

The Simple Reachability Problem in Switch Graphs

Klaus Reinhardt

University of Tübingen

`reinhardt@informatik.uni-tuebingen.de`

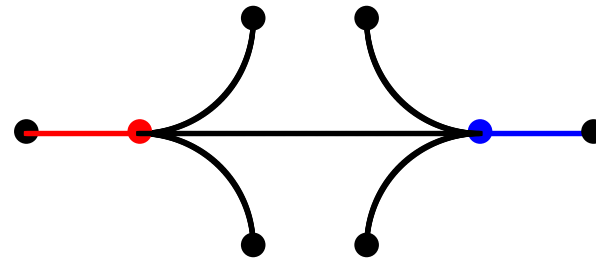
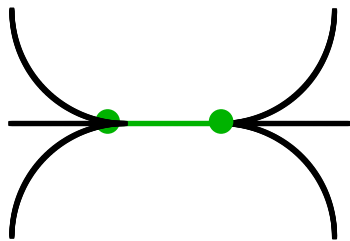
Contents

- Switch-graphs, reachability problems
- NP-completeness in the directed case (idea)
- Algorithm using polynomial time in the undirected case
- Matching, augmenting path
- Bigamist matching
- Further generalizations and open Problems

Simple switch-graph reachability

Definition 1 switch graph: (V, E, o) : “switches”: V , edges:

$E \subset V \times V$, each switch $v \in V$ has an obligatory incident edge $o(v)$.



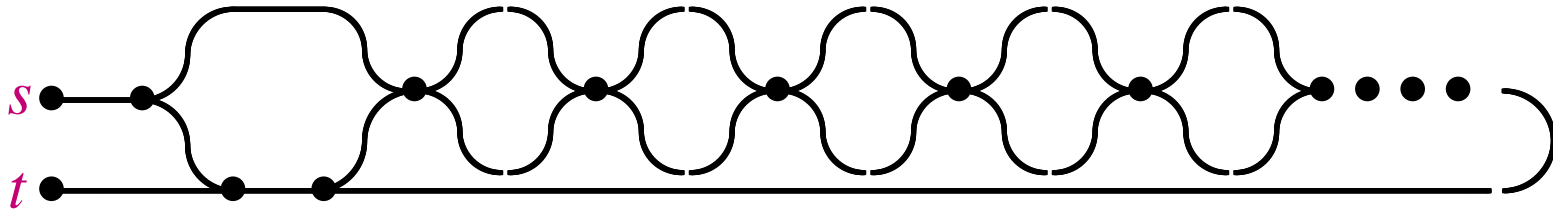
The **simple switch graph reachability** problem is the following:

Given (V, E, o) and $s, t \in V$, is there a simple path from s to t using $o(v)$ for all v in the path.

Dijkstra's algorithm is not sufficient:

We have to avoid using the same edge again in opposite direction.

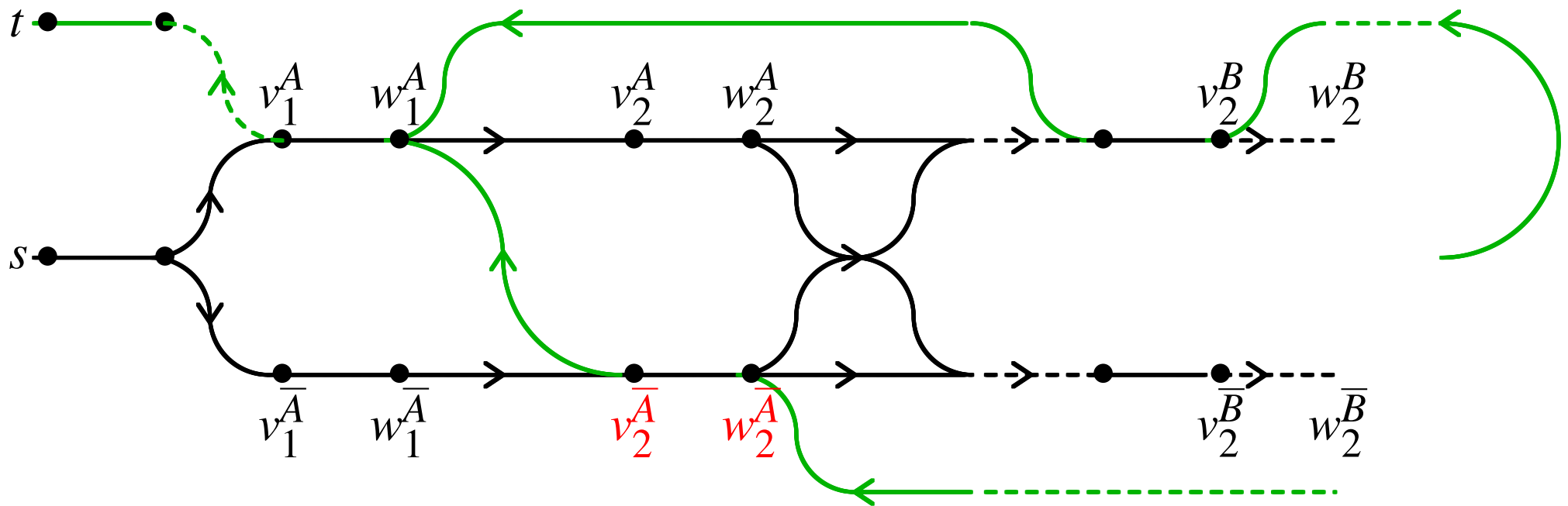
Backtracking algorithm marking the path: exponential



	simple switch graph reachability	graph reachability
directed	NP-complete	NL-complete
undirected	in P	in L [Reingold04]

