北京大学

硕士研究生学位论文

题目：**基于程序依赖图融合的错误定位算法的设计与评估**

姓　　名：	张知凡
学　　号：	1501214426
院　　系：	信息科学技术学院
专　　业：	计算机软件与理论
研究方向：	群体软件开发方法与技术
导师姓名：	张伟 副教授

二〇一八年五月

版权声明

任何收存和保管本论文各种版本的单位和个人，未经本论文作者同意，不得将本论文转借他人，亦不得随意复制、抄录、拍照或以任何方式传播。否则，引起有碍作者著作权之问题，将可能承担法律责任

摘要

软件开发过程本质的困难与人类个体智能的局限性使得开发者在软件的编码活动中会因个体能力的局限或失误等因素而不可避免地出现错误。而互联网的发展促成了面向人类群体的基于互联网的群体智能。基于互联网的群体智能为突破人类个体智能的局限指出了一种可能的方向。在基于互联网的群体协同软件开发过程中，开发者可以克服物理时空上的限制结成大规模群体进行合作，从而为弥补单一个体在软件开发过程中的能力局限和失误创造了可能。

在基于互联网群体智能的群体协同软件开发中，群体间信息的交流与共享可以为个体优化自身开发成果提供帮助。在群体协同编程求解同一问题的场景下，当群体对于目标问题的解决方案形成一定共识时，则可以将群体对于目标问题的解决方案加以融合，并以此为基础对群体中的每个个体进行反馈以帮助个体发现自身程序的缺陷。实现这一过程的关键技术问题之一，则是构造一个自动化的群体程序融合与反馈系统。

针对这一问题，本文提出了一种以程序依赖图为表示方式的程序融合算法与一种基于群体共识的错误定位算法。程序融合算法将程序转化为程序依赖图表示，并对程序依赖图进行融合。错误定位算法在程序融合的基础上，利用群体对目标问题解决方案达成的共识对个体程序可能存在的错误进行定位与反馈。

本文的主要工作包括：

(1) 提出了一种以程序依赖图为表示方式的程序融合算法。该算法将程序转化为程序依赖图表示，以广义熵作为融合质量的度量方式，通过遗传算法对最优融合方案进行搜索。

(2) 提出了一种以程序融合结果为基础，基于群体共识的错误定位算法。算法基于程序融合结果与个体程序之间的差异性对个体程序中的错误进行定位。

(3) 设计了基于同构程序的程序融合实验。实验在不同规模的群体中，对多种程序进行融合，以验证本文融合算法的有效性。同时设计了基于变异程序的仿真群体错误定位实验与基于实际群体的错误定位实验。两种实验分别在仿真群体与实际群体进行程序融合与错误定位，并对错误定位效果进行了评估。

关键词：群体智能，程序依赖图，融合算法，错误定位

A Program Dependence Graph Merging Based Fault Localization Algorithm: Design and Evaluation

Zhifan Zhang (Computer Software and Theory)
Directed by Associate Prof. Wei Zhang

ABSTRACT

The essential difficulties of the software development process and the limitations of the human individual intelligence make developers inevitably make mistakes in the software coding activities due to individual's error and the limitations of individual capabilities. The development of the Internet has led to Internet-based collective intelligence. Internet-based collective intelligence points out a direction for breaking the limitations of human individual capabilities. In the process of Internet-based collaborative software development, developers can overcome the limitations of physical space-time and form large-scale groups to cooperate. This creates a possibility to overcome the individual's ability limitations and mistakes in software development.

In collaborative software development based on Internet-based collective intelligence, the exchange and sharing of information between individuals can help individuals to optimize their own development results. In the scenario where the group cooperates to solve the same problem, when the group forms a certain consensus on the solution to the target problem, each individual solutions can be merged together. Based on this, each individual can get feedbacks from the group and discover the faults in their program. One of the key technical issues to implement this process is to construct an automated group program merging and feedback system.

To solve this problem, this paper proposes a program merging algorithm based on program dependence graph and a fault localization algorithm based on group consensus. The program merging algorithm converts the program into a program dependence graph and merge the program dependency graphs. Based on the merging result, the fault localization algorithm uses the group consensus on the solution to the target problem to locate the faults and generate feedbacks.

The main work of this article includes:

(1) We propose a program merging algorithm based on program dependence graphs. The algorithm transforms the program into a program dependence graph, and uses the generalized

entropy as a measure of the merging quality. The algorithm uses genetic algorithm to search the optimal merging result.

(2) We propose a fault localization algorithm based on group consensus and the results of program merging. The algorithm locates faults in individual programs based on the differences between program merging results and individual programs.

(3) We design a program merging experiment based on isomorphic programs. Experiments are conducted on a variety of programs in different size groups to verify the effectiveness of the merging algorithm. We design a simulation experiment on groups of mutated program and a fault localization experiment on real group. We use these two kinds of experiments to verify the effectiveness of program merging algorithm and fault localization algorithm.

KEY WORDS: Collective Intelligence, Program Dependence Graph, Merging Algorithm, Fault Localization

目录

第一章 引言 1
1.1 研究背景 1
1.1.1 群体智能 1
1.1.2 群体协同软件开发 2
1.1.3 程序融合与错误定位 2
1.2 相关研究工作 3
1.2.1 程序依赖图 3
1.2.2 融合算法 4
1.2.3 错误定位 4
1.3 本文主要解决的问题 5
1.4 本文的组织 6

第二章 基于程序融合的错误定位 7
2.1 基于程序融合的错误定位框架 7
2.2 核心概念 8
2.2.1 程序依赖图及程序依赖融合图 8
2.2.2 基于广义熵的融合质量度量 14
2.3 本章小结 17

第三章 基于遗传算法的程序依赖图融合 18
3.1 程序依赖图融合基本框架 18
3.2 程序依赖融合图表示方法 20
3.3 程序依赖融合图的交叉运算 22
3.3.1 节点迁移 22
3.3.2 程序依赖融合图的编辑距离与编辑路径 22
3.3.3 基于编辑路径的交叉运算 24
3.4 程序依赖融合图的变异运算 27
3.5 节点相似度的计算与基于广义熵的适应度函数 28
3.5.1 基于抽象语法树的节点相似度计算 29
3.5.2 基于广义熵的适应度函数 32
3.6 遗传算法中其他步骤的设计 33
3.6.1 种群初始化 33
3.6.2 选择 33

3.6.3 种群多样性调整 34
3.6.4 终止条件设定 34
3.7 本章小结 35
第四章 基于程序依赖融合图的错误定位 36
4.1 基于群体共识的错误定位基本思想 36
4.1.1 程序依赖融合图的意义 37
4.1.2 错误定位的形式 38
4.2 错误定位方法 38
4.2.1 基于节点融合度的错误定位 38
4.2.2 基于边融合度的错误定位 39
4.2.3 基于节点内部相似度的错误定位 40
4.3 本章小结 43
第五章 实验与评估 45
5.1 实验程序的选取 45
5.2 基于同构程序依赖图的融合实验 46
5.2.1 实验方案 46
5.2.2 实验结果与分析 48
5.3 基于变异程序的仿真群体错误定位实验 49
5.3.1 实验方案 50
5.3.2 实验结果与分析 52
5.4 基于实际群体的错误定位实验 54
5.4.1 实验方案 54
5.4.2 实验结果与分析 56
5.5 本章小结 59
第六章 总结与展望 60
6.1 本文总结 60
6.2 进一步工作设想 61
参考文献 62
致谢 64
北京大学学位论文原创性声明和使用授权说明 65

