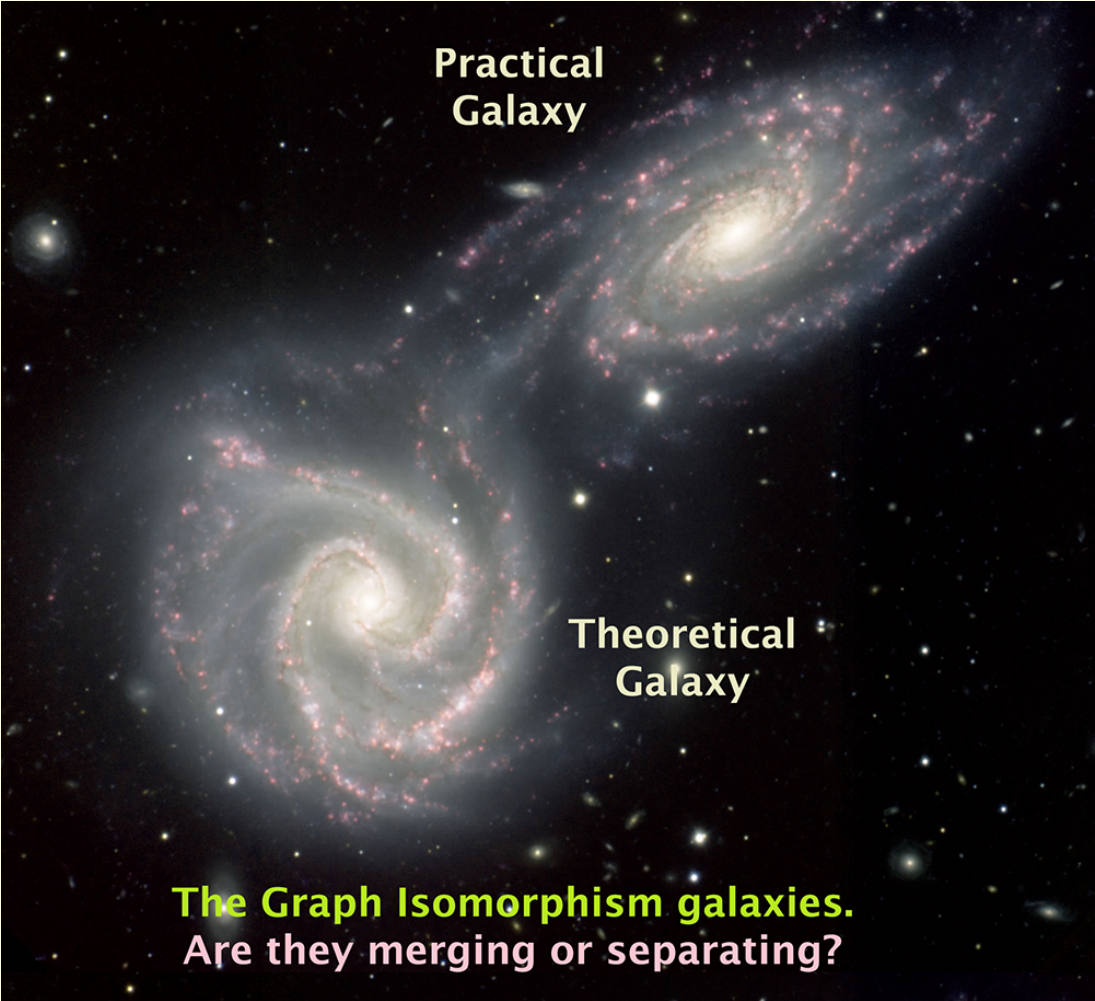


55 Years of Graph Isomorphism Testing

Brendan McKay

Australian National University

The image shows two spiral galaxies in a dark, star-filled space. The galaxy in the upper right is labeled 'Practical Galaxy' and has a bright, well-defined central core and clear spiral arms. The galaxy in the lower left is labeled 'Theoretical Galaxy' and appears more diffuse, with a less distinct core and more irregular spiral structure. The background is filled with numerous small, distant stars of varying colors.

**Practical
Galaxy**

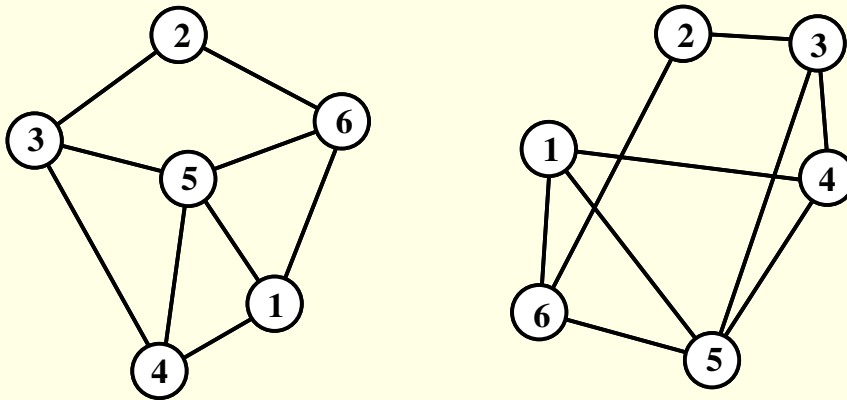
**Theoretical
Galaxy**

**The Graph Isomorphism galaxies.
Are they merging or separating?**

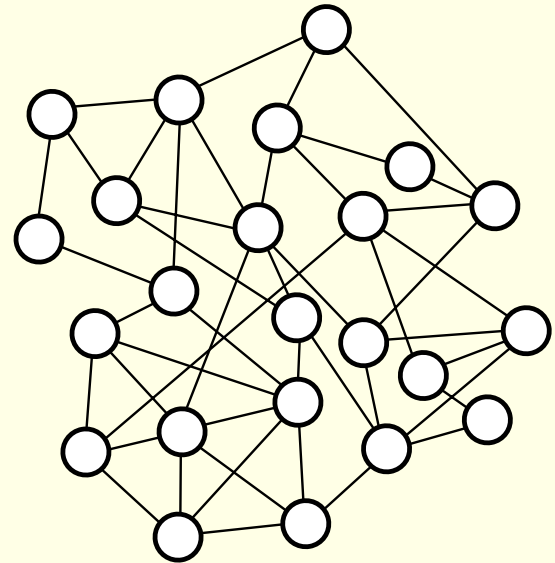
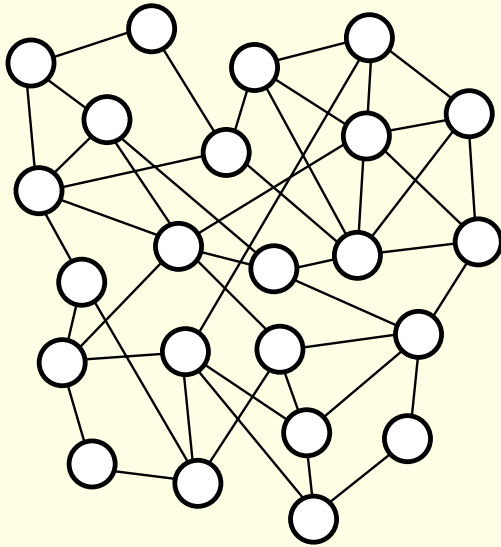
The concept of isomorphism

We have some **objects** consisting of some finite sets and some relations on them.

An **isomorphism** between two objects is a bijection between their sets so that the relations are preserved. Hopefully, isomorphism is an equivalence relation on the set of all objects.



The concept of isomorphism (continued)



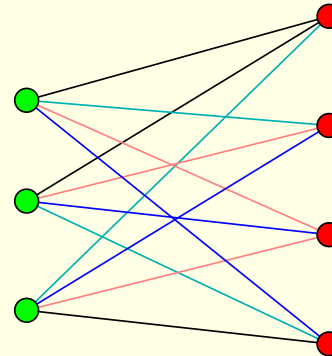
Are these isomorphic?

The computer answers in about 5 microseconds.

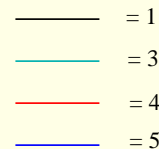
Ubiquity of graph isomorphism

Very many isomorphism problems can be efficiently expressed as **graph isomorphism** problems. For convenience, we use **coloured graphs**, in which the vertices (and possibly the edges) are coloured. Isomorphisms must preserve vertex and edge colour.

$$\begin{pmatrix} 1 & 3 & 5 & 4 \\ 1 & 5 & 4 & 3 \\ 3 & 4 & 5 & 1 \end{pmatrix}$$



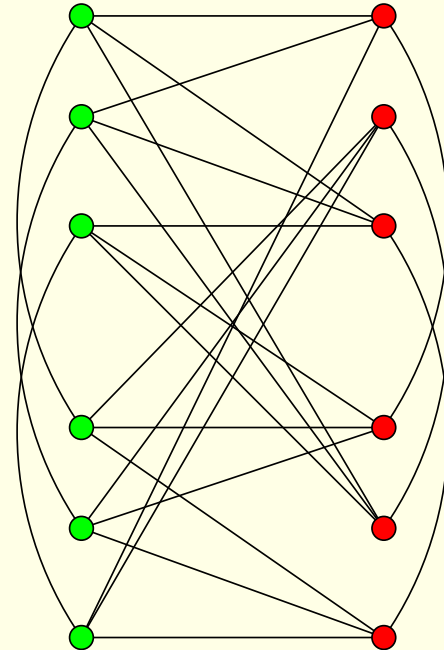
Say two matrices are **isomorphic** if one can be obtained from the other by permuting rows and columns.



