Computers & Industrial Engineering 148 (2020) 106715

Contents lists available at ScienceDirect

Computers & Industrial Engineering

journal homepage: www.elsevier.com/locate/caie

Printed circuit boards isomorphism: An experimental study

Aida Abiad [a,b,*], Alexander Grigoriev [c], Stefanie Niemzok [c]

[a] Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands
[b] Department of Mathematics, Analysis, Logic and Discrete Mathematics, Ghent University, Ghent, Belgium
[c] Department of Data Analytics and Digitalization, Maastricht University, Maastricht, The Netherlands

ARTICLE INFO

Keywords:
Printed circuit board search engine
Information retrieval
Graph isomorphism
Graph invariant
Heuristic
Preprocessing

ABSTRACT

We evaluate the discrimination power of different graph invariants in order to identify an appropriate measurement of similarity among printed circuit boards. We show that computationally expensive algorithmic evaluation of circuit boards equivalence based on the topology of the circuit boards is unnecessary in practice. Instead, the information about the circuit boards can be stored in terms of graph invariants, namely, number sequences characterizing the boards, which requires much less memory and enables quicker recognition of board equivalence. This straightforwardly leads to fast practical implementations of the search engines for the printed circuit boards. Our approach is multidisciplinary, as the tools were drawn from two fields; the industrial electronic engineering field, and the theoretic fields of graph theory and algorithms. A real case study on a search engine for printed circuit boards is conducted to illustrate the proposed approach. Finding an easily computable and complete set of graph invariants remains a challenging mathematical question.

1. Introduction

The relevance of search results influences a user's probability to buy from a company's website. Constructing an accurate and fast search engine is therefore crucial to any platform providing content. This paper investigates the opportunities to integrate companies specific knowledge in their search methods to level up outputs and simplify the process of finding the right answers.

1.1. The practical use case

This research is conducted in collaboration with an electronics provider (in the following referred to as 'the company'), which targets the emerging market of open source hardware projects. A hardware project is characterized by a self-contained unity of an electronic application designated to perform a certain function. A project is called *self-contained* if it works as it is presented without the necessity of adding more components to the electronic circuit board. A *printed or electronic circuit board* is an essential part of modern electronic circuits. It is made of a flat panel of insulating materials with patterned copper foils that act as electric pathways for various components such as ICs, diodes, capacitors, resistors, and coils (Kwon, Omitaomu, & Wang, 2008). While the term *open source* refers to the public availability of the hardware's source that anyone can study, modify, distribute or sell. Electronic schematics communicate the relevant details about structure and components contained in the aforesaid electronic application. Habituated to access many ingenious software products for free under the open source software licenses, this way of publishing hardware projects is fairly recent and just starting to develop. The company is establishing a platform that will facilitate access to such hardware projects. In particular, the company provides the opportunity for hobbyists to both officially publish their projects and place them on sale at considerably lower prices than individual purchases entail. On the other side, there are the users, who use the platform as of finding published projects and purchasing the assembly kit containing all required electrical components.

Hence, the problem at hand that the company faces is to identify a fast search strategy that provides users with relevant content. The current elemental used search is based on text and generally yields exact string matches and merely includes similarity measures on customization which handle typos and singular-plural matters cogently (for instance distance measures and fuzzy string matching). This paper aims at enriching the company's current search engine by using additional knowledge retrieved from the actual products sold on the platform, the hardware projects, which include descriptions about built-in components and their interconnections.

1.2. Notation and preliminaries

This section sets up the notation and the necessary definitions.

* Corresponding author at: Department of Mathematics, Analysis, Logic and Discrete Mathematics, Ghent University, Ghent, Belgium.
E-mail addresses: a.abiad.monge@tue.nl (A. Abiad), a.grigoriev@maastrichtuniversity.nl (A. Grigoriev), stefanie@niemzok.de (S. Niemzok).

https://doi.org/10.1016/j.cie.2020.106715
Received 6 February 2020; Received in revised form 14 July 2020; Accepted 29 July 2020
Available online 5 August 2020

A graph is a structure amounting to a set of objects in which some pairs of the objects are in some sense related. The objects correspond to mathematical abstractions called *vertices* (also called *nodes*) and each of the related pairs of vertices is called an *edge*. Therefore, formally a *graph* is a pair $G = (V, E)$, where V is a set whose elements are called vertices and E is a set of two-sets (sets with two distinct elements) of vertices, whose elements are called *edges*.

A *simple graph* describes a graph that contains at most one edge between any two distinct vertices. A graph is said to be *loopless* if edges from a vertex to itself are not allowed.

Graphs that contain additional text information are commonly referred to as *labeled graphs*. Labeled graphs can possess labels on the vertices, on the edges or even on both.

The *degree* of a vertex of a graph is the number of edges that are incident to the vertex. The *diameter* of a graph is the path with the greatest length of all shortest paths connecting any pair of vertices in the graph.

The *Graph Isomorphism* problem aims to answer whether two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic or not. An *isomorphism* between two graphs is affirmed if a connection preserving mapping from all of the nodes of V_1 to all of the nodes of V_2 is found.

A *graph invariant* is a property that is preserved in all isomorphic graphs. Some examples of graph invariants include the number of vertices, the number of edges or the degree sequence. Recall that it is not necessarily true that two graphs possessing identical graph invariants are isomorphic. On the other hand, if two graphs have a different output for the same graph invariant, then they are clearly non isomorphic. In this paper, graphs invariant will play an important role for the preprocessing step in the proposed heuristics. Among the graphs invariants that we will consider, there are the degree and the status sequence. The *degree sequence* describes an ordered list of all occurring degrees in a graph, and this sequence has the length of the number of nodes in the graph. The *status sequence*, which also has length equal to the number of nodes in the graph, is an ordered list of all status of the nodes of the graph. The *status* of a node is the sum of the shortest paths to all other nodes contained in the graph.

1.3. The mathematical problem

The main research question of this article:

How to include physical electronic circuit boards in a company's practice of search?

breaks down to determining which boards are actually the same, or displaying similarities in features.

The idea of integrating electronic circuit boards in search necessitates the transformation into a valid data type, readable by a computer and containing the same information as the physical board. Although most search platforms accept content from various sources, they most commonly have to contain textual information (the concatenation of characters and numbers). The first step is therefore to transform the schematics of the electronic circuit boards into graphs, and to take advantage of the related theoretical tools on the field of graph theory. The precise conversion from electric schemes to mathematical graphs is detailed in Section 3.2. Since we use graphs as underlying structure, searching on them will involve some comparison of the graphs' components in order to detect matches in patterns. Abstractly this is the famous Graph Isomorphism problem.

The Graph Isomorphism problem is not known to be solvable in polynomial time nor to be NP-complete. However, for certain classes of graphs this problem is known to be solvable in polynomial time with respect to the input size. The Graph Isomorphism problem deals with exact graph matches, whereas the more general problem, the Subgraph Isomorphism, is about partial matchings (Eppstein, 1995).

1.4. The new practical approach

In our case study, the company's database comprises already more than 30,000 projects which are searched through for the users. This bears further challenges in terms of performance, i.e. query times. Computations that must run during query time urge being fast and not overtraining server capacities. To avoid this, we will break down the complexity of the Graph Isomorphism problem by applying first a preprocessing method. Such preprocessing is necessary in order to run the Isomorphism test only on promising pairs of two graphs, thus reducing running times.

Our preprocessing will use several graph invariants. A graph invariant with high discriminating power will produce relatively few cases of pairs of non-isomorphic graphs having the same value. In particular, we consider three graph invariants (degree sequence, status sequence and eigenvalues) and three combinations of them (degree–status sequence, label–degree sequence and label–status sequence). Measures corresponding to simple invariants that can be computed collectively in almost linear time (here, sorting the numbers) could be used to achieve a high degree of discrimination. By using such measures as heuristics we drastically reduce the number of graphs in a collection to be tested against a given graph. Our computational results show that from thousands of graphs in the collection only a few graphs potentially equivalent to the given one remain to be tested with an expensive graph isomorphism algorithm.

Thus far, both isomorphism testing on electric circuit boards and the study of graph invariant discrimination power have been studied independently from each other. We suggest a combined approach, where preprocessing reduces the amount of test graphs and a graph isomorphism algorithm completes the work. Based on graph invariants, a signature of a graph is generated which, later on, supports similarity detection in the search engine. Experiments on retrieving information and running times are presented and evaluated respectively to the potential asset they hold for the company.

