Journal of Computer Science

VOLUME 10 – N.1 – 2011


Indexed in: INSPEC Qualis-CAPES

Solicita-se permuta/Exchange desired Tiragem/Quantity issued per print: 310

Ministério da Educação
Ministro: Fernando Haddad
Reitor: Aníbal Nazareno Guimarães Mendes
Vice-Reitor: Elias Tadeu Fialho
Pré-Reitora de Pesquisa: Edila Viléia de Resende Von Pinho

Editora UFLA
Diretoria Executiva: Renato Paiva (Diretor)
Elias Tadeu Fialho
Carlos Alberto da Silva

Associate Editors
Abdelmalek Amine, Univ. Djillali Liabes-Sidi, Algeria
Alessandra Alainz Macedo, USP, Brazil
Antônio Nazareno Guimarães Mendes

Executive Editor
Heitor Augustus Xavier Costa, UFLA, Brazil

Editorial Board
Editor-in-Chief
Luiz Henrique Andrade Correia, UFLA, Brazil

Scientific Editors
Hélcio Hideki Yamasaki, INPE, Brazil
João M. da S. Tavares, FEUP, Portugal
Mohu Ramachandran, Leeds Metr. Univ., UK
Paolo Paleari, UNIP, Brazil

Associate Editors
Abdelmalek Amine, Univ. Djillali Liabes-Sidi, Algeria
Alessandra Alainz Macedo, USP, Brazil

Technical Staff
Ariana da Silva Laureado (Secretary)
Túlio Vono Siqueira (Secretary)

©2011 Editora UFLA
ISSN 1807-4545
Ciência da Computação. 1. Ciência da Computação I.Universidade Federal de Lavras. II. Departamento de Ciência da Computação.
Algorithms over Dynamic Graphs

RAFAEL A. GARCÍA GÓMEZ

Politécnico Grancolombiano
Departamento Académico de Ingeniería de Sistemas y Telecomunicaciones
Calle 57 No. 3-00 Este - Bogotá - Colombia
rgarcia@poligran.edu.co

Abstract. Graph theory provides mathematical models with computational realizations for a wide range of problems. The classic version provides static models and solutions for these problems. These solutions are often insufficient to versions of the problems in which the information changes with respect to a continuous variable, e.g. time. In the same way one can think on dynamic graphs as graphs in which some components (arcs, edges, costs) change with respect to a continuous variable. This paper explores graphs with dynamic costs as a bundle, and includes the formulation and the solution for the shortest path problem and the maximum flow problem on these structures. This paper provides an unexplored connection between the dynamic graph theory and the topology, presents approaches to the solution of dynamic versions for the shortest path problem and the maximum flow problem, and proposes both a new source of applications of the metric bundles theory and the type two theory of effectivity.

Keywords: Bundles of metric spaces, graphs with dynamic costs, dynamic graphs.

(Received December 30, 2010 / Accepted May 20, 2011)

1 Introduction

Graph theory provides mathematical models with computational realizations for a wide range of problems. The classic version provides static models and solutions for central problems in graph theory: shortest path problem (SPP), minimum spanning tree (MST), maximum flow problem (MFP) and many others. The static solutions are, for example, the Dijkstra algorithm to find the shortest path between two nodes in a connected graph with nonnegative costs [4], the Ford-Fulkerson algorithm to find the maximum flow between two nodes in a graph with limited capabilities in edges [6], the Kruskal’s algorithm [10] to find the minimal spanning tree in a connected graph, etc.

Obviously, one of the complications offered by the reality to these problems is related with the fact that the real information is constantly changing. For example, the length of the roads in a city is not enough information to decide the route that will consume less time to go from the home to the workplace in the morning, and the most efficient path in a particular time is often not in another time.

In general, the graphs in that part of the information changes with respect to a variable are called dynamic graphs. This paper is limited to dynamic graphs where the costs in the edges change with respect to a continuous variable, for example, time: graphs with dynamic costs or dynamically costed graphs.

The paper is structured as follows: section 2 is devoted to introduce the basic concepts that will be used in the paper, in section 3 we introduces the concept of graphs with dynamic costs and built the metric bundle associated with a graph of this nature, in section 4 we presents the shortest path problem in a graph with dynamic costs and solutions are proposed, section 5 presents the related work on the maximum flow problem on dynamic graphs. To conclude, section 6 presents conclusions and future work. Additionally, appendix 7 presents the basic concepts of bundles of metric spaces or metric bundles.
2 Basic Concepts

The first part of this section is devoted to presents the basic concepts of graph theory. [3] is one of the many references for further in graph theory. The second part presents the basic concepts of topology. The recommended reference is [17].

2.1 Graph Theory

Definition 1 (Graph) A graph is a pair \( G = (V_G, E_G) \) such that \( V_G \) is a set and \( E_G \subseteq V_G \times V_G \). If \( v \in V_G \) then \( v \) is a vertex (node) in \( G \) and, if \( e \in E_G \) then \( e \) is an edge (arc) in \( G \). If \( c_G : E_G \to \mathbb{R} \) is a function, then \( G = (V_G, E_G, c_G) \) is a costed graph and \( c \) is the costs function of \( G \).

Throughout this paper we will assume that \( G \) is a finite graph unless otherwise stated, i.e. \( V \) is a finite set.

Definition 2 (Graph isomorphism) \( \psi \) is graph isomorphism between \( G = (V_G, E_G) \) and \( H = (V_H, E_H) \), if \( \psi \) is a function such that

1. \( \psi : V_G \to V_H \) is a bijection, and
2. for all \( (v_1 ,v_2) \in V_G \times V_G \), \( (v_1, v_2) \in E_G \) if and only if \( \psi(v_1), \psi(v_2) \in E_H \).

In this case, we say that \( G \) and \( H \) are isomorphic graphs and write \( G \sim H \).

Note that an isomorphism \( \psi \) induces a bijection \( \psi_E : E_G \to E_H \) defined by \( \psi_E ((u,v)) = (\psi(u), \psi(v)) \).

Definition 3 (Path between two nodes) Let \( G \) be a graph. If \( v_1 \) and \( v_2 \) are nodes in \( V_G \) and \( \beta = (w_1 \ldots w_n) \) is a list of nodes in \( V_G \), we say that \( \beta \) is a path between \( v_1 \) and \( v_2 \) if and only if

1. \( w_1 = v_1 \).
2. \( w_n = v_2 \), and
3. \( (w_i ,w_{i+1}) \in E_G \) for all \( i = 1, \ldots, n-1 \).

If all nodes in \( \beta \) are different, then we say that \( \beta \) is a simple path between \( v_1 \) and \( v_2 \).

If \( G \) is a costed graph, then \( c_G \) can be extended to all paths in \( G \): If \( \beta = (w_1 \ldots w_n) \) is a path between \( v_1 \) and \( v_2 \), then

\[
c_G(\beta) = \sum_{i=1}^{n-1} c_G((w_i ,w_{i+1}))
\] (1)

Definition 4 (Adjacency set) Let \( G = (V_G, E_G) \) be a graph, and \( v \in V_G \) be a vertex in \( G \). The adjacency set of \( v \) is the set of all vertex in \( G \) that are connected directly with \( v \), that is

\[
\text{adj}(v) = \{ w | (v, w) \in E_G \}.
\] (2)

2.2 Topology

Definition 5 (Topological space) The tuple \( X = (X, \tau) \) is a topological space if and only if \( X \) is a set and \( \tau \) is a family of subsets of \( X \) such that

- \( X \in \tau \),
- \( \emptyset \in \tau \),
- \( X \) is closed under arbitrary unions, in other words, if \( \beta \subseteq \tau \) then \( \bigcup_{B \in \beta} B \in \tau \), and
- \( X \) is closed under finite intersections, in other words, if \( B_1, \ldots, B_n \in \tau \) then \( \bigcap_{k=1}^n B_k \in \tau \).

In this case we say that \( \tau \) is a topology over \( X \), and if \( U \in \tau \) we say that \( U \) is an open set in \( X \).

Definition 6 (Base for a topology) A base for \( \tau \) is a class \( B \) of open sets with the property that every open set is a union of sets in \( B \).

Definition 7 (Continuous functions) Let \( (X, \tau_X) \) and \( (Y, \tau_Y) \) be two topological spaces, and \( f : X \to Y \) be a function, we say that \( f \) is a continuous function if and only if, \( f^{-1}(V) \in \tau_X \) whenever \( V \in \tau_Y \).

Definition 8 (Metric space) \( X = (X,d) \) is a metric space if \( X \) is a set and \( d : X \times X \to \mathbb{R}^+ \) is a function such that, for all \( x, y, z \in X \),

- \( d(x,y) = 0 \iff x = y \),
- \( d(x,y) = d(y,x) \), and
- \( d(x,y) + d(y,z) \leq d(x,z) \).

Every metric space can be viewed as a topological space with the topology generated by the base

\[
\{ B_r(x) | x \in X \wedge r \in \mathbb{R}^+ \}
\]

where

\[
B_r(x) = \{ y | d(x,y) < r \}.
\]
3 Dynamically costed graphs

In general, a dynamic graph is a graph in which part of the information changes with respect to a variable. This paper is limited to dynamic graphs in which the arc costs change with respect to a variable. Formally, we assume that \( G = (V_G, E_G) \) is a graph, \( T \) is a topological space, and, for all \( t \in T \), \( c_t : E_G \to \mathbb{R}^+ \) is a cost function over the arcs of \( G \). In other words, for all \( t \in T \), \( G = (V_G, E_G, c_t) \) is a costed graph.

Definition 9 (Dynamically costed graphs) Let \( G \) be a graph, and let \( T \) be a topological space. Let \( C \) be a family, indexed by \( T \), of cost functions over the arcs of \( G \). In other words, for all \( t \in T \), there exists a unique function \( c_t : E_G \to \mathbb{R}^+ \) in \( C \) such that \( G = (V_G, E_G, c_t) \) is a costed graph. If, for all edge \( e \in E_G \), the function \( k_e : T \to \mathbb{R}^+ \) defined by \( k_e(t) = c_e(e) \) is continuous, then we say that \( \mathcal{G} = (G, C, T) \) is a graph with dynamic costs or dynamically costed graph.

In general, unless otherwise stated, we assume that all elements of \( C \) take nonnegative values.

If \( \mathcal{G} = (G, C, T) \) is a dynamically costed graph, and \( \beta = (w_k)_{k=0}^{q} \) is a path, then we say that the cost of \( \beta \) in \( \mathcal{G} \) is defined by

\[
c_\mathcal{G}(\beta) = \sum_{k=0}^{q-1} c_{t_k}(\langle w_k, w_{k+1} \rangle)
\]

where \( t_0 = 0 \), and \( t_k = t_{k-1} + c_{t_{k-1}}(\langle w_{k-1}, w_k \rangle) \) for \( k = 1, \ldots, q-1 \).

3.1 Graphs with dynamic costs as bundles of metric spaces

Check for the appendix 7 for concepts on bundles of metric spaces.

Henceforth it is understood that \( \mathcal{G} = (G, C, T) \) is a graph with dynamic costs such that \( G = (V_G, E_G) \) is connected.

Note that in these conditions, for all \( t \in T \), it is possible to define the function \( d_t : V \times V \to \mathbb{R}^+ \) such that \( d_t(v_1, v_2) \) is the length of the shortest path between the nodes \( v_1 \) and \( v_2 \) in \( G \) with respect to the cost function \( c_t \). \( d_t \) can be calculated, for example, by Dijkstra’s algorithm [4] or the Floyd-Warshall algorithm.

Proposition 1 For all \( t \in T \), \( \mathcal{G} = (G, d_t) \) is a metric space.

The following theorem is an application of the Existence Theorem for bundles of metric spaces presented in [7].

Theorem 1 Let \( E = V_G \times T \), \( \pi_2 : E \to T \) the second projection \( \pi_2(v, t) = t \), \( \Sigma = \{ \alpha_v : v \in V_G \} \) with \( \alpha_v : T \to \mathbb{R} \) defined by \( \alpha_v(t) = \langle v, t \rangle \) for all \( t \in T \), and the function \( d : E \times E \to [0, +\infty] \) defined by \( d(\langle v_1, t_1 \rangle, \langle v_2, t_2 \rangle) = +\infty \) if \( t_1 \neq t_2 \) and \( d(\langle v_1, t_1 \rangle, \langle v_2, t_2 \rangle) = d_t(v_1, v_2) \) if \( t_1 = t_2 = t \). Then \( \langle E, \pi_2, T \rangle \) is a bundle of metric spaces, \( \Sigma \) is a full set of global sections for \( \pi_2 \) and the family of all \( \epsilon \)-tubes around \( \alpha_v \) \( \{U \in \Sigma \} \), where \( \epsilon > 0 \), \( \alpha_v \) runs \( \Sigma \) and \( U \) through the collection of nonempty open sets in \( T \), is a base for the topology of \( E \).

Proof. It is clear that \( d \) is a metric for \( \pi_2 \), and, since for every \( v \in V \), \( \pi_2(\alpha_v(t)) = \pi_2(v, t) = t \), the collection \( \Sigma \) constitutes a family of selections for \( \pi_2 \). Furthermore, if \( (v, t) \in E \), then \( (v, t) = \alpha_v(t) \), and thus \( (v, t) \in \pi_2^{-1}(\xi) \) for every \( \epsilon > 0 \).

Let \( v_1, v_2 \in V \) and define \( \Phi_{v_1, v_2} : T \to [0, +\infty] \) by

\[
\Phi_{v_1, v_2}(t) = d(\alpha_{v_1}(t), \alpha_{v_2}(t)).
\]

Since \( \pi_2 (\alpha_{v_1}(t)) = \pi_2 (v_1, t) = t \), we must have

\[
\Phi_{v_1, v_2}(t) = d(\alpha_{v_1}(t), \alpha_{v_2}(t)) = +\infty
\]

and

\[
\Phi_{v_1, v_2}(t) = d(\alpha_{v_2}(t), \alpha_{v_2}(t)) = d_t(v_1, v_2).
\]

For every simple path, \( \beta = \langle w_k \rangle_{k=0}^{q} \), from \( v_1 \) to \( v_2 \) and for every \( t \in T \), let

\[
c_t(\beta) = \sum_{i=0}^{q-1} c_t(\langle w_i, w_{i+1} \rangle)
\]

be the cost of path \( \beta \) over the graph \( G_t = (V_G, E_G, c_t) \), that is, the cost of path \( \beta \) with regard to the cost function \( c_t \).

Under these conditions, given that \( k_{(w_i, w_{i+1})} \) is a continuous function, that is to say,

\[
\lim_{u \to t} c_u(\langle w_i, w_{i+1} \rangle) = c_t(\langle w_i, w_{i+1} \rangle),
\]

we must have

\[
\lim_{u \to t} c_u(\beta) = \lim_{u \to t} \sum_{i=0}^{q-1} c_t(\langle w_i, w_{i+1} \rangle) = c_t(\beta).
\]

So that if \( \beta_1, \ldots, \beta_m \) are paths starting at \( v_1 \) and ending at \( v_2 \), the function

\[
F_{\beta_1, \ldots, \beta_m}(t) = \min_{i=1, \ldots, m} \{ c_t(\beta) \}
\]

...
is continuous on \( t \).