Hadamard equivalence

$$\begin{pmatrix} 1 & -1 & 1 \\ 1 & -1 & 1 \\ -1 & -1 & 1 \end{pmatrix}$$

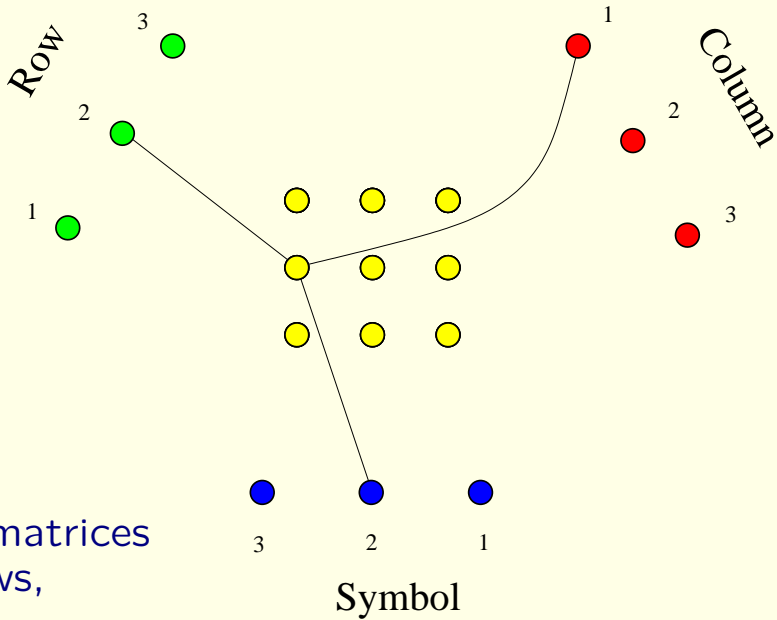
Say **isomorphism** of two ± 1 matrices allows permutation and negation of rows and columns.



Isotopy

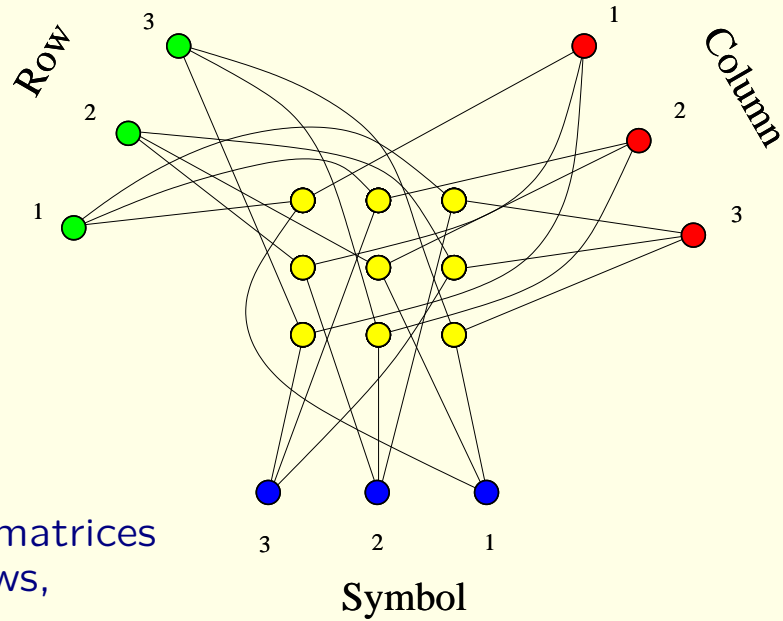
$$\begin{pmatrix} 1 & 3 & 2 \\ 2 & 1 & 3 \\ 3 & 2 & 1 \end{pmatrix}$$

Say **isomorphism** of two matrices allows permutation of rows, columns and symbols.



Isotopy

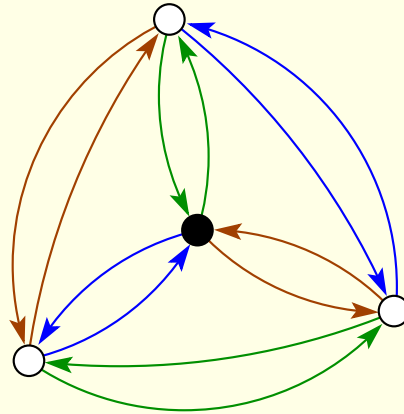
$$\begin{pmatrix} 1 & 3 & 2 \\ 2 & 1 & 3 \\ 3 & 2 & 1 \end{pmatrix}$$



Say **isomorphism** of two matrices allows permutation of rows, columns and symbols.

Group isomorphism

$$\begin{bmatrix} e & g_1 & g_2 & g_3 \\ g_1 & e & g_3 & g_2 \\ g_2 & g_3 & e & g_1 \\ g_3 & g_2 & g_1 & e \end{bmatrix}$$



The group elements are edge colours and vertices.

The black vertex is the identity element.

Preserve vertex colours but **allow permutation of edge colours**.

The fastest known algorithm for group isomorphism is trivial and has time $n^{O(\log n)}$.

Permutation equivalence of linear codes

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

?

Suppose **isomorphism** of two 0-1 matrices means that their columns can be permuted so that the rows generate the same vector space over $GF(2)$.

Nobody knows how to make an equivalent graph problem of size which is polynomial in the size of the generator matrix.

Theoretical complexity

The graph isomorphism problem is:

GI: Given two graphs, determine whether they are isomorphic.

Theoretical complexity

The graph isomorphism problem is:

GI: Given two graphs, determine whether they are isomorphic.

- The fastest known algorithm for GI is $n^{O((\log n)^b)}$ time for constant b (Babai, 2016); for maximum degree d the best is $n^{O((\log d)^c)}$ time for constant c (Grohe, Neuen & Schweitzer, 2018).
- GI is not known to be NP-complete (but there are strong indications that it isn't).
- GI is not known to be in co-NP.
- There are polynomial time algorithms for many special classes of graphs (bounded degree, bounded genus, classes with excluded minors or topological subgraphs). Few of these are practical, planar graphs being a notable exception.

Canonical labelling

Given a class of objects, and a definition of “isomorphism”, we can choose one member of each isomorphism class and call it the **canonical** member of the class.

The process of finding the canonical member of the isomorphism class containing a given object is called “**canonical labelling**”.

Two labelled objects which are isomorphic become identical when they are canonically labelled.

Since identity of objects is usually far easier to check than isomorphism, canonical labelling is the preferred method of reducing a collection of objects to one member of each isomorphism class: **Canonically label each object then use a sorting or hashing algorithm to detect isomorphs.**

Canonical labelling (continued)

A possible definition of the canonical member of an isomorphism class would be the member which maximises some linear representation (such as a list of edges).

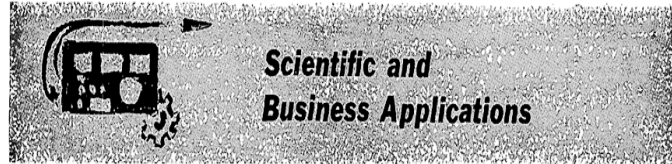
$$\begin{aligned} & \{\{0, 2\}, \{0, 3\}, \{1, 2\}, \{1, 3\}, \{2, 4\}, \{2, 5\}, \{3, 6\}, \{4, 6\}\} \\ & < \{\{0, 2\}, \{0, 3\}, \{1, 2\}, \{1, 4\}, \{2, 4\}, \{2, 5\}, \{3, 6\}, \{4, 6\}\} \end{aligned}$$

In practice, most programs compute a canonical labelling which is easy for computers to find rather than easy for humans to define.

Some history

Actual programs for graph isomorphism began to appear in the early 1960s.

The main applications were chemistry, physics, linguistics, and combinatorial generation.



D. TEICHROEW, Editor

GIT—A Heuristic Program for Testing Pairs of Directed Line Graphs for Isomorphism*

STEPHEN H. UNGER
Columbia University, New York, N. Y.

