Interfaces and Applications of Graph Packages

Mukkai S. Krishnamoorthy, Ugur Dogrusoz, Mark Gilder, John Punin,
Andrew Shapira, Bhaskar Vasudevan and David Zelnis

Department of Computer Science
Rensselaer Polytechnic Institute
Troy, NY 12180

August 25, 1995

Abstract

This paper discusses two applications of graph packages. Some of these packages were developed at Rensselaer [6, 10, 2]; others were developed elsewhere [5, 8]. By properly interfacing with other packages, we can solve a rich variety of graph problems. In this paper, we show how these packages can be interfaced, and provide two representative examples. In the first example, we used *Nauty* to generate all non-isomorphic regular graphs (undirected and unlabeled) of degree 3. We used *GraphPack* to draw these graphs. We also used the *Maple* interface to *GraphPack* to calculate the characteristic polynomials of these graphs. In the second example, we used *GraphPack* with an object oriented parallelizing compiler called *HICOR* [2] to display the results of different scheduling disciplines. We have also developed a new graph drawing system called *Zgraph*, using *Motif* under *X Windows* system [10]. Using *Zgraph* one can draw graphs of up to 10,000 vertices. We were also able to interface *Zgraph* with other packages. All of our packages are written in *C* and run under *X Windows*.

1 Introduction

In this paper, we present two graph drawing problems and show how we arrived at the solution. We used several software packages. *GraphPack* is a graph and digraph manipulation system that was developed at Rensselaer eight years ago. It has a display/edit window and a language interpreter. The drawback of this system is that it can only draw graphs of up to 60 vertices. Also, it cannot perform any symbolic computation. *Nauty* [8] is a system developed by *B. McKay* which tests for isomorphism between graphs, but contains no display features. *Maple* [1] is a symbolic computation system which has limited display facilities. The *HICOR* system [2] developed at Rensselaer is a parallelizing compiler based on an object oriented intermediate code representation. This compiler utilizes *GraphPack* to display the control flow digraph and the schedule digraph. *The Stanford GraphBase*, or *GraphBase*, developed at Stanford is a carefully crafted system that performs efficient computations on large graphs. Since *GraphPack* cannot handle graphs of large sizes, we have developed a *Zgraph* system which can be used as a front-end system for *GraphBase*.

2 Enumeration of 3-Regular Graphs

In the design of computer interconnection networks, one is often interested in regular graphs of small degree (i.e., all vertices have the same small degree). Such graphs have been studied. For example

Robinson and Wormald [9] have enumerated all regular graphs of degree 3 (also called cubic graphs) up to 40 vertices. In their paper, they only provide the count and not the graphs themselves. For, when the number of vertices is greater than 10, there are too many cubic graphs (i.e., regular graphs of degree 3) to list them all. However, for small number of vertices, one can list them all. In this section, we provide the listing of these graphs for number of vertices=8 and 10 using *Nauty* and *GraphPack*. *Nauty* [8] is a software package written by Brendan McKay for doing computations about graph isomorphism, such as determining whether two graphs are isomorphic, or finding the automorphism group of a graph. (At the time of this writing, *Nauty* can be obtained by *ftp* on the inter-net from `bellatrix.anu.edu.au` in the directory `pub/nauty19`, using the name `anonymous` with any password.) Included with *nauty* is a graph generation program called *makeg*. *Makeg* can generate all graphs with a given number of vertices, and can accommodate various restrictions, including generating only bipartite graphs, connected graphs, or graphs with a given number of edges between two given integers.

For small graphs, *makeg* provides a good way to obtain all the graphs of a certain type. One simply generates all graphs of a given order, using the most restrictive graph class provided by *makeg*, and then filters out unwanted graphs. We generated the cubic graphs of order n for $n = 8, 10$ by instructing *makeg* to list all the graphs of order n with $3n/2$ edges, and discarding the non-cubic graphs. For $n = 10$, *makeg* used about 105 seconds on a SPARC 20 to generate 136433 order 10 graphs with 15 edges; 21 of these graphs turned out to be cubic. While this is not an especially compute-efficient process, it is quite efficient in terms of human time.

