Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Measurement and Information Systems

Elektronikus terelők

Scientific Students' Association Report

Author:

Jakab Gipsz
Második Szerző

Advisor:

dr. Egy Konzulens
Kettő Konzulens

2014

Contents

Kivonat

Jelen dokumentum egy diplomaterv sablon, amely formai keretet ad a BME Villamosmérnöki és Informatikai Karán végző hallgatók által elkészítendő szakdolgozatnak és diplomatervnek. A sablon használata opcionális. Ez a sablon LaTeX alapú, a *TeXLive* TeX-implementációval és a PDF-LaTeX fordítóval működőképes.

Abstract

This document is a LaTeX-based skeleton for BSc/MSc theses of students at the Electrical Engineering and Informatics Faculty, Budapest University of Technology and Economics. The usage of this skeleton is optional. It has been tested with the *TeXLive* TeX implementation, and it requires the PDF-LaTeX compiler.

Chapter 1

Introduction

Subgraph isomorphism is an NP-hard problem which has many application areas. It is called substructure search in cheminformatics and it is used to find similar molecular compounds based on their structural formula. In bioinformatics, complex biological systems are decomposed into several different networks, such as protein-protein interaction, metabolic interaction or hormone signaling networks, which are represented as graphs. Analyzing and understanding these large networks requires finding certain topological patterns, i.e. subgraph isomorphism.

Graph pattern matching is also a core concept of social network analysis. Such graphs tend to be extremely large with millions of vertices and billions of edges in the real world. Although subgraph isomorphism has been extensively studied in the past, there was a renewed interest in the topic recently, which yielded some notable results. The newer algorithms significantly out-perform the previous state-of-the-art solutions , sometimes even in order of magnitudes. This made it possible to query subgraph isomorphisms in such large graphs. However social networks are not static. In practice, they are frequently updated with typically small changes like adding or removing edges. Despite the changes being small, they will still have an impact on the matches. This means that the matches have to be re-computed from scratch on every update, which is highly infeasible even with the newer and faster subgraph isomorphism algorithms. To minimize unnecessary re-computations, incremental algorithms can be used, that compute the changes in matches based on the changes in the search graph.

ullman, vf2

[] discusses in depth several types of incremental graph pattern matching algorithms. However the authors' topic of interest in [] is incremental graph pattern matching with (bounded) graph simulation. A graph G matches a pattern q via graph simulation if there exists a binary relation $S \subseteq V_q \times V$ such that

cite https://www.pure.ed.ac.uk/ws/portalfiles/portal/17894104/Fan_Li_ET_AL_2011_Incremental_Graph_Pattern_Matching.pdf

1. for each $u \in V_q$, there exists $v \in V$ such that $(u, v) \in S$;
2. for each $(u, v) \in S$,
    (a) $\mathcal{L}(u) = \mathcal{L}(v)$, and
    (b) for each edge $(u, u') \in E_q$, there exists a non-empty path $\rho = v \rightsquigarrow v'$ in G such that $(u', v') \in S$ and the length of ρ is less than the maximum allowed length defined on the given edge in q.

Graph simulation is less strict about the topology of its results than graph isomorphism. This can be beneficial if we want to express loose connections in our query patterns, and on top of that, pattern matching with graph simulation can be done in $\mathcal{O}(n^3)$. However if we do require strict matches, only subgraph isomorphism can come into play. Although the authors provided an incremental algorithm for subgraph isomorphism, it was more of a demonstration that even in an NP-hard case, computing matches incrementally (which is also NP-hard) can out-perform a fast solution, VF2. The approach introduced there does not take full advantage on previous computations, and it was also not evaluated in much detail. Note that this was not the main focus of the paper.

In this work, we investigate how two state-of-the-art algorithms (VF2++, DAF) can be converted into their incremental version. First, we give an introduction how the two algorithms work. Then we describe a method to make them incremental. Finally, we evaluate the results both in terms of complexity and practical measurements.