第一章　引言

1.1　研究背景

在软件的编码活动中，开发者个体的能力局限或失误等因素，会使得软件源码中不可避免的包含错误。软件开发具有本质性的困难。这些困难难以克服的主要原因之一来自于人类个体智能的局限性。基于互联网的群体智能为突破人类个体智能的局限指出了一种可能的方向 [1]。基于互联网的群体智能是由互联网的出现而促成的面向人类群体的群体智能。互联网的出现使得开发者可以克服物理时空上的限制，结成更大规模的协作群体。如何利用互联网群体智能，通过群体协同开发来弥补单一个体在软件开发过程中的能力局限和失误，从而使得群体开发者在面对困难问题时显现出超越个体开发者的更高智能水平，成为一个重要的研究方向。

相比于个体独立开发，基于互联网群体智能的群体协同软件开发的优势在于群体间的信息交流与共享可以使得个体的开发成果得到优化。特别的，在多人共同编程求解同一问题的场景下，相同的目标问题使得群体中个体的解决方案（即程序）呈现出一定的相似性。基于这样的相似性，有效地将群体中每个个体提供的程序进行融合，并对个体解决方案进行有针对性的反馈可以帮助个体发现自身程序的缺陷，从而提高个体程序的质量。然而，随着群体规模的扩大与程序复杂性的提升，这样的程序融合过程难以通过人工的方式进行。因此，如何构造一个自动化的群体程序融合与反馈系统，成为实现上述过程的关键技术问题之一。

针对这一问题，本文基于群体智能的思想，设计了一种基于程序依赖图的程序融合与错误定位算法。该算法收集群体中每个个体的程序，将其转化为程序依赖图表示，并对程序依赖图进行融合。进一步地，根据融合后的程序依赖图，算法尝试对每个个体程序中可能存在的错误进行定位与反馈，以帮助个体发现并修复错误。

1.1.1　群体智能

群体智能源于科学家们对于自然界中一些低等生物群体的观察：它们每个个体具有的智能有限，往往只能完成一些单一的任务；然而在集群行动时，这些生物却显现出远超个体能力的复杂智能行为。例如蚂蚁会在觅食路径上遗留信息素，并根据路径上的信息素浓度决定行进方向。蚁群通过重复这一行为，最终能够正确选择出通往目标食物的最短路径 [2]。

群体智能的显现源于大规模群体在同一公共环境中进行频繁信息交互。基于这一原理，出现了利用计算机模仿生物群体的群体智能算法 [3,4,5]，以及在人类社会中应用相

同机制的面向人类群体的群体智能。互联网的发展促成了基于互联网的群体智能的出现。基于互联网的群体智能以通过互联网联结的大规模人群为基础，将每个个体的贡献聚合为巨大的信息整体。目前基于互联网的群体智能主要应用于知识收集与共享①、文本或图像识别 [6]、软件开发等 [7]。本文的工作针对软件开发的编码阶段，基于群体智能的思想，利用自动化的方法对群体开发者进行辅助。

1.1.2 群体协同软件开发

目前，群体协同的软件开发较为常见的两种形式为开源软件 [8]与软件众包 [6]。开源软件通过开源协议为用户赋予了软件代码的使用、分发、修改权 [9]。开源软件是开放式协同的典型例子 [10]：开源软件是由组织形式较为宽松的一组参与者在共通目标导向下合作而产生的产品。如今，在以 Github②为代表的代码托管平台与开源社区中，一个典型的开发模式是：开发者基于现有的开源软件代码创建一个分支，在该分支上对开源软件进行代码的修改，最后向开源软件所有者或管理者提交一个合并申请。所有者或管理者在进行审核后，决定是否将该开发者修改的代码引入开源软件本身 [11]。在软件众包中，发布者以奖励机制吸引多个开发者参与到针对某个问题的软件开发问题中。在开发者提供了各自的解决方案，即软件后，发布者对这些软件进行评估并排名，并继而根据其质量与排名发布奖励与决定采用。在软件众包中，开发者之间更多的是竞争关系而非协作关系，而发布者需要以人工的形式对各个解决方案进行评估。

在上述群体协同软件开发的形式中，群体中开发者的协作较为有限，而群体中信息的聚合过程则由单一的领导个体（开源软件的管理者或众包项目的发布者）人工实施。在这样的形式中，基于群体智能的思想，进一步引导群体中个体之间的协作与自动化群体中信息的聚合成为提升软件开发活动速度、提高产品质量的一个途径。

例如，在软件众包项目中，自动化的程序融合与反馈方法可以帮助项目发布者审阅每个参与者提供的程序。在每个参与者提交的程序均不完美的情况下，根据程序融合的反馈信息对其中的程序进行进一步优化，从而提高软件质量。一个类似的场景是，在编程教学的过程中，学生往往会通过在线评测系统进行编程的练习。对于学生而言，在练习过程中需要发现自身程序的错误。在在线评测系统中，针对同一题目进行过提交的学生构成了一个协同开发的群体。基于这一群体，通过群体程序融合与反馈的方法，可以对学生程序中的错误进行提示，以帮助学生发现并修改错误。

1.1.3 程序融合与错误定位

程序融合是将多个程序合并为一个整体的过程，其目标是将多个程序中相同或相近

① 维基百科：https://www.wikipedia.org/
② https://github.com/

的程序部分进行统一，同时使各自不同的部分保持分离。程序融合的结果是一个包含了所有个体程序的全部信息的整体。在群体协同软件开发的环境中，程序融合则是聚合群体信息的一种方式。

程序错误定位的目标是采用自动化的方法发现程序中可能存在的错误。由于部分程序错误涉及到对目标问题的理解，因此现有的错误定位方法大多依赖于测试用例对目标问题进行刻画。基于测试用例，自动化算法可以发现导致程序结果与预期不符的关键路径或程序节点，从而发现可能的错误位置。而在群体协同软件开发的环境中，群体智能的思想为错误定位提供了新的角度：即借助群体中其他个体提供的信息进行错误定位。对群体信息的利用使得错误定位方法可以减少或消除对测试用例的依赖。与其他群体智能的实例类似的，利用群体智能进行程序错误定位需要一种有效的群体信息收集与聚合方式，即程序融合。在群体协同开发的环境中，程序融合提供了一种群体信息的整合方式，而基于程序融合的错误定位则利用整合的群体信息对每个个体进行分析，以提高个体程序的质量。