Given a pair of directed line graphs, the problem of ascertaining whether or not they are isomorphic is one for which no efficient algorithmic solution is known. Since a straightforward enumerative algorithm might require 40 years of running time on a very high speed computer in order to compare two 15-node graphs, a more sophisticated approach seems called for. The situation is similar to that prevailing in areas such as game-playing and theorem-proving, where practical algorithms are unknown (for the interesting cases), but where various practical though only *partially* successful techniques are available. GIT—Graph Isomorphism Tester—incorporates a variety of processes that attempt to narrow down the search for an isomorphism, or to demonstrate that none exists. No one scheme is relied upon exclusively for a solution, and the program is designed to avoid excessive computation along fruitless lines. GIT has been written in the COMIT language and successfully tested on the IBM 7090.

1. Introduction

The object of the research reported here is to investigate

these are of an enumerative nature, and the amount of time required grows so steeply with the problem size that except for trivial cases, such solutions are utterly impractical with any conceivable computer. Thus the mere existence of an algorithm is of little practical significance unless it is reasonably efficient.

Often, we simply do not know whether or not there exists an efficient algorithmic solution. In such cases, either of two approaches is possible: an effort might be made to find an efficient algorithm, or a nonalgorithmic approach may be attempted. The choice must be made on the basis of time available, the importance of coming up with *some* solution, and estimates as to the likelihood of success in pursuing the first course.

It is difficult to specify general procedures to be followed in constructing nonalgorithmic (frequently termed *heuristic*) solutions. (An important goal of the research described here is to shed light on this question.) However, a few guiding principles can be stated at the outset. In the absence of a single road leading directly to a solution, it is necessary for the program to explore many paths which appear to lead in the correct general direction. It has been pointed out by Newell, Shaw and Simon [5] that it is usually more profitable to try a large number of approaches rather than to follow a small number of paths a great distance. In other words, a broad approach in which a variety of incomplete measures are brought to bear on the problem without exhausting the possibilities of any of them is likely to be more successful than an

Unger (1964)

A Graph-Theoretic Algorithm for Matching Chemical Structures*

EDWARD H. SUSSENGUTH, JR.,**

The Computation Laboratory of Harvard University, Cambridge 38, Massachusetts

Received May 26, 1963

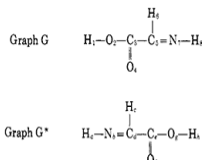
I. INTRODUCTION

A chemical information retrieval system consists primarily of a large file or library of chemical compounds which is to be searched in response to the request of a chemist. The chemist formulates his request as the structural diagram of a chemical. This diagram may be in either of two forms. In the first form the diagram is a complete compound, thereby indicating that the chemist wishes to locate that compound in the file which exactly matches the query structure. In the second form the diagram represents a piece or fragment of a compound, and the retrieval system is required to locate all compounds in the library which contain the query as an integral part; that is, the query item must exactly match part of the library item.

There are many chemical retrieval systems which process the first type of request efficiently. Most of these systems are also capable of handling certain fragment requests; however, the fragments which can be processed are frequently of a restricted nature. For example, in retrieval systems which are based on linear ciphers, only those fragments which are explicit in the cipher are readily detected. To allow a completely general specification of fragments it seems inevitable that a detailed atom-by-atom comparison is required of the query and library structures. A technique for making such detailed comparisons is presented in this report. This technique is novel in that it avoids the excessive backtracking and restarting required by other atom-by-atom matching procedures.

Before giving the details of the proposed algorithm,

A graph consists of a set of *nodes* and a set of *branches* connecting pairs of nodes. The set of nodes connected to node x is denoted as Γx . The operator Γ is also extended to sets, so that ΓA denotes the set of nodes connected to any node of set A . For example, in graph G of Figure 1, $\Gamma 5 = \{3, 6, 7\}$ and $\Gamma\{2, 4\} = \{1, 3\}$. Two graphs G and G^* are *isomorphic* if there is a one-to-one correspondence between the nodes of G and G^* which preserves a one-to-one correspondence between the branches of the graphs. In other words, for all nodes x in G there must be a unique correspondent x^* in G^* , and the correspondence must be such that x is connected to y in G if and only if x^* is connected to y^* in G^* . It is evident that in the graph of a chemical compound the nodes correspond to atoms and the branches to interatomic bonds, so that the nodes and branches are assigned values corresponding to the atom or bond type, respectively. For such graphs, which have values associated with their nodes and branches, the definition of isomorphism is extended in an obvious manner to ensure that the correspondents of node x and branch (x, y) have the same values as x and (x, y) , respectively.



Sussenguth (1964)

A Quasi-Decision Algorithm for the P -Equivalence of Two Matrices

by CORRADO BÖHM and ADELINA SANTOLINI (1)

0. Introduction and summary.

By a *quasi-decision* algorithm is meant in the following an algorithm which gives in most cases a decision procedure. Two matrices are said *P-equivalent* if they are equal up to some permutations of rows and (independently) of columns.

We were led to the problem of deciding efficiently the P -equivalence of pairs of *boolean* matrices by dealing with some questions of mathematical programming and application of the automata theory (2).

The proposed algorithm, though it is not a full decision algorithm has the following features:

1) It is not restricted to boolean matrices as in [1].

2) Its essential step consists of building from a given matrix $A(B)$ a new one of not greater size $A^*(B^*)$ where the number of different elements is generally greater than that in A and with the property that A is S -equivalent (3) to B iff (4) A^* is S -equivalent to B^* . The new pair of matrices is then checked by some elementary necessity tests. If their non- S -equivalence does

Böhm and Santolini (1964)

The Generation of a Unique Machine Description for Chemical Structures—A Technique Developed at Chemical Abstracts Service

H. L. MORGAN

Chemical Abstracts Service, The Ohio State University, Columbus, Ohio 43210

Received January 15, 1965

I. INTRODUCTION

As part of the development of a computer-based chemical information system at CAS, it has been necessary to devise techniques for the registration of drawings of chemical structures. A major purpose of the CAS registration process is to determine whether a particular structure has already been stored in the system. The ability to make this determination makes it possible to

utilize a computer to assign to every chemical structure a unique identifying label. This identifying label, referred to as a registry number, is the thread that ties together all information associated with a particular compound throughout the developing CAS computer system. It is because of this association, made possible by the registration process, that CAS will be able to provide multiple-file correlative searches with assurance that all information on file for a particular compound has been located.

Morgan (1965)

On Ordering and Identifying Undirected Linear Graphs

JOHN F. NAGLE*

Wheatstone Physics Laboratory, King's College, London, England

(Received 17 March 1966)

A general linear ordering relation is presented which completely orders any subset of the undirected linear graphs with the same number of vertices. The discussion is then specialized to the ordering and the identification of the unlabeled stars with no vertices of degree two, which faithfully represent the basic topologies of all stars.

INTRODUCTION

THE study of linear graphs is important to mathematical physics because graphs frequently provide a convenient representation of the terms in series (or perturbation) expansions which are used in discussing otherwise intractable problems. Particular problems in statistical mechanics which may be cited are the Mayer cluster expansion for the virial series of a gas,¹ the Ising-model series expansions,²⁻⁴ the percolation problem,⁵ the excluded volume problem,⁶ and the residual entropy of ice problem.⁷

When it becomes necessary to consider complex linear graphs certain problems arise. First, it becomes difficult to obtain all the graphs of the type wanted. Secondly, there is the problem of identifying a graph with its isomorph in a list. The technique of drawing the graph in several different ways with the hope of making it look like one in the list is very tedious for even moderately complex graphs. The solution to the identification problem is related to the solution to the next problem. This is the question of how to order the graphs conveniently in a numbered list in such a way as to maximize the usefulness of the list to a large number of research workers.

Despite the obvious advantages of adopting a

"natural" groupings of linear graphs. For example, most problems require only well-defined subsets of linear graphs, such as the Mayer virial series for a gas which requires only the multiply connected (no articulation vertices) and unlabeled graphs called stars.¹ Furthermore, within these subsets different gross criteria are important for different problems. For example, for the high-temperature Ising-model series the number of edges e is of more importance than the number of vertices v , while the reverse is true for the residual entropy of ice series.⁷ Other criteria for classification are the cyclomatic number; the degree of each vertex (number of edges incident to the vertex); the number of double, triple, \dots , edges; and so on.

The second reason for not adopting a systematic ordering of linear graphs, even for use with specialized problems, is the difficulty in finding a completely systematic ordering. For example, if one uses any combination of the criteria mentioned above one finds that eventually there are distinct graphs not differentiated by the criteria which must then be ordered arbitrarily. This ultimate arbitrariness has undermined the effort to organize graph data in a systematic way.

