$ vertices. We can see by looking at the construction of $f(\phi)$ that $k < n$, and so the total size of $G'$ is $\mathcal{O}(mn)$. Each vertex and edge of $G'$ can be found in polynomial time, and similarly copying $k$ takes polynomial time, and so the entire reduction is polytime.

So #FEEDBACK-ARC-SET is $\#\mathcal{P}$-complete.

■

9. **Bonus** (10 marks — your mark will be rounded to the nearest multiple of 2.5)

Consider the SPIRAL-GALAXIES reduction described in the Polytime_Reduction_Examples_2 handout.

**Note:** *This handout won't be out at the time Assignment 3 is posted, but it will be up soon.*

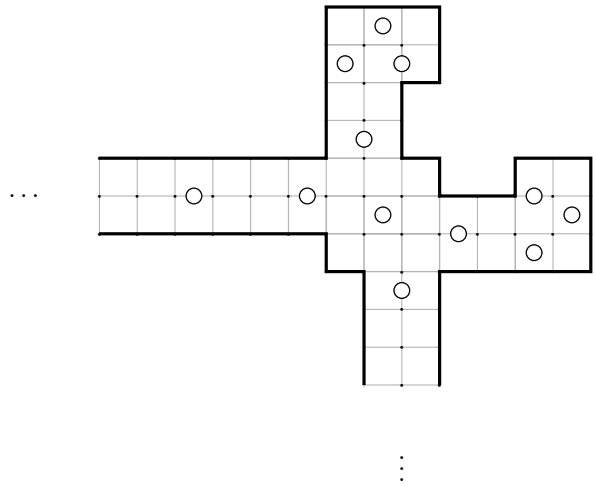
In this reduction we encode a number of widgets that represent wires, inputs, *and*-gates, and other circuit components, and we use those components to build a representation of a 3CNF $\phi$. But in that reduction we miss one component: we don't show a way to bend the wired we build, and so we re forced to use a workaround in which the circuit wires always face in the same direction and move laterally by shifting.

*Technically we'd also need to modify the crossover widget to get this to work as well, but this is the interesting part.*

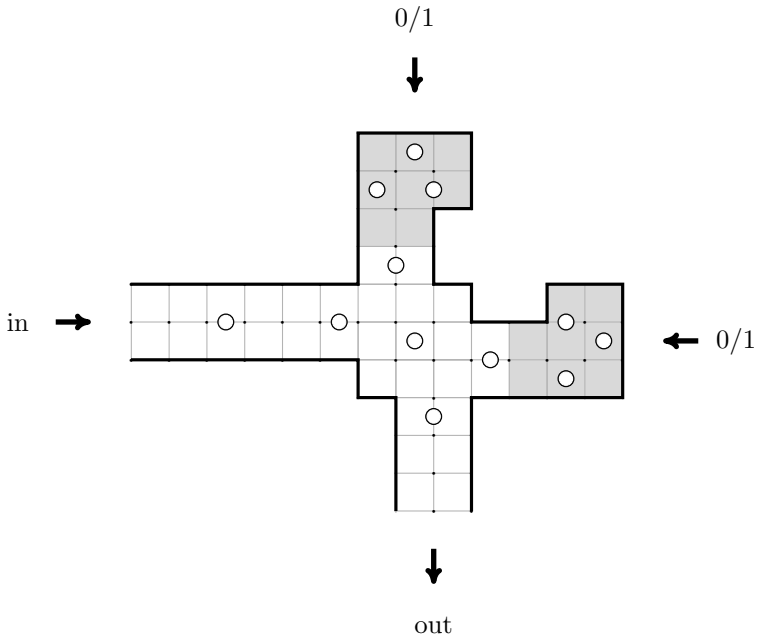
To answer this question, find a way to encode a bent wire (using the endcoding from the reduction) into a SPIRAL-GALAXIES game. Carefully explain why your widget works.

