Symmetry-based Graph Clustering Partition Stability

Fabian Ball and Andreas Geyer-Schulz

Abstract Stability of clustering partitions (e. g. Gfeller et al., 2005; von Luxburg, 2010) and the problem of how to define good clusters (e. g. Hennig, 2015) are recurring topics in data analysis. We present a novel approach to characterize the stability of graph clustering partitions that is based solely on the graph's automorphism group. All in all, three formally equivalent definitions are given, each from a different point of view. Two of these conditions are exploited for the design of an algorithm for fast stability detection of a graph clustering partition. These characterizations can be perfectly combined with others in terms of an additional constraint that a "good" clustering solution must fulfill. Our propositions are likely to be generalized for other data formats than graphs, provided the automorphism group is known.

Fabian Ball
Karlsruhe Institute of Technology (KIT), Kaiserstraße 12, D-76131 Karlsruhe, Germany,
✉ fabian.ball@kit.edu

Andreas Geyer-Schulz
Karlsruhe Institute of Technology, (KIT), Kaiserstraße 12, D-76131 Karlsruhe, Germany,
✉ andreas.geyer-schulz@kit.edu

Archives of Data Science, Series A
(Online First)
KIT Scientific Publishing
Vol. 4, No. 1, 2018

DOI 10.5445/KSP/1000085951/01
ISSN 2363-9881

1 Introduction

The symmetry in graphs is captured by its automorphism group and it is a theoretically well understood and analyzed problem (e. g. Lauri and Scapellato, 2016). Graph symmetry can informally be seen analogously as in geometry: When drawing a graph on a plane (e. g. practically a sheet of paper), there exist several ways of drawing the same graph differently, but this is unnoticeable unless some additional highlighting (e. g. colors) is used to make the differences explicit. Of course, a more formal definition will follow in Section 2.

Mostly theorists from different mathematical sciences have worked and still work (Babai, 2016a,b; Helfgott et al., 2017) on this topic. Their effort seems to have brought down the problem's time complexity from being exponentially in the worst case to being quasi-polynomial (i. e. $\exp(O(\ln n)^c)$, $c > 1$ with n the number of nodes). Besides these theoretical considerations, since the beginning of the 1980s more and more practically usable algorithms were presented, starting with `nauty` (McKay, 1981; McKay and Piperno, 2014), followed by `saucy` (Darga et al., 2004, 2008), `bliss` (Junttila and Kaski, 2007), and lastly `Traces` (Piperno, 2008; McKay and Piperno, 2014). All these algorithms are capable of combinatorially determining a graph's automorphism group. Especially `saucy` performs very well on real-world networks (Newman, 2003), which are generally very sparse (i. e. they contain only few edges in contrast to the maximum possible number). Sparsity is one of the main assumptions of the `saucy` algorithm.

However, causes of symmetry and its effects on the *analysis* of graphs (like clustering) don't seem to be a topic, yet. Ben-David et al. (2006) mention that symmetry causes instability in spectral clustering. This is clear as symmetry results in a possible reordering of the graph's adjacency matrix without altering the matrix itself. Spectral clustering uses information from eigenvalue analysis to separate the data into classes, which is of course independent of the ordering of rows and columns in the matrix. Other authors provide evidence for the existence of symmetry in real-world graphs (Darga et al., 2008; Xiao et al., 2008; MacArthur et al., 2008; Wang et al., 2009). Because these analyses are only on a very small scale (less than 30 graphs maximal) we were motivated to conduct our own empirical investigation of about 900 graphs. The result is that about 70% of all analyzed graphs contain symmetries (Ball and Geyer-Schulz, 2018), which shows the risk of not analyzing the effects of symmetry in graph-clustering.

These findings and the ongoing discussion on stability of clustering solutions inspired us to improve the connection of theory and practice by making the concept of graph partition stability operational by a formal definition of stability in the form of permutation invariance. First, we formally define graph symmetry and basic concepts from the theory of permutation groups (Section 2). This section is followed by the three equivalent definitions of stability (Section 3). Each of them (i) represents a different point of view on the problem and (ii) may be better practically applicable, depending on the setting of the analysis. We use those definitions to present an algorithm which tests partition stability (Section 4). Finally, we show by example that conflicts between partition quality measures and stability exist. The resolution requires the solution of multi-criteria optimization problems by clustering algorithms (Section 5). This article ends with a short summary and outlook in Section 6.

2 Preliminaries

In our setting we want to restrict ourselves to simple graphs, i. e. graphs $G = (V, E)$ with the node set $V = \{1, 2, \ldots, n\}$ ($n < \infty$) and the edge set $E \subseteq \{\{u, v\} \mid u, v \in V,\ u \neq v\}$ (edges $\{u, v\}$ are abbreviated uv). These graphs are undirected, finite, have no loops, no weights, no multiple edges, and are connected. All symmetry definitions that follow can easily be extended to capture non-simple graphs as well. A graph partition is a set $P = \{C_1, \ldots, C_k\}$ of non-overlapping subsets $C_i \subseteq V$ that is complete ($\bigcup_i C_i = V$) and all pairs of subsets are disjoint ($\forall i \neq j : C_i \cap C_j = \emptyset$). Empty subsets are prohibited ($\forall i : C_i \neq \emptyset$).

