Acta Universitatis Sapientiae

Informatica

Volume 11, Number 1, 2019

Sapientia Hungarian University of Transylvania
Scientia Publishing House

Acta Universitatis Sapientiae, Informatica
is covered by the following services:

DOAJ (Directory of Open Access Journals)
EBSCO (relevant databases)
EBSCO Discovery Service
io-port.net
Japan Science and Technology Agency (JST)
Micosoft Academic
Ulrich's Periodicals Directory/ulrichsweb
Web of Science – Emerging Sources Citation Index
Zentralblatt für Mathematik

Contents

Acta Univ. Sapientiae, Informatica **11,** 1 (2019) 5–23

DOI: 10.2478/ausi-2019-0001

Estimating the maximum rise in temperature according to climate models using abstract interpretation

Peter Z. REVESZ
University of Nebraska–Lincoln
Department of Computer Science & Engineering
Lincoln NE 68588-0115, USA
email: revesz@cse.unl.edu

Robert J. WOODWARD
University of Nebraska–Lincoln
Department of Computer Science & Engineering
Lincoln NE 68588-0115, USA
email: rwoodwar@cse.unl.edu

Abstract. Current climate models are complex computer programs that are typically iterated time-step by time-step to predict the next set of values of the climate-related variables. Since these iterative methods are necessarily computed only for a fixed number of iterations, they are unable to answer the natural question whether there is a limit to the rise of global temperature. In order to answer that question we propose to combine climate models with software verification techniques that can find invariant conditions for the set of program variables. In particular, we apply the constraint database approach to software verification to find that the rise in global temperature is bounded according to the common Java Climate Model that implements the Wigley/Raper Upwelling-Diffusion Energy Balance Model climate model.

1 Introduction

The ability to predict climate change, which has potentially a huge impact on life on earth, is affecting the legislation of countries and their mitigation efforts

Computing Classification System 1998: G.2.2
Mathematics Subject Classification 2010: 68R15
Key words and phrases: constraint database, Datalog, climate model, invariant, MLPQ system, software verification

around the world [8]. The predictions of the impacts of climate change rely heavily on the simulations of global climate models. Regional climate models offer a finer level of detail than the global climate models, and are sometimes used to determine the impact of climate on smaller regions. Climate models are calibrated using historical weather data. The model scenarios have been standardized by the *Intergovernmental Panel on Climate Change (IPCC)*, which was established in 1988 by the World Meteorological Organization and the United Nations Environment Programme.

The IPCC used several global climate models in its Third Assessment Report (TAR) [8] and its successor the Fourth Assessment Report (AR4) [2]. The TAR and AR4 assessment reports continue to be updated to include new information and research conducted since their dates. Chapter 9 of [8] defines that climate change simulations are to be assessed over the period from 1990 to 2100.

Current climate models, including the ones in TAR and AR4 [8, 2], are computer programs that use iterative methods to compute the values of climate variables, such as the rise in global average temperature above a baseline year, one year at a time for a fixed number of iterations. The computer programs become nonterminating when we drop the restriction of a fixed number of iterations. Nevertheless, we need to drop the restriction of a fixed number of iterations if we want to ask some of the most basic questions about climate change, such as "Will the global average temperature rise without a bound?" Saying that under a certain scenario of carbon emissions, the global average temperature will rise only one degree during the next twenty-five, fifty or seventy-five years is not satisfying. Our generation cannot claim to have found a sustainable, long-term solution, even a model of a solution, to the problem of climate change if the global average temperature rise cannot be bounded. The goal of this paper is to determine if there is a maximum invariant value for the global average temperature change from a baseline using software verification techniques.

This paper is organized as follows. Section 2 reviews some basic concepts, including the constraint database approach to software verification and the Java Climate Model used by the IPCC. Section 3 describes the climate model's implementation in the MLPQ constraint database system. Section 4 discusses the implications of the results. Section 5 summarizes related work. Finally, Section 6 gives some conclusions and future work.

2 Basic concepts

Next we review some concepts of the constraint database approach to software verification [16] and on climate models [8, 2].

2.1 Addition-bound matrixes or ABMs

Addition-bound matrixes, or ABMs, are designed to represent a set or conjunction of addition constraints and (lower and upper) bound constraints. The following definitions are based on the standard textbook description by Revesz [17].

Any set of addition, lower bound and upper bound constraints over the variables $V = \{x_1, \ldots, x_n\}$ is representable by a set of difference constraints over variables $V^+ = \{x_1^+, x_1^-, \ldots, x_n^+, x_n^-\}$. Note that in the difference constraint representation each variable x_i has two forms, namely a positive one, which is denoted by x_i^+ and a negative one, which is denoted by x_i^-. Here the first form is equivalent to x_i, while the second form is equivalent to $-x_i$. The logical equivalences shown below explain the rewriting of the constraints over V into constraints over V^+.

$$\begin{aligned} -x \geq b &\quad\equiv\quad x^- - x^+ \geq 2b \\ x \geq b &\quad\equiv\quad x^+ - x^- \geq 2b \\ x - y \geq b &\quad\equiv\quad x^+ - y^+ \geq b \\ x + y \geq b &\quad\equiv\quad x^+ - y^- \geq b \\ -x - y \geq b &\quad\equiv\quad x^- - y^+ \geq b \\ -x + y \geq b &\quad\equiv\quad x^- - y^- \geq b \end{aligned}$$

After applying the above rewriting rules, it may happen that we have two constraints $x - y \geq b$ and $x - y \geq c$. Suppose without loss of generality that $b > c$. Then $x - y \geq c$ can be deleted because it is implied by $x - y \geq b$. After similarly deleting all constraints that are implied by other constraints, for each pair of variables x and y, there can be only one difference constraint with $x - y$ on the left side.

Therefore any set of addition, lower bound and upper bound constraints over V is representable by an *ABM* A with rows and columns labeled by the elements of V^+. Further, the $A[i, j]$ entry of this ABM contains the right side constant of the difference constraint associated with the ith row and the jth column labels.

Next we show on an example set of constraints how it can be rewritten into an ABM. Suppose we have:

$$-x \geq -25,\ y \geq 3,\ x - y \geq 4,\ x + y \geq 10,\ -x - y \geq -40$$

then by using the above rewriting rules and simplifications, it can be represented by the following set of difference constraints:

$$x^- - x^+ \geq -50,\ y^+ - y^- \geq 6,\ x^+ - y^+ \geq 4,\ x^+ - y^- \geq 10,\ x^- - y^+ \geq -40$$

Finally, the ABM A below can represent the above set of difference constraints.

	x^+	x^-	y^+	y^-
x^+	$-\infty$	$-\infty$	4	10
x^-	-50	$-\infty$	-40	$-\infty$
y^+	$-\infty$	$-\infty$	$-\infty$	6
y^-	$-\infty$	$-\infty$	$-\infty$	$-\infty$

2.2 Operations on ABMs

Below we review the main ABM operators [17] that are used in later sections.

Definition 1 *Given two ABMs* A *and* B*, the minimum of* A *and* B*, denoted by* $A \vee B$*, is:*

$$[A \vee B]\,[i,j] = \left\{ \begin{array}{ll} A[i,j] & \text{if } A[i,j] \leq B[i,j] \\ B[i,j] & \text{if } B[i,j] < A[i,j] \end{array} \right\}$$

Definition 2 *Given two ABMs* A *and* B*, the widening of* A *and* B*, denoted by* $A \triangledown B$*, is:*

$$[A \triangledown B]\,[i,j] = \left\{ \begin{array}{ll} A[i,j] & \text{if } A[i,j] \leq B[i,j] \\ -\infty & \text{if } B[i,j] < A[i,j] \end{array} \right\}$$

Definition 3 D *is a domain of an ABM* A *if each entry* $A[i,j] \in D$*. When for some integer constants* l *and* u *each* $A[i,j]$ *is greater than or equal to* l *and less than or equal to* u *or is equivalent to* $-\infty$*, then* $\{-\infty\} \cup \{l, l+1, \ldots, u-1, u\}$ *is a domain of* A*.*

Definition 4 *Let* $l < 0$ *and* $u > 0$ *be two integer numbers and let* A *be an ABM with domain* $\{-\infty\} \cup \{l, l+1, \ldots, u-1, u\}$*. Given also another ABM* B*, the l-u-widening of* A *by* B*, denoted by* $A \diamondsuit_{l,u} B$*, is:*

$$[A \diamondsuit_{l,u} B]\,[i,j] = \left\{ \begin{array}{ll} A[i,j] & \text{if } A[i,j] \leq B[i,j] \\ B[i,j] & \text{if } l \leq B[i,j] < A[i,j] \\ -\infty & \text{if } B[i,j] < l \leq A[i,j] \end{array} \right\}$$

Example 5 (Revesz [17]) *Consider again* A *at the end of Section 2.1 and also the following ABM* B*:*

	x^+	x^-	y^+	y^-
x^+	$-\infty$	$-\infty$	15	10
x^-	-60	$-\infty$	$-\infty$	$-\infty$
y^+	$-\infty$	7	$-\infty$	2
y^-	$-\infty$	$-\infty$	$-\infty$	$-\infty$

Here $A \vee B$ *is:*

	x^+	x^-	y^+	y^-
x^+	$-\infty$	$-\infty$	4	10
x^-	-60	$-\infty$	$-\infty$	$-\infty$
y^+	$-\infty$	$-\infty$	$-\infty$	2
y^-	$-\infty$	$-\infty$	$-\infty$	$-\infty$

and $A \triangledown B$ *is:*

	x^+	x^-	y^+	y^-
x^+	$-\infty$	$-\infty$	4	10
x^-	$-\infty$	$-\infty$	$-\infty$	$-\infty$
y^+	$-\infty$	$-\infty$	$-\infty$	$-\infty$
y^-	$-\infty$	$-\infty$	$-\infty$	$-\infty$

Finally, $A \Diamond_{-50,50} B$*, that is when* $l = -50$ *and* $u = 50$*, gives:*

	x^+	x^-	y^+	y^-
x^+	$-\infty$	$-\infty$	4	10
x^-	$-\infty$	$-\infty$	$-\infty$	$-\infty$
y^+	$-\infty$	$-\infty$	$-\infty$	2
y^-	$-\infty$	$-\infty$	$-\infty$	$-\infty$

In addition to the above operators, we also consider the union operator $\cup$ of two ABMs. When A and B are AMBs, then the union operator $A \cup B$ simply returns the set of constraints that either A or B contains. The following theorem from [17] shows the relationship among the different AMB operators.

Theorem 6 (Revesz [17]) *Let* $\mathcal{S}$ *be the set of assignments to the variables that satisfy all the constraints of an ABM or union of ABMs. For any* $l < 0$ *and* $u > 0$*, the following holds:*

$$\mathcal{S}(A \cup B) \subseteq \mathcal{S}(A \vee B) \subseteq \mathcal{S}(A \Diamond_{l,u} B) \subseteq \mathcal{S}(A \triangledown B).$$

2.3 Abstract fixed point semantics

Each procedural program has a *collecting semantics*, which consists of a set of em invariants. Each invariant is associated with a line l in the procedural program and is intended to describe all possible values of all the variables when the program enters line l. Software verification is based on finding an over-approximation of the collecting semantics.

A general method to compute an over-approximation is called *abstract interpretation*. Abstract interpretation evaluates the procedural program by an *abstract execution* that starts with some abstract representation of the input data. The abstract execution at each entry of line l generalizes the invariant associated with l using a *widening operator* until the line invariant cannot be further widened.

A widening operator generalizes at a program location an invariant constraint A with some constraint B that describes an additional set of possible values of the program variables at that location. There are different types of widening operators proposed by various authors.

When we use the widening operator $A \Diamond_{l,u} B$, then it always leads to a terminating program execution. Note that $A \cup B$ is not a suitable widening operator because it may lead to a non-terminating abstract program execution.

Theorem 7 *Let P be a program with n integer (or rational) variables, only addition bound constraints on these variables, and k lines. Let l and u be two constants, and let an addition-bound matrix A_i be assigned to each line $1 \leq i \leq k$ of the program. Let each A_i contain no constraints initially, and as we execute line i of program P, widen A_i by the constraints B implied in line i using the widening operator $A_i \Diamond_{l,u} B$. Then the abstract program execution will terminate.*

Programs with integer and rational variables, if statements, go to statements, while statements, and assignment statements, where a variable is assigned the value of a linear arithmetic expression, can be represented as a Datalog program with constraints [9, 15]. The abstract program execution finds an *abstract fixed point* semantics, which will contain the least fixed point semantics [17]. The containment allows us to answer some questions about the possible values that variables in the program could take.

One can compute an abstract fixed point semantics of any climate change model that is equivalent to a complex computer program that would not terminate under normal program execution. If the abstract fixed point semantics of that computer program does not contain the possibility that the global

temperature reaches x degrees Fahrenheit, then we can conclude that according to that model the global temperature will not reach x degrees Fahrenheit. However, if the abstract fixed point semantics *contains* x as a possibility, then we cannot conclude anything definite because the abstract fixed point semantics may be an over-approximation of the least fixed point semantics, which actually does not contain x as a possibility.

The constraint database approach to software verification [16], which is a novel way to perform an abstract interpretation [4], uses the above idea to verify that a program functions correctly on a valid input by avoiding certain program states, where a program state is the values assigned to the variables in the program at a specific line of the program code [1, 16]. The Management of Linear Programming Queries (MLPQ) database [19, 1] is a constraint database that implements the above described widening operator and can be applied to Datalog programs with addition constraints. Hence we need to convert any computer program to a Datalog with addition constraint program as part of the constraint database approach to software verification.

2.4 Climate models

A climate system is a physical system that consists of five major components. The first component is the *atmosphere*, which is the air and space surrounding the earth. The second component is the *hydrosphere*, which is the water surrounding the earth. The hydrosphere is an important component because oceans form about two-third of the earth's surface. The third component is the *cryosphere*, which consists of the parts of the earth where water is frozen. This needs to be tracked separately from the oceans because the oceans and the cryosphere have very different physical properties in terms of absorption and reflection of sun light. The fourth component is the *land surface*, which is the part of the earth that is covered by land. The fifth and final component is the *biosphere*, which is the parts of the earth covered by living organisms.

Climate models try to model the climate system and predict some values for the climate, such as the following:

1. Land-surface temperature and land-surface air temperature.
2. Sea-surface temperature and ocean air temperature.
3. Land and sea combined temperature.
4. Sub-surface ocean temperature.
5. Upper air temperature.
6. Snow cover, including snowfall.

7. Sea-ice extent and thickness.

A good model considers all the possible types of interactions among the components. For example, the biosphere affects the concentration of carbon dioxide in the atmosphere [8]. Figure 1 shows a schematic diagram of a climate model.

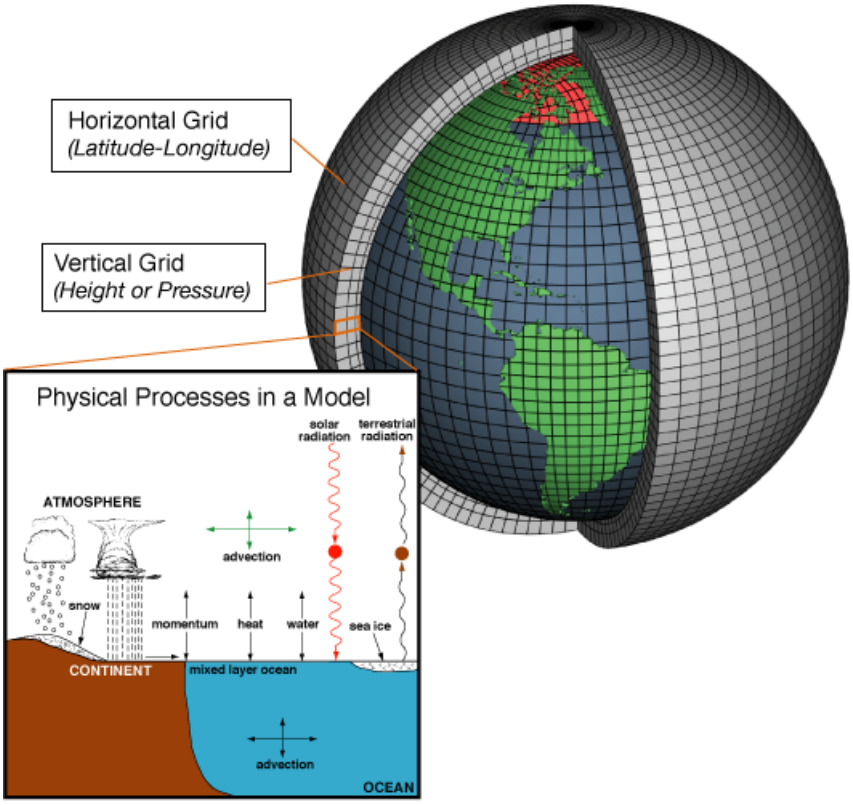


Figure 1: Some elements of a climate model from the Wikipedia entry "General Circulation Model."

A report of the UN's Intergovernmental Panel on Climate Change (IPCC) outlines some of the ways to accurately predict the above values [8]. The *Wigley/Raper Upwelling-Diffusion Energy Balance Model (UD/EBM)* climate model is a simple climate model that differentiates the hemispheres, and the land and ocean regions in each hemisphere [12]. The model uses heat flux equations to model the transfer from one year to the next and from one re-

gion to another. The UD/EBM climate model must be tuned to simulate an *atmosphere-ocean coupled general circulation model (AOGCM)*, without which it is not a complete model [8]. This combined model can iteratively compute each year's value, and the computation can be repeated without any termination.

The Java Climate Model (JCM)[1] implements the UD/EBM and was properly tuned to match a AOGCM [11]. Rather than using direct integration to compute the values for the heat fluxes, the JCM uses an eigenvector calculation method. This method finds the exact analytical solution, given the assumption that the non-linear fluxes change linearly within one time-step of a year [11].

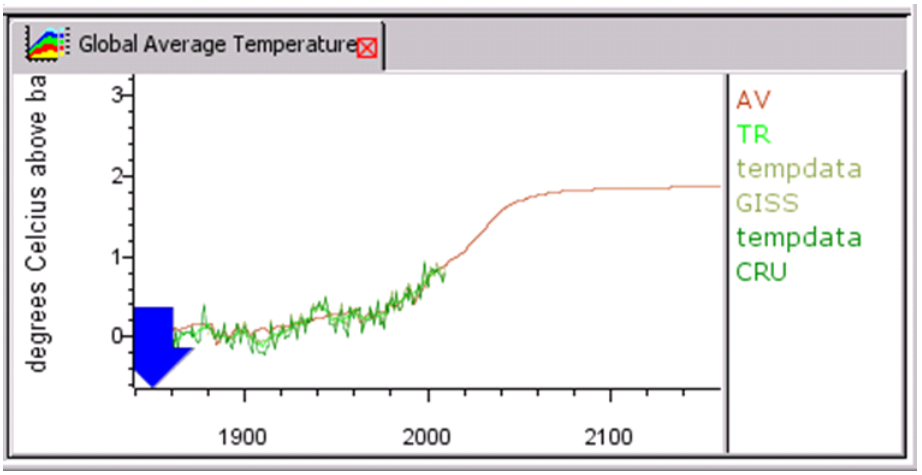


Figure 2: The global average temperature change given by the Java Climate Model.

The JCM was downloaded from http://jcm.climatemodel.info/. The SVN code repository for JCM was not operational, however, the source code was included inside of the distributed Java Archive (JAR) file. After extracting the source folders from the JAR file, a new project was created in NetBeans IDE 7.0.1 (`http://netbeans.org/`) and the source folders were imported. We needed to set up the following libraries:

- substance.jar – included in the JCM JAR.
- lucdata.jar – included in the JCM JAR.
- labdoc.jar – included in the JCM JAR.

[1] `http://jcm.climatemodel.info/`

- match-emitdata.jar – included in the JCM JAR.
- JCM.jar – included in the JCM JAR.
- javaws.jar – included in the Jave Runtime Environment (JRE) library folder.
- Jama-1.0.2.jar – downloaded from `http://math.nist.gov/javanumerics/jama/`.

The method that computes the average temperature change iteratively, one year at time, for the JCM is the ADJUST method, inside 'udebclimod.java,' and is shown in Algorithm 1. Line 1 of Algorithm 1 contains a for-loop to compute the global average temperature change for each year. The initial values of the variables used in ADJUST are set up in the SETUPFLUXES method, which is not shown here as their computation is not relevant. These intitial values were used as constants in our approach, which is discussed in more detail in Section 3.1. Figure 2 shows the predicted global average temperature change for each year given by the Java Climate Model until 2150. Note that the model seems to level off at a value around 2. However, there is no guarantee that the value is will leveled off at 2 or will spark later according to the model. Hence it is an important open question whether 2 is the maximum value. We will try to determine that in the rest of this paper.

3 The climate model's implementation in the MLPQ system

The goal of the experiment is to determine an invariant value on the average temperature change above a baseline year. Typically the value of the average temperature change above a baseline year is computed between 1990 and 2100 [8]. Instead, in our experiment the variant average temperature change above a baseline year will not depend on any year but will be an abstract fixed-point semantics upper-bound that will apply to all future years.

In this section, we first give an overview of how we will convert the ADJUST method from the Java Climate Model (JCM) code into Datalog to use with the MLPQ system, which can compute the abstract fixed point semantics to find all the possible values of the average climate temperature change. Second, we present the conversion process, showing the difference and gap-order constraints [13, 14]. Finally, we describe our implementation of the code using Datalog with constraints.

Algorithm 1: ADJUST

Input: numYears: Number of years in the future to compute the weather for
Output: The global temperature change computed for each year

```
for year = 0; year < numYears; year = year + 1 do
    guess = nstd + (nstd − nstdold);
    nstd = guess;
    nstdold = nstd;
    for o = 0; o < 2; o = o + 1 do
        for n = 0; n < nhb; n = n + 1 do
            hiq[o][n] = hpropf[o][n] ∗ hiq[o][n] + shicML[o][n] ∗ qinold[o];
        qinbase[o] = rf[o + 1];
        qinbase[o]+ = rf[o∗3]∗(frac[o∗3]/frac[o+1])∗klo/(kls∗frac[o∗3]+klo);
        qinbase[o]+ = spaceflux[o] ∗ qpt ∗ tstart;
    nit = 0;
    while |diff| > 0.01 && nit < 10 do
        for o = 0; o < 2; o = o + 1 do
            qin[o] = qinbase[o] + (o == 0? − 1.0 : 1.0) ∗ nstd ∗ kns/frac[o + 1];
            dqin = qin[o] − qinold[o];
            mlt[o] = 0;
            for n = 0; n < nhb; n = n + 1 do
                hiqi[o][n] = hiq[o][n] + rhicML[o][n] ∗ dqin; mlt[o]+ =
                hrML[o][n] ∗ hiqi[o][n];
            mlt[o]/ = qpt;
        diff = (mlt[0] − mlt[1]) ∗ cice − nstd; nstd+ = diff;
        nit = nit + 1;
    hiq = hiqi;
    qinold = qin;
    for o = 0; o < 2; o = o + 1 do mlt[o]− = tstart;
    bt[0][year] = mlt[0] ∗ cice ∗ klo + frac[0] ∗ rf[0])/(kls ∗ frac[0] + klo);
    bt[3][year] = mlt[1] ∗ cice ∗ klo + frac[3] ∗ rf[3])/(kls ∗ frac[3] + klo);
    bt[1][year] = mlt[0] ∗ cice;
    bt[2][year] = mlt[1] ∗ cice;
    globavtemp[year] = bt[0].get(year) ∗ frac[0] + bt[1].get(year) ∗ frac[1] +
    bt[2].get(year) ∗ frac[2] + bt[3].get(year) ∗ frac[3];
```

3.1 The experimental design

The code was examined to determine the constant values that do not change between iterations of the program (e.g., year-to-year). For the JCM, these values are typically the flux equations, or values that are specified by the user

to adjust the model to match an AOGCM. The default values were taken in this conversion to Datalog. Once these values were identified, because of the assumptions made by JCM to use the eigenvector calculation method instead of integration, all of the resulting equations were linear. Having linear equations was the goal of the model to study because MLPQ is a linear constraint database.

In the converted Datalog code, we refer to "linei" as the values of the variables at the start of line i of the program. Each line of the code was converted to Datalog to represent the change of values. These conversions are easy because we have only linear equations, For example, line 2 states the following:

```
guess=n+(n-nold)
```

and can be converted to Datalog:

```
line3(n,ndold,guess):- line2(n,ndold), guess=n+(n-ndold).
```

This Datalog code states that at the start of line 3, we are taking the same state as the start of line 2, except that we updated guess to have the value that is the value of the arithmetic expression on the right hand side of the equation in line 2 of the computer program.

The computer program we are converting from JCM contains two for-loops. The iterations of the two for-loops are independent from one-another. Since the method in JCM is called once for each year, after computing the last line in Datalog, we create a rule for the first line that propagates the values from the last line back as input. This creates the loop in the code that can then compute the abstract fixed point semantics. After the MLPQ system finished computing the values, we can look at the relation of the last line and see the possible values of the global average temperature change.

3.2 The coversion process

To give more details about the conversion process, we focus on the 29 lines of the ADJUST method in the JCM code (Algorithm 1), which was written in Java and consists of linear equations. The problem is that these linear equations are embedded in non-terminating for loops when we drop the condition which limits the numbers of years. In order to guarantee termination and execution in our Datalog with constraints program, we need to simplify the some parts of the computer program, where it contains either (1) global variables, or (2) large arrays. Lines 6 and 7 of the code offers a good example of these two types of simplifications:

for $n \leftarrow 0; n < nhb; n \leftarrow n + 1$ **do**
  $hiq[o][n] \leftarrow hpropf[o][n] * hiq[o][n] + shicML[o][n] * qinold[o];$

Global variables: The hpropf and shicML variables are both global variables, which are set from a different method call. These values are abstracted out as constants and loaded directly into the constraint database.

Large arrays: The for loop will iterate 40 times ($nhb = 40$, a constant) and thus the array for hiq[o] will have 40 entries. Having that many variables seems too prohibitive as a first-step towards modeling the program. Therefore, we restrict $nhb = 3$, and only have three values in the hiq[o] array.

Table 1 gives the gap-order constraints for Algorithm 1. In the table, the variable var_previous denotes the value of var from the previous line or previous iteration. Variables in all capital letters are treated as constants.

In a further simplification, we assumed that the ocean and the land areas for each the hemispheres took the same values. This simplification allowed the for-loops on line 5 and line 11 to be removed. However, this simplification was later removed by unrolling the content in the for-loops, once for each loop of the for-loop. The reason this simplification could be made was because the loops were independent from one another, which allowed the code to be unrolled.

Another simplification made, that is still in place, simplifies the for-loops of line 6 and 15 to only compute the first three values. Normally, these for-loops iterate over 40 values. All of the implementation details are in-place to remove this simplification.

3.3 A Datalog implementation

The simplified code was implemented in Datalog with the following steps: 1) allowing the insertion of constants into the Datalog program, 2) convert equations for MLPQ compatibility, and 3) allow more complicated arithmetic operations on constants (i.e., multiplication).

Prior to using the converter, we inserted constants into the Datalog program by fixing the variable assignment. For example, consider the constant $x = 123$:

```
CONST_X(x) :- 123
```

We found that using constants in this fashion over-complicated the program and caused significant overhead. Therefore, we wrote the converter such that

Line	Gap-Order Constraints
1	initialize rf, nstd, hiq, nstdold, qin, qinold
2	$guess - nstd - (nstd - nstdold) = 0$
3	$nstd - guess = 0$
4	$nstdold - nstd = 0$
5,6	For loop is unrolled
7	$hiq[o][n] - HPROPF[o][n] * hiq\_previous[o][n] - SHICML[o][n] * qinold[o] = 0$
8	$qinbase[o] - rf[o+1] = 0$
9	$qinbase[o] - qinbase\_previous[o] - (FRAC[o*3]/FRAC[o+1]) * KLO/(KLS * FRAC[o*3] + KLO) * rf[o*3] = 0$
10	$qinbase[o] - qinbase\_previous[o] * SPACEFLUX[o] * QPT * TSTART$
11	Not computed
12	Constraint posted on line 17
13	For loop is unrolled
14	$qin[o] - qinbase[o] - (o == 0? - 1.0 : 1.0) * KNS/FRAC[o+1] * nstd = 0$
15	$dqin - qin[o] + qinold[o] = 0$
16	$mlt[o] = 0$
17	For loop is unrolled
18	$hiqi[o][n] - hiq[o][n] - RHICML[o][n] * dqin \wedge mlt[o] - mlt\_previous[o] - HRML[o][n] * hiqi[o][n]$
19	$[1/QPT]mlt[o] = 0$
20	$diff - (mlt[0] - mlt[1]) * CICE + nstd = 0 \wedge nstd - nstd\_previous - diff = 0 \wedge diff > 0.001$
21	[not required]
22	$hiq - hiqi = 0$
23	$qinold - qin = 0$
24	$mlt[o] - mlt\_previous[o] = TSTART$
25	$bt[0] - ((CICE * KLO)/(KLS * FRAC[0] + KLO))mlt[0] - (FRAC0/(KLS * FRAC[0] + KLO))rf = 0$
26	$bt[3] - ((CICE * KLO)/(KLS * FRAC[3] + KLO))mlt[1] - (FRAC3/(KLS * FRAC[3] + KLO))rf = 0$
27	$bt[1] - CICEmlt[0] = 0$
28	$bt[2] - CICEmlt[1] = 0$
29	$globavtemp - FRAC[0] * bt[0] - FRAC[1] * bt[1] - FRAC[2] * bt[2] - FRAC[3] * bt[3] = 0$

Table 1: Conversion of Algorithm 1 to gap-order constraints.

it inserts the constants directly into the Datalog code. Note that we could have put the constants into the original Datalog program directly, but using a converter increases the readability of the code and gives us the ability to change the constants if required.