To complete the hypothesis of the theorem for Existence of Bundles of Metric Spaces [7], it is enough to notice that if \( P(v_1, v_2) \) is the set of paths connecting \( v_1 \) and \( v_2 \), \( P(v_1, v_2) \) is finite. Thus,

\[
\Phi_{v_1, v_2}(t) = \min_{\beta \in P(v_1, v_2)} \{ c_\beta(\beta) \}
\]

is continuous. This allows us to conclude that \( \mathcal{G} = (V_G \times T, \pi_2, T) \) is a bundle of metric spaces, \( \Sigma \) is a full set of global sections over for \( \pi_2 \) and that the collection of \( \epsilon \)-tubes around \( \alpha_v \mid_U \), for \( \epsilon > 0 \), \( \alpha_v \in \Sigma \) and \( U \) a non-empty open set of \( T \), constitute a basis for the topology of \( E \). \( \square \)

**Definition 10** The bundle of metric spaces \( \mathcal{G} \) constructed previously is known as the bundle of metric spaces associated with the dynamically costed graph \( \mathcal{G} \), or the Bundle of Metric Spaces of \( \mathcal{G} \).

**Proposition 2** Let \( t_1, t_2 \in T \) and denote by \( \pi_2^{-1}(t_1) \) and \( \pi_2^{-1}(t_2) \) their corresponding fibers in \( \mathcal{G} \). In this case

\[
G_{t_1} = \langle \pi_2^{-1}(t_1), E_G, c_{t_1} \rangle
\]

and

\[
G_{t_2} = \langle \pi_2^{-1}(t_2), E_G, c_{t_2} \rangle
\]

and in this case, both \( G_{t_1} \) and \( G_{t_2} \) are isomorphic graphs.

**4 Shortest path in dynamically costed graphs**

A problem that cannot be obviously modelled using statically-costed graphs is the problem of traffic distribution. Indeed, finding the shortest path between two locations in a city along a -directed- network of roads, must take into account phenomena such as rush-hours, traffic jams, and works on the roads.

This problem can be modelled in a particular situation by introducing a costed graph \( G \), onto which to find the shortest path. Conceptually, this is done making \( G = (V, E) \) to have a function \( c : E \rightarrow \mathbb{R}^+ \) assigning the cost of traversing edge (i.e. street) \( e \). The problem of finding the shortest route between vertices \( v_1 \) and \( v_2 \) can be described as the problem of finding a path \( \beta = \langle w_k \rangle_{k=0}^q \), from \( v_1 \) to \( v_2 \), in such a way that \( c(\beta) \) is minimal among all paths connecting node \( v_1 \) and node \( v_2 \).

The static version can be formulated as follows:

- **Instance:** A connected graph \( G = \langle V_G, E_G \rangle \), a cost function \( c_G : E_G \rightarrow \mathbb{R}^+ \), and two nodes \( v_1, v_2 \in V_G \).

- **Answer:** A path \( \beta = \langle w_k \rangle_{k=0}^q \) from \( v_1 \) to \( v_2 \) in \( G \), such that \( c_G(\beta) \leq c_G(\gamma) \), for all path \( \gamma = \langle w_k \rangle_{k=0}^q \), from \( v_1 \) to \( v_2 \).

An algorithmic solution to the shortest-path problem in static graphs is known and can be calculated using a variety of techniques for example Dijkstra’s algorithm [4]. However, in order to take into account the problem of time-varying costs of traversing a path (in the sense of time spent), dynamically costed graphs must be introduced. In this instance, the cost function \( c \) of graph \( G \) varies along time: \( c : E \times \mathbb{R}^+ \rightarrow \mathbb{R} \).

This situation can be explored in a more general setting, allowing the description and exploration of shortest-path problems in -for example- traffic.

Indeed, shortest-path in dynamically costed graphs is a problem that can be tackled using bundles of metric spaces (see appendix 7), by regarding the graph itself as a bundle of metric spaces:

\[
\mathcal{G} = (V_G \times \mathbb{R}^+, \pi_2, \mathbb{R}^+) \,
\]

where graph \( G \) can be written as \( G = (V_G, E_G) \) and is required to be connected. The problem of finding the shortest path between nodes \( v_1 \) and \( v_2 \) in a dynamically costed graph can be formulated as follows:

- **Instance:** A graph with dynamic costs or dynamically costed graph \( \mathcal{G} = (G, c, \mathbb{R}^+) \) where \( G = (V_G, E_G) \) is a connected graph, and two nodes \( v_1, v_2 \in V_G \).

- **Answer:** A path \( \beta = \langle w_k \rangle_{k=0}^q \) from \( v_1 \) to \( v_2 \) in \( G \), such that \( c(\beta) \leq c(\gamma) \), for all path \( \gamma = \langle w_k \rangle_{k=0}^q \), from \( v_1 \) to \( v_2 \).

**4.1 Traffic distribution problem**

Consider the situation of finding the shortest path between two points \( A \) and \( B \) in a city, whose street structure is describable by a costed graph \( G = (V, E, l) \) where the costs function \( l \) is associate to the length of streets: \( l(e) \) is the length of the street \( e \). In order to drive from point \( A \) to point \( B \) we define for each street \( e \), the average velocity on \( e \) on time instant \( t \) by \( v_e(t) \).

In this sense, the time spent in traversing path \( e \) at time \( t \), \( c_t(e) \), is given by

\[
\int_t^{c_t(e)} v_e(s) ds = l(e). \quad (4)
\]
The time cost of traversing a path \( \beta = (w_k)_{k=0}^{p} \) is given by
\[
c(\beta) = \sum_{k=0}^{p-1} c_{t_k} ((w_k, w_{k+1}))
\]
where \( t_0 = 0 \) and \( t_k = t_{k-1} + c_{t_{k-1}} ((w_{k-1}, w_k)) \) for \( k = 1 \ldots p-1 \).

Notice that the problem posed in this way implies a non-overpass condition: vehicles cannot cut ahead of one another and must finish the path in the same ordering as they enter. Formally speaking, for every edge \( e \in E_G \) and every pair of time instants \( t_0, t_1 \in \mathbb{R}^+ \) where \( t_0 < t_1 \) we must have \( c_e (t_0) \leq c_e (e) + t_0 \).

In terms of bundles of metric spaces \( \mathbb{G} \) of \( \mathbb{G} \), the existence of the number
\[
\tau = c_t ((u, v)) \neq +\infty,
\]
represents the possibility of reach the point \( (v, t + \tau) \) from the point \( (u, t) \).

In this sense, \( \mathbb{G} \) can be associated to an infinite graph \( G = (V_G, E_G) \) where:
- \( V_G = V \times \mathbb{R}^+ \) and
- \( \{(u, t), (v, t')\} \in E_G \) if and only if \( (v, v) \in E \) and \( t' = t + c_t ((u, w)) \).

A characterization of the arcs present in graph \( G \) that is useful comes in terms of the adjacency set of a vertex \( (u, t) \in V_G = V \times \mathbb{R}^+ \). Indeed, by defining the set
\[
adj ((u, t)) = \{(w, t') \mid (u, w) \in E \land t' = t + c_t ((u, w))\}.
\]
To calculate the time-cost of a minimal-cost path between nodes \( u \) and \( w \) in \( \mathbb{G} \) at time instant \( t \) is equivalent to find the minimal value of \( \tau > 0 \) for which
\[
((u, t), (w, t + \tau)) \in G
\]
This problem can be solved, when posed like this, by either introducing a classical sideways-search algorithm in \( G \) starting from point \( (u, t) \) or by using dynamic programming and recursively calculating the expression
\[
D_t (u, w) = \min_{r \in adj (u)} \{ c_t (u, r) + D_{t+c_t (u, r)} (r, w) \},
\]
with \( D_t (w, w) = 0 \), to find the minimal cost path between nodes \( u \) and \( w \) in \( \mathbb{G} \).

5 Maximum flow in dynamically costed graphs
When simulating transport phenomena using graphs, the concept of flow arises in a very natural sense. The amount of material (particles, vehicles or load) that the graph can transport. The problem of maximum flow in classical graph theory, deals with finding the the largest amount of material that can be sent between a node known as the source and a node known as the destination.

More formally, if \( G \) is a graph (or network) with \( n \) vertices, where the first vertex is regarded as the source, and the last one as the destination. If the capacity associated to every edge \( (i, j) \) is denoted by \( w_{ij} \), we need to determine the collection of values \( x_{ij} \) (for \( 1 \leq i < n \) and \( 1 \leq j \leq n \)), such that
- for all \( 1 \leq i < n \) and \( 1 \leq j \leq n \), \( 0 \leq x_{ij} \leq w_{ij} \),
- for all vertex \( 1 < i < n \),
\[
\sum_{k=1}^{n-1} x_{ki} = \sum_{k=1}^{n-1} x_{ik}, \quad (8)
\]
and
- the sum
\[
\sum_{k=2}^{n} x_{ik} = \sum_{k=1}^{n-1} w_{kn} = F \quad (9)
\]
is maximal with regard to every other set of values.

In addition to the solution offered by linear programming, this problem has been explored in classical graph theory, providing solutions as the Ford-Fulkerson and the Edmonds-Karp algorithms [6, 5].

A dynamic version of the maximum flow problem tries to determine the maximum amount of material that can be sent throughout a dynamic network \( G \) in a given time span \( [t_0, t_1] \) from the source to the destination, assuming that the capacity of the edges changes continually according to functions \( w_{ij} (t) \).

The solution to the maximum flow problem in a dynamic graph can be estimated by defining a function \( F : [t_0, t_1] \rightarrow \mathbb{R}^+ \) such that, for every \( t \in [t_0, t_1] \), \( F(t) \) is the value of the maximum flow of graph \( G \) at time instant \( t \) using for example the Ford-Fulkerson algorithm. The maximum flow along time period \( [t_0, t_1] \) can be found using the integral
\[
\int_{t_0}^{t_1} F(t) dt \quad (10)
\]
In order to produce an algorithm approximating the solution to this problem, we define a partition \( \{u_i\}_{i=0}^n \) of interval \([t_0, t_1] , t_0 = u_0 < u_1 < \ldots < u_{n-1} < u_n = t_1 \), and we execute an algorithm to solve the problem of maximal flow on each time instant \( u_i \). If we denote by \( F_i \) the value of function \( F \) at time instants \( u_i \), we can approximate the integral on equation 10 using -among others- Simpson’s rule

\[
\int_{t_0}^{t_1} F(t) dt = \frac{1}{2} \sum_{i=0}^{n-1} (F(u_i) + F(u_{i+1}))(u_{i+1} - u_i).
\]

(11)

6 Conclusions and Future Work

Under the presentation made, the problem of finding the shortest path between nodes on a dynamic graph can be proven to be solvable in polynomial time -whenever conditions of non-overpass are met- [11]. Moreover, the structure and concepts presented in this paper allow the exploration of the generalized shortest-path problem under a new perspective, it is important to notice that this problem is actually NP-hard [12]. In turn, maximum flow in dynamic graphs can be regarded to have a complexity of

\[
\mathcal{O} \left( |E| \max_{i=0,\ldots,n} \left( \max_{e \in E_i} w_{u_i}(e) \right) \right)
\]

with the solution presented here. Furthermore, dynamic graphs can be analyzed using bundles of metric spaces. This induces new perspectives of both the theory of graphs with dynamic costs and introduces new applications of bundles of metric spaces. To be more concrete, this new approximation on the theory of graphs with dynamic costs in such a way that:

1. it establishes an unexplored connection between the theory of graphs with dynamic costs and computable topology, under which several new applications of the theory of Bundles of metric spaces may flourish, and
2. it proposes a new source of ideas and induces unexplored applications for Type-2 Theory of effectiveness (TTE for short), when conditions of computability are imposed on either the base or the fiber space (see [8, 16]).

References

7 Appendix. Bundles of metric spaces.

According to [2] and [7], the category of bundles of metric spaces over a given topological space $T$ can be thought of as a generalization of the category of metric spaces over a field—conveniently named $T$-

Definition 11 (Sections and selections) Let $G$ be a set, $X$ be a topological space and $p : G \to X$ be a surjective function. A selection for $p$ is a function $\alpha : A \subseteq X \to G$ such that $p \circ \alpha$ is the identity function over $Q = \text{dom}(\alpha)$. If $Q = X$ we say that $\alpha$ is a global section. If $\alpha$ is continuous, we say that $\alpha$ is a section.

Given a set $\Gamma$ of selections over a function $p : G \to X$ we say that collection $\Gamma$ is full if for every $u \in G$ we can find $\alpha \in \Gamma$ such that $\alpha(p(u)) = u$.

Definition 12 (Metric for a surjective function) Let $G$ be a set, $X$ be a topological space and $p : G \to X$ be a surjective function. A function $d : G \times G \to [0, +\infty]$ is called a metric for function $p$ if for every $u, v, w \in G$:

1. $d(u, v) = +\infty$ if $p(u) \neq p(v)$
2. $d(u, v) = 0$ if and only if $u = v$
3. $d(u, v) \leq d(u, w) + d(w, v)$

Provided with a metric for a function, we can define $\epsilon$-tubes around selections, that can be used as basic open subsets to generate topologies. An $\epsilon$-tube around a selection $\alpha$ of radius is defined as the set

$$\mathcal{T}_\epsilon(\alpha) = \{ u \in G : p(u) \in \text{dom}(\alpha) \land d(u, \alpha(p(u))) < \epsilon \},$$

the idea behind constructing bundles of metric spaces is to generalize the notion of metrizability. This generalization is condensed in the following definition:

Definition 13 (Bundle of metric spaces) Let $G$ and $X$ be topological spaces, $p : G \to X$ a continuous surjective function. If $d$ is a metric for $p$ such that for every $u \in G$ there exists a local selection $\alpha$ for $p$, such that $u \in \mathcal{T}_\epsilon(\alpha)$ for some $\epsilon > 0$, we say that the triplet $(G, p, T)$ is a bundle of metric spaces, whenever the collection of all $\epsilon$-tubes defines a topology over $G$.

In the language of bundles of metric spaces, the set $X$ is called the base space and has the most structure in terms of topological properties. Space $G$ is called the fiber space, since we can write

$$G = \bigcup_{x \in X} p^{-1}(x)$$

where each $G_x = p^{-1}(x)$ is called a fiber above $x \in X$.

7.1 Existence of Bundles of Metric Spaces: a Theorem

We wish to present a modified version of a central result in computable analysis: an existence theorem for bundles of metric spaces as proven in [1]. This theorem establishes conditions under which metric spaces can occur and its computable version allows the introduction of computability over a bundle, provided with minimal conditions on the base space.

Theorem 2 (Existence of bundles of metric spaces) If $X$ is a topological space, $\beta_X$ a basis for the topology of $X$, $G$ a non-empty set of at most continuum cardinality and $p : G \to X$ a surjective function. Let also $d$ be a metric for $p$ and $\Gamma$ a collection of local selections for $p$. Assume also that

1. For every $u \in G$ and every $\epsilon > 0$, there exists a local selection $\gamma \in \Gamma$ and a rational number $r \in \mathbb{Q} \cap (0, \epsilon)$ such that $u \in \mathcal{T}_r(\gamma)$.
2. For every $\gamma, \zeta \in \Gamma$, the function

$$\Phi_{\gamma \zeta} : \text{dom}(\gamma) \cap \text{dom}(\zeta) \to \mathbb{R}$$

defined by

$$\Phi_{\gamma \zeta}(p) = d(\gamma(p), \zeta(p))$$

is upper-semi-continuous.

Then $G$ can be given a topology $\Sigma$ in such a way that

1. The collection $\beta_G = \{ \mathcal{T}_r(\gamma_Q) \}$, where $r \in \mathbb{Q} \cap (0, +\infty)$, $\gamma \in \Gamma$, $Q \subseteq \text{dom}(\gamma)$ and $Q \in \beta_X$, is a basis for the topology $\Sigma$ over $G$.
2. Under the topology $\Sigma$, $\Gamma$ is a family of local sections over $p$.
3. $(G, p, T)$ is a bundle of metric spaces.