Symmetry in graphs is described by permutations $\pi : V \to V$, which are bijective maps of nodes onto nodes. $u^\pi : u \mapsto u^\pi$ is the image of π applied on u. A permutation π is an automorphism of G iff $G^\pi = G$, i. e. $(V^\pi, E^\pi) = (V, E)$ with $E^\pi = \{u^\pi v^\pi \mid uv \in E\}$. $V^\pi = V$ always holds by definition of π. We additionally define $C^\pi := \{u^\pi \mid u \in C\}$ and $P^\pi := \left\{C_i^\pi \mid C_i \in P\right\}$, which is the application of a permutation on a set C and on a partition P, respectively. The complete symmetry information of a graph G is captured by the automorphism group $Aut(G)$ containing all permutations $V \to V$ that are automorphisms of G. This means the automorphism group of a graph is a permutation group with

the composition (catenation) of permutations as group operation and the usual group properties:

- Identity: $\mathbf{1} \in Aut(G) : \mathbf{1}\pi = \pi\mathbf{1} = \pi \quad \forall \pi \in Aut(G)$
- Inverses: $\pi \in Aut(G) \iff \pi^{-1} \in Aut(G)$ with $\pi\pi^{-1} = \pi^{-1}\pi = \mathbf{1}$
- Closure: $\forall \pi, \tau \in Aut(G) : \pi\tau \in Aut(G)$
- Associativity: $\forall \pi, \tau, \rho \in Aut(G) : (\pi\tau)\rho = \pi(\tau\rho)$

For two permutations $\pi, \tau \in Aut(G)$ their composition $\pi\tau$ is defined as the successive application of the mappings. We define the order of the application of mappings from left to right as it is the natural way of reading texts: $u^{\pi\tau} = (u^{\pi})^{\tau}$. The special permutation $\mathbf{1}$ is called identity as it fixes every node ($\forall u \in V : u^{\mathbf{1}} = u$). The orbit of a node is defined as $u^{Aut(G)} = \{u^{\pi} \mid \pi \in Aut(G)\}$ and is just the set of all nodes which can be mapped onto each other by $Aut(G)$. All orbits of $Aut(G)$ together form the orbit partition $O = \left\{u^{Aut(G)} \mid u \in V\right\}$.

3 Three Equivalent Partition Stability Definitions

Clustering partition stability is not a very tangible term and there are several definitions (probably not exhaustive) that could be thought of. Given a partition P . . .

. . . as the result of an algorithm (e. g. clustering), the algorithm's result on the *same* input should always be the same. I. e. the used method shall be stable/deterministic.

. . . as the result of an algorithm (e. g. clustering), the algorithm's result on *slightly perturbed* input should not differ significantly (Ben-Hur et al., 2001; Tibshirani and Walther, 2005; von Luxburg, 2010).

. . . being optimal concerning some graph partition (quality) criterion (e. g. Newman and Girvan, 2004), the addition and/or removal of a few edges should change the criterion's value only marginally so that the optimal partition stays the same (e. g. Karrer et al., 2008).

This implies stability of the graph's topology concerning the given criterion against small changes.

... the class memberships should be unambiguous (Gfeller et al., 2005), which means there should exist a unique clustering partition and no nodes that cannot be assigned to exactly one cluster.

... (independent from its origin), no visible changes should occur due to symmetries of the graph (captured by its automorphism group).

The last point is the idea of our approach. It is somewhat similar to the idea of Gfeller et al. (2005) who identify nodes that cannot be uniquely assigned to exactly one cluster of the partition. However, such ambiguities need not be caused by automorphisms of the graph. Nonetheless, our stability definition together with the one of Gfeller et al. (2005) – and probably even more – can be combined as we argue in Section 5.

3.1 Splitting the Automorphism Group

The automorphism group $Aut(G)$ is a finite set of permutation functions. In the first approach this set is split into two subsets, one of which may be empty.

Definition 1 Let P be a partition of G. $Aut(G)$ is split into two subsets Π_{intra} and Π_{inter} with Π_{intra} containing all permutations for which $P^\pi = P$ and Π_{inter} containing all permutations for which $P^\pi \neq P$. The partition P is stable iff $\Pi_{inter} = \emptyset$.

Definition 1 implies that for a stable partition P every automorphism of G must only act "locally" by mapping nodes onto each other that are within the same cluster anyway or "globally" by mapping entire clusters onto each other. Unfortunately, the automorphism group can easily become very large so that testing stability for each of the permutations is very expensive to perform. However, generating the whole group is not necessary!

Therefore we introduce another representation of the automorphism group in terms of a set S of so called generators. $Aut(G)$ is said to be generated by $S \subset Aut(G)$ if systematic composition of permutations in S yields $Aut(G)$. We

define this generation as $\langle S \rangle = Aut(G)$. Normally, $|S| \ll |Aut(G)|$, which can be seen in Table 1. Theorem 1 formalizes the idea to split S instead of $Aut(G)$.

Table 1: Comparison of the number of generators $|S|$ and the size of the completely enumerated group $|Aut(G)|$. Information and references of the networks can be found in (Ovelgönne et al., 2010, Table 1). The results were obtained with `saucy`. The computation time for the LiveJournal graph took about one hour on an eight core *Intel(R) Xeon(R) E5-2640 v3 2.60GHz* CPU with over 60GB of memory. (*) Network is directed; (**) Network is weighted; (***) Network is disconnected; For directed/weighted networks we omitted the directions/weights. `saucy` is capable of analyzing disconnected networks, so no transformation (like extracting the largest connected component) was done.

Network	n	m	$\lvert S\rvert$	$\lvert Aut(G)\rvert$	
Karate	34	78	6	4.8000×10^{2}	
Jazz	198	2742	7	1.2800×10^{2}	
Email	1133	5451	27	1.5288×10^{9}	
PGP	10 680	24 316	2586	4.4963×10^{1251}	
Cond. Mat.	31 163	120 029	8327	1.0760×10^{6175}	**/***
WWW	325 729	1 090 108	189 865	$3.9818 \times 10^{246\,936}$	*
Amazon	410 236	2 439 437	8907	8.0969×10^{3295}	*
LiveJournal	4 847 571	68 993 773	416 660	$7.1830 \times 10^{207\,553}$	*/***

Theorem 1 Given a partition P of G and a set of generators S ($\langle S \rangle = Aut(G)$), which is split into $\tilde{\Pi}_{intra} = \{\pi \in S \mid P^{\pi} = P\}$ and $\tilde{\Pi}_{inter} = \{\pi \in S \mid P^{\pi} \neq P\}$. P is stable iff $\tilde{\Pi}_{inter} = \emptyset$.