1.1 Background

Definition 1 (Graph). A graph is a pair $G = (V, E)$, where V is a set of vertices, E is a set of paired vertices that denotes the undirected edges of the graph. ▪

Definition 2 (Labelling). $\mathcal{L} : V \to K$, is a vertex labelling function which maps vertices into arbitrary sets whose elements are the labels of the given node. Two vertices, u, v are equivalent if $\mathcal{L}(u) = \mathcal{L}(v)$. ▪

Definition 3 (Isomorphism). G_1 and G_2 are isomorphic if a bijection exists between V_1 and V_2 such that two vertices are neighbours in G_1 if and only if their respective pairs in G_2 are neighbours, neighbouring vertex pairs have the same number of edges between each other, and a vertex and its pair have the same labels. ▪

Definition 4 (Subgraph). G_1 is a subgraph of G_2 if $V_1 \subseteq V_2$, $E_1 \subseteq E_2$ and two vertices are neighbours in G_1 only if they are neighbours in G_2. ▪

Definition 5 (Induced subgraph). If E_1 consists of those edges from E_2 whose both vertices are in V_1, and E_1 contains all these edges, then G_1 is an induced subgraph of G_2. ▪

Definition 6 (Subgraph isomorphism). G_1 is subgraph isomorphic to G_2 if G_1 is isomorphic to any subgraphs of G_2. ▪

Definition 7 (Induced subgraph isomorphism). G_1 is induced subgraph isomorphic to G_2 if G_1 is isomorphic to any induced subgraphs of G_2. Throughout this paper, we refer to induced subgraph isomorphism with the term of subgraph isomorphism. $M(q, G)$ denotes the set of mappings found by an arbitrary subgraph isomorphism algorithm. ▪

This paper concentrates on the (induced) subgraph isomorphism problem in dynamic graphs, i.e. graphs that change with time. After finding the initial matches given a query graph q and a data graph G, keep the set of matches up to date in response to small updates ΔG on G without recomputing all matches from scratch. ΔG can be one of the following operations:

- add a new node to G,
- remove an existing node from G,
- add an edge between two nodes of G and
- remove an existing edge between two nodes of G.

1.2 Algorithms

This section gives a brief overview on how the two algorithms of interest (DAF and VF2++) work.

1.2.1 VF2++

1.2.1.1 DAF

Chapter 2

Incremental algorithms

This chapter describes two incremental algorithms for finding subgraph isomorphisms in dynamic graphs. First we introduce the approach proposed in []. Then we present our generic method which can be applied to DAF and VF2++ respectively.

2.1 Locality based method

Let d denote the diameter of the query graph q which corresponds to the length of the longest shortest path in q. Let Δe denote the addition or deletion of edge $e = (v, v')$ in G. Moreover let $V(d, e)$ be the set of vertices in G that are within a distance d from v or v'. Finally, let $G(d, e)$ be the subgraph of G induced by $V(d, e)$, and $\Delta G(d, e)$ be the subgraph of ΔG induced by $V(d, e)$, where $\Delta G = (V, E \setminus \{e\})$ or $\Delta G = (V, E \cup \{e\})$ depending on the operation of Δe. With these notations, we can define the locality property of subgraph isomorphism. For any given changes of Δe in G, the changes ΔM in the set of mappings $M(q, G)$ is the difference between $M(q, G(d, e))$ and $M(q, \Delta G(d, e))$, i.e. the difference between the mappings found in the dth neighborhood of the original graph and in the neighborhood's modified version. This is a trivial property because new mappings can appear or become obsolete only in the immediate vicinity of the affected edge e. Mappings further away from e than d are not affected by Δe because neither vertices of e are reachable from there. Regardless of that a new edge was added or an old one was removed, mappings can both appear and become obsolete in the updated graph. The mappings that have to be added to the existing set is $M(q, \Delta G(d, e)) \setminus M(q, G(d, e))$, while the mappings that have to be removed are $M(q, G(d, e)) \setminus M(q, \Delta G(d, e))$.