1.2 相关研究工作

1.2.1 程序依赖图

M. J. Harrold 等人[12]提出了一种高效构造程序依赖图的算法。针对一个输入程序，该算法基于程序的抽象语法树先后构造程序依赖图的两个子图，即控制依赖子图与数据依赖子图。同时，该算法对不同的程序结构或语句形式对程序依赖图及程序依赖图生成算法的影响进行了分析。本文基于该算法实现了一个简化的针对 C 程序的程序依赖图生成工具。

Krinke[13]提出一种基于程序依赖图的程序中重复代码识别方法。该方法利用对程序依赖图中相似子图的识别，对同一程序中由复制粘贴后修改造成的相似代码进行识别。该方法所针对的场景是同一程序中的相似代码，即代码之间无论从形式上或语义上均具有较高的相似度，这使得程序依赖图的比较过程得以简化。

Chao Liu 等人[14]提出了一种利用程序依赖图进行代码抄袭检测的方法。利用程序依赖图不受一般的以对抗抄袭检测为目的的代码混淆（如变量名替换，语句顺序修改，无关代码插入等）影响的特点，该方法使用寻找程序依赖图之间的子图同构的方法判断代码抄袭。

上述两个方法能够发现程序依赖图之间的相同部分，然而只能进行程序依赖图之间的两两子图匹配。随着程序规模与数量的上升，由于子图同构问题本质的复杂度，两两匹配的方法无法适用于多图的情况，因而难以应用于本文的群体程序融合问题。

1.2.2 融合算法

Robert. A. Cochran 等人 [15]一种名为 Program Boosting 的正则表达式融合方法。该方法针对构建正则表达式的众包任务的场景，基于参与者提供的不完美的正则表达式，合成出一个拥有更高准确度的新的正则表达式。该方法使用符号自动机表示正则表达式，并在其基础上通过遗传编程进行正则表达式的合成。该方法在正则表达式构造问题上拥有较稳定的准确度提升效果，但无法应用于结构更加复杂的一般程序。

Horwitz 等人 [16]提出了一种基于程序依赖图的多版本程序合并算法。对于一个程序 A 的两个派生版本 B 与 C，该算法利用一种与程序依赖图相似的程序表示方法，将派生版本 B 与 C 合并为一个新的程序版本 M，M 中同时包含了 B 与 C 对程序 A 所做的修改。该算法实现了两个程序的融合，对于多个程序融合问题而言，只能通过依次两两相融的方式进行。另外，参与融合的程序限定为同一程序的不同派生版本，这使得该算法要求被融合程序在形式上与结构上均具有极高的相似性。

Kuchaiev 等人 [17]提出了一种基于网络拓扑结构的网络对齐算法。网络对齐即对网络的图结构进行比较。算法构建了节点数为 5 以下的所有基础连通图形式。基于这些基础连通图，算法对两个不同图的节点间局部拓扑结构相似度进行比较，进而采用贪心法与二部图匹配等方法对两个图的整体相似度进行比较。该算法在图匹配过程中利用了图节点的邻近拓扑结构，但该算法只限定于对两图进行匹配，无法扩展于多图匹配的情况。

1.2.3 错误定位

传统的程序错误定位方法包括：Mark Weiser 提出的程序切片技术 [18]与 Janusz W. Laski 等人提出的动态切片技术 [19]。程序切片或动态切片可以提取程序中与指定程序位置或指定执行路径相关联的程序局部。Renieres 与 Reiss 提出了基于程序频谱的错误定位技术 [20]，其通过分化通过的测试用例与失败的测试用例识别导致测试用例失败的可疑程序片段。这些方法均依赖于通过与失败的测试用例，因此其定位质量也依赖于测试用例质量。

Weimer 等人 [21]提出了一种基于演化计算的自动化程序错误修复算法。针对一个含有错误的 C 程序，该算法以一组能够描述该程序功能的测试用例以及一个能体现出目标错误的测试用例为输入，通过分析测试用例的执行路径发现错误位置，并尝试以程序中其他位置的代码片段构造一个错误修复。该算法同样依赖于测试用例，且对测试用例的质量有较强依赖。且该算法假设能够修复错误的代码可以通过程序中已有片段构造而出，使得该算法能够修复的错误类型较为有限。

Suguna 与 Chandrasekaran 提出了一种基于概率程序依赖图的错误定位算法 [22]。该算法基于程序依赖图与一组测试用例分析程序节点状态之间的依赖关系，从而生成概

率程序依赖图，进而利用程序状态间的概率信息对节点的可疑程度进行排序。同样的，该算法对测试用例有较强依赖，同时测试用例可能影响概率程序依赖图的结构，从而影响定位结果。

Chun-Hung Hsiao 等人 [23]提出了一种利用 n-gram 模型与互联网程序语料库的程序分析方法。该方法设计了一种面向程序代码的 n-gram 模型，并基于从互联网提取 280 万个 JavaScript 程序，利用 tf-idf 度量构建了一个包含程序片段重要性信息的程序语料库。以此为基础，该工作提出了一种程序抄袭检测方法与代码克隆缺陷检测方法。

1.3　本文主要解决的问题

由于程序错误中许多错误与程序所解决的目标问题相关联，因此程序错误定位方法往往需要以一定的方式对目标问题本身及其求解方式进行理解，进而发现程序中的错误。由于构造测试用例刻画程序行为远易于采用计算机可理解的方式描述目标问题，因此现有的自动化程序错误定位方法大多基于测试用例，而测试用例的数量与质量则决定了错误定位方法的准确性。但是，对于计算机或自动化算法而言难以实现的目标问题理解，在群体协作的环境下可以由群体中的每个个体开发者完成。在群体协作的环境下，当一个开发者对目标问题的理解或思考出现偏差或错误，使得其编写的程序出现错误时，这一错误可能能够借助群体中其他开发者对目标问题的正确理解或思路解决。本文的目标即为设计一个自动化的方法，在群体协同开发的环境下，辅助群体中的每个开发者利用群体提供的信息发现自身程序中的错误。

在群体协同开发的环境下，令群体中的开发者相互协助的一个方式是将群体提供的信息进行有效融合，并基于融合结果对群体中的每个个体进行反馈，从而促进个体对自身解决方案的改进。具体地，在多人共同编程求解同一问题的场景下，实现群体信息融合与反馈的方式之一就是对群体程序进行融合，并根据融合结果对每个个体进行错误定位。

程序融合的过程的主要目标是尽可能将每个个体程序中相似的语句融合在一起。在程序中，语句的相似主要体现在语句自身语义的相似，以及语句之间关系（即语句上下文）的相似。为了更好地将群体程序进行融合，程序融合时需要一种能够提取程序逻辑结构与语句关系的程序表示方法，并设计相应的程序语句相似度的计算方法。程序融合的过程是一个复杂的组合优化问题。为了寻找融合问题的最优解，需要设计适当的融合方案定量评估方法，并设计一个高效的算法寻找最优融合方案。最终，以融合结果为依据，对个体程序进行分析，并形成反馈。