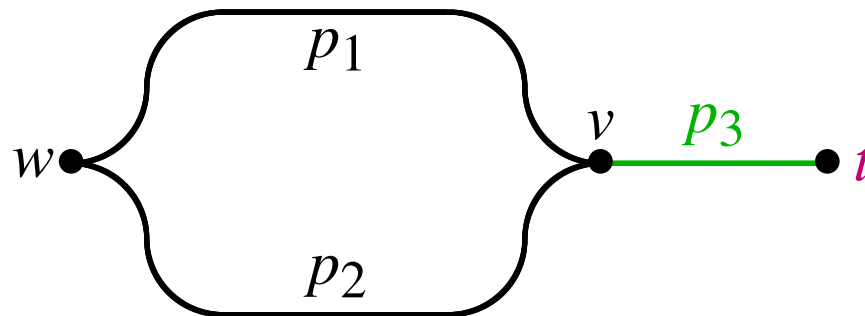
Reduction of SAT to undirected simple switch graph reachability

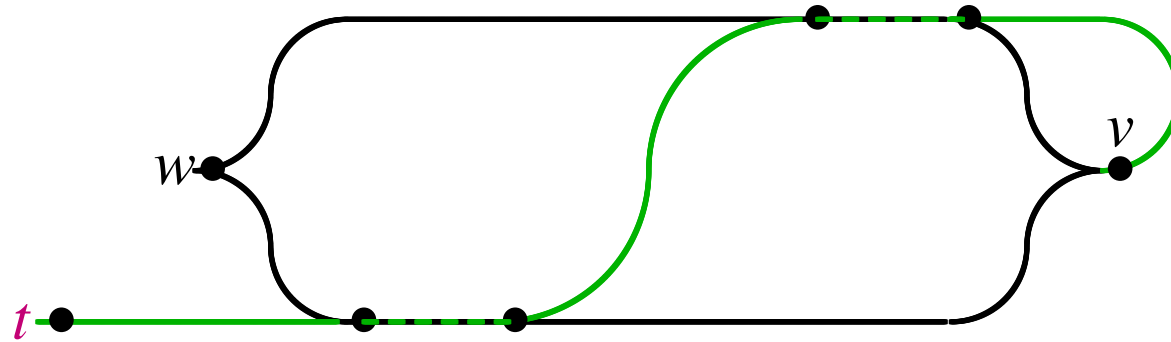
Graph for $\phi = \dots \wedge c_1 \wedge c_2 \dots$ with $c_1 = (\dots \vee A \dots)$ and $c_2 = (\dots \vee \bar{A} \vee B \dots)$:



Algorithm using polynomial time

Lemma 1 *If we have two simple paths p_1 and p_2 from w to v and a simple path p_3 containing $o(v)$ from v to t where p_1 and p_2 do not use the same edge in the same direction, then we can construct a simple path from w to t . Furthermore, the construction takes $\leq |p_1 p_2 p_3|^2$ steps.*





This allows to introduce a “shortcut edge” from w to v .

Theorem 1 *The simple switch graph reachability problem can be solved in polynomial time.*

Function $\text{Visit}(path)$:

Let (x, v) be the last edge in $path$;

If $v = t$ **then** $\text{return}(path)$

else if $o(v) = (v, x)$ **then**

$foundpath := \text{nil}$;

Mark all incident non-obligatory edges “unvisited” for v ;

While \exists “unvisited” edges for v and $foundpath = \text{nil}$ **do**

Let (v, u) be the newest unvisited edge;

if $u \notin path$ **then** $foundpath := \text{Visit}(path \circ (v, u))$;

mark (v, u) “visited” for v ;

$\text{return}(foundpath)$

else

Let $(w', w) = o(w)$ be the last edge in $path$ occurring also in $pathto(v)$ in the same direction

if $w = x$ or $pathto(v) = \text{nil}$ **then**

$pathto(v) := path$;

return(**Visit** $(path \circ o(v))$);

else if $(w, v) \notin pathto(v)$ **then**

add edge (w, v) ; mark (w, v) “unvisited” for w ;

remember both paths from w to v for (w, u) ;

return(nil)

else return(nil)

Function `expand`($path$):

While \exists a last shortcut edge (w, v) in $p = q(w, v)p_3$ **do**

Let p_1, p_2 be the remembered paths from w to v ;

Construct a simple path r from w to t by Lemma 2;

$path := q \circ r$;

return($path$)

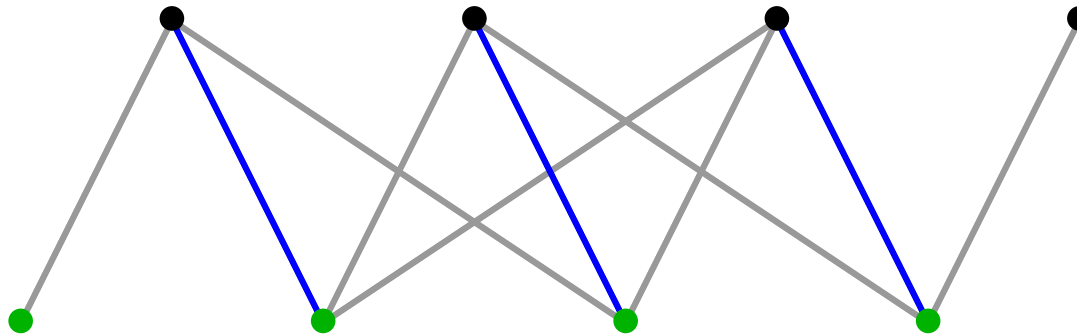
The path `expand(Visit($o(s)$))` is calculated in $O(|V|^4)$ time.