Here we display all 3-regular non-isomorphic undirected and unlabeled graphs of size 8 and 10 vertex graphs and the characteristic polynomials of these graphs. The characteristic polynomial of a graph is the characteristic polynomial of its adjacency matrix. Characteristic Polynomials are important in the design of interconnection networks to study its expansion property. These graphs are drawn with a barycentric embedding using *GraphPack* and its interface to *Xfig*, a drawing tool that is based under *X Windows*. It is interesting to see that almost all connected bridgeless graphs are Hamiltonian.

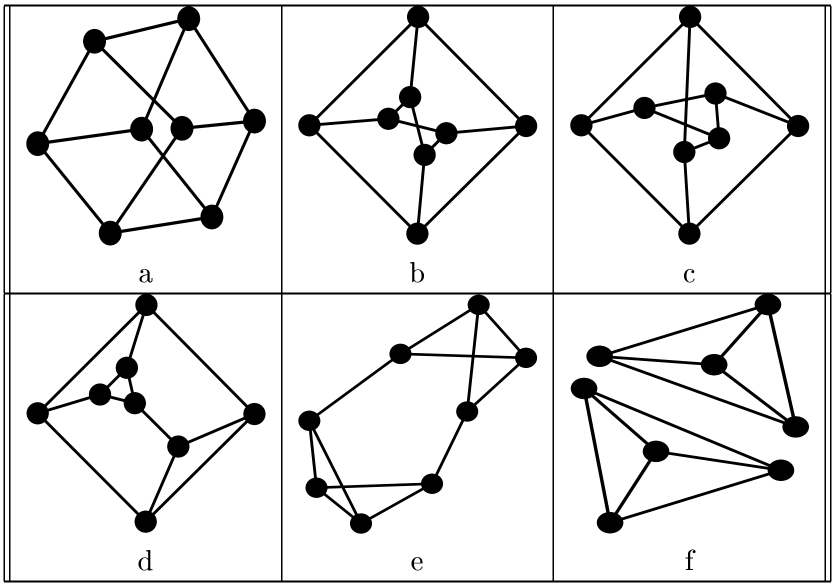

Figure 1: The Cubic Graphs on 8 Vertices

The characteristic polynomials of these graphs in Figure 1 are as follows.

a. $9 - 28x^2 + 30x^4 - 12x^6 + x^8$
b. $-3 + 16x - 20x^2 - 16x^3 + 34x^4 - 12x^6 + x^8$
c. $12x - 31x^2 + 36x^4 - 2x^5 - 12x^6 + x^8$
d. $9 - 12x - 36x^2 + 16x^3 + 38x^4 - 4x^5 - 12x^6 + x^8$
e. $-15 - 40x - 12x^2 + 48x^3 + 38x^4 - 8x^5 - 12x^6 + x^8$
f. $9 + 48x + 100x^2 + 96x^3 + 30x^4 - 16x^5 - 12x^6 + x^8$