针对上述程序融合与反馈问题，本文主要做了以下工作：

1)　提出了一种以程序依赖图作为融合时程序的表示方法，以广义熵作为融合效果

的定量评估标准，基于熵最小化原则的程序融合算法。

2) 提出了基于程序融合结果的错误定位算法，该算法基于群体共识思想，根据融合程序与个体程序的差异性进行反馈。

3) 设计了基于同构程序的融合实验。实验对不同规模的程序在多种大小的群体中进行融合，并对融合结果进行评估，以验证程序融合算法的有效性。设计了基于程序变异的仿真群体错误定位实验与基于实际群体程序的错误定位实验，并对实验结果进行了评估，以验证错误定位算法的有效性。

1.4 本文的组织

本文余下部分共五个章节：

第二章介绍程序融合与错误定位的基本方法与核心概念。本章首先对本文所采用的基于程序依赖图的程序融合算法与基于程序依赖融合图的错误定位方法进行了概述。之后介绍了本文中使用的程序依赖图与程序依赖融合图相关的概念。

第三章介绍程序依赖图融合算法。本章描述了基于遗传算法的融合算法的设计，包括遗传算子的设计、节点相似度的计算、基于广义熵的适应度函数设计等。

第四章介绍基于本文采用的基于群体共识的错误定位方法。本章对这一方法的思想进行介绍，之后描述了本文提出了三种基于该思想的错误定位方法。

第五章对本文提出的算法进行了实验。本章介绍三种实验以验证融合与反馈算法的有效性，并对实验结果进行评估与总结。

第六章对本文工作进行了总结，并提出进一步改进方向。

第二章　基于程序融合的错误定位

在群体协同开发过程中，群体中每个开发者会针对同一问题，提交各自的解决方案，即一段试图解决目标问题的程序。然而，由于编程问题的复杂性与开发者自身能力的局限性，问题往往由于程序中存在错误而无法一次性得以解决。针对这样的场景，程序融合与反馈的思想是：尽管每个个体的程序可能是不完美的，但群体中各个开发者所提交的程序中蕴含的信息可能能为其他开发者提供辅助。将群体的信息进行融合，并根据群体信息对每个个体解决方案进行分析，进而形成反馈，则可能帮助个体开发者在更短时间内发现并修复自身的错误，进而提高软件开发效率。而这一过程的核心即为程序融合。

对于程序融合过程，本文提出了一种基于程序依赖图的程序融合算法。算法采用程序依赖图作为程序的中间表示形式，基于熵最小化原则，使用遗传算法在融合状态空间中寻找最优的融合方案。

本章对本文提出的基于程序依赖图融合的错误定位方法进行概述，并对后文使用的一些概念与定义进行介绍。本章各节内容安排如下：2.1 节对本文提出的程序融合与反馈算法框架进行概述；2.2 节介绍程序融合时使用的程序依赖图与程序依赖融合图定义，并描述了其生成算法；2.3 节介绍用于程序依赖图融合质量度量的广义熵；2.4 节对本章进行总结。

2.1　基于程序融合的错误定位框架

本文提出的程序依赖图融合与错误定位方法由三部分组成，即：程序依赖图生成、程序依赖图融合、融合信息反馈。图 2.1 所示是程序融合与反馈算法的主要步骤：

1) 程序依赖图生成：算法首先收集群体针对目标问题提交的解决方案，即一组程序代码。程序依赖图生成部分将群体中所有程序转换为程序依赖图表示。此部分首先对代码进行预处理，之后利用词法、语法分析工具生成代码对应的抽象语法树，最后基于抽象语法树生成程序依赖图。
2) 程序依赖图融合：程序融合是本文工作的核心，融合算法以一组程序依赖图作为输入，基于遗传算法框架，使用基于广义熵的适应度函数，寻找一个最优的融合结果，即一个程序依赖融合图。
3) 融合信息反馈：以程序依赖图融合部分得到的融合结果为基础，融合信息反馈部分基于群体共识思想，分析每个个体程序依赖图与融合方案之间的差异性，对每个个体程序中可能的错误进行定位并进行反馈。

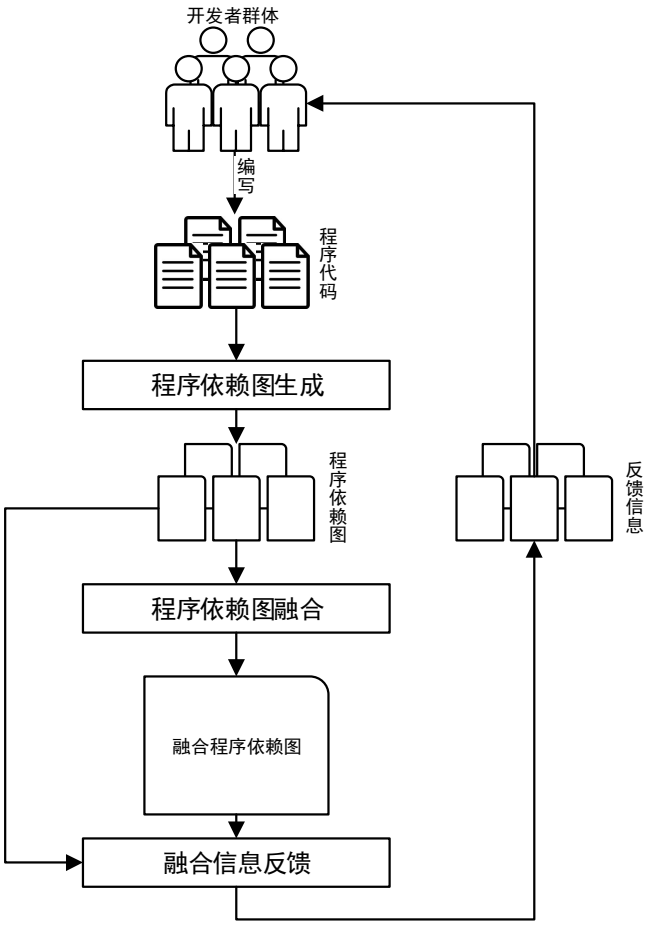


图 2.1 基于程序依赖图融合的错误定位流程图

2.2 核心概念

2.2.1 程序依赖图及程序依赖融合图

程序融合的目标是将多个程序中，具有相同或相似语义的程序语句融合在一起，同时使语义不同的语句保持分离。直观上，不同程序中的程序语句的语义相似是指这些语句在各自所在程序的执行过程中起到相似的作用。具体而言，程序语句的相似性体现于两个方面：一是程序语句自身具有形式上的相似性，二是程序语句和与其相关的其他语句之间关系上的相似性。对于多个由不同开发者编写的程序，以一般的代码形式描述时，即使程序逻辑结构相似，也会因开发者个人习惯的不同而在表现形式上呈现出较大的差异性。另一方面，代码形式描述的程序无法显式表述程序语句之间的关系，难以区分程序语句间由关系不同而引起的差异。这使得在程序融合过程中，若对程序采用代码形式表示，则难以准确衡量语句之间的语义相似度，进而难以获得较优的融合结果。