$\Rightarrow O(|V|^5)$ total time needed to iterate the maximal bigamist matching.

Matching

Given: a bipartite graph $G = (F, M, E)$

Wanted: A maximum number of pairwise disjoint **pairs** in E

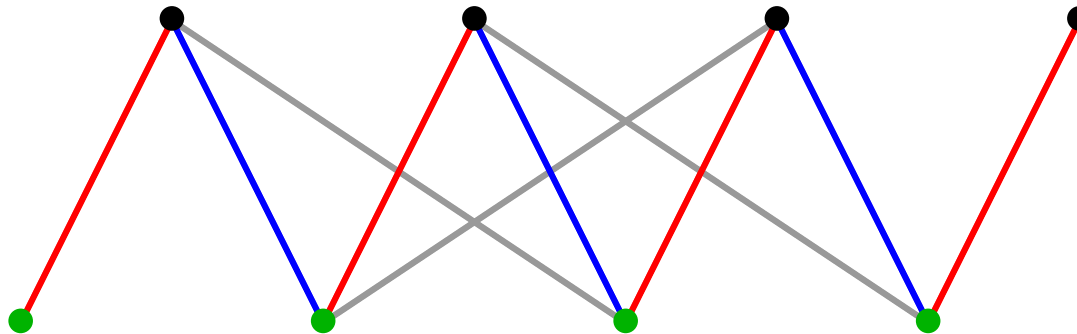


[HK73]: polynomial time algorithm.

Matching, augmenting path

Given: a bipartite graph $G = (F, M, E)$

Wanted: A **maximum** number of pairwise disjoint **pairs** in E

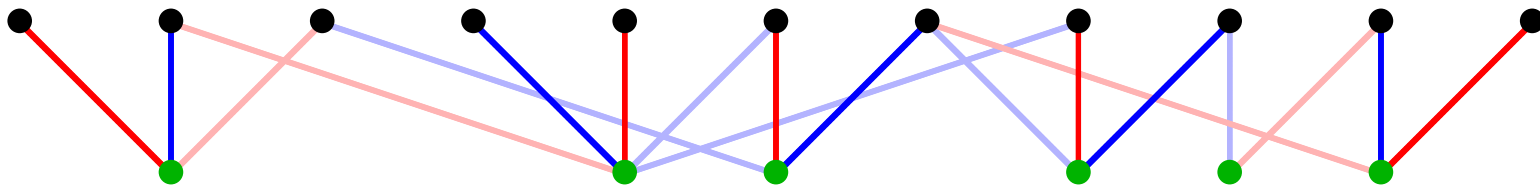


[HK73]: polynomial time algorithm.

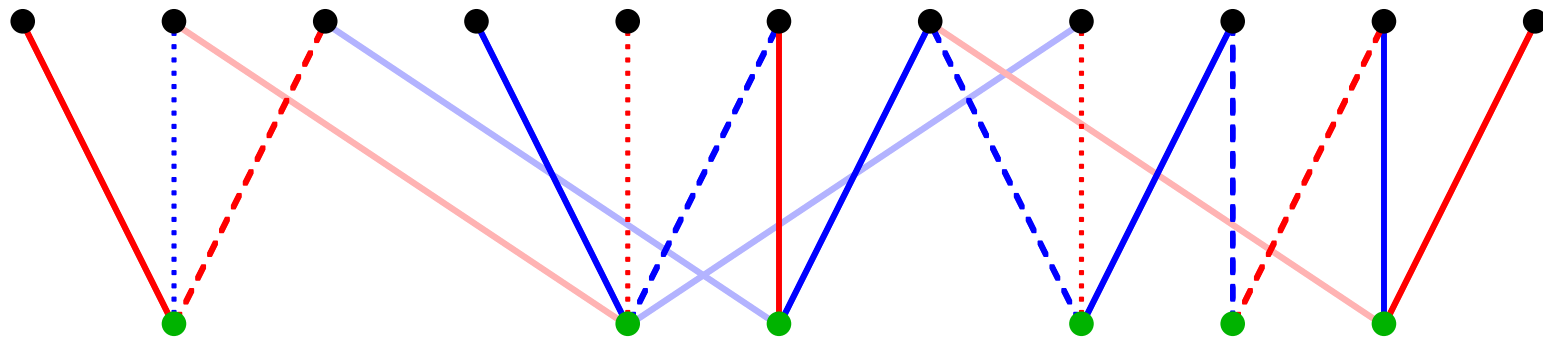
Bigamist matching

Given: a bipartite graph $G = (B, M, E)$, where B is the set of bigamist vertices and M are the monogamists; additionally the edges have 2 colors red and blue thus $E = E_r \cup E_b \subset B \times M$.

Wanted: A maximum number of pairwise vertex-disjoint triples (v, u, u') with $(v, u) \in E_r$ and $(v, u') \in E_b$.



Claim: If the bigamist matching is not maximum, we can always find an augmenting path.



Previously: dotted and dashed. New matching: dotted and solid.

