Red Scare! Report

by Johannes Brown Porsgaard, Malte Helin Johnsen, and Nikolaj Munk Binder Jensen.

Results

The following table gives our results for all graphs of at least 500 vertices.

Instance name	n	A	F	M	N	S
bht	5,757	False	0	?!	6	True
common-1-1000	1,000	False	-1	?!	-1	False
common-1-1500	1,500	False	-1	?!	-1	False
common-1-2000	2,000	False	-1	?!	-1	False
common-1-2500	2,500	False	1	?!	6	False
common-1-3000	3,000	False	1	?!	6	False
common-1-3500	3,500	False	1	?!	6	False
common-1-4000	4,000	False	1	?!	6	False
common-1-4500	4,500	True	1	?!	6	False
common-1-500	500	False	-1	?!	-1	False
common-1-5000	5,000	True	1	?!	6	False
common-1-5757	5,757	True	1	?!	6	False
common-2-1000	1,000	True	1	?!	4	False
common-2-1500	1,500	True	1	?!	4	False
common-2-2000	2,000	True	1	?!	4	False
common-2-2500	2,500	True	1	?!	4	False
common-2-3000	3,000	True	1	?!	4	False
common-2-3500	3,500	True	1	?!	4	False
common-2-4000	4,000	True	1	?!	4	False
common-2-4500	4,500	True	1	?!	4	False
common-2-500	500	True	1	?!	4	False
common-2-5000	5,000	True	1	?!	4	False
common-2-5757	5,757	True	1	?!	4	True
gnm-1000-1500-0	1,000	False	1	?!	-1	True
gnm-1000-1500-1	1,000	False	2	?!	-1	True
gnm-1000-2000-0	1,000	False	0	?!	7	True
gnm-1000-2000-1	1,000	False	2	?!	-1	False
gnm-2000-3000-0	2,000	False	0	?!	8	True
gnm-2000-3000-1	2,000	True	2	?!	-1	False
gnm-2000-4000-0	2,000	False	0	?!	6	True
gnm-2000-4000-1	2,000	False	0	?!	5	True
gnm-3000-4500-0	3,000	False	0	?!	10	True
gnm-3000-4500-1	3,000	False	2	?!	-1	True

Instance name	n	A	F	M	N	S
gnm-3000-6000-0	3,000	False	0	?!	6	True
gnm-3000-6000-1	3,000	False	2	?!	6	False
gnm-4000-6000-0	4,000	False	0	?!	7	True
gnm-4000-6000-1	4,000	False	1	?!	15	False
gnm-4000-8000-0	4,000	False	0	?!	5	True
gnm-4000-8000-1	4,000	True	2	?!	6	False
gnm-5000-10000-0	5,000	False	2	?!	5	False
gnm-5000-10000-1	5,000	True	1	?!	5	False
gnm-5000-7500-0	5,000	False	-1	?!	-1	False
gnm-5000-7500-1	5,000	False	-1	?!	-1	False
grid-25-0	625	True	0	?!	324	True
grid-25-1	625	True	0	?!	123	True
grid-25-2	625	True	5	?!	-1	True
grid-50-0	2,500	False	0	?!	1,249	True
grid-50-1	2,500	False	0	?!	521	True
grid-50-2	2,500	False	11	?!	-1	False
increase-n500-1	500	True	2	16	1	True
increase-n500-2	500	True	1	17	1	True
increase-n500-3	500	True	1	16	1	True
rusty-1-2000	2,000	False	-1	?!	-1	False
rusty-1-2500	2,500	False	-1	?!	-1	False
rusty-1-3000	3,000	False	0	?!	14	True
rusty-1-3500	3,500	False	0	?!	14	True
rusty-1-4000	4,000	False	0	?!	13	True
rusty-1-4500	4,500	False	0	?!	7	True
rusty-1-5000	5,000	False	0	?!	7	True
rusty-1-5757	5,757	False	0	?!	7	True
rusty-2-2000	2,000	False	0	?!	5	True
rusty-2-2500	2,500	False	0	?!	4	True
rusty-2-3000	3,000	False	0	?!	4	True
rusty-2-3500	3,500	False	0	?!	4	True
rusty-2-4000	4,000	False	0	?!	4	True
rusty-2-4500	4,500	False	0	?!	4	True
rusty-2-5000	5,000	False	0	?!	4	True
rusty-2-5757	5,757	False	0	?!	4	True
smallworld-30-0	900	False	0	?!	9	True
smallworld-30-1	900	True	1	?!	11	False
smallworld-40-0	1,600	False	0	?!	8	True
smallworld-40-1	1,600	True	1	?!	13	False
smallworld-50-0	2,500	False	0	?!	3	True
smallworld-50-1	2,500	True	2	?!	-1	False
wall-n-100	800	False	0	?!	1	False