The algorithm uses this locality property as follows. Given a query graph q and a data graph G, compute the initial set of mappings M with a classic subgraph isomorphism algorithm, e.g. VF2++. Then, for each update Δe, (1) find the diameter d of q, (2) extract the subgraph $\Delta G(d, e)$ from G, (3) compute $M(q, \Delta G(e, d))$, (4) then update M as described above.

Instead of re-computing all mappings in G, this method works on a subset of G. Although a regular subgraph isomorphism is running in the background, reducing the size of the input data graph can make this variant faster. The effectiveness depends on the query graph and the data graph. If $G(d, e)$ remains small compared to G, there will be a massive speed up. However if $G(d, e) \sim G$, for example in case of small world networks, the algorithm performs the same as a regular subgraph isomorphism algorithm.

Algorithm 1: Locality algorithm

```
  Input: $q, G, \Delta e, M = VF2++(q, G)$
1 $d \leftarrow diameter(q)$;
2 $G_{d,e} \leftarrow G(d, e)$;
3 $\Delta M \leftarrow VF2++(q, G_{d,e})$;
4 $M = M \cup M(q, \Delta G(d, e)) \setminus M(q, G(d, e))$;
5 $M = M \setminus M(q, G(d, e)) \setminus M(q, \Delta G(d, e))$;
6 return $M$
```

2.2 Search tree based method

This chapter describes our method of making the previously introduced algorithms incremental. One way or another, both algorithms traverse a search tree eventually. In both cases the search tree is purely abstract, it has no physical manifestation in the memory whatsoever. We propose to store the traversed search tree while running subgraph isomorphism initially, and make use of it in future updates. $T_{q,G}$ is a search tree of query graph q and data graph G, where paths from the root to a leaf correspond to (partial) mappings, and a node contains an (u, v) pair where $u \in V_q, v \in V_G$ denotes a single vertex mapping. The root r of $T_{q,G}$ contains an empty mapping. A root-to-leaf path, whose length $l = |V_q|$ is a complete mapping because it maps all nodes of q. The rest of the root-to-leaf paths correspond to partial mappings.

In a typical scenario at the end of an execution of a subgraph isomorphism algorithm, the search tree contains some complete mappings and orders of magnitude more partial mappings that for some reason could not be extended any further. These partial mappings play a crucial role in the incremental version. One can think about them as potential partially precalculated mappings. The reason, why a partial mapping could not be extended is because the algorithm ran out of valid candidates in the area of the mapped nodes' neighborhood in G. This is caused because by some kind of conflict between the topology of q and the topology of the mapped nodes and their neighborhood. These conflicts could be resolved when a new edge is added to or an old one is deleted from G. At this point, having these partial mappings in memory can speed up the process of removing matches that became obsolete and finding new ones that were impossible in the past.

In the upcoming sections, we define an incremental algorithm for each type of graph modifications.

2.2.1 Node deletion

Deleting a node from v_d from G can only reduce the original number of mappings. Mappings that did not contain v_d are unaffected, while those that contained v_d have to be removed since they no longer can map to v_d. Now we only have to think through that no new mappings could arise. Indeed, that cannot be the case because the only edges that were removed are edges belonging to v_d which means only partial mappings which contain v_d are affected. However since v_d itself is also deleted, these mappings become obsolete, which means that no new mappings can be found.

Since there are no new mappings to be found, the only work to do in case of a node deletion is pruning the search tree $T_{q,G}$. More specifically, cut off every sub-branch that starts with a node (u, v) where $v = v_d$. In worst case scenario this would mean traversing the whole search space. We already saw however, that any given change can only affect mappings whose vertices are in the dth neighborhood $N_d(G, v_d)$ of v_d. Thus we can skip any branches that map a query graph vertex to a node $v \notin N_d(G, v_d)$.