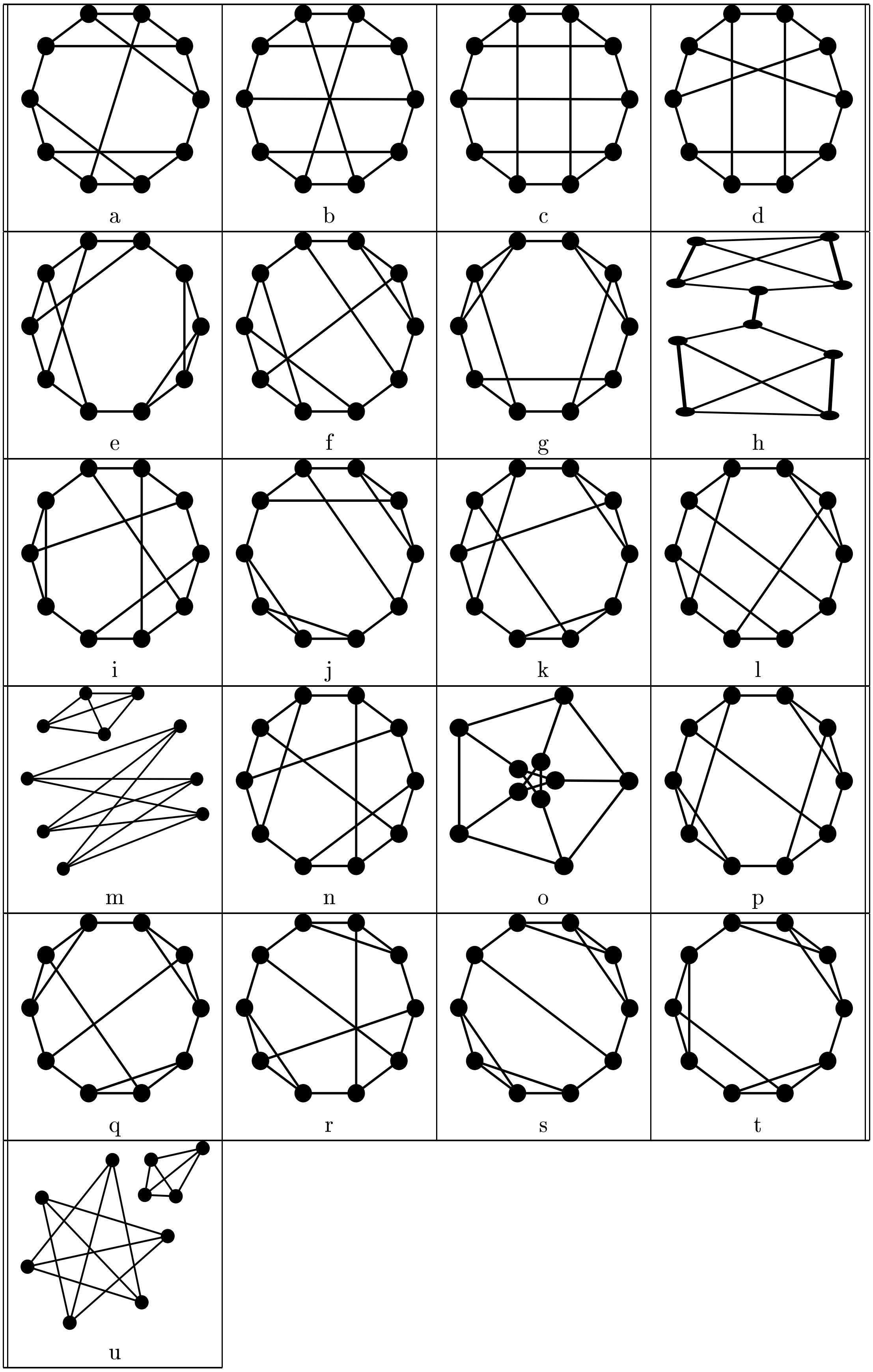

Figure 2: The Cubic Graphs on 10 Vertices

The characteristic polynomials of the graphs in Figure 2 are as follows.

