International Journal of Modern Engineering Research (IJMER)
www.ijmer.com Vol.2, Issue.5, Sep-Oct. 2012 pp-3429-3433 ISSN: 2249-6645

Analyzing Fault Diagnosis Using PPDG & Its Application's

Sambangi Swathi (M. Tech.), Mudiganti Vijaya Bhaskar (Associate Professor)
Department of Computer Science Gokul Institute of Technology and Sciences, Bobbili, India.

ABSTRACT: *In this paper a Model based Fault Localization Technique is used, which is called Probabilistic Program Dependence Graph, a contemporary model that scans the internal behavior of the project over a set of test inputs. The PPDG model captures the conditional statistical dependence and independence relationships among program elements in such a way that it facilitates by making probabilistic inferences about program behaviors. PPDG construction is enhanced by Program Dependence Graph (PDG) that represents the structural dependences of a program with estimations of statistical dependences between node states, which are computed from the test set. The acquirement of probabilistic graphical models, which are widely used in applications such as medical diagnosis are the basis for the PPDG.This paper discusses the algorithms needed for constructing PPDGs and the applications of the PPDG to fault diagnosis. This paper also outlines that Probabilistic Program Dependence Graphs can simplify fault localization and fault comprehension.*

Keywords**:** PPDG, Fault Localization technique, dependencies, Graphs.

I. INTRODUCTION

In software engineering applications to abstract relevant relationships between program elements or states, a variety of graphical models have been used and thereby those models facilitate program analysis and understanding. These models include control-flow graphs, call graphs, finite-state automata, and program dependence graphs. If the models are generated by static analysis, they indicate that certain occurrences are possible at run time where as models produced by dynamic analysis indicate what actually does occur during one or more executions. The commonly used graphical models do not support making conclusions about the program behavior and also limits the utility of the models for reasoning about the causes and effects of inherently uncertain program behaviors, such as runtime failures.

In this paper, we show how the program dependence graph can be used to know the program behavior. The model captures the conditional statistical dependence and independence relationships among program elements in a way that facilitates making probabilistic inferences about program behaviors. We call this model a Probabilistic Program Dependence Graph (PPDG). Our technique produces the PPDG for a program by augmenting its program dependence graph automatically. The technique associates a set of abstract states with each node in the PPDG. Each abstract state represents a (possibly large) set of concrete nodes states in a way that is chosen to be relevant to one or more applications of PPDGs. Each node has a conditional probability distribution that relates the states of the node to the states of its parent nodes. The technique estimates the parameters of the probability distribution by analyzing executions of the program, which are induced by a set of test cases or captured program inputs.

II. BACKGROUND & PREVIOUS WORK

In this section, we briefly review two models that form the basis for the Probabilistic Program Dependence Graph. The first is the program dependence graph, which represents structural dependences between program statements. The second is a dependency network, which is a type of probabilistic graphical model that represents conditional dependence and independence relationships between random variables.

A) Program Dependence Graph

Before describing a program dependence graph, we define and illustrate the control flow graph, which is used to construct the program dependence graph.

Definition 1**:** A control flow graph for a program P is a pair(N,E), where N is a set of nodes that represents statements in P and E is a set of directed edges in which each edge (ni, nj) represents the flow of control from node ni to node nj. Edges representing conditional branches are labeled to represent the conditions under which those edges are taken.

To illustrate, consider example program findmax, shown in Fig. 1, which outputs the maximum of a set of integers. Fig. 2 shows the control flow graph for findmax. In the graph, each node is labeled with the number of the program statement that it represents, and each edge shows the flow of control between the corresponding statements. For example, node 1 represents the first statement in the program and node 10 represents the last statement in the program. For another example, node 4 has two outgoing edges: Edge (4, 5) is taken if the condition at 4 is true (i.e., the while loop is entered) and edge (4, 10) is taken if the condition at 4 is false.

```
0  void findmax() {
1     int i = 0;
2     int n = read_int();
3     int max = 0;
4     while (i < n){
5        int v = read_int();
6        if (v > max)
7           max = v;
8        i++;
9     }
10    print(max);
11 }
```

Fig.1:An Example Program of FindMax

Definition 2: In a control flow graph G, node n1 is control dependent on node n2 if n2 has outgoing edges e1 and e2 such that 1) every path in G starting with e1 and ending with an exit node contains n1 and 2) there is a path starting with e2 and ending with an exit node that does not contain n1.

