International Journal of Scientific and Technical Research in Engineering (IJSTRE)
www.ijstre.com Volume 6 Issue 4 ǁ July-Aug 2021.
ISSN: 2581-9941

Target Cell Selection Algorithms in Graph Isomorphism problem

Utku Cem Sankal[1], Nisar Ahmed [1], Farhan Ajmal Khan[2], Rubiya Hanni[1]
[1]*(Department of Computer Science, Sapienza University of Rome, Italy)*
[2]*(Department of Artificial Intelligence , Sapienza University of Rome, Italy)*
sankal.1765821@studenti.uniroma1.it

ABSTRACT : *The Graph Isomorphism problem is an important problem that occur in many fields including computer science, chemistry, mathematics, image processing and geography. Graphs are used to represent several situations and structures from real life, and we want to know that whether two graphs are same or different from a selected perspective. All competitive GI testing tools follow the so called "individualization-refinement "procedure. Target cell selection play an important role within this procedure. In this study we have implemented four different target cell selectors using one of the best competitive GI tools, "traces" of Adolfo Piperno. The experiments we have conducted shows that the "largest" target cell selector performs better in terms of CPU time for most of the graphs, which have a group size greater than 2400.*

Keywords: *GraphIsomorphism Problem, Refinement-Individualization Method, Target Cell Selection, Largest Target Cell Selection, Max-Join Target Cell Selection.*

I. INTRODUCTION

The Graph Isomorphism problem is an important problem that occurs in many fields including computer science, chemistry, mathematics, image processing and geography (Somkunwar et al., 2017). Graphs can be used to represent several situations and structures from real life, and we want to know that whether two graphs are the same or different from a selected perspective. For instance, the graphs in the figure below are equal even though they seem to be different.

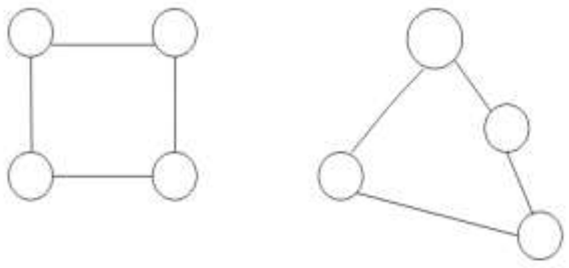

Figure 1. A simple example of two isomorphic graphs

The graph isomorphism problem is determining whether there is an isomorphism between two given graphs. GI has been a favorite discussion topic among algorithm designers since long time and described as a disease in 1976 by Read and Corneil (McKay et al., 2014).GI has several application areas. For instance, in chemistry, GI is used to identify a chemical compound within a chemical database.Image processing can be given as another application area of GI (Somkunwar et al.,2017). In image processing GI is used to match two different images. To do that, the input image first converted into graph. After obtaining the graph, graph isomorphism algorithms applied to decide whether input graph is same as one of the graphs of the database.
The first program that could handle regular graphs with hundreds of vertices was the software of McKay that later became known as "nauty". In literature, there exists hundreds of published GI algorithms. Up to date, the best performing algorithms follow "individualization-refinement" paradigm. This paradigm was introduced by Parris and Read (1969) and developed by Corneil and Gotlieb (1970)and Arlazarov. (1974) (McKay et al., 2014). Currently "nauty "is being considered as the best performing algorithm package for small graphs while "Traces" is the leader for most of the difficult graph classes. In this study, our main objective is observing the effects of different target cell selectors on some of the difficult graph types by taking advantage of the "traces" software. To do this, we will first introduce graph and Graph Isomorphism Problem. Then, we will talk about the complexity of the GI. After that we are going describe what a target cell is and explain the importance of target cell selection in Graph Isomorphism Problem. In the last part of the section 2 we are going to describe principles behind the individualization-refinement based graph isomorphism testing algorithms and explain the working logics of some of the well-known graph isomorphism testing packages.

In upcoming section, we are going to introduce working principles of the “traces” software.Then we will mention about some of the difficult graph types for isomorphism testing.Moving forward we will describe four different target cell selection functions that we have implemented using “traces” software and then provide experimental results for each of them regarding to experiments that we have conducted on some of the difficult graph types. Finally, we are going to discuss the experimental results and conclude the thesis after talking about possible future works.

II. LITERATURE REVIEW

Graph Isomorphism Problem

In this Section we introduce the Graph Isomorphism problem (GI), together with some fundamental notions and definitions, which will lead us to introduce the topic of this research.

Graph.

A graph is a tuple:

G= (V ,E)

where V represents the set of vertices and E represents the set of edges.

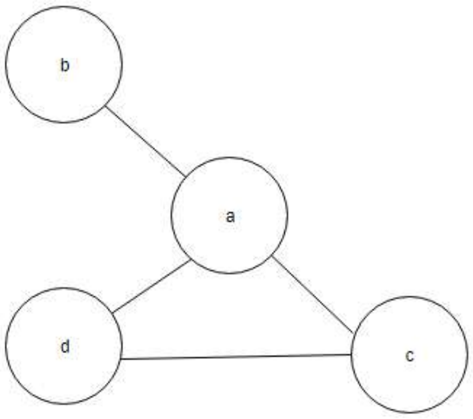

Figure 2 . A Simple Graph

For example, in the figure above, a graph including four nodes and four edges is shown. This graph can be expressed as:

G= ({a,b,c,d}, {(a,b),(a,c),(c,d),(a,d)})

Given two graphs F and G, we say that F and G are isomorphic if there exists a bijection of vertices of F and G which preserves the edges. Clearly, two isomorphic graphs have the same number of vertices and edges. The basic but not efficient way to prove isomorphism is by checking all possible permutations of the vertices and checking whether there exists a one-to-one edge preserving mapping between the vertices of the two graphs (John et al.,1974). This is a brute force approach.To make the concept clearer, consider the Figure-2 and Figure-3.