a. $36x^2 - 85x^4 + 63x^6 - 15x^8 + x^{10}$
b. $-9 + 55x^2 - 105x^4 + 65x^6 - 15x^8 + x^{10}$
c. $3 + 20x + 35x^2 - 20x^3 - 85x^4 - 4x^5 + 65x^6 - 15x^8 + x^{10}$
d. $-9 - 12x + 59x^2 + 36x^3 - 117x^4 - 12x^5 + 69x^6 - 15x^8 + x^{10}$
e. $-12x^2 - 56x^3 - 61x^4 + 36x^5 + 63x^6 - 4x^7 - 15x^8 + x^{10}$
f. $12x + 35x^2 - 22x^3 - 96x^4 + 12x^5 + 67x^6 - 2x^7 - 15x^8 + x^{10}$
g. $3 + 20x + 23x^2 - 64x^3 - 105x^4 + 32x^5 + 69x^6 - 4x^7 - 15x^8 + x^{10}$
h. $-12x^2 - 56x^3 - 37x^4 + 64x^5 + 63x^6 - 8x^7 - 15x^8 + x^{10}$
i. $-12 - 8x + 91x^2 - 2x^3 - 132x^4 + 8x^5 + 71x^6 - 2x^7 - 15x^8 + x^{10}$
j. $3 + 26x + 30x^2 - 76x^3 - 96x^4 + 48x^5 + 69x^6 - 6x^7 - 15x^8 + x^{10}$
k. $-21 + 16x + 99x^2 - 52x^3 - 141x^4 + 28x^5 + 73x^6 - 4x^7 - 15x^8 + x^{10}$
l. $3 + 26x + 54x^2 - 24x^3 - 116x^4 + 12x^5 + 69x^6 - 2x^7 - 15x^8 + x^{10}$
m. $27x^4 + 72x^5 + 51x^6 - 8x^7 - 15x^8 + x^{10}$
n. $-48x + 76x^2 + 64x^3 - 133x^4 - 16x^5 + 71x^6 - 15x^8 + x^{10}$
o. $48 - 160x + 120x^2 + 120x^3 - 165x^4 - 24x^5 + 75x^6 - 15x^8 + x^{10}$
p. $24x + 64x^2 - 48x^3 - 121x^4 + 28x^5 + 71x^6 - 4x^7 - 15x^8 + x^{10}$
q. $12 + 68x + 75x^2 - 114x^3 - 144x^4 + 48x^5 + 75x^6 - 6x^7 - 15x^8 + x^{10}$
r. $-48 + 16x + 144x^2 - 36x^3 - 157x_2^4 4x^5 + 75x^6 - 4x^7 - 15x^8 + x^{10}$
s. $48x + 44x^2 - 104x^3 - 101x^4 + 64x^5 + 71x^6 - 8x^7 - 15x^8 + x^{10}$
t. $-36x^2 - 132x^3 - 93x^4 + 68x^5 + 71x^6 - 8x^7 - 15x^8 + x^{10}$
u. $-36x^2 - 84x^3 - 13x^4 + 96x^5 + 63x^6 - 12x^7 - 15x^8 + x^{10}$

3 Compiler Interface

In compilers, one is interested in seeing the structure of the intermediate code, as well as the effect of the schedule on the intermediate code. Besides displaying the Intermediate Code representation, *Interactive Visualization Tool* along with *GraphPack* is responsible for displaying both the data dependency DAG and the compile-time execution graph representing the schedules.

3.1 Dependency Graph Display

The GENERATEDEPENDENCIES C++ method computes a DAG that represents the dependencies between children of the selected *HICOR* object. The resulting information is stored in a specialized *Dag* object. This object contains a variety of specialized methods for manipulating *Dags*. The vertex numbers of the *Dag* correspond directly to the associated *HICOR* object number. Each *edge* object contains an attribute which holds the estimated cost of sending the necessary data to the *HICOR* object represented by the sink vertex.

The *Interactive Visualization Tool* uses the following methods to display a *Dag*.

- `GetVertexSet`: returns the set of all vertices for the *Dag*.
- `GetSourceVertices`: returns an ordered set of source vertices for the *Dag*.
- `GetSinkVertices`: returns an ordered set of sink vertices for the *Dag*.
- `GetVertexEdgeSet`: returns the set of neighboring vertices of a given input vertex.

- `FindEdge`: returns the *edge* object for the edge between specified vertices.
- `FindVertex`: returns the *vertex* object representing a specified vertex.
- `GetWeight`: returns the estimated cost of computing the associated *HICOR* object when applied to a *vertex* object, or the data communication cost when applied to a *edge* object.

The tool traverses the *Dag* using a breadth-first search traversal. The set of starting vertices are initially retrieved via the `GetSourceVertices` method. Then, for each of these vertices, the neighboring vertices are found using the `GetVertexEdgeSet` method. Information about each vertex and edge is retrieved by applying the available methods when visiting each *vertex* and *edge* object, respectively.

3.2 Compile-Time Schedule Display

The schedule is maintained in a specialized object called the *LayerRep* object. This object consists of a series of successive layers, each of which contains tasks which may all be executed concurrently. Each task in a particular layer may also consist of several subtasks, indicating that several *HICOR* objects were merged to form a task with a bigger grain size in order to reduce overhead. As with the display of the dependency graph, the numbers of each vertex (or task) correspond directly to the associated *HICOR* object number. Layers are labeled from 1 to n, top to bottom. Tasks are labeled from 1 to m, left to right. The *Interactive Visualization Tool* uses the following methods for displaying the schedules.