For example, in Fig. 2, nodes 1-4 and 10 are control dependent on the program entry point—by convention, a dummy edge is added from the program entry point to each program exit point so that top-level nodes are control dependent on the entry. Nodes 5, 6, and 8 are control dependent on node 4, and node 7 is control dependent on node 6.

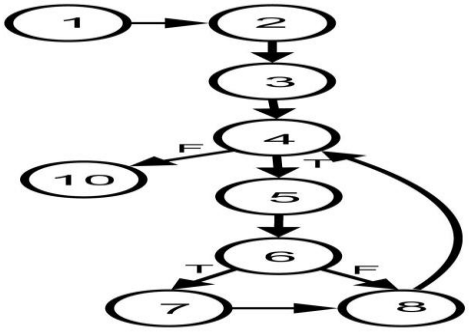

Fig.2: FindMax with its CFG

Definition 3: In a control flow graphG, node n1 is data dependent on node n2 if 1) n2 defines a variable v, 2) there is a path in G from n2 to n1 that does not redefine v, and 3) n1 uses v.

For example, in Fig. 2, nodes 4 and 8 are data dependent on node 1 for variable i and node 4 is data dependent on node 2 for variable n. Now, using control dependence and data dependence, we can define a program dependence graph [8].

Definition 4: A program dependence graph (PDG) is a directed graph whose nodes represent program statements and whose edges represent data and control dependences. Labels on the control dependence edges represent the truth values of the branch conditions for those edges, and labels on data dependence edges represent the variables whose values flow along those edges.

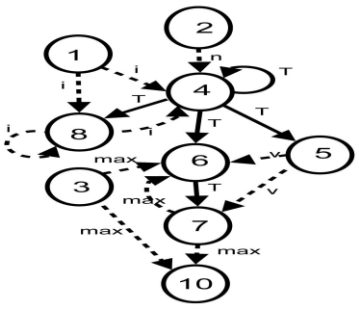

Fig.3:FindMax with its PDG

Fig. 3 shows the PDG for program findmax in Fig. 1. The nodes in the PDG are labeled with the line numbers of the corresponding statements in the program. Solid edges represent control dependences between nodes and dotted edges represent data dependences between nodes. Labels on the control dependence edges are either "T" for true or "F" for false. Labels on the data dependence edges represent the variables involved in the data flows between the nodes. For example, in Fig. 1c, node 6 is control dependent on node 4 and it is data dependent on nodes 3, 5, and 7. The control dependence edge between node 4 and node 6 has the label "T," which indicates that node 6 is executed when the branch condition at node 4 is true. The data dependence edge between node 3 and node 5 has the label "max," which indicates that the value of variable "max" at node 3 flows to node 6. The precision of a PDG depends on the precision of the underlying analyses. For example, the precision of the pointer analysis affects the precision of the data dependences.

B) Dependency Network

A dependency network is a type of probabilistic graphical model.

Definition 5: A probabilistic graphical model is an annotated graph that captures the probabilistic relationships among a set of random variables. The nodes in the graph represent random variables and the edges represent conditional dependences between the random variables.

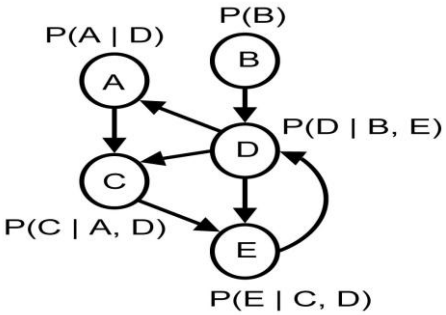

Fig.4: An Example for Dependency Network

III. SYSTEM ANALYSIS & DESIGN

A) System Analysis

A probabilistic program dependence graph (PPDG) is created by transforming the PDG of a program into a dependency network. We use a dependency network because it permits directed cycles, which are present in the PDGs of typical programs because of loops. Henceforth, we use the terms "loop" and "cycle" interchangeably. The process of producing a PPDG consists of five main steps, as illustrated in Fig. 5. First, the PDG-generation step generates the PDG of the input program P. Second, the PDG-transformation step takes the PDG, and transforms it by structurally changing the PDG and specifying states at nodes in the PDG, which results in a transformed PDG. Third, the Instrumentation step inserts probes into P to gather the execution data needed to estimate the parameters of the PPDG, and produces the instrumented program P0. Fourth, the Execution step executes P0 with its test suite TP to generate the execution data. Finally, the Learning step generates a PPDG based on the execution data and the transformed PDG by estimating the parameters of the PPDG. The resulting PPDG is formally defined as follows:

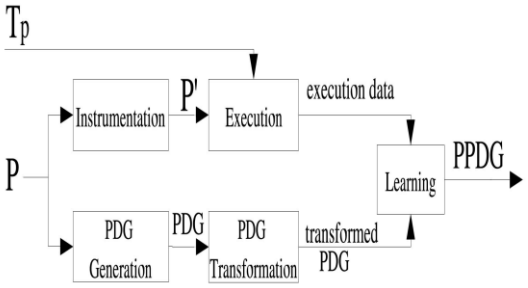

Fig.5: Construction of PPDG

B) Module Analysis

There are four modules in our paper

- ***New user Registration:***

User can be providing the personal information at the registration phase. These sensitive user results of information can be placed inside the database. Automatically user can be get the information or credentials (username and password).

- ***Testing, Debugging and Maintenance:***

Based on the user credentials enter inside the project deployment to enter inside the homepage. Browse or select the program and deploy the program. Program can be executed indentify the relationship of information from one state to another state. Identify the program behavior of information and faults comprehension information. Whenever to identify the faults automatically to define that information like quality representation process.

- ***Probabilistic Program Dependence Graph:***

PPDG produces the PPDG for a program by augmenting its program dependence graph automatically. The technique associates a set of abstract states with each node in the PPDG. Each abstract state represents a (possibly large) set of concrete nodes states in a way that is chosen to be relevant to one or more applications of PPDGs. Each node has a conditional probability distribution that relates the states of the node to the states of its parent nodes. The technique estimates the parameters of the probability distribution by analyzing executions of the program, which are induced by a set of test cases or captured program inputs.

- ***Show the dependency network:***

There are different kinds of probabilistic graphical models, including Bayesian networks, Markov random fields, and dependency networks. Bayesian networks are directed acyclic graphs, whereas Markov random fields are undirected graphs. Dependency networks are similar tom Bayesian networks except that they may contain cycles.

C) System Design

Class diagrams model class structure and contents using design elements such as classes, packages and objects. The association relationship is the most common relationship in a class diagram. The association shows the relationship between instances of classes. For example, the class Order is associated with the class Customer. The multiplicity of the association denotes the number of objects that can participate in then relationship.

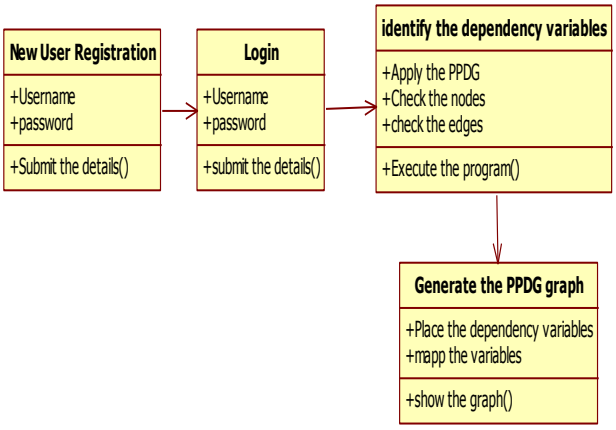

Fig.6: Inter-Operational Class Diagram for Framework

A use case illustrates a unit of functionality provided by the system. The main purpose of the use-case diagram is to help development teams visualize the functional requirements of a system, including the relationship of "actors" (human beings who will interact with the system) to essential processes, as well as the relationships among different use cases. Use-case diagrams generally show groups of use cases -- either all use cases for the complete system, or a breakout of a particular group of use cases with related functionality