Theorem 2 *The problem of finding an augmenting path for a biga-*

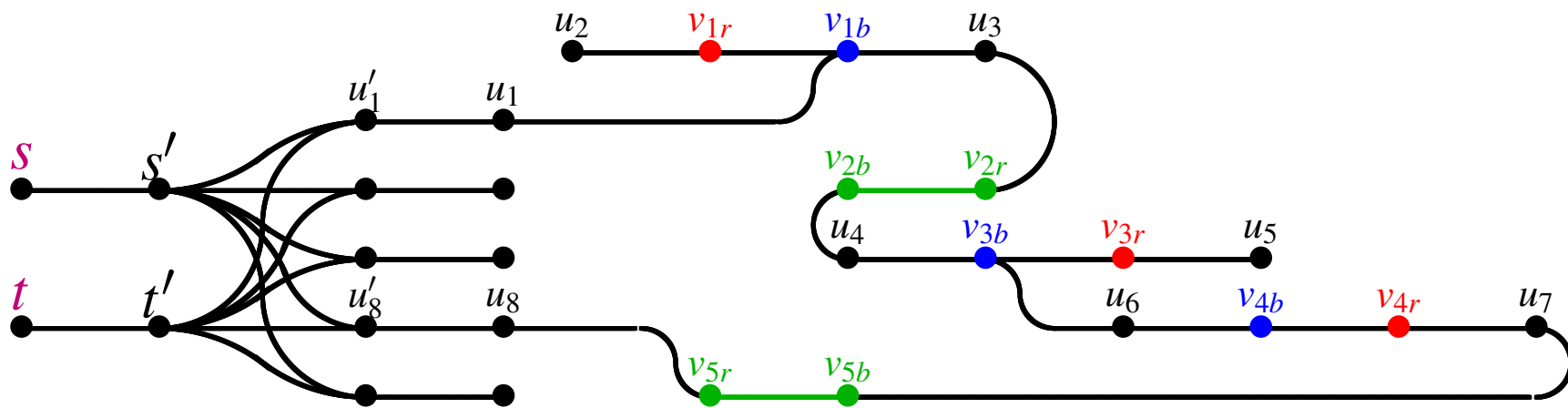
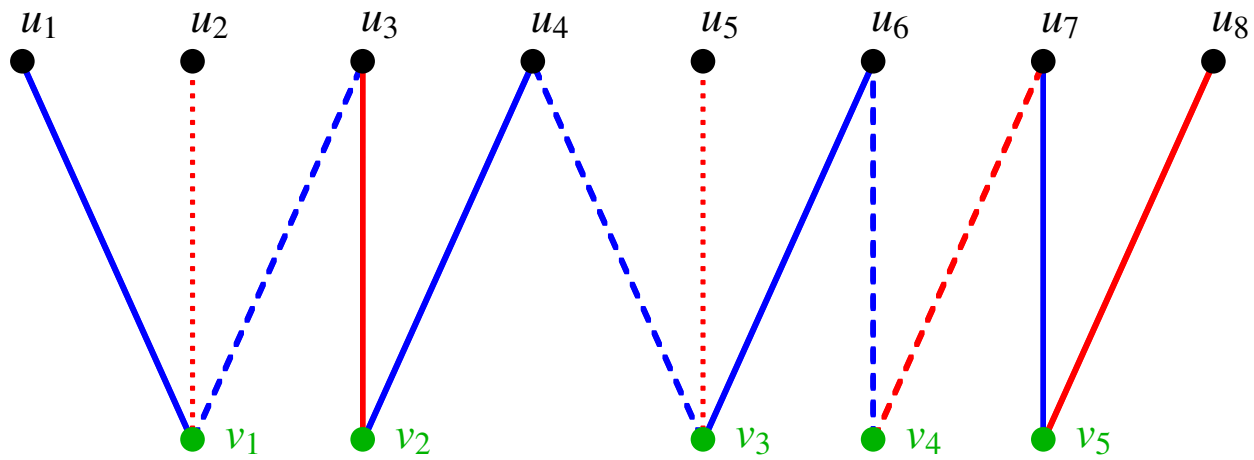
mist matching is logspace and linear time reducible to the simple switch graph reachability problem.

Unmarried bigamist v : v_r, v_b connected by $o(v_r) = o(v_b) = (v_r, v_b)$.

Bigamist v married with u_1 and u_2 : u_1, v_r, v_b, u_2 connected by the edges $o(v_r) = o(u_1) = (u_1, v_r)$, (v_r, v_b) and $o(v_b) = o(u_2) = (v_b, u_2)$.

$(v, u) \in E_r$: the edge (v_r, u) .

$(v, u) \in E_b$: the edge (v_b, u) .



NP-completeness of matching for 3-sexes and Paths of length 2

[GJ78]: 3-dimensional matching (marriage for 3 sexes) is NP-complete.

[KH83]: It is NP-complete for any connected graph G with more than 2 vertices (like the “triangle” $G = K_3$ in 3-dimensional matching) to decide whether a given graph H can be “G-factored” (that means disjoint copies of G form a spanning subgraph of H).

In particular, this holds for $G = P_3$ the path of length 2.

