Symmetric Graph Algorithms

Brent Kirkpatrick

November 15, 2021

Intrepid Net Computing

www.intrepidnetcomputing.com

Document Revision History

October 12, 2021 Drafted.

November 15, 2021 Published original on www.intrepidnetcomputing.com

Abstract

Every graph algorithm can be revised to consider symmetries. Whether considering the symmetries improves the running time depends on the algorithm and the problem instance. Here, we consider some classic graph algorithms: max clique, independent set, traveling sales person, max cut, and graph coloring. All of these algorithms are classic exponential-time algorithms for NP-hard problems. In the case where a graph with symmetries is presented to the algorithm, we can sometimes dramatically improve the running time.

Graphs are discrete objects having nodes and edges. These objects can guide a computation. For example, we can give a graph as an input to a shortest-path algorithm. When a graph has structure, the computation can sometimes be improved. For example, a lattice is one of the most useful structures for computation. Once we discover the structure in a graph, we can make explicit the running-time improvements obtained for a specific graph.

Background

Recent results show that graph isomorphism is a problem with a polynomial-time algorithm [1]. This follows on a decades-long discussion of whether graph isomorphism is NP-complete. We know of no other problem that was conjectured to represent a new complexity class [2, 3]. As far as we know, every problem explored is either in P or in NP.

Now we define graphs and isomorphisms of graphs. Let $G = (V, E)$ be an **undirected graph** where V is a set of n vertices and U is a set of edges. Let each edge be a tuple of vertices $(u, v) \in E$ where $u, v \in V$. Now, we define the graph isomorphism problem. Let $G = (U, V)$ and $G' = (U', V')$ be two undirected graphs. Let an **isomorphism** be a bijection $\phi : U \to U'$ such that edge $(u, v) \in E$ if and only if $(\phi(u), \phi(v)) \in E'$. These two graphs, G and G' are **isomorphic** if and only if there exists an isomorphism $\phi : U \to U'$.

Now we define the crucial aspect of graphs where NP-complete problems can have improved running time with existing algorithms. Let an **automorphism** be an isomorphism of a graph to itself. The identity map is always an automophism. A graph is **symmetric** if there is an automophism that is not the identity. Let A^G be the set of all automorphisms of graph G.

It is easy to see that automorphisms can sometimes simplify an NP-complete problem. If we find a solution to the problem on part of the graph, we can use the automorphism to 'project' that solution onto the symmetric portions of the graph. This possibly improves the running time, because there is a polynomial number of automorphisms [4].

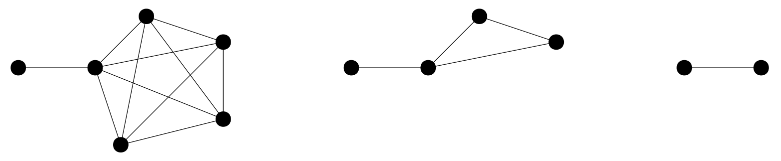

Figure 1: **Canonical Graphs.** (left) The original graph is a hammer graph, which is a clique with a path coming off of one node. This graph has several canonical graphs. (middle) One canonical graph is produced by reducing the five-clique to a three-clique. This canonical graph is produced by a group of two automorphisms. (right) The minimal canonical graph is shown with the five-clique reduced to a single node.

Now, we define a new object to help us understand NP-complete problems and their solutions on symmetric graphs. Let a **canonical graph** be a graph obtained from undirected graph G by applying each automorphism, $a \in A^G$, to G and collapsing node $a(u)$ onto node u, removing node $a(u)$ and all edges

$(a(u), x)$ for all $x \in V$. For example, in Figure 1 we see a graph that has two automorphisms removed. Counting the automorphisms is a topic of a different paper [4], as there is one isomorphism that describes the minimal canonical graph. However, under one attempt at writing the automorphisms for Figure 1, we can say that we have removed two of them from the original graph to obtain the middle graph and that we have have removed all of them to obtain the graph on the right. A canonical graph can be obtained from G in polynomial-time, since there are at most a polynomial number of automorphisms.

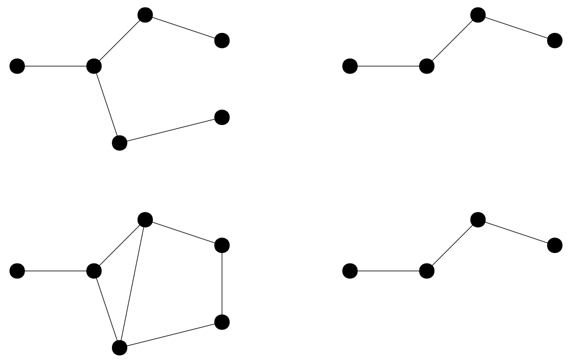

Figure 2: **Canonical Graphs.** (left) The original graph is given. (right) The minimal canonical graph is shown.