1.5. Outline

This paper is structured as follows: In Section 2, we provide historical background and previous research on Graph Isomorphism and preprocessing. Section 3 presents the transformation from the physical electronic board to an abstract level originating from graph theory. Section 4 shows the used algorithm to determine isomorphism and the tested graph invariants for the preprocessing. It also analyzes the discrimination power of individual graph invariants and some combinations of them. Section 5 presents a discussion regarding practical aspects that the results would require in order to achieve feasibility in the company's application. In Section 6 we conclude with a summary of the new results and some directions for further research.

2. Related work and motivation

Retrieving information of graphs has been a vigorously studied topic since the emergence of graph theory. Many practical problems can be described by graphs, and it is clear that graphs play an important role in interrelating data and facilitate efficient computations in any field of science (Singh, 2014). This section reviews classic results and recent work on the Graph Isomorphism problem.

2.1. Algorithmic approaches

The Graph Isomorphism problem has been subject to numerous studies within the last 60 years. Graph Isomorphism problem is one of the few known problems belonging to NP and yet not known to be neither NP-complete nor in P (Read & Corneil, 1977). This is why scientists use a different, polynomially equivalent complexity class to the

Graph Isomorphism: Graph Isomorphism complete problems (Jenner, Köbler, McKenzie, & Torán, 2003).

Its popularity is due to the fact that several research fields can make use of the knowledge about graph similarity: from biological and chemical studies of molecules and their structures, to broadly applicable fields such as data mining.

Some algorithms have been developed to support faster information retrieval from any kind of data, also those represented through graphs. A famous graph algorithm is `subdue`. The graph based knowledge discovery system `subdue` is specialized to find structural patterns in relational data by minimizing the length of graph descriptions (Cook & Holder, 0000). Up to now, practical graph isomorphism algorithms have an exponential upper bound time complexity, although many of them work fine for many graph families. Among the algorithms, the most powerful contemporary algorithm is Brendan McKay's `nauty` package (McKay, 2004). Despite its impressive performance in most cases, there are some families of graphs that force it to run in exponential time (Miyazaki, 1997). The algorithm `nauty` computes automorphism groups of graphs to determine isomorphisms, and it is often cited to be the practically fastest algorithm known so far for testing isomorphism (McKay & Piperno, 2014).

Since 1983, the best known theoretical running time for solving the Graph Isomorphism problem, displacing brute force with $O(n!)$, remained being subexponentially bounded by $exp(O(\sqrt{n \log n}))$ (Babai & Codenotti, 2008), where n denotes the number of vertices. Only recently, Babai announced a quasi polynomial time algorithm for solving Graph Isomorphism problem, bounded by $exp((\log n)^{O(1)})$ (Babai, 2015).

Indeed, the difficulty of the problem prompted researchers to study specific graph classes, exploiting graph families characteristics in order to speed up the process of testing isomorphism. In this direction, Aho et al. presented a polynomial time algorithm to show isomorphism of trees (Aho & Hopcroft, 1974) and Hopcroft and Wong showed an algorithm for the special case of planar graphs, requiring linear time as well (Hopcroft & Wong, 1974). Further results were achieved considering further restricted class of graphs. In the case that the eigenvalues of the adjacency matrix have bounded multiplicity, i.e. they do not occur more often than a certain times, Babai et al. solved the problem in polynomial time (Babai, Grigoryev, & Mount, 1982). Luks' algorithm is designated to solve the Graph Isomorphism problem for graphs with bounded degree (Luks, 1982). However, most of these algorithms, although polynomial in time, are not designed to be implemented or they are not of practical use for their hidden complexity.

To complete the group of polynomially solvable Graph Isomorphism graph classes, Johnson's conjectures (Johnson, 1985) and Bodlaender's work (Bodlaender, 1990) must be mentioned. They state that it can be determined in polynomial time whether two partial k-trees are isomorphic, for a constant k. A partial k-tree describes the feature of a graph in which its tree decomposition does not result in a treewidth larger than k (Bodlaender, 1990). Series–parallel, outerplanar and k-outerplanar graphs are only a selection of graphs being associated with a constant k for their treewidth (Bodlaender, 1990). These results imply that we can check isomorphism in polynomial time on series–parallel graphs that are traditionally derived from electronic circuit boards. Bodlaender lists a theoretic bound on the running time of $O(n^{k+1})$. His outlined algorithm decomposes graphs into their search trees. If and only if two graphs can be decomposed in the same way they are said to be isomorphic. This result is especially interesting for our work, since series–parallel graphs have tree-width value of at most two (Brandstädt, Spinrad, et al., 1999).

2.2. General approaches

Over the decades, different solving techniques emerged to approach the Graph Isomorphism problem. Roughly speaking, they can be split in exact and approximate methods. Approximate methods focus on an optimization problem aiming to maximize either the number of matched edges or vertices (Arvind, Köbler, Kuhnert, & Vasudev, 2012). Exact methods either succeed in matching all vertices and edges or they do not (Gori, Maggini, & Sarti, 2005). In this paper we consider exact matching methods.

Within the field of exact solving approaches, common solving techniques involve canonical labeling. Canonical labeling pursues the idea of relabeling a graph in such a way that isomorphic graphs possess identical labels after the relabeling procedure (Babai & Kucera, 1979).

We should also mention breadth and depth first search. In this direction, Cordella, Foggia, Sansone, and Vento (2004) reported an algorithm that is purely based on depth first search, the so called VF2, whose main advantage is that it is highly efficient in term of space usage. They introduce a data structure that allows acceptable observed running times and space complexity. Indeed many industrial applications seem to make use of the VF2 as some implementations can be found. Characterizing the performance of five algorithms for Graph Isomorphism, Foggia et al. identify VF2 being among the faster solving techniques in all test classes. In particular, processing simulated application data demonstrates an advantage of VF2 against the other tested algorithms: in Foggia, Sansone, and Vento (2001) it is shown its ability to handle certain regularities occurring in real world data better where some solvers reached their limits at the count of a few dozen nodes.

However, all of the aforementioned algorithms have rather inefficient theoretical running times on the worst input instances. Thus, for investigating a large dataset for isomorphisms, intuition aims to reduce the search space significantly, i.e. detect early potential candidates from the set. As shortly introduced before, the method of sorting out potential candidates from a large set is called *preprocessing* and it uses the necessary condition that two isomorphic graphs share identical graph invariants. Unfortunately, none of the so far considered graph invariants is sufficient. If a sufficient graph invariant computable in polynomial time was found, the Graph Isomorphism problem would be solved (Dehmer, Grabner, Mowshowitz, & Emmert-Streib, 2013). Some examples of graph invariants range from simply the number of vertices or edges, over the sorted sequence of occurring degrees in the graph to more complex invariants such as the eigenvalues of the adjacency matrix. They also provide the opportunity to include semantic information which is why they play an important role throughout the following sections.

Other recent approaches to some practical instances of the Graph Isomorphism problem appear in Bonnici, Giugno, Pulvirenti, Shasha, and Ferro (2013), Meissner, Mitea, Luy, and Hedrich (2012), Raymond and Willett (2002), Somkunwar and Moreshwar Vaze (2017), Tabak (2020). However, it is important to note that the classical Graph Isomorphism problem does not completely cover the challenge faced by the company. The application case at hand involves physical circuit boards compounded of electrical components that possess distinguishable designations. This invokes the idea of exploiting the semantic information in addition to the relations reflected by edges in the graphs. This should lead to a reduction of the search space, too. Graphs that contain this additional text information are commonly referred to as *labeled graphs*. Section 3.2, illustrates the graph generation and stresses implications on details and characteristics.

Most approaches to the Graph Isomorphism problem using graph discriminating invariants are due to Dehmer et al. See Dehmer, Chen, Emmert-Streib, Shi, and Tripathi (2018), Dehmer et al. (2013). In a series of papers, Dehmer et al. investigate highly discriminating measures to distinguish graphs (networks) based on their topology. For instance, in Dehmer et al. (2013) the authors present a preprocessing method designed to reduce the amount of computation required to determine whether a given graph is isomorphic to one in a set of graphs. Their method is based on topological graph measures associated with graph invariants, in particular to the notion of degeneracy, which captures the extent to which a measure can discriminate between graphs. As a result, they identified invariants of higher complexity to be more qualified

to reduce the search space substantially. However, their test dataset only consists of graphs with $5 \leq n \leq 9$ which is why their results are representative for rather small graphs. Still, the idea brings promising outlooks concerning their ability to generate signatures, and this paper plans to investigate this further.

From the point of view of isomorphism test on circuit boards, Meissner et al. present an algorithm for Isomorphism testing for a graph-based analog circuit synthesis framework (Meissner et al., 2012). Cordella et al. apply graph matching for the recognition of mechanical components in CAD drawings (Cordella, Foggia, Sansone, & Vento, 2000).