To accomplish multiplication, we tried to generalize an approach of multiplying two integer variables (See page 240 of [17]) to using floating point numbers, and first attempted to create a more general multiplication in Datalog as follows:

```
mult(x, y, z) :- y = 0, z = 0.
mult(x, y, z) :- y - y1 = 1, z - z0 - x = 0, mult(x, y1, z0).
```

However, in the above multiplication, where $x \times y = z$, the value of x is allowed to be a floating point number, but y is still required to be an integer. Since in some calculations both x and y need to be floating point numbers, we utilized the converter because all of our multiplications are on constants.

The converter has three parts:

1. A set of assignments used to convert constants to floating-point numbers. These assignments are stored in the 'ASSIGNMENTS' variable in the form 'VARIABLE=NUMBER'. VARIABLE is the text to search for, and NUMBER is a floating-point number that can be positive or negative. All fractions of numbers must have a leading '0' prior to the decimal point.

2. Converts double negation into a plus, for MLPQ compatibility. (E.g., $2 - -2$ becomes $2 + 2$.)

3. Evaluates arithmetic operations on numbers that are included in square brackets [ ]. This functionality allows more advanced arithmetic operations to be applied on constants (e.g., multiplication, division).

The steps in the converter could have been manually done when creating the Datalog file. However, the automated converter allows for more flexability when writing the Datalog code.

The converter script was created to work as a UNIX shell script. The script assumes a file named 'datalog.txt' is in the same working directory as the script, and will output a file named 'datalog_convert.txt' in the same working directory as the script. To run the script, simply type './convert.sh'. Note, the script requires the proper permissions set (e.g., chmod 700). **Note:** When using a Windows computer, the script might need to be converted not to have the Windows line returns (e.g., dos2unix convert.sh).

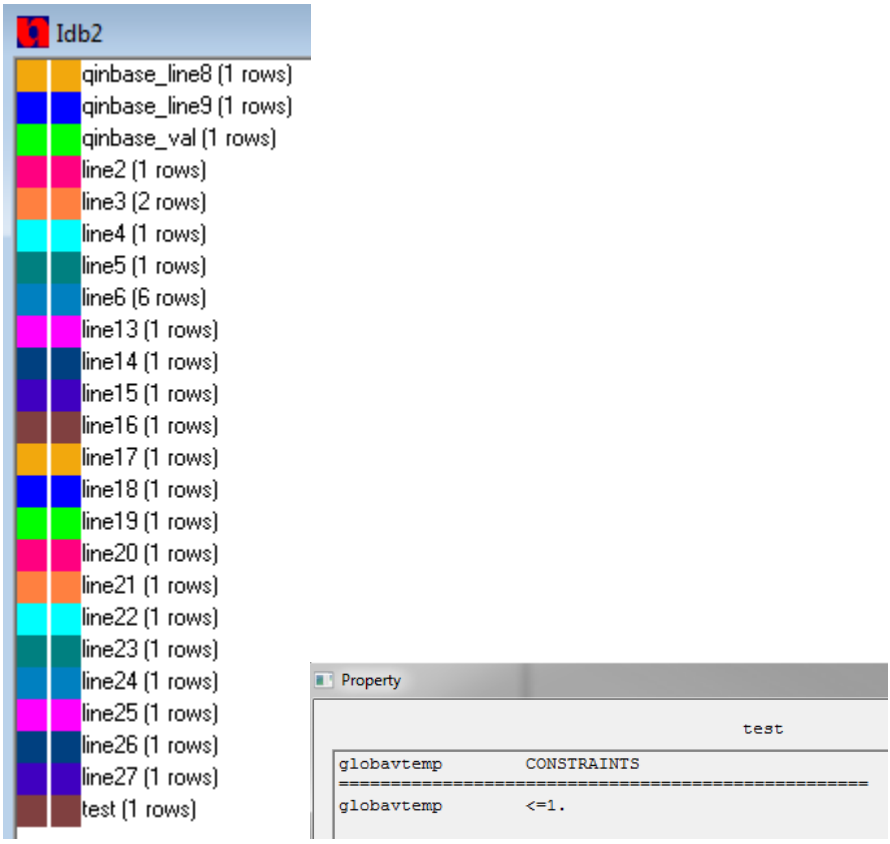


Figure 3: The relations loaded into MLPQ (left) and the resulting output of the global average temperature change above baseline of value one (right).

4 Discussion of the results

One of the problems we had early on when writing the Datalog code was not always determining when we had a typing error in one of the variable names in our code. MLPQ would then, correctly, interpret in the code the mistyped variable as a 'free-variable,' one that does not have any constraints on it. This problem caused errors early on in values not being computed properly. One way around this issue would be to use the Datalog Mode for Eclipse[2], which allows syntax highlighting for Datalog programs (but is not a Datalog interpreter).

[2] `http://suif.stanford.edu/~livshits/work/datalogeditor/`

Another struggle was getting MLPQ to properly load the relations. Some relations would take a huge amount of time for MLPQ to compute the value of the relation, which caused the program to look like it crashed. However, after waiting patiently, the program would load the relation. In order to get around this issue, we tweaked the order of the relations that were loaded and optimized the code by factoring out common code fragments and creating smaller relations.

In our tests, MLPQ returned the value of 1 for the bound on the global average temperature change as shown in Figure 3 for the results of the MLPQ system execution of the Datalog program. Although the idea of testing the long-range predictions of climate models using software verification is an intuitive and valid idea, the value of 1 should not be taken as conclusive because of the possible errors in the translation process from Java to Datalog and some simplifications we had to make to the original code. We need further tests of the algorithm to achieve confidence in its correctness and conclusions. Nevertheless, our experiment shows the soundness of the constraint database approach to being able to compute invariants for climate models.

5 Related work

There are a growing number of climate models. For example, the Intermediate Global Circulation Model (IGCM) (`http://www.met.rdg.ac.uk/~mike/dyn_models/igcm/`) implementing the baroclinic model of Hoskins and Simmons [7] and the Earth System Modeling Framework (ESMF) [3, 6] (`http://www.earthsystemmodeling.org`) are other climate models that are more complex than JCM.

A preliminary version of this paper was presented at [20]. To the best of our knowledge, no other researcher has previously attempted to compute an invariant value for any climate model. In fact, the calculation of invariants is not even considered in Chapter 9 of [8], where all simulations arbitrarily end at year 2100.

6 Conclusions and future work

This paper made the first attempt to investigate whether the climate models contain any inherent bounds. The paper combined climate modeling and software verification techniques, in particular software verification using the constraint database approach [1, 16]. Software verification techniques are able

to answer for even nonterminating computer programs what are the minimum and the maximum bounds on the variables.

The idea of combining climate models with abstract interpretation software verification techniques is a general contribution that is applicable to other climate models and other software verification techniques. Since the primary aim of our paper was only to show the feasibility of applying software verification techniques to testing the long-range implications of climate models, we started with the simpler JCM model. It remains a future work to investigate other combinations, for example using the IGCM or the ESMF climate models. Instead of being globally oriented like the JCM, some of the other climate models predict the future climate at a set of specific locations. These more refined climate models with values at specific locations at specific times may be also combined with spatio-temporal interpolation methods [5, 10, 18, 21] to generate temperature surface equations over each point of the globe.

References

[1] S. Anderson, P. Z. Revesz, CDB-PV: A constraint database-based program verifier, *Proc. of the 7th International Symposium on Abstraction, Reformulation and Approximation*, LNCS 4612, Springer, 2007, pp. 35–49. ⇒11, 21

[2] L. Bernstein et al., *Climate Change 2007: Synthesis Report*, Cambridge University Press, 2007. ⇒6, 7

[3] N. Collins et al., Design and implementation of components in the Earth System Modeling Framework, *International Journal of High Performance Computing Applications*, Fall/Winter 2005. DOI= 10.1177/1094342005056120. ⇒21

[4] P. Cousot, R. Cousot, Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints, *Proce. ACM Principles on Programming Languages*, ACM Press, 1977, pp. 238–252. ⇒11

[5] S. Haesevoets, B. Kuijpers, P. Z. Revesz, Affine-invariant triangulation of spatio-temporal data with an application to image retrieval, *ISPRS International Journal of Geo-Information* **6**, 4 (2017) 100. 37 pp. ⇒22

[6] C. Hill et al., Architecture of the Earth System Modeling Framework, *Computing in Science and Engineering* **6** 2004, Fall/Winter 2005. DOI= 10.1109/MCISE.2004.1255817. ⇒21

[7] B. J. Hoskins, A. J. Simmons, A multi-layer spectral model and the semi-implicit method, *Quarterly Journal of the Royal Meteorological Society* **101**, 429 (1975) 637–655. ⇒21

[8] J. T. Houghton et al. (editors), *Climate Change 2001: The Scientific Basis*, Cambridge University Press, 2001. ⇒6, 7, 12, 13, 14, 21

[9] P. C Kanellakis, G. M. Kuper, P. Z. Revesz, Constraint query languages, *Journal of Computer and System Sciences* **51,** 1 (1995) 26–52. ⇒10

[10] L. Li, P. Z. Revesz, Interpolation methods for spatio-temporal geographic data, *Computers, Environment and Urban Systems*, **28,** 3 (2004) 201–227. ⇒22

[11] B. Matthews, *Java Climate Model*, 2011. [Online]. Available: `http://jcm.climatemodel.info/` ⇒13

[12] S. C. B. Raper, J. M. Gregory, T. J. Osborn, Use of an upwelling-diffusion energy balance climate model to simulate and diagnose A/OGCM results, *Climate Dynamics*, **17** (2001) 601–613. ⇒12

[13] P. Z. Revesz, A closed form evaluation for Datalog queries with integer (gap)-order constraints, *Theoretical Computer Science*, **116,** 1 (1993) 117–149. ⇒14

[14] P. Z. Revesz, Safe query languages for constraint databases, *ACM Transactions on Database Systems*, **23,** 1 (1998) 117–149. ⇒14

[15] P. Z. Revesz, *Introduction to Constraint Databases*, Springer, 2002. ⇒10

[16] P. Z. Revesz, The constraint database approach to software verification, *Proc. 8th International Conference on Verification, Model Checking, and Abstract Interpretation*, LNCS 4349, Springer, 2007, pp. 329–345. ⇒7, 11, 21

[17] P. Z. Revesz, *Introduction to Databases: From Biological to Spatio-Temporal*, Springer, 2010. ⇒7, 8, 9, 10, 19

[18] P. Z. Revesz, A recurrence equation-based solution for the cubic spline interpolation problem, *International Journal of Mathematical Models and Methods in Applied Sciences* **9** (2015) 446–452. ⇒22

[19] P. Z. Revesz, R. Chen, P. Kanjamala, Y. Li, Y. Liu, Y. Wang, The MLPQ/GIS constraint database system, *ACM SIGMOD Record*, **29,** 2 (2000) p. 601. ⇒11

[20] P. Z. Revesz, R. J. Woodward, Variable bounds analysis of a climate model using software verification techniques, *Proc. 13th International Conference on Software Engineering, Parallel and Distributed Systems*, Gdansk, Poland, 2014, pp. 31–36. ⇒21

[21] P. Z. Revesz, S. Wu, Spatiotemporal reasoning about epidemiological data, *Artificial Intelligence in Medicine*, **38,** 2 (2006) 157–170. ⇒22

Received: December 5, 2018 • Revised: April 10, 2019

Acta Univ. Sapientiae, Informatica **11,** 1 (2019) 24–40

DOI: 10.2478/ausi-2019-0002

A study on the pendant number of graph products

Jomon K. SEBASTIAN
Manonmaniam Sundaranar University
Tirunelveli, Tamil Nadu 627012, INDIA
email: jomoncmi@gmail.com

Joseph Varghese KUREETHARA
CHRIST (Deemed to be University)
Bangalore, India
email: frjoseph@christuniversity.in

Sudev NADUVATH
CHRIST (Deemed to be University)
Bangalore, India
email: sudev.nk@christuniversity.in

Charles DOMINIC
CHRIST (Deemed to be University)
Bangalore, India
email:
charles.dominic@christuniversity.in

Abstract. A path decomposition of a graph is a collection of its edge disjoint paths whose union is G. The pendant number Π_p is the minimum number of end vertices of paths in a path decomposition of G. In this paper, we determine the pendant number of corona products and rooted products of paths and cycles and obtain some bounds for the pendant number for some specific derived graphs. Further, for any natural number n, the existence of a connected graph with pendant number n has also been established.

1 Introduction

We refer to West [1] and Harary[2] for terms and definitions in graph theory. All graphs we consider in this paper are undirected, simple, finite and connected.

Computing Classification System 1998: G.2.2
Mathematics Subject Classification 2010: 05C70, 05C76, 05C38
Key words and phrases: graph operations, path decomposition, pendant number

Partition of a graph G into its subgraphs is also termed as *decomposition* of G. A *path-decomposition* of a graph G is the partitioning of its edges into subgraphs S_i, $1 \leq i \leq n$, where each of the subgraph S_i is a path in G.

Definition 1 [3] The *pendant number* of a graph G, denoted by $\Pi_p(G)$, is the least number of vertices in a graph such that they are the end vertices of a path in a given path decomposition of a graph G. If $V_p(G)$ denotes the set of all $u \in V(G)$ such that u is an end vertex of a path in P-decomposition in G, then $\Pi_p(G) = \min\{|V_p(G)|\}$.

An introductory study on pendant number of graphs is available in [3]. A similar study on the star number of graphs can be seen in [4]. For the discussions in this paper, we use the following theorems.

Theorem 2 [3] *Let G be a connected graph with n vertices. If G has l odd degree vertices, then $l \leq \Pi_p(G) \leq n$.*

Theorem 3 [3] *Let T be tree on n vertices of which k vertices are of even degree. Then, $\Pi_p(T) = n - k$.*

Theorem 4 [3] *For a unicyclic graph G of order n; $n \geq 3$ with l odd degree vertices, we have*

$$\Pi_p(G) = \begin{cases} 2 & \text{if } m = 0; \\ l+1 & \text{if } m = 1; \\ l & \text{otherwise,} \end{cases}$$

where m is the number of vertices on C with $\deg(v) \geq 2$.

Proposition 5 [3] *If G is the cycle C_n on $n \geq 3$ vertices, then $\Pi_p(G) = 2$.*

2 Properties of pendant number

Even though there is no direct relation between the pendant number and other known and popular graph parameters, it is observed that the pendant number is highly influenced by the number of odd degree vertices in a graph. Moreover, the pendant number of a graph G has plenty of interesting properties, some of which, we deal with in the following discussion.

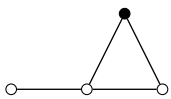
Definition 6 A *Π_p-realisation* of a positive integer $k \geq 2$ is a minimal connected graph G, whose pendant number is k.

By the *one-point union* of a collection of graphs (possibly with different order), we mean a graph obtained by replacing some or all edges of a path P by some graphs in the collection. In view of this notion, the following theorem establishes the existence of Π_p-realisation for any given positive integer.

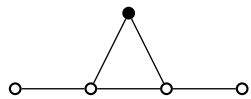
Theorem 7 *For every positive integer $k \geq 2$, there exists a Π_p-realisation for k.*

Proof. We can iteratively construct a (minimal) connected graph with pendant number k using 1-point union of K_2 and K_3 (in alternative manner) as shown in Figure 1. We note that for an even integer k, taking the one-point union of $\frac{k}{2}$ number of K_2's and $\frac{k}{2} - 1$ number of K_3's alternatively, we can construct a graph with $\Pi_p(G) = k$, whereas for an odd integer k, taking the one-point union of $\frac{k-1}{2}$ number of K_2's and K_3's alternatively, we can construct a graph with $\Pi_p(G) = k$. This completes the proof.

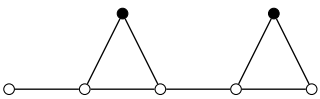
(a) A graph with $\Pi_p = 2$.



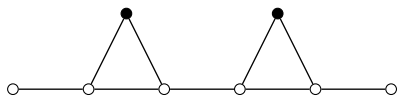
(b) A graph with $\Pi_p = 3$.



(c) A graph with $\Pi_p = 4$.



(d) A graph with $\Pi_p = 5$.



(e) A graph with $\Pi_p = 6$.

Figure 1: A Π_p realisation of the given positive integer $k \geq 2$.

□

Remark 8 It has been determined that for every $k \in \mathbb{N}$, there exists a graph with $\Pi_p(G) = k$. More precisely, for every $k \in \mathbb{N}$, there exists a graph with cycles such that $\Pi_p(G) = k$ and there exists an acyclic graph corresponding to any even natural number.

Theorem 9 *If u, v are two non-adjacent vertices in a graph G of order n, then $|\Pi_p(G + uv) - \Pi_p(G)| \leq 2$.*

Proof. Let G be a graph with $\Pi_p(G) > 2$. Let u and v be two non-adjacent vertices in G. Here, one may consider the following two cases:

Case-1: If u and v are of even degree and neither of them is an end point of a path in the decomposition of G, then in $G + uv$, u and v are of odd degree and become end points of some path in G, while the degrees of all other vertices remain the same as those in G. Hence, in this case, $\Pi_p(G + uv) = \Pi_p(G) + 2$ (see Figure 2 for illustration).

(a) A graph G with no even degree vertex as end point of a path.

(b) A graph $G+uv$ with $\Pi_p(G+uv) = \Pi_p(G) + 2$.

Figure 2

Case-2: Let P and P' be two (vertex-) disjoint paths in G such that u is an end vertex of P in G and v is an end vertex of P' in G. Join the edge uv. Now the path $P+uv+P'$ becomes a longer path, in which neither u nor v is an end point. In this case, $\Pi_p(G+uv) = \Pi_p(G)-2$ (see Figure 3 for illustration).

(a) A graph G with end vertices of paths.

(b) A graph $G+uv$ with $\Pi_p(G+uv) = \Pi_p(G) - 2$.

Figure 3

It can be verified that in all other possible cases, $\Pi_p(G + uv)$ lies between $\Pi_p(G) - 2$ and $\Pi_p(G) + 2$. This completes the proof. □

Invoking the results mentioned, we discuss some immediate observations in this section. By *one-point union* of cycles, we mean a graph obtained from a path by replacing its edges by cycles (possibly of different order). The following result provides the pendant number of one-point union of cycles.

Proposition 10 *If* G *is the one-point union of cycles, then* $\Pi_p(G) = 2$.

Proof. The proof is clear from Figure 4. The two edge disjoint (u, v)-paths of the one-point union of cycles are illustrated in the figure. □

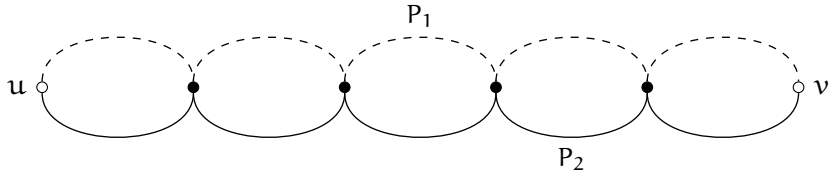
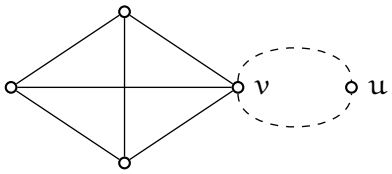


Figure 4: One-point union of cycles.

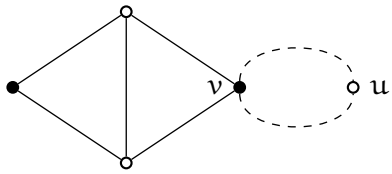
Proposition 11 *Let* G *be a graph, which is neither a cycle nor a one-point union of cycles. Then, the one-point union of* G *and a cycle* C_n *has the pendant number* $\Pi_p(G) + 1$.

Proof. Let G^* be the one-point union of a given graph G and a cycle C_n and let v be the vertex common to G and C_n in G^*. If v is a pendant vertex of any path decomposition of G, it will be a pendant vertex of some paths in G^* also. It can be taken as an end vertex of a path in C_n too. The other end vertex of this path can be arbitrarily chosen on C_n. If v is not a pendant vertex of any path decomposition of G, some paths passing through v in G can be extended to a vertex u of C_n in G^*. Hence, in this case also, one vertex of C_n will be a pendant vertex in G^*, other than the pendant vertices in G.

Therefore, in both cases, the pendant number is $\Pi_p(G) + 1$ (see Figure 5). □



(a) The one-point union of G and C_n with v as an end vertex of paths.



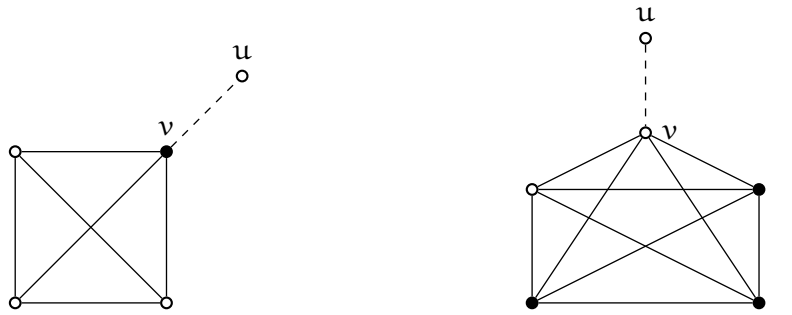
(b) The one-point union of G and C_n with v not as an end vertex of paths.

Figure 5

Proposition 12 *If* G *is a graph with cycles such that* v *is an end vertex of a path and* G^* *is the one-point union of* G *and a path* P_m *joined at* v*, then* $\Pi_p(G) \leq \Pi_p(G^*) \leq \Pi_p(G) + 1$.

Proof. If v is an end vertex of a single path, then the path can be extended to P_m so that $u \in P_m$ is the end vertex of the resultant path (see Figure 6a). If v is the end vertex of more than one path, then v together with $u \in P_m$ become the end vertices of the new path (see Figure 6b). □

Example 13 Let $G = K_n; n$ even and G^* be the (n, m)-*shovel graph*, which is the one-point union of K_n and P_m (that is, $(K_n \dot{\cup} P_m)$) [7]. Then, $\Pi_p(G^*) = \Pi_p(G)$. Let $G = K_n$; n odd and G^* be the (n, m)-shovel graph. Then, $\Pi_p(G^*) = \Pi_p(G) + 1$.



(a) A graph G with $\Pi_p(G^*) = \Pi_p(G)$ (b) A graph G with $\Pi_p(G^*) = \Pi_p(G)+1$.

Figure 6

A maximal, bi-connected, edge-disjoint subgraph of a graph G is called a *block* of G. Note that two blocks may have utmost one vertex in common.

Proposition 14 *If* $B_1, B_2, \ldots, B_k$ *are* k *distinct blocks of a graph* G*, then* $\Pi_p(G) \leq \Sigma_{i=1}^{k} \Pi_p(B_i)$.

The Figure 4 explains the case when the equality in the above result holds. The next proposition establishes an illustration of the case where there is the strict inequality in the above proposition.

Definition 15 A *non-uniform friendship graph* F_n^* is defined as the graph obtained by joining n cycles (need not be of same order) to a common vertex. Cycles in this graph are called as *petals* of F_n^*.

The following result discusses the pendant number of non-uniform friendship graphs.

Proposition 16 *For a non-uniform friendship graph* F_n^* *on* n *petals,* $\Pi_p(F_n^*) = n$.

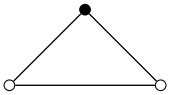
Proof. Let there be n petals in F_n^*. Let v_0 be the common vertex and choose n vertices $v_1, v_2, \ldots, v_n$ randomly from each petal. Join the path from v_1 to v_2 through v_0. Then, v_1 and v_2 becomes the end vertices of a path. Similarly, construct the paths from v_2 to v_3, from v_3 to v_4, ..., from v_{n-1} to v_n, and from v_n to v_1. There are n such paths with end vertices of every path is a starting point of another path. Hence, $\Pi_p(G) = n$. □

This class of non-uniform friendship graphs is another example for Π_p-realisation of positive integers. This fact is immediate from the above result. We consider $\delta(G)$ as the lowest degree among all the degrees of the vertices of the graph G.

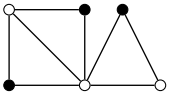
Theorem 17 *There exists a connected graph* G *with* $\delta(G) \geq 2$ *corresponding to any natural number* $k \geq 2$.

Proof. A (minimal) connected graph G with $\delta(G) \geq 2$ can be iteratively constructed corresponding to any natural number $k \geq 2$. Consider a cycle C_k with chords and a cycle C_3 in alternative manner. For an even integer $2k$, take the cycle C_{2k} with every vertex has exactly one chord results in an even number as pendant number. For an odd integer $2k+1$, take the one point union of the cycle C_{2k} with every vertex have exactly one chord and a C_3 attached to any one of its vertices, result in an odd number as pendant number (by Theorem 11). That is, consider the cycle C_3 to get the pendant number 2 (see Figure 7a). The one-point union of a diamond with C_3 has the pendant number 3 (see Figure 7b). The cycle C_4 with every vertex having exactly one chord has the pendant number 4 (see Figure 7c). The one point union of the above C_4 and a C_3 has the pendant number 5 (see Figure 7d). The cycle C_6 with every vertex having exactly one chord has the pendant number 6 (see Figure 7e). The one point union of the above C_6 and a C_3 has the pendant number 7 (see Figure 7f). This process gives the pendant number of a (minimal) connected graph with $\delta(G) \geq 2$ for any natural number. This completes the proof. □

In view of the Theorem 17, instead of taking C_3, if one takes a cycle of desired length, then it leads to the existence of a graph with desired pendant number and desired number of vertices. Note that to get the required pendant

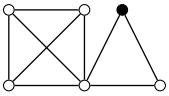


(a) A graph with $\kappa \geq 2$ and $\Pi_p = 2$.

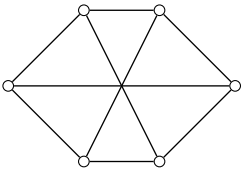


(b) A graph with $\kappa \geq 2$ and $\Pi_p = 3$.

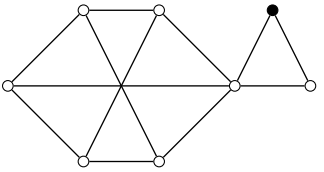
(c) A graph with $\kappa \geq 2$ and $\Pi_p = 4$.



(d) A graph with $\kappa \geq 2$ and $\Pi_p = 5$.



(e) A graph with $\kappa \geq 2$ and $\Pi_p = 6$.



(f) A graph with $\kappa \geq 2$ and $\Pi_p = 7$.

Figure 7

number, one must have at least the same number of odd vertices. It leads to the following result.

Corollary 18 *There exists a connected graph* G *of order* n *with* $\delta(G) \geq 2$ *corresponding to any natural number* $k \geq 2$ *and corresponding to any order* $n \geq 3$ *with desired pendant number.*

Theorem 19 *A graph* G *of order* $n \geq 3$*, with at least one even degree vertex has the pendant number at most* $n - 1$*.*

Proof. Since the result has already been proved for acyclic graphs (see Theorem 3), it is sufficient to prove the result for cyclic graphs on n vertices using the method of induction on the number of vertices.

The smallest graph with cycles with at least one even degree vertex is C_3 and $\Pi_p(C_3) = 2$. Hence the result is true for $n = 3$. Assume the result is true for $n = k$. Let G be a graph with k vertices and $\Pi_p(G) \leq k - 1$. Let v_1 be an even degree vertex, which is not an end vertex of any path in G.

Now let $n = k+1$. Add one more vertex v_{k+1} to the above graph G. Connect any number of vertices of G except v_1 to v_{k+1} (if v_1 is joined to v_{k+1}, then degree of v_1 becomes odd). For v_1 is not an end vertex of any path in G, $\Pi_p(G) \leq k$. Hence, the proof. □

Theorem 20 *Let* G *be a graph of order* n*, consisting of cycles and* $k > 0$ *even degree vertices. Then,* $n - k \leq \Pi_p(G)$*.*

Proof. Let us use the induction on k, viz., $k = 1, n-1$ and n. Let $k = 1$. Then, by Theorem 19, $n-1 \leq \Pi_p(G) \leq n-1$. Let $k = n-1$ then, $n-(n-1) = 1 \leq \Pi_p(G)$. Let $k = n$ then, $n - n = 0 \leq \Pi_p(G)$. Hence the result is true. □

Combining Theorem 19 and Theorem 20, it follows:

Theorem 21 *If a graph G, with $n \geq 3$ vertices, has k even degree vertices, then $n - k \leq \Pi_p(G) \leq n - 1$.*

Proposition 22 *If both G and its complement $\overline{G}$ are connected graphs with odd degree, then $|V(G)| \geq 5$.*

Proof. Let a graph G and its complement $\overline{G}$ be connected graphs such that all their vertices are of odd degree. Then, at least two vertices each of G and $\overline{G}$ will be with degree ≥ 3. If all vertices except one is of degree one in G, then this vertex must be isolated in $\overline{G}$. It implies that the number of vertices of G must be at least 5. □

A pineapple graph [9], denoted by K_n^m, is a graph obtained by appending m pendant edges to a vertex of a complete graph K_n; $m \geq 1, n \geq 3$. Let $\mathcal{A}$ be the collection of graphs given in Figure 8 and Figure 9. Even though $K_3^m \subseteq$ one-point union of a triangle and an odd degree tree, $\overline{K}_3^m$ is disconnected. Moreover, it is clear that the complement of one-point union of triangle and an odd degree tree always has $\Pi_p \leq n - 2$ (see Theorem 3 and Theorem 4).