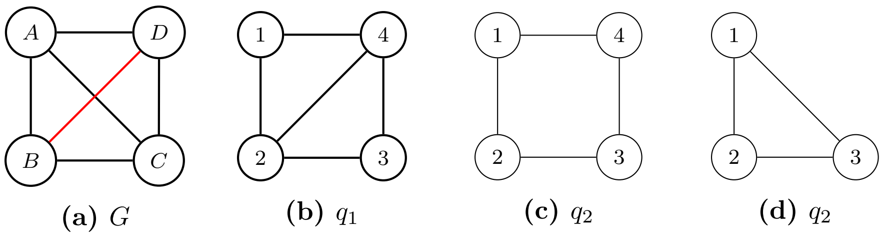
```
Algorithm 2: Delete node incrementally
   Input: $q, G, T_{q,G}, v_d$
2  $d \leftarrow$ diameter($q$)
4  $N_d(G, v_d) \leftarrow d$th neighborhood of $v_d$ in $G$
5  Procedure delete_node(node, $v_d$)
6      foreach child $\in$ node.children do
8          $(u, v) \leftarrow$ child.mapping
9          if $v = v_d$ then
11             node.remove(child)
12         else if $v \in N_d(G, v_d)$ then
14             delete_node(child, $v_d$)
15         end
16     end
```

2.2.2 Node insertion

Inserting a new node has no effects on mappings whatsoever because at the time of insertion, it cannot be part of any mappings because it has not connections. Thus inserting a new node needs no special care.

2.2.3 Edge operations

In case of induced subgraph isomorphism, both inserting and deleting an edge can result in loosing and gaining mappings. Removing an edge e makes mappings where nodes of e were part of the mapping no longer valid. At the same time, in case of partial mappings where e made it impossible to further extend, removing it may cause new mappings to appear. For example in 2.1, as a result of removing the BD edge from G, $M(q_1, G')$ will loose half of its mappings, while $M(q_2, G')$ will double the amount of its mappings.



(a) G **(b)** q_1 **(c)** q_2 **(d)** q_2

Figure 2.1: Example how mappings can appear and disappear in both cases of edge deletion and insertion.

The same goes for edge insertion. If we were to insert the BD edge into G then $M(q_1, G')$ would find new mappings, while $M(q_2, G')$ should remove the half of them. It is clear that these can happen simultaneously, as well.

2.2.3.1 Edge deletion

When an edge $e = (v_1, v_2)$ is removed from G, first, we have to prune the search tree to remove mappings that are no longer valid and we have to somehow find the new mappings. We have to prune all branches that correspond to a partial mapping which both v_1 and v_2 are part of, and there is an edge between $m^{-1}(v_1)$ and $m^{-1}(v_2)$ in q. We find all paths that correspond to such partial mappings, and prune them down from the first point where v_1 and v_2 both appeared in the path. 3 shows the pruning algorithm when an edge was deleted.

Algorithm 3: Prune search tree on edge deletion

```
Procedure prune(node, $v_1$, $v_2$)
    $(u, v)$ = node.mapping
    if $v_1$ not marked as found $\wedge v = v_1$ then
        mark $v_1$ as found
    else if $v_2$ not marked as found $\wedge v = v_2$ then
        mark $v_2$ as found
    end
    if $v_1$ is marked $\wedge$ $v_2$ is marked then
        if $(m^{-1}(v_1), m^{-1}(v_2)) \in E_q$ then
            return true
        end
    end
    $node.children = \{child \in node.children : \neg \texttt{prune}(child, v_1, v_2)\}$
```

Fig 2.2 shows an example partial search tree of G and q_3 from Fig. 2.1. The red nodes represent the pruned sub-branches as a result of removing the BD edge from G.