For a proof of the this theorem, and for basic references on this topic, the reader is encouraged to revise [2] and [7].
A Queueing Network Model for Performance Analysis of Single-Radio and Dual-Radio 802.11 Wireless Mesh Networks

MEJDI KADDOUR

Laboratory of Computer Science and Information Technologies of Oran
University of Oran Es-Senia
B.P. 1524 El M’Naouer, Oran, ALGERIA
kaddour.mejdi@univ-oran.dz

Abstract. Wireless mesh networks address the growing requirements for networks that are highly scalable and cost-effective, offering end-users large access areas beyond traditional WLAN boundaries, and provide a viable alternative when wired backhaul cannot be supported or afforded. The success of these networks depends on the availability of accurate models and tools for assessing their performances at each layer. However, to date, most analysis of wireless networks has been focused on single-hop and ad hoc multi-hop networks under saturated conditions. In this paper, we introduce an analytical model specifically designed to the performance evaluation of 802.11-based wireless mesh networks under finite load. This model relies on Markov chains for evaluating packet service delays of mesh clients and mesh routers that operate with 802.11 DCF, and open G/G/1 queueing networks for deriving explicit expressions of average end-to-end delay on single-radio and dual-radio WMNs. Our analytical results are verified through extensive simulations. The results quantify the impact of the dedicated backhaul channel and the strong dependency between end-to-end delay and radio range.

Keywords: WMN, 802.11 DCF, end-to-end delay, Markov model, queueing network, diffusion approximation.

(Received September 12, 2010 / Accepted May 02, 2011)

1 Introduction

The popularity of 802.11 technologies has generated a lot of interest in developing wireless networks that support seamless and ubiquitous access across very large areas. Various access points with large coverage are available to these scenarios but their interconnection remains mostly dominated by the deployment of costly wired backhaul networks. Wireless mesh networks (WMN) [1] offer a promising alternative for building robust and reliable wide-area wireless broadband services, in particular when reducing up-front investments is a key concern as in developing countries. Despite the availability of several wireless products and the simplicity of deployment, huge efforts still needed to address the issues behind WMNs. Many WMNs have inherited the MAC and the physical layers of conventional WLANs which have been proved to lack scalability when applied to large areas and multi-hop network settings [1]. Throughput drops significantly as the number of nodes or hops in a WMN increases. Similar problems exist in other networking layers. Consequently, many existing protocols need to be fully assessed in this context to identify the key issues and to guide their further enhancements. In particular, a predictable and low end-to-end delay of data packets is a critical performance issue in many applications expected to be a driving force in the deployment of WMNs: voice over IP, video streaming, videoconferencing, interactive gaming, etc.

The efficiency of the IEEE 802.11 protocol has been subjected to numerous investigations which targeted the modeling of single-hop 802.11 WLANs to
find the maximum achievable throughput and to characterize capacity-delay tradeoffs [3, 16, 11]. In his famous work, Bianchi [3] derived a model that incorporates the inherent exponential backoff process as a bi-dimensional Markov chain and determines the maximum achievable throughput of 802.11 DCF by assuming that every node is saturated and the packet collision probability is constant regardless of the state or station considered. Despite the remarkable accuracy of the obtained results, the saturation assumption is unlikely to be valid in most real networks. Several other papers in the literature have analyzed the performance of multihop 802.11 networks. In [13], Ray and al. quantified the impact of hidden nodes on the performance of linear wireless networks taking into consideration the effects of queueing and retransmissions at each node. Many other studies apply queueing theory for performance analysis of 802.11 networks. In [8], Dong and al. employ a Markov chain model to analyze the probability of transmission in an arbitrary slot at each node in multihop wireless network. The average end-to-end delay is evaluated through the derivation of channel access delay and the queuing delay at each node modeled as an M/G/1 queue.

In this paper, we introduce a detailed analytical model to evaluate the average end-to-end delay in 802.11 WMNs where nodes operate in DCF mode. We study a particular form of WMNs where mesh routers are organized into honey-grid topology providing access to randomly distributed mesh clients which generate finite load traffic. Moreover, our analysis includes two types of WMNs: single-radio mesh and dual-radio mesh. Unlike some existing work, our analysis of end-to-end delay of data packets relies on an accurate modeling of backoff and collision avoidance mechanisms of 802.11 DCF and takes into consideration several prominent parameters: number of radio channels, the offered and the forwarded load across the network, the expected number of hops between source and destination, and the number of interfering nodes whether they are visible or hidden from the point of view of each intermediate node.

The remaining of this paper is outlined as follows. In section 2, we introduce our network model topology, provide some background about the 802.11 DCF mechanism and introduce the Markov model employed for the evaluation of channel access and transmission delays. In section 3, we give some insights about the well known diffusion approximation and describe the proposed open queueing network for end-to-end delay analysis. Section 4 evaluates the accuracy of our model by comparing the analytical results with those obtained by mean of simulation. The discussion of the various results leads to some considerations about the influential factors that determine end-to-end delay in single-radio and dual-radio WMNs. Concluding remarks and some perspectives are given in section 5.

2 Delay Analysis of 802.11 DCF

As depicted in figure 1, The mesh routers are distributed over a two-dimensional area with limited size called the service area. Their positions constitute a regular lattice called the honey-grid model that enables the radio coverage of all mesh clients disseminated throughout the service area. As stated in [9], this model minimizes the number of expected hops between mesh routers, and enables a perfect overlapping between the positions of the routers on the lattice and the positions of their interfering nodes. The size of the network can be expressed in terms of \( k \) co-centered hexagonal rings around the central node 0, or by \( N \) the total number of nodes in this configuration. In addition, \( m \) mesh clients are uniformly and independently distributed inside the service area delimited by the outmost ring. The radio range of each client is defined by \( \alpha R \), where \( R \) is the maximum achievable radio range and \( \alpha \in [0, 1] \). We define the value of \( \alpha \) in such a way that mesh clients are always covered by at least one mesh router and at most 3 mesh routers.

2.1 Single-Radio Mesh

In this scenario, mesh routers and mesh clients share the same 802.11 radio channel. Thus, the co-located client/router and router/router transmissions interfere with each other. We assume that each mesh client generates traffic of fixed-size data packets following a Poisson process with average rate \( \lambda_c \), destined to a randomly chosen mesh client. The wireless channel is considered as error-free.

The CSMA/CA function of the 802.11 DCF based on binary exponential backoff is modeled through the Markov chain depicted in figure 2. This chain was first proposed in the original work of Bianchi [3], and then extended by [8] to deal with non-saturated traffic conditions. It may be described as follows: the state \textit{Empty} occurs when the node has a empty queue of packets waiting for transmission, whereas the state \textit{FirstPkt} denotes a node which receives its first packet for transmission. The states \((i, j)\) represent a node in the backoff mode, where \(i\) denotes the backoff stage, and \(j\) denotes the value of the backoff window which is uniformly chosen in the range \((0, W_i)\). The backoff window size \(W_i\) is equal to \(2^i W_0\), where \(W_0\) is the initial window.
size. As we assume that the number of retransmission in infinite, this chain has an infinite number of states \((i,j)\).

The states \((\theta', j)\) on the left of the figure denote the post-backoff procedure which runs after a successful transmission. The 802.11 DCF standard [10] specifies that the node must execute this backoff procedure with the initial window value \(W_0\) even if there is no packets waiting in the queue. If no packet arrives during this time, which may happen with a probability \(P_{nk}\), the node returns to the state Empty and senses the channel, otherwise it continues with the regular backoff procedure. We use one instance of this Markov chain for modeling the channel access of mesh clients and another one for modeling the channel access of mesh routers because the expected handled traffic and the number of interfering nodes are not equal in these two cases.

Let define the probabilities associated with the Markov chain’s transitions of mesh client. \(P_{idle1}\) denotes the probability that no transmission occurs in the radio range of a mesh client in a randomly chosen system time slot of duration \(\sigma\), it is given by:

\[
P_{idle1} = (1 - \tau_c)^{n_{ce} + 1} \cdot (1 - \tau_e)^{n_{cr}}
\]

where \(\tau_c\) and \(\tau_e\) are the probabilities of a transmission in the time slot of a mesh client and a mesh router, respectively. \(n_{ce}\) is the expected number of neighbor mesh routers within the coverage area of a mesh client, and \(n_{cr}\) is the expected number of mesh clients within the radio range of a mesh client. \(P_{col1}\) denotes the probability that a transmitted frame encounters a collision, it is given by:

\[
P_{col1} = 1 - (1 - \tau_c)^{n_{ce} + h_{ce}} \cdot (1 - \tau_e)^{n_{cr} + h_{cr}}
\]

where \(h_{ce}\) is the expected number of hidden mesh clients from a mesh client, and \(h_{cr}\) is the expected number of hidden mesh routers from a mesh client. A hidden node from a given node is a neighbor of any neighbor of this node without being its direct neighbor. Note we have derived explicit formulas for \(n_{ce}, n_{cr}, h_{ce}\) and \(h_{cr}\) but we do not give them here for the sake of conciseness.

\(P_e\) denotes the probability that a mesh client has an empty queue. Considering an Poisson arrival process and an average packet service time, \(E[C]\), following a general distribution, \(P_e\) is given by:

\[
P_e = 1 - \lambda_c \cdot E[C]
\]

with \(\lambda_c \cdot E[C] \leq 1\). From the balance equations of the chain in the steady state, we obtain the following relationships:

\[
b_{i,0} = P_{col1}^i \cdot b_{0.0}
\]

\[
b_{i,k} = \frac{W_0 - k}{W_0} \cdot P_{e1} \cdot b_{succ}
\]

\[
b_{i,k} = \frac{W_i - k}{W_i} \cdot P_{col1}^i \cdot b_{0.0}
\]

\[
\tau_c = b_{0,0} \left( \frac{1}{1 - P_{col1}} \right) + b_{empty} \cdot P_{idle1} \left( 1 - e^{-\lambda_c \cdot \sigma} \right)
\]

The probability \(b_{empty}\) that the current state is the state Empty is given by:

\[
b_{empty} = \frac{b_{0.0} \cdot P_e \cdot P_{nk1}}{1 - P_{idle1} \cdot P_{e1} \cdot P_{nk1} \cdot (1 - P_{col1})}
\]

The probability \(P_{nk1}\) may be expressed as

\[
P_{nk1} = \exp \left( -\lambda_c \cdot \left( W_0 + 1 \right) \cdot \sigma_1 \right)
\]

where \(\sigma_1\) denotes the average time between successive timer decrements when the node is in the backoff mode. This timer is frozen each time the node detects an activity in the channel [10]. Substituting (8) in (7), we get:

\[
\tau_c = b_{0,0} \left[ \frac{1}{1 - P_{col1}} + \frac{P_{e1} \cdot P_{nk1} \cdot P_{idle1} \cdot (1 - e^{-\lambda_c \cdot \sigma})}{1 - P_{idle1} \cdot P_{e1} \cdot P_{nk1} \cdot (1 - P_{col1})} \right]
\]
Applying the normalization condition of the chain, we obtain:

\[ \sum_{i=0}^{\infty} W_i \cdot b_{i,k} + \sum_{k=0}^{W_i-1} b_{\emptyset,k} + b_{\emptyset} \cdot (1 + P_{idle}) = 1 \]

By substituting (5), (6) and (8) into this equation:

\[ b_{\emptyset,0} \cdot \left[ W_0 + 1 - (W_0 + 2) \cdot P_{\text{col}} \right] \]
\[ = \left[ (1 - P_{\text{col}}) \cdot (1 - 2P_{\text{col}}) \right] \cdot P_{r \cdot 1} \cdot \left( 2P_{n \cdot 1} \cdot (P_{\text{idle}1} + 1) + W_0 - 1 \right) \]
\[ \cdot \left[ 1 - P_{idle} \cdot P_{r \cdot 1} \cdot P_{n \cdot 1} \cdot (1 - P_{\text{col}}) \right] = 1 \]  \hspace{1cm} (11)

Let \( P_{r \cdot 1} \) denotes the probability that at least one node transmits within the radio range of the mesh client when this latter is in the backoff mode. It can be expressed as

\[ P_{r \cdot 1} = 1 - (1 - \tau_c)^{n_{ce}} \cdot (1 - \tau_r)^{n_{cr}} \]  \hspace{1cm} (12)

The probability \( P_{\text{succ}1} \) that this transmission succeeds is

\[ P_{\text{succ}1} = \left[ n_{ce} \cdot \tau_c \cdot (1 - \tau_c)^{n_{ce}-1} \cdot (1 - \tau_r)^{n_{cr}} + n_{cr} \cdot \tau_r \cdot (1 - \tau_r)^{n_{cr}-1} \cdot (1 - \tau_c)^{n_{ce}} \right] \times \]
\[ \frac{1}{1 - (1 - \tau_c)^{n_{ce}} \cdot (1 - \tau_r)^{n_{cr}}} \]  \hspace{1cm} (13)

Thus, the parameter \( \sigma_1 \) can be expressed now as follows:

\[ \sigma_1 = (1 - P_{r \cdot 1}) \cdot \sigma + P_{r \cdot 1} \cdot P_{\text{succ}1} \cdot (T_s + \sigma) + P_{r \cdot 1} \cdot (1 - P_{\text{succ}1}) \cdot (T_c + \sigma) \]  \hspace{1cm} (14)

where \( T_s \) is the length of a successful transmission time slot, and \( T_c \) is the length of a collision time slot. Let \( \text{Size}(P) \) be the size of data packets, \( H_{\text{phy}} \) and \( H_{\text{mac}} \) be the transmission durations of physical and MAC packet headers, respectively, and \( \delta \) be the propagation delay. \( T_s \) and \( T_c \) are given by:

\[ T_s = DIFS + RTS + CTS + H_{\text{mac}} + \text{Size}(P) + ACK + 3SIFS + 4H_{\text{phy}} + 4\delta \]  \hspace{1cm} (15)
\[ T_c = RTS + CTS + SIFS + DIFS + 2H_{\text{phy}} + 2\delta \]  \hspace{1cm} (16)

As we assume that all mesh routers are equal, the probability that a traffic flow transits by a randomly chosen router is \( \frac{c_r}{N} \), where \( c_r \) is the expected number of router to router hops traversed by data packets. Hence, the expected average arrival traffic on a mesh router is given by:

\[ \lambda_r = \lambda_c \cdot M \cdot \frac{c_r + 1}{N} \]  \hspace{1cm} (17)

By symmetry with the results obtained for mesh clients, we derive in a straightforward manner a set of corresponding relations for mesh routers.