**Solution:**

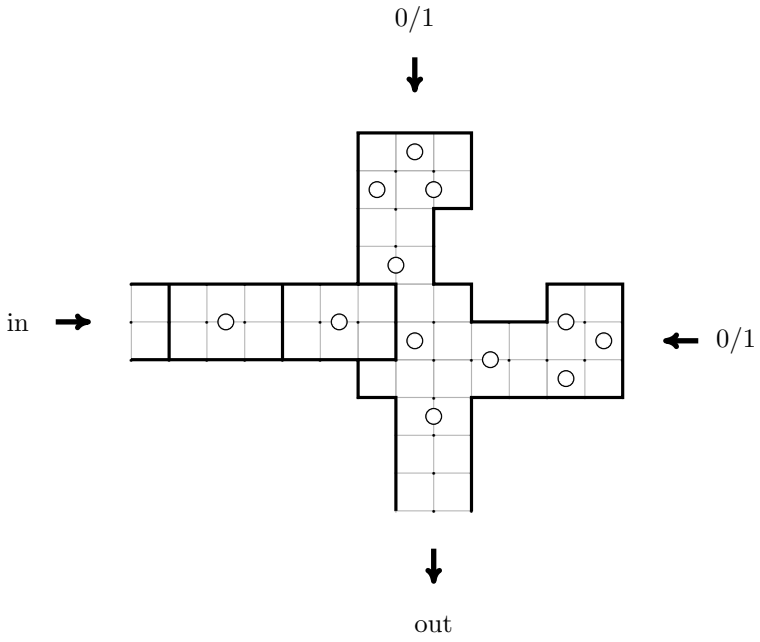
Consider the following widget:



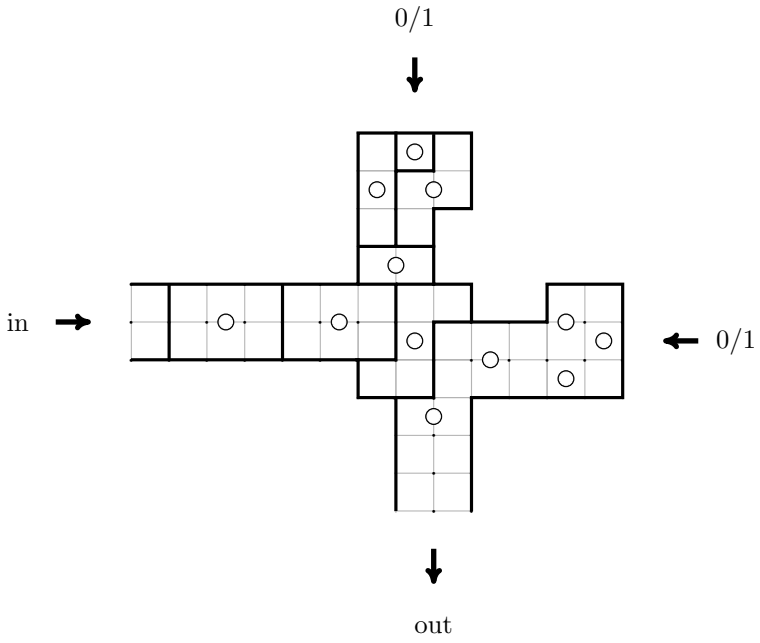
Now, we'll treat the left-hand side of this widget as the signal input, and you'll notice that there are two additional input widgets attached:



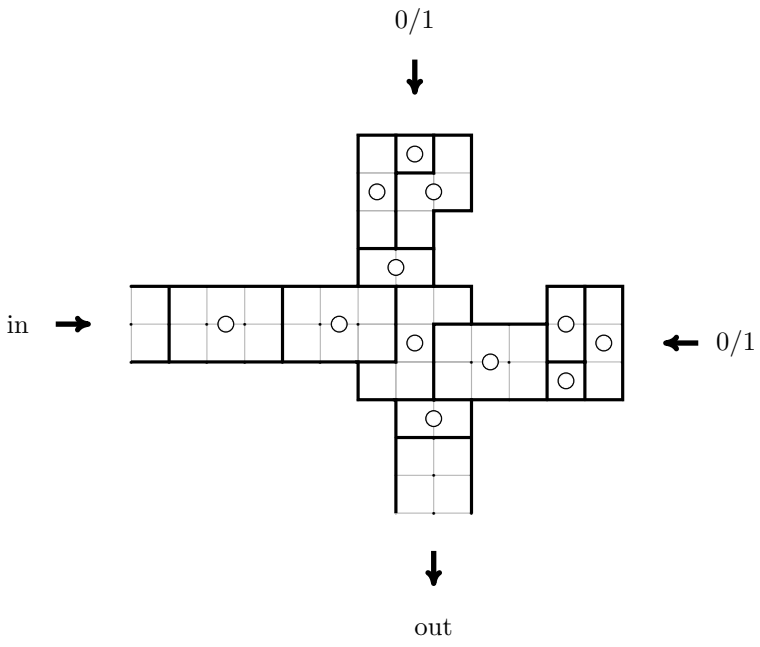
We know from the first Polytime Reductions handout that the signal inputs always either give a *true* or *false* signal, and we can reasonably assume that the signal input is either *true* or *false*. If the signal input is *true*, then we start with the following pattern:



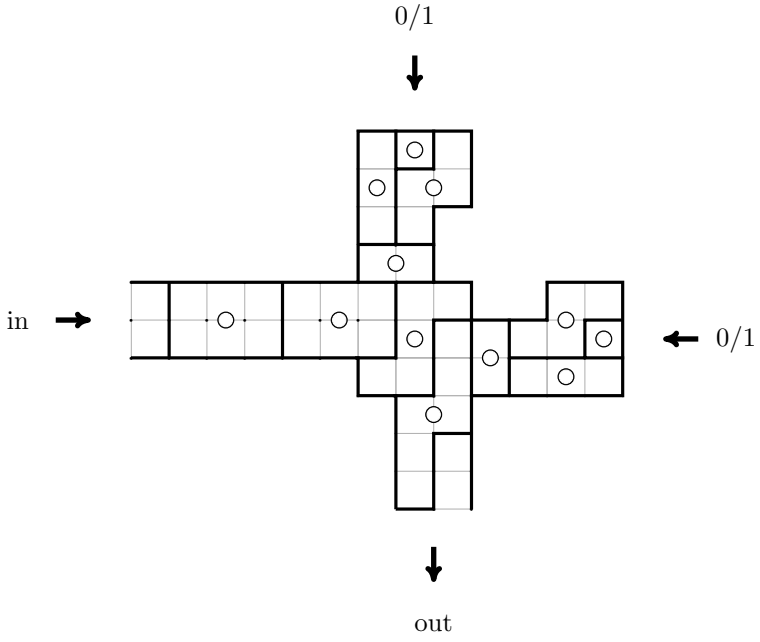

The top signal has to be either *true* or *false*, and so we must have:



Thre are two ways of completing this pattern:

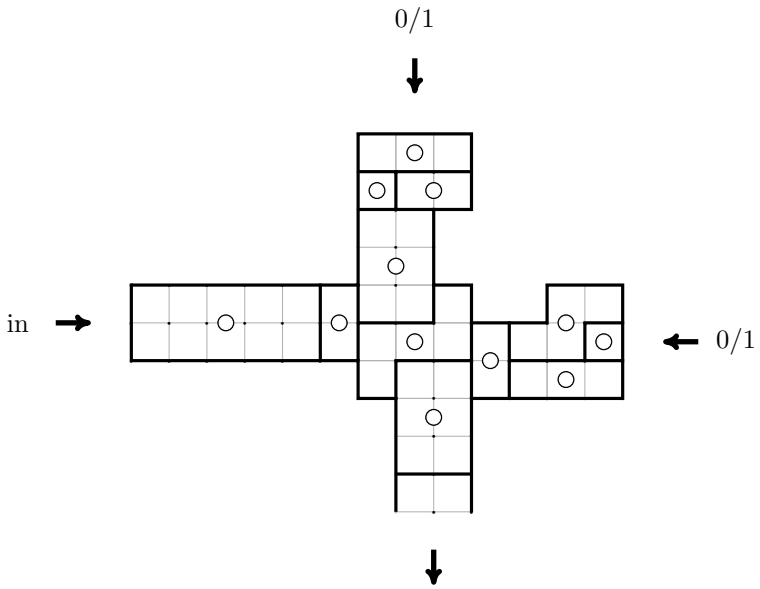



or

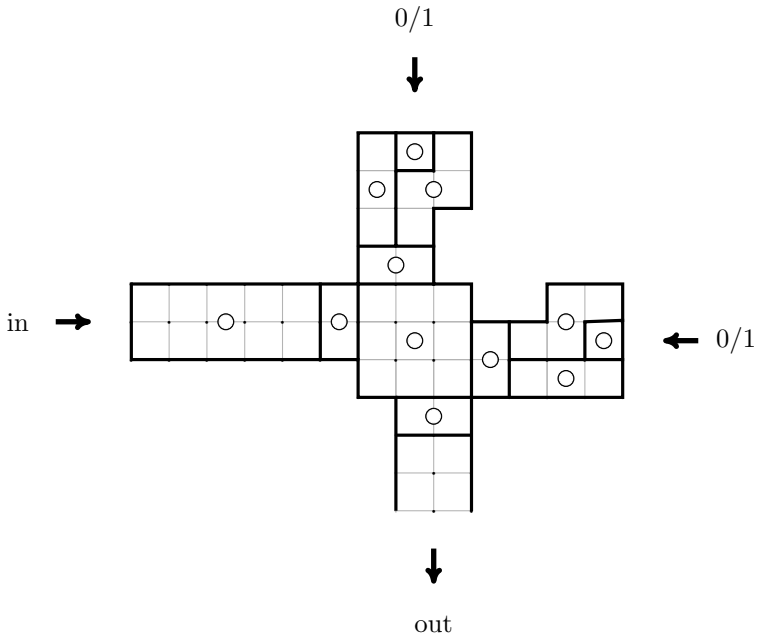



The second pattern does not output a valid signal, but we can use a half-step to the right (using the widget from the top of page 67 of the second Polytime Reductions handout) to prevent this illegal pattern form occuring. So we can effectively only get the first of these patterns.

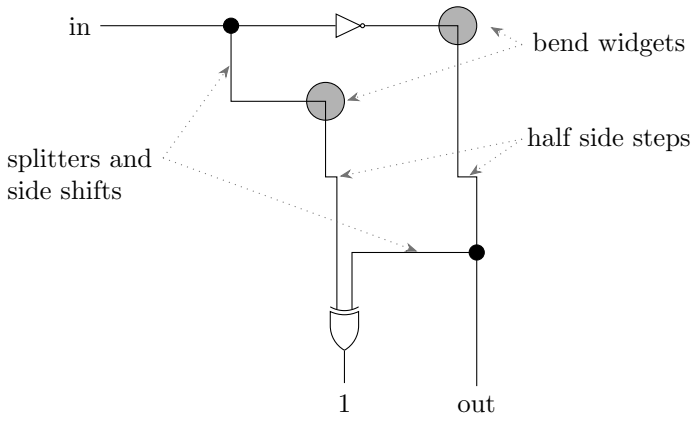
So let's see what happens if we feed a *false* signal into this widget. You'll find that we get one of two outcomes:




or



So if the input is *true* then we output a *false* or an invalid signal, and if the input is *false* we can output either *true* or *false*. We can correct the flipping of the signal using a *not*-gate and correct the two wrong outcomes by using two of these bend widgets in parallel:




We can see that the right half-steps after the bend widgets ensure that the output signal is always a valid signal, and the *xor* gate ensures that the two bend signals output different values, and So whichever bend recieves a *false* signal must output a *true*. Finally, we see that the not gate connected to the output bend widget ensures that the output signal is flipped twice, and so the output is a copy of the original signal flipped by 90 degrees. So this construction forms a bend widget for out wires.

□

> **Note:** *At first glance, some of the submissions look to have working bend widgets that are more efficient than this one. If you are one of the students who has submitted these versions, congratulations!*