Coloring Algorithm- Voice and Animation using GraphPack and a Parallel Heuristic

Mukkai S. Krishnamoorthy, Alex Robayo, Katsumi Koyanagi
Department of Computer Science
Rensselaer Polytechnic Institute
Troy, NY 12180

January 14, 1993

Extended Abstract

1 Introduction

GraphPack is an application developed at RPI that provides users with the capability to study and visualize graph theory algorithms. The graphical user interface is X windows. This application can manipulate digraphs and graphs and also draw various types of graphs. GraphPack itself is composed of two components: the kernel and Xgraphpack. The kernel, which has an interpreter, accepts LiLa (Little Language) commands from the user and communicates with Xgraphpack. Xgraphpack is an X Window program that draws graphs and responds to certain commands (i.e., clicking on pre-defined buttons).

Presently, GraphPack contains a feature that provides data communication between itself and other related applications. These related applications include Maple, Mathematica, Matlab, and SetPlayer. From a computer networks approach, this feature allows GraphPack to be a client and the other application to be a server. In other words, GraphPack is able to send requests (commands) to another applications and the application services the request by returning a result. The aim of this paper is two-fold:

1. To make GraphPack act as a server. This will grant users flexibility when interfacing with GraphPack by writing a multimedia animation program for coloring. This is accomplished by the ability to send commands to GraphPack without setting up the communication mechanism and with minimal knowledge of LiLa.

2. To write a parallel coloring heuristic algorithm and test it on various classes of graphs.

2 Overview

Sound and Visual Animation of a Coloring Algorithm: It is desirable in certain applications for GraphPack to respond like a server. For instance, suppose a user writes a 'C' program that does some computation on a graph. If the user wants to incorporate GraphPack into his/her program in order to more intuitively analyze the results, then it would be beneficial if the user could send commands to GraphPack. These commands might manipulate a graph or draw the resulting graph. We support this feature by function calls from a "C" or "C++" source code. These function calls are developed as a part of our project. Additionally, this system is able to set up the communication mechanism, run GraphPack, and allows a user to interface with GraphPack directly or indirectly. A direct interface with GraphPack occurs when a user selects one of Xgraphpack's buttons by use of the mouse. An indirect interface with GraphPack is through the GraphPack kernel. In this case, the user supplies a LiLa command and the kernel interprets this command and sends appropriate X Window commands to Xgraphpack. In short, this system allows a user's program (coloring program) to become a client and GraphPack to become a server, where the user can send commands to GraphPack through function calls. Additional animations for coloring could be accomplished through LiLa, as graphical animations are supported by GraphPack. Sound animations are done through the sound facility provided by the Sun systems using C interface.

Parallel Graph Coring using MasPar: A graph coloring program is developed and tested using a parallel heuristic on a MasPar MP-1. The MasPar MP-1 is based on SIMD architecture. The parallel program gets a GraphPack file as an input and converts it to an adjacency matrix in order to work on the MasPar. In this version, however, a Bitmap was chosen to simulate the adjacency matrix for better algorithmic efficiency.

For each colored vertex, all of its adjacent vertices are colored. The ones which are not connected to each other are colored concurrently and the rest are colored sequentially until all the vertices are colored.

Test Graphs: The parallel algorithm is tested on triangle graphs, triangle digraphs, and on graphs that are generated from Atlas. The triangle graphs are generated from a triangle (K_3) and merge two different graphs and make copies a few times, and connect all of them as one graph.