**Expression of packet service time** In 802.11 DCF, the packets arriving at a node experience a different medium access delay depending on whether or not the queue is empty. If the queue in non empty, the node enters a backoff procedure, as illustrated by figure 2, before trying to transmit the packet. Otherwise, one of the following three events can occur: (i) the medium is idle

---

**Figure 2:** Finite Load Markov Chain of 802.11 DCF.
and the node transmits the packet successfully; (ii) the medium is idle but a collision occurs at transmission, the node returns to backoff mode before it retransmits; (iii) the medium is busy, the node goes to the backoff mode before it transmits. Therefore, considering that the node is a mesh client, the service time of a packet arriving to an empty queue, $E[C_e]$, may be expressed as:

$$
E[C_e] = (1 - P_{idle1}) \cdot E[C_e] + P_{idle1} \times (P_{col1} \cdot (T_c + E[C_{ne}]) + (1 - P_{col1}) \cdot T_s)
$$  \hfill (18)

where $E[C_{ne}]$ is the service time of a packet arriving to a non empty queue. This latter is the sum of the total durations needed to reach a transmission state at each attempt, the total collision time and the transmission time. Let $\beta$ represents the number of retransmissions, the probability that the transmission succeeds after $r$ retransmissions is given by:

$$
P(\beta = r) = P_{col1} \cdot (1 - P_{col1})^r
$$

From the Markov chain, the conditional probability of $E[C_{ne}]$ given $r$ retransmission can be expressed as

$$
E[C_{ne} | \beta = r] = T_s + \sigma_1 \sum_{i=0}^{r} \sum_{j=0}^{W_i-1} \frac{1}{W_i} j + r \cdot T_c
$$

$$
= T_s + \frac{\sigma_1}{2} (W_0 \cdot (2^r+1) - r - 1) + r \cdot T_c
$$

Hence,

$$
E[C_{ne}] = \sum_{i=0}^{\infty} E[C_{ne} | \beta = i] \cdot Pr(\beta = i)
$$

We find that $E[C_{ne}]$ is convergent only when $P_{col1} < \frac{1}{2}$ to:

$$
E[C_{ne}] = T_s + T_c \cdot \frac{P_{col1}}{1 - P_{col1}} + \sigma_1 \cdot \frac{W_0 \cdot (1 - P_{col1}) - 1 + 2P_{col1}}{2(1 - 2P_{col1})} \cdot (1 - P_{col1})
$$  \hfill (19)

The average service time $E[C]$ depends in the general case on the state of the node’s queue:

$$
E[C] = (1 - P_{c1}) \cdot E[C_{ne}] + P_{c1} \cdot E[C_e] = ((P_{col1} - 1) \cdot P_{c1} \cdot P_{idle1} + 1) \cdot E[C_{ne}] + P_{c1} \cdot P_{idle1} \cdot (P_{col1} \cdot T_c + (1 - P_{col1}) \cdot T_s)
$$  \hfill (20)

We can derive in a similar way $E[R]$, the average service time on a mesh router. The previously defined variables related to the mesh clients $P_{idle1}$, $P_{col1}$, $P_{col2}$, $P_{nk}$, $\tau$, $\sigma_1$, $E[C]$, and the corresponding ones related to the mesh routers, along with the normalization equations of the chains, can be combined to form a nonlinear system the following constraints: $\lambda_c \cdot E[C] \leq 1$, $\lambda_r \cdot E[R] \leq \frac{1}{2}$ and $P_{col1} < \frac{1}{2}$ and $P_{col2} < \frac{1}{2}$, where $P_{col2}$ is the probability of collision when the frame is transmitted by a mesh router. This system can be solved using numerical techniques.

**The second moment of service time** An accurate analysis of the average packet delay in a queueing network requires the knowledge of the second moment of packet service time. First, let define the second moment of service time of a packet arriving at an empty queue of a mesh client:

$$
E[C^2_e] = (1 - P_{idle1}) \cdot E[C^2_{ne}] + P_{idle1} \cdot (1 - P_{col1}) \cdot T_s^2 + P_{idle1} \cdot P_{col1} \cdot E[(C_{ne} + T_c)^2]
$$

The second moment of service time of a packet arriving at a non empty queue can be obtained as follows:

$$
E[C^2_{ne}] = \sum_{j=0}^{\infty} \sum_{i=0}^{\infty} E[C^2_{ne} | \beta = j] \cdot Pr(\beta = j)
$$

$$
= \sum_{j=0}^{\infty} \sum_{i=0}^{\infty} VAR[B_i] \cdot Pr(\beta = j) + \sum_{j=0}^{\infty} \sum_{i=0}^{\infty} \left( \sum_{j=0}^{\infty} E[B_i] \cdot j \cdot T_c + T_s \right)^2 Pr(\beta = j)
$$  \hfill (21)

where $B_i$ is the duration of the backoff period at the $i$th retransmission attempt. It is given by:

$$
B_i = \sum_{j=0}^{w_i} \sigma_i
$$  \hfill (22)

where $\sigma_i$ is the duration of the backoff slot when the backoff window decrements from $i+1$ to $i$, and $w_i$ is the size of the backoff window uniformly selected between 0 and $2^iW_0 - 1$. $E[B_i]$ and $VAR[B_i]$ are obtained as follows:

$$
E[B_i] = \sigma_i \cdot E[w_i] = \sigma_i \cdot \frac{2^iW_0 - 1}{2}
$$  \hfill (23)
where \( \sigma^2_i \) is given in (14). Also, we have

\[
VAR[B_i] = E[B_i^2] - E[B_i]^2
\]

\[
= E\left[\sum_{j=0}^{w_i} \sigma_i^2 \right]^2 - \sigma_i^2 \cdot \frac{(2W_0 - 1)^2}{4}
\]

As \( w_i \) is uniformly distributed, it is easy to get:

\[
E[w_i^2] = \frac{2W_0 \cdot (2i+1W_0 - 1)}{6}
\] (24)

The variance of \( \sigma_i \) is:

\[
VAR[\sigma_i] = \sigma_2 - \sigma_1^2
\] (25)

with

\[
\sigma_2 = (1 - P_{tr1}) \cdot \sigma^2 + P_{tr1} \cdot P_{succ1} \cdot (T_c + \sigma)^2
\]

\[
+ P_{tr1} \cdot (1 - P_{succ1}) \cdot (T_c + \sigma)^2
\] (26)

Thus, \( VAR[B_i] \) can be expressed as follows:

\[
VAR[B_i] = \sigma_1^2 \cdot \frac{(2W_0 - 1) \cdot (2W_0 - 5)}{12}
\]

\[
+ \sigma_2 \cdot \frac{2W_0 - 1}{2}
\] (27)

By substituting (23) and (27) in (21), and after some straightforward algebra, we find a lengthy expression of \( E[C^2] \), which converges when \( P_{col1} < \frac{1}{2} \).

The second moment of packet service time in the general case can be expressed as follows:

\[
E[C^2] = P_{tr1} \cdot E[C^2] + (1 - P_{tr1}) \cdot E[C^2_{ne}]
\] (28)

The second moment of packet service time at a mesh router can also be obtained in a similar way.

### 2.2 Dual-Radio Mesh

In this second scenario, the mesh routers have two radios operating on different frequencies. One radio is used for mesh client access and the other radio provides wireless backhaul between routers. The radios operate in orthogonal frequency channels so they can run concurrently without interference. The two radio channels are occupied in the following way:

(i) **Client channel**: the mesh clients contend from one part with other neighbor mesh clients transmissions and from the other part with the transmissions from neighbor mesh routers to mesh clients. The probability that a randomly chosen mesh router is the first router on the path for a given mesh client’s traffic is \( \frac{1}{M} \). Hence the average arrival rate \( \lambda_{rc} \) at the router from its client radio interface is equal to \( \frac{\lambda_c}{M} \).

(ii) **Backhaul channel**: the mesh routers contend on this channel to forward client packets. The average arrival rate \( \lambda_{rr} \) is equal to \( \frac{\lambda_c}{M} \) considering that the generated traffic of a given client is forwarded on average \( c_{Ch} \) times inside the WMN.

Average packet service times of mesh clients and mesh routers using client channel, \( E[C] \) and \( E[R_c] \), respectively, can be determined by substituting \( \lambda_c \) with \( \lambda_{rc} \) in the non-linear system defined previously in the case of single-radio mesh. Whereas, the average service time of mesh routers on backhaul channel, \( E[R_r] \), depends only on the contention with interfering mesh routers.

### 3 End-to-End Delay in WMNs

#### 3.1 End-to-End Delay Analysis

Packet end-to-end delay in a WMN can be evaluated by the sum of sojourn times at the source mesh client and intermediate mesh routers. We derive in this section closed-form expressions of packet end-to-end delay in single-radio and dual-radio configurations.

**Single-radio mesh** We model the single radio WMN as a queuing network with two types of stations: \( M \) client stations, numbered from 1 to \( M \), which act as the entry point of external jobs corresponding to data packets, and \( N \) router stations, numbered from \( M + 1 \) to \( M + N \), which either forward data packets to other router stations or either absorb them if they are the last router towards the packet destination. Note that once a packet leaves a client station after its service completion, it goes to a router station and doesn’t return again to a client station.

Let assume that a given traffic with an effective rate of \( \lambda_c \) enters to each client station constituting a total external traffic with average rate of \( M \cdot \lambda_c \). Hence, the visit ratio of a client station \( i \) could be given by \( e_i = \frac{1}{M} \). All routers stations in the network have an equal chance to be an intermediate station of a given packet, hence considering that packets transits \( c_{Ch} \) times on average between router stations before reaching the destination, the visit ratio of the router station \( i \) can be expressed as \( e_{i} = \frac{c_{Ch}}{N} \). Considering that all router stations are similar, the forwarding probability \( p_{f} \), corresponding to the probability that a packet is transferred from client
station $i$ to router station $j$, is given by $p_{ij} = \frac{1}{N}$. By the same similarity rule, we could express the probability that a mesh router absorbs a packet as $\frac{1}{N}$. Thus, the forwarding probability, denoted by $q_{ij}$, corresponding to the probability that a packet is transferred between the router stations $i$ and $j$ ($i \neq j$) is the joint probability that $i$ doesn’t absorb the packet and also transfer it to $j$. It is given by:

$$q_{ij} = \left(1 - \frac{1}{N}\right) \cdot \frac{1}{N - 1} = \frac{1}{N} \quad (29)$$

Using the diffusion approximation [5], we evaluate the squared coefficient of variation of interarrival times at router station $i$ as

$$c_{Ai}^2 = 1 + \sum_{j=1}^{M} \left( c_{Bj}^2 - 1 \right) \cdot p_{ij}^2 \cdot e_j \cdot e_i^{-1} + \sum_{j=M+1, j \neq i}^{M+N} \left( c_{Bj}^2 - 1 \right) \cdot q_{ji}^2 \cdot e_j \cdot e_i^{-1} \quad (30)$$

Let $c_{Be}$ and $c_{Br}$ denote the squared coefficient of variation of service times of client stations and router stations, respectively. Assuming that all routers are identical, the squared coefficient of variation of interarrival times at a router station is hence given by

$$c_{Ar}^2 = 1 + \frac{c_{Be}^2 - 1}{N \cdot (e_h + 1)} + (N - 1) \cdot \frac{c_{Br}^2 - 1}{N^2} \quad (31)$$

where from the results obtained in section 2.1, $c_{Be}^2$ and $c_{Br}^2$ are given by:

$$c_{Be}^2 = \frac{E[C^2] - E[C]^2}{E[C]^2} \quad (32)$$

$$c_{Br}^2 = \frac{E[R^2] - E[R]^2}{E[R]^2} \quad (33)$$

Also according to the diffusion approximation, the following approximated expression for the steady state probabilities of the number of jobs at a router station can be obtained:

$$\hat{p}(k) = \begin{cases} 1 - \rho_r & k = 0 \\ \rho_r \cdot (1 - \hat{\rho}_r) \cdot \rho_r \cdot e_i^{k-1} & k > 0 \end{cases} \quad (34)$$

with:

$$\rho_r = \frac{M \cdot \lambda_e \cdot (e_h + 1) \cdot E[R]}{N} \quad (35)$$

$$\hat{\rho}_r = \exp \left( - \frac{2(1 - \rho_r)}{c_{Ar}^2 \cdot \rho_r + c_{Br}^2} \right) \quad (36)$$

From little law’s [2], the average sojourn time of packets at mesh routers, denoted $T_r$, can now be derived as

$$T_r = \frac{E[R]}{1 - \rho_r} \quad (37)$$

On the other hand, as the arrival traffic at client stations follows an exponential distribution, we can use an M/G/1 queue to analyze their average service delay. According to the Pollaczek-Khinchin formula [2], the expected waiting delay, denoted $W_e$, at client stations is

$$W_e = \frac{\lambda_r \cdot E[C^2]}{1 - \lambda_r \cdot E[C]} \quad (38)$$

Therefore, the average end-to-end packet delay, $E[D]$, in a single-radio mesh is the sum of queuing and packet service times at the source client station, and sojourn times at intermediate router stations:

$$E[D] = W_e + E[C] + (e_h + 1) \cdot T_r \quad (39)$$

**Dual-radio mesh** The modeling of the dual-radio configuration is accomplished by inserting $N$ other stations, numbered from $M + N + 1$ to $M + 2N$, to the previous queueing network. These stations, referred to as last-hop stations, represent mesh routers transmitting packets to a mesh client using the client channel. Packets arrive to these stations either from client stations or from backhaul stations which represent mesh router transmitting on backhaul channel. Last-hop stations absorb all the received packets. We refer to the router stations in the network by backhaul stations.

Data packets go through one hop less on average in backhaul channel than in single-radio configuration. Hence, the visit ratio of a backhaul station $i$ becomes in this case $e_i = \frac{1}{N}$. Likewise, the visit ratio of a last-hop station $i$ corresponds to the probability that a given router is the last one towards the recipient client, which is given by $e_i = \frac{1}{N}$. The forwarding probabilities between the stations of the network can be expressed as follows: from client station to backhaul station $\frac{N}{N-1}$; from client station to last-hop station $\frac{1}{N}$; from backhaul station to backhaul station $\frac{1}{N}$; and from backhaul station to last-hop station $\frac{1}{N(N-1)}$.

Therefore, the squared coefficient of variation of interarrival times at backhaul stations and last-hop sta-
lations, respectively $c_{Arr}$ and $c_{Arc}$, may be expressed as:
\[
c_{Arr}^2 = 1 + \frac{(N - 1)^2 (c_{Brc}^2 - 1)}{N^3 e_h} + \frac{(N - 1) (c_{Brr}^2 - 1)}{N^2} \tag{40}
\]
\[
c_{Arc}^2 = 1 + \frac{c_{Brc}^2 - 1}{N^3} + \frac{e_h (c_{Brr}^2 - 1)}{N^2 (N - 1)} \tag{41}
\]
where $c_{Brr}^2$ denotes the squared coefficient of variation of service time at backhaul stations. Consequently, the average service delays at backhaul stations and last-hop stations, respectively $\bar{T}_{rr}$ and $\bar{T}_{rc}$, may be derived as:
\[
\bar{T}_{rr} = \frac{E[R_e]}{1 - \hat{\rho}_{rr}} \tag{42}
\]
\[
\bar{T}_{rc} = \frac{E[R_e]}{1 - \hat{\rho}_{rc}} \tag{43}
\]
where $E[R_e]$ and $E[R_c]$ denote the average packet service time at backhaul stations and last-hop stations, respectively, and
\[
\hat{\rho}_{rr} = \exp\left(-\frac{2(1 - \rho_{rr})}{c_{Arr}^2 \cdot \rho_{rr} + c_{Brr}^2}\right) \tag{44}
\]
\[
\hat{\rho}_{rc} = \exp\left(-\frac{2(1 - \rho_{rc})}{c_{Arc}^2 \cdot \rho_{rc} + c_{Brr}^2}\right) \tag{45}
\]
\[
\rho_{rr} = \frac{M \cdot \lambda_c \cdot c_{Brr}}{N} \tag{46}
\]
\[
\rho_{rc} = \frac{M \cdot \lambda_c \cdot E[R_e]}{N} \tag{47}
\]
Finally, the expression of average end-to-end packet delay in a dual-radio wireless mesh is:
\[
E[D] = \bar{W}_c + E[C] + e_h \cdot \bar{T}_{rr} + \bar{T}_{rc} \tag{48}
\]
Note that $E[C]$ refers here to the average packet service time at client station in a dual-radio configuration.