So far, the study of discriminating measures to distinguish graphs, and the isomorphism testing for graphs based on printed circuits have been approached separately. The main contribution of this paper is to combine them: this paper demonstrates the utility of certain graph invariants as a preprocessing method for isomorphism testing of printed circuit boards.

3. Model derivation

It is clear that a search on an electronic circuit board dataset requires a transformation from the physical circuit into an appropriate data structure. Graphs are the modeling bases that we shall use. Although, the actual circuit board strongly resembles a graph itself, there are different ways of translating the board into a graph. This section aims to detail the graph representation of the electronic circuit board.

3.1. Assumptions

We have the following, very generic assumptions for a given collection of graphs representing a collection of printed circuit boards.

1. Same (electronic) components/elements have the same degree of difficulty to be soldered into a system. Furthermore, similar combinations of components are assumed to have a similar level of difficulty.
2. We consider the graphs $G = (V, E)$ representing the printed circuit boards to be simple and undirected.
3. We use both unlabeled and labeled graphs. In particular, for some of our labeled graph datasets, the nodes of the graphs are labeled to identify the use of a specific (electronic) component.
4. All graphs in the dataset are connected.
5. None of the graphs in the dataset (and further graphs generated in this manner from printed circuits) can be assumed planar. This assumption choice will be justified in Section 3.3. The property of series–parallel composition is also excluded from the list of potential graph features.
6. The flow of electric current is neglected and only the relation between components is considered.

Note that the aforementioned assumptions imply that we cannot make use of any of the algorithms discussed in Section 2.1 that run in polynomial time on certain graph classes, since our generated graphs are neither series–parallel nor planar graphs.

3.2. Graphs generation

As a circuit board is a relational representation of several components soldered on a physical board, we use graphs for its representation. Physical components such as transistors, diodes or capacitors are represented as nodes, and the wires conducting electric current from one component to another through the complete circuit are depicted by edges between the nodes. To achieve significant results in the search, it is important to utilize domain specific knowledge whenever it is meaningful (Washio & Motoda, 2003), which is why designations of the physical components are included as well. The combination of different components is what differentiates the functionality of the application. These designations are applied as labels on the nodes according to the graph generation process. This approach differs from many studies in electrical engineering where graph theoretic tools are likewise applied to gain insights about the behavior of different component combinations. In these studies, nodes often represent nets in the circuit, a conductor interconnecting at least two components. The electric components are settled on the edges as links conducting electric current from one net to the other. This creates the desired behavior of voltage in the application. Adding the component's label to the graph to exploit semantic information then implies labeled edges instead of labeled vertices (Berdewad & Deo, 2016).

In this article we choose a different approach than the common practice in electrical engineering. The first reason is to do so is that there is the overall objective in electrical engineering to determine the flow of current. Originating from a power source, electric current flows from net to net until it reaches the power source again. Electrical components are therefore conductors enabling the flow and influencing its behavior with their typical characteristics (Dorf & Svoboda, 2010).

However, in our case of study, recommendations triggered by platform queries shall diversify the results in terms of application or even purpose, yet achievable with similar component combinations. This results from the assumption that same components have the same degree of difficulty to be soldered into a system. Furthermore, similar combinations of components are assumed to have a similar level of difficulty. Next, we provide an example to illustrate what is meant by similarity: consider two projects with an integrated circuit (hereafter IC) built in, one using four pins of the IC, the other eight. These two projects are similar in the sense that both make use of the same component, however in a different manner. Most probably the second project applies the IC with extended functionality. Therefore, both projects are similar, considering the IC, but not identical due to the different pin-allocation. The resulting difference in functionality constitutes the enrichment the company wants, aiming to suggest further projects and ideas to work on which lie within the range of the user's abilities. Therefore, making recommendations is based on the knowledge of the component's connectivity. This connectivity can be well represented by the chosen graph model.

Secondly, electrical engineers care greatly about feasibility of electronic circuits, meaning the unimpeded flow of the current. Yet, electrical feasibility is not a requirement imposed on the mathematical model of the circuit. Accordingly, the approach of characterizing the components' relations in the graph is indeed valid to be considered for solving such a task. Eventually, the main goal of the search is not to identify exact matches but to propose similar projects within milliseconds based on manufacturing difficulty.

3.3. Characteristics of graphs representing printed circuit boards

Looking at the traditional relationship between electrical engineering and graph theory, it can be observed that graphs generated from circuit boards are generally classified by series–parallel graphs. In this case, series and parallel describe methods of interconnecting components in a circuit. Given are two graph components, g_1 and g_2, having each a source and a sink respectively. Connecting the sink of g_1 with the source of g_2 (or vice versa) in a line is denoted *series composition*. It implies that the current flowing through all the components with the same magnitude, whereas, *parallel composition* connects the sources of g_1 and g_2 and the two sinks, respectively. In electrical engineering, the two compositions imply different voltage levels and behavior of the electric current. As seen in the previous section, there are results showing the existence of a polynomial time algorithm to solve the Graph Isomorphism problem on partial k-trees with bounded k (Bodlaender, 1990). Observe that series–parallel graphs belong to this graph category, therefore by Bodlaender (1990) it follows that the Graph Isomorphism problem is solvable on the traditionally derived

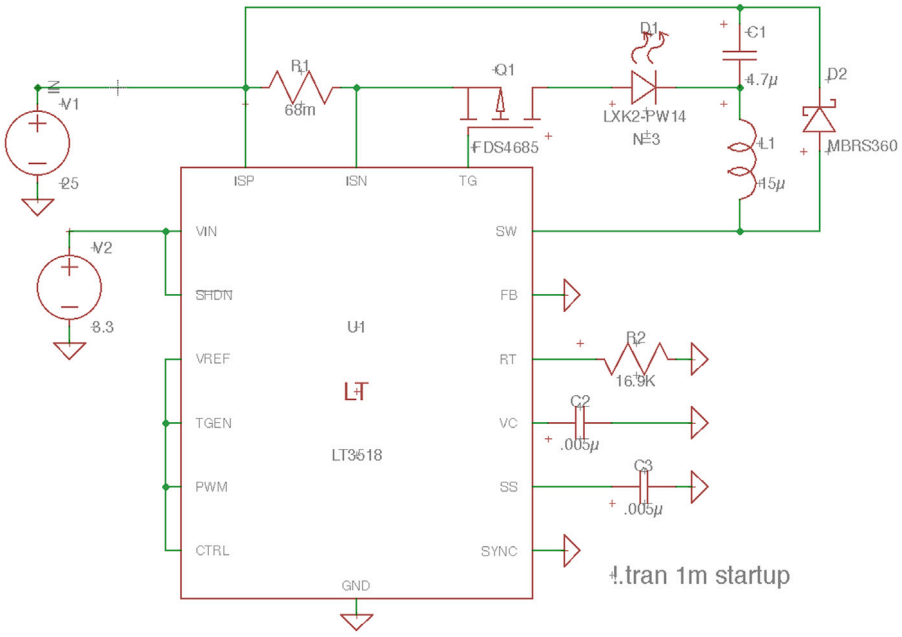

Fig. 1. Schematic figure of a circuit board.

graph models of electronic circuits. However, the following subsections illustrate the limited applicability of this result to our problem.

Fig. 1 shows the schematic representation of a small circuit board to base the discussion about structural implications upon. Nowadays, many components themselves are circuits, the previously mentioned ICs, that contain their separate logic gate, and are therefore rather complex. They possess different pins, electrical connectors, for different purposes. Therefore, it is rather likely that an IC is interconnected with itself on different pins. Adopting this characteristic leads to more edges through self loops in the graph. For the same reason, two components can be multiply interconnected with the IC, which leads to multiple edges in the respective graph representation. In Fig. 1, examples for these are the connections between the four pins of the component $U1$ on the left side, VREF, TGEN, PWM and CTRL, to show 'self-loops'. Furthermore, the connection between the same component and $R1$ via the pins ISP and ISN would be represented by multiple edges. Considering these connections ensure an image of the flow of current through the circuit. Including self-loops and multiple edges, however, entails the loss of the convenient property of being simple without providing information concerning the company's problem statement. Additionally, edges from a vertex to itself are not allowed. As mentioned in Section 3.1, the flow of electric current is neglected and only the relation between components is considered. Therefore, in this work it was decided to omit these edges as well. This ensures the classification as simple graphs and furthermore, reduces the density of the graphs. The potential computation time should therewith decrease for the following operations on the graphs.