Proof. Let G be a graph and S be a set of generators, i. e. $\langle S \rangle = Aut(G)$. Furthermore, let P be a partition of G, and let Π_{intra} and $\Pi_{inter} \neq \emptyset$ (i. e. P is unstable) be two subsets of $Aut(G)$ for a given partition P as proposed in Definition 1. Suppose $\tilde{\Pi}_{intra} = S$ and $\tilde{\Pi}_{inter} = \emptyset$. Then for some $\pi \in \Pi_{inter}$ (obviously $\pi \notin \tilde{\Pi}_{intra}$), there must exist a sequence $(\tau_1, \ldots, \tau_k)$ for which $\tau_1 \cdots \tau_k = \pi$ holds and $\tau_i \in \tilde{\Pi}_{intra}$, $i = 1, \ldots, k$. But this is a contradiction as each τ_i individually either fixes all nodes within all $C_i \in P$ or maps all nodes from one cluster to another cluster. For arbitrary compositions of multiple τ_i in the sequence above this property is retained as composing permutations means successively executing the mapping of each permutation in the given order. Associativity assures that it does not matter which permutation is used first as long as the order does not change. So for $\tau_1 \cdots \tau_k = \pi \in \Pi_{inter}$, at least one $\tau_i \in \tilde{\Pi}_{inter}$ must exist, which contradicts $\tilde{\Pi}_{inter} = \emptyset$. □

Example 1 Let G be a graph that consists of the nodes $V = \{1, \ldots, 8\}$ and its automorphism group is generated by $S = \{(1\,2), (7\,8)\}$ so that $\langle S \rangle = \{\mathbf{1}, (1\,2), (7\,8), (1\,2)(7\,8)\} = Aut(G)$. The partition $P = \{\{1, 2, 3\}, \{4, 5, 6\}, \{7, 8\}\}$ is stable because $P^{(1\,2)} = P$ as well as $P^{(7\,8)} = P$. A second partition $Q = \{\{1, 3\}, \{2\}, \{4, 5, 6\}, \{7, 8\}\}$ is of course unstable as $Q^{(1\,2)} = \{\{2, 3\}, \{1\}, \{4, 5, 6\}, \{7, 8\}\} \neq Q$. $S = \tilde{\Pi}_{intra} \cup \tilde{\Pi}_{inter} = \{(7\,8)\} \cup \{(1\,2)\}$ or $Aut(G) = \Pi_{intra} \cup \Pi_{inter} = \{\mathbf{1}, (7\,8)\} \cup \{(1\,2), (1\,2)(7\,8)\}$, respectively.

3.2 Partition of Blocks

For a given permutation group $Aut(G)$ a block is defined as a subset $C \subseteq V$ for which either $C \cap C^\pi = C$ or $C \cap C^\pi = \emptyset$ holds for all $\pi \in Aut(G)$. Informally, the permutations either map the block onto itself or all nodes to a different subset $C' \subseteq V$, which of course must also be a block. More details can be found in Wielandt (1964).

Definition 2 A partition P is stable

1. if every $C_i \in P$ is a block of $Aut(G)$ and
2. if every $C_i^\pi \in P$ for which $C_i \cap C_i^\pi = \emptyset$ holds.

Definition 2 is strongly coupled with the first approach in Section 3.1. It explicitly uses a general property of permutation groups (1.) but additionally requires (2.) to be true. The second condition is important as the example will show.

Example 2 Let again G be a graph that consists of the nodes $V = \{1, \ldots, 8\}$ and $S = \{(1\,4)(2\,3), (1\,5)(2\,6)(3\,7)(4\,8)\}$, which generates the permutation group

$$\begin{aligned} Aut(G) = \{&\mathbf{1}, (1\,4)(2\,3), (1\,5)(2\,6)(3\,7)(4\,8), (5\,8)(6\,7), (1\,8\,4\,5)(2\,7\,3\,6), \\ &(1\,4)(2\,3)(5\,8)(6\,7), (1\,5\,4\,8)(2\,6\,3\,7), (1\,8)(2\,7)(3\,6)(4\,5)\}. \end{aligned}$$

The partition $P = \{\{1, 2\}, \{3, 4\}, \{5, 6\}, \{7, 8\}\}$ is stable because both conditions hold. Each $\pi \in Aut(G)$ either fixes C_i or maps it onto another $C_j = C_i^\pi \in P$. Let $Q = \{\{1, 2, 3, 4\}, \{5, 6\}, \{7, 8\}\}$. E. g. $\{5, 6\}$ is a block of $Aut(G)$ but $\{5, 6\}^{(1\,5)(2\,6)(3\,7)(4\,8)} = \{1, 2\} \notin Q$. Therefore, Q is unstable because condition 2 of Definition 2 is not fulfilled and we see that condition 1 alone is not sufficient.

Finally, we give a counterexample that it is not possible to simply replace "$Aut(G)$" by "S" in Definition 2 similarly to Definition 1. When analyzing $R = \{\{1,2,3,4\},\{5,6,7\},\{8\}\}$ one can see that it is not sufficient to only consider all $\pi \in S$. $\{5,6,7\}^{(1\,4)(2\,3)} = \{5,6,7\}$ fixes the cluster and $\{5,6,7\}^{(1\,5)(2\,6)(3\,7)(4\,8)} \cap \{5,6,7\} = \{1,2,3\} \cap \{5,6,7\} = \emptyset$. But $\{5,6,7\}$ is not a block as e. g. $\{5,6,7\}^{(5\,8)(6\,7)} \cap \{5,6,7\} = \{8,7,6\} \cap \{5,6,7\} = \{6,7\} \notin \{\emptyset, \{5,6,7\}\}$ shows.