4 Numerical Results

We present in this section some results of both analysis and simulation to evaluate the effectiveness of the proposed analytical model. The mean and the second moment of channel access delays are computed by solving the non-linear systems defined in section 2 using the IPOPT software package for large-scale nonlinear optimization [15]. Simulations are performed using NS-2 [12]. We define two network topologies: (i) a one-ring network with 7 mesh routers and a number of mesh clients varying from 10 to 20 nodes; (ii) a two-ring network with 19 mesh routers and 40 mesh clients. The mesh clients are randomly distributed into the service area. The radio range of clients varies from $R_1 = (\frac{\sqrt{3}}{3} + 0.1)R$, $R_2 = (\frac{\sqrt{3}}{3} + 0.2)R$, to $R_3 = (\frac{\sqrt{3}}{3} + 0.25)R$ in each scenario. An exponential traffic source is attached to mesh clients producing packets of length 2000 bytes at several rates. Each mesh client selects a random mesh client destination. As a mean to exclude the mesh clients from participating in packet forwarding, the routes are built and maintained using the OLSR-UM [14] which implements the base specification of the proactive routing protocol OLSR [7] for NS-2 simulator. OLSR includes a very useful option, called willingness, which specifies the willingness of a node to carry and forward traffic for other nodes. Also, we minimize the frequency of exchanged routing updates to reduce their impact on data traffic. We summarize in table 1 the parameters used in both simulations and analytical model corresponding to the DSSS PHY layer with a raw bit rate of 11 Mbit/s.

Figure 3 and 4 plot the results of average end-to-end delays of the analytical model faced to those obtained from NS-2 simulation. We observe in case of light traffic, the analytical model provides accurate results for each WMN and each radio range. However, as the traffic load in the network increases, the average end-to-end delay of simulation becomes significantly larger than the results obtained through the analytical
model. This divergence has been also outlined in [4]. The main reason is that simulations rely on shortest path routing which may result in heavily loaded mesh routers across the network, whereas the analytical model relies on a probabilistic routing where the traffic is spread uniformly over the mesh routers.

Figures 5 and 6 compare the average end-to-end delays obtained with single-radio and dual-radio WMNs in 1-ring and 2-ring WMNs, respectively. The solid lines are for single-radio delays while the dotted lines are for dual-radio delays. The results show clearly that average delay depends on network load, the density of the network and the expected number of hops. In each configuration, the average delay grows exponentially as the network load approaches the saturation throughput. As demonstrated in [6], these results confirm that the binary exponential backoff tends to be very harmful in saturated multi-hop network where each packet is repeated by the adjacent router.

Theses figures show without surprise that in almost all cases dual-radio settings keep the average delay lower than single-radio do as the network load increases. This can be explained by the fact that as the load increases, the collision probability of packets, and therefore the backoff stage, increases more slowly in dual-radio where the forwarded packets are transmitted on a separate channel. The benefit of having two radio channels becomes even more substantial as the size of the network increases because the forwarded packets traverse more hops on average and consequently preventing mesh clients from accessing backhaul channel contribute to reducing delay more noticeably.

Furthermore, we observe that radio range of mesh clients is another predominant factor. When the network density is low, e.g. 1-ring WMN with 10 nodes, 2-ring WMN with 20 nodes, the highest radio range turns out to be most suitable because the gain from traversing less hops in average overcomes the inconvenient of having more interfering neighbors. This observation in inverted when the network density is high, e.g. 1-ring WMN with 20 nodes, 2-ring WMN with 60 nodes.

Table 1: PHY and MAC system parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packet payload</td>
<td>2000 bytes</td>
</tr>
<tr>
<td>MAC header</td>
<td>272 bits</td>
</tr>
<tr>
<td>PHY header</td>
<td>192 bits</td>
</tr>
<tr>
<td>RTS</td>
<td>160 bits + PHY header</td>
</tr>
<tr>
<td>CTS</td>
<td>112 bits + PHY header</td>
</tr>
<tr>
<td>ACK</td>
<td>112 bits + PHY header</td>
</tr>
<tr>
<td>Channel Bit rate</td>
<td>11 Mbps</td>
</tr>
<tr>
<td>Preamble rate</td>
<td>1 Mbps</td>
</tr>
<tr>
<td>Propagation Delay</td>
<td>1 \mu s</td>
</tr>
<tr>
<td>Slot Time</td>
<td>20 \mu s</td>
</tr>
<tr>
<td>SIFS</td>
<td>10 \mu s</td>
</tr>
<tr>
<td>DIFS</td>
<td>50 \mu s</td>
</tr>
</tbody>
</table>

Figures 5: Comparison of average end-to-end delays in 1-ring WMN.

Figures 6: Comparison of average end-to-end delays in 2-ring WMN.
Moreover, it’s interesting to note that some cases reveal that a lowest radio range have more impact in reducing delay than using a second radio channel. As an example, in a 1-ring WMN with 20 nodes, the dual-radio average delay using range $R_3$ is higher than the single-radio one using ranges $R_1$ or $R_2$.

5 Conclusion
This paper presented an analytical model using Markov chains and queueing networks to express the average end-to-end delay of WMNs operating with 802.11 DCF, the most widely accepted standard for wireless LANs. The derivation of average packet service times considers carefully the effects of visible and hidden interfering nodes. The proposed open G/G/1 networks determine the average waiting times in the queues of mesh clients and mesh routers, hence enabling the evaluation of end-to-end delays. The analytical results, which were validated through extensive simulations, show the impact of network density, the generated load on each mesh client, and the client’s radio ranges on the delay. We have pointed out the dramatic effect of using a separate backhaul channel on reducing end-to-end packet delays, particularly in large WMNs with low density of clients. Furthermore, we have exhibited that using high radio range in single-radio mesh is the best option in term of reducing delay when the number of nodes is low, whereas using a low radio range is the best one in the opposite case. The proposed model leads to several venues for future work. Our current directions include end-to-end delay and throughput analysis of multi-radio WMNs and the analysis of other channel access mechanisms, such as 802.11s. We envisage also to integrate more advanced routing protocols and QoS mechanisms in our model and study their effects on performance.

References
Abstract. The Binary Decision Diagram (BDD) is used to represent in symbolic manner a set of states. It is largely used in the field of formal checking. The variable ordering is a very important step in the BDD optimization process. A good order of variables will reduce considerably the size of a BDD. Unfortunately, the search for the best variables ordering has been showed NP-difficult. In this article, we propose a new iterative approach based on a hybrid Genetic Algorithm and Variable Neighborhood Search Algorithm. The obtained results are very encouraging and show the feasibility and effectiveness of the proposed hybrid approach.

Keywords: Binary decision diagram, Genetic algorithm, Variable neighborhood search, Hybrid methods

1 Introduction

The objective of the checking application and electric circuits is to detect the errors which they contain or to show that they function well. One of the methods used in system checking is the model-checking [17]

One of the difficulties encountered in the domain of formal verification is the combinatorial explosion problem. For example in the model checking, the number of states in the transition graphs can reach prohibitive level, which makes their manipulation difficult or impossible. Consequently, compression methods are used in order to reduce the size of the state graph. The compression is done by using data structures in order to represent in a concise manner the set of states. In this case, the operations are done so on set of states rather than on explicit states.

The representation by the Binary Decision Diagrams BDD [6] is among the most known symbolic notations. The BDD is a data structure used to represent Boolean functions. The BDD is largely used in several fields since they offer a canonical representation and an easy manipulation. However, the BDD size depends on the selected variable order. Therefore it is important to find variable order which minimizes the number of nodes in a BDD. Unfortunately, this task is not easy considering the fact that there is an exponential number of possible variable ordering. Indeed, the problem of variable ordering was shown NP-hard [2]. For that, several methods were proposed to find the best BDD variable order and which can be classified in two categories. The first class tries to extract the good order by inspecting the logical circuits [10], whereas, the second class is based on the dynamic optimization of a given order[11].

Evolutionary computation has been proven to be an effective way to solve complex engineering problems.
It presents many interesting features such as adaptation, emergence and learning. Artificial neural networks, genetic algorithms and artificial immune systems are examples of bio-inspired systems used to this end. One of the iterative methods that have been developed recently to solve this type of problem is Genetic Algorithms GA [16, 8]. It is a stochastic iterative algorithm which maintains a population of individuals. GA adapts nature optimizing principles like mechanics of natural selection and natural genetics. Each individual represents a feasible solution in the problem search space. Basically, a genetic algorithm consists of three essential selection, crossover, and mutation. The selection evaluates the fitness of each individual and keeps the best ones among them. The others are removed from the current population. The crossover merges two individuals to provide new ones. The operator of mutation allows moving each solution to one of its neighbours in order to maintain a good diversity during the process of optimization. GA allows guided search that samples the search space. Although GAs have been showed to be appropriate for solving BDD ordering problem [9], their computational cost seems to be a dissuasive factor for their use on large instances. To overcome this drawback and in order to get better speed and quality convergence, their implicit parallelism is exploited.

Metaheuristics are a family of optimization techniques inspired by nature used to solve difficult optimization problems for which we do not know the most effective method. The heuristic search procedures are recognized as the neighboring structures (NS), which transform a solution to one of its neighbours by applying some perturbations. Variable neighbourhood search (VNS) [14] is a recent metaheuristic for solving combinatorial optimization problems. The main idea of VNS is to change systematically the neighbourhood by simply passing one NS to another while they execute a search Local. The VNS changes the neighbourhood structure where the search is trapped in a local minimum. VNS offers a multiple neighbourhood structure with which one recovers the solutions trapped via the others. The main idea here is to choose heuristics (neighbourhood structures) complementary to each other [3].

In this context, we propose in this article, a new iterative approach called GAVNSBDD based on a hybrid GA and variable neighbourhood search. For that, a problem formulation in terms of genetic representation and evolutionary dynamic borrowing evolutionary operators were defined. To foster the convergence to optimality, the VNS has been embedded within the optimization process. VNS helps the search process to avoid local optima and explores the solution space economically and effectively without getting trapped into cycles. The experiences carried out on GAVNSBDD showed the feasibility and the effectiveness of our approach.

Consequently, the remainder of the paper is organized as follows: section 2 presents some basis concepts of BDD. A brief introduction to variable neighbourhood search is presented in section 3. The proposed approach is described in section 4. Section 5 illustrates some experimental results. Then, we finish by giving conclusion and some perspective.

2 Binary Decision Diagram

A Binary Decision Diagram or BDD is a data structure used for representation of Boolean functions in the form of rooted directed acyclic graph. A BDD is a rooted directed acyclic graph \( G = (V, E) \) with node set \( V \) containing two kinds of nodes, non-terminal and terminal nodes (Figure 1). A non-terminal node \( v \) has as tag a variable \( \text{index}(v) \in \{x_1, x_2, \ldots, x_n\} \) and two children low\( (v) \), high\( (v) \in V \). The final nodes are called 0-final and 1-final. A BDD can be used to compute a Boolean function \( f(x_1, x_2, \ldots, x_n) \) in the following way. Each input \( a = (a_1, a_2, \ldots, a_n) \in \{0, 1\}^n \) defines a computation path through the BDD that starts at the root. If the path reaches a non-terminal node \( v \) that is labelled by \( x_i \), it follows the path low\( (v) \) if \( a_i = 0 \), and it follows the path high\( (v) \) if \( a_i = 1 \). The label of the terminal node determines the return value of the BDD on input \( a \). The BDD is called "ordered" if the different variables appear in the same order on all the ways from the root (figure 1). It is important to note that for a given order of variables, the minimal binary decision graph is single. A BDD can be reduced while using the two following rules [7, 6, 4]:

- Recognize and share identical sub-trees.
- Erase nodes whose left and right child nodes are identical.

It is very important to take into account the order of variables to be used when using the BDD in practice. The size of a BDD is largely affected by the choice of the variable ordering (Figure 2).

Unfortunately, there are an exponential number of possible orders (permutation). It is completely clear that the problem of variables ordering is NP-difficult. The use of heuristics is essential to find acceptable solutions within reasonable times. Within this perspective, we are interested in applying evolutionary computing principles to solve the variable ordering problem.
3 Variable neighbourhood search

A Variable Neighbourhood Search (VNS) is a recent metaheuristic based on the idea of a systematic change in the search space. Basically, a local search algorithm performs exploration in a limited region of in the candidate solutions space. The effectiveness of VNS compared to simple local search method is based on the systematic change of neighbourhood. A simple algorithm VNS starts from an initial solution from the search space, than enhances it through a two nested loop in which the core one alters and explores via two main functions so called shake and local search. Shaking step is performed by randomly selecting a solution from the first neighbourhood. This is followed by applying a local search algorithm. This procedure is repeated as long as a new incumbent solution is found [14, 3].

In order to develop a competent VNS algorithm, the structure of neighbourhood and heuristic functions used must be selected with great accuracy in order to construct a powerful VNS algorithm. We can use more than one neighbourhood structure for shake and local search procedures. The VNS comprises the following steps:

1. **Initialization**: Find an initial solution $x$.

2. **Repeat**: the following steps until the stopping condition is met:
   
   (a) **Shake Procedure**: Generate at random a starting solution $x'$.

   (b) **Local Search**: Apply a local search from the starting solution $x'$ using the base neighbourhood structure until a local minimum $x''$ is found.

   (c) **Improve or not**: If $x''$ is better than $x$, do $x \leftarrow x''$.

The integration of the VNS in the genetic algorithm can be done in two ways. This integration can replace completely the mutation operation in a simple genetic algorithm, as we can combine the two operations into a single operation of mutation. Although this method has proved effective in solving several difficult problems, it remains, nonetheless difficult to adapt to the problem of BDD ordering. This is mainly due to the large number of parameters to define: initial solution, local search method, neighbourhood function, number of neighbours to explore.

3.1 Initial solution

It has been shown that the effectiveness of approaches based on the principle of local search depends deeply on
the quality of the initial solution. In our approach, the initial solution is obtained from the genetic algorithm and passed as a parameter to the operator of VNS. So, the solutions passed to the VNS procedure are mainly dependent on the quality of the current individual population in the genetic algorithm.

3.2 Neighbourhood function

The choice of the neighbourhood is very important in this type of local search method; it must be a compromise between efficiency and quality. The complexity of solving approach based on VNS depends mainly on the size of the current solution neighbourhood and the valuation method of each of these neighbours to determine which minimizes the cost function. It has been demonstrated that almost 90% of the execution time is spent in neighbourhood’s evaluation. Therefore, it is interesting to use if possible neighbourhoods with constraints to reduce the runtime complexity. The best neighbour generated belonging to the first neighbourhood is selected and the total number of neighbourhoods to generate for each iteration is well defined. The mechanism for the generating of neighbouring solutions is based on choosing a first point to move, then a second point which will be the target of movement. We selected a number of neighbourhoods equal to two, the point of movement is selected from the first neighbourhood defined by the VNS or we undertake a systematic change in the neighbourhood.

4 The Proposed Approach

The development of the suggested approach called GAVNSBDD is based basically on a genetic representation of the research space associated with the problem and an evolutionary dynamic used to explore this space by operating on the genetic representation by using evolutionary operations.

4.1 The fitness function

To select the best individuals of the current population, we must first evaluate these individuals to compute their efficiency or adaptation. The fitness in our case is the number of nodes in the resulting BDD. The best individual is one who gives a minimum number of nodes; the greater the size of the BDD is minimal, the resulting solution is optimal. In order to compute the fitness of variable order, we must first construct the corresponding BDD and then we compute the number of nodes in the resulting BDD. The implementation of the fitness function must be especially optimized because it will be executed many times during the execution of the genetic algorithm. Consequently, the algorithm convergence depends largely on the runtime of the fitness function.