Instance name	n	A	F	M	N	S
wall-n-1000	8,000	False	0	?!	1	False
wall-n-10000	80,000	False	0	?!	1	False
wall-p-100	602	False	0	?!	1	True
wall-p-1000	6,002	False	0	?!	1	True
wall-p-10000	60,002	False	0	?!	1	True
wall-z-100	701	False	0	?!	1	False
wall-z-1000	7,001	False	0	?!	1	False
wall-z-10000	70,001	False	0	?!	1	False

The columns are for the problems Alternate, Few, Many, None, and Some. The table entries either give the answer, or contain '?!' for those cases where we were unable to find a solution in reasonable time (because those cases are hard; see below).

For the complete table of all results, see the tab-separated text file `results.txt`.

Methods

In all the following, let G be an instance of a graph with vertex set V and edge set E. Let R be the specified set of red vertices and let $s, t \in V$ be the specified start and end vertices. We fix the notation $n = |V|$, $m = |E|$, and $r = |R|$. Additionally, we fix the notation $v \rightsquigarrow w$ to mean "a simple path from v to w".

Problem: Alternate We solved the problem *Alternate* for all graphs by constructing a digraph G' from the input graph G. We construct G' by excluding any edge between vertices of the same color (that is, excluding any edge $(u, v) \in E(G)$ such that either $u, v \in R$ or $u, v \notin R$). If G is an undirected graph, we then also add the reverse edge (v, u). We then perform a breadth-first search (using `BreadthFirstPaths` in [1]), to determine if there exists a path $s \rightsquigarrow t$ in G', since such a path will necessarily be alternating. If a path exists, we return `true`, else we return `false`. The running time of this algorithm is $O(n + m)$, since constructing G' is $O(n + m)$ and running breadth-first search in G' is also $O(n + m)$.

Problem: Few We solved the problem *Few* for all graphs by constructing an edge-weighted digraph G' from the directed instance G. We add the edge (u, v) with weight 1 to G' for every edge $(u, v) \in E(G)$ where $v \in R$, else we add edge (u, v) with weight 0. For the undirected graph instance G we add all edges described above, as well as (v, u) with weight 1 for edge $(u, v) \in E(G)$ where $u \in R$.

We can then run Dijkstra's algorithm (using `DijskstraSP` from [1]) to find all shortest paths from s in G'. If there exists a path $s \rightsquigarrow t$ in G', the shortest (lightest) path must be the path that uses the least number of red vertices. Furthermore, because we use unit weights for each edge with a red vertex head and a weight of 0 for anything else, the length of the path is equal to the number of red vertices. Therefore a shortest path in G' minimizes the number of red vertices.

If $s \rightsquigarrow t$ exists and $s \notin R$, we return the length (weight) of that path. If $s \in R$ we return the length of the path $+1$ to account for the initial red vertex. In all other cases we return -1.

Constructing G' takes $O(n+m)$ time and Dijkstra's algorithm takes $O(m \log n)$, so the running time of this solution is $O(m \log n)$.

Problem: Many We solved *Many* for all acyclic graphs.[†] In an undirected acyclic graph (i.e. a forest) there exists at most one path $s \rightsquigarrow t$. We therefore use breadth-first search (`BreadthFirstPaths` in [1]) to find $s \rightsquigarrow t$ and return the number of red vertices in the path (that is, $|\{s \rightsquigarrow t\} \cap R|$). The running time of BFS is $O(n+m)$ and so the running time of the solution for forests is $O(n+m)$.

