Chapter 2

Graph Representation

19:05 December 8, 2012 This is the cover page for Section 2.2: Graph Isomorphism

Section 2.1 Graph Isomorphism

Brendan D. McKay, Australian National University

INTRODUCTION

Isomorphism between graphs and related objects is a fundamental concept in graph theory and its applications to other parts of mathematics. The problem also occupies a central position in complexity theory as a proposed occupant of the region that must exist between the polynomial-time and NP-complete problems if P≠NP. Due to its many practical applications a considerable number of algorithms for graph isomorphism have been proposed.

2.1.1 Isomorphisms and Automorphisms

Informally, two graphs are isomorphic if they are the same except for the names of their vertices and edges. Formally, this relationship is defined by means of bijections between them.

Basic Terminology

DEFINITIONS

D1: Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be simple graphs. An ***isomorphism*** from G_1 to G_2 is a bijection $\phi : V_1 \to V_2$ such that $vw \in E_1$ if and only if $\phi(v)\phi(w) \in E_2$.

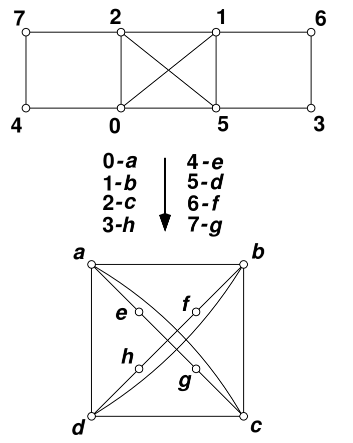
D2: A second way to define an isomorphism is that there are two bijections $\phi : V_1 \to V_2$ and $\phi' : E_1 \to E_2$, such that the incidence relation between vertices and edges is preserved. That is, $v \in V_1$ is incident to $e \in E_1$ if and only if $\phi(v)$ is incident to $\phi(e)$. This method is preferred if edges have additional attributes that should be preserved by the mapping. However, we will use the previous definition where it applies.

D3: An isomorphism from a graph to itself is called an ***automorphism*** or ***symmetry***.

D4: The set of automorphisms of a graph G form a group under the operation of composition, called the ***automorphism group*** $\mathrm{Aut}(G)$. The automorphism group of a simple graph is a subgroup of the symmetric group acting on the vertex set of the graph.

EXAMPLE

E1: Figure 2.1.1 shows an isomorphism between two graphs and gives the automorphism group of the first graph.



(1)
(0 2)(4 7)
(1 5)(3 6)
(0 2)(4 7)(1 5)(3 6)
(0 5)(1 2)(3 4)(6 7)
(0 1 2 5)(3 4 6 7)
(0 5 2 1)(3 7 6 4)
(0 1)(2 5)(3 7)(4 6)

Figure 2.1.1: An isomorphism between two graphs and the automorphism group of the first graph.

DEFINITION

D5: Closely related to isomorphism is the concept of canonical labeling. Arbitrarily choose one member of each isomorphism class of graphs, and call it the ***canonical form*** of that isomorphism class. Replacing a graph by the canonical form of its isomorphism class is called ***canonical labeling*** or ***canonizing the graph***. Two graphs are isomorphic if and only if their canonical forms are identical, as shown in Figure 2.1.2.

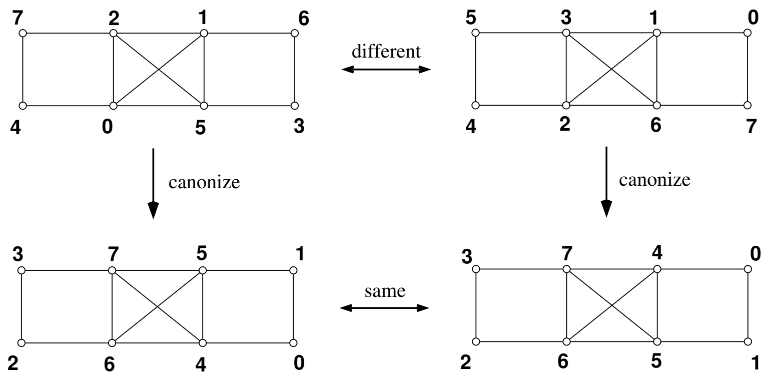


Figure 2.1.2: Isomorphism between graphs becomes equality when they are canonized.