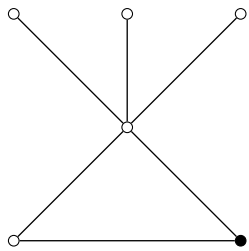


Figure 8: The graph K_3^m; m odd.

The next proposition can be proved in a similar manner as that of Theorem 19.

Proposition 23 *Let G be a graph G of order $n \geq 4$ and $G \notin \mathcal{A}$. If G has at least two even degree vertices, then $\Pi_p(G) \leq n - 2$.*

Using Proposition 22 and Proposition 23, we have the next result associated a graph G and its complement $\overline{G}$.

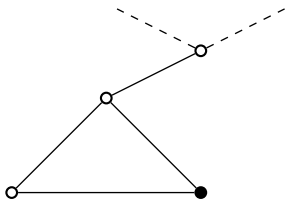


Figure 9: One-point union of K_3 and an odd degree tree.

Theorem 24 *If both the graphs* G *and its complement* $\overline{G}$ *are connected, then* $4 \leq \Pi_p(G) + \Pi_p(\overline{G}) \leq 2(n-1)$.

Proof. The lower bound of $\Pi_p(G)$ for any graph G is 2. Therefore, $2 \leq \Pi_p(G)$ and $2 \leq \Pi_p(\overline{G})$. Hence, $4 \leq \Pi_p(G) + \Pi_p(\overline{G})$.
To prove the other part, one may consider the following two cases:

Case-1: The number of vertices n of the graph G is odd. Since n is odd, at least one vertex each of G and $\overline{G}$ must be even. Thus, the result can be determined by using the Theorem 19, as $\Pi_p(G) \leq n-1$ and $\Pi_p(\overline{G}) \leq n-1$.

Case-2: The number of vertices n of the graph G is even. Let all the vertices of G be odd degree vertices. Hence, $\Pi_p(G) = n$. Thus, the degree of $\overline{G}$ must be even. By Proposition 22, $|V(G)| \geq 5$ and by Proposition 23, $\Pi_p(\overline{G}) = n-2$.

In both cases, it is determined that $\Pi_p(G) + \Pi_p(\overline{G}) \leq 2(n-1)$, completing the proof. □

3 Pendant number of graph products

The *rooted product* of two graphs G_1 and G_2, denoted by $G_1 \circ G_2$, is the graph obtained by taking $|V(G_1)|$ copies of G_2 and identifying one vertex (root) of each copy of G_2 to the corresponding vertex of G_1. The following result discusses the pendant number of the rooted products of cycles and paths.

Theorem 25 *Let* P_n *and* P_m *be any two paths and* C_n *and* C_m *be any two cycles. Then the pendant number of their rooted products are given by;*

(i) *For* $P_n \circ P_m$,

$$\Pi_p(P_n \circ P_m) = \begin{cases} 2(n-1) & \text{if the root vertex is a pendant vertex ;} \\ 2(n+1) & \text{if the root vertex is an internal vertex.} \end{cases}$$

(ii) *For* $C_n \circ P_m$, $\Pi_p(C_n \circ P_m) = 2n$.

(iii) *For* $C_n \circ C_m$, $\Pi_p(C_n \circ C_m) = n$.

Proof.

(i) Let $u_1, u_2, \ldots, u_n$ be the vertices of P_n and $v_1, v_2, \ldots, v_m$ be the vertices of P_m.

Case-1: Let the root vertex of P_m be a pendant vertex. Then, in the rooted product $P_n \circ P_m$, one copy of P_m is joined to each of the vertices $u_1, u_2, \ldots, u_n$. The longest path includes the one-point union of P_n between two copies of P_m (situated at the end points of P_n), yielding 2 pendant vertices. The remaining $n-2$ vertices of P_n together with the pendant vertices in the remaining $n-2$ copies of P_m result in $2(n-2)$ pendant vertices (see Figure 10 for illustration). Therefore, $\Pi_p(G) = 2 + 2(n-2) = 2(n-1)$.

Case-2: Let the root vertex of P_m be an internal vertex. Then, both end vertices of all the n copies of P_m and the end vertices of P_n are the end vertices of some paths in the path decomposition of $P_n \circ P_m$ (see Figure 11 for illustration). Hence, $\Pi_p(G) = 2(n+1)$.

(ii) For $C_n \circ P_m$, the collection $V_p(G)$ (see 1) is constituted by all the vertices of C_n together with one end vertex of each of the n copies of the paths P_m on the other end as seen in Figure 12. Hence, $\Pi_p(G) = 2n$.

(iii) In the rooted product $C_n \circ C_m$, one copy of C_m is joined at each vertex of C_n. Now take any vertex of degree 2 of $C_n \circ C_m$ as the first vertex of a path. This path passes through some vertices of the same copy of C_m, say $C_{m(i)}$, passes through two adjacent vertices of C_n and passes through some vertices of the next copy $C_{m(i+1)}$ before it terminates at some vertex of $C_{m(i+1)}$. The next path starts from this end vertex in $C_{m(i+1)}$ and will end at some vertex of the next copy $C_{m(i+2)}$. Continuing like this, one can find out paths which cover all edges of $C_n \circ C_m$, as shown in Figure 13. Hence, $\Pi_p(C_n \circ C_m) = n$.

□

The *corona product* $G \odot H$ of two graphs G and H is obtained by taking one copy of G and $|V(G)|$ copies of H; and by joining each vertex of the i^{th} copy of H to the i^{th} vertex of G; $1 \leq i \leq |V(G)|$ (see [8]). When we consider the corona of two paths, for certain initial values of n and m, the pendant numbers of $P_n \odot P_m$ have already determined (see [3],[6]). They are:

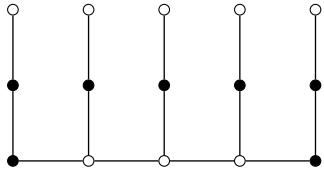


Figure 10: A rooted product $P_n \circ P_m$ with the root vertex as a pendant vertex.

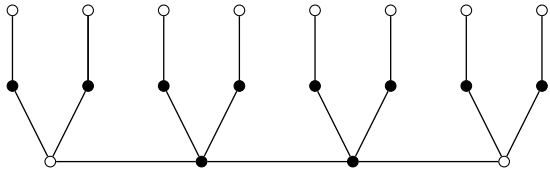


Figure 11: A rooted product $P_n \circ P_m$ with the root vertex as an internal vertex.

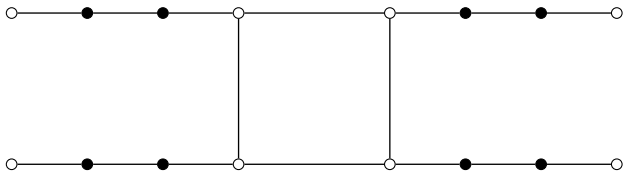


Figure 12: A rooted product $C_n \circ P_m$.

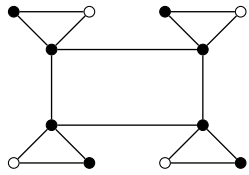


Figure 13: A rooted product $C_n \circ C_m$.

(i) We can exclude the cases $P_1 \odot P_1 = K_2$, $P_1 \odot P_2 = C_3$, $P_2 \odot P_1 = P_4$, whose pendant number is already determined in [3] as 2.

(ii) Since, $P_1 \odot P_m$; $m \geq 3$ is the $n-$ fan graph on $n+2$ vertices, its pendant number is determined in [6] as;

$$\Pi_p(P_1 \odot P_m) = \begin{cases} m-1 & \text{if } m \text{ is odd;} \\ m-2 & \text{if } m \text{ is even.} \end{cases}$$

(iii) $P_n \odot P_1$; $n \geq 3$ is the comb tree T, whose pendant number is $\Pi_p(P_n \odot P_1) = 2(n-1)$ (see[6]).

(iv) $P_n \odot P_2$; $n \geq 2$ is the lever graph L_n with pendant number $\Pi_p(P_n \odot P_2) = n + 2$ (see [6]).

(v) Since $P_n \odot P_3$; $n \geq 2$ is the diamond neckalce graph D_n, the pendant number is $\Pi_p(P_n \odot P_3) = 2n$ (see[6]).

Theorem 26 *The pendant number of the corona product* $P_n \odot P_m$ *with* $n \geq 2$ *and* $m \geq 4$ *of two paths* P_n *and* P_m *is given by*

$$\Pi_p(P_n \odot P_m) = \begin{cases} n(m-1)-2 & \text{if } m \text{ is odd;} \\ n(m-2)+2 & \text{if } m \text{ is even.} \end{cases}$$

Proof. Let $u_1, u_2, \ldots, u_n$ be the vertices of P_n and $v_1, v_2, \ldots, v_m$ be the vertices of P_m. Let $u_1, u_2, \ldots, u_n$ be the vertices on the root and $v_{11}, v_{12}, \ldots, v_{1m}$, $v_{21}, v_{22}, \ldots, v_{2m}$, ..., $v_{n1}, v_{n2}, \ldots, v_{nm}$ be the vertices on the crown. Then, the total number of vertices is $n(m+1)$. Let the j^{th} vertex of the i^{th} copy of P_m be denoted by $v_{i,j}$. The corona product of $P_n \odot P_m$ where $n \geq 2$ and $m \geq 4$ has two cases:

Case-1: Let m be odd. Since $v_{i1}, v_{im}; 1 \leq i \leq n$ and u_1, u_n are even degree ($= 2n+2$) and all other vertices are odd, one can make a path decomposition with the odd degree vertices as the end vertices of every path (see Figure 14) and it will be the least (see Theorem 2). Hence, $\Pi_p(G) = n(m+1) - (2n+2) = n(m-1) - 2$.

Case-2: Let m be even. Since $v_{i1}, v_{im}; 1 \leq i \leq n$ and $u_2, u_3, \ldots, u_{n-1}$ are even degree ($= 2n + (n-2)$) and all other vertices are odd, the path decomposition with minimum number of end vertices be the path decomposition with odd degree vertices as the end vertices of every path (see Figure 15). Hence,$\Pi_p(G) = n(m+1) - (2n+n-2) = n(m-2)+2$.

□

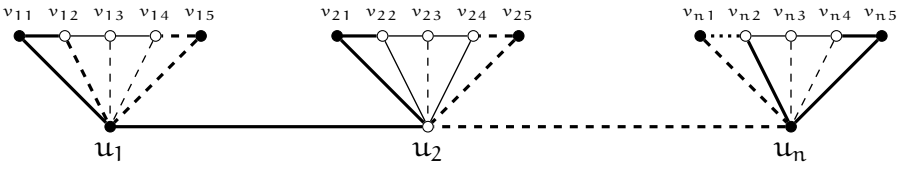


Figure 14: A corona product $P_n \odot P_m; m$ odd.

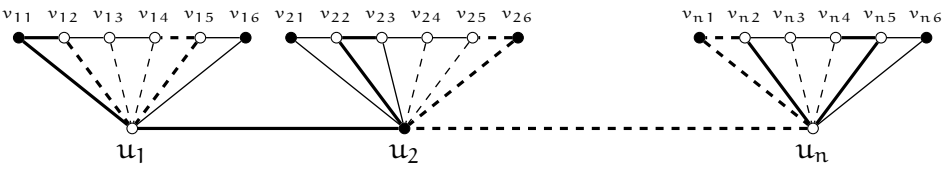


Figure 15: A corona product $P_n \odot P_m; m$ even.

Theorem 27 *The pendant number of the corona product of cycles and paths is given as:*

(i) For $C_n \odot P_m$,

$$\Pi_p(C_n \odot P_m) = \begin{cases} 2n & \text{if } m = 1; \\ n & \text{if } m = 2; \\ 2n\lfloor \frac{m-1}{2} \rfloor & \text{if } m \neq 1, 2. \end{cases}$$

(ii) The pendant number of the corona product $C_n \odot C_m$ *is given by,*

$$\Pi_p(C_n \odot C_m) = \begin{cases} nm & \text{if } m \text{ is even;} \\ n(m+1) & \text{if } m \text{ is odd.} \end{cases}$$

Proof.

(i) The first two parts of the result follow respectively from Part-(ii) and Part-(iii) of Theorem 25.

Let us now consider the corona product $C_n \odot P_m; m \neq 1, 2$. Note that for every vertex v of C_n, the degree of v in $C_n \odot P_m$ is $m + 2$ and for every vertex of each copy of P_m has degree one more than the degree of the corresponding vertex in P_m. Hence, the following two cases must be considered:

Case-1: Let m be odd. Then, in $C_n \odot P_m$, every vertex of C_n becomes odd degree vertex and all vertices of each copy of P_m, except two (corresponding to the end vertices of P_m) become odd. It is possible to find edge-disjoint paths in $C_n \odot P_m$ in such a way that the vertices of degree two are not pendant vertices of paths in the path decomposition of $C_n \odot P_m$ (see Figure 16 for example). Hence, $\Pi_p(C_n \odot P_m) = n + n(m-2) = n(m-1)$.

Case-2: Let m be even. Then, in $C_n \odot P_m$, every vertex of C_n remains as an even degree vertex and all vertices of each copy of P_m, except two (corresponding to the end vertices of P_m) become odd. It is possible to find edge-disjoint paths in $C_n \odot P_m$ in such a way that the even degree vertices are not pendant vertices of paths in the path decomposition of $C_n \odot P_m$ (see Figure 17 for example). Hence, $\Pi_p(C_n \odot P_m) = n(m-2)$.

Combining the above two cases, it is clear that $\Pi_p(C_n \odot P_m) = 2n\lfloor \frac{m-1}{2} \rfloor$.

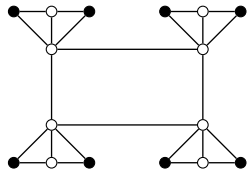


Figure 16: A corona product $C_n \odot P_m; m$ odd.

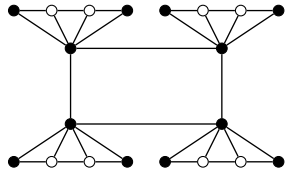


Figure 17: A corona product $C_n \odot P_m; m$ even.

(ii) Let $u_1, u_2, \ldots, u_n$ be the vertices of C_n and $v_1, v_2, \ldots, v_m$ be the vertices of C_m. Let $u_1, u_2, \ldots, u_n$ be the vertices on the root and $v_{11}, v_{12}, \ldots, v_{1m}$, $v_{21}, v_{22}, \ldots, v_{2m}$, $\ldots$, $v_{n1}, v_{n2}, \ldots, v_{nm}$ be the vertices on the crown. In $C_n \odot C_m$, $v_{i1}, v_{i2}, v_{im}; 1 \leq i \leq n$ always have odd degree with $\deg(v_{ij}) = 3; 1 \leq j \leq m$. The total number of vertices in the corona product is $n(m+1)$. There are two possibilities for $C_n \odot C_m$.

Case-1: Let m be even. Since every vertex of each copy of C_m is joined to each vertex of C_n, the degree of C_n remains even, one can make a path decomposition in such a way that none of the vertices of C_n is the end vertex of any path (see Figure 18) and it will be the least too (see Theorem 2). Thus the pendant number becomes $n(m+1) - n = nm$.

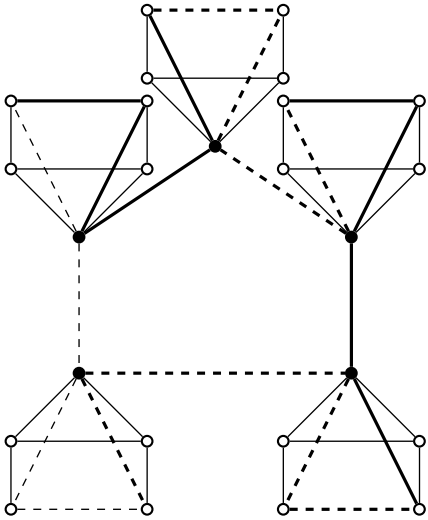


Figure 18: A corona product $C_n \odot C_m$; m *even*.

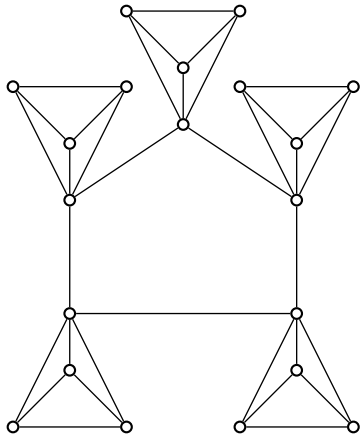


Figure 19: A corona product $C_n \odot C_m$; m odd.

Case-2: Let m be odd. Then, all the vertices of $C_n \odot C_m$ become odd. Thus, by Theorem 2, pendant number is $n(m+1)$ (see Figure 19), the entire vertex set of the resultant graph.

□

4 Conclusion

In this paper, we discussed certain properties of the pendant number of a given graph G. We have determined the pendant number of corona products and rooted products of paths and cycles. We have also obtained some bounds for the pendant number for some specific derived graphs. Corresponding to any natural number $\mathfrak{n}$, the existence of a connected graph with pendant number $\mathfrak{n}$ has also been established in this study. These studies can further be extended to other general graph classes, graph products and other graph operations. The relation of pendant numbers with other popular parameters like domination number, covering number, chromatic number etc. can also be studied.

References

[1] D. B. West, *Introduction to Graph Theory*, Prentice Hall of India, New Delhi, 2005. ⇒24

[2] F. Harary, *Graph Theory*, Narosa Publishing House, New Delhi, 2001. ⇒24

[3] J. K. Sebastian, J. V. Kureethara, Pendant number of graphs, *Int. J. Appl. Math.*, *IJAM* **31,** 5 (2018) 679–689. ⇒25, 34, 35

[4] J. K. Sebastian, J. V. Kureethara, N. K. Sudev, C. Dominic, On star decomposition and star number of some graph classes, *Int. J. Scien. Res. Mathe. Stati. Sci.*, *IJSRMSS*, **5,** 6 (2018) 81–85. ⇒25

[5] J. K. Sebastian, J. V. Kureethara, N. K. Sudev, C. Dominic, On the pendant number of some new graph classes, *Res. Rev. Discrete Math. Structures RRDMS*, **6,** 1 (2019), 15–21. ⇒34, 35, 36

[6] J. K. Sebastian, J. V. Kureethara, N. K. Sudev, The pendant number of line graphs and total graphs, *communicated.* ⇒34, 35, 36

[7] N. K. Sudev, K. A. Germina, A study on topological integer additive set-labeling of graphs, *Electron. J. of Graph Theory Appl.* Electronic Journal, **3,** 1 (2015), 70–84. ⇒29

[8] Y. N. Yeh, I. Gutman, On the sum of all distances in composite graphs, *Discrete Math.* Discrete, **135,** 1-3 (1994), 359–365. ⇒34

[9] X. Zhang, H. Zhang, Some graphs determined by their spectra, *Lin. Alg. and Its Appl.* Linear, **431,** 9 (2009), 1443–1454. ⇒32

Received: February 19, 2019 • Revised: May 29, 2019

Acta Univ. Sapientiae, Informatica **11,** 1 (2019) 41–51

DOI: 10.2478/ausi-2019-0003

Comparing two- and three-view computer vision

Zsolt Levente KUCSVÁN
Sapientia Hungarian University of Transylvania
Cluj-Napoca
Dept. of Mathematics and Informatics
Târgu Mureş, Romania
email: kzsolt@student.ms.sapientia.ro

Abstract. To reconstruct the points in three dimensional space, we need at least two images. In this paper we compared two different methods: the first uses only two images, the second one uses three. During the research we measured how camera resolution, camera angles and camera distances influence the number of reconstructed points and the dispersion of them. The paper presents that using the two-view method, we can reconstruct significantly more points than using the other one, but the dispersion of points is smaller if we use the three-view method. Taking into consideration the different camera settings, we can say that both the two- and three-view method behaves the same, and the best parameters are also the same for both methods.

1 Introduction

Computer Vision plays an increasingly important role in our days. It's use is very diverse. The diversity is reflected in it's use from engineering to medical applications but Computer Vision also plays a major role in the entertainment industry. Some examples of it's usage are: autonomous cars, face detection, three dimensional triangulation, extended reality, Google Street View etc.

Computing Classification System 1998: I.4.m
Mathematics Subject Classification 2010: 68R15
Key words and phrases: computer vision, triangulation, reconstruction

Most of the applications mentioned above require a variety of tools. Autonomous cars use different capture devices, such as LIDAR (laser-based sensor), see for instance [6], to achieve "vision". Likewise, there is a need for a device (eg. MRI or CT) that can capture an image from inside the body to segment the tumors. It is noticeable that similarly to the complex structure of the human visual system, for the computer vision, we also need a complex system of physical devices. Of course, this complex physical system is not enough. There is also a need for effective software, that processes and interprets the information gained through the devices.

From a large set of applications, the three-dimensional triangulation may seem to be the simplest. For this application, we do not need anything else but just different photos about the same object. It is not necessary for images to be made with the same camera and it is not important how the cameras are placed when capturing the images, but it is important to have at least one common object on the images. Based on these, it seems this problem can be solved more easily than the rest. Nevertheless, if we are getting deeper in this subject, it turns out that although this task is associated with the least constraint, it is one of the most difficult to solve mathematically and to write optimal software for this.

In this paper the main goals are to compare the two-view triangulation with the three-view one and to create a data set, that is suitable for the previously mentioned purpose. Comparing the two methods, we had the following aims: to compare the two methods based on input images with different resolution, in the case of different angles of the cameras and finally in the case of different distances of the cameras.

In order to make the right measurements, the following are required: to identify common pixels on images, to calculate the camera matrices based on the images and to triangulate the identified common pixels based on the camera matrices (using both two-view and three-view method).

2 Mathematical background and previous results

1997 is a significant year in the history of Computer Vision. At this point, there were methods for the triangulation problem, but in this year Richard I. Hartley with Peter Sturm published a method, which gives an optimal solution in case we are using image pairs. Their method differs from previous methods, so not the algebraic error was minimized but the geometrical. By this method they achieved to solve the problem optimally by finding the roots of a 6th grade polynomial, see [5].

In 2005 Henrik Stewenius, Frederik Schaffalitzky and David Nister published a paper about solving optimal the three-view triangulation. This method is based also on minimizing the geometric error, but instead of the 6th grade polynomial we have to find the roots of a 47th grade polynomial to get the optimal solution, see [10].

The three-view triangulation compared to the two-view has just one more input (the third picture) but the problem to find the optimal triangulation has become more complicated. This is also a question, that if we want to triangulate optimally from n pictures, how complicated will be the polynomial we have to solve. We don't know for sure the answer to this question, but there is a paper from 2016 in which there is formulated a conjecture about this. Conform to the conjecture, if we have n pictures, the grade of polynomial will be the following (see [2]):

$$\frac{9}{2}n^3 - \frac{21}{2}n^2 + 8n - 4$$

2.1 Camera coordinate system

The corresponding point of $X_{3D} = (x_1, x_2, x_3)^T$ is the $X_{2D} = (x_1, x_2)^T$ in the camera coordinate system. This has the following form in homogeneous coordinates (see [7]):

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ 1 \end{pmatrix} \longmapsto \begin{pmatrix} fx_1 \\ fx_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ 1 \end{pmatrix} \tag{1}$$

Transformation (1) is a simple product of matrices. The left-hand side of the product is the projection matrix, that contains the parameters of the camera. These parameters determine the formation of the image. The right-hand side of the product is the three-dimensional point itself, which we would like to take picture of. The actual projection matrix looks like the following (see [7]):

$$P = \begin{pmatrix} s_x f & a & s_x o_x & 0 \\ 0 & s_y f & s_y o_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \tag{2}$$

where f is the focal length, (s_x, s_y) the zoom of the camera, (o_x, o_y) the translation of the axes of the camera coordinate system and a determines the shape of the pixels. The parameter a is not null just in the case, where the axes of

the coordinate system are not perpendicular to each other, see [7].
The projection matrix has also the following form:

$$P = K[I|O], \tag{3}$$

where K is the calibration matrix, see [7].

2.2 World coordinate system

It is not enough to consider just the camera coordinate system, because the real points which we want to take pictures of are in the world coordinate system. The connection of the two coordinate systems are the following:

1. There is a translation t between them
2. The world coordinate system is rotated

Taking into consideration these, the projection matrix has the following form:

$$P = K[R|Rt] = KR[I|t] = K[R| - R\tilde{C}] = KR[I| - \tilde{C}], \tag{4}$$

where R is the rotation and, t is the translation. $\tilde{C}$ is inhomogeneous coordinate of camera center in the world coordinate system ($t = -\tilde{C}$), see [7].

2.3 Camera calibration

From (1), we can easily determine the image of a point if projection matrix P is known. The process of determining the projection matrix P is called camera calibration.

One method to this process is using calibration pattern. The corners of a chessboard can be easily identified, so they are often used. The most used calibration pattern is the Tsai grid, see [11].
P has 12 elements, but it's degree of freedom is just 11, so we need 11 equations to determine the projection matrix.

Because of the noise of the images, the 11 equations won't give an exact solution, so we use more equations and minimize the algebraic error of the over-defined equation system, see [7].

$$Ap = 0, \tag{5}$$

where A has size $2N \times 12$ and $p = (P_{11}, P_{12}, \ldots, P_{34})^T$ contains the unknown elements of the projection matrix ($P_{ij} \in P$, $i = \overline{1,3}$, $j = \overline{1,4}$).

2.4 Geometrical error

(5) minimizes the algebraic error of the system, but this won't be optimal geometrically. For minimizing the geometrical error of n-view method, we use the following equation, from [7]:

$$\mathrm{Err} = \sum_{i=1}^{n} \|X_{2D_i} - PX_{3D_i}\|^2, \tag{6}$$

where X_{2D} is the coordinate of a pixel on the image, and PX_{3D} is coordinate projected by the camera matrix. This is a least squares problem, which we can solve for example with *Levenberg-Marquard algorithm*, see [4].

3 Practical implementation

To make the proper measurements, we needed an appropriate set of data. Since we did not find a suitable data set, we generated one, see [8]. The pictures for this data set were made with a Google Pixel 2 smartphone, using the OpenCamera (see [3]) application freely available for Android phones. The specifications of the camera are the following:

Sensor type	CMOS
Sensor size	1/2.6"
Aperture	f/1.8
Focal length	≈4.47 mm
Image Resolution	4032x3024 (12.19 MP)
Pixel size	1.4 µm

Table 1: Google Pixel 2 – camera specifications [1]

3.1 OpenMVG library

For comparing the two methods, we used the OpenMVG open-source library, see [9]. This library is designed for computer-vision scientists. OpenMVG provides solutions for multiple problems in Computer Vision, such as the triangulation.

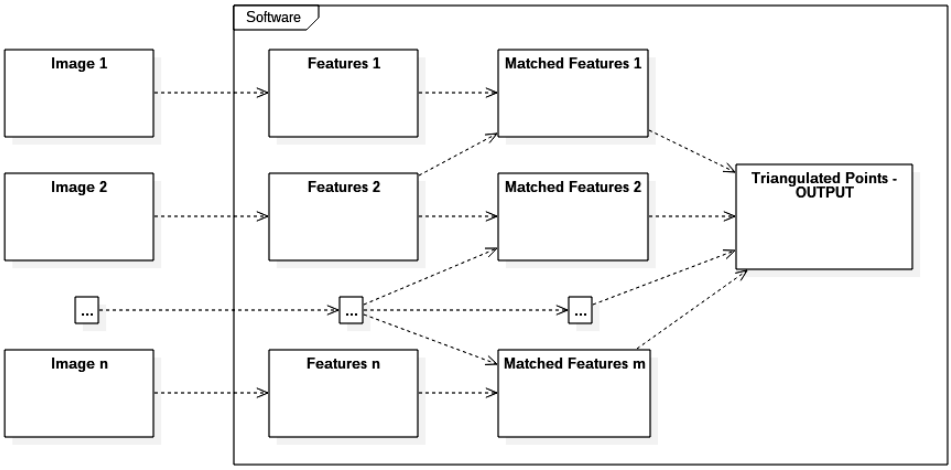


Figure 1: OpenMVG Reconstruction System

4 Evaluation and conclusion

Given the number of found common pixels, the two-view method founds $1.45-1.86$ times more common pixels than three-view one, but there are some questions. The first question would be, what happens with common pixels on all images? In this case the two-view method does three triangulations: using images $1-2$, $1-3$ and $2-3$. From these three triangulations two are redundant. Furthermore if there is noise on the images, the needlessly triangulated points won't coincide with each other, however they should.

4.1 Evaluation of different resolution measurements

We can notice, that the smaller the resolution, the less common points will find the software. At very low resolution (480×320), the software does not work at all. Up to FullHD resolution (1920×1080) the software finds less than $1,000$ while above FullHD resolution finds thousands of common pixels with the two-view method. The three-view method has similar results. Up to UltraHD resolution (3840×2160) the number of reconstructed pixels increases steadily, while at the highest resolution (4032×3024) a larger drop is detected (426 points for two-view and 122 for three-view method). One possible explanation for this is the higher noise entering at high resolution. If there was no noise on the images, the number of reconstructed points would increase steadily by increasing the resolution.