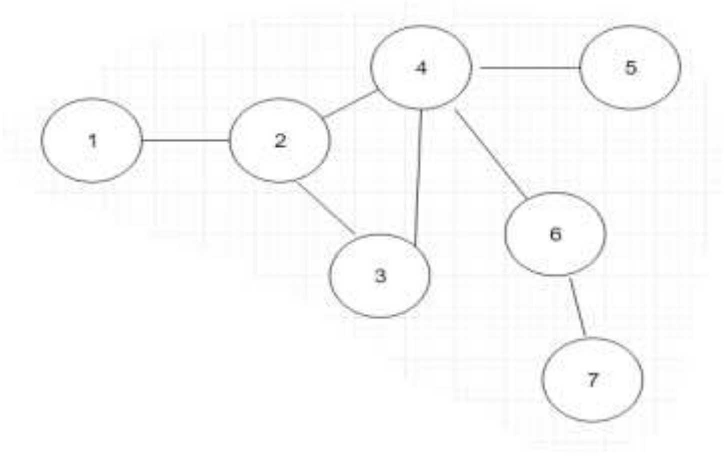

Figure 3. Graph F

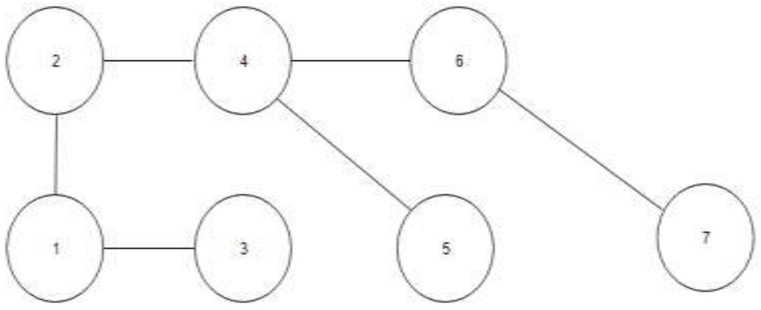

Figure 4. Graph G

One can prove that these two graphs are isomorphic by trying all possible permutations of vertices of graph F. After trying all possible permutations, it is possible to say that these two graphs are isomorphic. A one-to-one mapping between the vertices of the graphs F and G is shown in Table 1.

Table 1
Mapping of Vertices

Vertices of Graph F	**Vertices of Graph G**
1	7
2	6
3	5
4	4
5	3
6	2
7	1

Graph Isomorphism.
Two graphs:
$F=(V_F,E_F)$ and $G=(V_G,E_G)$
are isomorphic if and only if there exist one to one mapping,
$\square : V_G \rightarrow V_G$
such that,
$\forall u,v \in V_F : (u,v) \in E_F \leftrightarrow (\square(u), \square(v)) \in E_G$ and $(\forall u \in V_F : u \in E_F \leftrightarrow \square(u) \in E_G)$
An automorphism of a graph G means relabeling the vertices of the graph in such a way that the structure of the graph does not change (Somkunwar et al., 2017). For example, consider the graph in Figure 4. In this graph if we exchange the labels between the vertices F and C, the structure of the graph remains the same.

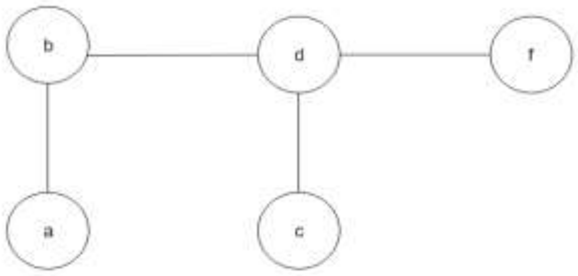

Figure 5. A graph for automorphism example

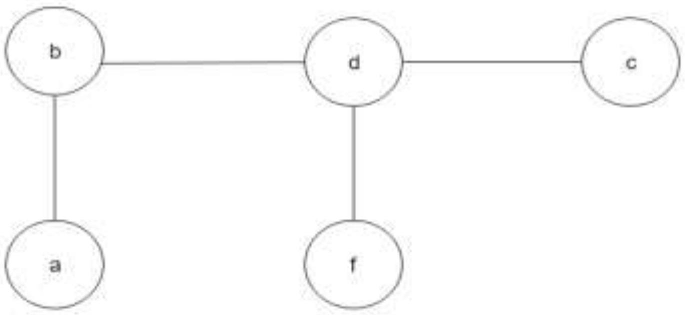

Figure 6. The automorphism of (C,F) for the graph in Figure 4

Automorphism.
An isomorphism between a graph and itself is defined as automorphism.Given a graph G=(V,E) a permutation γ: V → V which is an isomorphism from G to G is an automorphism of G.

Automorphism Group.
The set of automorphisms including the trivial one (the one that moves no label also referred as identity permutation) under composition is called automorphism group (Alfred, John& Jeffrey ,1974). An Automorphism Group is denoted by Aut(G).

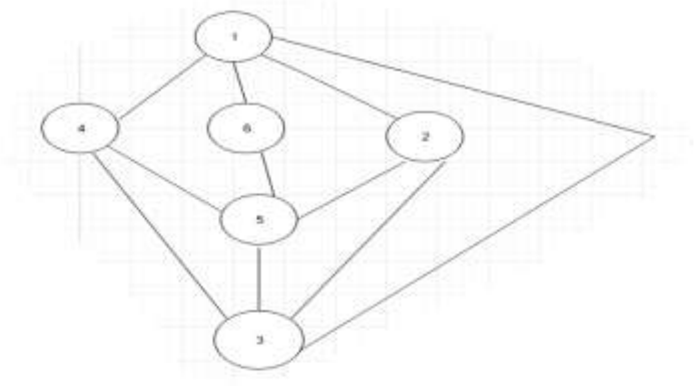

Figure 7. A graph for automorphism example

The automorphisms of the graph in Figure 2 can be written as:
1. The identity permutation (maps each vertex to itself)Γ1 = (1) (2) (3) (4) (5) (6)
2. Γ2 = (1 5) (2) (3) (4) (6) that swaps the vertices 1 with 5.
3. Γ3 = (1) (2 4) (3) (5) (6) that swaps the vertices 2 with 4.
4. Γ3 = (1 5) (2 4) (3) (6) that swaps the vertices 1 with 5 and 2 with 4.