REMARKS

R1: The labeled graph which gives the lexicographically greatest adjacency matrix is an example of an explicitly defined canonical form. In practice more complex definitions are used to assist efficient computation.

R2: Canonical labeling has central importance to practical applications. One task is to determine whether a graph is isomorphic to any graph in a database of graphs. This is best achieved by storing the canonical forms of graphs in the database and comparing them to the canonical form of the new graph. Another task is to remove isomorphs from a large collection of graphs. This is best achieved by applying a sorting algorithm to the canonical forms of the graphs. Both tasks are very expensive if only pair-wise isomorphism testing is available.

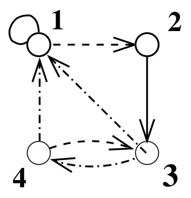
Related Isomorphism Problems

Many types of isomorphism problem can be modeled as isomorphism between simple graphs or digraphs.

FACTS

F1: *Vertex colors* that must be preserved by isomorphisms can be modeled by attaching gadgets to the vertices, a different gadget for each color. However, this is such an important generalization that most software can handle vertex colors directly.

F2: *Edge colors* can be modeled using layers, once vertex colors are available. Figure 2.1.3 illustrates one approach. The edge colors are assigned numbers according to the table in the center. The vertices of the original graph are assigned to vertical paths, with the first layer identified by vertex color. Then the original edges with each color c are represented by horizontal edges within those layers where the binary expansion of c has ones.



$$\begin{pmatrix} 3 & 1 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 2 & 0 & 0 & 2 \\ 2 & 0 & 1 & 0 \end{pmatrix}$$

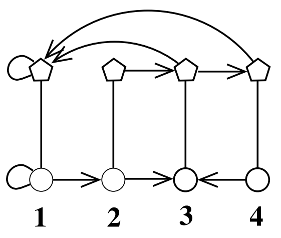


Figure 2.1.3: Modeling of graphs with colored edges.

F3: *Hypergraphs* and other types of incidence structures like *block designs* and *finite geometries* can be represented by bipartite graphs. One color class consists of the vertices of the hypergraph, while the other has a vertex for each of the hyperedges of the hypergraph. An edge of the bipartite graph represents a vertex of the hypergraph and a hyperedge it lies in.

F4: Other types of isomorphism easily modeled by graph isomorphism include equivalence of matrices defined by permutation of rows and columns, Hadamard equivalence, and isotopy (such as for Latin squares) [McPi12b].

2.1.2 Complexity Theory

The problem of determining whether two graphs are isomorphic, called ***GI*** or ***ISO***, has received a great deal of interest from theorists due to its unsolved nature.

FACTS

F5: GI is not known to have a polynomial-time algorithm, nor to be NP-complete. While obviously in NP, its presence in co-NP is also undecided. Indeed, it is considered a prime candidate for the intermediate territory between P and NPC that must exist if P≠NP. One reason for this is that the NP-completeness of GI would imply the collapse of the polynomial-time hierarchy [GoMiWi91].

F6: The fastest proven running time for GI has stood for three decades at $e^{O(\sqrt{n \log n})}$ [BaKaLu83].

F7: On the other hand, many special classes of graph are known to have polynomial-time isomorphism tests. The most general such classes are those defined by a forbidden minor [Po88, Gr10] or by a forbidden topological minor [Gr12]. These classes include many earlier classes, including graphs of bounded degree [Lu82], bounded genus [FiMa80, Mi80] and bounded tree-width [Bo90]. However, very few of these polynomial algorithms are practical.

DEFINITION

D6: A decision problem is called ***isomorphism-complete*** if it is polynomial-time equivalent to GI.

FACTS

F8: All of the isomorphism problems noted in the previous subsection are isomorphism-complete. Many other examples are known, including isomorphism of semigroups and finite automata [Bo78], homeomorphism of 2-complexes [STPi94] and polytope isomorphism [KaSc03].

F9: Isomorphism of linear codes (vector spaces over finite fields) defined by permutation of the coordinate positions, where the codes are presented as generator matrices, is at least as hard as graph isomorphism but might be harder [PeRo97].

F10: Isomorphism of groups given as multiplication tables is at least as easy as GI but might be easier [Bo78]. The best algorithm is very elementary and takes time $n^{O(\log n)}$ [Bo78, Mi78]).