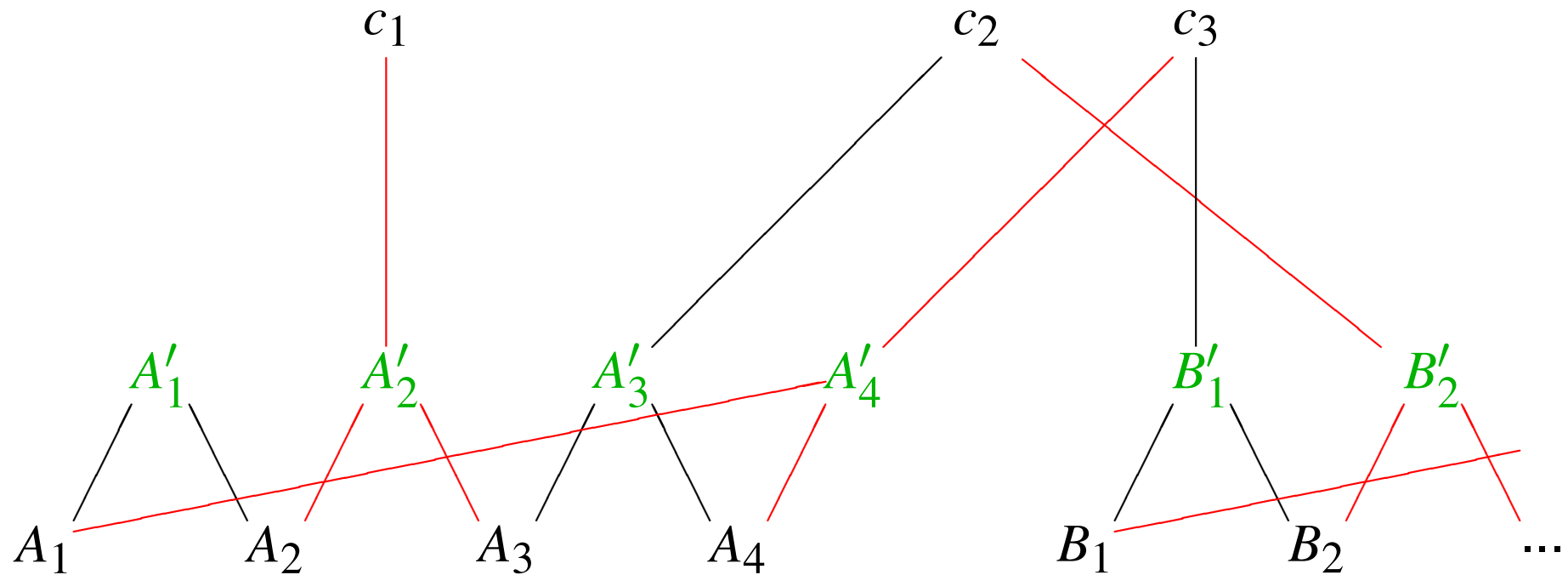
Trigamy is NP-complete

Given: a bipartite graph $G = (T, M, E)$, where T is the set of trigamist vertices and M are the monogamists; the edges are $E \subset T \times M$.

Question: Are there $|T|/2$ pairwise vertex-disjoint quadruples (v, u, u', u'') with $(v, u), (v, u'), (v, u'') \in E$?

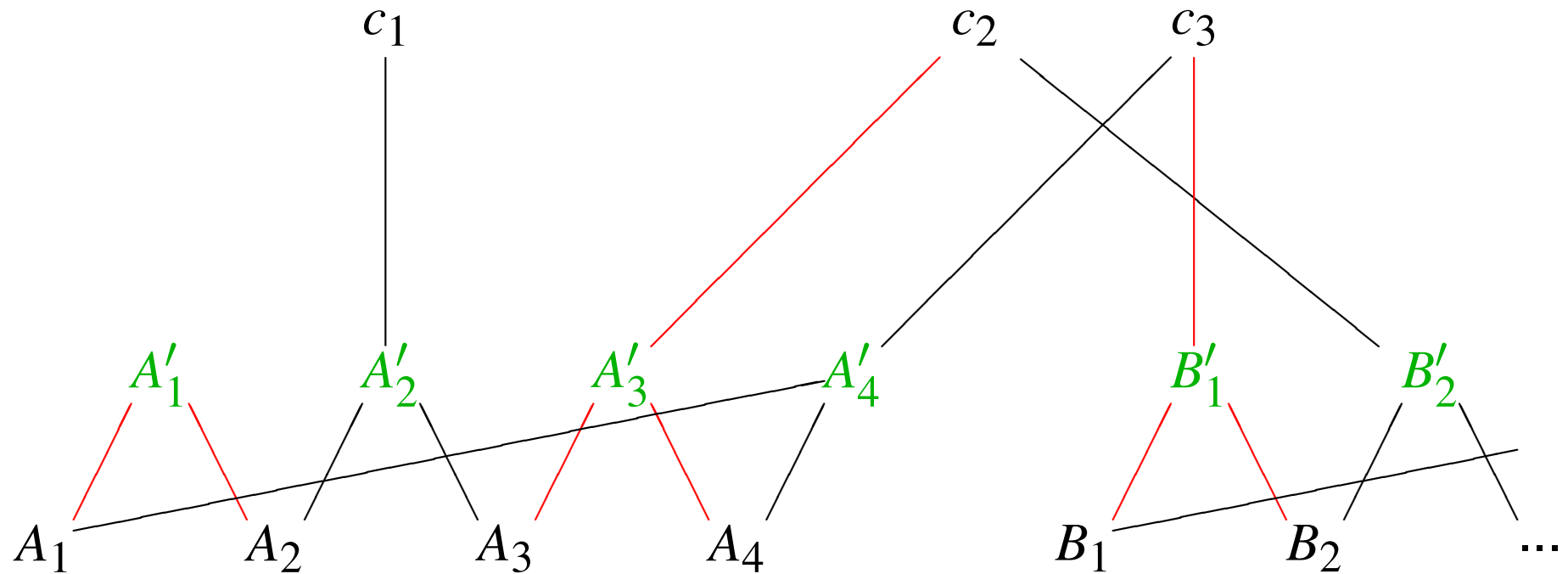
Graph for $\phi = \dots \wedge c_1 \wedge c_2 \wedge c_3 \dots$ with