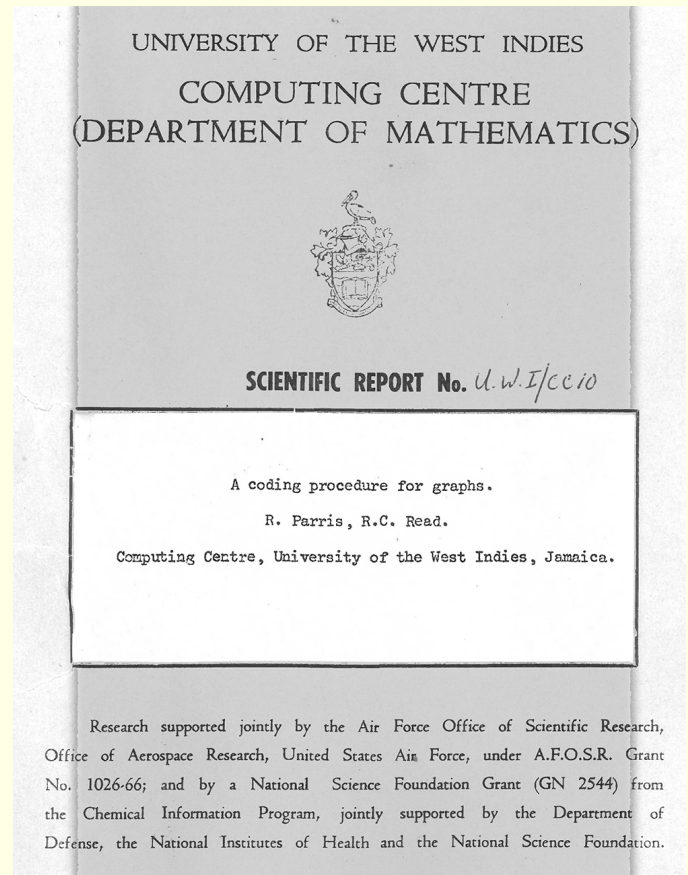
The primary purpose of this paper is to show in

Nagle (1966)

The refinement-individualization tree

Although hints of it appeared earlier, the first clear-cut definition of the refinement-individualization tree appeared in 1967.

Parris and Read explored it in **breadth**-first order.



Parris and Read (1967)

Systematic use of symmetries to prune the search tree may have been first introduced by this 1974 paper.

It didn't use partition refinement.

Other early examples: Tinhofer (1975), Beyer and Proskurowski (1976), McKay (1976).

AN ALGORITHM FOR THE REDUCTION OF FINITE NON-ORIENTED GRAPHS TO CANONICAL FORM*

V. L. ARLAZAROV, I. I. ZUEV, A. V. USKOV
and I. A. FARADZHEV

Moscow

(23 March 1973)

TRADITIONAL methods for establishing the isomorphism of graphs, using geometric invariants, are inefficient on graphs containing strongly regular fragments. An algorithm using the principle of a different approach to the solution of isomorphic problems, which works fairly efficiently on strongly regular graphs, is presented.

1. In the mass solution of the problem of establishing the isomorphism of two graphs it is convenient to use the following reduction. In each class of pairwise isomorphic graphs one graph is chosen, called the *canonical form* of any graph of this class. After this the question of the isomorphism of two graphs reduces to the construction and comparison of their canonical forms. The canonical form of a graph is determined by ordering the vertices of the graph in accordance with their properties, independent of the original numbering (*invariants*).

At the present time the question of the existence of an efficient canonisation algorithm, that is, an algorithm leading to the canonical form of an arbitrary graph with n vertices, after a number of operations less than $n!$, remains open. However, many canonisation algorithms exist, solving the problem with a more or less satisfactory working speed for fairly extensive classes of graphs. The majority of these algorithms use geometrical invariants of the vertices and arcs of the graph, of power of the vertices type, the number of paths of various lengths passing through vertices and arcs, etc. [1–4]. In [5, 6] it is shown that such algorithms are ineffective (they either give wrong results, or require of the order of $n!$ operations) for the canonisation of the class of graphs called cages, which include, in particular, *strongly regular* graphs [7].

Arlazarov, Zuev, Uskov and Faradzhev (1974)

Quantity vs quality

During those years, the subject became so popular that it was known as a “disease” .

Most programs were fairly useless and many were just wrong.

Production of wrong programs is still very popular.

The Graph Isomorphism Disease*

Ronald C. Read
UNIVERSITY OF WATERLOO

Derek G. Corneil
UNIVERSITY OF TORONTO

ABSTRACT

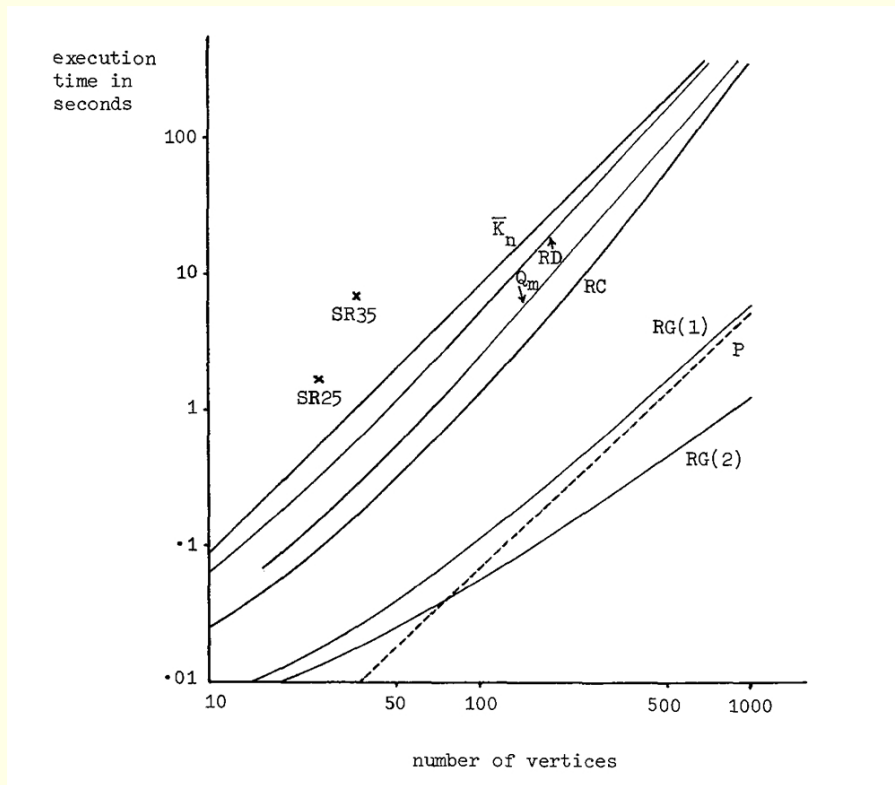
The graph isomorphism problem—to devise a good algorithm for determining if two graphs are isomorphic—is of considerable practical importance, and is also of theoretical interest due to its relationship to the concept of NP-completeness. No efficient (i.e., polynomial-bound) algorithm for graph isomorphism is known, and it has been conjectured that no such algorithm can exist. Many papers on the subject have appeared, but progress has been slight; in fact, the intractable nature of the problem and the way that many graph theorists have been led to devote much time to it, recall those aspects of the four-color conjecture which prompted Harary to rechristen it the “four-color disease.” This paper surveys the present state of the art of isomorphism testing, discusses its relationship to NP-completeness, and indicates some of the difficulties inherent in this particularly elusive and challenging problem. A comprehensive bibliography of papers relating to the graph isomorphism problem is given.

Read and Corneil (1977)

nauty was originally called GLABC and was written in a mixture of Fortran and Assembler starting in 1976.

The C edition called nauty came about 1981.

Important advances in the next 15 years: Kocay (1985), Kirk (1985), Leon (1991).



McKay (1977)

Current software for graph isomorphism

The most successful programs still supported are:

- **nauty** (McKay, 1976–) canonical label and automorphism group
- **VF2** (Cordella, Foggia, Sansone and Vento, 1999–) comparison of two graphs
- **saucy** (Darga, Sakallah and Markov, 2004–) automorphism group
- **bliss** (Juntilla and Kaski, 2010–) canonical label and automorphism group
- **Traces** (Piperno, 2008–) canonical label and automorphism group
- **conauto** (López-Presa, Anta and Chiroque, 2009–) automorphism group and comparison of two graphs
- **VSEP1** (Stoichev, 1997–) automorphism group and comparison of two graphs

Many similar principles appear in these programs.

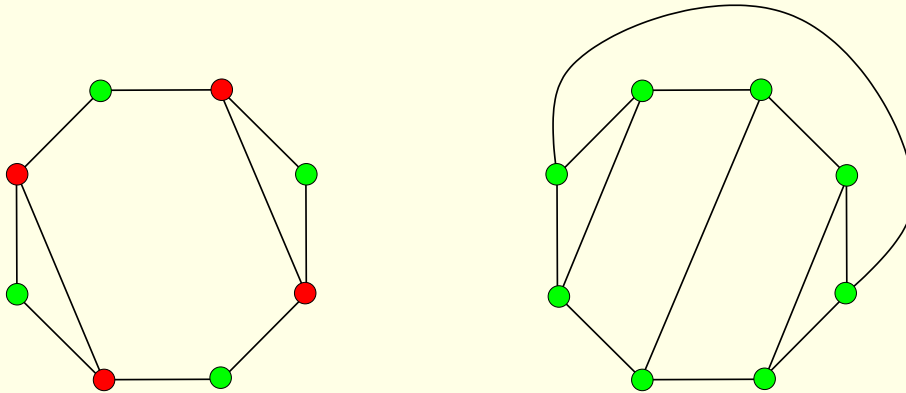
The individualization-refinement paradigm

All of the currently best programs use an individualization-refinement paradigm.