The second approach of defining the stability of a partition based on the graph's automorphism group shows the connection between a pure permutation group property and its application to data analysis. Compared to Definition 1 this comes at the cost of the necessity to completely enumerate the automorphism group. However, MacArthur et al. (2008) provide a decomposition method of the group based on its generators. This decomposition could be applied when this approach is implemented.

3.3 Partition Refinement Lattice

The last approach involves the refinement lattice of all possible partitions of the node set V. The set of all partitions is denoted by $\mathbf{P}(V)$. Its size increases rapidly with the number of nodes n and is captured by the Bell number $B(n)$ (Sloane, 2017a). One way to calculate $B(n)$ is summing over the Stirling numbers of the second kind $S(n,k) = \frac{1}{k!}\sum_{i=0}^{k}(-1)^{k-i}\binom{k}{i}i^n$ (Sloane, 2017b). $S(n,k)$ counts the number of possible partitions of a set of size n into k subsets.

$\mathbf{P}(V)$ can be arranged as lattice given the partial ordering $\leq$. $P \leq Q$ is defined as $\forall C \in P\ \exists D \in Q : C \subseteq D$. The lattice can be drawn in a hierarchical manner where each level $0 \leq l \leq n-1$ contains the $S(n,k)$ partitions that have cardinality $k = n - l$ (i. e. k subsets). If $P \leq Q$, P is equal or finer than Q and Q is equal or coarser than P. Any two partitions $P \leq Q$ are neighbors in the lattice if $Q = (P \setminus \{C', C''\}) \cup \{C' \cup C''\}$ for $C', C'' \in P$ and $C' \neq C''$. The lattice is itself a graph having the partitions as labeled nodes and edges for neighboring partitions. This understanding of the refinement lattice also illustrates the search space of a hierarchical clustering method. Each path through the lattice from level 0 to level $n-1$ corresponds to a dendrogram.

Definition 3 $P \in \mathbf{P}(V)$ is stable iff $\forall \pi \in Aut(G) : P^\pi \leq P$.

The point of view of Definition 3 shows that there must always exist some finest stable partition for a given automorphism group (see also Ball and Geyer-Schulz, 2017). Naturally, this is the partition of all orbits O (see Section 2).

Theorem 2 Given $P \in \mathbf{P}(V)$ and an orbit partition O concerning the automorphism group $Aut(G)$, P is stable if $P \geq O$.

Proof. Clearly, O is stable by definition as each subset/cluster contains exactly those nodes that can be mapped onto each other. This property is retained for any $C_\cup = C' \cup C'',\ C', C'' \in O$. The argument holds recursively for joins. Thus any partition P that can be created by successively joining clusters of O must be stable, too. However, this is exactly the refinement definition $P \geq O$. □

It is worth noting that the conclusion of Theorem 2 is not that *any* stable partition must be coarser than O! There can be numerous other stable partitions that are not coarser or even finer than the finest stable partition. One trivial example is the partition of singletons (each node forms one cluster), which is always stable.

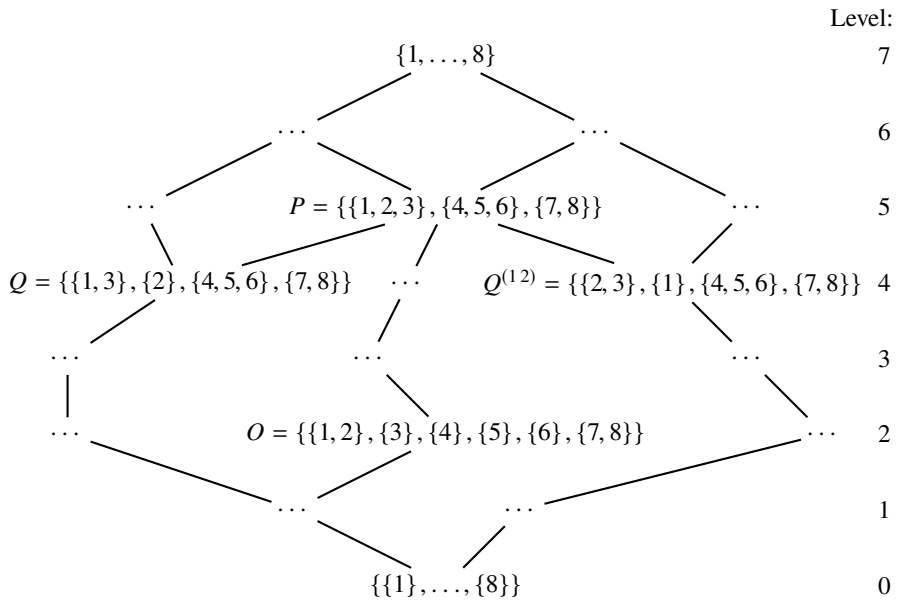


Figure 1: Extract of the partition refinement lattice for $V = \{1, \ldots, 8\}$. Partitions become coarser on higher levels. O is the orbit partition, which is the finest stable partition of $Aut(G) = \{\mathbf{1}, (1\,2), (7\,8), (1\,2)(7\,8)\}$.