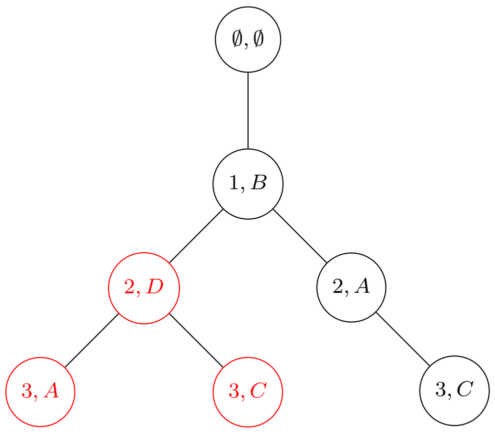


Figure 2.2: Pruning of a partial search tree of G and q_3 from Fig. 2.1.

Finding new mappings is somewhat harder because we have to extend the search tree instead of just pruning it. Because of the locality property of the incremental subgraph isomorphism problem, we know for sure that any new mappings that will appear have to contain either v_1 or v_2 or both. Thus our job is reduced to look for partial mappings that contain either of those two vertices. There are two kinds of partial mappings to consider. Typically, the search tree will contain some partial mappings that contain one of these two vertices but definitely not all of them. As a first step, we add these partial mappings to the search tree. We traverse the tree and check if a node has a child that maps the next vertex

$u \in q$ to v_i. If it does not, and v_i was not covered already in the mapping, and it causes no conflicts between the topology of the mapping and q then we create a new child under the current node with $m(u) = v_i$. Now that we have all the partial mappings containing v_1 or v_2 - note that not all of them are expanded yet - we find all those leaves whose path from the root corresponds to a partial mapping that contains v_1 or v_2 and we execute our choice of subgraph isomorphism algorithm from that point. Naturally, these two steps can be combined into one traversal. If we hit a node where (1) we can create a new child or (2) a node is a leaf and its path contains v_1 or v_2, and it is not a complete mapping, we execute our subgraph isomorphism algorithm. We can speed up the algorithm by only considering those branches that contain vertices such that $\forall v \in m : v \in N_d(G, v_1) \cup N_d(G, v_2)$, i.e. branches that contain only mapped vertices such that they are in the dth neighborhood of e. Alg. 4 formally describes the algorithm of finding new mappings after an edge has been deleted, and Alg. 5 shows the incremental algorithm for subgraph isomorphism in case of edge deletion.

Algorithm 4: Find new mappings incrementally

```
1  Procedure find_mappings(node, forced_candidates)
2      if $node.depth = |V_q| \vee node.mapping.v \notin N_d(G, (v_1, v_2))$ then
4          return
6      u ← node.extendable_vertex_of_children
7      foreach $v_c \in forced\_candidates$ do
8          if $\nexists child : child.mapping = (u, v_c)$ then
9              if $v_c$ is not covered $\wedge$ $v_c$ is good candidate then
11                 child = node.add_child(u, $v_c$)
13                 ISO(child)
14     end
15     if node is leaf $\wedge$ $(v_1 \in m \vee v_2 \in m)$ then
17         ISO(node)
18     else
19         foreach $child \in node.children$ do
21             find_mappings (child, forced_candidates)
22         end
23     end
```

Algorithm 5: Delete edge incrementally

```
   Input: $q, G, T_{q,G}, e$
2  $d$ ← diameter($q$)
4  $(v_1, v_2) = e$
6  $N_d(G, (v_1, v_2))$ ← $d$th neighborhood of $v_1$ and $v_2$ in $G$
8  prune (root($T_{q,G}$), $v_1, v_2$)
10 find_mappings (root($T_{q,G}$), $\{v_1, v_2\}$)
```

2.2.3.2 Edge insertion

Inserting a new edge into G will have a really similar effect on the mappings as and edge deletion. Mappings can both appear and disappear from the original mapping set. In fact, the algorithm described above is almost applicable to this problem out of the box. It is clear that finding new mappings will be the same in this case, as well. We have to take care about how we prune the search tree before that. Indeed, removing nodes that pass the test $(m^{-1}(v_1), m^{-1}(v_2)) \in E_q$ makes no sense because we just added that new edge. We can make this condition generic however, so that both edge operations can use the same predicate. Instead of separating the two cases, we will simply check that an edge between v_1 and v_2 in G exists if and only if an edge between $m^{-1}(v_1)$ and $m^{-1}(v_2)$ in q exists. With these modifications, the final pruning algorithm can be found in Alg. 6.