F11: Some problems similar to GI are NP-complete. The best known is the subgraph isomorphism problem: given two graphs, is the first isomorphic to a subgraph of the second? Another is the presence of an automorphism without fixed points, or of such an automorphism of order 2 [Lu81].

DEFINITION

D7: A ***graph invariant*** is a property of graphs that is equal for isomorphic graphs. A ***complete graph invariant***, also called a ***certificate***, is an invariant that always distinguishes between non-isomorphic graphs.

FACTS

F12: Examples of invariants include the degree sequence and the eigenvalue set of the adjacency matrix. However, neither of those invariants is complete. Nevertheless, even incomplete invariants can sometimes be used as a short proof of non-isomorphism.

F13: An example of a complete invariant is a canonical form. However, it is not known if there is a complete invariant computable in polynomial time. In fact, it is not even known if there is a complete invariant checkable in polynomial time (which would place GI in co-NP).

2.1.3 Algorithms

The development of computer programs for graph isomorphism has been such a popular pursuit that already in 1976 it was called a "disease" [ReCo77]. Literally hundreds of algorithms have been published (many wrong).

FACTS

F14: The earliest software appeared in the 1960s. The approach which has been the most successful is the "individualization-refinement" paradigm introduced by Parris and Read [PaRe69] and further developed by Corneil and Gotlieb [CoGo70] and Arlazarov et al. [ArZuUsFa74]. This genre is now represented by the author's `nauty` and other software mentioned below.

REMARK

R3: We will focus our attention on canonical labeling, which is the method used by the most useful modern algorithms.

DEFINITIONS

D8: A key routine is that of ***partition refinement***, which is any process of making a partition finer (i.e., breaking its cells into smaller cells) by detecting combinatorial differences between the vertices. For isomorphism purposes, only properties independent of the numbering of the vertices may be used. This implies that vertices equivalent under the action of an automorphism fixing the input partition cannot be separated.

D9: An ***equitable partition*** is a partition of the vertices of a graph into cells such that, for any two vertices v, w in the same cell, and any cell C, we have that v and w are adjacent to the same number of vertices in C.

EXAMPLE

E2: Figure 2.1.4 shows a graph with an equitable partition of two cells. Each black vertex is adjacent to no black and two white vertices, while each white vertex is adjacent to one white and two black vertices. As this example shows, vertices in the same cell of an equitable partition do not need to be equivalent under the automorphism group of the graph.

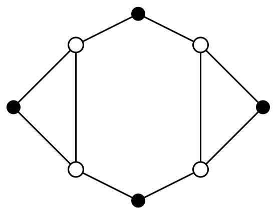


Figure 2.1.4: An equitable partition with two cells.

FACTS

F15: The most well-known partition refinement method splits cells until the partition becomes equitable. Given two cells C_1, C_2, the vertices in C_1 are separated into subcells according to their number of neighbors in C_2. This is repeated for different pairs of cells until no more splitting is possible. Algorithms differ according to the method used to choose the pairs. The fastest algorithm is in [Mc80].

F16: A generalization of this type of refinement, called ***k*-th order Weissfeiler-Lehman refinement**, uses partitions of the set of k-tuples of vertices rather than just a partition of the vertices. For some classes of graphs, it is known that there is a fixed k for which this refinement provides the automorphism partition of the graph, which can be used to build a polynomial-time isomorphism algorithm. However, Cai, Füredi and Immerman showed that no such k is sufficient for all graphs [CaFuIm92].

Search Tree

A partition refinement method is used to define a search tree that is used to find a canonical labeling and the automorphism group.

DEFINITION

D10: Consider a graph and an initial partition (perhaps trivial). Define a ***search tree*** whose nodes are refined partitions. The root of the tree is the refined initial partition. Any node which is a discrete partition (one with only singleton cells) is a leaf of the tree. Consider any node ν which is not discrete. Choose a non-singelton cell C. Then ν has one child for each $v \in C$, obtained by splitting C into two cells $\{v\}$ and $C - \{v\}$, then refining.

FACTS

F17: The leaves of the search tree are discrete partitions, and thus, lists of the vertices in a definite order. The orders define a set of numberings of the vertices of the graph.The maximum labeled graph, according to lexicographic or other convenient ordering, is a canonical form. Moreover, two numberings that define the same labeled graph yield an automorphism, and all automorphisms can be found in that way.