Babai has shown that the best bound for GI problem is in quasi-polynomial time (Somkunwar et al., 2017). Also, many works have been done for specific graph types. For some of these graph types of polynomial time solution is found.Kelly and Aho worked on trees and found a polynomial time algorithm for trees. Hopcroft found an algorithm that can solve GI for graphs (Hopcroft et al., 1974). Colbourn was also able to find an algorithm to solve GI on permutation graphs in polynomial time. Luks, proved that bounded valance graphs can be handled in polynomial time (Presa, 1974; Eugene et al., 1981).The most well-known and fastest ones of these software tools are; “nauty” of Brendan McKay, “Traces “of Adolfo Piperno, “bliss” of Tommi Juntilla and Petteri Kashi, “saucy” of Paul T. Darga, Karem A. Sakallah, and Igor L. Markov, “sinauto “and “conauto” of José Luis López Presa(Somkunwar ,2017; Krena et al., 2001).The common feature about the fastest implementations of GI testing tools is all they follow individualization/refinement approach.

Individualization-Refinement Method.
The individualization-refinement basically aims to classify the vertices of a given graph by similarity. It iteratively partitions or assign colors to vertices in a sequence of refinement rounds (colors have a predefined order). Refinement refers to assigning colors to the vertices of a graph due to some criteria. Most common criteria used for refinement process in the literature is considering the degrees of the vertices of the graph.

Partition.
A partition is a colored form of a graph, where each vertex of a graph is colored.

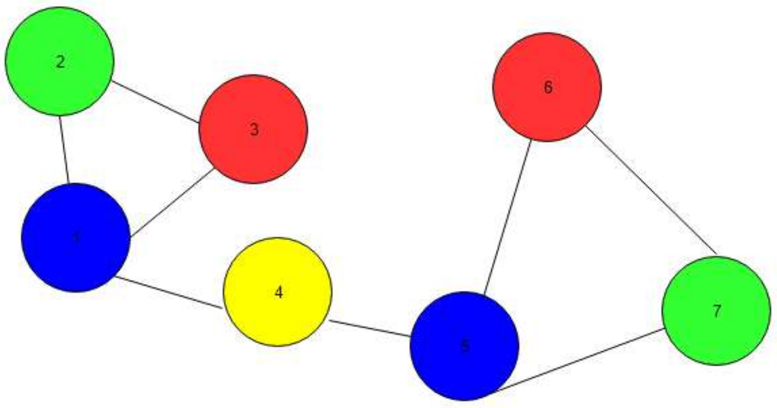

Figure 8. A partition

In individualization refinement method, all vertices are assigned to the same color. Then in each refinement round, any two vertices a, b that still have the same color are assigned to different colors if and only if there exist some color c such that a and b have different number of neighbors of color c. The refinement process ends if in some refinement round we obtain what is called an equitable partition.

Equitable Partition.
An equitable partition is a colored graph where every two vertices of the same color are adjacent to the same number of vertices of each color (Greg et al., 2005). For example, if we look at Figure 8, we can see that every two vertices of the same color are adjacent to the same number of vertices of each color. Because of this we can say that the partition in Figure 7 is also an equitable partition.After obtaining the equitable partition, individualization-refinement based algorithms individualize a vertex of a color.

Discrete Partition.
A discrete partition is a colored graph where every vertex has a unique color.
On the figure below, a discrete partition can be seen.

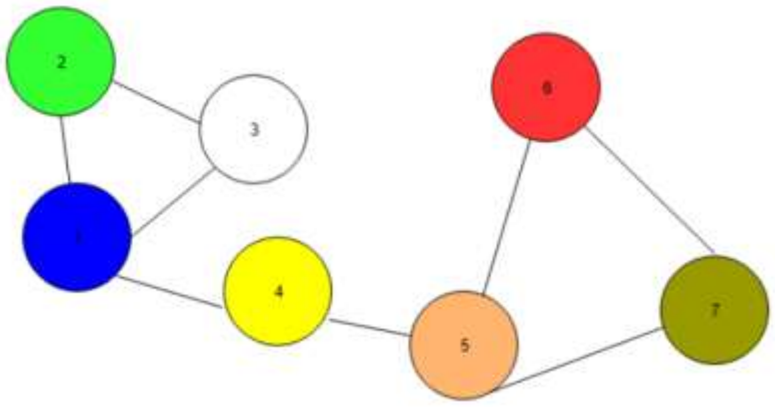

Figure 9. A discrete (also equitable) partition

To detect the isomorphism between two graphs, individualization-refinement based algorithms generate a search tree based on partition refinement. Then traverse the search tree of each graph and compare them. Two graphs are said to be isomorphic if and only if their search trees are isomorphic. The nodes of the tree are partitions. The root of the tree is initial partition. If a node corresponds to a discrete partition, then it is a leaf. If the node is not a discrete partition, then algorithm chooses a color (target cell) which is at least used more than once and individualizes one of the vertices from this color (assigns a unique color to the vertex) and then refines to get a child.

Target Cell and Importance of Target Cell Selectors.

Target cell selection algorithms play an important role in the performances of individualization-refinement based GI testing software tools. As we have previously mentioned, the most common approach to assign colors to vertices is considering the number of neighbors of the vertices. To understand the target cell and target cell selectors, we need to understand the concept of cells.

Cell.