Recalling series and parallel compositions, the question arises whether this property holds as well. As shown before, the derived graphs used in previous research were modeled by series–parallel graphs. Note that in our work the interpretation of nodes and edges is swapped. Takamizawa et al. defined series–parallel graphs as a restricted class of planar graphs (Takamizawa, Nishizeki, & Saito, 1982). Intuitively, this is reasonable, since circuit boards are printed on physical planes where crossings of wires have fatal implications on the functionality. However, in the following we show that, in our case, the graph model resulting from the circuit board is not necessarily planar, showing the choice of assumption 5 in Section 3.1. We do it by using the well-known Kuratowski's theorem (Kuratowski, 1930). Referring to Takamizawa, the graphs can therefore not be classified as series–parallel. Kuratowski's theorem states that a finite graph is planar if and only if neither K_5 nor $K_{3,3}$[1] can be created from it by deleting nodes, deleting edges or contracting edges.

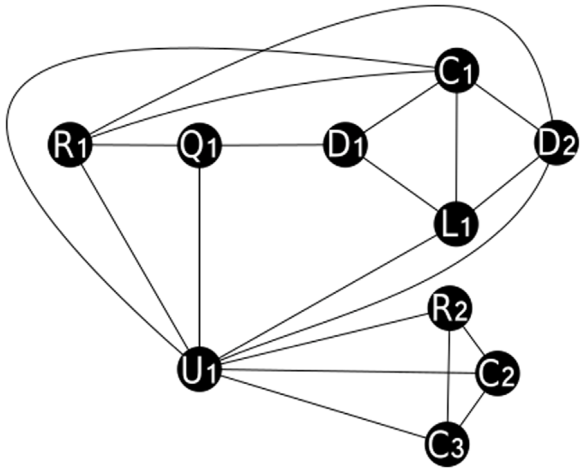

Fig. 2. The circuit board from Fig. 1 represented as a graph.

The graph shown in Fig. 2 depicts the graph theoretical interpretation of the electronic circuit board schematized in Fig. 1. Recall that components are represented by nodes (identified by the corresponding labels) and edges are between nodes if two components are at least once in the same net. These nets are identified via the lighter gray nodes in Fig. 1.

The five graphs in Fig. 3 serve as visual aid to prove that indeed this specific graph representation of circuit boards cannot be assumed planar. The graph in (a) shows the same graph as in Fig. 2, only the vertices R_2, C_2, C_3 and their incident edges are faded and dashed to indicate their deletion when transforming to (b). In the next step, (b), the edge between Q_1 and D_1 is contracted, therefore the two nodes

[1] K_5 denotes the complete graph on five vertices, whereas, $K_{3,3}$ denotes the complete bipartite graph on three vertices at each partition. They are the two smallest non-planar graphs (Kuratowski, 1930).

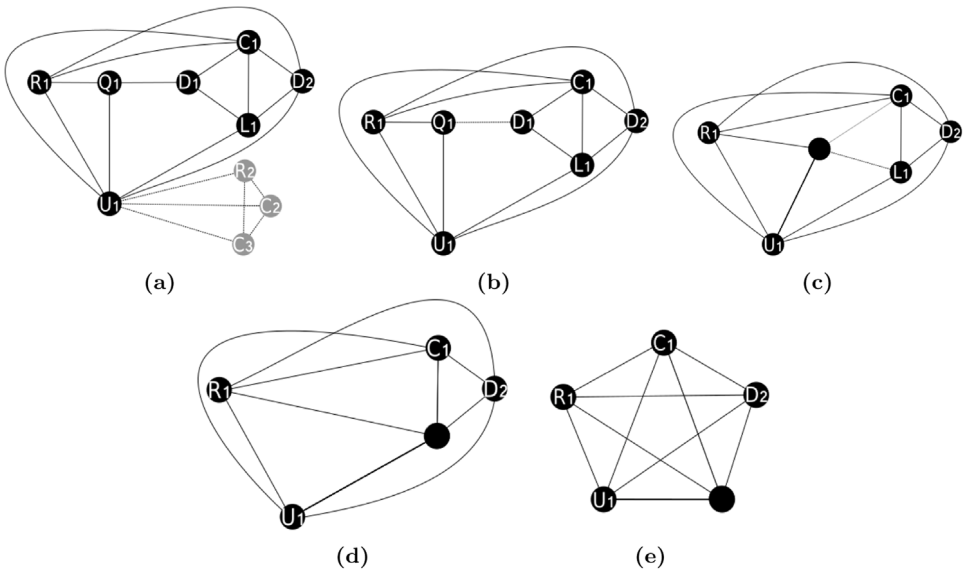

Fig. 3. Applying Kuratowski's theorem (Kuratowski, 1930) to show that the generated graphs are non-planar.

become one, resulting in the unlabeled node in (c). Afterwards the edge between this new node and L_1 is contracted plus the edge between this new node and C_1 is deleted. In (d), the resulting graph is K_5 which is presented in its most common form in (e). Therefore the graph derived from the exemplary circuit board contains K_5 as a subdivision. Following Kuratowski's theorem the graph is non-planar. Therefore, none of the graphs in the dataset and further graphs generated in this manner from printed circuits can be assumed planar.

What causes the loss of these properties (series–parallel composition and planarity) in the generated graphs? Let us justify this phenomenon with the following three points. First, as mentioned before, assigning components to nodes instead of to the nets, drops the idea of series and parallel composition as components do not have a source and a sink. Although components possess at least two pins, they cannot be considered as source and sink since its meaning would then be transferred to the edges as the component is the node. Connecting edges in a series composition can make sense. However, a parallel composition of edges does not. A possibly resulting salient point would be pulled smooth and hence it would not distinguish from a series composition.

This interpretation further implies that stars, as e.g. the composition of D_1, C_1 and L_1 in Fig. 2 becomes triangles. Imagining two more components connected to this net already inevitably creates K_5 as they are all connected to each other through this green node in the schematic in Fig. 1. Through advancing manufacturing technology and growing requirements to the circuit boards, they are nowadays printed on up to 16 layers[2] to avoid wire crossings. On the so called multi-layered circuit boards wires pass off different levels to avoid short circuits. In other words, these graphs are not drawn in a planar way on the physical board anymore. However this does not necessarily imply that they are non-planar as these additional layers are also introduced to reduce the size of a circuit board. This leaves the question if the statement of planar series–parallel graphs still holds for modern circuits or if planarity got lost with the introduction of multiple layers considering the traditional graph representation of electronic circuits. In this research we assume that the properties of series–parallel composition and planarity can be excluded from the list of potential graph features. To sum up, in this research we cannot use any of the algorithms mentioned in Section 2.1 that run in polynomial time on certain graph classes, since our generated graphs are neither series[-parallel nor planar graphs.

[2] https://learn.sparkfun.com/tutorials/pcb-basics.

Note also that our graphs consist of exactly one connected component. Connectivity comes from the physical fact that the circuit is at least connected via ground (labeled as GND or wires terminating red triangle in Fig. 1). Furthermore, as mentioned in Section 3.1 (assumption 2), in order to reduce the number of edges in the graph, multiedges are left out. Therefore all graphs have minimum degree one.

3.4. Implications. Non-labeled and extended graphs generation

In the previous section we saw that the chosen approach – interpreting the nodes and edges differently – bears several implications that need to be considered. Yet, since our primary goal is not to efficiently solve the Graph Isomorphism problem on these graphs, they are not necessarily disadvantageous. Indeed, as electrical components are identified as being the key to determine similarity and there are algorithms to match nodes onto each other, the Graph Isomorphism problem can still be tackled for our case.

In this section, we consider non-labeled and extended graphs. Instead of considering the labels as strings, they are assigned to a number of hanging vertices. We call a vertex a *hanging vertex* if it has a degree of one, meaning, exactly one incident edge. Such hanging vertices are assigned as a number to the label, appended to each node. Therefore, existing, general solving techniques can be applied to our graph database.

Next, we analyze the resulting growth of graphs in our database caused by the extension with hanging vertices. Consider a graph $G = (V, E)$ with $n = |V|$ vertices. In the worst case, G contains n distinct label types. Adding hanging vertices means to append an incident edge to the current node that connects it with a newly created node, not containing any information itself. That is, only the size of the complete group of hanging vertices that is appended to a node represents the label. To assign a number of hanging vertices to a label, they are counted throughout the graph. These counts are sorted in descending order and the most frequently occurring label is assigned to the number one. The next label is assigned to the number two and so on, iterating over the complete count list to decrease the growth of the present dataset as much as possible. The least occurring label is assigned to n hanging vertices. In the worst case, all labels are present in the graph, wherefore $\frac{n(n-1)}{2}$ hanging vertices are added. Leading to a total number $\frac{n^2+n}{2}$ vertices in the graph. This quadratic growth carries further when computing the Graph Isomorphism tests. In the theoretical best case, all vertices have the same label which leads to a linear growth of n hanging vertices.