程序依赖图 (Program Dependence Graph, PDG) 是一种用于描述程序结构的图。程序依赖图以程序节点或谓词表达式（或者更细粒度地，运算符与运算数）作为节点。节点之间的边代表节点语句所依赖的数据关系（数据依赖），或节点执行所依赖的控制条件（控制依赖）[24]。程序依赖图显式描述了程序语句之间的依赖关系，这些依赖关系只确定了为保证程序正确执行所必须的语句间的顺序关系，这使得程序依赖图能够以最少的限制刻画一个程序的行为。目前，程序依赖图的主要应用场景有程序优化、抄袭检测、程序切片等。

与代码形式相比，程序依赖图显式描述了程序语句间的关系信息，同时能够隐去对程序执行不产生影响的代码形式。与程序的其他表示形式，如抽象语法树（Abstract Syntax Tree, AST），控制流图（Control Flow Graph, CFG）等相比，控制依赖图蕴含了为还原程序执行过程所必须的所有信息[25]，具有更强的表述能力。因此本文采用程序依赖图作为程序融合时的程序表示方法。

本节介绍本文所使用的程序依赖图与程序依赖融合图及相关概念的定义，并描述了本文所采用的程序依赖图构造方法。

1)　程序依赖图结构

程序依赖图由一组表示程序语句的节点与一组表示节点之间的依赖关系的有向边组成。程序中每一个基本语句，如声明语句、赋值语句等，对应程序依赖图中的一个节点。此外，程序依赖图还包含一个用于标识程序入口的入口节点。每个节点拥有节点类型，且每个节点有且只有一种节点类型。表 2.1 列举了本文采用的节点类型及对应语句的描述。

表 2.1　节点类型与对应语句

节点类型	描述
ENTRY	程序入口节点
DECL	声明语句
ASSIGN	赋值语句
CONTROL	控制语句，如分支、循环语句等
CALL	函数调用语句
RETURN	返回语句

程序依赖图节点间的边由程序语句间的依赖关系确定。程序语句间的依赖关系分为控制依赖与数据依赖。

控制依赖描述了程序语句执行过程中产生的依赖关系。

定义（控制依赖, Control Dependence）：对于程序语句s_1与s_2，若s_1是控制语句，且在程序执行过程中，s_1的执行结果确定s_2是否会被执行，则称s_2控制依赖于s_1。

数据依赖描述了程序语句对变量的使用与赋值时产生的数据流关系。

定义（数据依赖, Data Dependence）：对于程序语句s_1与s_2，若存在变量x满足：

1) x在s_1中被赋值。
2) x的值在s_2中被使用。
3) 程序中存在一条由s_1到s_2的执行路径，且路径上s_1、s_2之外的程序语句均不对x进行赋值。

则称s_2数据依赖于s_1。

额外地，为使程序依赖图结构更加清晰并便于后续融合，本文在程序依赖图中引入第三种依赖关系：声明依赖。声明依赖是一种特殊的数据依赖，其含义是所有对某变量进行赋值的语句应对该变量的声明语句产生依赖关系。其定义如下：

定义（声明依赖, Declare Dependence）：对于程序语句s_1与s_2，若s_1是变量声明语句，s_1声明了变量x，且s_2对变量x进行了赋值，则称s_2声明依赖于s_1。

对于程序语句s_1与s_2，若s_2控制依赖于s_1，则在对应的程序依赖图中，对于s_1与s_2对应的程序依赖图节点v_1与v_2，存在一条由v_1指向v_2的控制依赖边。数据依赖、声明依赖同理。

定义（程序依赖图）：程序 P 的程序依赖图是一个六元组$G = (L_v, L_e, V, E, r_v, r_e)$，其中：

- $L_v = \{Entry, Decl, Assign, Control, Call, Return\}$ 是节点类型集合。
- $L_e = \{Control\ \ Dependence, Data\ \ Dependence, Declare\ \ Dependence\}$ 是边类型集合。
- V 是节点集合，也以$V(G)$表示。每个节点对应程序入口节点或 P 中的一个语句
- $E \subseteq V \times V$ 是边集合，也以$E(G)$表示。每条边是一个有序二元组$\langle s, t\rangle$，其中$s, t \in V$，第一个节点 s 为该边的源节点，记为$src(e)$，第二个节点 t 为该边的目标节点，记为$tgt(e)$。
- $r_v : V \to L_v$ 是节点类型映射函数。该函数为每一个节点赋予一个唯一的类型。为方便阅读，对于一个节点v，将该节点的类型记为$type(v)$。
- $r_e : E \to L_e$ 是边类型映射函数。该函数为每一个边赋予一个唯一的类型。为方便阅读，对于一个边e，将该边的类型记为$type(e)$。
- 边集合 E 满足：$\forall e_1 \neq e_2 \in E, if\ \ src(e_1) = src(e_2)\ \ and\ \ tgt(e_1) = tgt(e_2),\ then\ \ type(e_1) \neq type(e_2)$，即同一节点之间不存在两条同向且类型相同的依赖边。

由定义可知，程序依赖图是一个带标签的多重图，其允许两个节点间存在同向的多条边，但这些边必须具有不同的类型。

对于一个节点v或一条边e，令$graph(v)$或$graph(e)$表示该节点或该边所属的程序依赖图。即：$\forall v \in V(G), graph(v) = G. \forall e \in E(G), graph(e) = G$。

图 2.2 是一个计算累加和的程序及其对应的程序依赖图。图中节点内标识了节点对应的语句内容，边通过标签标注了边类型。

2)　程序依赖图生成算法

本文采用的程序依赖图生成算法基于现有工作 [12]。通用的程序依赖图的生成的过程可分如下几个步骤：

1) 对程序进行预处理，对特殊形式的语句（如++、+=）等进行规范化。
2) 对程序代码进行词法、语法分析，获得程序对应的抽象语法树。
3) 基于抽象语法树，进行控制流分析，获得程序对应的控制流图。此步骤同时为每一条程序语句生成了一个节点，这些节点最终成为程序依赖图中的节点。
4) 基于控制流图，进行数据流分析。这个步骤主要进行程序语句对变量的定义-使用分析与基于程序控制流的到达-定义分析。
5) 基于抽象语法树结构与控制流图结构，根据定义可以获得节点间的控制依赖关
6) 系；根据数据流分析结果，可以获得每个程序语句中使用的变量的值的来源信息，进而根据定义可以构造出节点间的数据依赖关系与声明依赖关系。
7) 根据上述步骤所得节点与依赖关系构建程序依赖图。

本文基于该算法，利用开源 C 程序分析工具 pycparser[①]作为词法语法分析工具，实现了一个面向 C 语言程序的程序依赖图生成工具，并使用该工具生成的程序依赖图进行后续程序融合与错误定位工作。