A cell is a set that consists of the vertices of a partition which have same color.If we consider the Figure 8 the cells of the graph can be written as:3 6|1 5|2 7| 4|. Here the sign of "|" is used to express the end of the cells.The performance of the graph isomorphism testing software depends on the number of operation that graph isomorphism testing software performs to obtain a discrete partition. At this point, cell selectors play an important role. Basically, cell selector selects a cell between many cells depending on some criteria and rest of the software performs its operations starting from the vertices in that cell.
To make the concept more solid let us consider the example below. Assume that we are working on a individualized-refinement based software and we have an equitable partition which is in Figure 8. Then, the target cell selection function within the software must choose a cell depending on some strategy and return the index number of the beginning of the selected cell.Assume that our strategy is choosing the first largest cell. Then according to our strategy, the target cell selector chooses the cell |3 6|. Then, it individualizes vertex 3, which is the first vertex of the cell according to our example. Then the new partition may look like as in the follows:

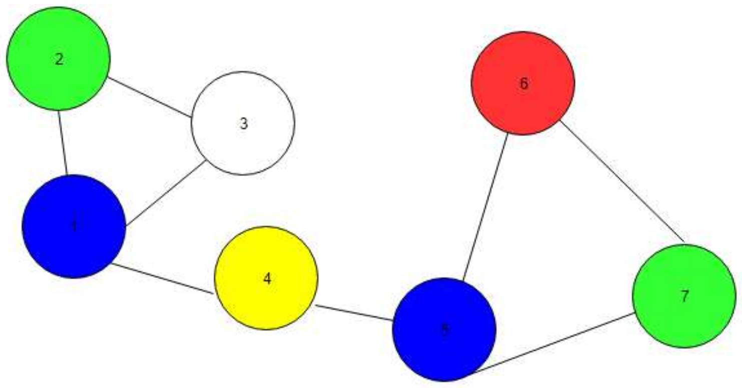

Figure 10. New partition after individualization

As we can see from the Figure 10, dark blue and green vertices do not follow the equitable partition definition (for instance while vertex 1 is adjacent to a light blue vertex 5 is not. This situation is also same for the green vertices). Because of this by refining one dark blue and one green vertex it is possible to obtain an equitable partition, which may look like the partition in Figure 9. An effective cell selector can lead to obtain a discrete partition with a smaller number of refinement operations and those it can enhance the performance of the graph isomorphism testing software. It is also possible to follow other target cell selection algorithms such as choosing the cell which is adjacent to maximum number of another cell or first smallest cell etc. All these different target cell selection strategies may lead to different numbers of refinement operations.

III. METHODOLOGY

Currently "nauty "is being considered as the best performing algorithm package for small graphs while "Traces" is the leader for most of the difficult graph classes. (Difficult graphs will be discussed in following sections).As we have explained within the previous sections target cell selection play an important role within the individualization-refinement method. The importance of the target cell selection can also be seen in the literature (Eugene, 1981; McKay et al., 2014).In literature there exist no work that discusses the effects of different target cell selection algorithms for different graph types.The goal of this thesis is observing the effects of the different choices of the target cell on "traces" software tool by conducting several experiments.

Canonical Labelling and Working Principles of Individualizing-Refinement Based Software

The canonical labeling or the canonical form of a graph G is a graph G', which is isomorphic to G and represents the whole isomorphism class of G.Simply, a canonical label is a unique representation of a graph and it literally behaves like a "label" for the graph. Most of the best performing GI testing algorithms aims the compute unique the canonical labels for the given graphs.Canonical labelling algorithms work on each compared graph and generate the canonical labels of them, independently from one another, and finally compare them to detect isomorphism.A canonical member of an isomorphism class is a member that is chosen from that isomorphism class. The process of the finding the canonical label of a given isomorphism class is called as canonical labelling. Two labelled graphs which are isomorphic, said to be identical when they are canonically labelled.

Individualization-Refinement Based GI Software Tools

Most of the GI software are focused on developing practical GI algorithms to be able to solve the general problem efficiently. On one side there exist software tools that solves the GI by canonically labelling the graphs. Some of them can be enumerated as; “bliss” of Petteri Kashi and Tommi Juntilla, “nishe “of Greg Tener, “nauty” of Brendan McKay and “traces” of Adolfo Piperno. On the other side there exist some tools that tries to find matches between two graphs by detecting symmetries. “Saucy” of Paul T. Darga, Karem A. Sakallah, and Igor L. Markov,software can be given as an example to this category. We need toanalyze the working principles of the Traces in means of the strategies that follow to solve GI.

Traces

“Traces” is another GI testing tool that outperforms its competitors for many graph classes. They are the two programs to compute the automorphism groups and canonical labels of the graphs. It is both written in C programming language and have been provided within the same software package with “nauty”. “Traces” starts its refinement processes from an initial partition in which, all vertices that have same degree are placed in the same cell. Then it chooses a target cell. Unlike “nauty”, Traces prefers large target cells. The logic underlying this idea is that larger target cells make less deep search trees (McKay et al.,2014). Traces chooses the first largest non-singleton cell as the target cell, which is a subset of the target cell in the parent node. If there is not any non-singleton cell, the target cell in the grandparent node is used. “Traces” individualizes the vertices of the target cell by one by until obtaining a discrete partition such as other competitive GI tools.
The advantage of the “Traces” approach is the provided ability to prune the search tree in the maximum extent possible, as theoretically proven in study (Kutz et al.,2007). When it comes to the detection of automorphisms nauty and Traces have similar strategies, but they differ in search tree generation and target cell selection processes as explained in the previous section..

Difficult Graphs for Isomorphism Problem

According to the studies, the performances of the best performing isomorphism programs decrease exponentially when they are dealing with graphs containing lower numbers of automorphisms, but higher degrees of regularity (Greg ,2005 ; McKay et al., 2009).From another studies it was found that the graph families listed below are considered as difficult instances for canonical labeling based graph isomorphism programs (Presa,2009; Miyazaki,1997; Kocay et al., 1996).These areStrongly Regular Graphs,Miyazaki Graphs,Hadamard Matrices, Affine and Projective Planes, Random Regular Graphs. To see a more detailed list of different graph types you can refer to the (Tommi et al., 2007).In the last part of our research, we are going to represent the experimental results that we have conducted on some of these graphs. Because of this reason we have found it beneficial to provide some information about the difficult graph types.