4.2 Genetic representation of variable order

The problem of variable ordering can be mathematically formulated as follow. Given a set of variables \( V = \{x_1, x_2, \ldots, x_n\} \), the problem of BDD variables ordering can be defined by specifying implicitly a pair \((\Omega, SC)\) where is the set of all possible solutions that is potentials variables order and \(SC\) is a mapping \(\Omega \rightarrow \mathbb{R}\) called score of the variable ordering. This score is the BDD size. Each solution is viewed as permutation of the \( V\) variables. Consequently, the problem consists to define the best permutation of \( V\) that gives the minimal BDD size. In our approach, the variable order is represented as integer vector (figure 3) satisfying the following criteria:

- For \( N\) variable, the size of the vector is \( N\).
- There is no repetitive variable in this vector

![Figure 3: Genetic representation of the variable ordering](image)

4.3 Initial Population

The initial population is an important factor in evolutionary algorithms. The generation process of individuals in this population must be carefully selected in order to provide to the genetic algorithm a set of potential and diverse individuals. The goal is to construct a number of different variable orders to be used by the genetic algorithm. In our case, the initial population was created randomly. However, it is important to use heuristics to construct initial solutions of good quality and thus to reduce the convergence time. Another side, the population size has a great impact on the performance of evolutionary algorithms. A large population represents a large space search of solutions, but increases the computational cost. On the other hand, a small population size can lead to local solutions. View we have used a local search method in the core of genetic algorithm, it is preferably to reduce the population size. We have found that a population between 10 and 30 can give good results.
4.4 Genetic operators

Selection: The selection is to select individuals that have the best fitness to construct the intermediate population in order to apply the evolutionary operators. Each individual is selected with a probability related to its fitness. Thus, individuals with high affinity value have more chance of being selected for the next generation. Selection strategies that we can use in genetic algorithms such as [8]:

- The roulette wheel selection.
- The opposite roulette wheel selection.
- The tournament selection.
- Selection by rank.

In our approach, we have used the roulette wheel selection. The basic motivation for adopting this fitness proportional rule is to preserve diversity of good and bad individuals in the pool, so that it can contain a large space search of potential solutions and to avoid being trapped in local minimum.

Mutation operator: this operator performs permutation between two variables (figure 4). It allows moving from the current solution to one of its neighbors. It consists first in selecting pairs of two variables are chosen randomly according to a defined probability, and then swap these two variables as in figure 4. This operator allows exploring new solutions and thus enhances the diversification capabilities of the search process.

Crossover operators: Crossovers are important for promoting the exchange of high quality blocks within the population (figure 5). They exchange subparts of two chromosomes. We have used Partial Mapped Crossover (PMC). The PMC was recommended by Goldberg and Linge [13]. It passes ordering and value information from the parent orders to the offspring orders. A portion of one parent’s string is mapped onto a portion of the other parent’s string and the remaining information is exchanged.

4.5 Local search procedure

To improve the efficiency of the exploration process, we incorporated a local search method in evolutionary dynamics. This form of hybridization has proved advantageous in the context of BDD problems. Indeed, the performances of the local search method allow deeper exploration of certain solution space identified as particularly promising. On the other hand, the diversification ability of the genetic algorithm allows the periodic shift of the solution search to regions rarely visited so far. The proposed hybrid approach is based on genetic algorithm enhanced by the local search method called variable neighbourhood search.

The local search method used in the VNS procedure is the Tabu Search (TS) [1, 12]. TS is among the most popular and robust local search methods. TS is found to be practical in many hard combinatorial optimization problems. The general procedure of TS is given in the figure 6. The main idea underlying is to diversify the search and avoid becoming trapped in local by forbidding or penalizing moves which take the solution, in the next iteration, to point in the solution space previously visited. For this, the algorithm creates a memory list of undesirable moves called "tabu list".

However, Tabu restrictions may be overruled under certain conditions, in which, a Tabu move leads to a better solution, this principle is called aspiration criterion.
Our approach is flexible, so we can use other stochastic local search algorithms.

4.6 Outline of the proposed framework

Now, we describe how the representation scheme including genetic representation and evolutionary operators has been embedded within a variable neighbourhood search algorithm and resulted in a hybrid stochastic algorithm performing BDD variable ordering.

Our approach is an evolutionary algorithm based on a genetic core. The approach consists of a population of individuals where each individual represents a specific variable order. To find the best solution, the optimization process of our algorithm consists of a set of steps. The first step is to create the initial population generated randomly. At each iteration of the algorithm, we apply the classical operators: selection, crossover, mutation, fitness evaluation, etc. To increase the performance optimization of our approach, we have incorporated variable neighbourhood search (VNS) in the genetic core. In more details, the proposed approach can be described as follow:

1. Construct the initial Population of Chromosomes POP
2. Evaluate the population
3. save the best solution
4. Repeat
5. Apply a crossover operation on POP according to crossover probability.
6. Apply a mutation operation on POP according to the probability pm.
7. Apply the variable neighbourhood search.
8. Evaluate the new population.
9. Update the best solution
10. Apply selection and reduce and merge operators
11. Until a termination-criterion is reached

Output: the best variable order.

5 Implementation and Evaluation

Our approach is implemented with Visual C ++ 2008 and tested on a PC with a 3.0 GHZ processor and 512 MB of memory. We have used the Paradiseo platform [5] to implement our approach. In addition, we have used the package Buddy [15], which contains a set of tools for creating and handling BDDs. For the performance evaluation of our approach, we have used several set of tests created with the gates NOT, AND, XOR, NAND. The Building of complex circuits is made by using the ITE operator. An operator, \( \text{ite} \), is defined as follows: for logic functions \( f, g, \) and \( h \), \( \text{ite}(f, g, h) = f.g + \sim f.h(\sim \text{not}) \). The \( \text{ite} \) operator can be used to realize all Boolean operations with two variables. For example, \( f + g = \text{ite}(f, 1, g) \), \( f.g = \text{ite}(f, g, 0) \), ... etc. We compared the results found by our approach with those of two programs based on a pure genetic algorithm and a pure VNS using Friedman statistic tests. The Friedman test is used test whether the difference between the medians of the methods is not significant. Test results are shown in tables 1, 2, 3. In each table, the second column shows the type of multifunction operator implemented by ITE, the third column contains the size of the BDD obtained from the method GABDD based on pure genetic algorithm, and the last column shows the size of the BDD obtained from the method GA VNSBDD based on hybrid method between GA and VNS. In all experiments, the parameters that we used in our method are: the population size is 10, the mutation rate is 0.1, the taboo list is static array of size 10, the number of the generated neighbours is 2 and the number of iterations in the genetic algorithm core is 100. In GABDD method, it was used the following parameters: the population size is 10, the mutation rate is 0.01, and the number of iterations is 1000.

The results of our method illustrate clearly the effectiveness of merging the genetic algorithm and VNS...
algorithm to perform the binary decision diagram ordering problem. The Freidman test (figure7) confirms that our approach ranks high in this experiment. However the performance of the pure genetic algorithm GABDD is a poor compared to GAVNSBDD or VNS-BDD. Moreover, Friedman’s test shows that the results obtained by the pure VNS approach are near to those of the hybrid method GAVNSBDD, this reflects the good method that uses VNS to change systematically the neighbourhood to find the optimal solution. The effectiveness of our approach is explained by the good combination between diversification and intensification which leads the algorithm to effectively explore the search space and locate a good solution.

The search for the optimal solution is not performed randomly but it runs iteratively by gradually improving the solution until the stop criterion is satisfied. Figures 8 and 9 show how the algorithm converges to the optimal solution. In the first figure (obtained by a test of two functions each of 20 variables), our method starts with an initial solution equal to 766 then it is gradually improved, sure there are mediocre solutions but it’s part of the evolution process which leads to better solutions. The algorithm then converges gradually to reach a solution equal to 200 which is the optimal solution found.

![Figure 7: Freidman test The nearest to zero is the best program](image1)

![Figure 8: The behaviour of the best fitness (two functions of 20 variables)](image2)

![Figure 9: The behaviour of the best fitness (two functions of 40 variables)](image3)

<table>
<thead>
<tr>
<th>Table 1: Results: BDD functions with 20 variables</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Initial solution</strong></td>
</tr>
<tr>
<td><strong>F xorG</strong></td>
</tr>
<tr>
<td><strong>F andG</strong></td>
</tr>
<tr>
<td><strong>F orG</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2: Results: BDD functions with 30 variables</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Initial solution</strong></td>
</tr>
<tr>
<td><strong>F xorG</strong></td>
</tr>
<tr>
<td><strong>F andG</strong></td>
</tr>
<tr>
<td><strong>F orG</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 3: Results: BDD functions with 40 variables</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Initial solution</strong></td>
</tr>
<tr>
<td><strong>F xorG</strong></td>
</tr>
<tr>
<td><strong>F andG</strong></td>
</tr>
<tr>
<td><strong>F orG</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 4: Results: BDD functions with 60 variables</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Initial solution</strong></td>
</tr>
<tr>
<td><strong>F xorG</strong></td>
</tr>
<tr>
<td><strong>F andG</strong></td>
</tr>
<tr>
<td><strong>F orG</strong></td>
</tr>
</tbody>
</table>
6 Conclusion

In this work, we have proposed a new approach called GAVNSBDD to deal with the BDD variable problem. GAVNSBDD is based on a hybridizing of genetic algorithm and variable neighbourhood search method. The experimental studies prove the feasibility and the effectiveness of our approach. We have shown that the uses of variable neighbourhood search can help the genetic algorithm to find better solutions. In the future, we try to use other search local strategies more adapted to the BDD problem like sifting technique. In order to accelerate our approach, we may apply parallelization techniques. Finally, the performance of the algorithm may be improved by using a clever startup solution.

References


Abstract. Let $l$ be a positive integer, and $G$ be a graph with nonnegative integer weights on edges. Then a generalized vertex-coloring, called an $l$-vertex-coloring of $G$, is an assignment of colors to the vertices in such a way that any two vertices $u$ and $v$ get different colors if the distance between $u$ and $v$ in $G$ is at most $l$. A coloring is optimal if it uses minimum number of distinct colors. The $l$-vertex-coloring problem is to find an optimal $l$-vertex-coloring of a graph $G$. In this paper we present an $O(n^2 + n\Delta^{l+1})$ time algorithm to find an $l$-vertex-coloring of a tree $T$, where $\Delta$ is the maximum degree of $T$. The algorithm takes $O(n^2)$ time if both $l$ and $\Delta$ are bounded integers. We compute the upper bound of colors to be $1 + \Delta \left(\frac{\Delta-1}{\Delta-2}\right)^{\lceil l/2 \rceil} - 1$. We also present an $O(n^2 + n\Delta^{l+1})$ time algorithm for solving the $l$-edge-coloring problem of trees. If both $l$ and $\Delta$ are bounded integers, this algorithm also takes $O(n^2)$ time.

Keywords: Algorithm, Chordal Graph, $l$-chromatic-number, $l$-edge-coloring, $l$-vertex-coloring, Graph, Tree.

(Received January 25, 2011 / Accepted May 09, 2011)

1 Introduction

A vertex coloring of a graph $G$ is an assignment of colors to the vertices in such a way that any adjacent vertices get different colors [14]. Let $l$ be a positive integer, and $G$ be a graph with nonnegative integer weights on edges. Then an $l$-vertex-coloring of $G$, is an assignment of colors to the vertices in such a way that any two vertices $u$ and $v$ get different colors if $\text{dist}(u,v) \leq l$, where $\text{dist}(u,v)$ denotes the distance between $u$ and $v$ in $G$, that is the length of the shortest path between $u$ and $v$ in $G$. Clearly an ordinary vertex coloring is a 1-vertex-coloring of a graph when weight of each edge is one. The chromatic number $\chi(G)$ of a graph $G$ is the minimum number of colors needed to color the vertices of $G$. The $l$-chromatic-number or the $l$-chromatic-index $\chi_l(G)$ of a graph $G$ is the minimum number of distinct colors needed to perform an $l$-vertex-coloring of $G$. The $l$-vertex-coloring problem or the distance-vertex-coloring problem is to compute the $l$-chromatic-index $\chi_l(G)$ of a given graph $G$. For example, Figure 1 depicts a 4-vertex-coloring of a graph $G$ using four colors, where a number next to an edge is its weight and a number next to a vertex is its color. One can easily observe that $\chi_4(G) = 4$ for $G$ in Figure 1.

Vertex coloring has diverse applications in problems such as time tabling and scheduling, frequency assignment for spectrum, register allocation in compiler, pattern matching, analysis of biological and archeological
A 4-vertex-coloring of a graph with four colors

Figure 1: A 4-vertex-coloring of a graph with four colors

data, etc. In a university we may want to assign time slots for final examinations so that two courses with a common student have different time slots. The minimum number of slots needed to schedule examinations without conflict is the chromatic number of the graph in which two courses are adjacent if they have a common student. Compiler optimization is the canonical application for coloring, where we seek to schedule the use of a finite numbers of registers. In a program fragment to be optimized, each variable has a range of times during which its value must be kept intact, in particular, after it is initialized and before its final use. Any two variables whose life spans intersect cannot be associated with the same register. Edge between any two vertices representing variables indicates that the variable life spans intersect. A coloring of the vertices of this graph assigns the variables to classes such that variables with the same color do not clash and so can be assigned to the same register.

Since the ordinary vertex coloring problem is NP-hard [6], the \( l \)-vertex-coloring problem is NP-hard in general [15]. So it is very unlikely that there exists an efficient algorithm to solve the \( l \)-vertex-coloring problem for general graphs. However, Zhou et al. presented an \( O(n^3 + n(\alpha + 1)^{2\lfloor\alpha/2\rfloor+1}) \) time algorithm to solve the \( l \)-vertex-coloring problem for partial \( k \)-trees, that is, the class of graphs of treewidth bounded by a fixed constant \( k \) [15]. Here a partial \( k \)-tree has an \( l \)-vertex-coloring: \( V \to C \), where \( C \) is a set of colors and \( |C| = \alpha \) and \( n \) is the number of vertices in \( G \). If both \( k \) and \( l \) are bounded integers, then their algorithm runs in polynomial time. Note that a tree is a partial 1-tree, whereas a series-parallel graph is a partial 2-tree. Thus putting \( k = 1 \) and \( 2 \) in their algorithm, we obtain an \( l \)-vertex-coloring algorithm for trees having time complexity of \( O(n^3 + n(\alpha + 1)^{2\lfloor\alpha/2\rfloor+2}) \) and an \( l \)-vertex-coloring algorithm for series-parallel graphs having time complexity of \( O(n^3 + n(\alpha + 1)^{2\lfloor\alpha/2\rfloor+13}) \) respectively. There is a polynomial-time 2-approximation algorithm for the \( l \)-vertex-coloring problem on planar graphs [1]. The algorithm for \( l \)-vertex-coloring of a tree \( T \), presented in [9], uses a greedy strategy to assign colors to the nodes in a post order fashion and runs in \( O(n\Delta^{\lfloor\alpha+1/2\rfloor}) \) time, where \( \Delta \) is the maximum degree of \( T \). Though this algorithm runs in linear time if both \( l \) and \( \Delta \) are bounded integers, it is not correct and does not guarantee an optimal solution. Kashem et al. showed that the \( l \)-vertex-coloring problem for series-parallel graphs can be solved in \( O(n \log \alpha) \) time [10]. The \( l \)-vertex-coloring problem on a weighted graph \( G = (V, E) \) can be easily reduced to the ordinary vertex coloring problem on a new non-weighted graph \( G_1 = (V, E_1) \) such that \( (u, v) \in E_1 \) for any two vertices \( u \) and \( v \) in \( V \) if and only if \( \text{dist}(u, v) \leq l \) in \( G \) [15]. Therefore, one may expect that the \( l \)-vertex-coloring problem for a tree can be solved by applying a linear-time algorithm to solve an ordinary vertex coloring problem for a tree [4]. However, it is not the case because \( G_1 \) obtained for a tree is not always a tree.