$c_1 = (\dots \vee \mathbf{A} \dots)$, $c_2 = (\dots \vee \bar{\mathbf{A}} \vee \mathbf{B} \dots)$ and $c_3 = (\dots \vee \mathbf{A} \vee \bar{\mathbf{B}} \dots)$:



Graph for $\phi = \dots \wedge c_1 \wedge c_2 \wedge c_3 \dots$ with

$c_1 = (\dots \vee A \dots)$, $c_2 = (\dots \vee \bar{A} \vee B \dots)$ and $c_3 = (\dots \vee A \vee \bar{B} \dots)$:



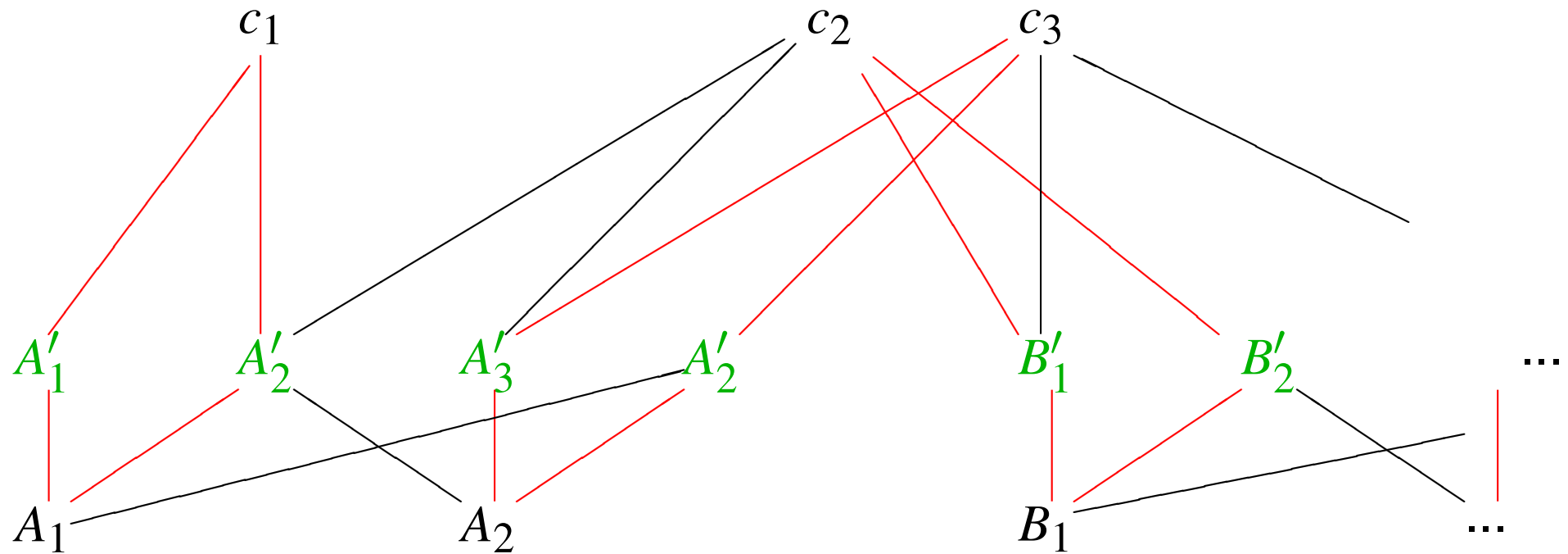
Tetragamy is NP-complete

Given: A bipartite graph $G = (F, M, E)$ with edges $E \subset F \times M$.

Question: Are there $|F|/2$ pairwise vertex-disjoint quadruples (v, v', u, u') with $(v, u), (v, u'), (v', u), (v', u') \in E$?

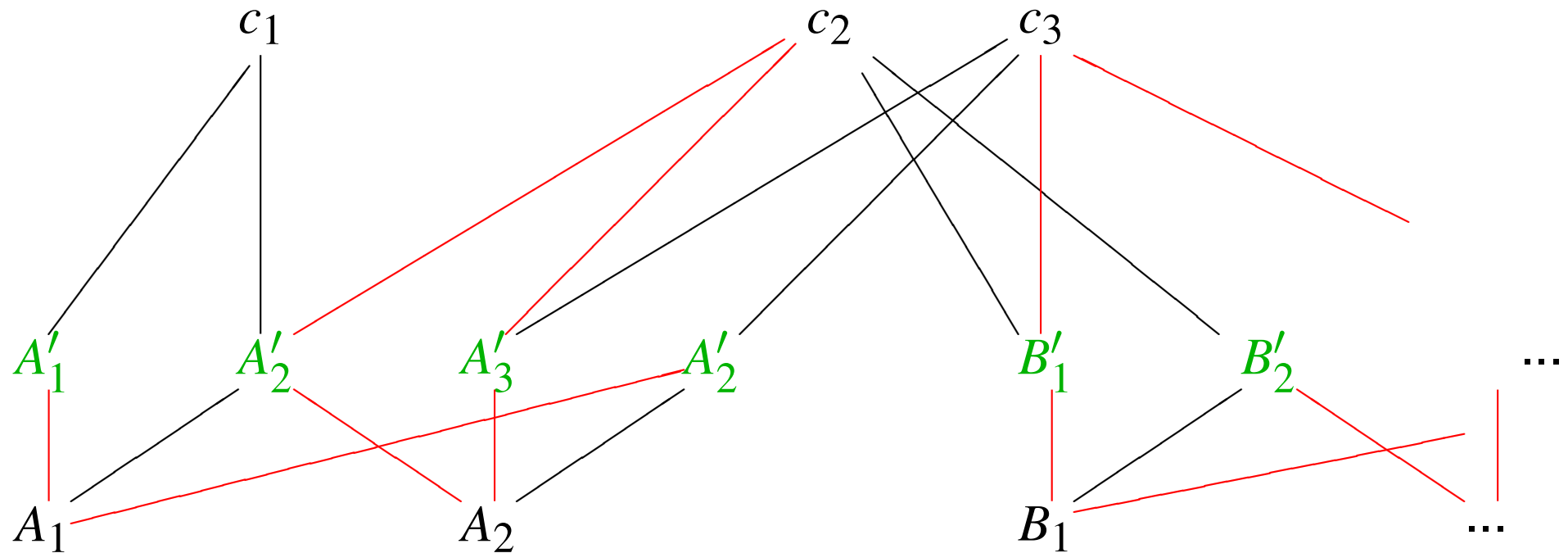
Graph for $\phi = \dots \wedge c_1 \wedge c_2 \wedge c_3 \dots$ with