Strongly Regular Graphs

A strongly regular graph with parameters (n, k, λ, μ) is a regular graph of degree k on n vertices, such that each pair of adjacent vertices has λ common neighbors, and each pair of non-adjacent vertices has μ common neighbors (Presa et al., 2009). Strongly regular graphs can be categorized into different subgroups as explained in study (Tommi et al., 2007). Lattice Graphs, Latin Square Graphs, Triangular Graphs are some examples to Strongly Regular Graphs.A Latin Square is an $n \times n$ array filled with n different symbols, each occurring exactly once in each row and exactly once in each column.The family of Latin Square Graphs is generated from Latin squares. A Latin square of order n, $n \geq 2$, is an $n \times n$ matrix with n different symbols, where each symbol occurs once per row and per column in the matrix (Tommi et al., 2007).Lattice Graphs are categorized under Latin Square Graphs. Its vertices are the m^2 elements of a Latin square of order m and there is an edge between two vertices if and only if they are in the same row or column (Kundeti et al., 2021). Lattice graphs are strongly regular graphs with the parameters of $n=m^2$, $\lambda=m-2$ and $\mu=2$ (Ayeh et al., 2009).Triangular Graphs are another kind of graphs which are categorized under the group of Strongly Regular Graphs.They are vertex transitive and have large automorphism groups (Tommi et al., 2007).In study, Miyazaki constructed regular graphs in order to force “nauty” to work in exponential time (Miyazaki et al., 1997). He also proved that his constructions were very hard for canonical labeling based isomorphism programs. Miyazaki showed that the choice of target cell had a significant effect on the performance of the canonical labeling based isomorphism programs. An r-Regular Graph is a graph in which all vertices have the degree of r. A random r-regular graph is selected from the set of all r-regular graphs. An n vertices r-regular graph has the properties where 3<=r<n and n*r are even.

IV. EXPERIMENTS AND RESULTS

Our main objective in this thesis is observing the effects of different target cell selection algorithms on difficult graphs for Graph Isomorphism Problem. To achieve that, we conducted several experiments with different target cell selection algorithms on “Traces” Graph Isomorphism Package.All the experiments were conducted on a 64-bit HP notebook laptop that has 8 GB main memory, 2200 MHz AMD A8 processor and Windows 10 as the operating system.As previously mentioned in Section 2.4.3, Traces chooses the first largest non-singleton cell as the target cell which is a subset of the target cell in the parent node.In this thesis we decided to test different target cell selection algorithms rather than those “Traces” uses by default. These target cell selection algorithms were largest target cell selection, smallest target cell selection, max-join target cell selection algorithm and max-cell target cell selection algorithm.In the next sections we will demonstrate the implementation strategies for these target cell selection algorithms.To do that, first we will talk about the data structures of “traces”which we have taken advantage of while implementing the target cell functions. Then, we will provide the experimental results we acquired.

Data Structures of Traces

In “traces”, the vertices of a graph with “n” vertices are numbered from 0 to n-1. To specify the colors of the vertices of a partition, two arrays are used. The first array is named as “lab”. The first index of the “lab” array starts from index 0 and it holds the vertices in some order.
For example, if we have a partition with 6 vertices, the “lab” array may look as in the follows:

Table 2
The "lab" array

4	2	5	0	1	3

Another important data structure in “traces” is the “ptn” array. The “ptn” array in “traces” used to specify the colors. The “ptn” array holds the value of 0 to indicate the ending position of each cell.If ptn[i]=0 for some i, it can be said that a cell ends at position i. The other entries of the array may be any other number except 0. For instance, assume that the array below is the “ptn” array of a partition whose “lab” array can be seen on Table3. Then, by considering the “lab” and “ptn” arrays together, partition can be represented as: {{4},{2},{0,1,5},{3}}.

Table 3
The "ptn" array

0	0	1	3	0	0

The other data structures that were used during our target cell selection function implementations were the arrays of “d,v and e. For each vertex d[i] hold the degree information of the vertex i. v[i] is an index into array “e” such that e[v[i]],e[v[i]+1],. . .,e[v[i]+d[i]-1] are the vertices to which vertex i is joined . The neighbor list can exist in array “e” in any order.

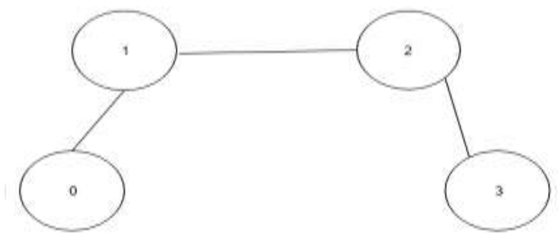

Figure 11. A Simple Graph with 4 Vertices

If we consider a simple graph with 3 vertices such as in the figure above, the “d”,”v” and “e”arrays look like as in the follows:
Table 4

Simple graph with 3 vertices Array

d:	1	2	2	1		
v:	0	1	3	5		
e:	1	2	0	3	1	2

As we can see from the arrays above, degree of vertices 0 and 3 are 1, 1 and 2 are 2. Also as it can be understood from the “v” and “e arrays, neighbor list The neighbor list of the vertices 0,1,2 and 3 starts at the indices 0,1,3 and 5 respectively. The last data structure that we have used in our implementations was the “cls” array.The cls[i] is a number which indicates the size of the cell that begins at index i. For example, for the partition |2 1 4 | 0 3 | cls[0]= 3 cls[3]=2.

Largest Target Cell Selection Function

We started our target cell selection function implementations with the largest target cell selection algorithm.