F18: In practice the search tree may be much too large, so various means are employed to reduce it. One way is to employ automorphisms as they are discovered to prune branches of the tree thus shown to be equivalent to other branches. Another way is to employ invariants computed at the nodes of the tree to perform a type of branch-and-bound. A third method is to use a more powerful refinement procedure.

Software

The first program that could process structurally highly-regular graphs and graphs with hundreds of vertices was that of the author, which became known as `nauty` [Mc78, Mc80] and dominated the field from the 1970s until recent years. Now there are several strong competitors.

EXAMPLES

E3: `nauty`, by this author, can find automorphisms groups and canonical forms, of graphs and digraphs. It comes in two forms, with either dense or sparse data structures [McPi12b, McPi13].

E4: `Saucy`, by Darga, Liffiton, Sakallah and Markov, computes automorphism groups and is especially efficient for large sparse graphs having many automorphisms that move few vertices [DaLiSaMa04, DaSaMa04].

E5: `Bliss`, by Juntilla and Kaski, can also perform canonical labeling and has very dependable performance for highly regular graphs [JuKa07, JuKa11].

E6: `Traces`, by Piperno, introduced an entirely new way of scanning the search tree, using a combination of breadth-first and depth-first search [Pi08, McPi13]. At the time of writing, `Traces` is the most efficient program for processing many classes of very difficult graphs, as well as being highly competitive for easy graphs [McPi13]. Since January 2013, `Traces` has been distributed with `nauty` [McPi12].

E7: Other worthy programs are `conauto` by López-Presa, Fernándes Anta and Núñes Chiroque [LPFe09, JPFe11], `VSEP` by Stoichev, and `VF` by Cordella, Foggia, Sansone and Vento.

E8: Packages which contain high-quality graph isomorphism facilities (usually via `nauty`) include Magma, GRAPE, LINK, Sage-combinat, and Macaulay2.

REMARKS

R4: An experimental comparison of `nauty`, `Traces`, `saucy`, `Bliss` and `conauto` can be found in [McPi13].

R5: All the named programs have exponential running time in the worst case. However, the worst-case graphs are rather difficult to find and most users will see only scaling according to a polynomial of low degree. The state of the art is that the easiest graphs can be handled if they fit into main memory (tens of millions of vertices). The most difficult graphs cause difficulty in the hundreds or thousands of vertices.

References

[ArZuUsFa74] V. L. Arlazarov, I. I. Zuev, A. V. Uskov, and I. A. Faradzev, An algorithm for the reduction of finite non-oriented graphs to canonical form. *Zh. vȳchisl. Mat. mat. Fiz.* 14 (1974) 737–743.

[BaKaLu83] L. Babai, W. M. Kantor, and E. M. Luks, Computational complexity and the classification of finite simple groups. In: Proceedings of the 24th Annual Symposium on the Foundations of Computer Science (1983) 162–171.

[Bo90] H. Bodlaender, Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees. *J. Algorithms* 11 (1990) 631–643.

[Bo78] K. S. Booth, Isomorphism testing for graphs, semigroups, and finite automata are polynomially equivalent problems. *SIAM J. Comput.* 7 (1978) 273–279.

[CaFuIm92] Jin-yi Cai ,Martin F´urer, and Neil Immerman, An optimal lower bound on the number of variables for graph identifications. *Combinatorica* 12 (1992) 389–410.

[CoGo70] D. G. Corneil and C. C. Gotlieb, An efficient algorithm for graph isomorphism. *JACM* 17 (1970) 51–64.

[DaLiSaMa04] P. T. Darga, M. H. Liffiton, K. A. Sakallah and I. L. Markov, Exploiting structure in symmetry detection for CNF. In: Proceedings of the 41st Design Automation Conference (2004), 530–534.

[DaSaMa04] P. T. Darga, K. A. Sakallah, and I. L. Markov,. Faster Symmetry Discovery using Sparsity of Symmetries. In: Proceedings of the 45th Design Automation Conference (2004), 149–154.

[FiMa80] I. S. Filotti and J. N. Mayer, A polynomial-time algorithm for determining the isomorphism of graphs of fixed genus. In: Proceedings of the 12th ACM Symposium on Theory of Computing (1980), 236–243.