① https://github.com/eliben/pycparser

```
int main() {
    int i, sum;
    sum = 0;
    for (i = 1; i < 10; i = i + 1) {
        sum = sum + i;
    }
    printf("%d\n", sum);
    return 0;
}
```

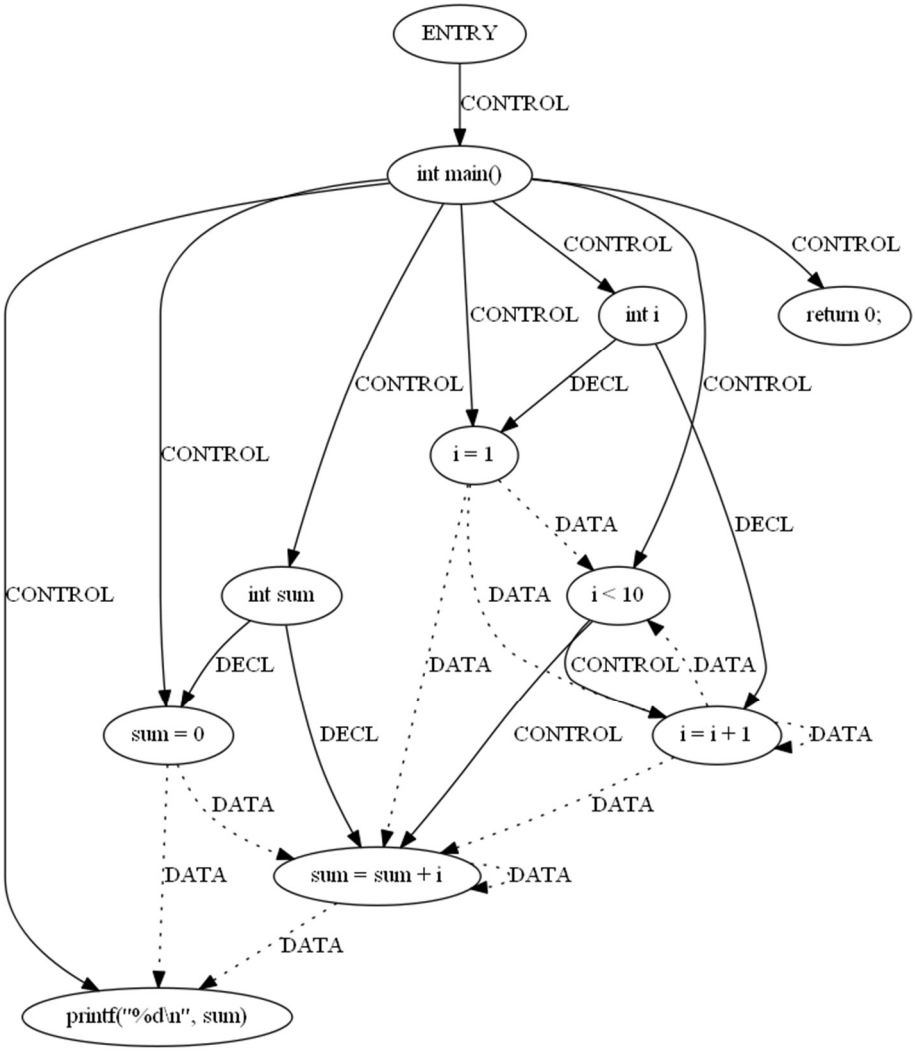


图 2.2　一个示例程序及其程序依赖图

3)　程序依赖融合图

对于一组程序依赖图，其融合结果是一个带标签的有向图，即程序依赖融合图。为使表述清晰，本文在叙述融合相关概念与算法时，将参与融合的程序依赖图称作被融合图，其中的节点与边分别称为被融合节点与被融合边，程序依赖融合图也简称为融合图。

定义（程序依赖融合图）：对于一组程序依赖图

$\mathrm{G}=\{g_i|i=1,2,\ldots,n,g_i=(L_{vi},L_{ei},V_i,E_i,r_{vi},r_{ei})\}$，程序依赖融合图$\mathcal{G}$是一个三元组$(\mathcal{V},\mathcal{E},\delta)$，其中：

- $\mathcal{V}\subseteq\mathbb{P}\left(\bigcup_{i=1}^{n}V_i\right)$ 是融合节点集合，记为$V(\mathcal{G})$。每一个融合节点是一个集合，表示被融合在一起的一组被融合节点。
- $\mathcal{E}\subseteq\mathbb{P}\left(\bigcup_{i=1}^{n}E_i\right)$ 是融合边集合，记为$E(\mathcal{G})$。每一个融合边是一个集合，表示被融合在一起的一组被融合边。
- $\delta:\mathcal{E}\to\mathcal{V}\times\mathcal{V}$ 是边的源节点与目标节点映射函数。对于一个融合边$e^m\in\mathcal{E}$，$\delta(e^m)$是一个二元组$\langle v_s^m,v_t^m\rangle$，其中$v_s^m,v_t^m\in\mathcal{V}$。$v_s^m$称为$e^m$的源节点，记为$src(e^m)$，$v_t^m$称为$e^m$的目标节点，记为$tgt(e^m)$。
- $\mathcal{G}$满足如下约束：

1) $\forall\boldsymbol{v^m}\in\mathcal{V},|\boldsymbol{v^m}|>0$。$\forall e^m\in\mathcal{E},|e^m|>0$。
   即，任意融合节点与融合边非空。

2) $\forall v\in\bigcup_{i=1,2,\ldots,n}V_i,\exists\boldsymbol{v^m}\in\mathcal{V}\ s.t.v\in v^m$

   $\forall e\in\bigcup_{i=1,2,\ldots,n}V_i,\exists e^m\in\mathcal{E}\ s.t.e\in e^m$

   即，任意的被融合节点均被包含在某个融合节点中，任意的被融合边均被包含在某个融合边中。

3) $\forall\boldsymbol{v_1^m},\boldsymbol{v_2^m}\in\mathcal{V},\boldsymbol{v_1^m}\cap\boldsymbol{v_2^m}=\emptyset$
   $\forall e_1^m,e_2^m\in\mathcal{E},e_2^m\cap e_2^m=\emptyset$
   即，任意两个融合节点交集为空，任意两个融合边交集为空。即，任意被融合节点不能同时属于两个融合节点，任意被融合边不能同时属于两个融合边。

4) $\forall\boldsymbol{v^m}\in\mathcal{V},\forall v_1\neq v_2\in\boldsymbol{v^m},graph(v_1)\neq graph(v_2)$
   $\forall e^m\in\mathcal{E},\forall e_1\neq e_2\in e^m,graph(e_1)\neq graph(e_2)$
   即，任意融合节点中，不存在两个来自同一被融合图的被融合节点，任意融合边中，不存在两个来自同一被融合图的被融合边。