A key concept is partition refinement (“partition” = “colouring”).

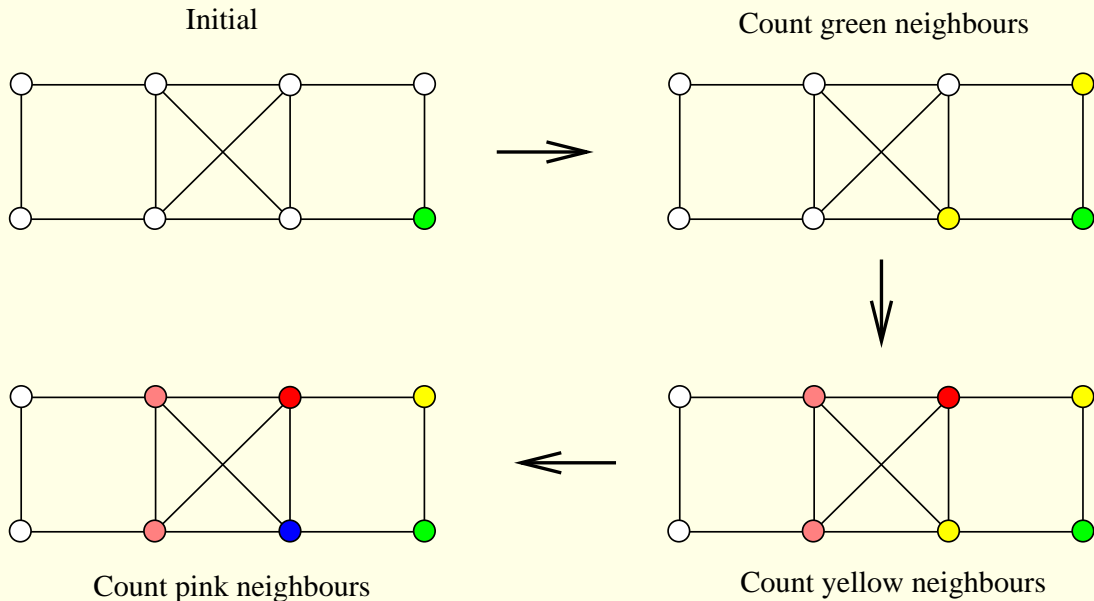
An equitable partition is one where every two vertices of the same colour are adjacent to the same number of vertices of each colour.

Two examples of equitable partitions:



Refinement of a partition means to subdivide its cells.

Given a partition (colouring) π , there is a unique equitable partition that is a refinement of π and has the least number of colours.



Individualization-refinement tree

The nodes of the tree correspond to equitable partitions.

The **root** of the tree corresponds to the initial colouring, refined.

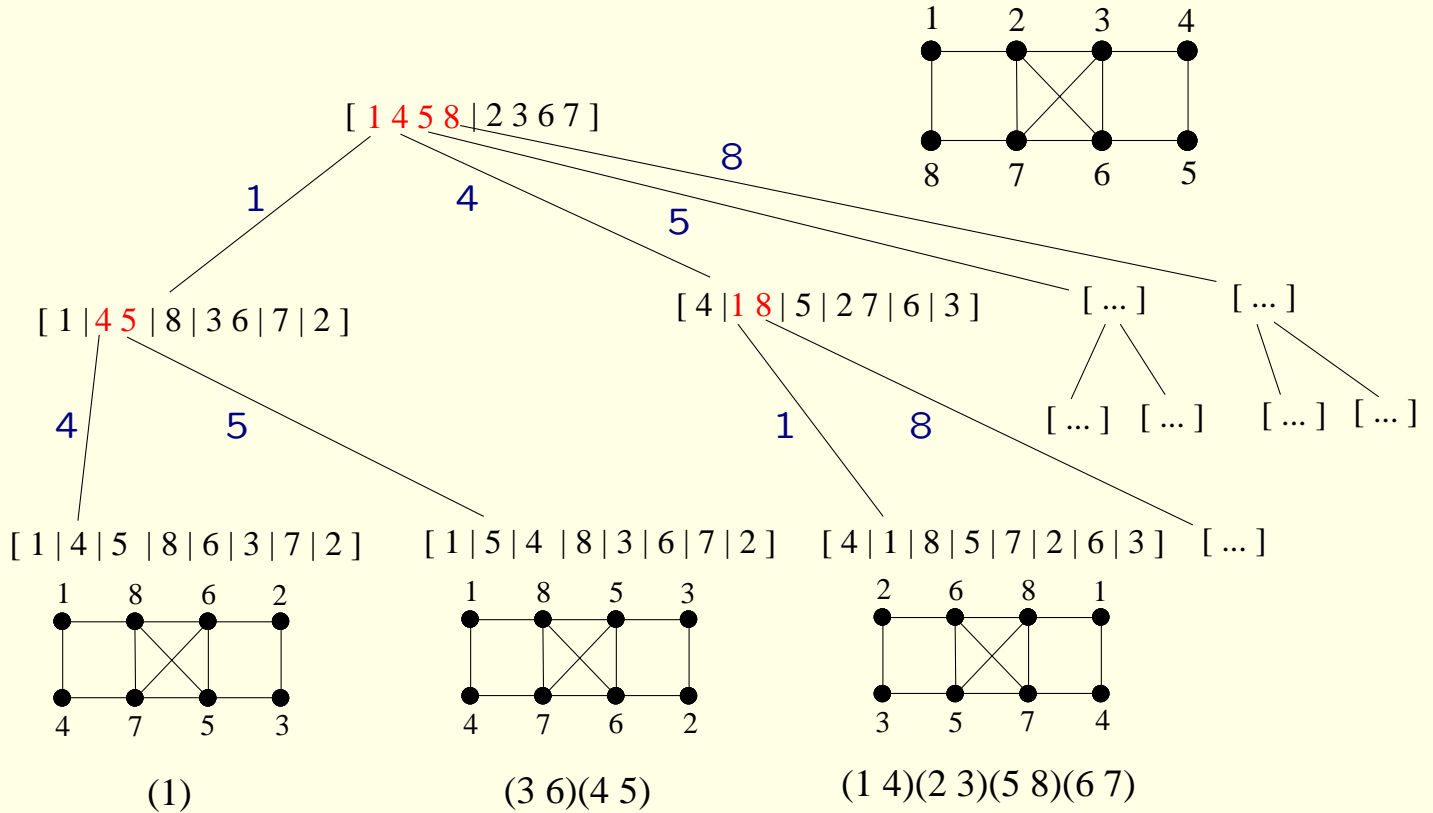
If a node corresponds to a discrete partition (each vertex with a different colour), it has no children and is a **leaf**.

Otherwise we choose a colour used more than once (the **target cell**), **individualize** one of those vertices by giving it a new unique colour, and refine to get a child.

Each leaf lists the vertices in some order (since colours have a predefined order), so it corresponds to a labelling of the graph.

If we define an order on labelled graphs, such as lexicographic order, then the greatest labelled graph corresponding to a leaf is a canonical graph.

Individualization-refinement tree



Node invariants

A **node invariant** is a value $\phi(\nu)$ attached to each node in the tree that depends only on the combinatorial properties of ν and its ancestors in the search tree, **satisfying some technical conditions**.

For example,

$$\phi(\nu) = (c(\nu'), c(\nu''), \dots, c(\nu))$$

is a node invariant with lexicographic ordering, where ν', ν'', \dots, ν is the path from the root of the tree to ν , and $c()$ is the number of colours.

Let ϕ^* be the greatest node invariant of a leaf. Then the lexicographically greatest labelled graph corresponding to a leaf ν with $\phi(\nu) = \phi^*$ is a canonical graph.

This allows parts of the tree which cannot contain a canonical graph to be pruned (**branch-and-bound**).

Automorphism group handling

Automorphisms can be discovered by several means.

- by comparing the labelled graphs corresponding to two leaves (all the programs)
- by monitoring the effect of refinement closely (**saucy** and **Traces**)
- by using the properties of equitable partitions (**nauty** and **Traces**)
- by being provided by the user (**Traces**)

In order to perform automorphism-based pruning of the search tree, we need to efficiently determine if two sequences of vertices are equivalent under the group generated by the automorphisms found so far.