[GoMiWi91] O. Goldreich, S. Micali and A. Wigderson, Proofs that yield nothing but their validity, or all languages in np have zero-knowledge proof systems. *JACM* 38 (1991) 690–728.

[Gr10] M. Grohe, Fixed-point definability and polynomial time on graphs with excluded minors. In: Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science (2010), 179–188.

[Gr12] M. Grohe, Structural and Logical Approaches to the Graph Isomorphism Problem, In: Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (2012), 188.

[JuKa07] T. Junttila and P. Kaski, Engineering an efficient canonical labeling tool for large and sparse graphs. In: Proceedings of the 9th Workshop on Algorithm Engineering and Experiments and the 4th Workshop on Analytic Algorithms and Combinatorics (2007), 135–149.

[JuKa11] T. Junttila and P. Kaski, Conflict Propagation and Component Recursion for Canonical Labeling. In: Proceedings of the 1st International ICST Conference on Theory and Practice of Algorithms (2011), 151–162.

[KaSc03] V. Kaibel and A. Schwartz, On the complexity of polytope isomorphism problems. *Graphs and Combinatorics* 19 (2003) 215–230.

[LPFe09] J. L. López-Presa and A. Fernández Anta, Fast algorithm for graph isomorphism testing. In: Proceedings of the 8th International Symposium on Experimental Algorithms (2009), 221–232.

[JPFe11] J. L. López-Presa, A. Fernández Anta and L. Núñez Chiroque, Conauto-2.0: Fast isomorphism testing and automorphism group computation. Preprint 2011. Available at `http://arxiv.org/abs/1108.1060`.

[Lu81] A. Lubiw, Some NP-complete problems related to graph isomorphism. *SIAM J. Comput.* 10 (1981) 11–21.

[Lu82] E. Luks, Isomorphism of graphs of bounded valence can be tested in polynomial time. *J. Comp. System Sci.* 25 (1982) 42–65.

[Mc78] B. D. McKay, Computing automorphisms and canonical labelings of graphs. In: Combinatorial Mathematics, Lecture Notes in Mathematics, 686. Springer-Verlag, Berlin (1978), 223–232.

[Mc80] B. D. McKay, Practical graph isomorphism. *Congr. Numer.* 30 (1980) 45–87.

[McPi12] B. D. McKay and A. Piperno, `nauty Traces`, Software distribution web page. `http://cs.anu.edu.au/∼bdm/nauty/` and `http://pallini.di.uniroma1.it/`.

[McPi12b] B. D. McKay and A. Piperno, nauty and Traces User's Guide (Version 2.5). available at [McPi12].

[McPi13] B. D. McKay and A. Piperno, Practical graph isomorphism II, to appear.

[Mi78] G. L. Miller, On the $n^{\log n}$ isomorphism technique. In: Proceedings of the 10th ACM Symposium on Theory of Computing (1978) 51–58.

[Mi80] G. L. Miller, Isomorphism testing for graphs of bounded genus. In: Proceedings of the 12th ACM Symposium on Theory of Computing (1980), 225–235.

[PaRe69] R. Parris and R. C. Read, A coding procedure for graphs. Scientific Report. UWI/CC 10. Univ. of West Indies Computer Centre, 1969.

[PeRo97] E. Petrank and R. M. Roth, Is code equivalence easy to decide? *IEEE Trans. Inform. Th.* 43 (1997) 1602–1604.

[Pi08] A. Piperno, Search space contraction in canonical labeling of graphs. Preprint 2008–2011. Available at `http://arxiv.org/abs/0804.4881`.

[Po88] I. N. Ponomarenko, The isomorphism problem for classes of graphs that are invariant with respect to contraction (Russian). *Zap. Nauchn. Sem. Leningrad. Otdel. Mat. Inst. Steklov. (LOMI)* 174 (1988) no. Teor. Slozhn. Vychisl. 3, 147–177.

[ReCo77] R. C. Read and D. G. Corneil, The graph isomorphism disease. *J. Graph Theory* 1 (1977) 339–363.

[STPi94] J. Shawe-Taylor and T. Pisanski, Homeomorphism of 2-complexes is graph isomorphism complete. *SIAM J. Comput.* 23 (1994) 120–132.