To improve the running time of a graph algorithm, we need the **minimal canonical graph** which is defined as a canonical graph produced by removing all of the automorphisms. For example, we can see two minimal canonical graphs in Figure 2. A canonical graph is given as the problem instance to the classic graph algorithm. The size of a canonical graph determines the running time of the classic graph algorithm on symmetric graphs.

Graph Algorithms

Each of these classic graph algorithms solve an NP-complete problem. Each algorithm is guaranteed to have an exponential running time when given a non-symmetric graph as input. In the case where a symmetric graph is input to one of these graph algorithms, the running time of the algorithm may improve. We now discuss the running-time improvements of five classic algorithms. Such improvements can be obtained for any NP-complete problem by using reductions.

Max Clique

Let a **clique** of an undirected graph G be a collection of vertices $C \subseteq V$ such that for every $u, v \in C$, $(u, v) \in E$, meaning that all the vertices in C are connected by edges. This makes C a complete sub-graph in G.

The **maximum clique problem**, or max clique, is to find a clique C of G such that for every set of vertices $C' \subseteq V$ with $|C'| > |C|$, C' is not a clique. The clique C is a maximum clique.

Now we consider a symmetric graph. For the canonical graph, we use max clique to obtain a list of edge disjoint cliques, so that every edge is in some clique. This is done by starting with every edge in a separate clique, and cliques are merged iteratively. These are maximal edge-disjoint cliques. Iteratively consider each automorphism adding nodes/edges. Each clique in the canonical graph is a clique in the original graph. The list of maximal edge-disjoint cliques from the canonical graph is updated by the automorphisms in polynomial time, to obtain the list of edge-disjoint cliques for the original graph. The largest clique in the list is the max clique of the original graph.

Independent Set

An **independent set** of an undirected graph G is a collection of vertices $I \subseteq V$ such that for all $u, v \in I$, $(u, v) \notin E$, meaning there are no edges in G between vertices of I. The independent set consists of vertices such that for all $(u, v) \in E$, either $u \in I$ or $v \in I$.

The **maximum independent set problem**, or independent set, is to find the maximum number of vertices, $I \in V$ such that I is an independent set of G. This means that for all other independent sets, I', $|I'| \leq I$.

For a symmetric graph, we compute an independent set as follows. From the canonical graph, compute an independent set with the most adjacent edges in the original graph. Now, we adjust that independent set as we add edges/nodes from the original graph. For each automorphism, we consider two steps. If the node added by the automorphism is adjacent to an independent set node, leave the set. If the node added by the automorphism is not adjacent to an independent set node, add this node to the independent set.

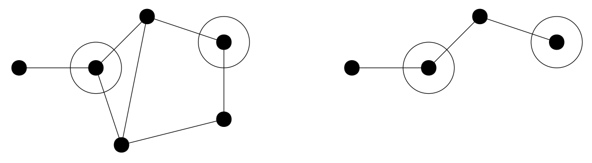

Figure 3: **Independent Set.** Here we have an example using an independent set from the canonical graph to obtain an independent set for the original graph. (left) Shows the original graph and the independent set. (right) Shows the canonical graph and the independent set.

As an example, we see that there are two covers possible for the canonical graph in Figure 3. The cover shown is the one with the most adjacent edges in the original graph. As shown in Figure 3 the cover can be modified using the algorithm above, to include all the edges (and nodes) in the original graph. We see that the cover remains the same.

Traveling Sales Person

First we define a weighted graph, symmetries of a weighted graph, and a reduction for graph isomorphism on a weighted graph. Then we can discuss the definition of the traveling sales person problem. Finally, we can give a solution to the symmetric traveling sales person problem.

First, we need to define a weighted graph $H = (V, E, w)$ where V are the vertices, E are the edges and W is the weight function, $w : V \times V \to \mathbb{R}_{>0}$. The weights are positive reals and are present for each edge, meaning $w(u, v) \in \mathbb{R}_{>0}$ for each $(u, v) \in E$ and $\mathbb{R}_{>0} = \{x \in \mathbb{R} | x > 0\}$.

Now, we discuss symmetries of weighted graphs. For a weighted graph, an automorphism, $a : V \to V$, is a bijection such that $(u, v) \in V$ is an edge with weight $w(u, v)$ if and only if $(a(u), a(v)) \in V$ is an edge with weight $w(u, v)$. For every automorphism defined, $a \in A$, making A the set of automorphisms of weighted graph H. It is evident that if an undirected graph G has a polynomial-sized set of automorphisms, then weighted graph H with the same edges as G has a polynomial-sized set of automorphisms.