Example 3 Let us again look at $Aut(G) = \{\mathbf{1}, (1\,2), (7\,8), (1\,2)(7\,8)\}$ and $P = \{\{1,2,3\},\{4,5,6\},\{7,8\}\}$. It is coarser than the finest stable partition O (see Figure 1) and is of course coarser than every $P^\pi,\ \pi \in Aut(G)$. The partition $Q = \{\{1,3\},\{2\},\{4,5,6\},\{7,8\}\}$ is unstable as e. g. $Q^{(1\,2)} \not\leq Q$. In Figure 1 can be seen that Q and $Q^{(1\,2)}$ are of course on the same level in the lattice but none of them is coarser than $O = \{\{1,2\},\{3\},\{4\},\{5\},\{6\},\{7,8\}\}$.

3.4 Equivalence of the Definitions

The equivalence of Definitions 1–3 is quite obvious, however, we formally prove it in this section.

Theorem 3 Definitions 1 and 2 are equivalent.

Proof. Given a partition P of G and the corresponding automorphism group $Aut(G)$, which is divided into Π_{intra} and Π_{inter}.

1. If $\Pi_{inter} = \emptyset$ then $P = P^\pi,\ \pi \in Aut(G)$ holds. This can only be true if $\forall C \in P,\ \forall \pi \in Aut(G) : C^\pi \in P$. $C^\pi \in P$ for some π and C means either $C^\pi = C$ or $C^\pi = C' \neq C$ with $C \cap C' = \emptyset$ and $C' \in P$. Therefore C is a block of $Aut(G)$ and $C^\pi \in P,\ \forall \pi \in Aut(G)$.
2. If conversely all $C \in P$ are blocks of $Aut(G)$ and $\forall C \in P,\ \forall \pi \in Aut(G) : C^\pi \in P$. This directly implies $P^\pi = P,\ \forall \pi \in Aut(G)$ and therefore $\Pi_{inter} = \emptyset$.

□

Theorem 4 Definitions 1 and 3 are equivalent.

Proof. Again, given a partition P of G and the corresponding automorphism group $Aut(G)$, which is divided into Π_{intra} and Π_{inter}.

1. From $P = P^\pi,\ \pi \in Aut(G)$ directly follows $P \geq P^\pi,\ \pi \in Aut(G)$.
2. Given $P \geq P^\pi$ for all $\pi \in Aut(G)$. By definition, for each $C' \in P^\pi$ must exist $C \in P$ so that $C' \subseteq C$. As permutations π are bijective, $|P| = |P^\pi|$.

By definition of the refinement lattice, there cannot be any neighbors on the same level. Therefore the only possibility for $P \geq P^\pi$ to hold is $P = P^\pi$.

□

Lemma **Definitions 2 and 3 are equivalent.**

Proof. Follows from the transitivity of the equivalence relation together with the equivalences of Definitions 1 and 2, and Definitions 1 and 3. □

Until now we did not use any special properties of graphs (besides how $Aut(G)$ is defined) but only required partitions of a finite set of length n and a permutation group that acts on this set and hence also on the partitions. This is because our definitions are completely generalizable for *any* partition of objects, provided the symmetry group is known. In Section 4 we exploit Definitions 1 and 3 for an implementation of a fast stability testing algorithm.

4 Fast Partition Stability Analysis

After the equivalence of the three provided characterizations of stability based on symmetry is clear, the question about which one is the best arises. Before arguing about practical uses of the definitions it is important to mention that all our considerations only become relevant if the data contains symmetry at all. For a trivial automorphism group (i. e. $|Aut(G)| = |\{\mathbf{1}\}| = 1$) every partition of nodes is stable.

From an analytic point of view, Definition 2 shows the connection to a permutation group property (blocks). The practical applicability of it is limited because it requires to actually enumerate the whole group to check if clusters of a partition are blocks. However, from a didactic point of view it shows that the transfer of results from the theory of permutation groups to data science often requires some modifications.

Definition 1 is of greater practical use. An algorithm that computes the automorphism group of a graph (like `nauty`[1] or `saucy`[2]) normally computes a set of generators of the group and returns them to the caller. Theorem 1 shows that the generators suffice to test stability by iterating over them and checking $P^\pi \geq P$.

Looking at Definition 3 also gives some insights what happens if symmetries exist, especially concerning the search space. The stability test procedure is similar to the one of Definition 1 as the proof of Theorem 4 shows the equality of $P = P^\pi$ and $P \geq P^\pi$.

The additional idea of having a finest stable partition leads to Algorithm 1. The orbit partition O can be computed from the generators and the test involves only one comparison of partitions (instead of $|S|$). The computation of the orbit partition O in Algorithm 1 should be memoized to give an advantage for repeated stability tests for different partitions. Memoization is a technique to cache the result of a function given the input so that no recomputation is necessary if it is called again with the same input (Geyer-Schulz, 1989). The actual implementation of this computation and the partition comparison test ($\geq$) strongly depend on the used data structures, which are used to represent partitions and permutations. `nauty` and `saucy` represent permutations using an explicit form where each permutation π is an array p of length n with $p[i] = i^\pi$ ($i \in V = \{0, \ldots, n-1\}$ to allow array indexing starting from 0, which normally is the standard way in programming languages). Partitions are represented as arrays of length n where the entry at position i is some arbitrary *cluster id*. All nodes in the same cluster have the same *cluster id*. Another representation, which relates to the mathematical notation of a set of sets, is storing a partition as array of arrays.