Algorithm 6: Prune search tree on edge operation

```
Procedure prune(node, $v_1$, $v_2$)
    $(u, v)$ = node.mapping
    if $v_1$ not marked as found $\wedge$ $v = v_1$ then
        mark $v_1$ as found
    else if $v_2$ not marked as found $\wedge$ $v = v_2$ then
        mark $v_2$ as found
    end
    if $v_1$ is marked $\wedge$ $v_2$ is marked then
        is_edge_in_q $\leftarrow (m^{-1}(v_1), m^{-1}(v_2)) \in E_q$
        is_edge_in_G $\leftarrow (v_1, v_2) \in E_G$
        if is_edge_in_q $\neq$ is_edge_in_G then
            return true
        end
    end
    $node.children = \{child \in node.children : \neg \texttt{prune}(child, v_1, v_2)\}$
```

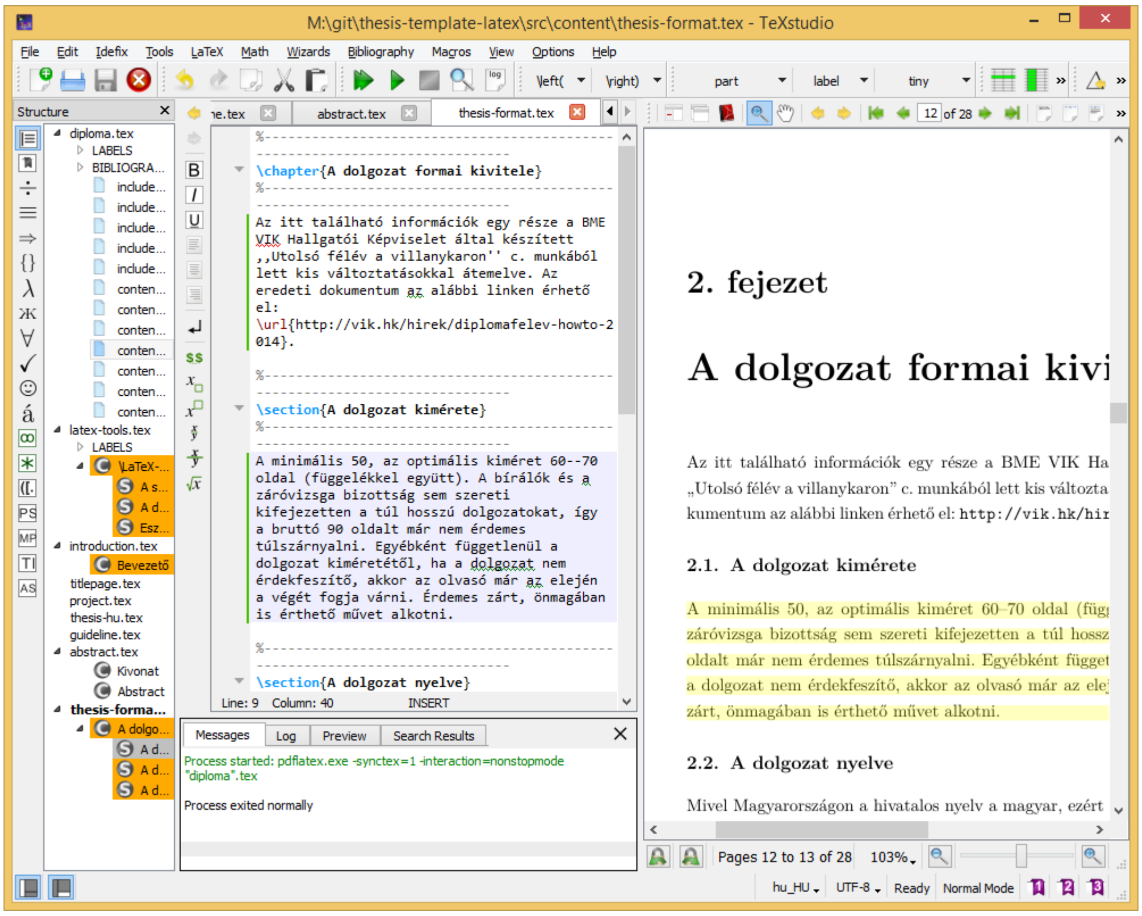
Acknowledgements

Ez nem kötelező, akár törölhető is. Ha a szerző szükségét érzi, itt lehet köszönetet nyilvánítani azoknak, akik hozzájárultak munkájukkal ahhoz, hogy a hallgató a szakdolgozatban vagy diplomamunkában leírt feladatokat sikeresen elvégezze. A konzulensnek való köszönetnyilvánítás sem kötelező, a konzulensnek hivatalosan is dolga, hogy a hallgatót konzultálja.

Bibliography

Appendix

A.1 A TeXstudio felülete



```
%--------------------------------------------
------------------------------
\chapter{A dolgozat formai kivitele}
%--------------------------------------------
------------------------------
Az itt található információk egy része a BME
VIK Hallgatói Képviselet által készített
,,Utolsó félév a villanykaron'' c. munkából
lett kis változtatásokkal átemelve. Az
eredeti dokumentum az alábbi linken érhető
el:
\url{http://vik.hk/hirek/diplomafelev-howto-2
014}.

%--------------------------------------------
------------------------------
\section{A dolgozat kimérete}