An edge version of the \( l \)-vertex-coloring problem has been studied for partial \( k \)-trees and planar graphs. An ordinary edge-coloring of a graph \( G \) is to color all edges of \( G \) so that any adjacent edges have different colors. For two edges \( e = (u, v) \) and \( e' = (u', v') \), the distance between \( e \) and \( e' \) in \( G \) is defined as follows, where \( \min(w, x, y, z) \) denotes the minimum of \( w, x, y \) and \( z \).

\[
\text{dist}(e, e') = \min\{\text{dist}(u, u'), \text{dist}(u, v'), \text{dist}(v, u'), \text{dist}(v, v')\}.
\]

For a given nonnegative integer \( l \), we wish to color all edges of \( G \) so that any two edges \( e \) and \( e' \) with \( \text{dist}(e, e') \leq l \) have different colors. Such a coloring is called an \( l \)-edge-coloring or a distance-edge-coloring of \( G \). Thus a 0-edge-coloring is merely an ordinary
edge-coloring, and a 1-edge-coloring is a “strong edge-coloring” [11, 13]. The \(l\)-chromatic-index \(\chi_l(G)\) of a graph \(G\) is the minimum number of distinct colors required for an \(l\)-edge-coloring of \(G\). The \(l\)-edge-coloring problem or the distance-edge-coloring problem is to compute the \(l\)-chromatic-index \(\chi_l(G)\) of a given graph \(G\). For example, Figure 2 depicts a 3-edge-coloring of a tree using five colors \(c_1, c_2, c_3, c_4,\) and \(c_5\). One can easily observe that \(\chi_3(G) = 5\) for \(G\) in Figure 2.

The edge-coloring problem arises in many applications, including various scheduling and partitioning problems [5]. Since the edge-coloring problem is \(NP\)-hard [7], the \(l\)-edge-coloring problem is \(NP\)-hard in general [8] and hence it is very unlikely that the \(l\)-edge-coloring problem can be efficiently solved for general graphs. However, the following results have been known. First, the ordinary edge-coloring problem can be solved in linear time for partial \(k\)-trees [16]. Second, the \(l\)-edge-coloring problem can be solved for partial \(k\)-trees in \(O(n(\alpha + 1)^{2^{2(\alpha + 1)+1}})\) time [13]. This algorithm takes linear time if \(k\) is a bounded integer. Putting \(k = 1\) and \(k = 2\) in their algorithm, we obtain a \(1\)-edge-coloring algorithm for trees having time complexity of \(O(n(\alpha + 1)^{2^\alpha})\) and a \(1\)-edge-coloring algorithm for series-parallel graphs having time complexity of \(O(n(\alpha + 1)^{2^\alpha})\) respectively. Third, there is an \(O(n(\alpha + 1)^{2^{2(\alpha + 1)+1}})\) time exact algorithm that determines whether a partial \(k\)-tree has an \(l\)-edge-coloring with a given number of \(\alpha\) colors [8]. This algorithm takes linear time if both \(l, k\) are bounded integers. Putting \(k = 1\) in their algorithm, we obtain an \(O(n(\alpha + 1)^{2^{2(\alpha + 1)+1}})\) time exact algorithm that determines whether a tree has an \(l\)-edge-coloring with a given number of \(\alpha\) colors. Putting \(k = 2\) in their algorithm, we obtain an \(O(n(\alpha + 1)^{2^{2(\alpha + 1)+1}})\) time exact algorithm that determines whether a series-parallel graph has an \(l\)-edge-coloring with a given number of \(\alpha\) colors. There is a polynomial-time 2-approximation algorithm for the \(l\)-edge-coloring problem on planar graphs [8]. The \(l\)-edge-coloring problem for a graph \(G\) can be reduced to an ordinary vertex coloring problem for a new graph \(G'\) obtained from \(G\) by some operations. However, \(G'\) is not always a partial \(k\)-tree or a planar graph even if \(G\) is a partial \(k\)-tree or a planar graph.

In this paper we give an \(O(n^2 + n\Delta^{l+1})\) time algorithm to find an \(l\)-vertex-coloring of a tree. The algorithm runs in \(O(n^2)\) time if both \(l\) and \(\Delta\) are bounded integers. We compute the upper bound of colors to be \(1 + \frac{\Delta^{l+1}}{(\Delta - 2)^{l+1}}\). We also present an \(O(n^2 + n\Delta^{l+1})\) time algorithm for solving the \(l\)-edge-coloring problem of trees. If both \(l\) and \(\Delta\) are bounded integers, then this algorithm also takes \(O(n^2)\) time. Early versions of this paper have been presented at [3] and [2].

The rest of the paper is organized as follows. Section 2 gives some definitions and preliminary ideas. In Section 3, we present an \(O(n^2)\) time algorithm for \(l\)-vertex-coloring of trees. In Section 4, we present an \(O(n^2)\) time algorithm for \(l\)-edge-coloring of trees. Finally, Section 5 is a conclusion.

2 Preliminaries

In this section we define several graph theoretical terms used in this paper and prove that the constraint graph for the \(l\)-vertex-coloring of a tree is a chordal graph.

Let \(G(V, E)\) be a connected simple graph with vertex set \(V(G)\) and edge set \(E(G)\). We denote by \(n\) the number of vertices in \(G\) and by \(m\) the number of edges in \(G\). Thus \(n = |V(G)|, m = |E(G)|\). An edge joining vertices \(u, v\) is denoted by \((u, v)\). The degree of a vertex \(v\) in a graph \(G\), denoted by \(d(v)\), is the number of edges incident to \(v\) in \(G\). The maximum degree of \(G\) is denoted by \(Δ\). We call a graph a weighted graph if each of the edges has a positive weight associated with it. Now if \(N\) is the set of all positive integers then we can define the weight function for the edges as \(w : E \rightarrow N\).

A walk, \(v_0, e_1, v_1, \ldots, v_{l-1}, e_l, v_l\), in a graph \(G\) is an alternating sequence of vertices and edges of \(G\), beginning and ending with a vertex, in which each edge is incident to two vertices immediately preceding and following it. If the vertices \(v_0, v_1, \ldots, v_l\) are distinct (except possibly \(v_0, v_l\)), then the walk is called a path and usually denoted either by the sequence of vertices \(v_0, \ldots, v_l\) or by the sequence of edges \(e_1, \ldots, e_l\).

The length of a path is \(l\) which is one less than the number of vertices on the path. A path or walk is closed if \(v_0 = v_l\). A closed path containing at least one edge is called a cycle. A clique in a graph is a set of pairwise adjacent vertices. The clique number of a graph \(G\), denoted by \(ω(G)\), is the maximum size of a set of pairwise adjacent vertices (clique) in \(G\). By \(N_G(v)\), we denote the set of vertices adjacent to \(v\) in \(G\).

A free tree is a connected acyclic undirected graph. We often omit the term “free” when we say that a graph is a tree. A rooted tree is a free tree in which one of the nodes is distinguished from others. This distinguished node is called the root that is drawn generally at the top. If a rooted tree is regarded as a directed graph in which each edge is directed from top to bottom, then every node \(u\) other than the root is connected by an edge from some other node \(p\), called the parent of \(u\). We also call \(u\) a child of \(p\). A leaf is a node of a tree that has no child. An internal node is a node that has one or more children. Every node of a tree is either a leaf or
an internal node. Consider a node \( x \) in a rooted tree \( T \) with root \( r \). Any node \( y \) on the unique path from \( r \) to \( x \) is called an \emph{ancestor} of \( x \). If \( y \) is an ancestor of \( x \), then \( x \) is a \emph{descendent} of \( y \). The subtree rooted at \( x \) is the tree induced by descendents of \( x \). The depth of a node \( x \) in a tree is the length of the path from the root to \( x \).

Let \( l \) be a positive integer, \( C \) be a set of colors and the number of colors used by an \( l \)-vertex-coloring of a tree \( T \) is denoted by \( \#\phi \). Then a function \( \phi : V \to C \) is an \( l \)-vertex-coloring of \( T \) if \( \phi(u) \neq \phi(v) \) for any two vertices \( u \) and \( v \) such that \( \text{dist}(u, v) \leq l \). Clearly \( \chi(T) \leq \#\phi \). We can assume without loss of generality that the consecutive integers \( 1, 2, \ldots, \#\phi \) are used as the colors. Then \( C \) is the color set having colors \( 1, 2, \ldots, \#\phi \). The \emph{greedy coloring} relative to a vertex ordering \( v_1, v_2, \ldots, v_n \) of \( V(G) \) is obtained by coloring vertices in the order \( v_1, v_2, \ldots, v_n \) assigning to \( v_i \) the smallest indexed color not already used on its lower-indexed neighbors.

A \emph{chord} of a cycle \( C \) is an edge, not in \( C \), whose end-points lie in \( C \). A \emph{chordless cycle} in a graph \( G \) is a cycle of length at least four in \( G \) that has no chord (that is the cycle is an induced subgraph). A graph \( G \) is \emph{chordal} if it is simple and has no chordless cycle. A graph \( G \) is \emph{perfect} if the chromatic number of every induced subgraph equals the size of the largest clique of that subgraph, i.e. \( \chi(H) = \omega(H) \) for every induced subgraph \( H \subseteq G \).

A vertex of a graph \( G \) is \emph{simplicial} if its neighborhood in \( G \) is a clique. A \emph{simplicial elimination ordering} is an ordering \( v_1, \ldots, v_n \) for deletion of vertices so that each vertex \( v_i \) is a simplicial vertex of the remaining graph induced by \( \{v_1, \ldots, v_i\} \). These orderings are called \emph{perfect elimination ordering}. We say that the vertex ordering \( \delta = (v_1, v_2, \ldots, v_n) \) is a \emph{maximum cardinality ordering} if for every \( i \in \{1, 2, \ldots, n - 1\} \) and \( j \in \{1, \ldots, i\} \), \( |N_G(v_i) \cap \{v_{i+1}, \ldots, v_n\}| \geq |N_G(v_j) \cap \{v_{i+1}, \ldots, v_n\}| \). In Figure 3 \( a, b, c, d, e, f \) is a maximum cardinality ordering of the vertices of \( G \).

![Figure 3: Maximum cardinality ordering.](image)

We call \( G_c = (V_c, E_c) \) the constraint graph for the \( l \)-vertex-coloring of a tree \( T = (V, E) \), if \( V_c = V \) and for any two vertices \( u \) and \( v \) with \( \text{dist}(u, v) \leq l \) in \( T \), there is an edge \( (u, v) \in E_c \). Let \( C = < v_1, v_2, \ldots, v_n, v_1 > \) be a cycle in \( G_c \) and \( p_m = < v_1, v_{i+1}, \ldots, v_k > \) be a path in \( C \) such that the path between \( v_1 \) and \( v_k \) in \( T \) also includes the vertices \( v_{i+1}, \ldots, v_{n-1} \) in the same order. Then we call \( p_m \) an \emph{original path}. If \( p_m \) is not a subpath of any original path in \( C \), we call \( p_m \) a \emph{maximal original path} in \( C \). For every original path \( p_m = < v_1, v_{i+1}, \ldots, v_k > \) in \( C = < v_1, v_2, \ldots, v_n, v_1 > \), we term the path \( < v_{i+1}, v_{i+2}, \ldots, v_n > \) as the return path \( p_r \). We have the following lemma and theorem.

**Lemma 2.1** The \( l \)-vertex-coloring problem for a tree \( T \) reduces to the ordinary vertex coloring problem for the constraint graph \( G_c \).

**Theorem 2.2** The constraint graph \( G_c \) for the \( l \)-vertex-coloring of a tree \( T \) is a chordal graph.

**Proof.** Let \( C = < v_1, v_2, \ldots, v_n, v_1 > \) be a cycle in \( G_c \), \( p_m \) be a maximal original path and \( p_r \) be its return path in \( C \). Without any loss of generality we can order the vertices in \( C \) so that \( p_m = < v_1, v_2, \ldots, v_k > \) and \( p_r = < v_{k+1}, v_{k+2}, \ldots, v_n > \). In the following figures the vertices of \( p_m \) are shown as shaded circles. If we can show that there exists at least one chord in \( C \), we shall be able to conclude that as long as the length of \( C \) remains greater than three, a chord will be found and the lengths of the newly formed cycles will be less than the original cycle. Eventually there will be no chordless cycle of length greater than three. According to the definition of maximal original path, \( p_m \) has at least two vertices. We have the following cases to consider.

**Case 1:** \( p_m \) has exactly two vertices and \( p_r \) has at least two vertices.

In this case \( p_m = < v_1, v_2 > \) and \( p_r = < v_3, \ldots, v_n > \). There are two possible subcases.

**Case 1a:** The original path between \( v_3 \) and \( v_n \) does not have any common subpath with the original path between \( v_1 \) and \( v_2 \) (see Figure 4). If the original path between \( v_3 \) and \( v_n \) shares only one vertex in common with the original path between \( v_1 \) and \( v_2 \), \( c \) will be 0.

![Figure 4: The original path does not have a common subpath](image)
and that will not affect the following proof. Now if $C$ is a chordless cycle, there could be no edges between $v_2$ and $v_n$ or between $v_1$ and $v_3$. So the following two conditions must hold.

$$a + c + e > l$$  \hspace{1cm} (1)

$$b + c + d > l$$  \hspace{1cm} (2)

But for the presence of edges $(v_2, v_3)$ and $(v_1, v_n)$ in $C$,

$$a + c + d \leq l$$  \hspace{1cm} (3)

$$b + c + e \leq l$$  \hspace{1cm} (4)

From equation 1 and 4 we get $a > b$. But from equation 2 and 3 we get $a < b$. So the conditions contradict. So either $\text{dist}(v_1, v_3) \leq l$ or $\text{dist}(v_2, v_n) \leq l$. Hence there exists at least one chord in $C$.

**Case 2:** $p_m$ has three or more vertices and $p_r$ has one or more vertex.

In this case $p_m = <v_1, v_2, \ldots, v_k, \emptyset_p = <v_{k+1}, \ldots, v_n>$ and $k > 2$. The path $p_m$ from $v_1$ to $v_k$ in $G_c$ indicates that the path between $v_1$ and $v_k$ in $T$ must also go through every vertex in $p_m$. So to make a cycle, the path $<v_{k+1}, v_{k+2}, \ldots, v_n, v_1>$ in $G_c$ must take a different path in the original tree other than the shortest path between $v_1$ and $v_k$ as covered by $p_m$. Now, for a tree, there exists exactly one simple path between each pair of vertices. So $p_r$ must have gone along a complex path and this path must visit all the vertices lying in the original path between $v_1$ and $v_k$. So there exists at least one edge $e_j$ between $v_j$ and $v_{j+1}, j \in \{k + 1, \ldots, n\}$ in $p_r$ for every $v_i, i \in \{2, \ldots, k - 1\}$ in $p_m$ such that $v_i$ lies in the original path between $v_j$ and $v_{j+1}$. This along with the presence of $e_j$ implies that the distance between $v_j$ and $v_{j+1}$ and $v_i$ is not more than $l$. Hence there must be a chord in $C$.

**Case 1b:** The original path between $v_3$ and $v_n$ shares a common subpath $v_x, \ldots, v_y$ with the original path between $v_1$ and $v_2$ (see Figure 5). If any vertex in the return path $p_r$ lies in the common subpath, then it eventually falls in the actual path between $v_1$ and $v_2$ resulting the existence of two chords between that vertex and $v_1$ and $v_2$. So let no vertex in $p_r$ lies in the original path between $v_1$ and $v_2$. Now, as the original path between $v_3$ and $v_n$ has some part lying in $p_m$, and no vertices of $p_r$ are allowed to be lying on the common subpath either, there exists at least one edge $(v_j, v_{j+1})$ in $p_r$ such that the original path between $v_j$ and $v_{j+1}$ includes the common subpath.