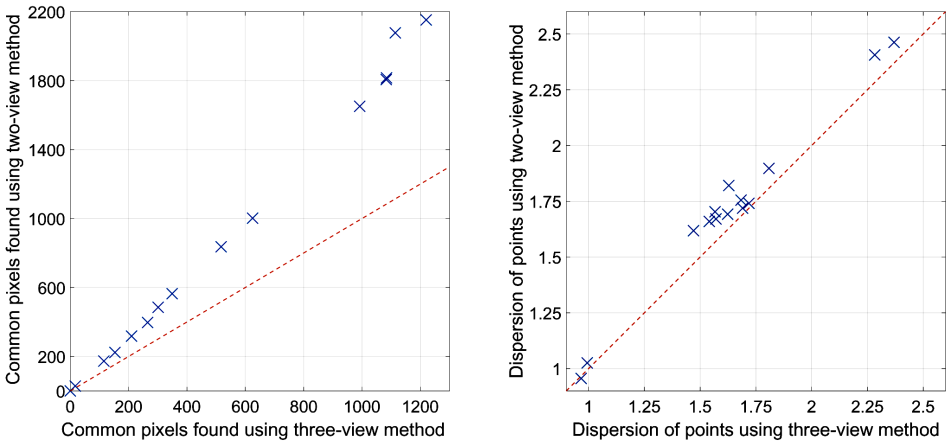


Figure 2: Different resolution measurements

Analyzing the dispersion, we can observe for the three-view method this is smaller. The dispersion at three-view method can be smaller from 1.2% up to 11.8%. On average, the dispersion of three-view triangulation is less by 5.6% than the two-view method.

4.2 Evaluation of different angles measurements

First of all, the middle camera was fixed for all measurements at 21cm distance from the object. For every three measurement, the left camera was fixed, while the right-one was positioned in $8.74°, 13.18°$ and $16.79°$ degrees to the middle camera. At the first three measurements, the left camera was held in $5.33°$, in the next three in $9.46°$ and at the last three in $10.72°$ degrees to the middle camera (see Figure 3).

The first conclusion we can observe is the resolution does not play a key role in either two- or three-view method. For both methods, the number of pixels matched increases in the same way as the angles increase, but for two-view method is still significantly higher. From the current measurements, we can conclude, the dispersion decreases if one camera is positioned in a low angle, and the other in a high angle to the middle camera.

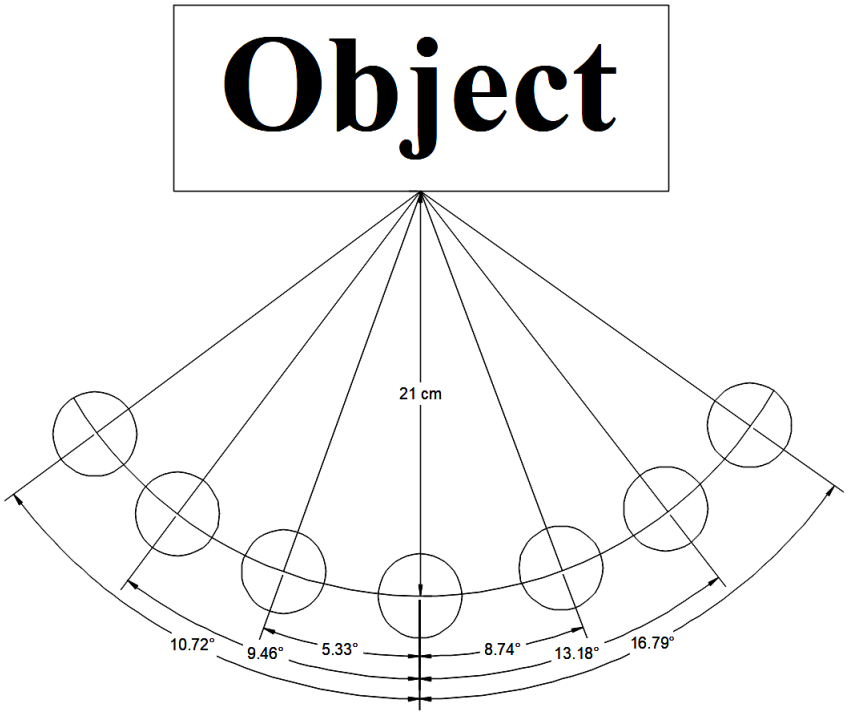


Figure 3: Camera positions for different angle measurements

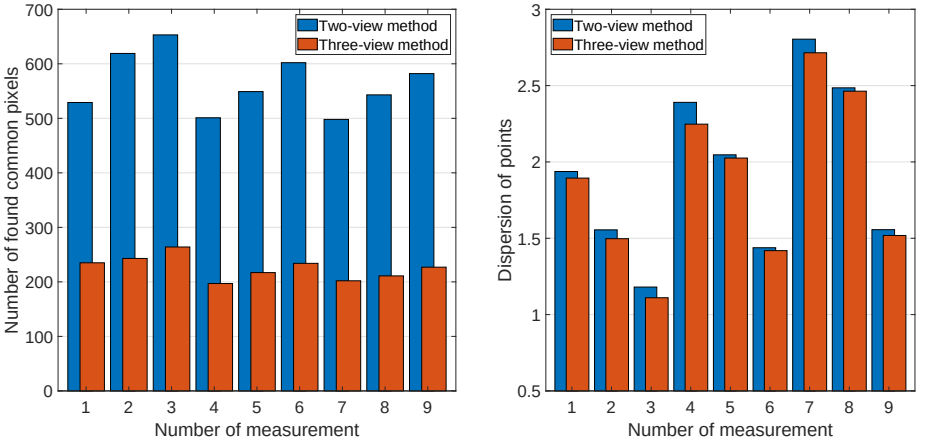


Figure 4: Different angle measurements

The best result is in the case when left-camera has 5.33° and the right-camera 16.79° degrees to the middle camera. In this case the number of found common pixels is 653 for the two-view method, and 264 for the three-view one. In this case also, the dispersion of points is minimal, 1.18 for two-view method, and 1.11 for three-view one.

The worst result is at 10.72° for the left camera and 8.74° for the right-one, with 498 common pixels found using two-view method, and 202 using three-view. The dispersions in this case are the largest, 2.8 and 2.71 respectively for the two-view and three-view methods.

4.3 Evaluation of different distances measurements

At these measurements, we fixed seven different points on a line. The points were at 1.73 cm, 3.46 cm, 5.33 cm, 7.2 cm, 9.16 cm, and 10.93 cm distances relatively to the first fixed point. We positioned the cameras in every combination of these points for measurements. The fourth point (with 5.33 cm distance to the left-most point) was on the perpendicular bisector of the object.

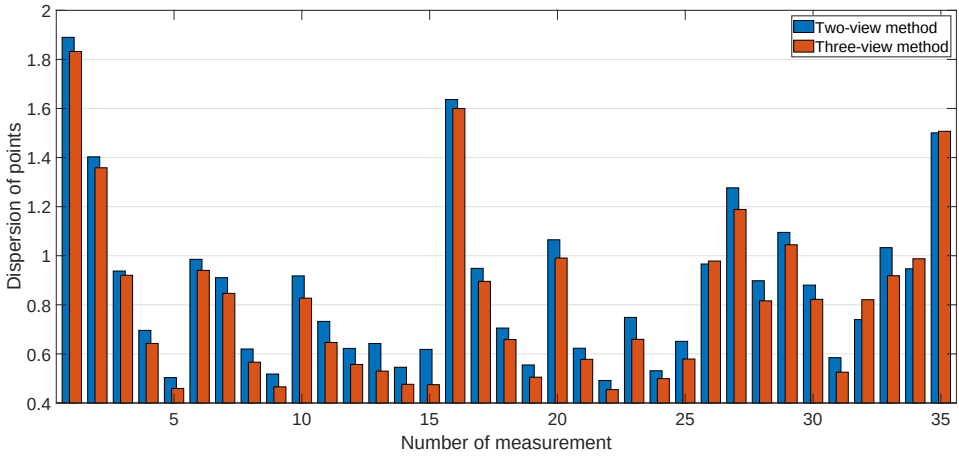


Figure 5: Different distances measurements

Our measurements don't show a clear correlation between the different distances between cameras and the number reconstructed pixels, but it is clearly apparent that the dispersion decreases when two cameras are close together and the third moves away.

The best result was when the left camera is at 1.73 cm, the middle-one at 5.33 cm and the right-one at 7.2 cm distance from the left-most point. In this case the dispersion is 0.49 for two-view method, and 0.45 for three-view one.

The worst result was the first measurement, when the left camera is at 0 cm, the middle-one at 1.73 cm and the right-one at 3.46 cm distance from the left-most point. In this case the dispersion is 1.89 for the two-view method, and 1.83 for the three-view one.

4.4 Conclusion

From the results, firstly, we can conclude, that using the two-view method there will be more common pixels but using the three-view method the dispersion of triangulated points will be smaller. Secondly, we conclude, that the number of found common pixels increases if we increase the angle of one camera but the other camera remains at a low angle to the middle one. The dispersion of triangulated points depends on the distance between the cameras, but also from the position relative to the object. The last conclusion is that until a certain resolution, the bigger resolution results in more triangulated points. This resolution may differ from the camera used for capturing the images. In our case, the best resolution was the UltraHD resolution (3840×2160 pixels). Summarizing the research output, we can formulate that using the three-view method, we get a better triangulation result for the human eye than using the two-view method. The question remains whether the three-view method is more accurate than the two-view method, or not?

Acknowledgements

This research was partially supported by the Forerunner Federation. The author is also deeply grateful for all the help received from his advisors: Csaba Farkas and Emil Horobeţ.

References

[1] DeviceSpecifications. Google pixel 2 device specifications. `https://www.devicespecifications.com/en/model/cd694619`. ⇒45

[2] J. Draisma, E. Horobeţ, G. Ottaviani, B. Sturmfels, R. R. Thomas, The euclidean distance degree of an algebraic variety, *Foundations of computational mathematics*, **16,** 1, (2016) 99–149. ⇒43

[3] M. Harman, Opencamera, `https://sourceforge.net/projects/opencamera/`. ⇒45

[4] R. Hartley, A. Zisserman. *Multiple view geometry in computer vision.* Cambridge University Press, 2003. ⇒45

[5] R. I. Hartley, P. Sturm, Triangulation, *Computer vision and image understanding*, **68,** 2 (1997) 146–157. ⇒42

[6] L. Hung, M. Barth, A novel multi-planar lidar and computer vision calibration procedure using 2d patterns for automated navigation, In *2009 IEEE Intelligent Vehicles Symposium*, IEEE, 2009 pp. 117–122. ⇒42

[7] Z. Kató, L. Czúni, *Számítógépes látás. Egyetemi tananyag*, Typotex, 2011. ⇒43, 44, 45

[8] Zs. L. Kucsván. Data set used for the research, `https://drive.google.com/open?id=18AbYIyJLjxRKZjbxb1hyALruduCgt8qO`. ⇒45

[9] P. Moulon, P. Monasse, R. Marlet et al. Openmvg, `https://github.com/openMVG/openMVG`. ⇒45

[10] H. Stewenius, F. Schaffalitzky, D. Nister. How hard is 3-view triangulation really? *Tenth IEEE Int. Conf. on Computer Vision, 2005. ICCV 2005.*, vol. 1, IEEE, 2005 pp. 686–693. ⇒43

[11] R. Y. Tsai, An efficient and accurate camera calibration technique for 3d machine vision, *Proc. of Comp. Vis. Patt. Recog.*, 1986, pp. 364–374. ⇒44

Received: December 17, 2018 • Revised: May 30, 2019

Acta Univ. Sapientiae, Informatica **11,** 1 (2019) 52–64

DOI: 10.2478/ausi-2019-0004

New results on connected dominating structures in graphs

Libin Chacko SAMUEL
CHRIST (Deemed to be University)
email:
libin.samuel@res.christuniversity.in

Mayamma JOSEPH
CHRIST (Deemed to be University)
email:
mayamma.joseph@christuniversity.in

Abstract. A set of vertices in a graph is a dominating set if every vertex not in the set is adjacent to at least one vertex in the set. A dominating structure is a subgraph induced by the dominating set. Connected domination is a type of domination where the dominating structure is connected. Clique domination is a type of domination in graphs where the dominating structure is a complete subgraph. The clique domination number of a graph G denoted by $\gamma_k(\mathsf{G})$ is the minimum cardinality among all the clique dominating sets of G. We present few properties of graphs admitting dominating cliques along with bounds on clique domination number in terms of order and size of the graph. A necessary and sufficient condition for the existence of dominating clique in strong product of graphs is presented. A forbidden subgraph condition necessary to imply the existence of a connected dominating set of size four also is found.

1 Introduction

The study of domination in graphs is to a great extent a result of the study of games and recreational mathematics. It began when C.F. De Jaenisch attempted to determine the minimum number of queens that can be placed on

Computing Classification System 1998: G.2.2
Mathematics Subject Classification 2010: 05C69
Key words and phrases: dominating structure, clique domination, connected domination, strong product, tensor product

an $n \times n$ chess board so that all squares are either attacked by a queen or are occupied by a queen [10]. Domination in graph can be defined in a similar terms as finding a set of vertices in a graph such that every vertex in the graph is either adjacent to some vertex in the set or is in the set. Further development in domination was observed in late 1950s with Claude Berge [3] introducing coefficient of external stability which is now known as domination number. A set of vertices in a graph is a *dominating set* if every vertex in the graph which is not in the dominating set is adjacent to one or more vertices in the dominating set. The *domination number*, $\gamma(G)$, of a graph G is the minimum number of vertices in a dominating set. Over the course of time different types of domination in graphs such as total domination, connected domination and independent domination were developed by imposing conditions on the dominating set. For example a *connected dominating set* is a dominating set that induces a connected subgraph. [17, 9, 7, 12, 2, 16].

A dominating structure in a graph is a subgraph induced by its dominating set. Identification of graphs possessing specific types of dominating structures is a problem that caught the attention of several researchers. In this paper we are exploring graphs having complete graphs as a dominating structure. Every graph referred to in this article is finite, undirected, simple and connected. [5, 14, 4] A *clique dominating set* is a dominating set that induces a complete subgraph. A *clique dominated graph* is a graph that contains a clique as a dominating structure. Cozzens and Kelleher were the first to deal with dominating cliques. The *clique domination number*, $\gamma_k(G)$, of a graph G is the minimum number of vertices in a clique dominating set.

The concept of domination is very useful to model several real-world problems such as social networks , bus routing, land surveying, computer and communication networks . Facility allocation is another area wherein one finds one of the most important applications of domination; in particular connected domination and clique domination. It involves optimal placement of facilities in a given area.[6, 7, 8] For example let us consider the problem of effective allocation of airports and air routes of a country. The airports in important cities of a country are connected with each other, while every other airport is connected with that of at least one of the important cities. Another instance is a wireless sensor network which is comprised of autonomous sensor nodes where the connected dominating set enable faster communication by forming a virtual network backbone for information and control routing.[15, 11, 13, 1]

2 Related results

It is note worthy that every graph need not have a dominating clique. The smallest clique being K_1, the smallest dominating clique is a single vertex. It is clear that a graph with a dominating vertex has a star as spanning tree. Wolk [18] gave the necessary condition for the graphs to have dominating clique of size one and he called such a dominating clique a central vertex or a central point. Dominating clique of size two is an edge called dominating edge.

Theorem 1 (Wolk [18]) *If G is a finite connected graph with no induced P_4 or C_4, then G has a dominating vertex.*

Cozzens and Kelleher [5] extended the theorem to get a forbidden subgraph condition to establish the existence of a dominating clique, which is presented below

Theorem 2 (Cozzens and Kelleher [5]) *If G is a connected graph that has no induced P_5 or C_5 then G has a dominating clique.*

Although the above result ensures the existence of a dominating clique, it does not specify the size of the dominating clique. In the direction,Cozzens and Kelleher [5] have explored the problem of identifying graphs possessing connected dominating set of size 3.

The notation K_{n+p} [5] represents the complete graph K_n on n vertices along with n pendants, one at each vertex of the complete graph. For example K_{3+p} is the net graph. K_{3+p} and K_{4+p} are shown in the Figure 1.

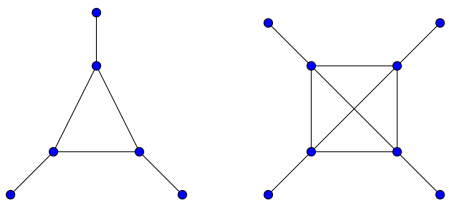


Figure 1: The graphs K_{3+p} and K_{4+p}

Note that connected dominating sets of size one and two respectively are defined uniquely whereas a connected dominating set of size three is either a P_3 or a K_3.

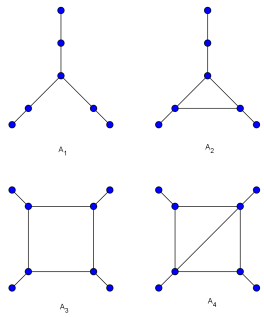


Figure 2: The graphs A_1, A_2, A_3 and A_4

The characterization obtained by Cozzens and Kelleher [5] was in terms of a class $\mathcal{A} = \{P_6, C_6, K_{4+p}, A_1, A_2, A_3, A_4\}$ of graphs where the graphs A_1, A_2, A_3 and A_4 are as shown Figure 2.

Theorem 3 (Cozzens and Kelleher [5]) *If* G *is a finite, connected graph with three or more vertices that has none of the graphs in Class* $\mathcal{A}$ *as an induced subgraph, then* G *has a connected dominating set of size three.*

We have settled the problem of obtaining a necessary condition for graphs to have a connected dominating set of size of 4 and the result is presented section 5. First we will explore the bounds for clique domination number γ_k.

3 Bounds for clique domination number

Recall that a private neighbour of a vertex v with respect to the set K is a vertex adjacent to only v from the set K. First we present the following proposition.

Proposition 4 *If* K *is a minimal dominating clique of a graph* G*, then every vertex in* K *has a private neighbour.*

Proof. On the contrary, assume that there is a vertex $v \in K$ having no private neighbor. Then v is adjacent to every vertex in K and v will have no private neighbour. This implies that $K - \{v\}$ is a smaller dominating clique contained in K, which contradicts the minimality of K. □

The bound obtained by Ore [10] for domination number, is true for clique domination as given below.

Proposition 5 *If a connected graph* G *of order* n *has a dominating clique, then* $\gamma_k(G) \leq n/2$.

Proof. Assume that $\gamma_k(G) > n/2$. Then γ_k-set of G being a minimal dominating clique, it is clear that there exists a vertex $v \in \gamma_k$-set of G which does not have a private neighbor which is a contradiction to the proposition 3.1. □

Remark 6 *The bound obtained in Proposition 3.2 is sharp and* K_{n+p} *is a class of graph that attains the bound. There are* $2n$ *vertices in* K_{n+p} *and the minimum dominating set is of size* n.

It is obvious that the domination number serves as a lower bound for clique domination number. Then the following inequality follows immediately.

Proposition 7 *If the graph* G *has a dominating clique, then* $\gamma(G) \leq \gamma_k(G) \leq \omega(G)$ *where* $\omega(G)$ *is the clique number of the graph.*

Remark 8 *Let* G *and* H *be two graphs. The corona product* $G \circ H$, *is the graph obtained by taking one copy of* G *and* $|V(G)|$ *copies of* H *and by joining each vertex of the* i*-th copy of* H *to the* i*-th vertex of* G, *where* $1 \leq i \leq |V(G)|$. *The graph* $K_r \circ K_s$, *where* $r \geq s$ *has* $\gamma(K_r \circ K_s) = \gamma_k(K_r \circ K_s) = \omega(K_r \circ K_s) = r$.

The next theorem gives a bound for the clique domination number of a graph in terms of its size.

Theorem 9 *If* G *is a graph with* m *edges possessing a dominating clique, then*

$$\gamma_k(G) \leq \frac{\sqrt{1+8m}-1}{2}.$$

Proof. We know that a clique of size γ_k has $\frac{\gamma_k(\gamma_k-1)}{2}$ edges. Therefore, $m \geq \frac{\gamma_k(\gamma_k-1)}{2} + \gamma_k$ so that $m \geq \frac{\gamma_k(\gamma_k+1)}{2}$. By solving which we will get $\gamma_k(G) \leq \frac{\sqrt{1+8m}-1}{2}$ or $\gamma_k(G) \leq \frac{-\sqrt{1+8m}-1}{2}$. Latter being impossible can be neglected. Hence $\gamma_k(G) \leq \frac{\sqrt{1+8m}-1}{2}$. □

Remark 10 *We can observe that the bound is sharp and* K_{n+p} *is a class of graph that attains the bound. Figure 1 shows the graphs* K_{3+p} *and* K_{4+p}.

Obviously a graph G with maximum degree $\Delta = n-1$ has a dominating vertex as the vertex with degree $n-1$ itself is a dominating vertex. We now consider graphs with $\Delta = n-2$ and obtain the following theorem.

Theorem 11 *If G is a connected graph with maximum degree $\Delta = n-2$, then G has a dominating edge.*

Proof. Since $\Delta = n-2$, there exists a vertex, say v in G which is adjacent to all but one vertex (obviously excluding v), say w, of the graph. But G is a connected graph and w is not adjacent to v which implies that w is adjacent to a neighbor of v say u, so that uv is a dominating edge of G. □

4 Clique domination in product of graphs

Clique domination problem for two types of graph product namely Lexicographical product and Cartesian product has been studied[4]. We now extend for clique domination in tensor products strong products of graphs.

The tensor product $G \times H$ of graphs G and H is a graph such that the vertex set of $G \times H$ is the Cartesian product $V(G) \times V(H)$; and any two vertices (u, v) and (u', v') are adjacent in $G \times H$ if and only if u is adjacent with u' in G and v is adjacent with v' in H.[17, 9]

We can understand by the definition of tensor product of graphs that any vertex (u', v') is not adjacent to any other vertex (u', v_i) and (u_j, v'), $\forall v_i \in V(H)$ and $\forall u_j \in V(G)$. For any two graphs G and H, $\gamma(G \times H) \geq 2$. For any graph G of order n, $G \times K_1$ is $\overline{K_n}$. And $G \times K_2$ is a bipartite graph.

Proposition 12 *For complete graphs K_r and K_s, $\gamma_k(K_r \times K_s) = 3$, if $r,s \geq 3$*

Proof. We can observe that the tensor product of two complete graphs K_n and K_m is a graph with any vertex (u_i, v_j) is adjacent to all vertices (u_k, v_l), $\forall k \neq i$ and $\forall l \neq j$ Therefore, by choosing three vertices (u_{i_1}, v_{j_1}), (u_{i_2}, v_{j_2}) and (u_{i_3}, v_{j_3}) where $i_1 \neq i_2 \neq i_3$ and $j_1 \neq j_2 \neq j_3$ we obtain a dominating clique, thus proving that $\gamma_k(K_r \times K_s) \leq 3$. As we have observed earlier, we require at least two vertices to dominate a graph. And if we consider an edge, the two vertices in the edge say (u_{i_1}, v_{j_1}) and (u_{i_2}, v_{j_2}) can dominate all the vertices but (u_{i_1}, v_{j_2}) and (u_{i_2}, v_{j_1}), hence the graph cannot be dominated by an edge. Therefore K_3 is the smallest clique dominating the tensor product $K_r \times K_s$ □
The strong product $G \boxtimes H$ of two graphs G and H is the graph with $V(G \boxtimes H) = V(G) \times V(H)$ and $(u, u')(v, v') \in E(G \boxtimes H)$ if and only if either $uv \in E(G)$

and $u' = v'$ or $u = v$ and $u'v' \in E(H)$ or $uv \in E(G)$ and $u'v' \in E(H)$. Note that if $C \subseteq V(G \boxtimes H)$, then the G-projection and H-projection of C are, respectively, the sets $C_G = \{u \in V(G) \ : \ (u, b) \in C \text{ for some } b \in V(H)\}$ and $C_H = \{v \in V(H) \ : \ (a, v) \in C \text{ for some } a \in V(G)\}$. [17, 9]

Theorem 13 *The graph* $G \boxtimes H$ *has a dominating clique if and only if the graphs* G *and* H *have dominating cliques.*

Proof. Suppose $G \boxtimes H$ has a dominating clique. Let $C \subseteq V(G \boxtimes H)$ be the dominating clique of $G \boxtimes H$. Consider the projections C_G and C_H of C on G and H respectively. We claim that C_G is a dominating clique of G and C_H is a dominating clique of H. Strong product being commutative, it is sufficient to show that C_G is a dominating clique of G. Let u, $u' \in C_G$ be distinct vertices. By the definition of projection we can observe that there exist adjacent vertices (u, v) and (u', v') in C. We know that (u, v) and (u', v') are adjacent in $G \boxtimes H$ implies that either $uu' \in E(G)$ and $v = v'$ or $u = u'$ and $vv' \in E(H)$ or $uu' \in E(G)$ and $vv' \in E(H)$. Since u and u' are distinct we can easily conclude that $uu' \in E(G)$. Therefore C_G forms a clique in G. Now, to show that C_G is a dominating set. Let $u_1 \notin C_G$ be vertex of G. There exists a vertex (u_1, v_1) in $G \boxtimes H$. Since C is a dominating clique in $G \boxtimes H$, there exists a vertex $(u_0, v_0) \in C$ adjacent to (u_1, v_1). Since u_0 and u_1 are distinct, by definition of an edge in strong product u_1 and u_0 are adjacent. Therefore, C_G is a dominating clique of G.

Conversely, let S_G and S_H be the dominating cliques in the graphs G and H. We claim that $S_G \times S_H$ forms a dominating clique in $G \boxtimes H$. Firstly to show that $S_G \times S_H$ is a clique in $G \boxtimes H$. Let (u, v) and (u', v') be two distinct vertices in $S_G \times S_H$. Either $u = u'$ or $uu' \in E(G)$ and $v = v'$ or $vv' \in E(H)$. Either ways (u, v) and (u', v') are adjacent. Hence, $S_G \times S_H$ is a clique in $G \boxtimes H$. Now to show that $S_G \times S_H$ dominates $G \boxtimes H$. Consider a vertex (u_1, v_1) not in $S_G \times S_H$. If u_1 not in S_G then there exists a u_0 in S_G adjacent to u_1 in G and a v_0 in S_H where $v_0 = v_1$ or v_0 and v_1 are adjacent in H. By the definition of strong product of graphs (u_1, v_1) is adjacent to (u_0, v_0). And if u_1 is in S_G since (u_1, v_1) not in $S_G \times S_H$ there exist $v_0 \neq v_1$ in S_H dominating v_1 in H. Owing to the definition of strong product of graphs (u_1, v_1) is adjacent to (u_1, v_0). Therefore, $S_G \times S_H$ forms a dominating clique in $G \boxtimes H$. □

Theorem 14 *If* G *and* H *are connected graphs with dominating cliques, then* $\gamma_k(G \boxtimes H) = \gamma_k(G) \times \gamma_k(H)$

Proof. Let S_G and S_H be the γ_k sets of G and H respectively. We know that $S_G \times S_H$ forms a dominating clique in $G \boxtimes H$. This implies

$$\gamma_k(G \boxtimes H) \leq \gamma_k(G) \times \gamma_k(H)$$

To show that $\gamma_k(G \boxtimes H) \geq \gamma_k(G) \times \gamma_k(H)$ we need to show that $\forall\ (u_i, v_i) \in S_G \times S_H$, $(u_i, v_i) \in \gamma_k$-set of $(G \boxtimes H)$. $u_i \in S_G$ implies u_i has a private neighbor say u_1. Similarly v_i has a private neighbor v_1. We claim that (u_1, v_1) is a private neighbor of (u_i, v_i), i.e. there is no (u_2, v_2) adjacent to (u_1, v_1) in $S_G \times S_H$. If there exists a vertex, say, (u_2, v_2) adjacent to (u_1, v_1) then by definition of strong product $u_2 = u_1$ and $v_1v_2 \in E(H)$ or $u_1u_2 \in E(G)$ and $v_2 = v_1$ or $u_1u_2 \in E(G)$ and $v_1v_2 \in E(H)$ all contradicting the fact that u_1 is the private neighbor of u_i and v_1 is the private neighbor of v_i. Which implies that $\forall\ (u_i, v_i) \in S_G \times S_H$, $(u_i, v_i) \in \gamma_k$-set of $(G \boxtimes H)$. Hence $S_G \times S_H$ is a minimal dominating clique of $G \boxtimes H$. To show that $S_G \times S_H$ is a γ_k set of $G \boxtimes H$, assume the contrary, if $S_G \times S_H$ is not a γ_k set of $G \boxtimes H$, then there exist a smaller dominating clique T whose projections T_G and T_H forms a smaller dominating clique for G and H respectively hence contradicting the minimality of S_G and S_H. □

5 Graphs with connected dominating structure of order four

A forbidden subgraph condition necessary for a graph to have a connected dominating set of size three was found by Cozzens and Kelleher [5]. We discuss a forbidden subgraph condition necessary to have a connected dominating set of size four. There are 6 connected graphs on four vertices :- K_4, C_4, P_4, Claw ($K_{1,3}$), Paw and Diamond ($K_4 - e$). Therefore a connected dominating set of size four can be any of the above mentioned graph.