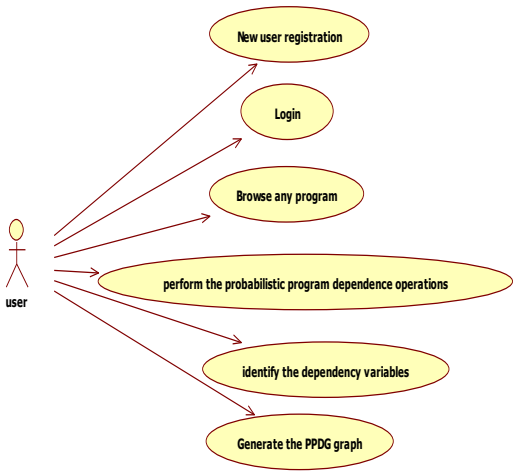

Fig.7: Inter-operational Usecase Diagram for the Framework

IV. RESULTS

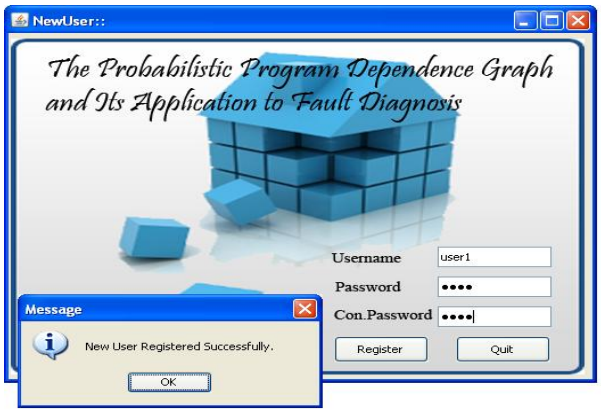

Fig.7: New User Registration

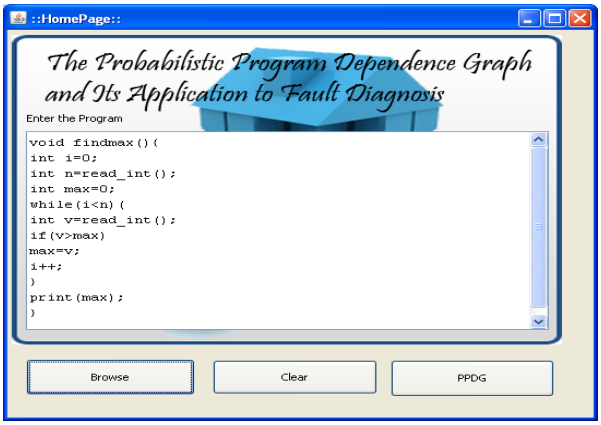

Fig.8: Giving Program to find the Dependency

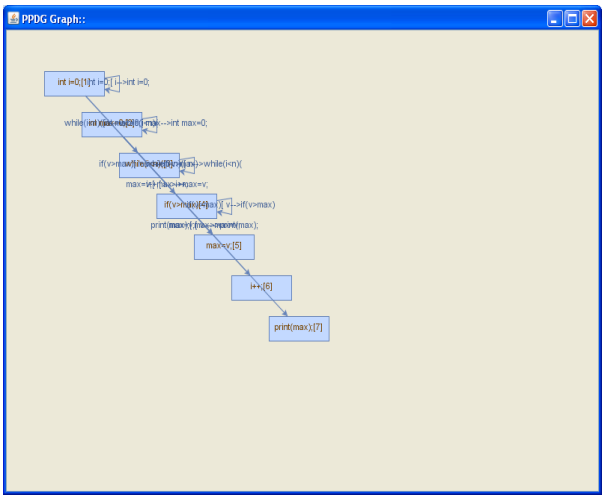

Fig.9: Extracting the PPDG Graph

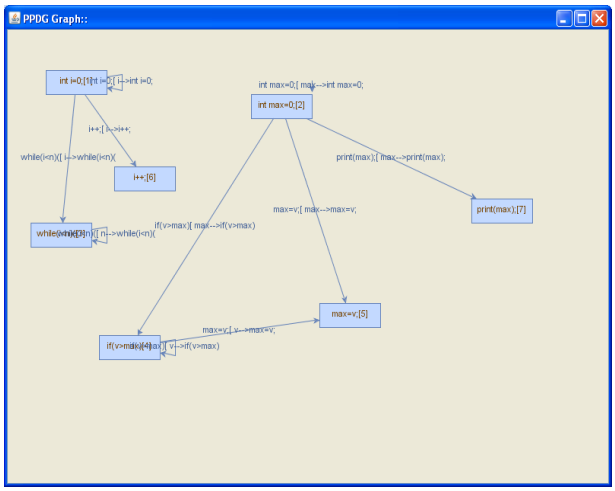

Fig.10: Identifyng the PPD Graph

V. CONCLUSION

In this paper, we presented our technique PPDG that uses the program dependence graph to create a novel probabilistic graphical model .PPDG depends on the PDG that captures the statistical dependences among program elements and enables the use of probabilistic reasoning to analyze program behaviors. This paper has discussed the two applications of the PPDG for Software engineering tasks. For the first task fault localization it has shown that how the PPDG can be used to overcome the limitations of current fault-localization techniques by introducing a simple ranking –based algorithm. Fault comprehension as a second task of application, we presented an algorithm that exploit the interpretive nature of the PPDG. *RankCP* is an alogorithm which uses the PPDG to rank statements to assist in fault localization and *FaultComp*, which uses the PPDG to generate explanations to aid in fault comprehension.RankCP and FaultComp were implemented for the evolution of the PPDG.