$c_1 = (\dots \vee \mathbf{A} \dots)$, $c_2 = (\dots \vee \bar{\mathbf{A}} \vee \mathbf{B} \dots)$ and $c_3 = (\dots \vee \mathbf{A} \vee \bar{\mathbf{B}} \dots)$:



Graph for $\phi = \dots \wedge c_1 \wedge c_2 \wedge c_3 \dots$ with

$c_1 = (\dots \vee A \dots)$, $c_2 = (\dots \vee \bar{A} \vee B \dots)$ and $c_3 = (\dots \vee A \vee \bar{B} \dots)$:



Further possible generalizations and open Problems

The algorithm can be extended to chains starting and ending with a monogamist and two bigamists in the middle or also allowing matchings of two monogamists.

Open problem: A general characterization of families of graphs for “factorizing” the input graph in polynomial time.

Open problem: What is the complexity of constructing a stable bigamist matching?

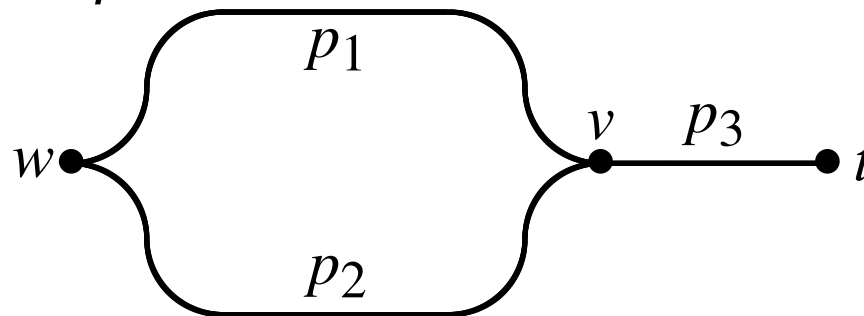
Open problem: What is the complexity of constructing an optimum or a Nash-equilibrium for a bigamist matching with weighted edges?

[ARZ98]: Matching is in nonuniform SPL.

Open problem: Is bigamist matching in nonuniform FSPL? End

Thank you

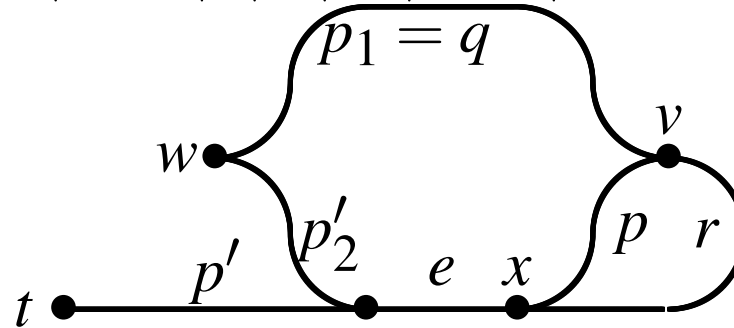
Lemma 2 *If we have two alternative simple paths p_1 and p_2 from w to u and a simple path p_3 containing $o(u)$ from u to t where p_1 and p_2 do not use the same edge in the same direction, then we can construct a simple path from w to t . Furthermore, the construction takes $\leq |p_1 p_2 p_3|^2$ steps.*