5) $\forall\boldsymbol{v^m}\in\mathcal{V},\forall v_1,v_2\in v^m,type(v_1)=type(v_2)$
   $\forall e^m\in\mathcal{E},\forall e_1,e_2\in e^m,type(e_1)=type(e_2)$
   即，任意融合节点中的所有被融合节点具有相同类型，任意融合边中的所有被融合边具有相同类型。此时，将该类型称为融合节点或融合边的类型，记为$type(v^m)$或$type(e^m)$。

6) $\forall e^m\in\mathcal{E},\forall e\in e^m,src(e)\in scr(e^m),tgt(e)\in tgt(e^m)$
   即，对于一条融合边中的任意被融合边，其源节点一定被融合于融合边的源节

点，其目标节点也被融合于融合边的目标节点。

7) $\forall e_1^m, e_2^m \in \mathcal{E}, if\ \ \delta(e_1^m) = \delta(e_2^m)\ \ and\ \ type(e_1^m) = type(e_2^m)\ \ then\ \ e_1^m = e_2^m$

即，任意两融合节点间，不存在同向且类型相同的两条融合边。

约束 2、3 限定了，任意被融合节点都被融合到唯一的融合节点中，任意被融合边都被融合到唯一的融合边中。约束 4 至 7 限定了融合边的结构。

程序依赖融合图的结构约束使其存在如下性质：

性质：对于由给定的一组被融合图融合而成的程序依赖融合图，其边集构成可由节点集构成唯一确定。

根据该性质，对于一个程序依赖融合图，根据其融合节点集与上述约束，可以计算出一组唯一满足约束的融合边，这些融合边构成了该程序依赖融合图的融合边集合。该性质使得使用融合节点集足以描述程序依赖融合图的结构。由此，本文在后文中提出了基于节点集结构的程序依赖融合图表示方法，并将其作为程序融合过程中的融合图编码方式。

通过将程序以程序依赖图形式表示，程序融合问题即转化为程序依赖图融合问题，即：给定一组程序依赖图作为被融合图，寻找一个满足约束的程序依赖融合图，即一个合法的融合方案。同时，程序融合希望找到一个“最优”的融合方案，本文采用广义熵作为融合方案质量的度量标准

2.2.2 基于广义熵的融合质量度量

程序融合（或程序依赖图融合）的目标是尽可能将语义上相似的程序语句对应的节点相融合，而使语义不同的节点保持分离。这一目标是希望最大程度地提取群体在程序片段上的共识并保持群体不同意见间的分歧。因此融合方案质量的度量实际上是对融合方案的共识程度（或冲突程度）进行度量。由于程序依赖融合图是基于融合节点而组成，因此问题也转化成对融合节点共识程度（冲突程度）的度量。本文基于王诗君提出的广义熵概念 [26]来度量融合节点的冲突程度。广义熵越低，则认为融合的效果越好。

1) 广义熵

广义熵用于刻画一组个体针对某一问题的观点的冲突程度。在广义熵度量中，个体的观点采用离散随机变量 X 表示，其取值空间以 $\chi = \{x_1, x_2, \ldots, x_n\}$ 表示，其中 $x_1, x_2, \ldots, x_n$ 表示个体的不同观点。其概率密度函数 $p(x) = P\{X = x\}, x \in \chi$ 则表示了群体中个体持有观点 x 的概率。基于此概率密度函数，群体在观点 X 上的广义熵定义为：

$$\mathcal{H}(X) = -\sum_{k=1}^{n} p(x_k) \log \left(\sum_{t=1}^{n} p(x_t) s(x_k, x_t) \right) \tag{2.1}$$

其中，$s(x_k, x_t)$ 表示观点 x_k 与 x_t 的相似度。随着问题的不同，观点相似度拥有不同的计算方法。第三章将介绍本文采用的程序节点关系相似度与内容相似度计算方法。当群体针对问题的观点冲突程度较高时，广义熵值较大；当群体观点冲突程度较低时，广义熵值较小；当群体观点完全达成一致时，广义熵值为 0。

2)　融合节点的广义熵

根据广义熵的概念，在程序依赖融合图中，可以为融合节点定义广义熵。针对程序依赖图节点，群体中个体的观点可以分类为两个方面：1、对于节点内容，即程序语句的观点；2、对于节点上下文关系的观点，即对于节点入边、出边情况的观点。因此，融合节点的广义熵也来自于两部分，节点内容的广义熵与节点关系的广义熵。

对于节点内容，广义熵能够衡量群体在程序语句形式上的观点分歧程度。具体地，对于由被融合节点 $v_1, v_2, \ldots, v_n$ 融合而成的融合节点 $v^m = \{v_1, v_2, \ldots, v_n\}$，令 $S(v_i)$ 表示被融合节点 v_i 所对应的程序语句，$p(v_i)$表示群体中认同节点v_i的个体的概率。则融合节点 v^m 的内容广义熵可定义为：

$$\mathcal{H}_c(v^m) = -\sum_{k=1}^{n} p(v_k) \log \left(\sum_{t=1}^{n} p(v_t) Sim(S(v_k), S(v_t)) \right) \tag{2.2}$$

其中，$Sim(S(v_k), S(v_t))$表示节点v_k与v_t内容的相似度，其计算方法将在第三章介绍。

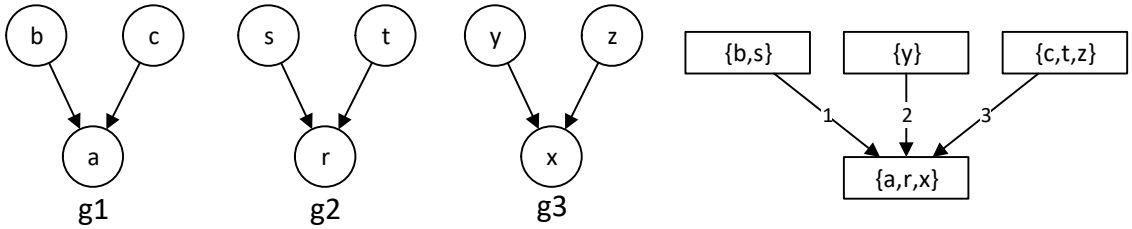
对于节点上下文关系，广义熵能够衡量群体在语句节点与其他节点间的边组成上的观点分歧程度。在这里，观点被定义为“当前节点的被融合边被融合到哪些融合边中”。具体地，假设被融合节点 $v_1, v_2, \ldots, v_n$ 融合为融合节点 $v^m = \{v_1, v_2, \ldots, v_n\}$，对于其中一个被融合节点$v_i$，$v_i$在其所属被融合图中的入边与出边集合分别为$\{e_{i1}, e_{i2}, \ldots, e_{ik}\}$与$\{e_{o1}, e_{o2}, \ldots, e_{ol}\}$。这些边分别被融合到了融合边$\{e^m_{i1}, e^m_{i2}, \ldots, e^m_{ik}\}$与$\{e^m_{o1}, e^m_{o2}, \ldots, e^m_{ol}\}$中，则$\{e^m_{i1}, e^m_{i2}, \ldots, e^m_{ik}\}$即为$v_i$对应个体在当前节点入边组成上的观点，$\{e^m_{o1}, e^m_{o2}, \ldots, e^m_{ol}\}$则是其在出边构成上的观点。令入边组成观点构成的集合为$X_{in} = \{\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_s\}$，$p(\mathcal{E}_i)$表示群体中在入边组成上持有观点$\mathcal{E}_i$的个体的概率。$Sim(\mathcal{E}_i, \mathcal{E}_j)$表示$\mathcal{E}_i$与$\mathcal{E}_j$的相似度，其计算方式为：

$$Sim(\mathcal{E}_i, \mathcal{E}_j) = \frac{|\mathcal{E}_i \cap \mathcal{E}_j|}{|\mathcal{E}_i \cup \mathcal{E}_j|} \tag{2.3}$$