%--------------------------------------------
------------------------------
A minimális 50, az optimális kiméret 60--70
oldal (függelékkel együtt). A bírálók és a
záróvizsga bizottság sem szereti
kifejezetten a túl hosszú dolgozatokat, így
a bruttó 90 oldalt már nem érdemes
túlszárnyalni. Egyébként függetlenül a
dolgozat kiméretétől, ha a dolgozat nem
érdekfeszítő, akkor az olvasó már az elején
a végét fogja várni. Érdemes zárt, önmagában
is érthető művet alkotni.

%--------------------------------------------
------------------------------
\section{A dolgozat nyelve}
```

Line: 9 Column: 40 INSERT

```
Process started: pdflatex.exe -synctex=1 -interaction=nonstopmode
"diploma".tex

Process exited normally
```

2. fejezet

A dolgozat formai kivi

Az itt található információk egy része a BME VIK Ha „Utolsó félév a villanykaron” c. munkából lett kis változta kumentum az alábbi linken érhető el: `http://vik.hk/hir`

2.1. A dolgozat kimérete

A minimális 50, az optimális kiméret 60–70 oldal (füg záróvizsga bizottság sem szereti kifejezetten a túl hossz oldalt már nem érdemes túlszárnyalni. Egyébként függet a dolgozat nem érdekfeszítő, akkor az olvasó már az elej zárt, önmagában is érthető művet alkotni.

2.2. A dolgozat nyelve

Mivel Magyarországon a hivatalos nyelv a magyar, ezért

Figure A.1.1: A TeXstudio LaTeX-szerkesztő.

A.2 Válasz az „Élet, a világmindenség, meg minden” kérdésére

A Pitagorasz-tételből levezetve

$$c^2 = a^2 + b^2 = 42. \tag{A.2.1}$$

A Faraday-indukciós törvényből levezetve

$$\operatorname{rot} E = -\frac{dB}{dt} \qquad \longrightarrow \qquad U_i = \oint\limits_{\mathbf{L}} \mathbf{Edl} = -\frac{d}{dt}\int\limits_{A} \mathbf{Bda} = 42. \tag{A.2.2}$$