Simply, our target cell function takes an equitable partition and its level in the search tree as inputs and returns the selected target cell’s index as output.To implement this target cell selection function, we used two different data structures of “Traces” package, which are “cls” and “lab” arrays.Assume that there are 100 vertices in a partition and that there are three cells with the sizes of 15, 35 and 50. Assume that the largest cell is the last cell. Then, since vertices are numbered starting from 0 in “traces”, returned index must be 49. Other than the cell size, we experimented withthe degree of the vertex, which is the first vertex of the target cell. Results showed that selecting the largest target cell with a size that is strictly greater than 2 provides slightly better outcomes for most of the graph types we tested on. The basic part for of this target cell selector.

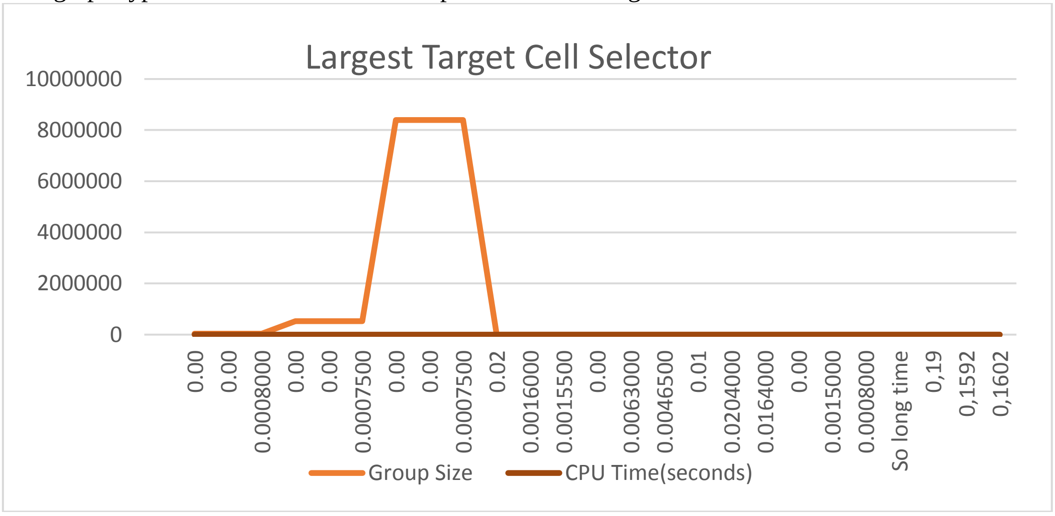

Figure 12. Performance of the largest target cell selector

Smallest Target Cell Selection Function

The second target cell selection function we implemented is the “smallest target cell selection function”. In order to implement this function we used the same data structures as we used for largest target cell selector.

This time we applied two different strategies. The first one was comparing the cell sizes and selecting the smallest cell. The second one was size based search. We started to search for cells with size 2. Once found, we returned the index number of the first cell. If not found, we searched for cells with size 3 and so on.

We observed that the first strategy was not working effectively for most of the graphs we tested, while the second strategy worked for all the graphs, but its performance was visibly worse than the largest target cell selector.

Max-Join Target Cell Selection Function

Another target cell selection function we implemented is the “max-join target cell selector”. Which means selecting the cell with the maximum number of vertices neighboring with other cells.To implement “max-join target cell selector”, we benefited Traces’s data structures; “e” and “d” arrays. Depending on the individualization refinement method we know two vertices that have different degrees cannot exist in the same cell and the vertices which have same degrees can be in different cells. For instance the figure below, it can be observed that vertices 0,1,2 and 3 have the degree of 2, but the vertex 3 belongs to another cell.

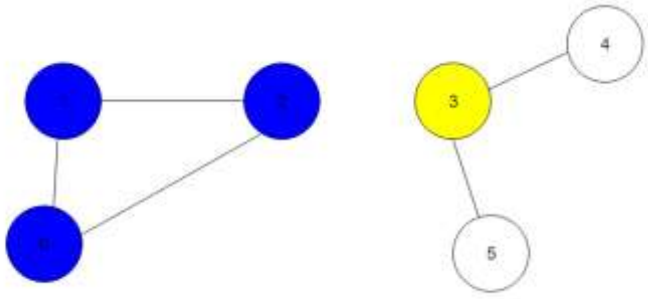

Figure 13. A colored graph that has multiple cells for the vertices with same degree.

To be able to distinguish which vertex belongs to which cell we have taken advantage of “d” and “e”arrays of the traces. Firstly, we have conducted several experiments with “e” array to be able to understand at which index neighbor list of each vertex start. We have noted that the starting index of neighbor lists increase regularly according to labelling of vertices. Consider a cell with size of 3. Assume that this cell is not joined to any other cell, which means that vertices of the cell are only adjacent to the vertices of the same cell. Then we can imagine this graph as the leftmost partition on Figure 19 (dark blue vertices). If every vertex were only neighbors with the vertices within its own cell,then we can expect that the neighbor list of the vertex 0 starts from index 0 and at index 4. Neighbor list of vertices 1 starts from index 4 and ends at index 8 and so on. Assume that for the same cell, starting index of the neighbor lists of the vertices are 0, 4 ,20. In this case the leftmost cell is |0,1,5| , in reality. Considering the individualizing-refinement procedure we have concluded this as a sign of a more connected partition and concluded that this cell is possibly adjacent with other cells. The partition in Figure 20 can be considered as a possible example to this idea.