[†]The problem is technically also solvable in polynomial time for graphs that contain self-loops but which are otherwise simple. This is doable by ignoring any edges (v, v) in the input instance. However, we assume all graphs to be simple as stated in the problem description (that is; no parallel edges, no self-loops), even though this is not the case for most of the input files `smallworld-*.txt`

In a directed acyclic graph we assign a weight of -1 to every edge going to a red vertex and a weight of 0 to every other edge. We then perform a shortest-path search from s to t using Bellman-Ford's algorithm (`BellmanFordSP` in [1]), resulting in a path whose negated length is equal to the maximum number of red vertices on an s,t-path. The running time of the solution for DAGs is then $O(nm)$ since that is the running time of Bellman-Ford's algorithm.

We were unable to find a polynomial-time solution for all graphs with a cycle. This is because *Longest-Path* $\leq_P$ *Many*. To see this, consider the following reduction. Given an instance $G(V,E), s, t$ of *Longest-Path*, construct an instance $H(V',E'), s', t', R$ of *Many* as follows: $V' := V$, $E' := E$, $s' := s$, $t' := t$, $R := V$. That is, let every vertex be colored red. A path $s' \rightsquigarrow t'$ in H that maximizes the number of red vertices will necessarily be equivalent to a longest path from s to t, so *Many* is NP-hard.

Problem: None We solved the problem *None* for all graphs by constructing a digraph G' from the directed instance G, adding only edges where the head of the edge is black or the head is t (that is, any edge $(u,v) \in E(G)$ such that $v \notin R$ or $v = t$). For an undirected instance of G, we add all edges described above, as well as edges (v,u) for all edges $(u,v) \in E(G)$ such that $u \notin R$ or $u = t$. That is, G' will contain all edges not going into a red node, unless that red node is t.

We then perform breadth-first search (using `BreadthFirstPaths` in [1]) to determine:

- If there exists a path $s \rightsquigarrow t$ in G' (since such a path will necessarily avoid all $v \in R$ where $v \neq t$)
- If so, the length of the shortest path $s \rightsquigarrow t$ in G'.

If no path exists, we return -1, else we return the length of the path found by the breadth-first search. The running time of this algorithm is $O(n + m)$, since constructing G' is $O(n + m)$ and subsequently running breadth-first search in G' is also $O(n + m)$.

Problem: Some *Some* is solvable using flow for undirected graphs, but is NP-hard in general for directed graphs.

Additionally, for any instance where either or both of s and t are in R, any path $s \rightsquigarrow t$ includes a red vertex, so the answer to "does $s \rightsquigarrow t$ exist?" is the answer to *Some* on all such instances. Our program accomplishes this with breadth-first search (`BreadthFirstPaths` in [1]) in running time $O(n + m)$.

Undirected If the graph is undirected, you can solve it by converting it to a flow problem in the following way:

First, split all vertices into two, and convert to a directed graph G'. For all vertices $v \in V(G)$, create an in-vertex v_{in} and an out-vertex v_{out}, and then create a directed edge (v_{in}, v_{out}) with capacity 1. Then, for all edges $(u, v) \in E(G)$, create two directed edges, (u_{out}, v_{in}) and (v_{out}, u_{in}), each with capacity 1.

Next, create a sink vertex t'. Then create two edges going from the original start and end to the sink, (s_{out}, t') and (t_{out}, t'), each with capacity 1.

Finally, for every red vertex r' in G, find a maximum flow on G' with $r'_{out} \in G'$ as the source vertex. If the max-flow for any red vertex is 2, then there is a simple path from s to t in G that goes through that vertex. Therefore, return true. If no red vertex has a max-flow of 2, return false.

