Contents lists available at ScienceDirect

Discrete Applied Mathematics

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

VF2++—An improved subgraph isomorphism algorithm

Alpár Jüttner [a,b,*], Péter Madarasi [b]

[a] MTA-ELTE Egerváry Research Group, Budapest, Hungary
[b] Department of Operations Research, Eötvös Loránd University, Budapest, Hungary

ARTICLE INFO

Article history:
Received 30 November 2016
Received in revised form 20 January 2018
Accepted 23 February 2018
Available online xxxx

Keywords:
Computational biology
Subgraph isomorphism problem

ABSTRACT

This paper presents a largely improved version of the VF2 algorithm for the *Subgraph Isomorphism Problem*. The improvements are twofold. Firstly, it is based on a new approach for determining the matching order of the nodes, and secondly, more efficient — nevertheless easier to compute — cutting rules are proposed. They together reduce the search space significantly.

In addition to the usual *Subgraph Isomorphism Problem*, the paper also presents specialized algorithms for the *Induced Subgraph Isomorphism* and for the *Graph Isomorphism Problems*.

Finally, an extensive experimental evaluation is provided using a wide range of inputs, including both real-life biological and chemical datasets and standard randomly generated graph series. The results show major and consistent running time improvements over the other known methods.

The C++ implementations of the algorithms are available open-source as part of the LEMON graph and network optimization library.

© 2018 Published by Elsevier B.V.

1. Introduction

In the last decades, combinatorial structures, and especially graphs have been considered with ever increasing interest, and applied to the solution of several new and revised questions. The expressiveness, the simplicity and the deep theoretical background of graphs make them one of the most useful modeling tool and appears constantly in several seemingly independent fields, such as bioinformatics and chemistry.

Complex biological systems arise from the interaction and cooperation of plenty of molecular components. Getting acquainted with the structure of such systems at the molecular level is of primary importance, since protein–protein interaction, DNA–protein interaction, metabolic interaction, transcription factor binding, neuronal networks, and hormone signaling networks can be understood this way.

Many chemical and biological structures can easily be modeled as graphs, for instance, a molecular structure can be considered as a graph, whose nodes correspond to atoms and whose edges to chemical bonds. The similarity and dissimilarity of objects corresponding to nodes are incorporated to the model by *node labels*. Understanding such networks basically requires finding specific subgraphs, thus it calls for efficient subgraph isomorphism algorithms.

Other real-world fields related to some variants of subgraph isomorphism include pattern recognition and machine vision [3], symbol recognition [9], and face identification [13].

Subgraph and induced subgraph isomorphism problems are known to be NP-Complete [6], while the graph isomorphism problem is one of the few problems in NP neither known to be in P nor NP-Complete. Although polynomial-time isomorphism

* Corresponding author at: Department of Operations Research, Eötvös Loránd University, Budapest, Hungary.
E-mail addresses: alpar@cs.elte.hu (A. Jüttner), madarasip@caesar.elte.hu (P. Madarasi).

https://doi.org/10.1016/j.dam.2018.02.018
0166-218X/© 2018 Published by Elsevier B.V.

algorithms are known for various graph classes, like trees and planar graphs [11], bounded valence graphs [15], interval graphs [14] or permutation graphs [5]. Furthermore, an FPT algorithm has also been presented for the colored hypergraph isomorphism problem in [1].

In the following, some algorithms which do not need any restrictions on the graphs are summarized. Even though, an overall polynomial behavior is not expectable from such an alternative, they may often have good practical performance, in fact, they might be the best choice in practice even on a graph class for which polynomial algorithm is known.

The first practically usable approach was due to *Ullmann* [20], which is a commonly used algorithm based on depth-first search with a complex heuristic for reducing the number of visited states. A major problem is its $\Theta(n^3)$ space complexity, which makes it impractical in the case of big sparse graphs.

In a recent paper, Ullmann [21] presents an improved version of this algorithm based on a bit-vector solution for the binary Constraint Satisfaction Problem.