Pruning operations

Node invariants, together with automorphisms, allow us to remove parts of the search tree without generating them.

1. If ν_1, ν_2 are nodes at the same level in the search tree and $\phi(\nu_1) > \phi(\nu_2)$, then no canonical labelling is descended from ν_2 .
2. If ν_1, ν_2 are nodes at the same level in the search tree and $\phi(\nu_1) \neq \phi(\nu_2)$, then no labelled graph descended from ν_1 is the same as one descended from ν_2 .
3. If ν_1, ν_2 are nodes with $\nu_2 = \nu_1^g$ for an automorphism g , then g maps the subtree descended from ν_1 onto the subtree descended from ν_2 . So any labelled graph descended from ν_2 is equal to some labelled graph descended from ν_1 .

Variations between programs

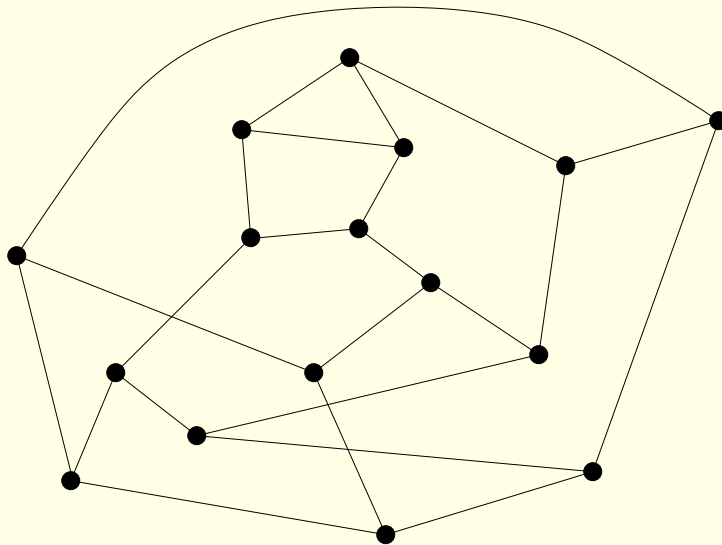
The competing programs vary from each other in ways that include:

1. Data structures
2. Strength of the refinement procedure
3. Order of traversal of the search tree
4. Means of discovering automorphisms
5. Processing of automorphisms

Stronger refinement — *nauty* “invariants”

Sometimes refinement to equitable partition is insufficient to separate vertices with clearly different combinatorial properties. Those properties can be used to make the refinement stronger.

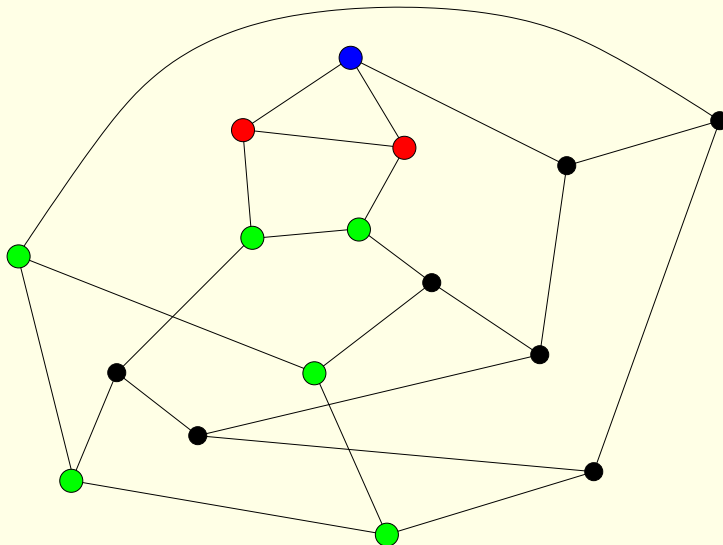
For example, we can count the number of 3-cycles and 4-cycles through each vertex and then refine:



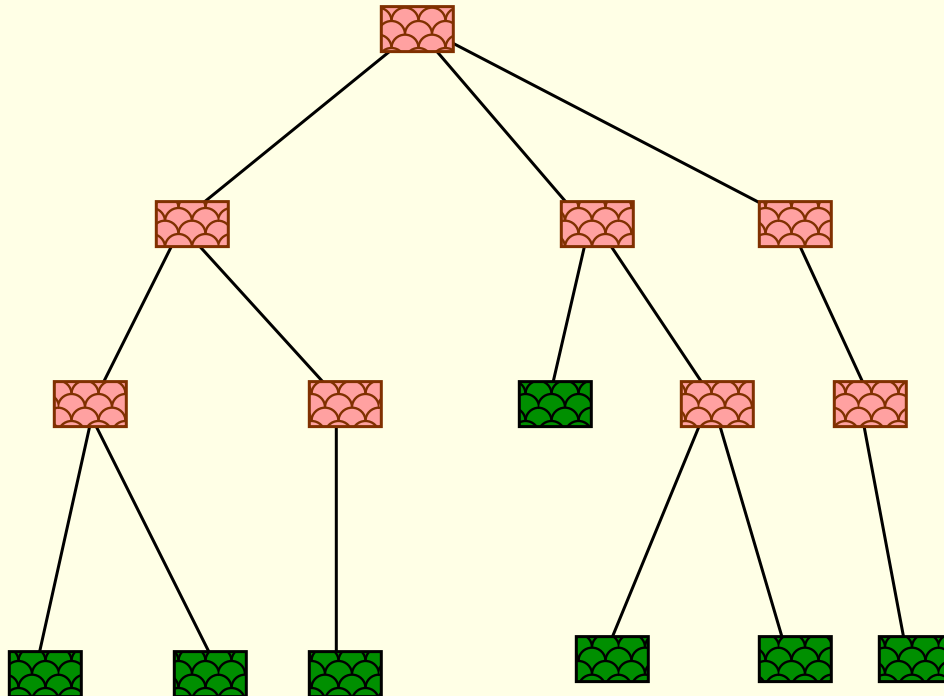
Stronger refinement — *nauty* “invariants”

Sometimes refinement to equitable partition is insufficient to separate vertices with clearly different combinatorial properties. Those properties can be used to make the refinement stronger.

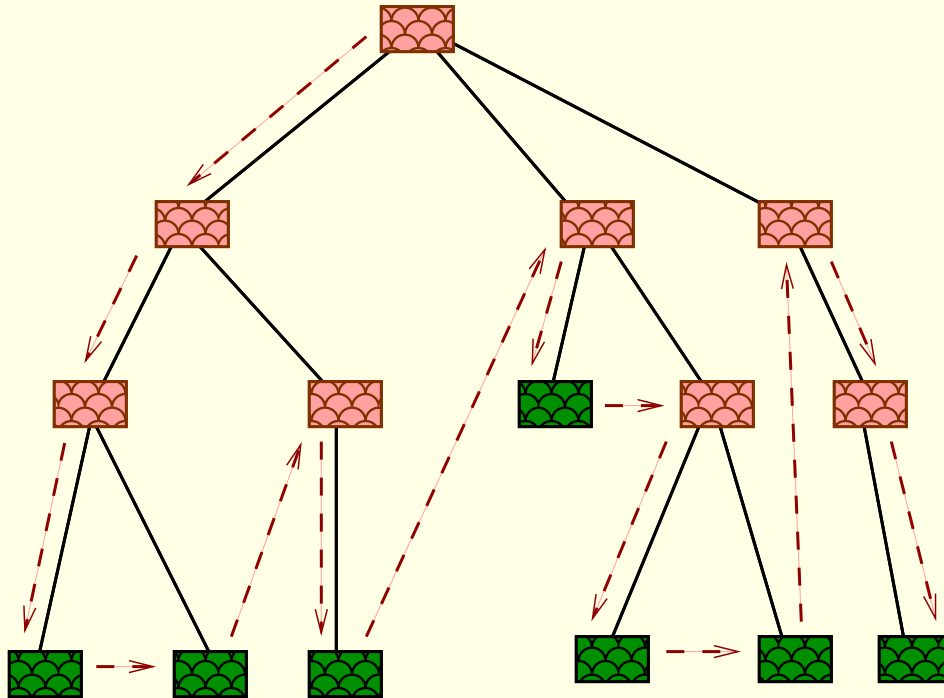
For example, we can count the number of 3-cycles and 4-cycles through each vertex and then refine:



Tree traversal order

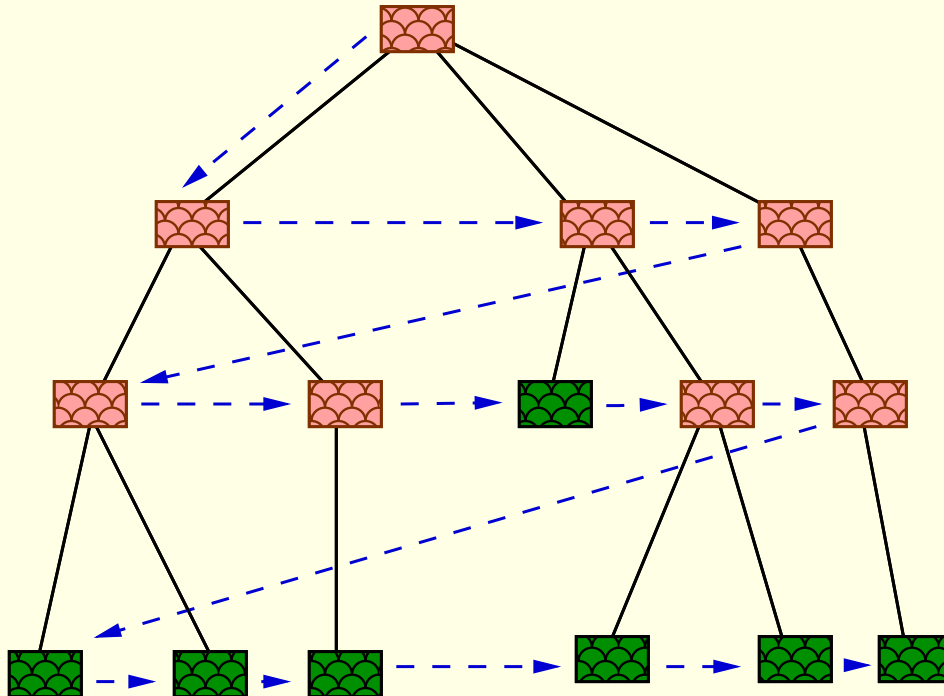


Tree traversal order



nauty order: depth-first search

Tree traversal order

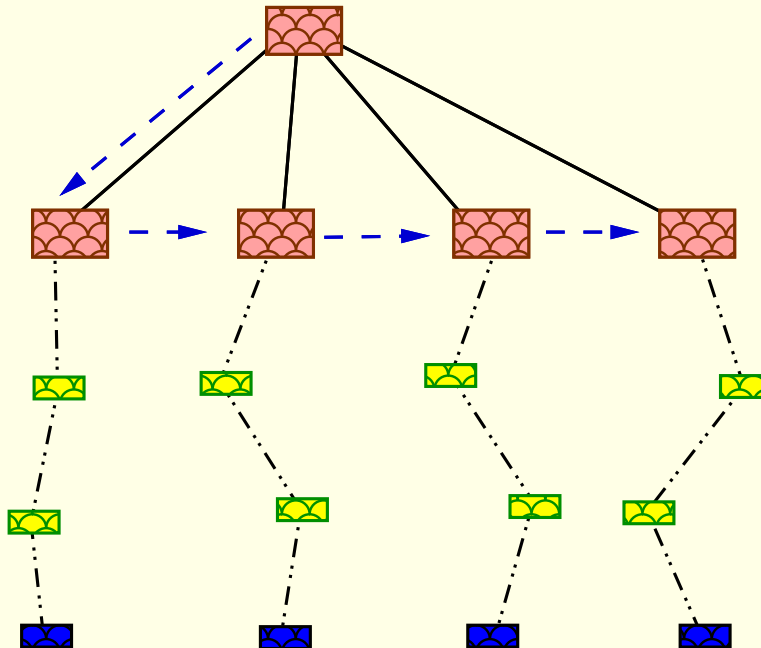


Traces order: breadth-first search

The problem with BFS is that automorphisms are discovered at leaves.

Tree traversal order — a **Traces** innovation

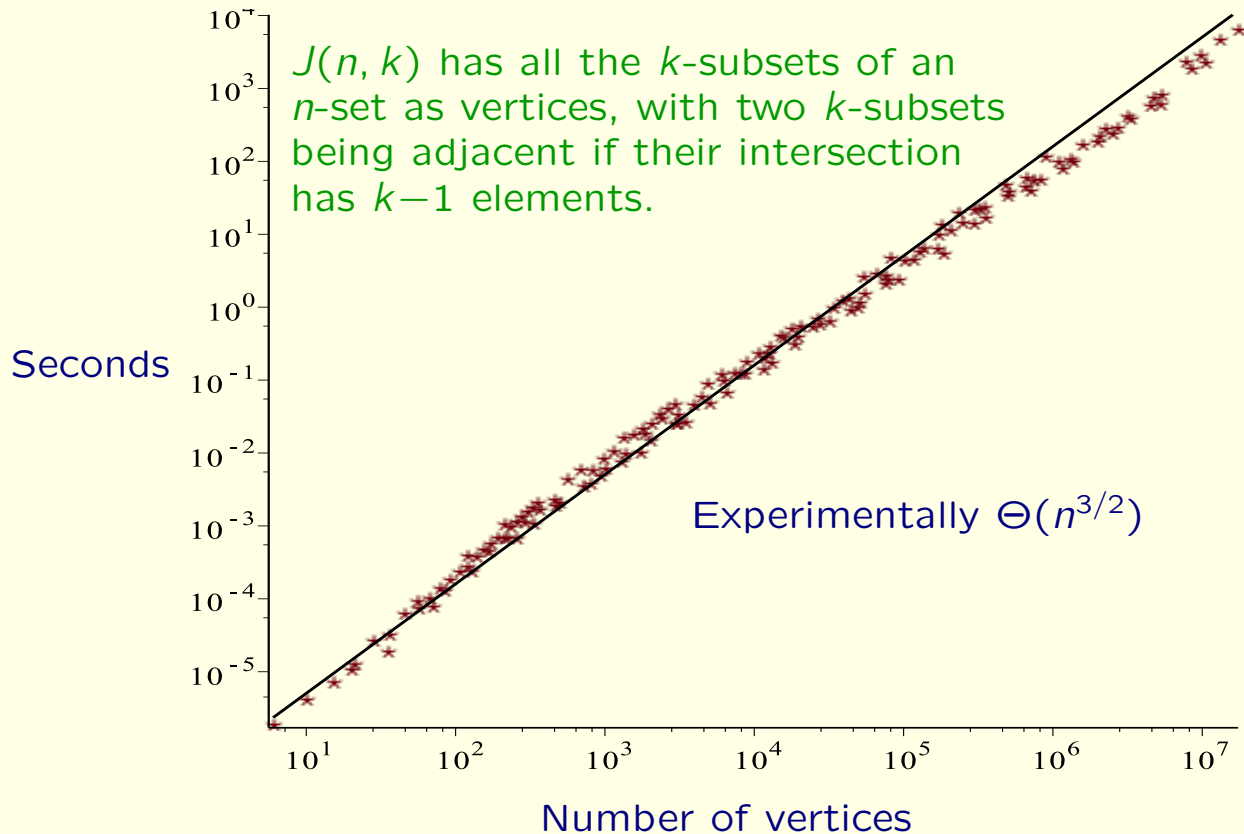
Tree traversal order — a **Traces** innovation



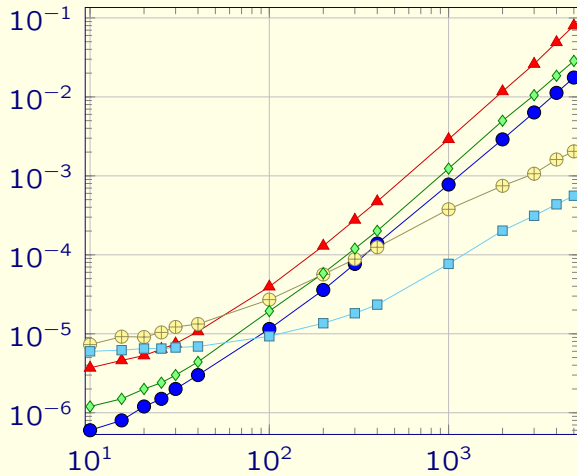
Traces: experimental paths

Experimental paths allow automorphism detection during BFS.