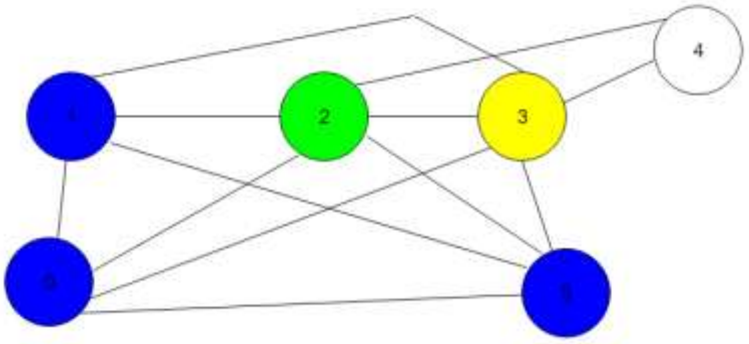

Figure 14. Partition of |0 1 5 | 2 | 3 | 4|

According to our assumption explained above, to implement the max-join target cell selector, firstly we have assigned a counter to 1 for each cell, traversed the vertices of each cell one by one and calculated the difference between the first indices of neighbor lists for consecutive vertices. Then we have compared the difference with the degree of the first vertex. If the degree of the first vertex is not same with the difference, we assumed that the cell is joined to another cell (as explained in the above paragraph) and increased the counter by 1. Finally, we have selected the cell with the maximum counter value and has a size strictly greater than 2.The part of the code that expresses the explained strategy is provided below:

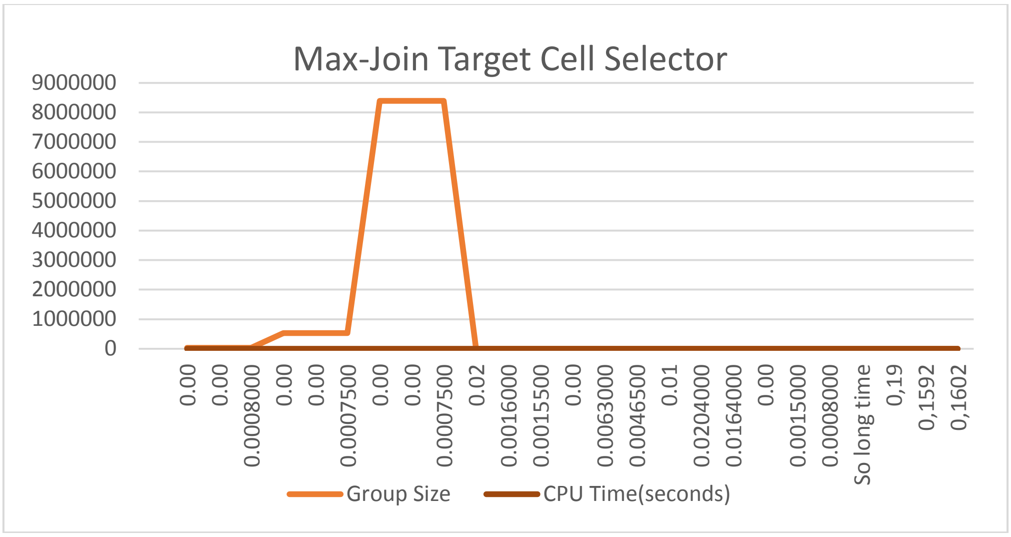

*Figure 15.*Performance of the max-join target cell selector

Max-Cell Target Cell Selection Function

The other target cell function we have implemented in this thesis was max refine target cell selector. The aim of this function was finding the cell which leads to maximum number of cells after individualizing each vertex of it.To implement this function we have taken advantage of the "cls "and "ptn" array. Firstly, we have created a new partition and copied the "lab" and "ptn" arrays of the current partition into it.Secondly, we have individualized each vertex in each cell starting from the first vertex of the cell. To do this we have move the individualized vertex to the end of the newly created partition's "lab" array.

For example, assume that for the partition |0 1 3 2 4| 6 5| 7 | the initial "lab"" array as as in the follows:

Table 4

Partition |0 1 3 2 4| 6 5| 7 | the initial "lab"" array

Lab:	0	1	3	4	2	5	6	7

As we can understand from the partition the cls[0]=5, cls[5]=2 and cls[7] =1.

After individualization, according to our implementation, the "lab" array of the newly created partition after individualization (lab2) should look like as in Table 5.

When it comes to "ptn" array, keeping in mind the explanation of it in section 5.1, the "ptn" array changes as shown in Table 6.

After arranging the lab and ptn arrays according to our individualization strategy, we have called the refinement function which was already implemented in "traces". Finally, refinement function returns a partition to our target cell selection function after performing refinement operation. Then, we have returned the index of the target cell which leads to maximum number of cells after individualizing its vertices and has a degree of strictly greater than 2. As shown in Figure 16 .