The company's requirements hold another issue that we have to take into account: electrical components are produced by many different suppliers, and each single supplier uses individualized patterns of nomenclature. Therefore, the issue is to deal with supplier overlapping projects following no consistent component's naming. Therefore, the complete component names are only comparable if all boards were assembled with the components of only one supplier. A diode, e.g., might be called D_x as is Fig. 1, yet other projects might call the very same component LED_x. To overcome this problem, a component type identifier was implemented. That means, instead of comparing the exact component labels, only the type of a component is considered. Part types such as resistors, oscillators, switches and many more are used. Thereby, details such as the magnitude of resistance or capacitance are neglected, and only the combination of component types is considered. As the graph representation is meant to depict abstract relations, this does not imply any drawback.

This is different with pin designation. If the goal was to detect similar functionalities of different circuit boards, this information would be key. As introduced before, ICs have pins for different purposes and connecting a component to one pin generally results in a different functionality than connecting it to another pin. The pin nomenclature is, however, even less standardized. Therefore, this information is not included in our study. However as mentioned before, this is not a drawback as we aim to enrich the search results also by different functionalities.

4. New results

In this section we start by providing basic statistics of the graph dataset (containing 101 graphs) generated following the description from Section 3.2. We then show the outputs of the used heuristic consisting of Graph Isomorphism and preprocessing tests.

4.1. The graph dataset

The experiments are conducted on 100 samples randomly drawn from the company's database which contains more than 30,000 projects, both finished and unfinished.

The main goal of this article consists of establishing a working solution for the general case of a feasible electronic circuit. In order to test the designs for feasibility, a third party software tool was used to detect formal errors.[3]

The randomly drawn circuit boards translate into graphs with sizes ranging from 2 to 170 nodes. Fig. 4 illustrates the spread of number of nodes, the number of edges, the degrees of the vertices, the diameters and the densities of the graphs. Note that this includes five of the graph invariants that we consider in this work. The box-and-whisker-plots present outliers as data points being further than 1.5 times from the upper or lower quartile. The following paragraphs elaborate in more details about the findings from Fig. 4.

In the first row, the number of nodes and edges are plotted according to their occurrences in the dataset. The smallest graphs possess solely two nodes, whereas the other extreme sizes up to 170 nodes. Furthermore, the nodes plot reveals that 75% of the graphs in the dataset possess less than 100 nodes, however more than 50% contain more than 20 nodes. Edges range even further, between one and 4389. Nevertheless, 75% of the graphs have 335 edges or less and 50% even less with 97 edges. This reveals a strong skew in the distribution with outliers containing up to 45 times more edges. Looking at Fig. 5, we see that the growth in the number of edges appears to be exponentially related to increasing number of nodes. The top right plot, with a logarithmic scale in terms of edges, supports this statement. The bottom left plot presents the degree distribution in the sample set. Varying from one to 121 incident edges to a node, half of the nodes have a degree of 16 or less. The middle plot in Fig. 4 shows the sample distribution of the diameter of the graphs. Recall that the diameter of a graph is the path with the greatest length of all shortest paths connecting any pair of vertices in the graph. Therewith, it measures the graph's expansion. In the present dataset, the diameter lies at 79 edges on average. Looking at Fig. 6, there might be a linear trend in the correlation between the number of edges and the diameter of a graph. Which, in return, might result in an increasing complexity with a growing number of edges. However, the amount of data points is not sufficient to draw this conclusion. Furthermore, a path is an intuitive counterexample as it possesses a respectively large diameter and yet no complex structure. In the end, the measured diameters accumulate rather in the bottom of the diagram indicating a greater connectivity, i.e. density. However, directly relating these graph invariants is difficult in terms of magnitudes.

The last measure chosen to analyze the dataset and estimate its computational complexity determines the density of a graph. The density associates the number of nodes with the number of edges to generate an index for connectivity, and it is computed as $\text{density}(G) = \frac{2m}{n(n-1)}$. This index ranges from 0 to 1, where 0 indicates a completely disconnected graph with no edges. Oppositely, 1 represents a fully connected graph with all nodes connected to one another (Coleman & Moré, 1983).

The above analysis of the descriptive statistical information derived from the dataset provides a general overview about the graphs considered during this study. From this we can conclude that the graphs differ very much in terms of node size. Especially, they can become rather large depending on the number of components built in a project. Furthermore, from the plots in Fig. 4, no probability distribution can be derived for any of the graph invariants. Each box-and-whiskers-plot expresses a disperse and a strongly varying characteristic in the dataset. In return therefore, these mentioned characteristics might bear strong discrimination power and thus are considered in the signature generation process, as we will discuss in the next subsection. From Fig. 4 we can conclude that the graphs do not present randomly distributed characteristics, which makes sense as the underlying electronic circuit boards are not created randomly. Note that Fig. 4 elucidates the varying levels of complexity that the company has to deal with. Considering edges and degrees, the complexity increases as the number of incident edges is a suspension point to check for potential matching. On a fixed number of nodes, a growing number of edges leads to higher degrees and leads to more combinations needing to be considered to compute both Graph Isomorphism and the preprocessing. Therewith, the complexity becomes visible in different dimensions where the most important ones are the size in terms of number of nodes and the density as it reflects the combinatorial difficulty in matching two graphs of the same connectedness. Observing high values in both measurements make very efficient computation methods indispensable to avoid drastically increasing processing times.

4.2. Tests of Graph isomorphism

4.2.1. The VF2 algorithm

The chosen method to test the above instances is the aforementioned Graph Isomorphism exact algorithm VF2 (Cordella, Sansone, & Vento, 1999). This algorithm, introduced by Cordella et al. is claimed to be efficient on large graphs and even more advantageous on smaller graphs due to an innovative data structure to keep track of the partial graph matches. The problem that Cordella et al. wanted to tackle is running out of memory when testing large graphs due to numerous combinatorial possibilities of nodes to match. Their algorithm initially solved the problem for directed graphs. Nevertheless, it can be trivially extended to undirected graphs. The algorithm is based on the idea of

[3] The open source tool CadSoft EAGLE v7 (a Printed Circuit Board designing tool) is the above mentioned third party software product. Errors caused by for example short circuits and insufficient pin-allocations can be easily identified [https://cadsoft.io/].

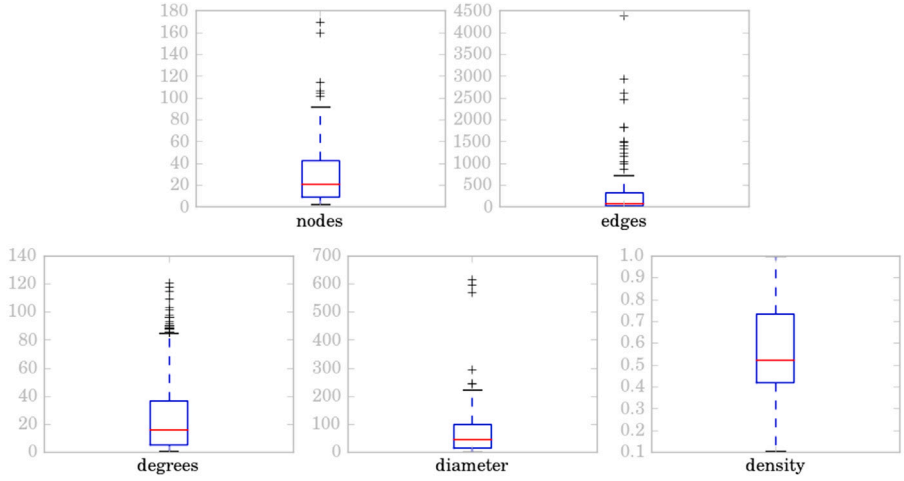

Fig. 4. Overview of the descriptive statistics of the graph dataset generated as described in Section 3.2.

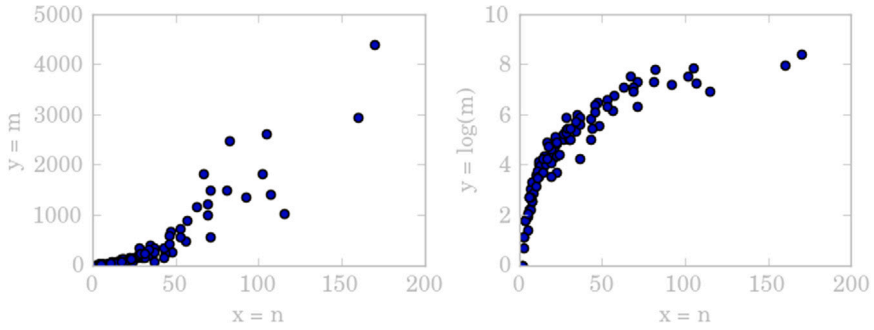

Fig. 5. Depiction of the relation between nodes and edges in the graph dataset. Both plots show the exponential growth of edges considering the increasing number of nodes in a graph.