[1] `http://pallini.di.uniroma1.it/`, accessed August 27, 2018

[2] `http://vlsicad.eecs.umich.edu/BK/SAUCY/`; `https://github.com/KIT-IISM-EM/pysaucy` (Python binding for `saucy`), both accessed August 27, 2018

Algorithm 1 Testing partition stability based on the automorphism group of the graph

Require: S is a set of generators for $Aut(G)$, i. e. $\langle S \rangle = Aut(G)$, P is a partition in array representation

```
1:  function testStability(P, S)
2:      if S = ∅ then                          ▷ The group is trivial, the graph is asymmetric
3:          return True
4:      end if
5:      O ← computeOrbitPartition(S, |P|)                    ▷ Pseudocode in Algorithm 2
6:      if P ≥ O then                                        ▷ Pseudocode in Algorithm 4
7:          return True                ▷ The partition is coarser than the orbit partition
8:      end if
9:      for π ∈ S do
10:         if not P ≥ P^π then                              ▷ Pseudocode in Algorithm 4
11:             return False      ▷ Instability detected; P ≥ P^π is equivalent to P = P^π
12:         end if
13:     end for
14:     return True
15: end function
```

We present efficient algorithms to compute the orbit partition from the set of generators S (Algorithms 2 and 3) and to check if one partition is coarser than another partition (Algorithm 4) in the appendix. The complexity of Algorithm 1 mostly depends on the number of generators $|S|$. Finding upper bounds or the exact value for the size of a minimal generating set is an old but still open research problem except for special classes of groups (Lucchini et al., 2004). However, Table 1 shows that the algorithmic output of `saucy` is very promising in terms of the actual number of generators returned.

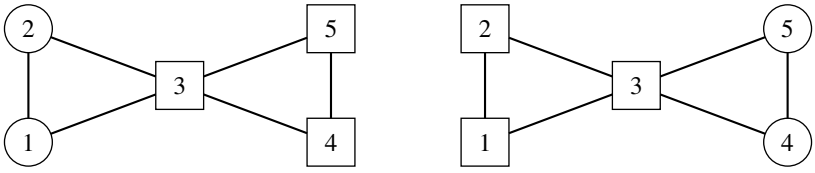
5 Goal Conflicts in Graph Clustering

Our stability definitions need not be seen as exclusive property a partition can have. Instead it should be considered an additional analytic tool, which can be used when determining the quality of a partition. Therefore, it is possible to combine our definitions with other partition quality measures like e. g. the modularity measure Q for partition quality (Newman and Girvan, 2004), which

quantifies the number of intra-cluster connections in contrast to the inter-cluster connections. Also the ideas of Gfeller et al. (2005) to identify nodes that cannot be assigned to exactly one cluster could be taken into account.

However, optimizing several partition quality measures leads to a multi-criteria optimization problem with the typical conflicts as the following example shows. The topic of actually integrating multiple objectives into graph clustering algorithms is left for further research.

Example 4 For the given graph in Figure 2, a modularity maximal partition is $P = \{\{1,2\},\{3,4,5\}\}$ with modularity $Q = \frac{1}{9}$. Unfortunately e. g. $\pi = (1\,4)(2\,5)$ shows that $P^\pi = \{\{1,2,3\},\{4,5\}\} \neq P$. There is a goal conflict between maximizing modularity and having a stable partition concerning graph symmetry as P^π has the same maximal modularity value as P. This means that the modularity optimal partition is not unique, which is a consequence of an instability caused by symmetry.



(a) $P = \{\{1,2\},\{3,4,5\}\}$. (b) $P^\pi = \{\{1,2,3\},\{4,5\}\}$.

Figure 2: A small graph which illustrates the problem of conflicting goals. The two clusters are distinguished using different node shapes. $\pi = (1\,4)(2\,5)$ "mirrors" the graph and therefore maps partition P (a) onto partition P^π (b) indicating instability.

We want to point out that the stable partitions $\{\{1,2\},\{3\},\{4,5\}\}$ and $\{\{1,2,3,4,5\}\}$ both have modularity zero.

6 Conclusion

We motivate the topic on stability of clustering partitions in our first section. Symmetries in "real-world" graphs frequently exist but it seems that there is no or only small awareness of possible side-effects on the results of data analysis methods – graph clustering in our case. After a short introduction of the needed

mathematical concepts, three different but equivalent methods for recognizing partition stability based on the automorphism group of the analyzed graph are presented. Each of them provides a different viewpoint on the topic and gives insights into the problem's structure. Equivalence of the three definitions is proved thereafter and we sum up the main part of this work by comparing them and giving advice on how to test stability of a partition.

The automorphism group of the analyzed graph has of course to be known, but we showed that it is in general sufficient to have a set of generators from which the partition of orbits can be computed. Algorithms that enumerate $Aut(G)$ normally provide a set of generators as the method's output. Especially `saucy` is worth to look at from a practical point of view as it performs very well on large and sparse graphs – which real-world networks normally are. Although the graph automorphism problem is considered relatively hard, existing algorithms perform well. Algorithm 1 for testing partition stability makes it possible to use our definitions in practice.

Exact symmetry is a very tight restriction and weaker symmetry definitions – based on stochastic properties – do also exist (e. g. Garlaschelli et al., 2010). However, our definitions are still applicable when the exact $Aut(G)$ is replaced by some other permutation group H that represents symmetry in a weaker sense (most likely $Aut(G) \leq H$). The overall stability would of course decrease when there are more possibilities to map nodes onto nodes by a replacement of $Aut(G)$. A quantification of instability in terms of the Kolmogorov-Sinai Entropy is presented by Ball and Geyer-Schulz (2017).

Appendix

Algorithms 2 and 3 show an efficient implementation of the computation of the orbit partition from a set of generators. The idea is to color all nodes that are on the same orbit by successively iterating over the cycles of each generator starting from a given node. Nodes which are on the same orbit are used as starting points in the next iteration. This procedure is similar to a mixture of breadth- and depth-first search. Coloring simply means to set a *cluster id* for each node, all nodes on the same orbit have the same color.

When a cycle of a permutation π is explored, for each node i on that cycle the tuple (i, π) is added to the set U to prevent that the same cycle is repeatedly

explored. This allows us to give an upper bound for the time complexity of Algorithm 2, which repeatedly calls Algorithm 3. The latter is a procedure that modifies O (the orbit partition), N (candidates for search), and U (memory of already explored cycles). Algorithm 2 relies on these modifications.