The most critical part of our PPDG construction is the execution information, which is used to estimate the parameters of the PPDG. This execution information is dependent on the test suite that is executed by the instrumented program. Our experiment, although limited, suggests that our technique is more efficient than existing techniques that consider single failing executions. We used PPDG in this paper depending on PDG, and hence the statistical dependences across the functions were not captured by PPDG.In the future we will use the PPDG as the base on the interprocedural PDG and facilitate PPDG in such a way that it can capture the statistical dependencies of program elements.

We have shown the potential utility of applying the PPDG to the problem of fault diagnosis but we believe that it has other applications. We therefore plan to investigate the potential application of the PPDG to other software engineering tasks.

REFERENCES

[1] S. Bates and S. Horwitz, "Incremental Program Testing Using Program Dependence Graphs," Proc. Symp. Principles of Programming Languages, pp. 384-396, Jan. 1993.
[2] J.F. Bowring, J.M. Rehg, and M.J. Harrold, "Active Learning for Automatic Classification of Software Behavior," Proc. Int'l Symp. Software Testing and Analysis, pp. 195-205, July 2004.
[3] S. Thrun, "Robotic Mapping: A Survey," Exploring Artificial Intelligence in the New Millennium, pp. 1-35, Morgan Kaufmann Publishers, Inc., 2002.
[4] W. Weimer and G. Necula, "Mining Temporal Specifications for Error Detection," Proc. Int'l Conf. Tools and Algorithms for the Construction and Analysis of Systems, pp. 461-476, Apr. 2005.
[5] M. Weiser, "Program Slicing," Proc. Int'l Conf. Software Eng., pp. 439-449, Mar. 1981.
[6] J. Ferrante, K.J. Ottenstein, and J.D. Warren, "The Program Dependence Graph and Its Use in Optimization," ACM Trans. Programming Languages and Systems, vol. 9, no. 3, pp. 319-349, July 1987.
[7] S. Galan, F. Aguado, F.J. Diez, and J. Mira, "NasoNet, Joining Bayesian Networks, and Time to Model Nasopharyngeal Cancer Spread," Artificial Intelligence in Medicine, pp. 207-216, Springer, 2001.
[8] R. Alur, P. Cern y, P. Madhusudan, and W. Nam, "Synthesis of Interface Specifications for Java Classes," Proc. Symp. Principles of Programming Languages, pp. 98-109, Jan. 2005.
[9] X. Zhang, N. Gupta, and R. Gupta, "Pruning Dynamic Slices with Confidence," Proc. Conf. Programming Language Design and Implementation, pp. 169-180, June 2006.
[10] B. Liblit, M. Naik, A.X. Zheng, A. Aiken, and M.I. Jordan, "Scalable Statistical Bug Isolation," Proc. Conf. Programming Language Design and Implementation, pp. 15-26, June 2005.
[11] C. Liu, X. Yan, L. Fei, J. Han, and S.P. Midkiff, "SOBER: Statistical Model-Based Bug Localization," Proc. European Software Eng. Conf.and ACM SIGSOFT Symp. Foundations of Software Eng., pp. 286-295, Sept. 2005.

AUTHORS LIST

SWATHI SAMBANGI received her Bachelor's Degree in Information Technology in GMR Institute of Technology Affiliated to JNTU Kakinada and pursuing Masters of Technology in Software Engineering in Gokul Institute of Technology and Sciences affiliated to JNTU Kakinada. Her research areas of interest are Software Engineering, Computer Networks and Data Mining.

MUDIGANTI VIJAYA BHASKAR, Completed his B.Tech ,Computer Science in (A.U) 1997 and M.tech,Computer Science (A.U) 2000 .Presently Pursuing Ph.D in Soft computing in Andhra University . His area of intrest are Image processing, Artificial Intelligence,Robotics and Soft computing.