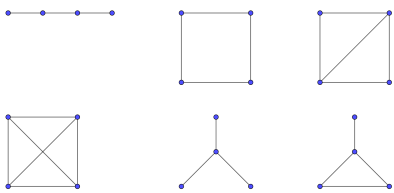


Figure 3: Connected graphs of order four

Theorem 15 *If G is a finite, connected graph with four or more vertices that has none of the graphs in $\mathcal{B}$ (Fig. 4) as an induced subgraph, then G has a connected dominating structure of order four.*

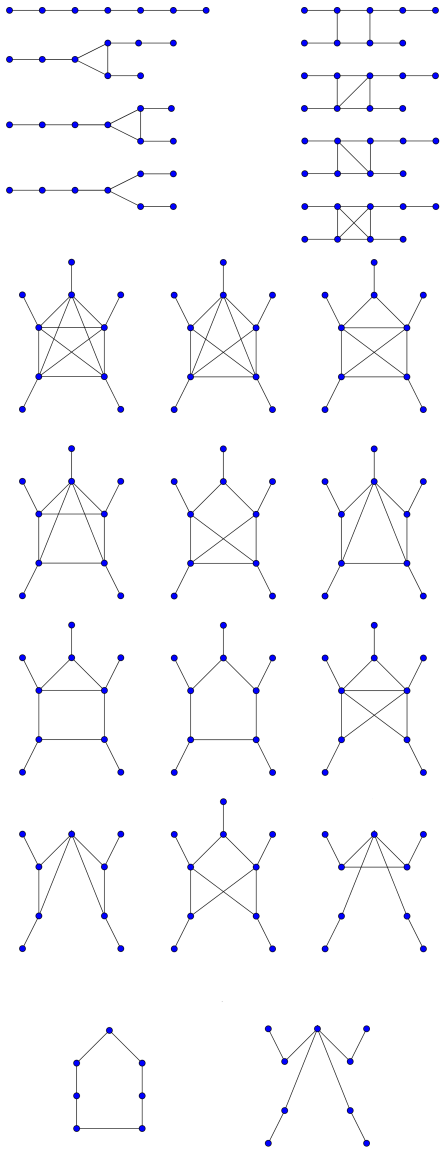


Figure 4: Class $\mathcal{B}$

Proof. By induction on n, the order of graph G,

(i) The theorem is true when $n = 4$.

(ii) Assume that any finite connected graph with n vertices, $n \geq 4$, that has none of the graphs in $\mathcal{B}$ as an induced subgraph has a connected dominating structure of size four.

(iii) Let G be a finite connected graph on $n + 1$ vertices, where $n \geq 4$, that has none of the graphs in class $\mathcal{B}$ as an induced subgraph. Let v be vertex of G which is not a cut vertex. Consider the graph G', subgraph of G induced by all vertices of G excluding v. Since G' is a finite connected graph with n vertices having no graphs from class $\mathcal{B}$ as an induced subgraph, by the induction hypothesis G' has a connected dominating structure of order four.

Let $S = \{a, b, c, d\}$ induce the connected dominating structure of order four of G'. If v is adjacent to any vertex in S, then S dominates G also.

Suppose that in G, v is not adjacent to any vertex in S. Since G is connected, v must be adjacent to some vertex of G, say x. And S being the connected dominating set of G', x must be adjacent to some vertex in S. The set $\{a, b, c, d, x\}$ induces a connected graph of 5 vertices. Therefore, the graph induced by $S \cup N(S) \cup \{v\}$ has one of the graphs from $\mathcal{B}$ as a subgraph, not necessarily induced, i.e, there might be edges between the pendant vertices and other vertices. If there are no edges between the pendant vertices and the other vertices, this implies that the subgraphs are induced, which contradicts the assumption that G has none of the graphs in class $\mathcal{B}$ as an induced subgraph.

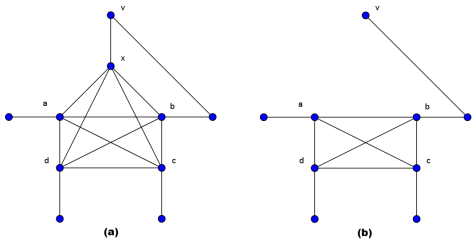


Figure 5: Graphs used in the proof of Theorem 15

Suppose G has at least one edge between the pendant vertices. If G has exactly one edge between vertices as shown in Figure 5(a) , then G has an induced subgraph shown in Figure 5(b), which is a forbidden subgraph from class $\mathcal{B}$.

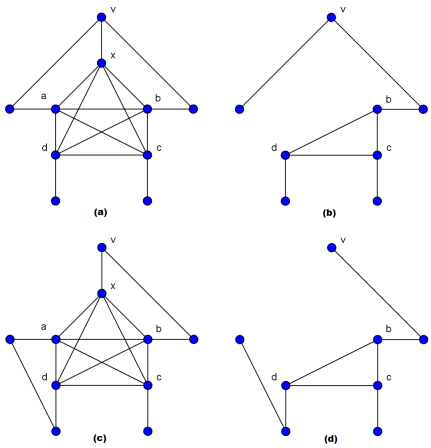


Figure 6: Graphs used in the proof of Theorem 15

If G has exactly two edges between the pendant vertices as shown in Figure 6(a) or 6(c), then G has an induced subgraph shown in Figure 6(b) or 6(d) , which is a forbidden subgraph from class $\mathcal{B}$.

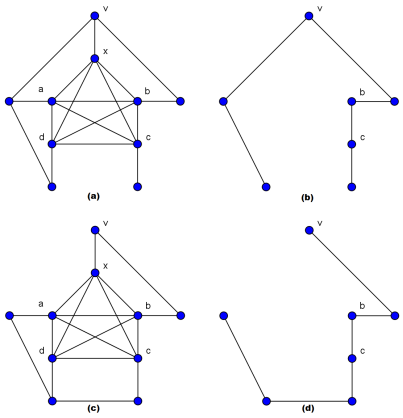


Figure 7: Graphs used in the proof of Theorem 15

If G has exactly three edges between the pendant vertices as shown in Figure 7(a) or 7(c), then G has an induced P_7, which is a forbidden subgraph from class $\mathcal{B}$.

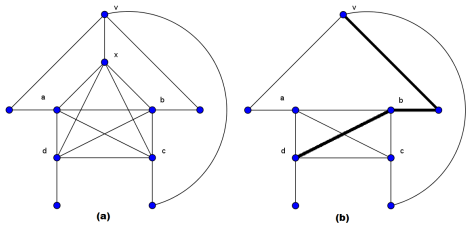


Figure 8: Graphs used in the proof of Theorem 15

If G has exactly three edges between the pendant vertices as shown in Figure 8(a), then G has a P_4 as shown in Figure 8(b) which is a connected dominating structure of order four.

We can now observe that an edge between the pendant vertices in the graphs in Class $\mathcal{B}$ will lead to obtaining a connected dominating structure of order four or a contradiction to the absence of an induced forbidden structure from class $\mathcal{B}$. Therefore, G has a connected dominating set of size four. □

As we have seen before, the converse of this theorem need not be true. A finite connected graph having graph from $\mathcal{B}$ as an induced subgraph can have a dominating clique of size four. An example is given in Fig. 9.

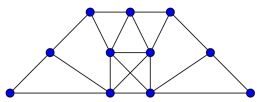


Figure 9: Graph with induced P_7 dominated by K_4

References

[1] R. E. Atani, A. H. Karbasi, Application of dominating sets in wireless sensor networks, *International Journal of Security and Its Applications* **7** (2013) 185–202. ⇒53

[2] S. Balamurugan, I. Sahul Hamid, Isolate domination in graphs, *Arab Journal of Mathematical Sciences* **22** (2016) 232–241. ⇒53

[3] C. Berge, *The Theory of Graphs*, Dover Publications 2001. ⇒53

[4] S. R. Canoy, T. V. Daniel, Clique domination in a graph, *Applied Mathematical Sciences* **9** 116 (2015) 5749–5755. ⇒53, 57

[5] M. B. Cozzens, L. L. Kelleher, Dominating cliques in graphs, *Discrete Mathematics*, **86** (1990) 101–116. ⇒53, 54, 55, 59
[6] M. B. Cozzens, L. L. Kelleher, Dominating sets in social network graphs, *Mathematical Social Sciences* **16** (1988) 267–279. ⇒53
[7] E. J. Cockayne, R. M. Dawes, S. T. Hedetniemi, Total domination in graphs, *Networks*, **10** (1980) 211–219. ⇒53
[8] W. Duckworth, B. Mans, Connected domination of regular graphs, *Discrete Mathematics* **309** (2009) 2305–2322. ⇒53
[9] F. Harary, *Graph Theory*, Addison-Wesley, Reading, MA 1969. ⇒53, 57, 58
[10] T. W. Haynes, S. T. Hedetniemi, P. J. Slater, *Fundamentals of Domination*, MD, 1998. ⇒53, 55
[11] M. A. Henning, S. Mukwembi, Domination, radius, and minimum degree, *Discrete Mathematics* **157** (2009) 2964–2968. ⇒53
[12] I. M. James, *History of Topology*, UK, 1999. ⇒53
[13] N. J. Kothari, S. K. Vaidya, Some new results on distance k-domination in graphs, *International Journal of Combinatorics* (2013). ⇒53
[14] D. Kratsch, M. Liedloff, An exact algorithm for the minimum dominating clique problem, *Theoretical Computer Science* , **385** (2007) 226–240. ⇒53
[15] G. Mekiš, Lower bounds for the domination number and the total domination number of direct product graphs, *Discrete Mathematics*, **310** (2010) 3310–3317. ⇒53
[16] E. Sampathkumar, H. B. Walikar, The connected domination number of a graph, *Jour. Math. Phy. Sci.* **13** (1979) 607–613. ⇒53
[17] D. B. West, *Introduction to Graph Theory* (2nd ed.), Pearson Education, 2002. ⇒53, 57, 58
[18] E. S. Wolk, A note on "The comparability graph of a tree", *Proc. Amer. Math. Sot.*, **16** (1965) 17–20. ⇒54

Received: March 25, 2019 • Revised: June 24, 2019

Acta Univ. Sapientiae, Informatica **11,** 1 (2019) 65–79

DOI: 10.2478/ausi-2019-0005

Automatic detection of hard and soft exudates from retinal fundus images

Bálint BORSOS
Sapientia Hungarian University of Transylvania,
Cluj-Napoca, Romania
Dept. of Electrical Engineering, Târgu Mureş
email: bborsos55@gmail.com

László NAGY
Óbuda University, Budapest, Hungary
University Research, Innovation and Service Center
email: lnagy.priv@gmail.com

David ICLĂNZAN
Sapientia Hungarian University of Transylvania,
Cluj-Napoca, Romania
Dept. of Mathematics-Informatics, Târgu Mureş
email: iclanzan@ms.sapientia.ro

László SZILÁGYI
Sapientia Hungarian University of Transylvania,
Cluj-Napoca, Romania
Dept. of Electrical Engineering, Târgu Mureş
Óbuda University, Budapest, Hungary
University Research, Innovation and Service Center
email: lalo@ms.sapientia.ro

Computing Classification System 1998: I.2.1
Mathematics Subject Classification 2010: 68T10
Key words and phrases: image segmentation, diabetic retinopathy, exudate detection

Abstract. According to WHO estimates, 400 million people suffer from diabetes, and this number is likely to double by year 2030. Unfortunately, diabetes can have severe complications like glaucoma or retinopathy, which both can cause blindness. The main goal of our research is to provide an automated procedure that can detect retinopathy-related lesions of the retina from fundus images. This paper focuses on the segmentation of so-called white lesions of the retina that include hard and soft exudates. The established procedure consists of three main phases. The preprocessing step compensates the various luminosity patterns found in retinal images, using background and foreground pixel extraction and a data normalization operator similar to Z-transform. This is followed by a modified SLIC algorithm that provides homogeneous superpixels in the image. The final step is an ANN-based classification of pixels using fifteen features extracted from the neighborhood of the pixels taken from the equalized images and from the properties of the superpixel where the pixel belongs. The proposed methodology was tested using high-resolution fundus images originating from the IDRiD database. Pixelwise accuracy is characterized by a 54% Dice score in average, but the presence of exudates is detected with 94% precision.

1 Introduction

Retinopathy is a severe complication of diabetes, which can lead to partial or total loss of sight. Several million people are affected by retinopathy of various grades [14]. Retinopathy can be diagnosed via analysing the image of the retina, which is usually acquired with a fundus camera. In the clinical practice of developing countries, fundus images are recorded in certain regional hospitals, but they are sent to a central hospital for evaluation by a qualified human expert. In order to build a screening for mass population, it would be necessary to purchase more fundus cameras and train lots of humans to become qualified experts. While the first condition can be fulfilled by raising funds, the second condition regarding human experts is a more difficult one. This is why there is a strong need for well trained computer systems that can reliably separate obvious negative cases from suspected positive ones, and draw the attention of the human experts to the latter. This way it is possible to create screening systems without needing lots of more human experts.

Retinopathy can cause various lesions on the retina. Some examples are shown in Fig. 1. Microaneurysms and hemorrhages are collectively called red lesions, while hard and soft exudates together are referred to as white lesions. Although retinopathy is usually manifested with all these lesion types, just

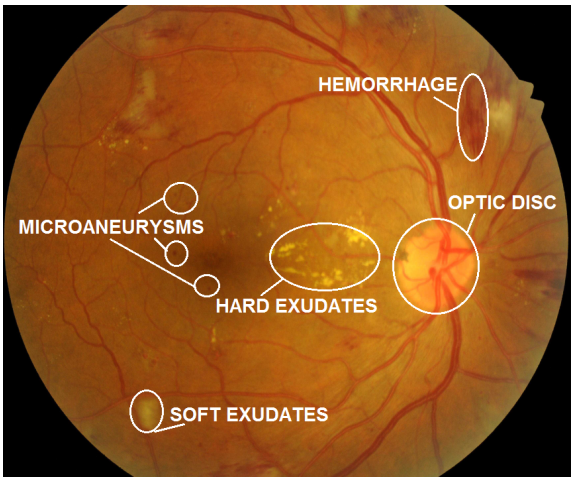


Figure 1: Retinal fundus image, indicating the anatomical parts relevant from the point of view of this study, and showing examples for the four types of the lesions. The original image was taken from the IDRid database [18].

as in the case shown in Fig. 1, the most studies from the literature focus on the detection or segmentation of only the red or the white lesions. To comply with this trend, we chose to dedicate the current study to the white lesion segmentation problem.

The computerized analysis of fundus images has been investigated for over two decades, and this process does not seem to slow down at all. The "Diabetic Retinopathy: Segmentation and Grading Challenge" [18] organized jointly with the IEEE International Symposium on Biomedical Imaging (ISBI 2018), with its high resolution retinal fundus image database, even seems to have an intensifying effect upon this research branch.

There are a few review articles in the field (e.g. [9, 16]), which give us a concise insight into the exudate detection problem. The methodology includes solutions based on: global and adaptive thresholding [19, 10], region growing [15], clustering in color space [7], morphological operations [17, 23], edge detection and mixture modeling [4, 5, 3], active contours and Naïve Bayes classifier [8], various supervised classifiers in competition [21], support vector machine [20, 6], perceptron network combined with graph-cut algorithm [13], circular Hough transform combined with CNN networks [2], and CNN with deep learning [11, 12].

This paper proposes a multi-step procedure for the exudate detection and segmentation problem, consisting of image intensity compensation, superpixel extraction, and supervised classification of pixels using a perceptron network, based on 15 features extracted from the neighborhood of the pixels and the superpixels they belong to. The proposed method will be evaluated using images from the IDRiD dataset [18].

The rest of this paper is structured as follows: Section 2 gives details on the proposed methodology. Section 3 exhibits and discusses the achieved results. Finally, Section 4 concludes the investigation.

2 Materials and methods

2.1 Data sets

This study relies on a subset of the Indian Diabetic Retinopathy Image Dataset (IDRiD) [18]. IDRiD contains 50 annotated positive images with soft and/or hard exudates, for which the optic disc mask is also available. Further 89 negative images were involved in the study, for these images we have produced optic disc masks. All these images were acquired using a Kowa VX-10 alpha digital fundus camera. Each positive image is accompanied by two masks that indicate the position of hard and soft exudates separately. We considered that the images in IDRiD have too high resolution (4288×2848 pixels) and too few components kept during JPEG encoding, so we resampled all retinal images and masks to 1072×712 pixels before proceeding to any processing step.

2.2 Data processing

The multi-step procedure proposed in this paper translates the image segmentation task into a classification problem, and provides a machine learning solution based on artificial neural networks (ANN) to complete the job. The classification takes place at the level of pixels. So each pixel is provided a feature vector, which includes properties of the neighborhood of the pixels, and properties of the superpixel it belongs to. This way the procedure needs to have the following steps (see Fig. 2): intensity compensation, superpixel generation, feature extraction, ANN training, ANN testing (prediction with ANN), and statistical evaluation.

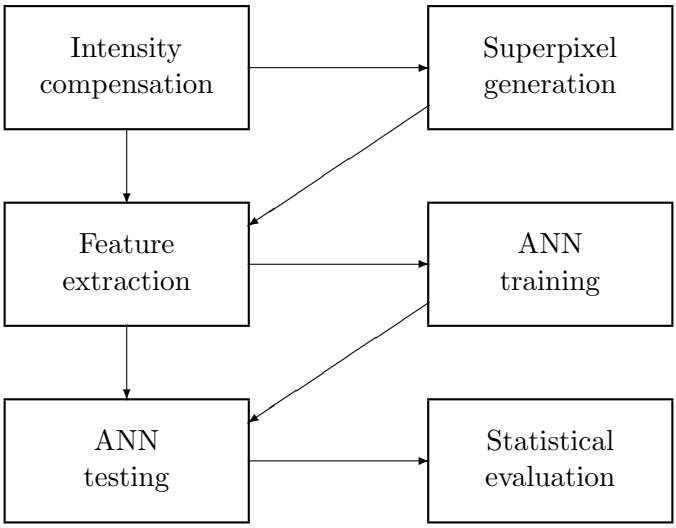


Figure 2: Block diagram of the proposed method.

2.3 Intensity compensation

A full processing of fundus images should be able to distinguish all anatomical structures and all lesions from the background. The ideal would be to have a constant background color. Unfortunately, the reality is usually very far from the ideal case. Some parts of most recorded fundus images appear darker than the others. This is why it is necessary to design a compensation scheme that would not affect the anatomical parts [22].

Our intensity compensation method is inspired from the work of Sánchez et al. [19]. First, we estimate which are the background pixel, and then we apply intensity correction based on the local statistics, average and standard deviation (STD) of intensities. Since the anatomical parts have significantly different color compared to their immediate neighborhood (background) and their edges are usually sharp, they are likely to be detected via thresholding. We performed a blur filter with mask sizes 39×39 pixels, and compared the blurred image from the original one. Wherever the absolute luminosity difference exceeds the half of the global intensity STD, those areas are declared anatomical parts. All other pixels are considered background pixels and they participate in the compensation process. At least 80% of the pixels belong to the background in most fundus images.

To produce a compensation of intensities based on background pixels, it is necessary to extract the local average and STD of intensities for every background pixels. Since this would be a very time consuming process, we adapted the recipe given in [19], and turned to an approximation scheme. The whole fundus image was divided into $4 \times 4 = 16$ equal rectangles, which in our case contained 268×178 pixels each. The average and STD intensity of background pixels situated in each rectangle (μ_i and σ_i, $i = 1 \ldots 16$) was then computed. Finally, the approximated average and STD intensity for each pixel (not only background ones) was interpolated from the values obtained in the four (or less in the proximity of margins) such rectangles, using as weighting coefficient the -1 power of the physical distance between the pixel and the center of the rectangle. This way the difference and STD values of neighbor pixels were found quite similar, which conforms to our previous expectation that the bias field varies smoothly along the original image.

The compensation of any pixel situated at coordinates (x, y) is finally performed with the formula:

$$\widetilde{I}(x, y) = \frac{I(x, y) - \hat{\mu}_{\text{neigh}}(x, y)}{\overline{\sigma}_{\text{neigh}}(x, y)} + \mu_{\text{global}} \ , \tag{1}$$

with

$$\overline{\sigma}_{\text{neigh}}(x, y) = \frac{\hat{\sigma}_{\text{neigh}}(x, y)}{\frac{1}{16} \sum_{i=1}^{16} \sigma_i} \ , \tag{2}$$

where $\hat{\mu}_{\text{neigh}}(x, y)$ and $\hat{\sigma}_{\text{neigh}}(x, y)$ represent the interpolated average and STD intensity at pixel (x, y), and μ_{global} is the desired average intensity of the compensated images, which can be a freely chosen value.

2.4 Superpixel generation

The images with compensated intensity obtained in the previous section were fed to a procedure that identified homogeneous spots or superpixels in them. The procedure was based on the so-called simple linear iterative clustering (SLIC) [1] algorithm. The original SLIC uses the k-means algorithm to cluster pixels using a composite distance function that includes components of physical distance and color difference. To assure the high speed of superpixel creation, when pixels are assigned to the closest cluster, only those cluster propotypes are tested which are situated within a short distance from the given pixel, as distant ones have no chance to be the closest according to the composite difference criterion.

SLIC offers the chance for the user to set the approximate size of clusters. As cluster centers are initially sampled on a regular grid Q pixels apart, the approximate number of pixels belonging to each cluster will be Q^2. In our implementation, the value of Q was set to 9, and the composite distance between two pixels at coordinates (x_1, y_1) and (x_2, y_2), having gray intensities g_1 and g_2, respectively, was considered as

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + \frac{1}{5}(g_1 - g_2)^2} \ . \tag{3}$$

With these settings, SLIC provided approximately 6300 superpixels in each image. As a further step, all such superpixels or clusters were analyzed from the point of view of their intensity distribution. Those clusters in which the standard deviation exceeded a predefined threshold value θ were further separated into two clusters using the k-means algorithm. In this case only the pixel intensities were considered and not their position within the cluster. The value of the threshold θ was chosen such a way, that $5 - 10\%$ of the clusters would be further separated into two parts. The value was chosen as $\theta = 7.7$ units on the $0 \ldots 255$ gray scale. Finally, the number of superpixels or clusters was close to 7000 in each image.

2.5 Feature generation

Pixels of the retina images were going to be classified using a supervised machine learning approach. A feature vector was extracted for each pixel of the 50 retina images, with the following composition:

- 1–3: pixel intensity in red, green, and blue channels;
- 4–9: minimum, maximum, and average of green channel intensities extracted from 5×5 and 11×11 sized neighborhood of the pixel;
- 10: distance of the pixel from the closest point of the optic disc;
- 11: size of the superpixels where the pixel belongs;
- 12–15: average value in green channel, and the 10, 50, and 90 percentile values of green channel intensity within the superpixel where the pixel belongs.

These fifteen features characterize each pixel of the retina images. In the next section we will attempt to distinguish exudates pixels from normal ones using an artificial neural network based approach.

2.6 ANN training and testing

An artificial neural network (ANN) was trained to separate exudates from other pixels of the retina images. The perceptron network employed for this problem consisted of four layers. The input layer has a dedicated neuron for each of the fifteen features listed in the previous subsection, the two hidden layers consists of seven neurons each, while there is a single neuron in the output layer.

To train the neural network, the "leave one out" technique was employed, namely we trained an ANN for each of the 50 retina images, using randomly selected pixels of the other 49 images as training data and their labeling as expected values. Train data for each network consisted of 4900 negative pixels, 600 hard exudate pixels and 500 soft exudate pixels.

The ANN deployed for hard and soft exudate detection was the one implemented in OpenCV ver. 3.1.0.

2.7 Evaluation criteria

Section 3 will provide a statistical evaluation of the obtained results. Pixel-wise evaluation is performed based on the number of true positives (TP), false negatives (FN), false positives (FP) and true negatives (TN), and the following accuracy indicators will be computed:

- Sensitivity or true positive rate (TPR), defined as $\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}$;
- Specificity or true negative rate (TNR), defined as $\mathrm{TNR} = \frac{\mathrm{TN}}{\mathrm{TN}+\mathrm{FP}}$;
- Dice score (DS), defined as $\mathrm{DS} = \frac{2\times\mathrm{TP}}{2\times\mathrm{TP}+\mathrm{FN}+\mathrm{FP}}$.

These statistical indicators will be analyzed separately in images with various amounts of exudate pixels.

3 Results and discussion

All 50 retina images with positive (exudate) pixels, and all 89 negative images underwent the above described processing steps. Figure 3 presents the intermediary results obtained for one of the images, starting from the original color image and its green channel, the region of interest consisting from the whole retinal part of the image with the optic disc removed, the estimated set of background pixels, the intensity compensated image and the superpixels

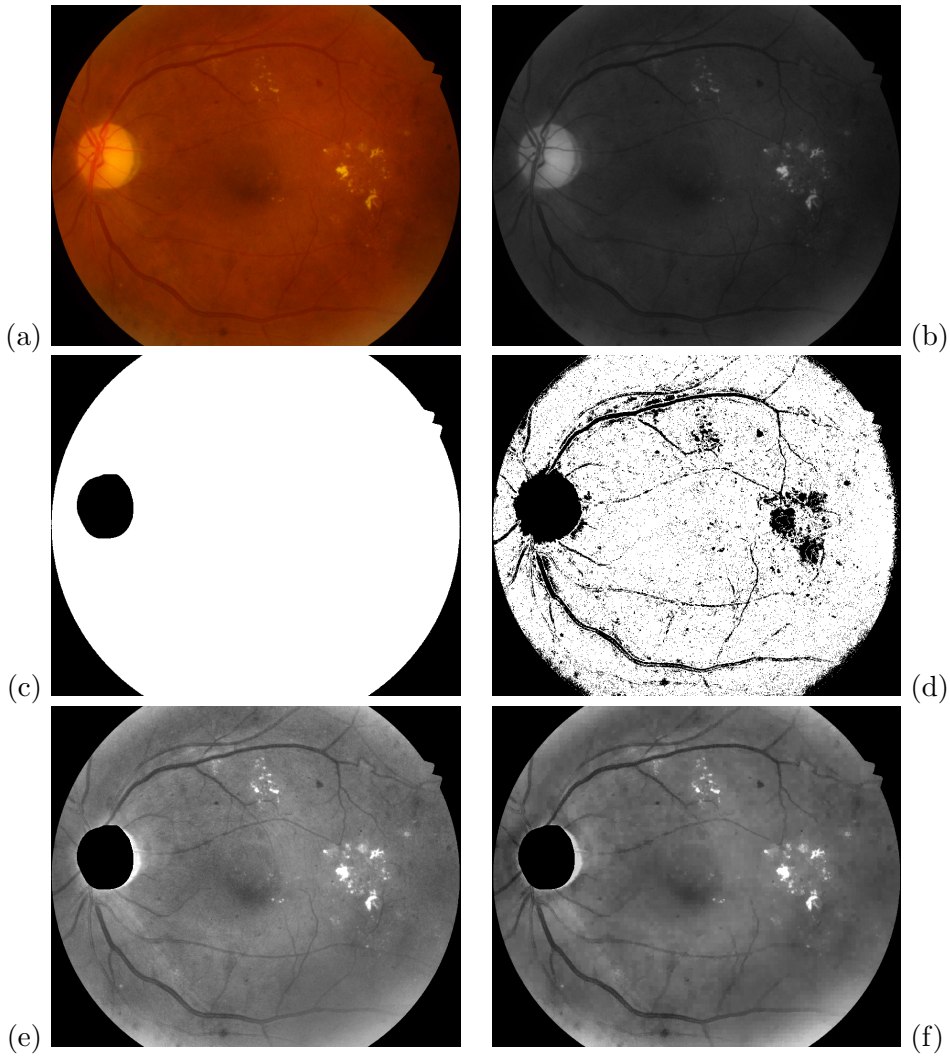


Figure 3: Intermediary results of the exudate detection procedure: (a) original color image; (b) the green channel of the original image; (c) region of interest: the whole retina image without the area of the optic disc; (d) the background pixels detected by the intensity compensation method; (e) the intensity compensated image; (f) the superpixels found by the modified SLIC algorithm.