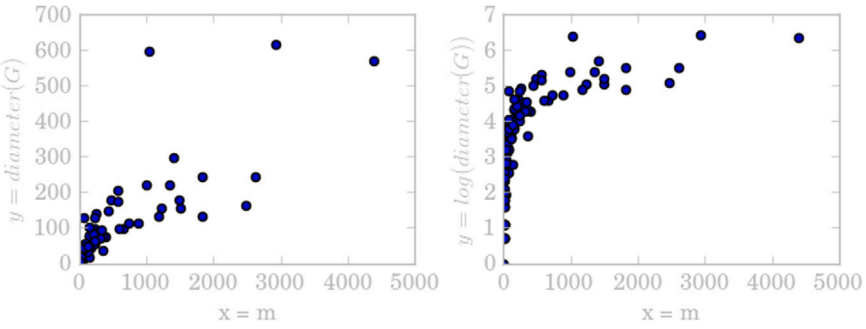

Fig. 6. Relation between the number of edges in a graph and its diameter being the longest of all shortest paths in a graph.

constructing a decision tree representing state spaces as its nodes. A state is determined by the partial mapping it contains which is only a subset of the included components. To attain the next state or prune the branch a Boolean feasibility function determines whether a pair (n_1, n_2), where $n_1 \in G_1$ and $n_2 \in G_2$, can be added to the partial mapping or not. Then, either a new state is created or the complete branch is pruned respectively. The algorithm terminates once a mapping is found or it is proven that no mapping exists. It proceeds as a depth-first-search to potentially speed up the finding of an isomorphism. On the other hand, depth-first-search will need to process the complete tree in order to be certain about the nonexistence of Isomorphism. The advantage of this approach is that both Graph Isomorphism and Subgraph Isomorphism can be solved for with merely one algorithm. In the VF2 algorithm, we denote by N to the number of nodes of the biggest graph, i.e. $N = \max(n_1, n_2)$.

Furthermore, the VF2 algorithm is extendable to include attributes on nodes or relations as well. Thus, the aforementioned label types are easy to include. However, as mentioned before, the actual contribution of this algorithm is to being able to process varying graph sizes efficiently which is necessary for the present dataset, as seen in Section 4.1. This is achieved by storing matchings in vectors for each graph and the adjacency matrices available among all the states (Cordella et al., 2004). Thereby, a storage requirement, proportionate to the number of nodes of the two graphs, is achieved. Effectively, the constant amount of memory and the depth-first-search ensure a memory requirement of $O(N)$, with a small constant factor. This is a major improvement compared to other algorithms, like their previous release with $O(N^2)$ or Ullmann's algorithm with $O(N^3)$. Not only larger graphs can be processed due to this advantage, but also smaller graphs can benefit from the usage of more cache memory (Cordella et al., 2004).

Table 1
Number of isomorphic graph pairs found in the datasets.

Dataset	Original graphs without labels (a)	Original graphs with semantic labels (b)	Extended graphs (hanging vertices) (c)
Number of isomorphic graphs found	10	1	1
Processing time [s] (complete set)	131,730	136,890	160,164

4.2.2. Output of isomorphism tests using the VF2 algorithm

The results of the Graph Isomorphism and preprocessing tests from this section have been obtained using the `networkx.algorithm` package from Python.

The Graph Isomorphism test was conducted on three different versions of the dataset. Set (a) contains the graphs constructed as described in Section 3.2, yet omitting any information about electrical component types, thus a set of unlabeled graphs. Set (b) and (c) are different from each other in the way they comprise the information of component types. Set (b) is provided with the semantic and therefore, represent labeled graphs which are labeled on their nodes. Set (c) embodies the experiment with hanging vertices that was introduced in Section 3.4.

It is worth noting, as mentioned in Section 3.4, that standardization in component's designation is hardly given. Therefore, comparability is achieved by using the type of electrical component concluded from the suppliers exact designations. This approach neglects specific component details that are needed for assembling, such as the magnitude of the required resistance. Nevertheless as argued before, this abstraction level is assumed to be sufficient for providing the user with qualified recommendations on what to pursue next. We are able to distinguish between 34 commonly used component types which can be determined by certain patterns. An additional type is assigned to all components that either are of another type or cannot be identified by the provided pattern.

Furthermore, having a graph with existent hanging vertices requires the differentiation between the component hanging vertex and the hanging vertex connoted to a label type. Since all graph nodes are fully labeled, it turns out to be sufficient starting the number assignment with two instead of one. Therefore, maximally 36 distinct attributes can be assigned to the nodes of the graphs. This shall provide two aspects: it decreases the worst case growth as described in Section 3.4 and also compromises between exactness and randomness which adds to the diversification of the search and especially its results.

The results from the Graph Isomorphism tests on the complete dataset are presented in Table 1. The columns refer to the differently attributed datasets as described before by (a), (b) and (c). Furthermore, the average time over 100 experiments is given in seconds that was needed to process all 5050 possible combinations within the dataset. As expected, in the more general set which is not provided with further information through attributes, more isomorphisms can be detected than in the other sets which include more details. The fact that ten times as many unlabeled graphs are determined being isomorphic than labeled graphs suggests that the general construction and structure of a circuit board is rather similar once they have the same amount of nodes. This makes sense, as the graphs are all derived from physical circuit boards which have to fulfill certain criteria to guarantee the flow of electric current. These requirements are mirrored in the structure of the circuit board which, in turn, becomes the same when translating the schematic to the model as described in Section 3.2. On the other hand, it is surprising that the unlabeled set, which generally includes more potential matches, is processed faster than the other sets. Apparently the algorithm performs quite well on the easier set as it detects isomorphism rather early.

Looking at the results concerning sets (b) and (c), the number of isomorphic graphs is, as expected, the same, since they exploit the same information merely in different forms. Therefore, it is confirmed that the dataset contains one pair of isomorphic circuit boards in the sense described throughout this paper so far. A check of the actual schemes of the boards concludes that they are indeed only different by the specifics of the components. Yet, the processing time of the extended graphs exceeds the one of the semantic labels by almost 20%. Therefore, the growth of the graphs seems to predominate the advantage of comparing merely relations of nodes. Although the growth as discussed in Section 3.4 reflects the worst case with distinct labels for every node, an even smaller amount of added vertices has the potential to blow up the graphs in size, too. This can be observed in the difference of resulting processing times between the sets (a) and (b).

The experiment of applying hanging vertices can be assessed successful and insightful. It produces valid graphs that contain encrypted information about node attributes. This transformation's beauty lies in the graph based encryption which makes them processable by any general graph processor, certainly not limited to general Graph Isomorphism solvers. However, as it is possible to take advantage of attributes in their semantics, this approach presented itself as being more advantageous. Firstly, since already large graphs are not further expanded. This reduces processing times. Secondly, the set (b) is understandable by human beings without the need of decoding tables or the like.

Lastly to note, despite the immense potential growth induced by hanging vertices, the VF2 algorithm performs fast on these large graphs. There might be two major explanations contributing to that outcome. Firstly, the graphs are very different in size, as seen in Fig. 4, which shows that the number of nodes of 50% of the graphs ranges between 2 and 96. With a big difference in the number of nodes, the isomorphism between two graphs can be excluded rather early in the process.[4] The theoretical upper bound on running time is $O(max(n_1, n_2))$.

Secondly, referring to the discussion from Section 3.4, the growth of the graphs is as minimized as possible by assigning the least occurring label types to a high number of hanging vertices. The most occurring label types are assigned to a smaller number of hanging vertices. Yet, the effect of appending the vertices resulted in an considerable average graph growth, showing that already a small number of known label types has a large impact on the graph sizes. Progressing in terms of label recognition and the involvement of 30,000+ projects give reasons to expect even bigger growth when hanging vertices are added.

Considering the savings in storage and processing time caused by smaller sizes and more versatile opportunities deriving graph invariants, the option of appending hanging vertices terminates at this point and is not pursued any further within the frame of this research. It was yet a successful experiment, since showed that labeled graphs can be accurately transformed into unlabeled graphs without loss of information.

4.3. Output of preprocessing tests using graph invariants

The following section presents the results of the preprocessing experiments. In particular, we describe the used graph invariants and we show the respective discrimination power on the dataset. Furthermore, the average processing times, needed to compare all 5050 possible graph pairs, is provided.

Running the Graph Isomorphism test throughout the dataset reveals important insights for the following preprocessing. It shows that actually one pair of graphs is isomorphic among the 101 graphs in the set. However, it is problematic that two input graphs are always needed to merely obtain a Boolean indicating whether they are isomorphic or not. Bethinking the original problem to query this information within

[4] Experiments that tested an average instance with 33 nodes with itself without label type information required already 296 secs whereas this same experiment on the instance with 170 nodes was terminated without result after three hours.