1. The outer loop is in $\mathcal{O}(n)$ (Algorithm 2, lines 5–13)
2. Initialization of O, $colored$, U, ... are all in $\mathcal{O}(1)$ or $\mathcal{O}(n)$ (Algorithm 2, lines 2–4, 7)
    - If a node is already colored, no work is performed (Algorithm 2, line 6)
    - If a node is uncolored or was just colored (i. e. is in N), Algorithm 3 is called (Algorithm 2, lines 8 and 12)
3. Algorithm 3 iterates over all generators, is thus in $\mathcal{O}(|S|)$ (Algorithm 3, lines 7–19)
    - If a cycle of a permutation was already explored, it is skipped (set insertion/lookup is normally in $\mathcal{O}(1)$) (Algorithm 3, line 8)
    - Otherwise, the cycle is explored, i. e. each uncolored node is colored and added to the set N of new starting points. (Algorithm 3, lines 9–19)
4. A permutation can consist of at most n cycles and each is guaranteed to be explored only once, independent of the starting node i

Combining all the facts results in an overall worst case complexity of $\mathcal{O}(|S| \cdot n)$ as over time each cycle is explored exactly once (at most, early exit is not considered).

Algorithm 2 Compute the orbit partition O from a set of generators

Require: S is a set of generators for $Aut(G)$, i. e. $\langle S \rangle = Aut(G)$, n is the number of nodes of G

1: **function** COMPUTEORBITPARTITION(S, n)
2: $O \leftarrow [\perp, \ldots, \perp]$ ▷ Initialize "empty" orbit partition of length n
3: $colored \leftarrow 0$
4: $U \leftarrow \emptyset$ ▷ Set to save which cycles were already explored
5: **for** $i \leftarrow 0, \ldots, n-1$ **do**
6: **if** $O[i] \neq \perp$ **then continue** ▷ Skip already colored nodes
7: **end if**
8: $N \leftarrow \emptyset$ ▷ Set of start nodes for the search
9: $colored \leftarrow colored +$ COLOR(O, i, S, i, N, U) ▷ Color node i and all reachable nodes
10: **if** $colored = n$ **then return** O ▷ Early exit if all nodes are already colored
11: **end if**
12: **while** $N \neq \emptyset$ **do** ▷ N contains all nodes that were reached by the last call of COLOR
13: $j \leftarrow$ POP(N) ▷ POP(N) removes and returns an arbitrary element from N
14: $colored \leftarrow colored +$ COLOR(O, j, S, i, N, U)
15: **if** $colored = n$ **then return** O
16: **end if**
17: **end while**
18: **end for**
19: **return** O
20: **end function**

Algorithm 3 Color nodes which are reachable from i with color col

Require: O is the orbit partition, S is a set of generators, i is the node to use as starting point for the coloring, col is the color to use, N is a set to save nodes for a later use as start point, U is a set of tuples (i, π) to save and look up already searched cycles

```
1:  procedure COLOR(O, i, S, col, N, U)
2:      if O[i] = ⊥ then
3:          O[i] ← col
4:          colored ← 1
5:      else
6:          colored ← 0
7:      end if
8:      for π ∈ S do
9:          if (i, π) ∈ U then continue          ▷ Do not search a cycle more than once
10:         end if
11:         U ← U ∪ {(i, π)}
12:         j ← π[i]
13:         while j ≠ i do                       ▷ Color all nodes on the cycle
14:             if O[j] = ⊥ then
15:                 O[j] ← col
16:                 N ← N ∪ {j}
17:                 colored ← colored + 1
18:                 U ← U ∪ {(j, π)}
19:             else
20:                 assert O[j] = col ▷ Must hold as we successively color all nodes on an
    orbit
21:             end if
22:             j ← π[j]
23:         end while
24:     end for
25:     return colored
26: end procedure
```

Algorithm 4 Test $P \geq Q$ for two partitions P and Q

Require: Two partitions P and Q of the same length in array representation

```
1: function GEQ(P, Q)
2:     maps ← HASHMAP( )                ▷ A hash map to save mappings of cluster ids
3:     n ← |P|
4:     for i ← 0, ..., n − 1 do
5:         if Q[i] ∈ maps then          ▷ Tests if there exists a value for the given key
6:             oid ← maps[Q[i]]         ▷ Get the already saved mapping
7:         else
8:             oid ← P[i]
9:             maps[Q[i]] ← oid
10:        end if
11:        if P[i] ≠ oid then return False
12:        end if
13:    end for
14:    return True
15: end function
```

Algorithm 4 is an efficient implementation to check if a partition P is coarser than or equal partition Q as defined in section 3.3. As the *cluster ids* are arbitrary, a hash map is used to keep track of the mappings of *cluster ids*. If the *cluster ids* also come from the domain $\{0, \ldots, n-1\}$, an array can be used instead of a hash map. The time complexity of the comparison is $O(n)$ as inserting and reading from a hash map is $O(1)$. We provide implementations of all four algorithms online under `https://github.com/KIT-IISM-EM/partitionstability` as Python and R code.