Table 5

The "lab" array of the newly created partition after individualization

Lab2:	7	1	3	4	2	6	0

Table 6

Changed "ptn" Array

Ptn(intial):	1	1	1	1	0	1	0	0
Ptn2(after individualization):	1	1	1	0	0	1	0	0

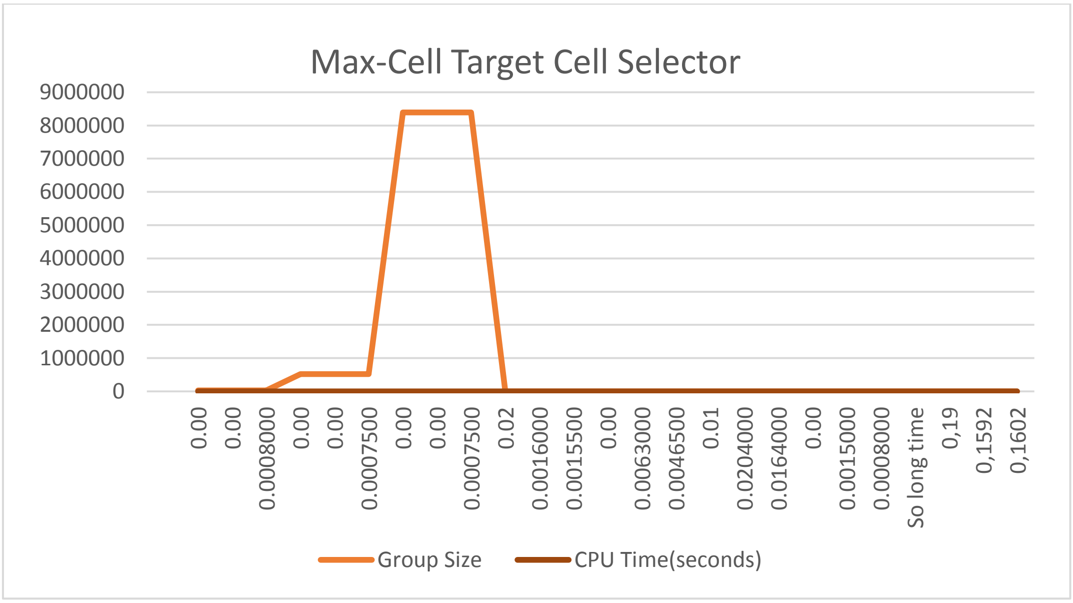

Figure 16. Performance of the max-cell target cell selector

V. CONCLUSION

The experiments we have conducted until now shows that the “largest” target cell selector performs better in terms of CPU time for most of the graphs which have an group size greater than 2400. We have also observed that for big hadamard graphs which have a group size greater than 19200 the “largest” target cell selector leads a smaller number of refinement operations compared with max-join and max-cell target cell selectors.Due to our experimental results, it can also be said that, almost for all of the miyazaki graphs that have been tested, all of the three target cell selectors showed similar performances but in terms of CPU time the “largest” target cell selector performed better than the others due to simplicity of its programming logic.

We have also observed that especially for bigger latin square and hadamard graphs, max-join target cell selector leads to a greater number of refinement operations compared to “max-cell” and “largest” target cell selectors.When it comes to max-cell target cell selector, we have observed that for bigger latin square graphs which have group size greater than 10800, it leads to less number of refinement operations compared with the other target cell selectors. Although it may lead to less number of refinement operations, we have observed that in means of CPU time it was the worst performing selector. The reason for this may be the complexity of the programming logic of the function. Also, we saw that max-cell target cell selection function leads to reduce the search tree depth for most of the graph types. Also, we think that there exist future works to do. First of all we have just conducted experiments with three different target cell selectors and in this thesis we did not consider the construction ways of the graph types while implementing these target cell selectors. An interesting future work may be conducting more experiments with different target cell selectors and then considering both experimental results and construction way of the graphs, implementing new target cell selectors for each graph type.

REFERENCES

[1]. Alfred, V. Aho., John, E. Hopcroft.,& Jeffrey, D. Ullman. (1974). *The design and analysis of computer algorithms*. Pearson Education, Inc and Dorling Kindersley.https://www.pearson.com/uk/educators/higher-education-educators/program/Aho-Design-and-Analysis-of-Computer-Algorithms-The/PGM405939.html

[2]. Ayeh,Eric. (2009, December). *An investigation into Graph Isomorphism based zero knowledge proofs*. [Master dissertation, University of North Texas]. https://digital.library.unt.edu/ark:/67531/metadc12076

[3]. Krena, Bohuslav. (2001). The Graph Isomorphism Problem. In *Proceedings of 7th Conference Student FEI, 2001*. [Doctoral dissertation, Brno University of Technology].http://www.fit.vutbr.cz/~krena/prace/stc2001.pdf

[4]. Eugene, M. Luks. (1980). Isomorphism of graphs of bounded valence can be tested in polynomial time. In *Proceedings of the 21st Annual Symposium on Foundations of Computer Science* (*SFCS '80*). IEEE Computer Society, USA, 42–49. DOI:https://doi.org/10.1109/SFCS.1980.24

[5]. Greg, Tener. (2009). *Attacks on difficult instances of graph isomorphism: sequential and parallel algorithms*. [Doctoral dissertation, University of Central Florida]. *Electronic Theses and Dissertations*, 2004-2019. 4004.https://stars.library.ucf.edu/etd/4004

[6]. Hopcroft, J. E. & Wong J. K. (1974). Linear time algorithm for isomorphism of planar graphs (Preliminary Report). In *Proceedings of the sixth annual ACM symposium on Theory of computing (STOC '74).* Association for Computing Machinery, New York, NY, USA, 172–184. DOI: https://doi.org/10.1145/800119.803896

[7]. Kocay W. (1996) *On Writing Isomorphism Programs. In: Wallis W.D. (eds) Computational and Constructive Design Theory*. Springer, Boston, MA. https://doi.org/10.1007/978-1-4757-2497-4_6

[8]. Kundeti, Vamsi. (2021). Algorithms for computing the Automorphism Group of a Graph and Sub Graph Isomorphism.

[9]. Kutz, M., & Schweitzer, P. (2007). ScrewBox: a Randomized Certifying Graph-Non-Isomorphism Algorithm. ALENEX.

[10]. McKay, B.D., & Piperno, A.(2016). Practical Graph Isomorphism, II.*Journal of Symbolic Computation, 60 (2014), pp. 94-112.*https://doi.org/10.1016/j.jsc.2013.09.003

[11]. Miyazaki, T. (1997). The complexity of McKay's canonical labeling algorithm.*DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. Volume 28. http://www.cs.trincoll.edu/~miyazaki/piscataway.pdf

[12]. Presa, J.L.L. (2009). *Efficient algorithms for graph isomorphism testing*. [Doctoral dissertation, Universidad Rey Juan Carlos, Madrid]. https://www.etsist.upm.es/uploaded/docs_personales/lopez_presa_jose_luis/tesis/thesis.pdf

[13]. Somkunwar, R., & Moreshwar, V. (2017). A Comparative Study of Graph Isomorphism *Applications. International Journal of Computer Applications*, 162(7), 34-37. doi:10.5120/ijca2017913414

[14]. Junttila, T., & Kaski, P. (2007). Engineering an efficient canonical labeling tool for large and sparse graphs. In *Proceedings of the ninth Workshop on Algorithm Engineering and Experiments and the fourth Workshop on Analytic Algorithmics and Combinatorics* (pp. 135-149)