Graph Algorithms and Their Applications

Thesis
submitted for the partial fulfillment of
the requirements of the M.Sc. degree
in Computer Science.

Presented by:

Ahmed Mohammed Hassan Abd El-Fattah
B.Sc. Pure Math. & Computer Science.

Supervised by:

Prof. Dr.
Mohamed Hamed El-Zahar
Department of Mathematics
Faculty of Science, Ain Shams University.

Asst. Prof.
Fayed Fayek M. Ghaleb
Department of Mathematics
Faculty of Science, Ain Shams University.

Asst. Prof.
Soheir Mohamed Khamis
Department of Mathematics
Faculty of Science, Ain Shams University.

Submitted to:
Department of Mathematics,
Faculty of Science,
Ain Shams University,
Cairo - Egypt.
2005.

Contents

Acknowledgements

All my praises and gratitude to $\mathcal{ALLAH}$, the most merciful, the most gracious, who gave me everything. Without the mercy and guidance of $\mathcal{ALLAH}$, this work could never be completed.

I'm very grateful to *Prof Dr. Mohamed H. El-Zahar*, my leader, for his great ideas that helped me during the preparation of the thesis. Since the start, he guided me in the selection of the topic and the related points. He supported me in a direct way through advises and corrections; and in an indirect way through encouragement, and by providing the most important references used.

Many great thanks to *Asst. Prof. Fayed Ghaleb* for his supervision, interest, and valuable criticism. His great experience played a very important role in completing the thesis in its final shape. He made very accurate corrections, and put the final touches.

I express my deep sense of sincere gratitude to *Asst. Prof. Sohier M. Khamis* for her infinite patience, intelligence, and hard working. Her moral support made me overcome all challenges I met in order to complete the work. She also supported me with all resources I needed, including interesting conversations.

Finally, and with great proud, I'd like to dedicate this work to my parents. It is one of the important moments in life to give them a great amount of thanks, still not sufficient compared to what they really deserve.

Summary

The combinatorial optimization algorithms are fundamental for solving hard optimization problems in many fields, especially that appear in Computer Science. However, the thesis focuses on graph algorithms and their applications because these algorithms have a great importance and have a variety of practical applications in Computer Science.

In this thesis we present an extensive study of applying the very recent technique of ant colony optimization to the hard optimization problem of graph coloring. We give a detailed description of the well-known $\mathcal{ANTCOL}$ algorithm that develops an ant algorithm for solving the graph coloring problem. Then we illustrate our modification to the original $\mathcal{ANTCOL}$ algorithm. Practical results that have been obtained by applying these algorithms are given and analyzed. Our results show notable improvements over that of the original ones.

The thesis consists of five chapters and an appendix.

Chapter One includes the terminologies in graph theory and in algorithmics, that are needed throughout the rest of the thesis. Also, the complexity measures of hard problems are discussed.

Chapter Two contains the formal definitions of optimization problems. The chapter includes a survey about some applications of different optimization problems. Also, a discussion about how hard optimization problems are dealt with in practice is given.

In the third chapter, definitions of approximation algorithms and heuristics are introduced. The properties of such algorithms are discussed. Also, examples of these algorithms are demonstrated in some details.

In the fourth chapter, a detailed introduction to the ant colony optimization concepts is given. Also, a heuristic algorithm based on these concepts is presented to solve the hard optimization problem of the Traveling Salesperson. In addition, a general method for solving many other optimization problems using these concepts is explained.

In Chapter Five, the graph coloring problem is studied extensively. A heuristic and an exact algorithm for solving

this problem are presented. The chapter includes a detailed description of an ant colony algorithm for solving the coloring problem with our modifications to it. Comparisons between the practical results obtained by implementing the original and the modified algorithms are discussed.

In the Appendix, we first explain how the presented graph algorithms may be implemented, then we give the script codes of our implementations using the programming language C++ and the LEDA® package. Finally, we present a conclusion that summarizes the obtained results and points out some prospectives for future works.

Chapter 1

Basic Definitions

This chapter gives an introduction to graph theory. The definitions needed in the next chapters, such as algorithms and complexity, are also introduced.

1.1 Graph terminologies

This section gives some terminologies in graph theory. For these and other terms, we refer to [13] and [19].