References

Babai L. (2016a): Graph isomorphism in quasipolynomial time. arXiv:1512.03547v2 [cs, math], URL: `https://arxiv.org/abs/1512.03547v2`

Babai L. (2016b): Graph isomorphism in quasipolynomial time [Extended abstract]. In: Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing, ACM, New York, NY, USA, STOC '16, pp. 684–697, DOI: 10.1145/2897518.2897542

Ball F., Geyer-Schulz A. (2017): Weak invariants of actions of the automorphism group of a graph. Archives of Data Science Series A, 2(1):1–22, DOI: 10.5445/KSP/1000058749/02

Ball F., Geyer-Schulz A. (2018): How symmetric are real-world graphs? A large-scale study. Symmetry, 10(1):17, DOI: 10.3390/sym10010029

Ben-David S., von Luxburg U., Pál D. (2006): A sober look at clustering stability. In: Learning Theory, Springer, Berlin, Heidelberg, pp. 5–19, DOI: 10.1007/11776420_4

Ben-Hur A., Elisseeff A., Guyon I. (2001): A stability based method for discovering structure in clustered data. In: Biocomputing 2002, Altman R. B., Dunker A. K., Hunter L., Lauderdale K., Klein T. E. (eds.), World Scientific, pp. 6–17, DOI: 10.1142/9789812799623_0002

Darga P. T., Liffiton M. H., Sakallah K. A., Markov I. L. (2004): Exploiting structure in symmetry detection for cnf. In: Proceedings of the 41st Annual Design Automation Conference, ACM, New York, NY, USA, DAC '04, pp. 530–534, DOI: 10.1145/996566.996712, URL: http://doi.acm.org/10.1145/996566.996712

Darga P. T., Sakallah K. A., Markov I. L. (2008): Faster symmetry discovery using sparsity of symmetries. In: Proceedings of the 45th Annual Design Automation Conference, ACM, New York, NY, USA, DAC '08, pp. 149–154, DOI: 10.1145/1391469.1391509, URL: http://doi.acm.org/10.1145/1391469.1391509

Garlaschelli D., Ruzzenenti F., Basosi R. (2010): Complex networks and symmetry I: A review. Symmetry, 2(3):1683–1709, DOI: 10.3390/sym2031683

Geyer-Schulz A. (1989): Memo. APL Quote Quad, 20(2):12–27, DOI: 10.1145/379209.379211

Gfeller D., Chappelier J.-C., De Los Rios P. (2005): Finding instabilities in the community structure of complex networks. Physical Review E, 72(5):056,135, DOI: 10.1103/PhysRevE.72.056135

Helfgott H. A., Bajpai J., Dona D. (2017): Graph isomorphisms in quasi-polynomial time. arXiv:1710.04574 [math], URL: http://arxiv.org/abs/1710.04574, 1710.04574

Hennig C. (2015): What are the true clusters? Pattern Recognition Letters, 64:53–62, DOI: 10.1016/j.patrec.2015.04.009

Junttila T., Kaski P. (2007): Engineering an efficient canonical labeling tool for large and sparse graphs. In: Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX), SIAM, pp. 135–149, DOI: 10.1137/1.9781611972870.13, URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611972870.13

Karrer B., Levina E., Newman M. E. J. (2008): Robustness of community structure in networks. Physical Review E, arXiv:0709.2108 [physics.data-an], 77(4):46,119, DOI: 10.1103/PhysRevE.77.046119, URL: https://arxiv.org/abs/0709.2108

Lauri J., Scapellato R. (2016): Topics in Graph Automorphisms and Reconstruction, 2nd edn. No. 432 in London Mathematical Society Lecture Notes Series, Cambridge University Press, Cambridge

Lucchini A., Menegazzo F., Morigi M. (2004): Generating permutation groups. Communications in Algebra, 32(5):1729–1746, DOI: 10.1081/AGB-120029899

von Luxburg U. (2010): Clustering stability: An overview. Foundations and Trends in Machine Learning, 2(3):235–274, DOI: 10.1561/2200000008

MacArthur B. D., Sánchez-García R. J., Anderson J. W. (2008): Symmetry in complex networks. Discrete Applied Mathematics, 156(18):3525–3531, DOI: 10.1016/j.dam.2008.04.008

McKay B. D. (1981): Practical graph isomorphism. Congressus Numerantium, 30:45–87, URL: http://users.cecs.anu.edu.au/~bdm/papers/pgi.pdf

McKay B. D., Piperno A. (2014): Practical graph isomorphism, II. Journal of Symbolic Computation, 60:94–112, DOI: 10.1016/j.jsc.2013.09.003

Newman M. E. J. (2003): The structure and function of complex networks. SIAM Review, 45(2):167–256, DOI: 10.1137/S003614450342480

Newman M. E. J., Girvan M. (2004): Finding and evaluating community structure in networks. Physical Review E, 69(2):026,113, DOI: 10.1103/PhysRevE.69.026113

Ovelgönne M., Geyer-Schulz A., Stein M. (2010): Randomized greedy modularity optimization for group detection in huge social networks. In: SNA-KDD'10: Proceedings of the 4th Workshop on Social Network Mining and Analysis, ACM, New York, NY, USA, pp. 1–9, URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.452.614&rep=rep1&type=pdf

Piperno A. (2008): Search space contraction in canonical labeling of graphs. arXiv:0804.4881 [cs], URL: http://arxiv.org/abs/0804.4881

Sloane N. J. A. (2017a): A000110 - Bell or exponential numbers: Number of ways to partition a set of n labeled elements. URL: https://oeis.org/A000110

Sloane N. J. A. (2017b): A008277 - Triangle of Stirling numbers of the second kind. URL: https://oeis.org/A008277

Tibshirani R., Walther G. (2005): Cluster validation by prediction strength. Journal of Computational and Graphical Statistics, 14(3):511–528, DOI: 10.1198/106186005X59243

Wang H., Yan G., Xiao Y. (2009): Symmetry in world trade network. Journal of Systems Science and Complexity, 22(2):280–290, DOI: 10.1007/s11424-009-9163-9

Wielandt H. (1964): Finite Permutation Groups. Academic Press, New York

Xiao Y., MacArthur B. D., Wang H., Xiong M., Wang W. (2008): Network quotients: Structural skeletons of complex systems. Physical Review E, 78(4):046,102, DOI: 10.1103/PhysRevE.78.046102