- `NumberLayers`: returns the number of layers in the *LayerRep* instance.
- `NumProcsForLayer`: returns the number of tasks (or processes) in a specific layer.
- `GetTasksForLayer`: returns a list of all tasks in the specified layer.
- `GetTaskSet`: returns the HICOR object(s) which makes up a specific task in a specific layer.
- `GetInputCommEdges`: retrieves the set of all incoming edges for the specified task in the specified layer.
- `GetOutputCommEdges`: retrieves the set of all outgoing edges for the specified task in the specified layer.
- `GetDag`: returns the *Dag* object from which the *LayerRep* instance was constructed. All methods available to the *Dag* object may be utilized to retrieve information about the associated *HICOR* object numbers in the *LayerRep* instance.

To display the schedule represented by the *LayerRep* object, the *IVT* simply traverses each layer extracting the task numbers. Once each task number is known, the methods applicable to the *Dag* object are invoked to extract detailed information; i.e., estimated execution costs and data communication costs. Three schedules are presented in the following Figures [3], [4]. Figure 3 displays a schedule for execution of an object (which corresponds to a segment of code in the source program). There are three layers. Two items in Layer 1 can be executed simultaneously. Two items in Layer 3 can also be executed simultaneously. Figure 4 shows a different schedule when the target architecture (when communication costs and process creation costs) change resulting in a different schedule. Here there are only two layers.