Graphs

A *graph* G is a pair $G = (V, E)$ consisting of a finite set $V \neq \emptyset$, called the *vertex set* of G, and a set E of two-element subsets of V called the *edge set* of G. An element $v \in V$ is called a *vertex* (or *node*), while an element $e = \{u, v\} \in E$ is called an *edge* with *end vertices* u and $v, u \neq v$. If $e = \{u, v\}$ is an edge, then the vertices u and v are said to be *incident* with the edge e in G, also u and v are called *adjacent* in

G, and we write $e = uv$. The notations $V(G)$ and $E(G)$ can be used to denote the vertex set and the edge set of a graph G, respectively.

Graphs are often illustrated by drawing pictures in the plane. The vertices of a graph G are represented by (bold) points, and edges are represented by (curved or straight) lines joining the end vertices. A graph G is called a *planar graph* if it can be drawn in the plane in such a way that no two edges (or rather, the curves representing them) intersect, except possibly at a common end vertex.

For any vertex v of a graph $G = (V, E)$, the *degree* of v in G, $deg_G(v)$, is the number of edges incident with v in G. If all vertices of G have the same degree r, then G is called a *regular graph of degree* r, or an r*-regular* graph.
There are some important series of examples of graphs:

- The *null graph* N_n, which has n vertices and no edges. The null graph is also called *trivial*.
- The *complete graph* K_n, which has n vertices and every two distinct vertices are adjacent.
- The *complete bipartite graph* $K_{m,n}$, which has as vertex set the disjoint union of two sets V_1 and V_2, such that V_1 has m vertices and V_2 has n vertices. The edge set is $E(K_{m,n}) = \{uv |\, u \in V_1,\ v \in V_2\}$.

The *complement of a graph* $G = (V, E)$ is the graph $\overline{G} = (V, \overline{E})$, where $\overline{E} = \{uv \notin E |\ u, v \in V, u \neq v\}$. For example, $N_n = \overline{K_n}$.

Subgraphs and cliques

Let $G = (V, E)$ be a graph, and $V' \subseteq V$. Denote the set of edges e, which have both their end vertices in V', by $E|V'$. The graph $(V', E|V')$ is called the *induced subgraph* on V' and is denoted by $G|V'$. Any graph $G' = (V', E')$, where $V' \subseteq V$ and $E' \subseteq E|V'$, is called a *subgraph* of G. A subgraph with $V' = V$ is called a *spanning subgraph* of G. A *clique* in a graph $G = (V, E)$ is a subset $C \subseteq V$, where any two vertices in C are adjacent in G. An *independent set* (or *stable set*) in G is a subset $I \subseteq V$, where no two vertices in I are adjacent in G. The maximal cardinality of a clique in a graph G is called the *clique number* of G, and is denoted by $\omega(G)$.

Paths and cycles

Let $G = (V, E)$ be a graph. A *path* P in G is a sequence of distinct vertices $(v_0, \ldots, v_k)$, $v_i \in V$ for $i = 0, \ldots, k$, where the edges of P are $e_i = v_{i-1}v_i \in E$ for $i = 1, \ldots, k$. The vertex v_0 is called the *start vertex* of P, and v_k is called the *end vertex* of P. The *length* of the path is the number of its edges. A *trivial path* is a path of length 0. If P is not trivial then it can also be written as a sequence of its edges

$(e_1, \ldots, e_k)$. If P has the start vertex u and the end vertex v then it is called a u-v path, and if $u = v$ then P is called a *cycle*. A graph is called *acyclic* if it does not contain a cycle. An acyclic graph is referred to as a *forest*.

A path that contains all the vertices of a given graph G is called a *Hamiltonian path* of G. A graph that contains a Hamiltonian cycle is called *Hamiltonian*.

Connected graphs

In a graph $G = (V, E)$, two vertices $u, v \in V$ are called *connected* if there exists a u-v path. If any two vertices of G are connected, then G is called a *connected graph*, otherwise, G is *disconnected*.

A *tree* is a connected acyclic graph. A tree $T = (V, E')$ is a *spanning tree* of $G = (V, E)$, where $E' \subseteq E$. Obviously, connectedness is an equivalence relation on the vertex set of a graph. The equivalence classes of this relation are called the *connected components* of the graph, or simply, the components of the graph. If a connected component contains only one vertex, then that vertex is called *isolated vertex*.