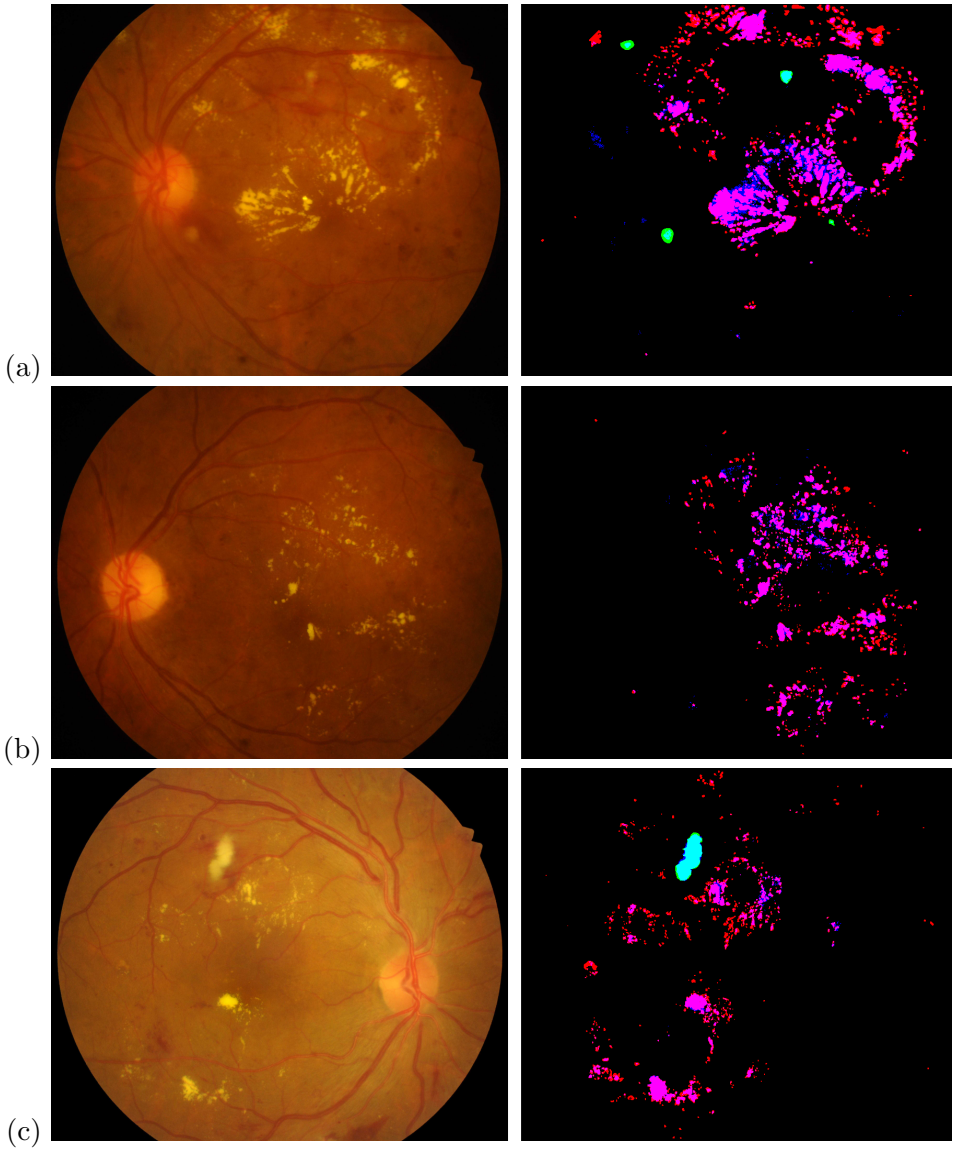


Figure 4: Three of the input retina images and the detected exudates: magenta and cyan represent detected hard and soft exudates, respectively; red and green indicate undetected hard and soft exudates, respectively; false positives are shown in blue.

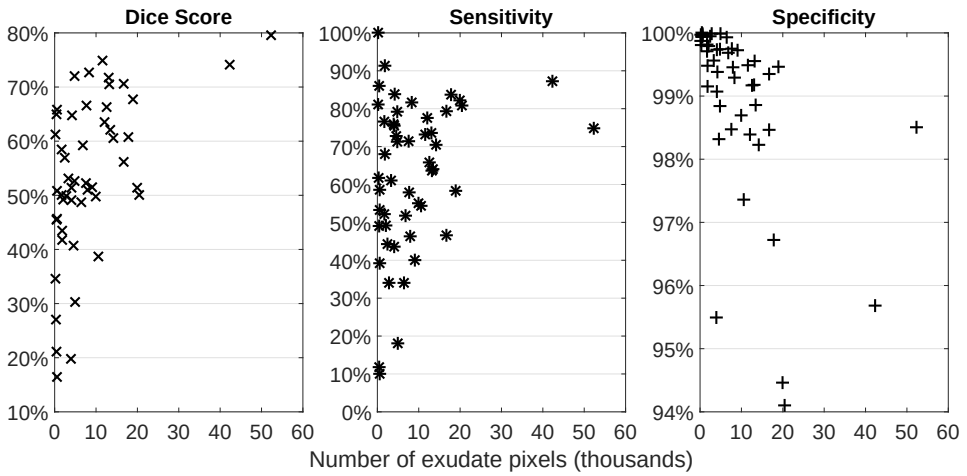


Figure 5: Segmentation quality indicators obtained for individual retina images, plotted against the actual number of exudate pixels in the image according to the ground truth.

identified in it. The feature vector was extracted for all pixels of all images involved in this study.

For each image with exudate pixels, a dedicated ANN was trained to obtain the segmentation, as described in Section 2.6. For the negative images, a single ANN was trained using pixels from all 50 images that contain positives. Detailed results of the segmentation are presented in the following.

Figure 4 presents the segmentation outcome for three images with positives. The left column shows the original color images, while the right column indicates the found and missed structures. The red, green and blue channels of the color image are used according to the following logic:

- red channel is set to maximum intensity wherever there is a hard exudate pixel according to the ground truth;
- green channel is set to maximum intensity wherever there is a soft exudate pixel according to the ground truth;
- blue channel is set to maximum intensity wherever the ANN estimates that there is an exudate pixel;
- all other pixels in all color channels are set to zero intensity.

Accuracy indicators	Cases with exudate pixel count				Average of all cases
	≥ 30000	≥ 10000	≥ 3000	≥ 1000	
Dice Score	76.83%	63.67%	57.85%	56.35%	53.75%
Sensitivity	80.99%	70.96%	65.43%	64.26%	62.42%
Specificity	97.09%	97.94%	98.52%	98.75%	98.99%

Table 1: Average values of the accuracy indicators for various subsets of the input images, defined in accordance to the actual number of exudate pixels in the images.

This notation means that correct decisions are denoted by magenta (identified hard exudates), cyan (identified soft exudates), and black (true negatives), while red (missed hard exudates), green (missed soft exudates), and blue (false positives) indicate wrong decisions. Larger spots of hard exudates are usually found, while smaller ones are more likely to be missed.

Figure 5 shows the main statistical quality indicator values obtained for individual retina images plotted against the number of actual exudate pixels present in the images. This figure shows that Dice Score and Sensitivity is generally higher in the segmentation outcome of images with larger number of exudate pixels. The variation os Specificity is apparently reversed.

Table 1 indicates the average values of main statistical quality indicators obtained in various subsets of the images that contain positive pixels. Subsets restrict the whole set of positive images to those, which contain at least a certain threshold number of positive pixels, the threshold varying from 1000 to 30000. As it is expected, images with more positive pixels are processed with better accuracy, at least from the point of view of Dice Score and Sensitivity.

If we consider the pixels of the whole set of positive images with their segmentation outcome, and compute the overall accuracy indicators for this whole set, we obtain $\mathsf{DS} = 60.52\%$, $\mathsf{TPR} = 68.98\%$, and $\mathsf{TNR} = 99.01\%$.

For the negative images it only makes sense to extract the Specificity value, which showed and average of 98.63%. For further development, it would be necessary to propose an additional processing step to suppress the false positives.

4 Conclusion

This paper proposed an image segmentation procedure to identify the presence of hard and soft exudates in retinal fundus images. The proposed method was tested on 50 positive and 89 negative cases taken from the IDRiD database. The sensitivity values above 1/3, which we consider enough to detect the presence of exudates, was found in 94% of the positive cases, making the proposed method a good candidate for integration into a future screening application.

Acknowledgements

The work of B. Borsos was funded by the Szekely Forerunner Federation. The work of L. Szilágyi was supported by the ÚNKP 18-4 New National Excellence Program of the Ministry of Human Capacities of Hungary, contract no. OE-OF, 325/6, 2018. L. Szilágyi is Bolyai Research Fellow of the Hungarian Academy of Sciences.

References

[1] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, S. Süsstrunk, SLIC superpixels compared to state-of-the-art superpixel methods, *IEEE Trans. PAMI* **34** (2012) 2274–2282. ⇒70

[2] K. Adem, Exudate detection for diabetic retinopathy with circular Hough transformation and convolutional neural networks, *Expert Syst. Appl.* **114** (2018) 289–295. ⇒67

[3] C. Agurto, V. Murray, H. Yu, J. Wigdahl, M. Pattichis, S. Nemeth, S. Barriga, P. Soliz, A multiscale optimization approach to detect exudates in the macula, *IEEE J. Biomed. Health Inf.* **18,** 4 (2014) 1328-1337. ⇒67

[4] K. S. Deepak, J. Sivaswamy, Automatic assessment of macular edema from color retinal images, *IEEE Trans. Med. Imag.* **31,** 3 (2012) 766-776. ⇒67

[5] M. Esmaeili, H. Rabbani, A. M. Dehnavi, A. Dehghani, Automatic detection of exudates and optic disc in retinal images using curvelet transform, *IET Image Proc.* **6** (2012) 1005–1013. ⇒67

[6] L. Giancardo, F. Meriaudeau, T. P. Karnowski, Y. Q. Li, S. Garg, K. W. Tobin Jr., E. Chaum, Exudate-based diabetic macular edema detection in fundus images using publicly available datasets, *Med. Image Anal.* **16,** 1 (2012) 216–226. ⇒67

[7] C. E. Hann, J. A. Revie, D. Hewett, J. G. Chase, G. M. Shaw, Screening for diabetic retinopathy using computer vision and physiological markers, *J. Diabetes Sci. Technol.* **3,** 4 (2009) 819–834. ⇒67

[8] B. Harangi, A. Hajdú, Automatic exudate detection by fusing multiple active contours and regionwise classification, *Comput. Biol. Med.* **54** (2014) 156–171. ⇒67

[9] S. Joshi, P. T. Kerule, A review on exudates detection methods for diabetic retinopathy, *Biomed. Pharmacoter.* **97** (2018) 1454–1460. ⇒67

[10] J. Kaur, D. Mittal, A generalized method for the segmentation of exudates from pathological retinal fundus images, *Biocybern. Biomed. Eng.* **38,** 1 (2018) 27–53. ⇒67

[11] P. Khojasteh, L. A. Passos Júnior, T. Carvalho, E. Rezende, B. Aliahmad, J. P. Papa, D. K. Kumar, Exudate detection in fundus images using deeply-learnable features, *Comput. Biol. Med.* **104** (2019) 62–69. ⇒67

[12] P. Khojasteh, B. Aliahmad, D. K. Kumar, A novel color space of fundus images for automatic exudates detection, *Biomed. Sign. Proc. Control* **49** (2019) 240–249. ⇒67

[13] W. Kusakunniran, Q. Wu, P. Ritthipravat, J. Zhang, Hard exudates segmentation based on learned initial seeds and iterative graph cut, *Comput. Meth. Prog. Biol.* **158** (2018) 173–183. ⇒67

[14] J. L. Leasher, R. R. Bourne, S. R. Flaxman, J. B. Jonas, J. Keeffe, K. Naidoo, K. Pesudovs, H. Price, R. A. White, T. Y. Wong, S. Resnikoff, H. R. Taylor, et al., Global estimates on the number of people blind or visually impaired by diabetic retinopathy: a meta-analysis from 1990-2010, *Diabetes Care* **39** (2016) 1643–1649. ⇒66

[15] J. Lowell, A. Hunter, D. Steel, A. Basu, R. Ryder, E. Fletcher, L. Kennedy, Optic nerve head segmentation, *IEEE Trans. Med. Imag.* **23,** 2 (2005) 256-264. ⇒67

[16] M. R. K. Mookiah, U. R. Acharya, C. K. Chua, C. M. Lim, E. Y. K. Ng, A. Laude, Computer-aided diagnosis of diabetic retinopathy: a review, *Comput. Biol. Med.* **43** (2013) 2136–2155. ⇒67

[17] J. Nayak, P. S. Bhat, U. R. Acharya, C. Lim, M. Kagathi, Automated identification of different stages of diabetic retinopathy using digital fundus images, *J. Med. Syst.* **32** (2008) 107–115. ⇒67

[18] P. Porwal, S. Pachade, R. Kamble, M. Kokare, G. Deshmukh, V. Sahasrabuddhe, F. Meriaudeau, Indian Diabetic Retinopathy Image Dataset (IDRiD): A database for diabetic retinopathy screening research, *Data* **3,** 3 (2018) 25. ⇒67, 68

[19] C. I. Sánchez, M. García, A. Mayo, M. I. Lopez, R. Hornero, Retinal image analysis based on mixture models to detect hard exudates, *Med. Image Anal.* **13,** 4 (2009) 650–658. ⇒67, 69, 70

[20] D. Sidibé, I. Sadek, F. Mériaudeau, Discrimination of retinal images containing bright lesions using sparse coded features and SVM, *Comput. Biol. Med.* **62** (2015) 175–184. ⇒67

[21] R. Sohini, P. Dara, K. K. Parhi, DREAM: diabetic retinopathy analysis using machine learning, *IEEE J. Biomed. Health Inf.* **18,** 5 (2014) 1717-1729. ⇒67

[22] L. Szilágyi, S. M. Szilágyi, B. Benyó, Efficient inhomogeneity compensation using fuzzy c-means clustering models, *Comput. Meth. Prog. Biol.* **108** (2012) 80–89. ⇒69

[23] X. Zhang, G. Thibault, E. Decencière, B. Marcotegui, B. Laÿ, R. Danno, G. Cazuguel, G. Quellec, M. Lamard, P. Massin, A. Chabouis, Z. Victor, A. Erginay, Exudate detection in color retinal images for mass screening of diabetic retinopathy, *Med. Image Anal.* **18,** 7 (2014) 1026–1043. ⇒67

Received: July 9, 2019 • Revised: July 27, 2019

Acta Univ. Sapientiae, Informatica **11,** 1 (2019) 80–94

DOI: 10.2478/ausi-2019-0006

Gesture-Driven LEGO robots

Lehel István KOVÁCS
Sapientia Hungarian University of Transylvania,
Cluj-Napoca
Department of Mathematics and Informatics,
Târgu-Mureş, Romania
email: klehel@ms.sapientia.ro

Abstract. In this short survey and case study we want to present our research experience through the project developed by our team, that involves the building of a LEGO MINDSTORMS EV3 robotic arm and tracked robot car which mimics the motion of the human arm and legs. We used 3 interconnected LEGO MINDSTORMS EV3 bricks to reach the desired degrees of freedom. Using a Kinect sensor, the system detects the motion of the human user's arm and creates the skeletal image of the arm. Coordinate geometry and different approximation methods are used to calculate the rotation angles between the bones connecting the joints. In our project the key is inverse kinematics, whitch makes use of the kinematics equations to determine the joint rotation parameters that provide a desired position for each of the robot's end-effectors – arms and legs (wheels). The combined motion of the LEGO MINDSTORMS EV3 motors results in a complete robotic forward or backward motion and arm movement which is a perfect mimic of the human arm movement.

1 Introduction and motivation

The purpose of this paper is to introduce and compare different implementations of inverse kinematics, and to present a LEGO [16] robot and Kinect

Computing Classification System 1998: J.1.3
Mathematics Subject Classification 2010: 70E60
Key words and phrases: human-robot interaction, robotics, end-effectors, Kinect, LEGO MINDSTORMS EV3.

[15] (for motion capture) system that mimics the motion of the human arm as a case study. Our application implements five different algorithms to solve inverse kinematics problem. We compare and analyze the algorithms to select the most appropriate one.

Inverse kinematics as an animation technique plays an important role in computer animation and as we will see, robots can be controlled with the help of it.

During the last decades several algorithms have been presented to produce fast and realistic solutions to the inverse kinematics problem, but most of the available algorithms have high computational cost and produce unrealistic poses. The [1] technical report summarizes the research to date and the results achieved and methods developed. Based on the [10] book, using OpenGL [17], we have developed a graphical system that displays the results of the movements, the animation.

Kinect is used to copy human movements, we recognize gestures, and execute them with our specially built LEGO robot.

Most of the results achieved have been presented at conferences [11], and details and new solutions will be presented.

The body of this survey paper is divided into 6 sections. The first section introduce readers to the robot control and animation, the second section describes the rotation in plane and space as the basis of animation. Section 3 presents the Inverse Kinematics (IK) problem and discusses the most popular and developed solutions during the last couple of decades. Section 4 presents the Kinect sensor and introduces gesture recognition methods. And finally, Section 5 describes how our built LEGO robot works. The final section summarizes the conclusions of our work.

2 Robot control and animation

Robot control is the study and practice of controlling robots [3]. Animation is a method in which standing pictures are manipulated by quick changes to appear as moving images [13]. Commonly the effect of animation (the movie, or motion) is achieved by a rapid succession of sequential images – drawn or computer generated – that minimally differ from each other.

But what does robot control and animation have in common?

If we want to answer the question briefly, we could say that the techniques. In the longer term, we need to look specifically at the techniques and how to apply them.

When we talk about animation techniques, we distinguish between simple and complex animation. *Simple animation* is the *key frame animation* or the *program-controlled animation*. A keyframe in computer assisted or aided animation and cinematography is a drawing suite that defines the starting, the middle and ending points of any smooth transition [19]. The drawings, the pictures (the given particular images) are called "frames" because their position in time is measured in frames on a strip of old type of cellulose film.

In program controlled or program-driven animation, we write scripts that implement the animation. *Complex animation* is *motion caption*, *forward kinematics* and *inverse kinematics*. Motion capture (mo-cap or mocap) is the process of recording the movement of objects or people using differend kind of sensors. In animation and film making, it refers to recording actions (movements) of human or in some cases, not human actors, and using that information to animate digital character models (skeletons) in 2D or 3D computer aided animation. If we use Kinect or VR equipment, we can realize a full robot control using motion capture. The disadvantage of the method is, that only human-shaped (android, humanoid) robots can be controlled in this way. But how to control for example a spider- or dog-shaped robot? For these cases, and not only, forward kinematics and inverse kinematics are suitable. For these animations we need a bone/joint system.

In computer aided animation a designed figure, character is represented in two parts: a surface representation (called *skin* or *mesh*), this is used to draw the character and a hierarchical, recursive set of interconnected bones (called the *skeleton* or *rig*) used to animate the figure, the character.

Each bone has a 2D (in plane) or 3D (in space) transformation (which includes its position, scale and orientation in plane, space), and an optional parent bone. The bones therefore form a hierarchy. This is a hierarchical and recursive structure, because the full transform of a child node is the product of its parent transform and in plus its own transform. So moving a thigh-bone will move the lower leg too. As the character is animated during the process, the bones change automatically their transformation over time, under the influence of some animation controller [12].

The essential concept of forward kinematic animation – one of the animation techniques – is that the positions of particular parts of the model at a specified time are calculated from the position and orientation of the object, together with any information on the joints of an articulated model. In this model the animator must to calculate each position of every joints. So for example if the object to be animated is an arm with the shoulder remaining at a fixed location, the location of the tip of the thumb would be calculated

from the angles of the shoulder, elbow, wrist, thumb and knuckle joints. The avantage of forward kinematic is, that we have a very precise control [18]. The disadvantage: difficult for complex movements, for example we can not climbing stairs with forward kinematic control.

For complex movements the effective method ist the inverse kinematics. In kinematics an animated figure is modeled with a skeleton of rigid segments connected with joints, called a *kinematic chain*. The kinematics equations of the figure define the relationship between the joint angles of the figure and its pose or configuration. In inverse kinematics the orientation of articulated parts is calculated from the desired position of certain points on the model.We move only the end effector, and the program calculates the kinematic path of each bones. The joints bind the bones. The bone rotates around the joint in a given range of angle. The length of the bones does not change during the movement. Mathematically, the system is complex. The exact method exists for only two bones, the system becomes very complicated for more bones. Nonlinear optimization is required.

In our project we use inverse kinematics for robot controll, but also motion caption is required.

3 Rotation in plane (2D) and space (3D)

As we have seen, motion is accomplished by inverse kinematics, which is obtained with rotation of bones around the joints. Rotations are described using *Euler angles*. The Euler angles are three angles introduced by Leonhard Euler to describe the orientation in space of a rigid body (described by the *pitch*, *yaw* and *roll* movements) with respect to a fixed coordinate system. Any orientation can be achieved by composing three elemental rotations. We use the following angles: ψ around Z, θ around X, and φ around Y axis. In the plane, rotation is accomplished by complex numbers. The multiplication rules for complex numbers make them suitable for representing rotational quantities in two dimensions. Using complex numbers, we can describe rotations as follows: $CR = C_1 \cdot C_2$, where C_1 is a complex number representing the initial orientation, C_2 is a complex number representing the subsequent rotation, and CR is a complex number representing the final orientation. If we want to rotate with two angles, first one with θ, after that with φ, we can use the exponential form of the complex number: $z = r \cdot e^{i\theta}$. Then the rotation can be traced back to the product of two complex numbers because: $e^{i(\theta+\varphi)} = e^{i\theta} \cdot e^{i\varphi}$. If we

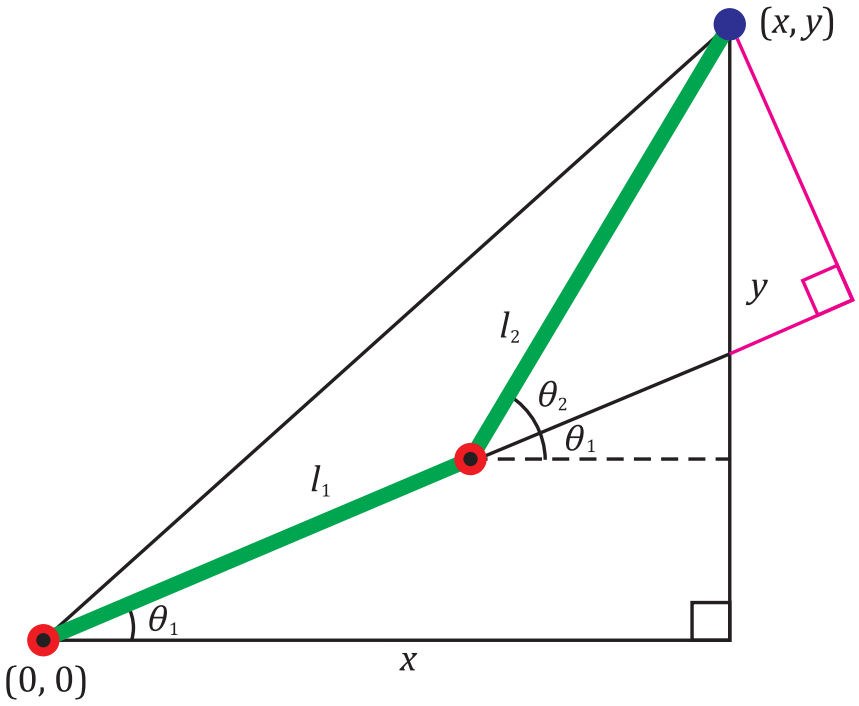


Figure 1: The analytical solution

want to repeat the rotation, the De Moivre theorem will simplify the situation: $(\cos\theta + i \cdot \sin\theta)^n = \cos n\theta + i \sin n\theta$.

In the space (3D) instead of complex numbers, we use *quaternions*. In mathematics, the quaternions are a number system that extends the complex numbers. They were first described by Irish mathematician William Rowan Hamilton in 1843 and applied to mechanics in three-dimensional space. Hamilton defined a quaternion as the quotient of two directed lines in a three-dimensional space or equivalently as the quotient of two vectors. The general form of a quaternion: $q = w + xi + yj + zk$, where $i^2 = j^2 = k^2 = -1$, and $ji = -k$, $kj = -i$, $ik = -j$. Thus, all operations can be defined on quaternions. Quaternions can be used to explain orientation and quaternion operations to track its changes.

For a given rotation axis, the rotation formula, using Euler angels, is:

$$\begin{aligned} q &= \cos\frac{\varphi}{2}\cos\frac{\theta}{2}\cos\frac{\psi}{2} - \sin\frac{\varphi}{2}\cos\frac{\theta}{2}\sin\frac{\psi}{2} \\ &+ i\cdot\left(\cos\frac{\varphi}{2}\sin\frac{\theta}{2}\cos\frac{\psi}{2} + \sin\frac{\varphi}{2}\sin\frac{\theta}{2}\sin\frac{\psi}{2}\right) \\ &+ j\cdot\left(-\cos\frac{\varphi}{2}\sin\frac{\theta}{2}\sin\frac{\psi}{2} + \sin\frac{\varphi}{2}\sin\frac{\theta}{2}\cos\frac{\psi}{2}\right) \\ &+ k\cdot\left(\sin\frac{\varphi}{2}\cos\frac{\theta}{2}\cos\frac{\psi}{2} + \cos\frac{\varphi}{2}\cos\frac{\theta}{2}\sin\frac{\psi}{2}\right). \end{aligned}$$

4 The solutions of inverse kinematic problem

As we said, the exact method to solve the inverse kinematic problem exists only for two bones, the system becomes very complicated for more bones. This is the *analytical solution.*

Problem statement:

There are two bones given, the first has one end in the origin $(0,0)$ and the second bone starts at the other end. With what kind of angles do you need to rotate the two bones so that the free end of the second bone reaches a given position (x,y)?

The solution of the problem, according to Figure 1 is:

$$\cos\theta_2 = \frac{x^2+y^2-l_1^2-l_2^2}{2l_1l_2}$$

and

$$\tan\theta_1 = \frac{y\cdot(l_1+l_2\cdot\cos\theta)-x\cdot l_2\cdot\sin\theta_2}{x\cdot(l_1+l_2\cdot\cos\theta)+y\cdot l_2\cdot\sin\theta_2}$$

Note: instead of $\mathtt{atan}$ function in computer science we use the $\mathtt{atan2}$ function. The definition of this function is:

$$\mathtt{atan2}(y,x) = \begin{cases} \mathtt{atan}(y/x), \text{ if } x>0, \\ \mathtt{atan}(y/x)+\pi, \text{ if } x<0 \text{ and } y\geq 0, \\ \mathtt{atan}(y/x)-\pi, \text{ if } x<0 \text{ and } y<0, \\ +\frac{\pi}{2}, \text{ if } x=0 \text{ and } y>0, \\ -\frac{\pi}{2}, \text{ if } x=0 \text{ and } y<0, \\ \text{undefined, if } x=0 \text{ and } y=0. \end{cases}$$

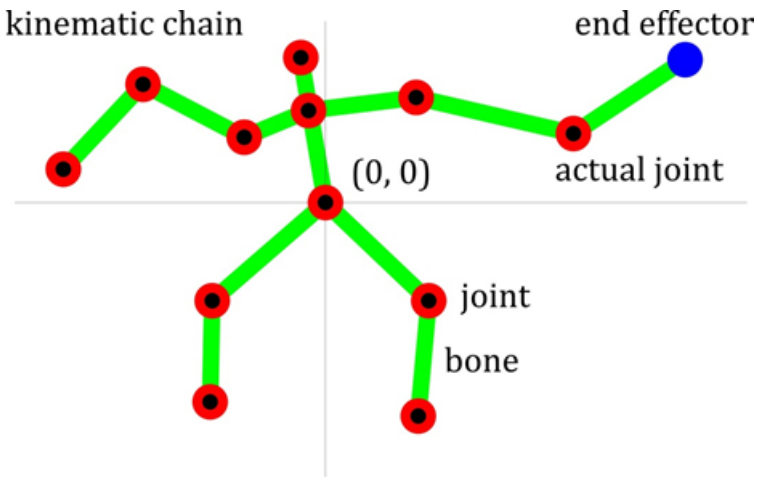


Figure 2: Kinematic chain – hierarchical movement

For more than two bones, we can solve the system by hierarchical movement based on Figure 2. We are iterating from the end effector to the base and optimizing each joint so that the last one comes close as possible to the target. So, we can have the same solution for more inverse kinematics problem.

Problem statement:

Known: The coordinates of end effector: $e = [e_1 e_2 ... e_N]$.

Unknown: The degree of freedom (DOF): $\theta = [\theta_1 \theta_2 ... \theta_M]$.

The solution of the problem: $\theta = f^{-1}(e)$.

Problems encountered: The f function is nonlinear. Calculation of the inverse function is not trivial. The solution is ambiguous, multiple states may have the same effector position.

The nonlinearity and ambiguity of inversion can be solved by an iteration that produces one of the possible solutions. The base idea of the iteration: if we known the position of end effector in a time moment t, we can calculate the position in time moment $t + \Delta t$. If Δt is small, we can approximate the function with its tangent (linear approximation).

In our research, a comparative analysis of four methods was made in this sense. We compared the approximation method based on *Jacobian matrix* [4], the correction of this method called *Damped Least Squares* [4], the *Cyclic Coordinate Descent* (CCD) method [14], and the *Constraint Relaxation* (CR) method [6, 9].

The Jacobian matrix is the matrix of the partial derivatives of the system: $J = \frac{\partial e}{\partial \theta}$. Then $\partial\theta = J^{-1} \cdot \partial e$.

If $e = [e_1 e_2 ... e_N]$ and $\theta = [\theta_1 \theta_2 ... \theta_M]$, then the Jacobian matrix is an $N \times M$ dimension matrix. So, we need a pseudoinverse: $J^+ = (J^T[M \times N] \cdot J[N \times M])^{-1} \cdot J^T[M \times N]$.

The state-changes in the pseudoinverse are minimal, so we get the possible movements in which the relative velocity of the bones in the joints is minimal. Iterative solution: in time moment t_0 all factors are known. The end effector is moved in small steps on the prescribed track, and in each step we calculate the change of state using the Jacobian matrix's pseudoinverse [13].

Algorithm 1: The Jacobian solution

$e = e(t_{start})$;
$\theta = \theta(t_{start})$;
for $t = t_{start}$ **to** t_{end} ***step*** Δt **do**
    Draw the animation if needed ;
    Compute the Jacobian-matrix J;
    Compute the pseudoinverse J^+;
    Compute $e(t + \Delta t)$;
    $\Delta e = e(t + \Delta t) - e(t)$;
    $\Delta\theta = J^+ \cdot \Delta e$;
    $\theta = \theta + \Delta\theta$;
end

For correction we can use the Levenberg–Marquardt algorithm (LMA, LM), also known as the Damped Least-Squares (DLS) method, which is used to solve non-linear least squares problems [20]. These minimization problems arise especially in least squares curve fitting. The essence of the method is that Instead of $\Delta\theta$, we minimize the following expression: $\|J\Delta\theta - \vec{e}\|^2 + \lambda^2 \|\Delta\theta\|^2$, where $\lambda \in \mathbb{R}$ is a damping constant depending on the target position and chosen so that the solution be numerically stable.