Table 2
Number of graph pairs that possess the identical graph invariant and average processing time of the complete dataset in 100 experiments.

Graph invariant	Number of potential graph pairs left	Processing time [s]
Nodes	73	0.003
Edges	22	0.004
Degree sequence	10	0.227
Status sequence	10	6.167
Eigenvalues	10	0.245
Degree–status sequence	10	6.195
Label–degree sequence	1	0.504
Label–status sequence	1	6.249

milliseconds for a vast amount of graphs, this solution proves elusive considering the computation's complexity and the growing amount of data. Therefore, a data format is required to store a graph's characteristics in an easily and fast accessible way. A quick idea might include adjacency matrices or, to reduce storage required, adjacency lists. Even a list storing all isomorphic graphs to a certain graph is imaginable. This might be feasible, however, updating this list is tedious repeating the computations for all the existing graphs paired with the newly added graph. To avoid these expensive computations, it is desirable to generate a certificate or a signature per graph that can be compared easily at query time. Therefore, a signature should be computed while the project is inserted into the database and does not consume a lot of storage. Besides the computational efforts, this approach additionally offers the opportunity to include distance measures evaluating similarity of graphs. That similarity can be the base for user recommendations later on. For this reason, exact matches are not necessary anyway in the broad picture as they detect identical projects, not comparable, related ones.

Originally, preprocessing is used to reduce the search space in a dataset, meaning the reduction of the number of graph pairs to only promising ones. They are determined by the use of some graph invariant. The conducted experiment involves different kinds of graph invariants ranging from very basic ones to new invariants which are more adapted to the problem statement. Therefore, the number of nodes and edges are included as well as the degree sequence, status sequence, a combination of the two, the eigenvalues and two label integrating invariants are tested upon.

The results of the preprocessing tests are presented in Table 2 in terms of number of graph pairs possessing identical invariants, and the average processing time needed to evaluate the complete dataset. The tests are conducted on dataset (b) whereas only the last two invariants exploit the additional information provided through the label types. As expected, the very general graph invariants, the number of nodes and the number of edges, return the highest number of potential pairs. All other invariants, a little more sophisticated, return significantly less. Notably is the result of the degree–status sequence. Although both degree and status individually return ten pairs, the combination of the two returns the same graph pairs. Instinctively, the combination contains more information and therefore it might be able to exclude further pairs from the potential list. However, it is not the case in the present dataset.

The two graph invariants exploiting the extra information given by the labels return merely the graph pair that was earlier, in Section 4.2.2, identified as being isomorphic. Their discrimination power therefore appears to be very strong on the given dataset. It is also worth noting that all graph invariants are quickly computable, concluding from the majority of average times that process the complete dataset in less than a second. Considering the effort to run the Graph Isomorphism test on the potential graph pairs time-wise, it goes down by at least 95% using any of the listed preprocessing invariants. However, the experiments show great discrimination power of different graph invariants on the dataset that come close to the Graph Isomorphism results and, more importantly, produce the required signature for each graph. Therefore, the experiments conclude successful which can be shown with the help of Section 4.2.2, knowing about the exact Isomorphism results.

After revealing the results of the conducted experiments, the next section presents the discussion and the embedding into the solution method of the initial problem statement.

5. Discussion

This section discusses and compares the results of the two heuristic methods that we propose to use to distinguish graphs coming from printed circuit boards: the Graph Isomorphism test using the VF2 algorithm, and the preprocessing test using several graph invariants. Tables 1 and 2 display the average time (in seconds) required to process the complete dataset of 101 graphs. Notably, all preprocessing variations run significantly faster than the Graph Isomorphism solver. This probably follows from the fact that the VF2 algorithm needs to process 5050 combinations, whereas the graph invariants are computed for 101 graphs individually. This reflects the great potential in saving computational efforts, thus required processing time. Therewith, inserting a new element into the dataset causes linear additional computations concerning the Graph Isomorphism test. The linear growth refers to the number of entries already existent in the dataset. On the other hand, when applying graph invariants, the solution handles the inclusion of a new project in constant time concerning the number of existing projects, with factor one. Table 3 contrasts the complexities of the theoretical run time of each procedure being investigated. As before, the complexities are based on the size of the input graph (n denotes the number of nodes). Observe that all time bounds described in Table 3 hold for general graphs, and recall that such bounds can be lowered for certain graph classes, as described in Section 2.1.

Comparing the realized processing times of Table 2 with the complexities described in Table 3, the different order of complexity classes becomes consistently visible. The eigenvalue computations, having the complexity of cubic order, are surprisingly fast considering the linear order of complexity of the degree sequence computations, and the similar observed processing times. Furthermore, compared to the graph invariants, which include the additional factor of $\log n$, the difference is quite large. This is explained by the fact that this factor's contribution increases drastically with growing n. However, these are solely upper bounds and cannot be directly connected to absolute running times. And yet, they indicate the difficulty of a computation that certainly is reflected in the time required.

Furthermore, Table 3 reveals how the relatively fast isomorphism times are achievable. For all procedures involving merely one graph (graph invariant computations) n denotes the number of nodes. In the VF2 algorithm N denotes the number of nodes of the biggest graph, i.e. $N = \max(n_1, n_2)$. The upper bounds were determined by analyzing our own code and available documentations. In the best case, having strongly varying input sizes or degree sequences, the upper bound on the running time can be reduced from factorial to merely quadratic order of complexity. The box-and-whiskers-plots in Fig. 4 in Section 4.1 depict exactly that best case for most of the combinations,[5] having diverse sizes of the graphs throughout the set and more varying degrees.

Applying the VF2 algorithm provides fast insights about isomorphic and non-isomorphic graph pairs in a dataset based on real world data. This supports the validation of the preprocessing outcomes in terms of the discriminating power of each graph invariant on the presented graph models. Additionally, the respective processing times given in Table 2 show their feasibility in a real application. Although eventually,

[5] This explains as well why the Graph Isomorphism test of an instance with itself, mentioned in an earlier footnote, endured much longer, as this depicts the worst case.

Table 3
Comparison of the worst case upper bounds on the time complexities of all applied procedures.

Graph invariant/VF2	Theoretical time complexity considering the worst case [best case]
Nodes	$O(1)$
Edges	$O(1)$
Degree sequence	$O(n)$
Status sequence	$O(n^3 log n)$
Eigenvalues	$O(n^3)$
Degree–status sequence	$O(n^3 log n)$
Label-degree sequence	$O(n)$
Label–status sequence	$O(n^3 log n)$
VF2	$O(N!N)$ $[O(N^2)]$

absolute processing times do not matter in terms of seconds or minutes since the signatures are computed in an offline manner and stored in the database. Later, they are requested at query time which is the requiredly fast process.

Considering the fact that the number of potential graph pairs comes out of a pool of 5050 possible combinations, all graph invariants show high discrimination power. Even considering the number of nodes, which is very basic and returns the highest number of candidates, sorts out the combinations leaving only a set as big as 1.4% of the original set. This is due to the graph diversity in the used dataset. However, the outcome of the two label inclusive invariants, returning only the isomorphic pair, shows even stronger discrimination power.

Naturally, in order to determine which signature is more appropriate to solve the problem at hand, different aspects can be considered. In fact, all of them have the quality of being integrable in a search platform like Solr or Elasticsearch. However, the choice for the present study falls on the label-degree sequence. Including the most important factor predetermined in Section 3.2, where it was argued that the amount of connected components to a certain component is a valid indicator for difficulty. This amount is directly reflected by the degree and captured in the chosen graph invariant. Moreover, it is paired with the component type and therefore the realization of what was requested in the beginning. Human readability, mentioned in Section 4.2.2, is also provided by including semantics describing the nodes. Assuming that the greater part of the components in the electronic schemes is assigned to interpretable designations, this invariant almost represents a so called netlist. This netlist gives an overview of what components are connected in which net. Therefore, this information is anyway needed and available. Furthermore, the complexity of linear running time to compute the invariant is very desirable and allows easily the inclusion of big hardware projects such as mainboards of a computer system or a robot's control system.

For the aforementioned reasons, the label-degree sequence seems like an adequate graph invariant to be included in the company's search engine. To do so, a number of possibilities open up. All the experiments presented in Section 4.3 are exact matches, although this restriction will reasonably loosened in the application.

6. Concluding remarks

How to enrich a search engine by a specific kind of data, namely electronic circuit boards? This is the question that this paper investigated. First, we transformed electrical circuits into an apposite graph representation. Such generated graphs were tested for Graph Isomorphism (using the VF2 algorithm) with different preprocessing strategies (using graph invariants). The graph invariant choice for the preprocessing is the label-degree sequence. Decisive factors for this choice are, firstly, the ability of providing signatures that can be accessed and compared with each other at query time. Secondly, it sums up the abstraction level that the company defined to determine soldering difficulty, especially, comparing different projects.