Let u and v be two vertices in the same component of a graph $G = (V, E)$, then the *distance* $d(u, v)$ between them is the minimum length of all u-v paths.

Directed graphs

A *directed graph* G is a pair $G = (V, E)$, where $V \neq \emptyset$ is the vertex set, and $E \subseteq \{(u, v) |\ u \neq v, u, v \in V\}$ is the set of directed edges or *arcs*. An arc $(u, v) \in E$, is said to leave u and enter v. An arc $e = (u, v)$ may also be written as $e = \overrightarrow{uv}$.

Weighted graphs

Let $G = (V, E)$ be a graph or a directed graph on which a mapping $w : E \to \mathbb{R}$ is defined, then G is called a *weighted graph*. The pair $\langle G, w \rangle$ is called a *network*. The number $w(e)$ is called the *weight* of the edge $e \in E$. The weight of any subset of edges $T \subseteq E$ is equal to the sum of the weights of the edges $e \in T$, that is

$$w(T) = \sum_{e \in T} w(e).$$

In particular, $w(P) = w(e_1) + w(e_2) + \ldots + w(e_k)$ is the weight of the path $P = (e_1, e_2, \ldots, e_k)$.

Graph coloring

A *coloring* of a graph $G = (V, E)$ is a mapping $\mathcal{C}$ from the vertex set of G to the set of positive integers $\{1, \ldots, |V|\}$. The mapping $\mathcal{C}$ assigns for each vertex $v \in V$ an integer

value $\mathcal{C}(v)$, called the color of the vertex v, such that any two adjacent vertices are mapped to different values. That is $\mathcal{C}(u) \neq \mathcal{C}(v)$, $\forall uv \in E$. A *k-coloring* of G is a coloring that maps V into the set $\{1, \ldots, k\}$, where $k \leq |V|$.

If there exists a k-coloring for the graph G, then G is said to be *k-colorable*. The *chromatic number* of the graph G, $\chi(G)$, is the minimal number k for which G is k-colorable. If $\chi(G) = k$ then G is said to be *k-chromatic*. An *optimal coloring* of G is a coloring that uses exactly $\chi(G)$ colors.

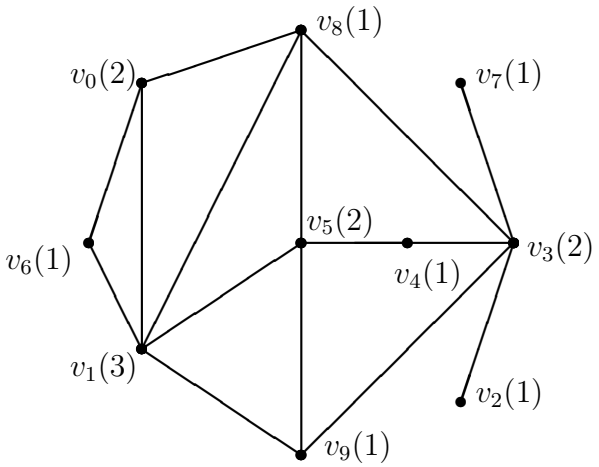

Figure 1.1: An optimal coloring for a graph. The colors are represented by the numbers between parentheses.

It is clear that the set of vertices having the same color is independent. A k-coloring of a graph $G = (V, E)$ gives a partition of the vertex set V into k independent sets of vertices. In Figure 1.1, a 3-coloring of a graph having 10

vertices is given. This coloring is optimal, hence G is 3-chromatic.

Random graphs

Generally, a *random graph* is a graph in which the edges are placed between pairs of vertices chosen uniformly at random. A random graph $G_{n,p}$ has n vertices such that each possible edge between any two vertices is present with a fixed probability p and absent with probability $1 - p$. The existence of one edge is independent of the existence of another.

1.2 Fundamentals of algorithmics

In this section, the algorithmic terminologies, notations, and basic concepts which will be used in all chapters that follow are recalled, [12, 17, 19, 21].

1.2.1 Problems and algorithms

A *problem* is a general question to be answered and may have several parameters. An answer to the problem is called a *solution*. A problem is specified by giving a general description of all its parameters, and a statement of what properties the solution is required to satisfy. An *instance* of a problem is obtained by specifying particular values for