Proof: If p_i is disjoint with p_3 for some $i \leq 2$, then we can simply

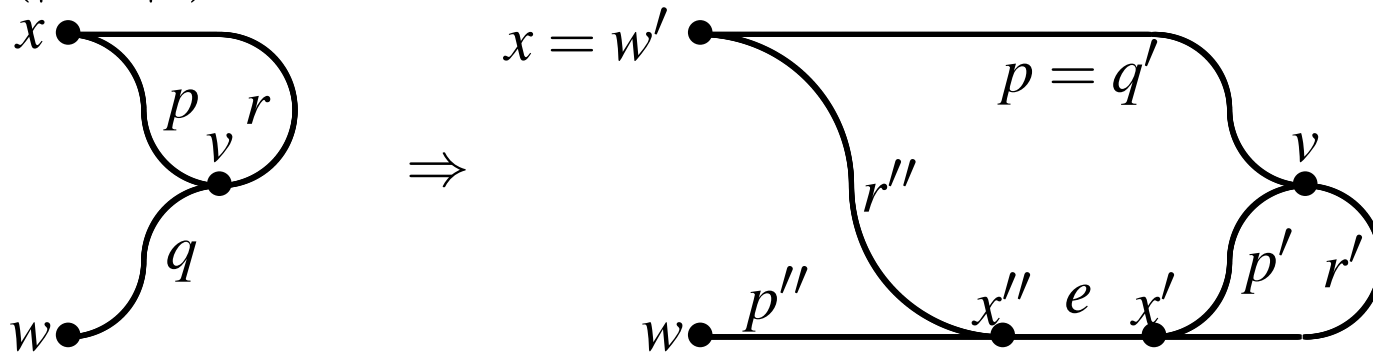
concatenate $p_i p_3$ to a simple path from w to t . Let $e = (x, x')$ be the last edge in p_3 which also occurs in p_i for some $i \leq 2$, that means $p_3 = rep'$ and p' is disjoint with p_1 and p_2 . If e is used in both paths in the same direction, that means $p_i = p'_i ep$, then we can concatenate $p'_i ep'$ to a simple path from w to t . Otherwise, let w.l.o.g. e occur in p_2 in opposite direction (as the following picture shows), this means $p_2 = p'_2 e^{rev} p$. Since p' is disjoint with p_1 and p_2 , we can apply Lemma 3 with $q = p_1$ to construct a simple path from w to x and append ep' .

Finding e requires $|p'|$ times searching through p_1p_2 . Together with $\leq |p_1pr|^2$ from Lemma 3, this is dominated by the time $|p_1p_2p_3|^2 \geq |p_1p_2p'|^2 + |p_1p_2r|^2 \geq |p_1p_2| \cdot |p'| + |p_1pr|^2$. ■



Lemma 3 *If we have a simple path r starting with $o(v)$ from v to x , a simple path p from x to v and a simple path q from w to v , where q and p do not use the same edge in the same direction, then we can*

construct a simple path from w to x . Furthermore, the construction takes $O(|prq|^2)$ time.



Proof: If q and r are disjoint, then qr is a simple path from w to x . Let $e = (x'', x')$ be the first edge in q which also occurs in $p = p_1 e p_2$ or in $r = r' e r''$. In the first case, we must have $q = q_1 e^{rev} q_2$ (not the same direction) and $q_1 e^{rev} p_1^{rev}$ is a simple path from w to x . If, in the

second case, e is used in q and r in the same direction, that means $q = q_1eq_2$, then q_1er'' is a simple path from w to x . In the remaining case (right side of the above picture) we have $q = p''e^{rev}p'$ and, since p'' is disjoint with p and r , we can apply Lemma 3 by induction over the length of r to construct a simple path from $w' = x$ to x' and append it reversed to $p''e^{rev}$.

Finding e requires $|p''| \cdot |pr|$ searching steps. Together with $\leq |pr'p'|^2$ from the recursion, this is dominated by the time $|prq|^2 \geq |prp''|^2 + |prp'|^2 \geq |pr| \cdot |p''| + |pr'p'|^2$. ■

Lemma 4 *Each time the search algorithm in Figure ?? visits a vertex u using an obligatory edge, that means with the path $p(o(u))^{rev} = p(v, u)$, the path p has to be shorter than at the last such visit of u .*

Proof: One of the following two cases must hold:

(i) The edge $o(v) = (v, u)$ was also obligatory for v . This implies either $\text{pathto}(u) = \text{nil}$, which means that u is visited the first time, or the edge $(w', w) = o(w)$ for the vertex w of the edge (w, v) at the end of path $p = \text{pathto}(w')(w', w)$ already occurred in the previous visit of v with the path $\text{pathto}(v)(v, u)$. Then $\text{pathto}(w')(w', w)(w, v)(v, u)$

is shorter than $\text{pathto}(v)(v, u) = p'(w', w)p''(v, u)$ because the algorithm either backtracked on w , which means $\text{pathto}(w') = p'$ and p'' is unequal and thus longer than (w, v) , or $p'' = (w, v)$ and p' is longer than $\text{pathto}(w')$ by induction on the length of the path.

(ii) The path p' with $p = p'(o(v))^{rev}$ has to be shorter than at the last such visit of v by induction on the length of the path. ■

From Lemma 4 it follows that the algorithm will visit every switch at most $|V|$ times using an obligatory edge and thus, at most $|V|^2$ times using an obligatory edge. Finding w can be done in $\leq |V|^2$

steps. Going through all unvisited edges v and u and checking if u is not in $path$ can also be done in $\leq |V|^2$ steps. Thus, the function **Visit** will terminate in $\leq |V|^4$ steps. When a path to t is found, it may contain shortcut edges which have to be expanded by Lemma 2 using the function **expand**.

All edges in p_1 , p_2 and p_3 where visited with a path going through w and therefore p_1 , p_2 , p_3 and r must be disjoint with the path q in function **expand**. Since p_1 and p_2 contain only older shortcut edges than (w, v) , each shortcut edge is expanded at most once in

function `expand`. Thus, at most $|V|^2$ shortcut edges are each expanded in $\leq |V|^2$ steps by Lemma 2. This results in $\leq |V|^4$ steps to calculate `expand` and thus, $O(|V|^4)$ time for calculating the path `expand(Visit($o(s)$))`.