Red Scare! Report

Lasse Faurby (lakl), Nikolaj Worsøe Larsen (niwl), Philip Flyvholm (phif), Simon Johann Skødt (sijs) and Thomas Rørbech (thwr).

November 30, 2023

Results

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

Instance name	n	A	F	M	N	S
bht	5757	false	0	?	6	true
common-2-5757	5757	true	1	?	4	true
gnm-5000-10000-1.txt	5000	true	2	?	-1	true
grid-50-1	2500	false	12	?	521	true
increase-n500-3	500	false	0	16	1	true
rusty-2-5757	5757	false	0	?	4	true
smallworld-50-1	2500	true	2	?	-1	true
wall-z-10000	70001	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 the problem is NP-hard.

Methods

The experiments were conducted on a MacBook Pro with an Apple M1 chip (2021) with 16 GB of unified memory. Therefore, any references to the execution time of specific test files apply to this particular machine.

None

For the *None* problem, we solved it by removing all red colored vertices and then running a Breath-First Search on the resulting graph. If a path is found, then the length of the path is returned, otherwise -1 is returned. This solves the problem since removing all red-colored vertices makes it impossible to create a path from start to finish if the only paths available traverse a red vertex. The BFS would then only have the possibility of returning the shortest path without red vertices.

Removing the red vertices takes $O(R)$ time, where R is a subset of all the vertices in the graph where the vertices are red. Running the BFS [flo, 2023] takes $O(V + E)$ time, where V is the number of vertices, and E is the number of edges. Since R is a subset of V, then this means that the total run time of this algorithm is $O(V + E)$. Our implementation took 131.32ms for *common-1-5757* and 64.4ms for *grid-50-1*.

Many

We were unable to solve the *Many* problem for all instances, however, Directed Acyclic Graph instances, can be solved in polynomial running time. In general, this problem is NP-hard. To see this, consider the following:

Given a graph G, which contains n vertices and a subset of red vertices, $R \subseteq V(G)$, return the s, t-path with the maximum number of red vertices.

Longest path problem: Given a graph G, find the longest simple path between two vertices. *"Simple"* referring to the fact that there can be no repeated vertices.

Reduction: The longest-path problem (LP-problem) can be reduced to the given problem (*Many*). Therefore, *Many* is polynomially equivalent to or harder than the LP-Problem: $LP \leq_P Many$.

- Given: A graph G; this is the input instance for the LP-problem.
- Create: A graph G', where all vertices are red, $R = V(G')$. This is an instance of the *Many* problem.
- Use: An algorithm for the *Many* problem, which finds the maximum number of R in an s, t-path in the graph G'. This is the solution S'.
- Deduce: Given the solution S' for the *Many*-problem, use this to solve the LP-problem. Since the graph is unweighted, the s, t-path with the maximum number of red vertices in S' is also the longest path.

In the case of a Directed Acyclic Graph, run the following algorithm [pse, 2011]: Create a Graph G' where all outgoing edges e from red vertices $v \in R$ have a weight of -1, and the remaining edges have a weight of 0. Use the Bellman-Ford algorithm [bel, 2011] to construct a map D containing the distances from the source s to all vertices $v \in G'$. By design of the graph G' and Bellman-Ford, the distance can either be negative or 0. If the distance is negative, the absolute value is equal to the maximum number of red vertices on the path from s to v. Since we use negative edges, Bellman-Ford will find the shortest path, which means a path with maximum red vertices, but a resulting distance that is negative. To answer the *Many* problem, we must assess the distance to t from s in $D(t)$. If the path contains red vertices the distance is negative therefore we have to take the absolute value of $D(t)$. If $t \in R$, then we must add 1 to the distance.

The resulting running time is: Augmenting the graph plus Bellman-Ford $O(R \cdot E) + O(V \cdot E) = O(V \cdot E)$.

Our implementation took 1.87 seconds for *increase-n500-3*.

Some

We were unable to solve the *Some* problem for all types of graphs, however, for Undirected Graphs and Directed Acyclic Graphs instances we would be able to solve this problem. In general, this problem is NP-hard.

Given a graph G, which contains n vertices and a subset of red vertices, $R \subseteq V(G)$, return 'true' if there is a path from s to t that includes at least one vertex from R. Otherwise, return 'false'; the problem is a Decision problem.

Undirected Graphs

For the Undirected Graph instance, we are able to solve the *Some* problem using Ford-Fulkerson. Create a graph G', add a sink node t' and source node s'. Create an edge from s to s' and t to s', and set the capacity on every edge to 1. Then run the following algorithm:

- For each red vertex r in G' until result is 'true'.

- Create a copy of $g = G'$.
- Add an edge e' to G' from r to t' with a capacity $c = 2$.
- Run Ford-Fulkerson.
  - if $flow(G') = 2$ return 'true'.
  - else remove e and continue.

The resulting time complexity is $O(R \cdot E)$. This is only a valid solution for Undirected Graphs since with directed instances it is not ensured that there is a valid path from s to t.

Directed Acyclic Graph

In the instance of a Directed Acyclic Graph, a solution can be achieved by utilizing the Bellman-Ford algorithm and an augmented graph as in the solution for the **Many** problem. In particular, create a *graph* G' where all out-going edges from red vertices have a weight of -1 and everything else is 0. Run the Bellman-Ford algorithm to construct the distance map D containing the distance from s to all vertices $v \in G'$. Finally, a solution is found by evaluating the expression $D(t) < 0$. if $D(t) < 0$ there must be a path from s to t with 1 or more red vertices, in that case, return 'true' else 'false'. This results in a running time of $O(V \cdot E)$.

Our implementation took 25.24 seconds for *common-2-5757* and took 8,4415 minutes for *wall-z-10000*.

Few

We solved the *Few* problem for all instances using the greedy algorithm Dijkstra [dij, 2011] for path finding. We set all the weights on the edges to zero, except for the outgoing edges from red vertices; they are set to one. This means that Dijkstra finds the shortest path between s and t, avoiding red vertices as much as possible. So if there is a path without a red vertex, the algorithm finds it. If there is no path, it returns -1. The running time of this algorithm is the same as that of Dijkstra, which means that the running time is therefore $O(E \log(V))$.

Out implementation took 5.79 seconds for *common-2-5757* and 6.06 seconds for *rusty-2-5757*.

Alternate

For the *Alternate* problem, we solved it by starting at the start node and removing all neighbours of the same color, and then iteratively did the previous step for neighbouring nodes. After this, a Breath-First Search was run on the resulting graph. If a path is found, we would return true, otherwise false. This solves the problem since only paths with alternating colors can be available after removing neighbouring nodes of the same color. Thereby any path from the Breath-First Search would result in a valid path. The worst-case running time of removing the neighbouring nodes of the same color takes $O(V + E)$, where V is the number of vertices, and E is the number of edges. This is the same running time as running the Breath-First Search. This means that the total running time of this algorithm is $O(V + E)$.

Our implementation took 6.99 seconds for *common-2-5757* and 0.01ms for *rusty-2-5757*.

References

[bel] Bellman-Ford algorithm, 2011. Github, `https://gist.github.com/ngenator/6178728`.

[dij] Dijkstra implementation, 2011. Stack Overflow, https://stackoverflow.com/questions/22897209/dijkstras-algorithm-in-python.

[pse] Graph serialization, 2011. Stack Overflow, https://stackoverflow.com/questions/4168/graph-serialization/4577#4577.

[flo] Flow Assignment, 2023. Github, https://github.com/simonskodt/aldes-flow-behind-enemy-lines.