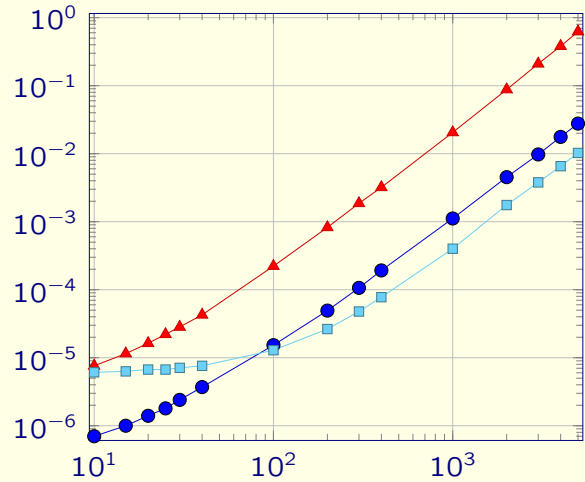
Example: Johnson graphs in nauty



Automorphism group



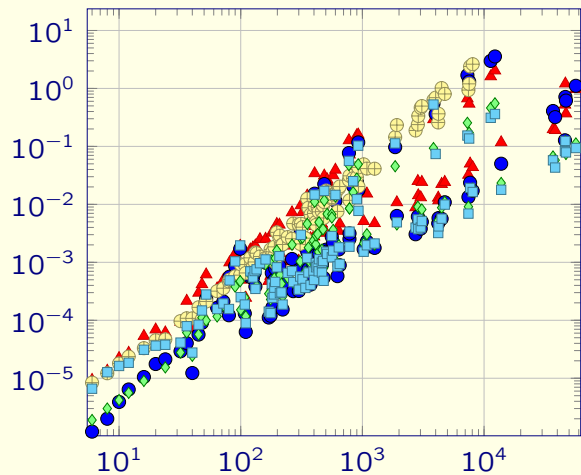
Canonical label



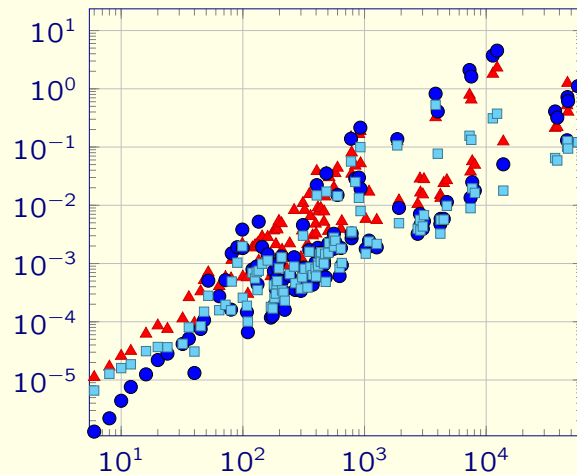
▲ bliss ◆ saucy ⊕ conauto ● nauty ■ Traces

Random graphs with $p = \frac{1}{2}$

Automorphism group



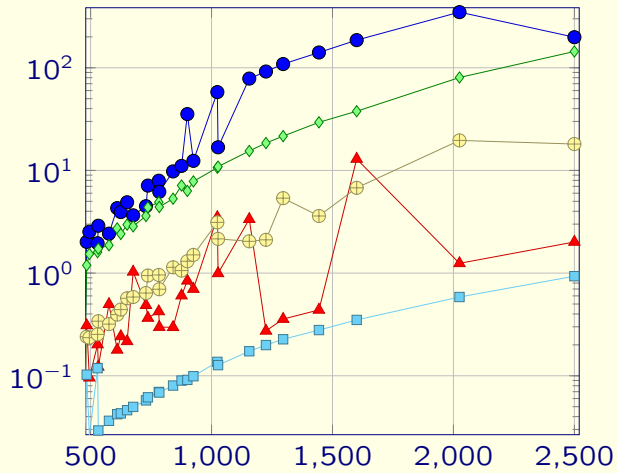
Canonical label



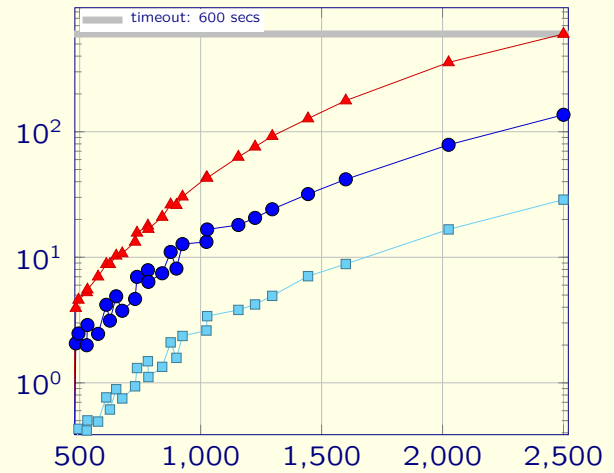
▲ bliss ◆ saucy ⊕ conauto ● nauty ■ Traces

Miscellaneous vertex-transitive graphs

Automorphism group



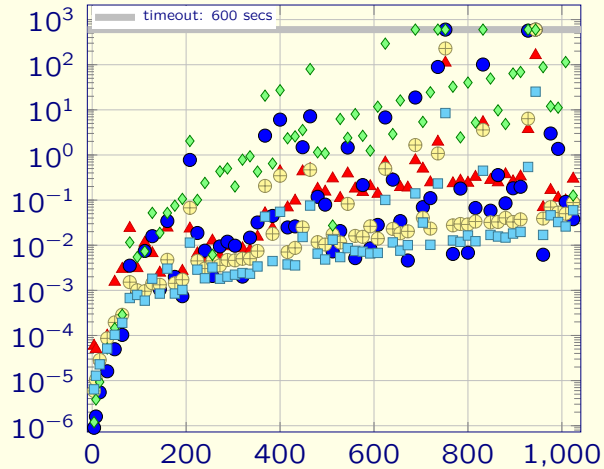
Canonical label



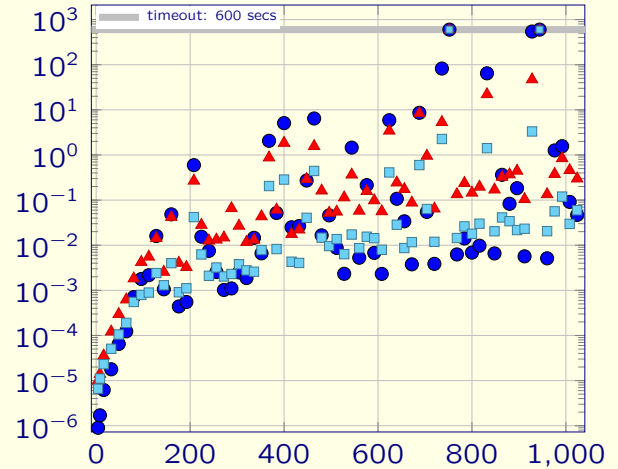
▲ bliss ◆ saucy ⊕ conauto ● nauty ■ Traces

Large strongly-regular graphs

Automorphism group



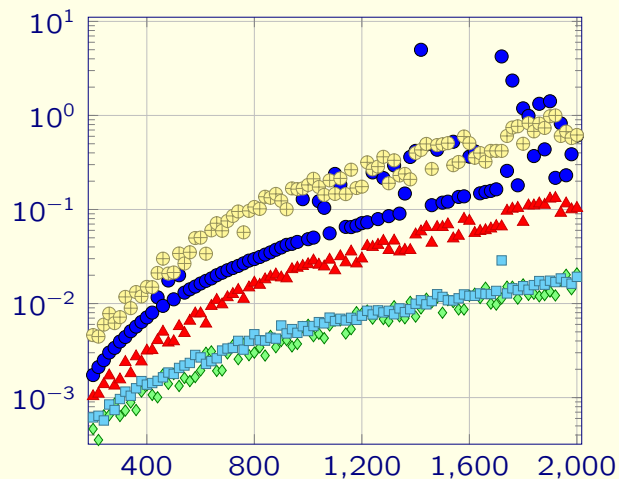
Canonical label



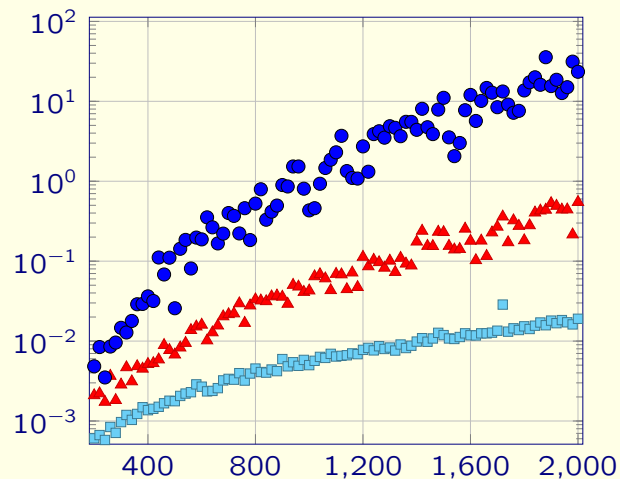
▲ bliss ◆ saucy ⊕ conauto ● nauty ■ Traces

Hadamard matrix graphs

Automorphism group



Canonical label



▲ bliss ◆ saucy ⊕ conauto ● nauty ■ Traces

Cai-Fürer-Immerman graphs

The **bad** news

David Neuen and Pascal Schweitzer (2017) proved that no algorithm using the individualization-refinement paradigm takes less than exponential time in the worst case.

The result remains true if stronger refinement is used (k -dimensional Weisfeiler-Lehmen refinement) and the algorithm is presented in advance with the full automorphism group.

However, no algorithms performing better in practice are known.

So it seems that the theoretical and practical galaxies are separating.