Looking at the realized running times of the VF2 algorithm, we obtain an interesting remark. $O(n!n)$ upper bounds the running time in the worst case, a genuinely undesirable running time. However, the processing times reported in Section 4.2.2 are reasonable. This implies that there are algorithms that perform practically well although they possess poor theoretic running times. Being independent of this phenomenon, the signature generation excels as running times become rather irrelevant. Existing algorithms and the introduced data structure execute the instantaneous access and ameliorate user experience. In fact, as seen in Section 3.2, our approach is very general and can thus be employed for further applications where graphs are used to determine similarity of objects.

To sum up, this work contributes to understand the practical application of solving the Graph Isomorphism problem. Furthermore, it performs research on preprocessing strategies on special graphs, namely node attributed graphs generated from electronic circuits, and shows the effectiveness of exploiting this information. It is furthermore outlined how this academic solution approach can be implemented in a real world application to solve related problems. Our results show the extensibility of basic search frameworks to integrate company specific, physical and relational knowledge and therefore, enhance the search results' relevance.

This research is also backing the company to improve its search performance. As mostly basic search functionalities provided by common frameworks are implemented, the company's new search engine will incorporate elemental text-base and industry specific knowledge, improving their data's accessibility.

Further research could look for an appropriate measurement of distance between the electronic circuits. It would also be interesting to study whether even entire functionalities can be predicted merely ensuing from the schematics. This could include, for instance, considering pins, as it was shortly addressed in Section 3.3. Furthermore, future research could take labeled edges into account (thus extending assumption 3 in Section 3.1) following the argumentation from Section 3.2 that electrical engineers switch the meaning of nodes and edges to determine feasibility. Also, the results in Section 3.3 regarding the special characteristics to describe the graphs coming from electric circuit boards could be studied in more detail. Lastly, other properties of graphs coming from electric circuit boards, like treewidth, could be examined in the future, since there exist polynomial time algorithms to solve the Graph Isomorphism on such graph classes holding certain properties.

CRediT authorship contribution statement

Aida Abiad: Supervision, Writing - original draft. **Alexander Grigoriev:** Supervision, Writing - original draft. **Stefanie Niemzok:** Investigation, Data curation, Software.

References

Aho, A. V., & Hopcroft, J. E. (1974). *The design and analysis of computer algorithms*. Pearson Education India.

Arvind, V., Köbler, J., Kuhnert, S., & Vasudev, Y. (2012). Approximate graph isomorphism. In *LNCS, MFCS'2012* (pp. 100–111). Springer.

Babai, L. (2015). Graph isomorphism in quasipolynomial time. arXiv preprint arXiv: 1512.03547.

Babai, L., & Codenotti, P. (2008). Isomorhism of hypergraphs of low rank in moderately exponential time. In *FOCS'08* (pp. 667–676). IEEE.

Babai, L., Grigoryev, D. Y., & Mount, D. M. (1982). Isomorphism of graphs with bounded eigenvalue multiplicity. In *STOC'82* (pp. 310–324). ACM.

Babai, L., & Kucera, L. (1979). Canonical labelling of graphs in linear average time. In *FOCS'79* (pp. 39–46). IEEE.

Berdewad, O., & Deo, S. (2016). Application of graph theory in electrical network. *International Journal of Science and Research*, *5*(3), 981–982.

Bodlaender, H. L. (1990). Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees. *Journal of Algorithms*, *11*(4), 631–643.

Bonnici, V., Giugno, R., Pulvirenti, A., Shasha, D., & Ferro, A. (2013). A subgraph isomorphism algorithm and its application to biochemical data. *2013 BMC Bioinformatics*, *14*.

Brandstädt, A., Spinrad, J. P., et al. (1999). *Graph classes: A survey, Vol. 3*. SIAM.

Coleman, T. F., & Moré, J. J. (1983). Estimation of sparse Jacobian matrices and graph coloring problems. *SIAM Journal on Numerical Analysis*, *20*(1), 187–209.

Cook, D., & Holder, L. (0000). SUBDUE – Graph Based Knowledge Discovery. http://ailab.wsu.edu/subdue/.

Cordella, L. P., Foggia, P., Sansone, C., & Vento, M. (2000). Fast graph matching for detecting CAD image components. In *Pattern Recognition'2000, vol. 2* (pp. 1034–1037). IEEE.

Cordella, L. P., Foggia, P., Sansone, C., & Vento, M. (2004). A (sub) graph isomorphism algorithm for matching large graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *26*(10), 1367–1372.

Cordella, P., Sansone, C., & Vento, M. (1999). Performance evaluation of the VF graph matching algorithm. In *ICIAP'99* (pp. 1172–1177).

Dehmer, M., Chen, Z., Emmert-Streib, F., Shi, Y., & Tripathi, S. (2018). Graph measures with high discrimination power revisited: A random polynomial approach. *Information Sciences*, *467*, 407–414.

Dehmer, M., Grabner, M., Mowshowitz, A., & Emmert-Streib, F. (2013). An efficient heuristic approach to detecting graph isomorphism based on combinations of highly discriminating invariants. *Advances in Computational Mathematics*, *39*(2), 311–325.

Dorf, R. C., & Svoboda, J. A. (2010). *Introduction to electric circuits*. John Wiley & Sons.

Eppstein, D. (1995). Subgraph isomorphism in planar graphs and related problems. In *SODA'95, vol. 95* (pp. 632–640). ACM-SIAM.

Foggia, P., Sansone, C., & Vento, M. (2001). A performance comparison of five algorithms for graph isomorphism. In *Proceedings of the 3rd IAPR TC-15 workshop on graph-based representations in pattern recognition (pp. 188–199)*.

Gori, M., Maggini, M., & Sarti, L. (2005). Exact and approximate graph matching using random walks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *27*(7), 1100–1111.

Hopcroft, J. E., & Wong, J.-K. (1974). Linear time algorithm for isomorphism of planar graphs (preliminary report). In *STOC'74* (pp. 172–184). ACM.

Jenner, B., Köbler, J., McKenzie, P., & Torán, J. (2003). Completeness results for graph isomorphism. *Journal of Computer and System Sciences*, *66*(3), 549–566.

Johnson, D. S. (1985). The NP-completeness column: an ongoing guide. *Journal of Algorithms*, *6*(3), 434–451.

Kuratowski, C. (1930). Sur le probleme des courbes gauches en topologie. *Fundamenta Mathematicae*, *15*(1), 271–283.

Kwon, Y., Omitaomu, O. A., & Wang, G.-N. (2008). Data mining approaches for modeling complex electronic circuit design activities. *Computers & Industrial Engineering*, *54*(2), 229–241.

Luks, E. M. (1982). Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of Computer and System Sciences*, *25*(1), 42–65.

McKay, B. D. (2004). The nauty page.

McKay, B. D., & Piperno, A. (2014). Practical graph isomorphism, II. *Journal of Symbolic Computation*, *60*, 94–112.

Meissner, M., Mitea, O., Luy, L., & Hedrich, L. (2012). Fast isomorphism testing for a graph-based analog circuit synthesis framework. In *Design, automation & test in europe conference & exhibition (DATE)* (pp. 757–762).

Miyazaki, T. (1997). Complexity of mckay's canonical labeling algorithm. In *DIMACS series in discrete mathematics and theoretical computer science: Vol. 28*, (pp. 239–256). American Mathematical Society.

Raymond, J., & Willett, P. (2002). Maximum common subgraph isomorphism algorithms for the matching of chemical structures. *Journal of Computer-Aided Molecular Design*, *16*, 521–533.

Read, R. C., & Corneil, D. G. (1977). The graph isomorphism disease. *Journal of Graph Theory*, *1*(4), 339–363.

Singh, R. P. (2014). Application of graph theory in computer science and engineering. *International Journal of Computer Applications*, *104*(1), 10–13.

Somkunwar, R., & Moreshwar Vaze, V. (2017). Comparative study of graph isomorphism applications. *International Journal of Computer Applications*, *162*(7), 34–37.

Tabak, P. (2020). Distance based canonical labeling algorithms with applications to graph matching.

Takamizawa, K., Nishizeki, T., & Saito, N. (1982). Linear-time computability of combinatorial problems on series-parallel graphs. *Journal of the ACM*, *29*(3), 623–641.

Washio, T., & Motoda, H. (2003). State of the art of graph-based data mining. *Acm Sigkdd Explorations Newsletter*, *5*(1), 59–68.