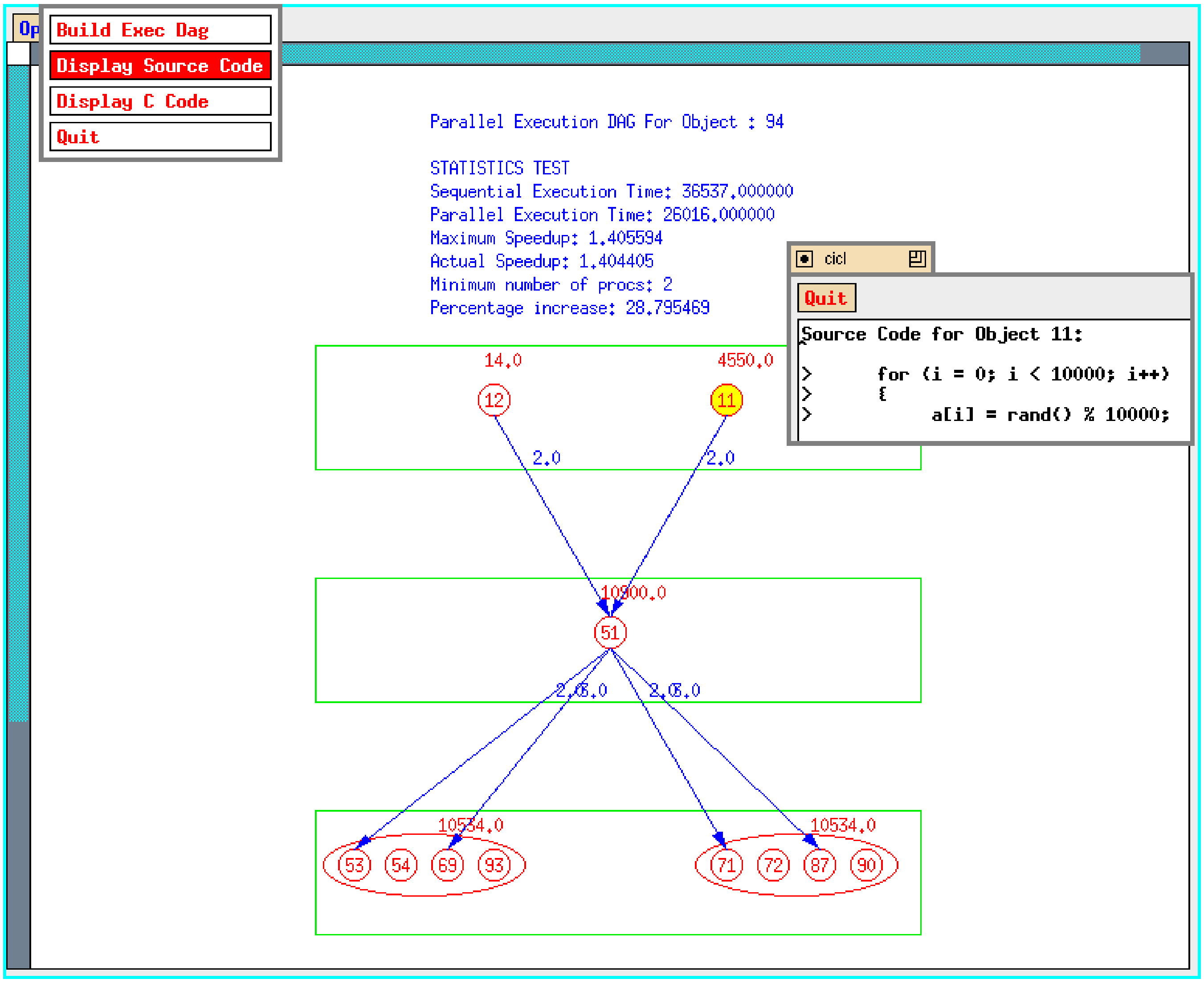

Figure 3: Schedule 1

4 GraphBase and Zgraph

GraphBase is a tool similar to *GraphPack* for the creation and manipulation of graphs. *GraphBase* consists of a library of *C* routines that can be used in *C* programs to create and use graphs. *GraphPack* is based on an underlying interpreter and is interactive. *GraphBase* does not have any display related functions. *Zgraph* is a carefully crafted package using Motif and *X Windows* that can display graphs of up to 10,000 vertices.

An interface between *GraphBase* and *Zgraph* has been implemented so that graphs created with *GraphBase* can be viewed and displayed using *Zgraph* and also manipulated graphically, if necessary. An utility `gb2zgr` was developed at Rensselaer. This enables the conversion of large sized graphs created textually using *GraphPack* to be viewed using *Zgraph*.

Since *GraphBase* does not specify any display related information such as the layout of the vertices on the drawing canvas, default values are assumed while converting graphs from *GraphBase* to *Zgraph* format. The conversion program `gb2zgr` assumes that the graphs will be displayed with a circular placement of vertices; the vertices and the edges will be displayed in default colors. In addition, default