The Cyclic Coordinate Descent (CCD) method traces the calculation of θ to vector operations (scalar product, cross product, sin, cos, etc.).

The Constraint Relaxation (CR) algorithm is capable to calculate the exact position of any bone in an arbitrary long kinematic chain, and it is easy to implement in larger dimensions. The kinematic chain of joints without rotational constraints can also be understood as point clouds, with distances between points (definite length bones). The kinematic chain of joints without

rotational constraints can also be understood as point clouds, with distances between points (definite length bones). The essence of the algorithm is to stretch the bones one by one to the target and then restore the original size to get closer to the solution. This involves scaling of vectors.

In conclusion, we used the Damped Least-Squares (DLS) Constraint Relaxation (CR) and methods, which we found to be the most appropriate.

5 The Kinect, the skeleton, and the gestures

Developed by Microsoft, Kinect (initial codenamed Project Natal during development) is a line of motion sensing input devices for Xbox 360 and Xbox One video game consoles (release date: 2010) and Microsoft Windows PCs (release date: 2012) [21]. Based around a webcam-style add-on peripheral, it enables users to control and interact with their game console or personal computer without the need for a game controller, only through a natural user interface using gestures and spoken commands. The skeletal mapping technology shown in 2009 was capable of simultaneously tracking four people, with a feature extraction of 48 skeletal points on a human body at 30 Hz. There are two skeleton versions for Kinect 1 and Kinect 2. The Kinect can track up to six skeletons at one time. According to Figure 3 each of these skeletons has 25 joints.

The Kinect skeleton returns joints not bones [5]. The joints form a point cloud in space:

```
Vector4 skeletonPosition[NUI_SKELETON_POSITION_COUNT];
```

Each joint has 11 properties: color (x, y); depth coordinates (x, y); camera coordinates in homogeneous coordinate system (x, y, z, w); and orientation coordinates in homogeneous coordinate system (x, y, z, w), where w is the divider coordinate. In projective geometry *homogeneous coordinates* or *projective coordinates* were introduced by August Ferdinand Möbius in 1827 for handling the infinity. The color coordinates (x, y) are the coordinates of the joint on the image from the color camera. The depth coordinates (x, y) are the coordinates of the joint on the image from the depth camera. The Kinects camera coordinates use the Kinects infrared sensor to find 3D points of the joints in space. The camera space coordinates are handled differently from the color and depth coordinates. Kinect uses quaternions to deliver joint orientation [8].

Kinect's skeleton can be used for gesture recognition, and gestures can be used to control robots.

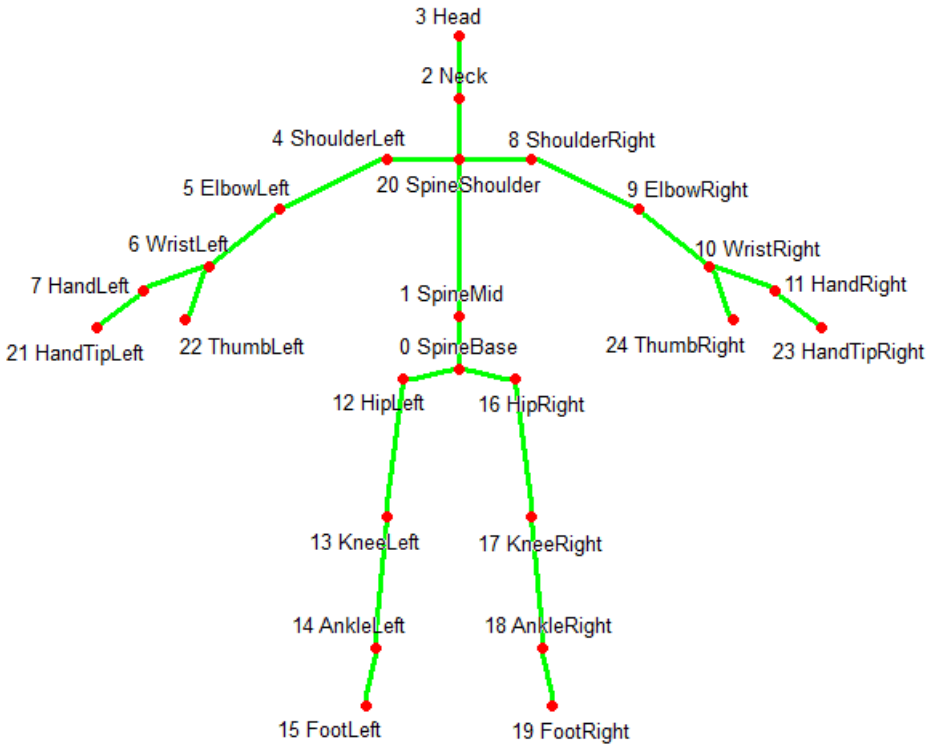


Figure 3: The skeleton of Kinect 2.0 [7]

A gesture is a form of non-verbal communication or non-vocal communication in which visible bodily actions communicate particular messages, either in place of, or in conjunction with, speech. Gestures include movement of the hands, legs, face, or other parts (mostly limbs) of the body. Gesture recognition is a topic in computer science with the goal of interpreting human gestures via mathematical algorithms. We used a 3D model-based skeletal gesture recognition technique.

The method we use assumes that each gesture is represented by 33 frames [2]. We use spherical coordinates, because coordinates are easier to normalize. Regardless of the size of the user, only the distance from the origin will vary, instead all the coordinates in the Cartesian system and the angles will remain constant.

The proposed algorithm for gesture recognition involves 4 steps:

1. detection of the human user,
2. extraction of the features,
3. a stage of warping, in which the gestures are compared to reference gestures,
4. gesture recognition.

Human detection is facilitated by the Kinect sensor, so step 1. is almost trivial. The most important feature of a gesture is limb movement. In step 3. we use a DTW - In time series analysis, *dynamic time warping* (DTW) is one of the algorithms for measuring similarity between two temporal sequences, which may vary in speed. For instance, similarities in walking could be detected using DTW, even if one person was walking faster than the other [2, 22].

The algorithm consists of the following steps:

Algorithm 2: Gesture recognition

```
Detection of the human user;
Extraction of the features;
DTW-warping;
Gesture recognition;
if YES then
    Robot action;
    Learning a new task;
end
```

Gesture recognition requires the following steps:

- geting the 33 frames video sequence
- geting the joint coordinates
- converting to spherical coordinates
- converting to normalized coordinates
- building the vector of features
- DTW-algorithm

DTW compares the sequence obtained of an unknown gesture with one or more reference patterns or templates. With several such templates, the recognition rate will be more big, but the calculation time increases. Having two sequences represented by the time series: $x = x_1, x_2, ..., x_n$, and $y = y_1, y_2, ..., y_n$,

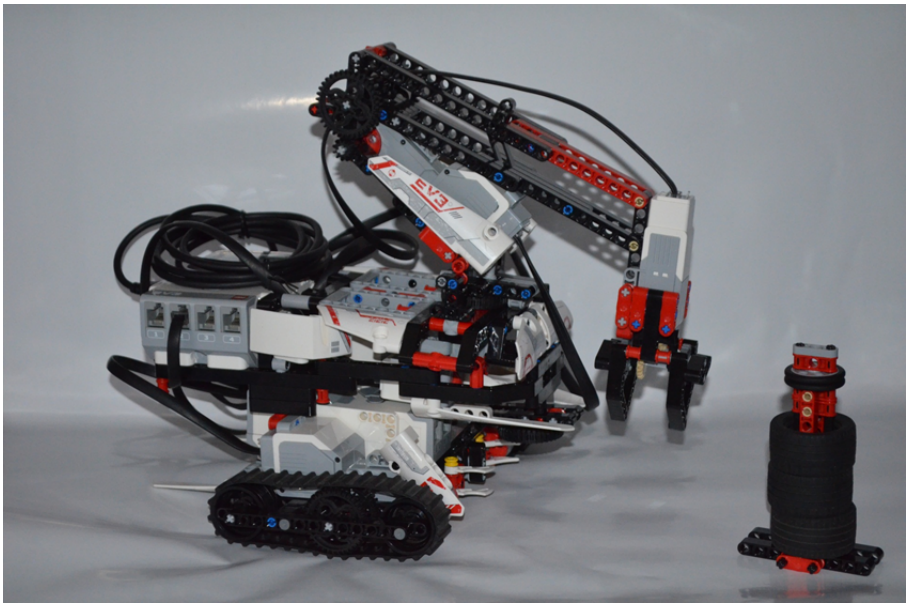


Figure 4: The LEGO robot

a matrix of $n \times m$ elements can be obtained, where each element of the matrix represents the distance between two elements of the time series, called the *cost of the matrix*. To find the best match between these two sequences, a matrix path must be found that minimizes the cumulative total distance between their elements. The *warping path* defines the mapping between the elements of the two time series: $w = w_1, w_2, ..., w_k, ...w_p$, where $DTW(x, y) = \min \sum_{k=1}^{p} d(w_k)$, and $d(w_k)$ represents the distance between elements x_i and y_j of the time series, that is: $d(x_i, y_j) = |x_i - y_j|$.

6 The LEGO robot

Using three interconnected bricks, we built an LEGO MINDSTORMS EV3 [16] robot (Figure 4), which we can control with gestures. Lego Mindstorms EV3 is the third generation robotics kit in Lego's Mindstorms line after NXT (2006) and RCX (1998). The home and education editions were released in 2013. The LEGO MINDSTORMS EV3 set includes motors (large and medium), sensors (touch, color, infrared gyroscopic, ultrasonic), the EV3 programmable brick, cables, more than 550 LEGO Technic elements and a remote control.

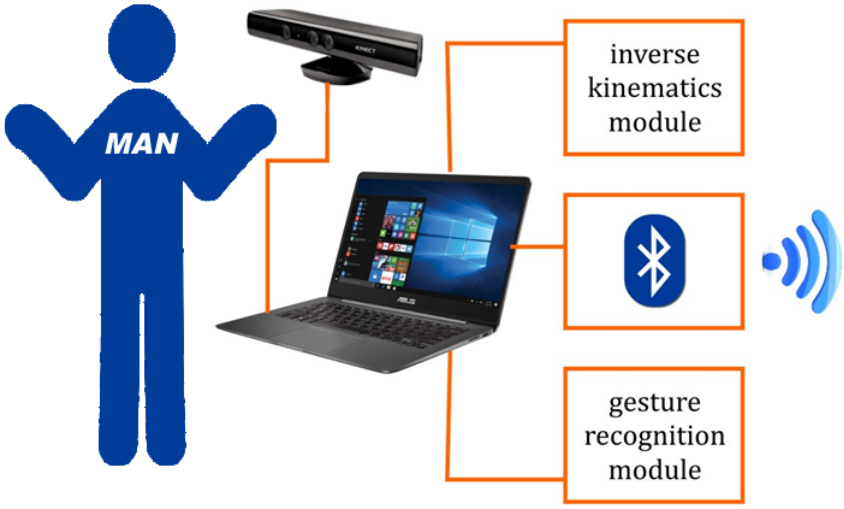


Figure 5: The human part

We have created the system according to Figures 5 and 6, which consists of the following parts:

- Human part:
  - Human user,
  - Kinect,
  - Computer,
  - Inverse kinematics module,
  - Gesture recognition module,
  - Bluetooth communication module.
- Robot part:
  - The robot,
  - Inverse kinematics module,
  - Bluetooth communication module.

The computer-connected Kinect recognizes the user's gestures and calculates the coordinates of the joints using the inverse kinematics module. The data (coordinates, controll sequences) are transmitted via Bluetooth to the LEGO robot. The LEGO robot uses the inverse kinematics module to convert the coordinates into its own coordinate system and then execute the movement.

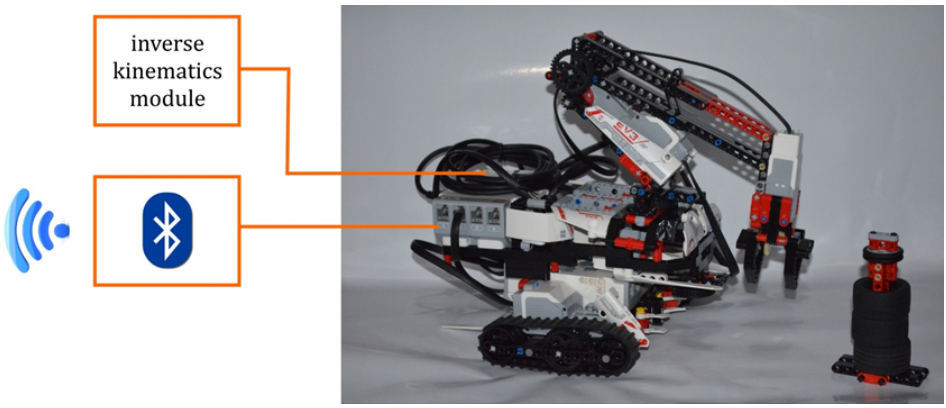


Figure 6: The robot part

7 Conclusion

Experimental results show that the linear approach with the Jacobian solution contains quite a lot of singularity, it must be definitely improved, but it is a fast, effective method. CCD performs better on tracking a moving target avoiding the oscillations and motion discontinuities exhibited by the Jacobian methods [1]. CCD has a low computational cost and solves the IK problem in real-time. The CR algorithm is more difficult to understand but produces suitable results in real time, so it's worth using.

With the presented methods we can easily control robots, of course, we need to practice gestures well enough to achieve the required accuracy.

The methods presented are general enough to control not only LEGO robots, but any robot.

The solution described here is light enough, perfectly suited for educational purposes, and also suitable for lab exercises or simulation exercises.

References

[1] A. Andreas, L. Joan, *Inverse Kinematics: a review of existing techniques and introduction of a new fast iterative solver*, University of Cambridge, Technical Report, 2009. ⇒81, 93

[2] R. G. Boboc, *Natural human-robot interaction for assistive robotics applications*, PhD Thesis, Universitatea Transilvania, Braşov, 2015. ⇒89, 90

[3] T. Brogårdh, Present and future robot control development—An industrial perspective, *Annual Reviews in Control*, **31,** 1 (2017) 69–79. doi:10.1016/j.arcontrol.2007.01.002 ⇒81
[4] S. R. Buss, *Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares methods*, University of California, San Diego, 2009. ⇒86
[5] L. Jamhoury, *Understanding Kinect V2 Joints and Coordinate System*, Medium, 2018. ⇒88
[6] L. Han, L. Rudolph, *Inverse Kinematics for a Serial Chain with Joints under Distance Constraints*, Clark University Worcester, USA. ⇒86
[7] R. Hoover, *Adventures in Motion Capture: Using Kinect Data*, 2016. ⇒89
[8] L. Jamhoury, *Understanding Kinect V2 Joints and Coordinate System*, 2018. ⇒88
[9] R. Juckett, *Constraint Relaxation IK in 2D*, 2009. ⇒86
[10] L. I. Kovács, *Számítógépes grafika*, Ed. Scientia, Kolozsvár, 2009. ⇒81
[11] L. I. Kovács, *Gesztusokkal vezérelt robotkar*, in: SzámOkt'2017, EMT, Kolozsvár, 2017. ⇒81
[12] N. Sfetcu, *The Art of Movies*, Ebook, 2011. ⇒82
[13] L. Szirmay-Kalos, Gy. Antal , F. Csonka, *Háromdimenziós grafika, animáció és játékfejlesztés*, ComputerBooks, Budapest, 2006. ⇒81, 87
[14] S. J. Wright, *Coordinate Descent Algorithms*, University of Wisconsin, 2010. ⇒86
[15] ∗ ∗ ∗ *Kinect homepage*. ⇒81
[16] ∗ ∗ ∗ *LEGO homepage*. ⇒80, 91
[17] ∗ ∗ ∗ *OpenGL homepage*. ⇒81
[18] ∗ ∗ ∗ *Regression estimation is smoothed values in space as a dynamic movement Motion Selection Principles in Action Robo*. ⇒83
[19] ∗ ∗ ∗ *Wikipedia: Key frame*. ⇒82
[20] ∗ ∗ ∗ *Wikipedia: Levenberg–Marquardt algorithm*. ⇒87
[21] ∗ ∗ ∗ *Wikipedia: Kinect*. ⇒88
[22] ∗ ∗ ∗ *Wikipedia: Dynamic time warping*. ⇒90

Received: June 30, 2019 • Revised: August 6, 2019

Acta Univ. Sapientiae, Informatica **11,** 1 (2019) 95–108

DOI: 10.2478/ausi-2019-0007

J-coloring of graph operations

Sudev NADUVATH
Department of Mathematics
CHRIST (Deemed to be University
Bangalore, INDIA
email: sudev.nk@christuniversity.in

Johan KOK
Department of Mathematics
CHRIST (Deemed to be University
Bangalore, INDIA
email: jacotype@gmail.com

Abstract. A vertex v of a given graph is said to be in a rainbow neighbourhood of G if every color class of G consists of at least one vertex from the closed neighbourhood $N[v]$. A maximal proper coloring of a graph G is a J-coloring if and only if every vertex of G belongs to a rainbow neighbourhood of G. In general all graphs need not have a J-coloring, even though they admit a chromatic coloring. In this paper, we characterise graphs which admit a J-coloring. We also discuss some preliminary results in respect of certain graph operations which admit a J-coloring under certain conditions.

1 Introduction

For general notations and concepts in graphs and digraphs we refer to [1, 3, 9]. For further definitions in the theory of graph coloring, see [2, 4]. Unless specified otherwise, all graphs mentioned in this paper are simple, connected and undirected graphs.

The degree of a vertex $v \in V(G)$ is the number of edges in G incident with v and is denoted $d_G(v)$ or when the context is clear, simply as $d(v)$. A *pendant vertex* or an *end vertex* of a graph G is a vertex having degree 1. A vertex

Computing Classification System 1998: G.2.2
Mathematics Subject Classification 2010: 05C15, 05C38, 05C75, 05C85.
Key words and phrases: rainbow neighbourhood, J-coloring, J*-coloring.

which is not a pendant vertex is called an *internal vertex* of G (see [3]). A *pendant edge* of G is an edge incident on a pendant vertex of G. Also, unless mentioned otherwise, the graphs we consider in this paper has the order n and size p with minimum and maximum degree δ and Δ, respectively.

Recall that if $\mathcal{C} = \{c_1, c_2, c_3, \ldots, c_\ell\}$ and ℓ sufficiently large, is a set of distinct colors, a proper *vertex coloring* of a graph G is a vertex coloring $\varphi : V(G) \mapsto \mathcal{C}$ such that no two distinct adjacent vertices have the same color. The cardinality of a minimum set of colors which allows a proper vertex coloring of G is called the *chromatic number* of G and is denoted by $\chi(G)$. When a vertex coloring is considered with colors of minimum subscripts, the coloring is called a *minimum parameter coloring*. Unless stated otherwise, all colorings in this paper are minimum parameter color sets.

The number of times a color c_i is allocated to vertices of a graph G is denoted by $\theta(c_i)$ and $\varphi : v_i \mapsto c_j$ is abbreviated, $c(v_i) = c_j$. Furthermore, if $c(v_i) = c_j$ then $\iota(v_i) = j$. The color class of a color c_i, denoted by $\mathcal{C}_i$, is the set of vertices of G having the same color c_i.

We shall also color a graph in accordance with the *rainbow neighbourhood convention* (see [5]), which is stated as follows.

Rainbow neighbourhood convention: ([5]) For a proper coloring $\mathcal{C} = \{c_1, c_2, c_3, \ldots, c_\ell\}$, $\chi(G) = \ell$, we always color maximum possible number of vertices with the color c_1, then color the maximum possible number of remaining vertices by the color c_2 and proceeding like this and finally color the remaining vertices by the color c_ℓ. Such a coloring is called a χ^--coloring of a graph.

The inverse to the convention requires the mapping $c_j \mapsto c_{\ell-(j-1)}$. Corresponding to the inverse coloring we define $\iota'(v_i) = \ell - (j-1)$ if $c(v_i) = c_j$. The inverse of a χ^--coloring is called a χ^+-coloring.

The closed neighbourhood $N[v]$ of a vertex $v \in V(G)$ which contains at least one colored vertex of each color in the chromatic coloring, is called a *rainbow neighbourhood*. That is, a vertex V is said to be in a rainbow neighbourhood if $\mathcal{C}_i \cap N[v] \neq \emptyset$, for all $1 \leq i \leq \chi(G)$. The number of vertices of a graph G, which belong to some rainbow neighbourhoods of G is called the *rainbow neighbourhood number* of G, denoted by $r_{\chi(G)}$ (see [5]). The rainbow neighbourhood number of certain graph classes have been determined in [6, 7].

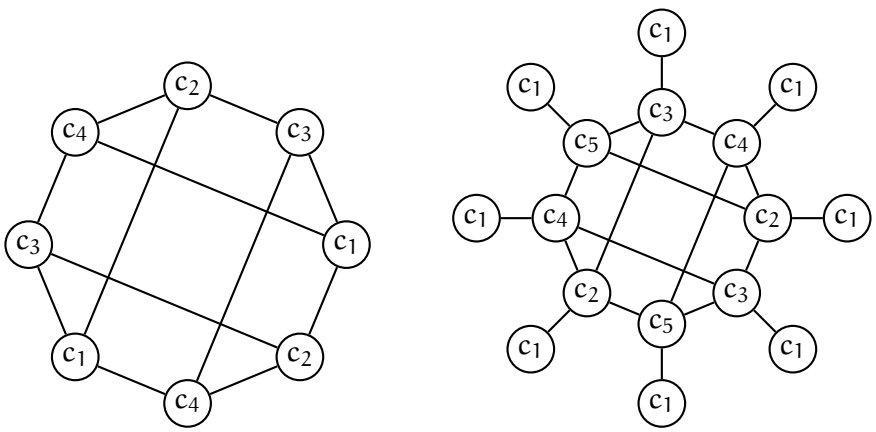
Motivated by these studies, two types of vertex colorings in terms of rainbow neighbourhoods have been introduced in [8] as follows.

Definition 1 [8] A maximal proper coloring of a graph G is a *Johan coloring* of G, or J-*coloring* in short, if and only if every vertex of G belongs to a rainbow neighbourhood of G. The maximum number of colors in a J-coloring

is called the J-*chromatic number* of G, denoted by $\mathcal{J}(G)$.

Definition 2 [8] A maximal proper coloring of a graph G is a *modified Johan coloring*, or J*-*coloring* in short, if and only if every internal vertex (a vertex having degree at least 2) of G belongs to a rainbow neighbourhood of G. The maximum number of colors in a J*-coloring is denoted by $\mathcal{J}^*(G)$.

Figure 1 illustrate a J-colorable and a J*-colorable graph.



(a) A graph with a J-coloring. (b) A graph with a J*-coloring.

Figure 1: J-colorable and J* colorable graphs

In this paper, we characterise the graphs which admit J-coloring. We also discuss some preliminary results in respect of certain graph operations which admit a J-coloring under certain conditions.

2 Results and discussions

A *null graph* on n vertices is an edgeless graph and is denoted by $\mathfrak{N}_n$. We follow the convention that $\mathcal{J}(\mathfrak{N}_n) = \mathcal{J}^*(\mathfrak{N}_n) = 1,\ n \in \mathbb{N}$. Also, note that for any graph G which admits a J-coloring, we have $\chi(G) \leq \mathcal{J}(G)$.

Note that if a graph G admits a J-coloring, it also admits a J*-coloring. However, the converse need not be true always. It can also be noted that if graph G has no pendant vertex and it admits a J-coloring, then $\mathcal{J}(G) = \mathcal{J}^*(G)$.

In view of the above mentioned concepts and facts, we have the following theorem.

Theorem 3 *If* G *is a tree of order* $n \geq 2$*, then* $\mathcal{J}(G) < \mathcal{J}^*(G)$.

Proof. A tree G of order $n \geq 2$ has at least two pendant vertices, say u and v. Therefore, the maximum number of colors which will allow both vertices u and v to yield rainbow neighbourhoods is $\chi(G) = 2$. Therefore, G admits a J-coloring and $\mathcal{J}(G) = 2$.

Any internal vertex w of G has $d(w) \geq 2$. Therefore, $\mathcal{J}^*(G) \leq 3$. Consider any diameter path of G say $P_{diam(G)}$. Beginning at a pendant vertex of the diameter path, label the vertices consecutively $v_1, v_2, v_3, \ldots, v_{diam(G)}$. color the vertices consecutively $c(v_1) = c_1$, $c(v_2) = c_2$, $c(v_3) = c_3$, $c(v_4) = c_1$, $c(v_5) = c_2$, $c(v_6) = c_3$ and so on such that

$$c(v_{diam(G)}) = 1; \quad \text{if } diam(G) \equiv 1 (\text{mod } 3) \tag{1}$$

$$c(v_{diam(G)}) = 2; \quad \text{if } diam(G) \equiv 2 (\text{mod } 3) \tag{2}$$

$$c(v_{diam(G)}) = 3; \quad \text{if } diam(G) \equiv 0 (\text{mod } 3). \tag{3}$$

Clearly, in respect of path $P_{diam(G)}$, it is a proper coloring and all internal vertices yield a rainbow neighbourhood on 3 colors. Consider any maximal path starting from, say $v \in V(P_{diam(G)})$. Hence, v is a pendant vertex to that maximal path. color the vertices consecutively from v as follows:

(a) If $c(v) = c_1$ in $P_{diam(G)}$, color as $c_1, c_2, c_3, c_1, c_2, c_3, \ldots, \underbrace{c_1 \text{ or } c_2 \text{ or } c_3}$

(b) If $c(v) = c_2$ in $P_{diam(G)}$, color as $c_2, c_3, c_1, c_2, c_3, c_1, \cdots, \underbrace{c_2 \text{ or } c_3 \text{ or } c_1}$.

(c) If $c(v) = c_3$ in $P_{diam(G)}$, color as $c_3, c_1, c_2, c_3, c_1, c_2, \cdots, \underbrace{c_3 \text{ or } c_1 \text{ or } c_2}$.

It follows from mathematical induction that all maximal branching can receive such coloring which remains a proper coloring with all internal vertices $v \in V(G)$ having $|c(N[v])| = 3$. Furthermore, all nested branching can be colored in a similar way until all vertices of G are colored. Therefore, $\mathcal{J}^*(G) \geq 3$. Hence, $\mathcal{J}(G) < \mathcal{J}^*(G)$. □

An easy example to illustrate Theorem 3 is the star $K_{1,n}$, $n \geq 2$ for which $\mathcal{J}(K_{1,n}) = 2 < n + 1 = \mathcal{J}^*(K_{1,n})$. This example prompts the next results.

Corollary 4 *For any graph* G *which admits a* J^**-coloring, we have* $\mathcal{J}^*(G) \leq \Delta(G) + 1$.

Corollary 5 *If $\mathcal{J}^*(G) > \mathcal{J}(G)$ for a graph G, then G has at least one pendant vertex.*

Proof. Since all $v \in V(G)$ are internal vertices and any vertex u for which $d(u) = \delta(G)$ must yield a rainbow neighbourhood, it follows that any maximal proper coloring $\mathcal{C}$ are bound to $|\mathcal{C}| = |N[u]| = \delta(G) + 1$. Therefore, if $\mathcal{J}^*(G) > \mathcal{J}(G)$, then G has at least one pendant vertex. □

In [5], the rainbow neighbourhood number $r_\chi(G)$ is defined as the number of vertices of G which yield rainbow neighbourhoods. It is evident that not all graphs admit a J-coloring. Then, we have

Lemma 6 *(i) A maximal proper coloring $\varphi : V(G) \mapsto \mathcal{C}$ of a graph G which satisfies a graph theoretical property, say $\mathfrak{P}$, can be minimised to obtain a minimal proper coloring which satisfies $\mathfrak{P}$.*

(ii) A minimal proper coloring $\varphi : V(G) \mapsto \mathcal{C}$ of a graph G which satisfies a graph theoretical property, say $\mathfrak{P}$, can be maximised to obtain a maximal proper coloring which satisfies $\mathfrak{P}$.

Proof.

(i) Consider a maximal proper coloring $\varphi : V(G) \mapsto \mathcal{C}$ of a graph G which satisfies a graph theoretical property say, $\mathfrak{P}$. If a minimum color set $\mathcal{C}'$, with $|\mathcal{C}'| < |\mathcal{C}|$, such that a minimal proper coloring $\varphi' : V(G) \mapsto \mathcal{C}'$ which satisfies the graph theoretical property $\mathfrak{P}$ cannot be found, then $|\mathcal{C}|$ is minimum.

(ii) Consider a minimal proper coloring $\varphi : V(G) \mapsto \mathcal{C}$ of a graph G which satisfies a graph theoretical property say, $\mathfrak{P}$. If a maximum color set $\mathcal{C}'$, $|\mathcal{C}'| > |\mathcal{C}|$, such that a maximal proper coloring $\varphi' : V(G) \mapsto \mathcal{C}'$ which satisfies the graph theoretical property $\mathfrak{P}$ cannot be found, then $|\mathcal{C}|$ is maximum.

□

The following theorem characterises those graphs which admit a J-coloring.