The *Nauty* algorithm [16] transforms the two graphs to a canonical form before starting to look for an isomorphism. It has been considered as one of the fastest graph isomorphism algorithms, although graph categories were shown in which it takes exponentially many steps. This algorithm handles only the graph isomorphism problem.

The *LAD* algorithm [19] uses a depth-first search strategy and formulates the isomorphism as a Constraint Satisfaction Problem to prune the search tree. The constraints are that the mapping has to be one-to-one and edge-preserving, hence it is possible to handle new isomorphism types as well.

The *RI* algorithm [2] and its variations are based on a state space representation. After reordering the nodes of the graphs, it uses some fast executable heuristic checks without using any complex pruning rules. It seems to run really efficiently on graphs coming from biology, and won the International Contest on Pattern Search in Biological Databases [22].

Currently, the most commonly used algorithm is the *VF2* [8], an improved version of *VF* [7], which was designed for solving pattern matching and computer vision problems, and has been one of the best overall algorithms for more than a decade. Although, it is not as fast as some of the new specialized algorithms, it is still widely used due to its simplicity and space efficiency. VF2 uses a state space representation and checks specific conditions in each state to prune the search tree.

Meanwhile, another variant called *VF2 Plus* [4] has been published. It is considered to be as efficient as the RI algorithm and has a strictly better behavior on large graphs. The main idea of VF2 Plus is to precompute a heuristic node order of the graph to be embedded, on which VF2 works more efficiently.

This paper introduces *VF2++*, a new further improved algorithm for the graph and (induced) subgraph isomorphism problems. It is based on efficient cutting rules and determines a node order in which VF2 runs significantly faster on practical inputs.

The rest of the paper is structured as follows. Section 2 defines the exact problems to be solved, Section 3 provides a description of VF2. Based on that, Section 4 introduces VF2++. Some technical details necessary for an efficient implementation are discussed in Section 5. Finally, Section 6 provides a detailed experimental evaluation of VF2++ and its comparison to the state-of-the-art algorithm.

It is also worth mentioning that the C++ implementations of the algorithms have been made publicly available for evaluation and use under an open-source license as a part of LEMON [10,12] graph library.[1]

2. Problem statement

This section provides a formal description of the problems to be solved.

2.1. Definitions

Throughout the paper $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ denote two undirected graphs.

Definition 2.1.1. $\mathcal{L} : (V_1 \cup V_2) \longrightarrow K$ is a **node label function**, where K is an arbitrary set. The elements in K are the **node labels**. Two nodes, u and v are said to be **equivalent** if $\mathcal{L}(u) = \mathcal{L}(v)$.

For the sake of simplicity, the graph, subgraph and induced subgraph isomorphisms are defined in a more general way.

Definition 2.1.2. G_1 and G_2 are **isomorphic** (by the node label $\mathcal{L}$) if $\exists \mathfrak{m} : V_1 \longrightarrow V_2$ bijection, for which the following is true:

$\forall u \in V_1 : \mathcal{L}(u) = \mathcal{L}(\mathfrak{m}(u))$ and
$\forall u, v \in V_1 : (u, v) \in E_1 \Leftrightarrow (\mathfrak{m}(u), \mathfrak{m}(v)) \in E_2$

Definition 2.1.3. G_1 is a **subgraph** of G_2 (by the node label $\mathcal{L}$) if $\exists \mathfrak{m} : V_1 \longrightarrow V_2$ injection, for which the following is true:

$\forall u \in V_1 : \mathcal{L}(u) = \mathcal{L}(\mathfrak{m}(u))$ and
$\forall u, v \in V_1 : (u, v) \in E_1 \Rightarrow (\mathfrak{m}(u), \mathfrak{m}(v)) \in E_2$

[1] The source code repository of the implementation is available at http://lemon.cs.elte.hu/hg/lemon-vf2. It is inclusion to the official LEMON library has also been proposed, see http://lemon.cs.elte.hu/trac/lemon/ticket/597.