The reduction for graph isomorphism on a weighted graph is a straight-forward affair. First, we take H, our weighted graph, remove the edge weights to obtain G, and compute graph isomorphism of G and G to obtain the automorphisms. This is done in polynomial time. Then, we check the polynomial-sized list of automorphisms of G to see if the edge weights match for each automorphism, keeping only the automorphisms where the edge weights match. This is done in polynomial time.

A **tour** of a weighted graph G is a ordered collection of vertices $T \in V$, such that for each $t_i \in T$, there is an edge $(t_i, t_{i+1}) \in E$ for $i < |T|$ and an edge $(t_{|T|}, t_1) \in E$. This means that between each ordered pair of vertices in T, there is an edge, and there is an edge from the last vertex in T back to the first vertex in T. A tour describes a loop of edges in G.

The **traveling sales person problem**, or traveling sales person (TSP), is to find the shortest weight tour that connects a list of input vertices.

Now we consider how to obtain a solution to a problem instance with a symmetric graph. Given the canonical graph, we compute a TSP circuit. Now, we use symmetry to choose the reflected circuit. Let a **bridge edge** be an edge between two TSP nodes. Pick bridge edges to "short" circuit between the reflected tour and the canonical tour. Let all these edges be in the **candidate edge** set. Select the subgraph created by the nodes adjacent to the candidate edges. Now find a minimum spanning tree of the subgraph. Finally, close the tour with one of the edges in the candidate edge set.

Max Cut

Given an undirected graph G and vertices $u, v \in V$, a **cut** is the minimum number of edges C on paths between u and v that when removed from the graph, yield two separate connected components, $C_u \in V$ and $C_v \in V$. This means that in a connected graph, the cut between any pair of vertices must be at least one edge.

The **max cut problem**, or max cut, is to find the pair of vertices $u, v \in V$ such that the minimum cut is of maximum size. This means that for all other $p, q \in V$ such that $p \notin \{u, v\}$ and $q \notin \{u, v\}$, the cut for p, q has fewer edges than the cut for u, v.

Now, we consider the max cut of a symmetric graph. Take the canonical graph. Choose a max cut of the canonical graph with the most adjacent edges in the original graph. An edge in the original graph is adjacent to the max cut if it is incident on one of the two nodes of the cut. This cut is the max cut of the original graph.

Graph Coloring

A **coloring** of an undirected graph G is a minimal mapping of the vertices $c : V \to \mathbb{Z}$ such that for all edges $(u, v) \in E$, the color $c(u) \neq c(v)$ do not match. This means that the colors are represented as integers in $\mathbb{Z}$. We want the minimum number of colors, such that every edge connects vertices of different colors.

The **graph coloring problem**, or graph coloring, is to find the coloring of a graph. The number of colors in the graph coloring is k, and the coloring is called a k-coloring.

To consider the solution for a symmetric graph, we do the following. Pick a k-coloring of the canonical graph. Add nodes from the original graph using the automorphisms. If the node to be added is adjacent to all k colors, add a new color $k' = k + 1$ and color that node the new color. In the next iteration of adding a node k' becomes k.

Conclusions

In the context of graphs, symmetries are easy to understand. We can readily see where symmetries can save us compute time and can improve the running-time. The canonical graph is the graph on which the exponential-time algorithm can run, and the automorphisms provide a convenient description for 'projecting' the solution to the entire graph. In this way, we can conveniently find solutions for very large graphs.

In the context of 3SAT, bin-packing, integer programming, or subset sum, the symmetries are less apparent. However, using the reductions, we can easily see how the graph representations of solutions provides symmetries that can be exploited to improve running times. Each of these logical problems also has symmetries.

We can now characterize the running-time of problem instances when they are input into an algorithm for an NP-complete problem. The space of symmetric problem instances can have faster than exponential running times.

References

[1] B. Kirkpatrick. A polynomial-time algorithm for graph isomorphism. *www.intrepidnetcomputing.com*, 2016.

[2] R. Uehara, S. Toda, and T. Nagoya. Graph isomorphism completeness for chordal bipartite graphs and strongly chordal graphs. *Discrete Appl. Math.*, 145:479482, January 2005.

[3] B. Kirkpatrick, Y. Reshef, H. Finucane, H. Jiang, B. Zhu, and R. M. Karp. Comparing pedigree graphs. *Journal of Computational Biology*, 19(9):998–1014, 2012.

[4] Rudolf Mathon. A note on the graph isomorphism counting problem. *Inf. Process. Lett.*, 8(3):131–132, 1979.