Theorem 7 *A graph G of order n admits a J-coloring if and only if $r_\chi(G) = n$.*

Proof. If $r_\chi(G) = n$, then every vertex of G belongs to a rainbow neighbourhood. Hence, either the chromatic coloring $\varphi : V(G) \mapsto \mathcal{C}$ is maximal or a maximal coloring $\varphi' : V(G) \mapsto \mathcal{C}'$ exists.

An immediate consequence of Definition 1 is that if graph G admits a J-coloring then each vertex $v \in V(G)$ yields a rainbow neighbourhood. This consequence also follows from the the result that for any connected graph G, $\mathcal{J}(G) \leq \delta(G) + 1$ (see [8]). Hence, from Lemma 6 it follows that either the J-coloring is minimal or a minimal coloring $\varphi' : V(G) \mapsto \mathcal{C}'$ exists such that $r_\chi(G) = n$. □

The following theorem establishes a necessary and sufficient condition for a graph G to have a J-coloring with respect to a χ^--coloring of G.

Theorem 8 *A graph G admits a J-coloring if and only if each $v \in V(G)$ yields a rainbow neighbourhood with respect to a χ^--coloring of G.*

Proof. If in a χ^--coloring of G, each $v \in V(G)$ yields a rainbow neighbourhood it follows from the second part of Lemma 6 that the corresponding proper coloring can be maximised to obtain a J-coloring.

Conversely, assume that a graph G admits a J-coloring. Then, it follows from Lemma 6(i) that the corresponding proper coloring can be minimised to obtain a minimal proper coloring for which each $v \in V(G)$ yields a rainbow neighbourhood. Let the aforesaid set of colors be $\mathcal{C}'$. Assume that a minimum set of colors $\mathcal{C}$ exists which is a χ^--coloring of G and $|\mathcal{C}| < |\mathcal{C}'|$. It implies that there exists at least one vertex $v \in V(G)$ for which at least one distinct pair of vertices, say $u, w \in N(v)$ exists such that u and v are non-adjacent. Furthermore, $c(u) = c(w)$ under the coloring $\varphi : V(G) \mapsto \mathcal{C}$.

Assume that there is exactly one such v and exactly one such vertex pair $u, w \in N(v)$. But then both u and w yield rainbow neighbourhoods in G under the proper coloring $\varphi : V(G) \mapsto \mathcal{C}$, which is a contradiction to the minimality of $\mathcal{C}'$. By mathematical induction, similar contradictions arise for all vertices similar to v. This completes the proof. □

3 Analysis for certain graphs

Note that we have two types of operations related to graphs, that is: operations on a graph G and operations between two graphs G and H. Operations on a graph G result in a well defined derivative of G. Examples are the complement graph G^c, the line graph $L(G)$, the middle graph $M(G)$, the central graph $C(G)$, the jump graph $J(G)$ and the total graph $T(G)$ and so on. Recall that the jump graph $J(G)$ of a graph G of order $n \geq 3$ is the complement graph of the line graph $L(G)$. Also, note that the line graph is the graphical realisation of edge adjacency in G and the jump graph is the graphical realisation of edge

independence in G. Some other graph derivative operations are edge deletion, vertex deletion, edge contraction, thorning a graph by pendant vertex addition and so on.

Examples of operations between graphs G and H are, the corona between G and H denoted, $G \circ H$, the join denoted, $G + H$, the disjoint union denoted, $G \cup H$, the Cartesian product denoted, $G \square H$ and so on.

3.1 Operations between certain graphs

The following result establishes a necessary and sufficient condition for the corona of two graphs G and H to admit a J-coloring.

Theorem 9 *If graphs* G *and* H *admit* J*-colorings, then* $G \circ H$ *admits a* J*-coloring if and only if either* $G = K_1$ *or* $\mathcal{J}(G) = \mathcal{J}(H) + 1$.

Proof. *Part 1:* If $G = K_1$ assume $\mathcal{C} = \{c_1, c_2, c_3, \ldots, c_{\mathcal{J}(H)}\}$ provides a J-coloring of H. color K_1 the color $c_{\mathcal{J}(H)+1}$. Clearly, $\mathcal{C}' = \mathcal{C} \cup \{c_{\mathcal{J}(H)+1}\}$ is a J-coloring of $K_1 \circ H$.

Part 2: If $G \neq K_1$ and $\mathcal{J}(G) = \mathcal{J}(H) + 1$ let $\mathcal{C} = \{c_1, c_2, c_3, \ldots, c_\ell,\ \ell = \mathcal{J}(G)\}$ and $\mathcal{C}' = \{c_1, c_2, c_3, \ldots, c_{\ell-1},\ \ell = \mathcal{J}(G)\}$ provide the J-colorings of G and H, respectively. Assume that $v \in V(G)$ has $c(v) = c_i$ then color all $u \in V(H)$ for the copy of H corona'd to v for which $c(u)_{(\text{in } H)} = c_i$, $1 \leq i \leq \ell$, to be $c_{\ell+1}$. Clearly every vertex $v \in V(G) \cup V(H)$ yields a rainbow neighbourhood and $|\mathcal{C}|$ is maximal.

Conversely, let $G \circ H$ admit a J-coloring. Then, for any vertex $v \in V(G)$ the subgraph $v \circ H$ holds the condition $c(v) \neq c(u),\ \forall u \in V(H)$. Therefore, either $G = K_1$ or $\mathcal{J}(G) = \mathcal{J}(H) + 1$. □

The next corollary requires no proof as it is a direct consequence of Theorem 9.

Corollary 10 *If* $G \circ H$ *admits a* J*-coloring then:* $\mathcal{J}(G \circ H) = \mathcal{J}(G)$.

The following theorem discusses the admissibility of J-coloring by the join of two graphs.

Theorem 11 *The join* $G + H$ *admits a* J*-coloring if and only if both graphs* G *and* H *admit a* J*-coloring.*

Proof. Assume that both G and H admit a J-coloring. Without loss of generality, let $\mathcal{J}(G) \leq \mathcal{J}(H)$. Assume that $\varphi : V(G) \mapsto \mathcal{C}$, $\mathcal{C} = \{c_1, c_2, c_3, \ldots, c_\ell\}$

and $\varphi' : V(H) \mapsto \mathcal{C}'$, $\mathcal{C}' = \{c_1, c_2, c_3, \ldots, c_{\ell'}\}$ is a J-coloring of G and H, respectively. For each $v \in V(G)$, $c(v) = c_i$ recolor $c(v) \mapsto c_{i+\ell'}$. Denote the new color set by $\mathcal{C}_{i+\ell'}$. Clearly, each vertex $v \in V(G)$ is adjacent to at least one of each color in $G + H$ hence, each such vertex yields a rainbow neighbourhood in $G + H$. Similarly, each vertex $u \in V(H)$ is adjacent to at least one of each color in $G + H$ and hence each such vertex yields a rainbow neighbourhood in $G + H$. Furthermore, since both $|\mathcal{C}|$, $|\mathcal{C}'|$ is maximal color sets, the set $|\mathcal{C}_{i+\ell'} \cup \mathcal{C}'|$ is maximal. Therefore, $G + H$ admits a J-coloring.

The converse follows trivially from the fact that the additional edges between G and H as defined for join form an edge cut in $G + H$. □

The following result discusses the existence of a J-coloring for the Cartesian product of two given graphs.

Theorem 12 *If graphs* G *and* H *of order* n *and* m *respectively admit a* J*-coloring, then*

(i) $G \square H$ *admits a* J*-coloring.*

(ii) $\mathcal{J}(G \square H) = \max\{\mathcal{J}(G), \mathcal{J}(H)\}$

Proof.

(i) Without loss of generality assume $\mathcal{J}(H) \geq \mathcal{J}(G)$. Also, assume that $V(G) = \{v_i : 1 \leq i \leq n\}$ and $V(H) = \{u_i : 1 \leq i \leq m\}$. From the definition of $G \square H$ it follows that $V(G \square H) = \{(v_i, u_j) : 1 \leq i \leq n, 1 \leq j \leq m\}$. For $i = 1$, if $u_j \sim u_k$ in H, where $\sim$ denotes the adjacency, then $(v_1, u_j) \sim (v_1, u_k)$ and hence we obtain an isomorphic copy of H. Such a copy admits a J-coloring identical to that of H in respect of the vertex elements $u_1, u_2, u_3, \ldots, u_m$. Now obtain the disjoint union with the copies of H corresponding to $i = 2, 3, 4, \ldots, n$. Apply the definition of $G \square H$ for u_1 and if $v_i \sim v_j$ in G, then $(v_i, u_1) \sim (v_j, u_1)$. An interconnecting copy of G is obtained which result in the first iteration connected graph. Similarly, this copy of G admits a J-coloring identical to that of G in respect of the vertex elements $v_1, v_2, v_3, \ldots, v_n$. Proceeding iteratively to add all copies of G for $i = 2, 3, 4, \ldots, n$ in terms of the definition of $G \square H$, clearly shows that a J-coloring is admitted.

(ii) The second part of the result follows from the similar reasoning used to prove and hence, $\chi(G \square H) = \max\{\chi(G), \chi(H)\}$.

□

3.2 Operations on certain graphs

Recall that for any connected graph G, $\mathcal{J}(G) \leq \delta(G) + 1$ (see [8]) and for $n \geq 2$, $\mathcal{J}(P_n) = 2$ and $\mathcal{J}^*(P_n) = 3$. In view of these results, we have the following results in respect of certain operations on paths and cycles.

Proposition 13 *For a path P_n, $n \geq 2$ with edge set consecutively labeled as $e_1, e_2, e_3, \ldots, e_{n-1}$ and the corresponding line graph vertices consecutively labeled as $u_1, u_2, u_3, \ldots, u_{n-1}$. We have*

(i) $\mathcal{J}(L(P_n)) = 2$ *and* $\mathcal{J}^*(L(P_n)) = 3$.

(ii) $\mathcal{J}(M(P_2)) = 2$ *and* $M(P_n)$ $n \geq 3$ *does not admit a* J*-coloring and* $\mathcal{J}^*(M(P_n)) = 3$.

(iii) $\mathcal{J}(T(P_n)) = \mathcal{J}^*(T(P_n)) = 3$.

(iv) For connectivity, let $n \geq 5$. *Then* $\mathcal{J}(J(P_5)) = 3$ *and* $\mathcal{J}^*(J(P_5)) = 3$ *and for* $n \geq 6$,

$$\mathcal{J}(J(P_n)) = \mathcal{J}^*(J(P_n)) = \begin{cases} \frac{n}{2} & n \text{ is even} \\ \lfloor \frac{n}{2} \rfloor & n \text{ is odd.} \end{cases}$$

(v) $\mathcal{J}(C(P_n)) = \mathcal{J}^*(C(P_n)) = 3$.

Proof.

(i) Since $L(P_n) = P_{n-1}$, the result follows from the result that for any connected graph G, $\mathcal{J}(G) \leq \delta(G) + 1$.

(ii) Since $M(P_2) = P_3$ the result follows from the result that for any connected graph G, $\mathcal{J}(G) \leq \delta(G)+1$. For $n \geq 3$, the middle graph contains a triangle hence, $\mathcal{J}(M(P_n)) \geq \chi(M(P_n)) \geq 3$. Also $M(P_n)$ has two pendant vertices therefore $r_\chi(M(P_n)) \neq n$. So $M(P_n)$, $n \geq 3$ does not admit a J-coloring. The derivative graph $G' = M(P_n) - \{v_1, v_n\}$ contains a triangle and $\delta(G') = 2$. Therefore, $\mathcal{J}^*(M(P_n)) = 3$.

(iii) Since $\mathcal{J}(T(P_n)) \leq \delta(\mathcal{J}(T(P_n)) + 1 = 3$ and $T(P_n)$ contains a triangle, $\mathcal{J}(T(P_n)) = 3$. As $T(P_n)$ has no pendant vertex and contains an odd cycle C_3, the result $\mathcal{J}^*(T(P_n)) = 3$ is immediate.

(iv) For P_5 we have $J(P_5) = P_4$. Hence, the result follows from for any connected graph G, $\mathcal{J}(G) \leq \delta(G) + 1$. For a path P_n, $n \geq 6$ and edge set consecutively labeled as $e_1, e_2, e_3, \ldots, e_{n-1}$ and the corresponding line graph vertices consecutively labeled as $u_1, u_2, u_3, \ldots, u_{n-1}$, we

have the consecutive vertex χ^--coloring sequence of $J(P_n)$ is given by $c_1, c_1, c_2, c_2, c_3, c_3, \ldots, c_{\frac{n}{2}}$ if n is even and $c_1, c_1, c_2, c_2, c_3, c_3, \ldots, c_{\lfloor \frac{n}{2} \rfloor}, c_{\lfloor \frac{n}{2} \rfloor}$ if n is odd. Since the vertices u_i, u_{i+1}, $1 \leq i \leq n-2$ are pairwise not adjacent, the χ^--coloring is maximal as well. Clearly, every vertex u_i yields a rainbow neighbourhood. Therefore, the result follows.

(v) Since $C(P_n)$ has no pendant vertex and contains an odd cycle C_5, the result is immediate.

□

Next, we consider cycles C_n, $n \geq 3$. In [8], it is proved that

Theorem 14 *[8] If C_n admits a J-coloring then:*

$$\mathcal{J}(C_n) = \begin{cases} 3 & \text{if } n \equiv 0 \pmod 3 \\ 2 & \text{if } n \equiv 0 \pmod 2 \text{ and } n \not\equiv 0 \pmod 3. \end{cases}$$

Analogous to the proof of Theorem 2.7 in [8], we now establish the corresponding results for the derivatives of cycle graphs in the following proposition.

Proposition 15 *For a cycle $C_n, n \geq 3$ and edge set consecutively labeled as $e_1, e_2, e_3, \ldots, e_n$ and the corresponding line graph vertices consecutively labeled as $u_1, u_2, u_3, \ldots, u_n$, we have*

(i) $\mathcal{J}(L(C_n)) = \mathcal{J}^*(L(C_n)) = 2$ *if and only if* $n \equiv 0 \pmod 2$ *and* $n \not\equiv 0 \pmod 3$*, and* $\mathcal{J}(L(C_n)) = \mathcal{J}^*(L(C_n)) = 3$ *if and only if* $n \equiv 0 \pmod 3$*, else,* $L(C_n)$ *does not admit a J-coloring.*

(ii) For $n \geq 3$, $\mathcal{J}(M(C_n)) = \mathcal{J}^*(M(C_n)) = 3$ *if* $n \equiv 0 \pmod 3$*, or if,* $M(C_n)$ *for* $n \not\equiv 0 \pmod 3$*, and without loss of generality admits the coloring:* $c(v_1) = c_1$, $c(u_1) = c_2$, $c(v_2) = c_3$, $c(u_2) = c_1$, $c(v_3) = c_2$, $c(u_3) = c_3, \ldots, c(v_{n-1}) = c_1, c(u_{n-1}) = c_2, c(v_n) = c_1, c(u_n) = c_3$*, else,* $M(C_n)$ *does not admit a J-coloring.*

(iii) $\mathcal{J}(T(C_n)) = \mathcal{J}^*(T(C_n)) = 4$ *if and only if* n *is even, else,* $T(C_n)$ *does not admit a J-coloring.*

(iv) For $n \geq 6$, $\mathcal{J}(J(C_n)) = \mathcal{J}^*(J(C_n)) = \begin{cases} \frac{n}{2} & n \text{ is even} \\ \lfloor \frac{n}{2} \rfloor & n \text{ is odd.} \end{cases}$

(v) $\mathcal{J}(C(C_n)) = \mathcal{J}^*(C(C_n)) = 3$.

Proof. (i) Because $L(C_n) = C_n$ the result follows from Corollary 3.6. Also because $L(C_n)$ has no pendant edges, $\mathcal{J}(L(C_n)) = \mathcal{J}^*(L(C_n))$.

(ii) If $M(C_n)$ admits a J-coloring then $\mathcal{J}(M(C_n)) \leq \delta(\mathcal{J}(M(C_n)) + 1 = 3$. For $n \equiv 0 \pmod 3$, consider the coloring: $c(v_1) = c_1$, $c(u_1) = c_2$, $c(v_2) = c_3$, $c(u_2) = c_1$, $c(v_3) = c_2$, $c(u_3) = c_3, \ldots, c(u_{n-1}) = c_1, c(v_n) = c_2, c(u_n) = c_3$.

From the definition of the middle graph, we know that $M(C_n)$ has n triangles stringed so clearly the proper coloring is maximum and all vertices yield a rainbow neighbourhood. Part 2 follows by similar reasoning and hence the result follows. Also, since $M(C_n)$ has no pendant edges, $\mathcal{J}(M(C_n)) = \mathcal{J}^*(M(C_n))$. In all other cases, $\chi((M(C_n)) = 4$ and a J-coloring does not exist.

(iii) Note that $\mathcal{J}(T(C_n)) \leq \delta(\mathcal{J}(T(C_n)) + 1 = 5$. Since $T(C_n)$ contains a triangle, $\mathcal{J}(T(C_n)) \geq 3$. Furthermore, $\chi((T(C_n)) = 4$ if and only if $n \equiv 0 \pmod 2$ and $n \not\equiv 0 \pmod 3$, and all vertices yield a rainbow neighbourhood. Also, for any set of vertices $V' = \{v_i, v_{i+1}, v_{i+2}, v_{i+2}, v_{i+3}, v_{i+4}\} \mapsto \{v_i v_j : 1 \leq i \leq n,\ 0 \leq j \leq 4,\ \text{and } (i+j) \mapsto (i+j) \pmod 6\}$, the induced subgraph $\langle V' \rangle \neq K_5$. Therefore, $\mathcal{J}(T(C_n)) = 4$. Also because $T(C_n)$ has no pendant edges, $\mathcal{J}(T(C_n)) = \mathcal{J}^*(T(C_n))$. Otherwise, $\chi((T(C_n)) = 5$, and not all vertices yield a rainbow neighbourhood and hence a J-coloring is not obtained.

(iv) For $n = 5$, $J(C_5) = C_5$ and thus, does not admit a J-coloring. For a path C_n, $n \geq 6$ and edge set consecutively labeled as $e_1, e_2, e_3, \ldots, e_{n-1}$ and the corresponding line graph vertices consecutively labeled as $u_1, u_2, u_3, \ldots, u_{n-1}$, we have the consecutive vertex χ^--coloring sequence of $J(C_n)$ is given by $c_1, c_1, c_2, c_2, c_3, c_3, \ldots, c_{\frac{n}{2}}$ if n is even and $c_1, c_1, c_2, c_2, c_3, c_3, \ldots, c_{\lfloor \frac{n}{2} \rfloor}, c_{\lfloor \frac{n}{2} \rfloor}$ if n is odd ($n-1$ entries). As the vertices u_i, u_{i+1}, $1 \leq i \leq n-2$ are pairwise not adjacent, the χ^--coloring is maximal as well. Clearly, every vertex u_i yields a rainbow neighbourhood. Therefore, the result follows.

(v) The result is trivial for $C(C_3)$. For $n \geq 4$, $\mathcal{J}(C(C_n)) \leq \delta(\mathcal{J}(C(C_n))+1 = 3$. Since $\chi((C(C_n)) = 3$ and all vertices yield a rainbow neighbourhood and $C(C_n)$ contains a cycle C_5, the result $\mathcal{J}(C(C_n)) = 3$ holds immediately. Also, since $C(C_n)$ has no pendant edges, $\mathcal{J}(C(C_n)) = \mathcal{J}^*(C(C_n))$. □

4 Extremal results for certain graphs

For a graph G of order $n \geq 1$, which admits a J-coloring the minimum (or maximum) number of edges in a subset $E'_k \subseteq E(G)$ whose removal ensures that $\mathcal{J}(G - E'_k) = k,\ 1 \leq k \leq \mathcal{J}(G)$, is discussed in this section. These extremal variables are called the minimum (or maximum) rainbow bonding variables and are denoted $r_k^-(G)$ and $r_k^+(G)$, respectively. A graph G which does not

admit a J-coloring has $r_k^-(G)$ and $r_k^+(G)$ undefined. For such aforesaid graph it is always possible to remove a minimal set of edges, E'', which is not necessarily unique such that $G - E''$ admits a J-coloring. This is formalised in the next result.

Lemma 16 *For any connected graph G which does not admit a J-coloring, a minimal set of edges, E'' which is not necessarily unique, can be removed such that $G - E''$ admits a J-coloring.*

Proof. Since any connected graph G of order n and size $\varepsilon(G) = p$ has a spanning subtree and any tree admits a J-coloring, at most $p - (n-1)$ edges must be removed from G. Therefore, if $p - (n-1)$ is not a minimal number of edges to be removed then a minimal set of edges E', $|E'| < p - (n-1)$ must exist whose removal results in a spanning subgraph G' which allows a J-coloring. □

It is obvious from Lemma 16 that the restriction of connectedness can be relaxed if $G = \bigcup H_i$, $1 \leq i \leq t$ and it is possible that $\mathcal{J}(H_i - E''_i)_{\forall i} = k$, k some integer constant.

It is obvious that for a complete graph K_n, $\mathcal{J}(K_n) = n$. To ensure $\mathcal{J}(K_n) = n$, no edges can be removed. Therefore, $r_n^-(K_n) = r_n^+(K_n) = 0$.

Theorem 17 *For a complete graph K_n, $n \geq 1$ we have*

(i) For n is even and $\frac{n}{2} \leq k \leq n$ and $\mathcal{J}(K_n - E'_k) = k$, then $r_k^-(K_n) = n - k$.

(ii) For n is odd and $\lceil \frac{n}{2} \rceil \leq k \leq n$, and $\mathcal{J}(K_n - E'_k) = k$, then $r_k^-(K_n) = n - k$.

(iii) For $n \in \mathbb{N}$ and $1 \leq k \leq n$, and $\mathcal{J}(K_n - E'_k) = k$, then $r_k^+(K_n) = \frac{1}{2}(n + 1 - k)(n - k)$.

Proof. (i) For n is even and $\frac{n}{2} \leq k \leq n$, exactly 0 or 1 or 2 or 3 or $\cdots$ or $\frac{n}{2}$ edges between distinct pairs of vertices can be removed to obtain $\mathcal{J}(K_n - E'_k) = n, n-1, n-2, \ldots, \frac{n}{2}$. Hence, $r_k^-(K_n) = 0, 1, 2, 3, \ldots, \frac{n}{2}$. In other words $r_k^-(K_n) = n - k$, $\frac{n}{2} \leq k \leq n$.

(ii) The result follows through similar reasoning as that in (i).

(iii) In any clique of order t, the removal of the $\frac{1}{2}t(t-1)$ edges is the maximum number of edges whose removal renders $\mathcal{J}(\mathfrak{N}_t) = 1$ hence, all vertices can be colored say, c_1. Through immediate mathematical induction it follows that we iteratively remove the maximum number of edges $r_k^+(K_n) = 0, 1, 3, 6, 10, \ldots, \frac{1}{2}(n+1-k)(n-k)$, $1 \leq k \leq n$ of cliques $K_1, K_2, K_3, \ldots, K_n$ to obtain $\mathcal{J}(K_n - E'_k) = n, n-1, n-2, \ldots, 1$. Hence, the result follows. □

Theorem 18 *A graph* G *of order* n *which allows a* J*-coloring, has* $r_k^-(G) = r_k^+(G)$ *if and only if* $\mathcal{J}(G) = 2$.

Proof. If $\mathcal{J}(G) = 2$ then all edges are incident with colors c_1, c_2. Therefore all edges must be removed to obtain the null graph $\mathfrak{N}_0$ for which $\mathcal{J}(\mathfrak{N}_0) = 1$. Hence, $r_k^-(G) = r_k^+(G)$.

Conversely, let $r_k^-(G) = r_k^+(G)$. Then, assume that at least one edge say, e is incident with color c_3. It implies that G contains at least a triangle or an odd cycle. Therefore, $\varepsilon(G) \geq 3$. To ensure a proper coloring on removing edge e the color c_3 must change to either c_1 or c_2 which is always possible. If $\mathcal{J}(G-e) = 2$ then $r_k^+(G) = 1$ which is a contradiction because any one additional edge may have been removed, implying $r_k^+(G) \geq 2$. For colors $c_4, c_5, c_6, \ldots, \mathcal{J}(G)$, similar contradictions follows through immediate induction. Therefore, if $r_k^-(G) = r_k^+(G)$ then, $\mathcal{J}(G) = 2$. □

5 Conclusion

Clearly the cycles for which the the middle graphs admit a J-coloring in accordance with the second part of Proposition 13(ii) require to be characterised if possible. It follows from Theorem 18 that for the cases n is even and $1 \leq k < \frac{n}{2}$, or n is odd and $1 \leq k < \lceil \frac{n}{2} \rceil$, determining $r_k^-(K_n)$ remains open. It is suggested that an algorithm must be described to obtain these values.

Example 19 For the complete graph K_9 with vertices $v_1, v_2, v_3, \ldots, v_9$, Theorem 17(ii) admits the minimum removal of $r_{n,k}^-(K_n) = 4$ edges to obtain $\mathcal{J}(K_n - E_k') = 5$. Without loss of generality say the edges were. v_1v_2, v_3v_4, v_6v_6, v_7v_8. To obtain $\mathcal{J}(K_n - E_k') = 4$ we only remove without loss of generality say, the edges v_7v_9, v_8v_9. To obtain $\mathcal{J}(K_n - E_k') = 3$ we only remove without loss of generality say, the edges v_1v_3, v_1v_4, v_2v_3, v_2v_4. To obtain $\mathcal{J}(K_n - E_k') = 2$ we only remove without loss of generality say, the edges v_5v_7, v_5v_8, v_5v_9, v_6v_7, v_6v_8, v_6v_9. To obtain $\mathcal{J}(K_n - E_k') = 1$ we remove all remaining edges. It implies that as $\mathcal{J}(K_n - E_k')$ iteratively ranges through the values $5, 4, 3, 2, 1$ the value of $r_k^-(K_9)$ ranges through, $4, 6, 10, 16, 36$.

Determining the range of minimum (maximum) rainbow bonding variables for other classes of graphs is certainly worthy research. For a graph G which does not allow a J-coloring it follows from Lemma 16 that a study of $r_k^-(G')$ and $r_k^+(G')$ with G' a maximal spanning subgraph of G which does allow a J-coloring, is open.

References

[1] J. A. Bondy and U. S. R. Murty, *Graph Theory*, Springer, New York, 1976. ⇒95

[2] G. Chartrand and P. Zhang, *Chromatic Graph Theory*, CRC Press, 2009. ⇒95

[3] F. Harary, *Graph Theory*, New Age International, New Delhi, 2001. ⇒95, 96

[4] T. R. Jensen and B. Toft, *Graph Coloring Problems*, John Wiley & Sons, 1995. ⇒95

[5] J. Kok, N. K. Sudev and M. K. Jamil, Rainbow neighbourhood number of graphs, *Proy. J. Math.*, **38,** 3 (2019) 471–487. ⇒96, 99

[6] S. Naduvath, S. Chandoor, S.J. Kalayathankal, J Kok, A note on the rainbow neighbourhood number of graphs, *Nat. Acad. Sci. Letters*, **42,** 2 (2019) 135–138. ⇒96

[7] S. Naduvath, S. Chandoor, S. J. Kalayathankal, J Kok, Some new results on the rainbow neighbourhood number of graphs, *Nat. Acad. Sci. Letters*, **42,** 3 (2019) 249–252. ⇒96

[8] N. K. Sudev, On certain J-coloring parameters of graphs, *Nat. Acad. Sci. Letters*, (2019), in press. ⇒96, 97, 100, 103, 104

[9] D. B. West, *Introduction to Graph Theory*, Pearson Education Inc., Delhi, 2001. ⇒95

Received: July 4, 2019 • Revised: August 6, 2019

Acta Universitatis Sapientiae

The scientific journal of Sapientia Hungarian University of Transylvania publishes original papers and surveys in several areas of sciences written in English.
Information about each series can be found at
http://www.acta.sapientia.ro.

Acta Universitatis Sapientiae, Informatica

Each volume contains two issues.


Sapientia University


Sciendo by De Gruyter


Scientia Publishing House

ISSN 1844-6086
http://www.acta.sapientia.ro

Information for authors

Acta Universitatis Sapientiae, Informatica publishes original papers and surveys in various fields of Computer Science. All papers are peer-reviewed.

Papers published in current and previous volumes can be found in Portable Document Format (pdf) form at the address: http://www.acta.sapientia.ro.

The submitted papers should not be considered for publication by other journals. The corresponding author is responsible for obtaining the permission of coauthors and of the authorities of institutes, if needed, for publication, the Editorial Board is disclaiming any responsibility.

Submission must be made by email (acta-inf@acta.sapientia.ro) only, using the LaTeX style and sample file at the address http://www.acta.sapientia.ro. Beside the LaTeX source a pdf format of the paper is necessary too.

Prepare your paper carefully, including keywords, ACM Computing Classification System codes (http://www.acm.org/about/class/1998) and AMS Mathematics Subject Classification codes (http://www.ams.org/msc/).

References should be listed alphabetically based on the Intructions for Authors given at the address http://www.acta.sapientia.ro.

Illustrations should be given in Encapsulated Postscript (eps) format.

One issue is offered each author free of charge. No reprints will be available.

Contact address and subscription:
Acta Universitatis Sapientiae, Informatica
RO 400112 Cluj-Napoca
Str. Matei Corvin nr. 4.
Email: acta-inf@acta.sapientia.ro

Printed by Idea Printing House
Director: Péter Nagy

ISSN 1844-6086
http://www.acta.sapientia.ro