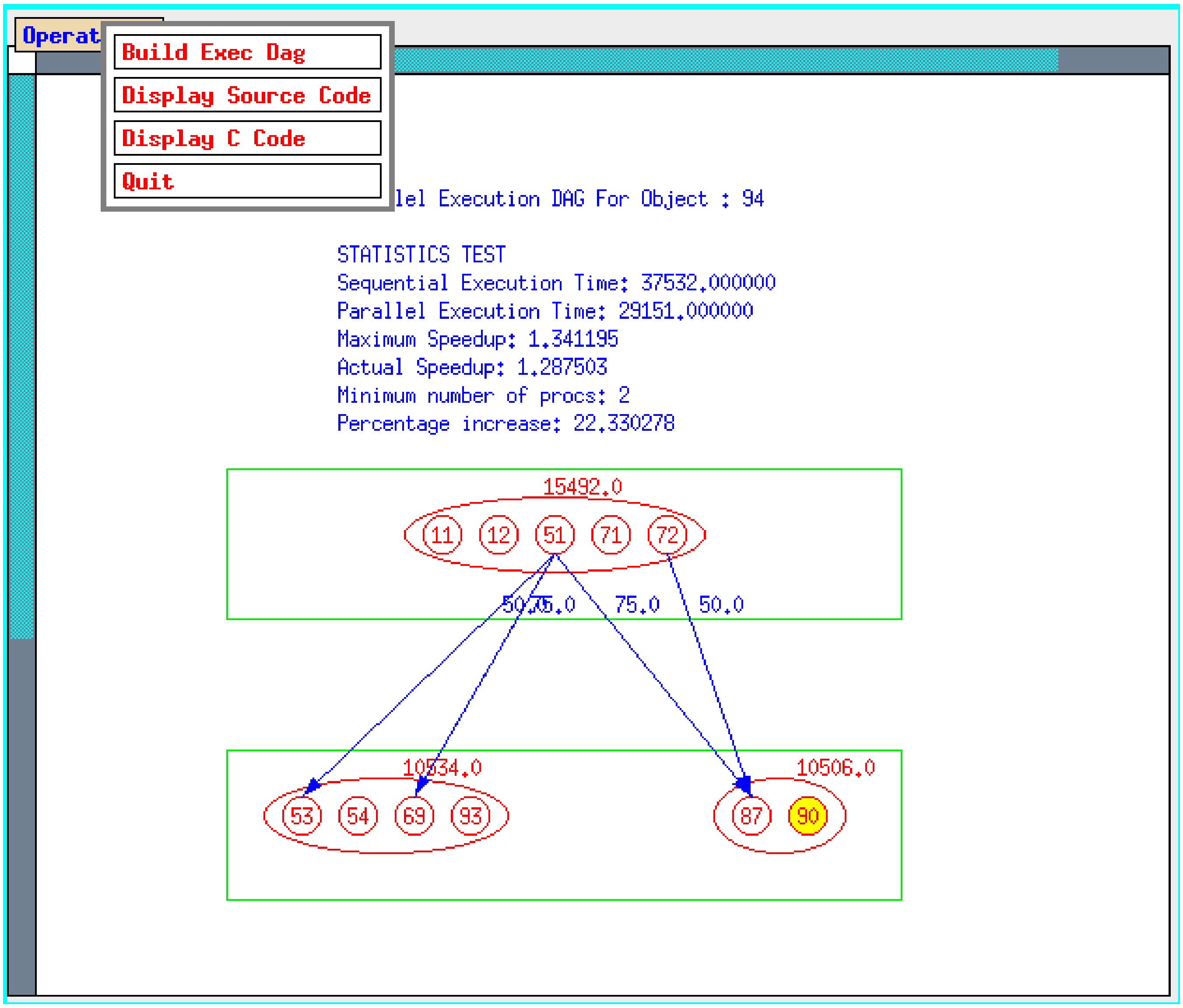

Figure 4: Schedule 2

values are assumed for the dimensions of the drawing canvas that is required to display the graph. Large sized graphs are then rescaled to fit this canvas.

A similar mechanism is used for developing an interface between *GraphPack* Interpreter and *Zgraph*.

5 Acknowledgement

The authors wish to thank Ms.Joyce Brock for her help with the manuscript.

References

[1] B.W. Char et al. , *Maple V Language Reference Manual*, Springer-Verlag, NY, 1991.

[2] M. Gilder and M. Krishnamoorthy, "An Object-Oriented Intermediate Code Representation for the Development of Parallelization Tools," *Journal of Object Oriented Programming*, January 1995, pp. 50-57.

[3] M. Gilder and M. Krishnamoorthy, "Automatic Source-Code Parallelization using HICOR Objects," *International Journal of Parallel Programming*, Vol. 22, No.3, 1994, pp. 309-356.

[4] M. Gilder, J. Punin and M. Krishnamoorthy, "A GUI for Parallel Code Generation," in *Programming Environments for Massively Parallel Distributed Systems*, Karsten M. Decker and Rene M. Rehmann (ed), Birkhäuser Verlag, Basel, Switzerland, July 1994, pp. 69-79.

[5] D. Knuth, *The Stanford GraphBase*, Addison Wesley, Reading, Mass. 1994.

[6] M. Krishnamoorthy, A. Suess, M. Onghena, F. Oxaal and T. Spencer "Improvements to GraphPack: A System to Manipulate Graphs and Digraphs," *DIMACS Series on Computational Support for Discrete Mathematics,* Vol. 15, 1994, pp. 279-296.

[7] M. Krishnamoorthy, T. Spencer,and E. McCaughrin, *GraphPack: A software system to manipulate graphs and directed graphs*, Department of Computer Science, Technical Report 90-7, Rensselaer Polytechnic Institute, Troy, NY, 1990.

[8] B. McKay, *Nauty User's Guide (Version 1.5)*, TR-CS-90-02, The Australian National University, Canberra, Australia, 1990.

[9] R. Robinson and N. Wormald, "Number of Cubic Graphs," *Journal of Graph Theory*, Vol. 7, 1983, pp. 463-467.

[10] D. Zelnis, *Zgraph - A system to visualize large graphs*, Rensselaer Polytechnic Institute, Troy, NY (unpublished), 1994.