The reason this works is that max-flow in a graph with edges of unit capacity finds edge-disjoint paths between the source and sink. In an undirected graph, the question "is there a simple path between s and t that includes the red vertex r'" is equivalent to "are there two mutually vertex-disjoint paths, $r' \rightsquigarrow s$ and $r' \rightsquigarrow t$". By converting the undirected graph G to the directed graph G' in the manner described above, we convert our problem of *vertex-disjoint paths* to one of *edge-disjoint paths*, since for any path passing through v in G, there exists a corresponding path in G' that includes the edge (v_{in}, v_{out}). Since there is only a single edge (v_{in}, v_{out}), no two paths can both pass over (v_{in}, v_{out}) while remaining edge-disjoint, and edge-disjoint paths in G' correspond to vertex-disjoint paths in G.

We use our own implementation of the Ford-Fulkerson algorithm for finding max-flow, which takes time $O(Cm)$, with C being the maximal flow, and m being the number of edges. The number of edges in G' is $2m + n + 2$. By construction of the graph, the maximum possible flow is 2, so the running time of finding a single max-flow is $O(m + n)$ (which is also the running time of constructing G'). Since we find a max-flow at most once for each of the r red vertices in R, the final running time is $O(r(m + n))$.

Directed For DAGs[††], the problem is quite simple. For each red vertex $r' \in R$, find a path from the start $s \rightsquigarrow r'$ and a path to the end $r' \rightsquigarrow t$. Since the graph is acyclic, the two paths can't share any vertex besides r'. Thus, if you can find both paths for any red vertex, the answer is yes, otherwise it's no. We accomplish this with BFS (`BreadthFirstPaths` in [1]) for a running time of $O(n + m)$.

[††] Again we assume only simple input graphs; no loops, no parallel edges.

In the general case, the problem is hard. In [2], Fortune et. al. proved that, in a directed graph, *pairwise-disjoint paths* is an NP-hard problem (though this was stated with different words). Stating the problem more concretely, let u, v, w and x be four different vertices in a directed graph. Then, finding a simple path $u \rightsquigarrow v$ and a path $w \rightsquigarrow x$, such that the two paths share no vertices, is an NP-hard problem. Call this problem *2-disjoint-paths*.

Fortune et. al. proved this by showing that *3-SAT* $\leq_P$ *2-disjoint-paths*. Here, we will show that *2-disjoint-paths* $\leq_P$ *dir-Some*, where *dir-Some* is *Some* for directed graphs. This will prove the NP-hardness of *dir-Some*.

We will consider en arbitrary instance of *2-disjoint-paths* on a graph G, with vertices V, edges E, and vertices u, v, w, x. The task is to find two pairwise disjoint paths $u \rightsquigarrow v$ and $w \rightsquigarrow x$. To transform this instance into an instance of *dir-Some*, add a single red vertex r', then add two edges, $v \to r'$ and $r' \to w$. Set $s = u$ and $t = x$.

If a simple path exists that starts at u, ends at x and includes a red vertex, it must take the form $p = u \rightsquigarrow v \to r' \to w \rightsquigarrow x$. This is the only possible form a solution to *dir-Some* can take, since r' is the only red vertex, and $v \to r'$ and $r' \to w$ are the only ways to enter and exit r', respectively.

Since p is a simple path, the paths $u \rightsquigarrow v$ and $w \rightsquigarrow x$ must by definition include no repeat vertices and thus be pairwise disjoint.

Therefore, the *2-disjoint-paths* instance is satisfiable iff the corresponding *dir-Some* instance is satisfiable, and since the reduction took polynomial time, *2-disjoint-paths* $\leq_P$ *dir-Some*.

References

[1] *Algs4 library for python 3*, by itu-algorithms, Dec. 22, 2020. [Online]. Available: https://github.com/itu-algorithms/itu.algs4 (visited on 11/23/2023).

[2] S. Fortune, J. Hopcroft, and J. Wyllie, "The directed subgraph homeomorphism problem," *Theoretical Computer Science*, vol. 10, no. 2, pp. 111–121, Feb. 1, 1980, ISSN: 0304-3975. DOI: 10.1016/0304-3975(80)90009-2. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0304397580900092 (visited on 11/23/2023).