To be chordless, $C$ cannot have any edge either between $v_2$ and $v_{j+1}$ or between $v_1$ and $v_j$. So at least the following two conditions must hold.

$$a + b + e > l$$  \hspace{1cm} (5)

$$b + c + d > l$$  \hspace{1cm} (6)

But for the presence of edges $(v_1, v_2)$ and $(v_j, v_{j+1})$ in $C$,

$$a + b + c \leq l$$  \hspace{1cm} (7)

$$b + d + e \leq l$$  \hspace{1cm} (8)

From equation 5 and 7 we get $e > c$. But from equation 6 and 8 we get $c > e$. So the conditions contradict. So either $\text{dist}(v_1, v_j) \leq l$ or $\text{dist}(v_2, v_{j+1}) \leq l$. Hence there exists at least one chord in $C$.

**Case 3:** $p_m$ has four or more vertices and $p_r$ has no vertex.

In this case, $p_m = <v_1, v_2, \ldots, v_k>$, $\emptyset_p = \phi$ and $k > 3$. This is the simplest of all cases because the edge $(v_k, v_1)$ in $G_c$ implies $\text{dist}(v_k, v_1) \leq l$. But from the definition of maximal original path, all the vertices of $p_m$ must lie in the original path between $v_1$ and $v_k$. So $\text{dist}(v_k, v_1) \leq l$ implies $\text{dist}(v_i, v_j) \leq l$ for all $i \in \{2, \ldots, k - 1\}$. Each of these inequalities induces a chord in $C$. Q.E.D.

In Section 3, we provide an $O(n^2)$ time algorithm for solving the $l$-vertex-coloring problem on trees.
3 \textit{l-vertex-coloring}

In this section we present an $O(n^2 + n\Delta^{l+1})$ time algorithm for solving the \textit{l-vertex-coloring} problem on trees.

Given a tree $T$, we first transform $T$ into its constraint graph $G_c$, then find maximum cardinality ordering of the vertices of $G_c$ and perform greedy coloring in the reverse of the obtained perfect elimination ordering to obtain the required \textit{l-vertex-coloring}. The following algorithm \text{Generate\_Constraint\_Graph} generates the constraint graph for the \textit{l-vertex-coloring} of a tree.

\textbf{Algorithm 1: Generate\_Constraint\_Graph}
\begin{quote}
\textbf{Input}: A weighted tree $T = (V, E)$ and $l$.
\textbf{Output}: A constraint graph $G_c = (V_c, E_c)$ for the \textit{l-vertex-coloring} of a tree $T$.
\begin{algorithmic}
\State $G_c \leftarrow T$
\For{each vertex $u \in V$}
\State \text{BFS-\text{Traverse}}(T, u, l)
\EndFor
\end{algorithmic}
\end{quote}

\textbf{Procedure BFS-\text{Traverse}}($G, s, l$)
\begin{quote}
\begin{algorithmic}
\State \textbf{begin}
\For{each vertex $u \in V[G]$}
\State $d[u] \leftarrow \infty$
\State $d[s] \leftarrow 0$
\State $Q \leftarrow \{s\}$
\While{$Q \neq \emptyset$}
\State $u \leftarrow \text{head}(Q)$
\For{each vertex $v \in \text{Adj}[u]$}
\If{$\text{color}[v] = \text{white}$}
\State $d[v] \leftarrow d[u] + w(u, v)$
\EndIf
\If{$d[v] \leq l$}
\State $\text{Enqueue}(Q, v)$
\EndIf
\EndFor
\If{$v \notin \text{Adj}[s]$}
\State $E_c \leftarrow E_c \cup \{(s, v)\}$
\EndIf
\State $\text{Dequeue}(Q)$
\State $\text{color}[u] \leftarrow \text{black}$
\EndWhile
\State \textbf{end}
\end{algorithmic}
\end{quote}

We have the following lemmas and theorem.

\textbf{Lemma 3.1} Let $\chi_i(T)$ be the $i$-chromatic-index of a tree $T$ with maximum degree $\Delta$. Then
\[
\chi_i(T) \leq 1 + \Delta \frac{(\Delta - 1)^{\lfloor l/2 \rfloor - 1}}{(\Delta - 2)}.
\]

\textbf{Proof.} Since the \textit{l-vertex-coloring} problem for a tree $T$ reduces to the ordinary vertex coloring for the chordal graph $G_c$, $\chi(Gc)$ will be the same as $\chi_i(T)$. Now a chordal graph is a perfect graph and for a perfect graph $G$, $\omega(G) = \chi(G)$. So to determine the upper bound of $\chi_i(T)$, we have to know the size of the maximum clique in $G_c$. We need to know how many vertices could be at maximum such that any two of them remain at max $l$ distance apart in $T$. We assume the weights of each edges as minimum which is one.

We can add vertices to the set constituting the clique until we could add no more vertices keeping it within $l$ distance from all the members of the clique. Now if we start with the vertex $v_i$ and add vertices accordingly, we can add vertices to the set till all the leaves of the sub-tree rooted at $v_i$ are within $l$ distance from $v_i$. The depth of the sub-tree could be no more than $\lceil l/2 \rceil$ as any two leaves must be within $l$ distance from each other and the distance between them is twice the depth of the sub-tree. The number of vertices could be found by adding up the following series, where the $i$-th term denotes the number of vertices in $(i + 1)$-th level.

\[
\chi_i(T) = \chi(Gc) = 1 + \Delta + \Delta(\Delta - 1) + \Delta(\Delta - 1)^2 + \ldots + \Delta(\Delta - 1)^{\lfloor l/2 \rfloor - 1} = 1 + \Delta \frac{(\Delta - 1)^{\lfloor l/2 \rfloor - 1}}{(\Delta - 2)}\]

\textbf{Q.E.D.}

\textbf{Lemma 3.2} The constraint graph $G_c = (V_c, E_c)$ has $O(n\Delta^l)$ edges.

\textbf{Proof.} We have to add edges between all the vertices that are at most $l$ distance apart in $T$. For each vertex the number of vertices within $l$ distance is found by the exact analogy used in lemma 3.1. The same edge is considered twice if the edges for all the vertices are taken separately. Hence
\[
|E_c| = \frac{n \Delta^l}{2} \frac{\Delta + \Delta^2 + \ldots + \Delta^l}{(\Delta - 1)^{\lfloor l/2 \rfloor} - 1} = O(n\Delta^l).
\]

\textbf{Q.E.D.}

\textbf{Lemma 3.3} The constraint graph $G_c$ can be generated in $O(n^2 + n\Delta^{l+1})$ time.

\textbf{Proof.} Algorithm \text{Generate\_Constraint\_Graph} exhaustively adds edges to each pair of vertices $u$ and $v$ with $\text{dist}(u, v) \leq l$ in $T$. The BFS-\text{Traverse} part of the algorithm traverses the subtree rooted at any vertex $u$, up to $l$ level deep at maximum when each edge has a minimum weight of one. The calculation for number of vertices covered in each such subtree is similar to the number of vertices obtained in lemma 3.1 with only going $l$ level deep rather than halting at level $\frac{l}{2}$.
So maximum number of vertices to traverse for BFS-Traversal is $1 + \Delta (\Delta - 1)/(\Delta - 2)$ = $O(\Delta^2)$. From lemma 3.2 the maximum number of edges for each vertex in $G_c$ is $O(\Delta^2)$. Since the adjacency list of each vertex is scanned at most once, at most $O(\Delta^2)$ time is spent in total scanning adjacency lists. The overhead for initialization is $O(n)$, and thus the total running time of BFS-Traversal is $O(n) + O(\Delta^2) + O(\Delta^2) = O(n + \Delta^2)$. Hence the run time of Generate_Constraint_Graph would be $O(n^2 + n\Delta^2)$.

Lemma 3.6 If greedy coloring is applied in the reverse perfect elimination ordering of a chordal graph, the coloring will be optimal.

Proof. We prove it by induction on the number of vertices colored.

Base case: $v_n$ can arbitrarily be given color 1.

Induction step: Assume that $v_n, v_{n-1}, \ldots, v_{i+1}$ have been colored optimally. We want to show that choosing the color for $v_i$ in a greedy manner will yield an optimal coloring for $v_i, v_{n-1}, \ldots, v_{i+1}, v_n$. We choose an arbitrary color for $v_i$ that has not been used by any of its neighbors. If any color that has already been used is available, we use that for $v_i$ keeping the number of colors as before. Otherwise, if all the already colored neighbors of $v_i$ form a clique, there is no way they (including $v_i$) could be colored by any fewer colors than the size of this clique. If there is no other color than the colors used by the neighbors, we have no choice but to use a new color for $v_i$ and this coloring is optimal. Since the coloring was optimal for $v_n, v_{n-1}, \ldots, v_{i+1}$, it must also be optimal for the subgraph $v_n, v_{n-1}, \ldots, v_{i+1}, v_i$ and thus we have a new optimal sub-solution. This completes the inductive step.

We have the following lemmas and theorem.

Lemma 3.4 Algorithm MCS can be implemented in time $O(n\Delta^2)$.

Proof. To implement MCS in linear time, we define $S_i$ to be the set of vertices with label $i$. Then we present every set by a list. Also for each vertex we store its label $i$ and pointer to its position in $S_i$. When a vertex $v$ receives a number, we remove it from the corresponding list and move each of its neighbors one list upwards. This takes $O(1 + |Adj[v]|)$ time. So MCS can be implemented in $O(|V_c| + |E_c|) = O(n\Delta^2)$ time.

Lemma 3.5 A graph is chordal if and only if every ordering obtained by Algorithm MCS is a perfect elimination ordering [12].

Algorithm 2: MCS

Input: A chordal graph $G_c = (V_c, E_c)$.
Output: A maximum cardinality ordering of $V_c$.

begin
    currentOrder ← |V_c|
    currentLabel ← 1
    for each vertex $v \in V_c$ do
        label[v] ← 0
    end
    for $i = 1$ to |V_c| do
        pick an unordered vertex $u$ with the highest label
        order[u] ← currentOrder
        for each vertex $w \in Adj[u]$ do
            label[w] ← currentLabel
            $E_c \leftarrow E_c \setminus \{(u, w)\}$
        end
        $V_c \leftarrow V_c \setminus \{u\}$
        currentOrder ← currentOrder - 1
        currentLabel ← currentLabel + 1
    end

end

We first transform $T$ into its constraint graph $G_c$. Then we run the MCS algorithm on $G_c$, which is a chordal graph and greedy coloring is applied in the reverse of the obtained perfect elimination ordering to get the required l-vertex-coloring. The simple greedy-coloring algorithm can be implemented in $O(n\Delta^2)$ time. Thus from lemma 3.3 and 3.4, the total running time is $O(n^2 + n\Delta^2) + O(n\Delta^2) + O(n\Delta^2) = O(n^2 + n\Delta^2)$. Hence if both $l$ and $\Delta$ are bounded integers, then our algorithm runs in $O(n^2)$ time.

Proof. At first the input tree $T$ is transformed into constraint graph $G_c$. Then we run the MCS algorithm on $G_c$ which is a chordal graph and greedy coloring is applied in the reverse of the obtained perfect elimination ordering to get the required l-vertex-coloring. The simple greedy-coloring algorithm can be implemented in $O(n\Delta^2)$ time. Thus from lemma 3.3 and 3.4, the total running time is $O(n^2 + n\Delta^2) + O(n\Delta^2) + O(n\Delta^2) = O(n^2 + n\Delta^2)$. Hence if both $l$ and $\Delta$ are bounded integers, then our algorithm runs in $O(n^2)$ time.

Theorem 3.7 The l-vertex-coloring of a tree $T$ can be solved in time $O(n^2 + n\Delta^2)$, where $l$ is a positive integer and $\Delta$ is the maximum degree of $T$. If both $l$ and $\Delta$ are bounded integers, then an l-vertex-coloring of $T$ can be found in $O(n^2)$ time.
In this section we present an $O(n^2 + n\Delta^{l+1})$-edge-coloring algorithm for solving the $l$-edge-coloring problem on trees.

The constraint graph $G_c' = (V_c', E_c')$ for the $l$-edge-coloring of a tree $T = (V,E)$ is the graph whose vertices are the edges of $T$, with $(e,f) \in E_c'$ when $e = (u,v), f = (u',v')$ and $\text{dist}(e,f) \leq l$ in $T$. We have the following lemmas and theorem.

**Lemma 4.1** The $l$-edge-coloring problem for a tree $T$ reduces to the ordinary vertex coloring problem for the constraint graph $G_c'$.

**Theorem 4.2** The constraint graph $G_c'$ for the $l$-edge-coloring of a tree $T$ is a chordal graph.

**Proof.** The proof is similar to that of Theorem 2.2.

We first transform $T$ into its constraint graph $G_c'$. Figure 9(b) shows the constraint graph $G_c'$ for the $l$-edge-coloring of the input weighted tree $T$ in Figure 9(a) with $l = 3$. Then using Algorithm MCS, we find
the maximum cardinality ordering of the vertices of $G'$. Figure 10(a) shows the maximum cardinality ordering of the vertices of $G'$, where a number next to a vertex is its order. Then we perform greedy coloring in the reverse of the obtained perfect elimination ordering in Figure 10(a). Figure 10(b) shows the greedy coloring, where a number next to a vertex is its color. Hence Figure 11 shows the required $l$-edge-coloring using five colors $c_1, c_2, c_3, c_4$, and $c_5$.

5 Conclusion
In this paper we present an $O(n^2 + n\Delta^{l+1})$ time algorithm for solving the $l$-vertex-coloring problem on trees. If both $l$ and $\Delta$ are bounded integers, then our algorithm takes $O(n^2)$ time. We compute the upper bound of colors to be $1 + \Delta \left(\frac{\Delta-1}{\Delta-2}\right)^{l+1}$. We also present an $O(n^2 + n\Delta^{l+1})$ time algorithm for solving the $l$-edge-coloring problem on trees. If both $l$ and $\Delta$ are bounded integers, then this algorithm also takes $O(n^2)$ time.

References
1. INFOCOMP publishes original scientific and technological papers in Portuguese and English. The papers should be related to Computer Science.

2. The papers should be submitted in PDF format, without the author’s informations, by the JEMS system (https://submissoes.sbc.org.br/infocomp) of the Brazilian Computer Society (SBC). Login and password are necessary, but it can be obtained on-line by new users of the JEMS system.

3. The papers should be in Paper Letter (21.5 x 28.0 cm), space 1.1 between lines (or at least 13 pts), font Times New Roman 10, justified text, with superior, inferior, left and right margins of 2.5 cm. The number of pages should not exceed 10. Papers with more than 10 pages may be accepted after an analysis by the editorial board.

4. The first page should contain TITLE, AUTHORS (only in the final version), ABSTRACT and Keywords. After, it should contain the following text centralized: “(Received January 01, 2005 / Accepted December 31, 2005)”, for posterior edition with the correct dates. These informations should not exceed 01 page and should be in 01 column. The TITLE should be in font size 14.

5. The AUTHOR’S names (only in the final and accepted version) should be sided horizontally, identified by a superscripted number. The affiliation and the email of each author should be written right below the names.

6. The text of the paper should be formatted in 02 columns, separated by 0.5cm, with numerated sections and subsections. Ex: 1. Introduction, 1.1. Terminology. Figures and tables may occupy the page width, if necessary. The titles of figures and tables should be numerated and centralized below.

7. The references should be numerated and listed in alphabetical order. Ex: [5] Hougardy, S. Even pairs and the strong perfect graph conjecture, Discrete Math. v.154, p.277-288, 1996. Citations in the text should be made based on the number of the reference. Ex, “In [5], it was proved that...”.

8. Footnotes may be accepted, when strongly necessary, for explanations that cannot be included in the text, like: (a) name of the research institution; (b) support organizations and other financial supports; (c) reference to the publication as part of MSc or PhD thesis ; (d) personal communication. Citations of footnotes in the text should be made based on the superscripted number.

9. The authors retain author’s rights for future publications, but the journal is allowed to edit and reproduce its papers.

10. Non compliance with these norms will result in the non acceptance of the paper.