融合节点v^m入边关系广义熵可定义为：

$$\mathcal{H}_{in}(v^m) = -\sum_{k=1}^{n} p(\mathcal{E}_k) \log \left(\sum_{t=1}^{n} p(\mathcal{E}_t) Sim(\mathcal{E}_k, \mathcal{E}_t) \right) \tag{2.4}$$

类似的，可以定义融合节点v^m出边关系广义熵$\mathcal{H}_{out}(v^m)$。



(a) 被融合图　　(b) 融合图

图 2.3　融合节点的关系广义熵示例

以图 2.3 为例。图 2.3 (a)中的三个被融合图g_1, g_2, g_3被融合为图 2.3 (b)所示的融合图。融合节点$v^m = \{a, r, x\}$有 3 条入边，根据图中编号分别记为e_1^m, e_2^m, e_3^m。g_1的a节点有 2 条入边，分别被融入了融合边e_1^m与e_3^m，则g_1在v^m入边关系上的观点为$\{e_1^m, e_3^m\}$。类似的，g_2在v^m入边关系上的观点为$\{e_1^m, e_3^m\}$，g_3的观点为$\{e_2^m, e_3^m\}$。群体中入边组成观点的集合为$X_{in} = \big\{\{e_1^m, e_3^m\}, \{e_2^m, e_3^m\}\big\}$, 其概率分布为：

$\mathcal{E}_i$	$\{e_1^m, e_3^m\}$	$\{e_2^m, e_3^m\}$
$p(\mathcal{E}_i)$	$\frac{2}{3}$	$\frac{1}{3}$

而$Sim(\{e_1^m, e_3^m\}, \{e_2^m, e_3^m\}) = \frac{|\{e_3^m\}|}{|\{e_1^m, e_2^m, e_3^m\}|} = \frac{1}{3}$ ，故v^m入边关系广义熵为：

$$\mathcal{H}_{in}(v^m) = -\frac{2}{3}\log\left(\frac{2}{3} + \frac{1}{3}\times\frac{1}{3}\right) - \frac{1}{3}\log\left(\frac{1}{3} + \frac{2}{3}\times\frac{1}{3}\right) \approx 0.085 \tag{2.5}$$

融合节点v^m的边关系广义熵定义为其入边关系广义熵与出边关系广义熵之和，即：

$$\mathcal{H}_e(v^m) = \mathcal{H}_{in}(v^m) + \mathcal{H}_{out}(v^m) \tag{2.6}$$

融合节点的整体广义熵定义为其节点内容广义熵与边关系广义熵之和，即：

$$\mathcal{H}(v^m) = \mathcal{H}_c(v^m) + \mathcal{H}_e(v^m) \tag{2.7}$$

3)　程序依赖融合图的广义熵

由被融合图集合$\mathrm{G} = \{g_1, g_2, \ldots, g_n\}$融合而成的程序依赖融合图$\mathcal{G} = (\mathcal{V}, \mathcal{E}, \delta)$的广义熵定义如下：

$$\mathcal{H}(\mathcal{G}) = |\mathcal{E}| \cdot \sum\nolimits_{v^m \in \mathcal{V}} \big(\mathcal{H}(v^m) + f(v^m)\big) \tag{2.8}$$

其中，$f(v^m)$为修正函数。由于融合节点自身的广义熵只刻画了节点内部的观点冲突程度，因此若一个融合节点中只包含了一个被融合节点（即该节点未与其他节点融合），则该融合节点内不存在观点冲突，其广义熵为 0。在基于广义熵的融合效果评判中，广义熵越低，则认为融合效果越好。因此单纯基于广义熵来看，未融合节点的融合效果会被认为最好，这与程序融合过程中希望节点尽可能融合的目标相悖。因此，在融合图的广义熵计算中，引入修正函数对未融合节点进行“广义熵惩罚”，修正函数定义如下：

$$f(v^m) = \Delta - \Delta^{\frac{|v^m|}{|\mathrm{G}|}} \tag{2.9}$$

其中$|\mathrm{G}|$为被融合图数，$\Delta = 2\log_2(|\mathrm{G}|)$为常量，表示一个融合节点可能拥有的最大广义熵值。对于一组被融合节点，当它们被平均地融合到多个融合节点时，它们的修正函数值较高；当他们集中地融合到少量融合节点时，它们的修正函数值较低。

在后文介绍的融合算法中，采用程序依赖融合图的广义熵作为优化目标函数，可以使得算法趋向于将尽可能多的相似节点相融合，以提高程序依赖图的融合程度，优化融合结果。

2.3　本章小结

本章首先介绍了基于程序融合的错误定位的基本框架。基于程序融合的错误定位主要由三个步骤组成，即程序依赖图生成、程序依赖图融合、融合信息反馈。之后介绍了基于程序融合的错误定位中核心部分——程序融合中的核心概念，包括程序依赖图、程序依赖融合图与广义熵的概念。

第三章　基于遗传算法的程序依赖图融合

程序融合是将多个程序中，语义相同或相似的语句进行合并，而令语义不同的语句保持分离。程序融合的目的在于发现多个程序中存在的共性与个体程序之间的差异。基于程序依赖图的程序融合是以程序依赖图作为程序的表示形式，对多程序依赖图进行融合。使用程序依赖图作为程序的表示形式，使得程序融合转化为一个带标签有向图的多图融合问题。

多图融合问题本质上是一个组合优化问题，其目标是在大量合法的融合方案中寻找一个最优的融合方案。对程序融合问题而言，融合方案的质量则是根据语义相同或相似的程序语句是否被正确地融合在一起，而不相似的语句是否被正确分离而评判。本文使用广义熵作为融合方案的定量评判标准。广义熵刻画了融合在一起的程序语句的冲突程度，广义熵越低，则融合在一起的程序语句越相似，则认为融合方案越好。

在程序融合问题中，由于问题的解空间规模会随程序数量的提升与程序规模的扩大呈指数级增长，枚举遍历寻找最优解的解法不具备现实上的可行性。因此本文设计了一种基于遗传算法的程序依赖图融合算法。算法设计了针对程序依赖图融合问题的编码方案、交叉变异算子、基于广义熵的适应度函数等。

本章各节安排如下：3.1 节介绍基于遗传算法的程序依赖图融合的基本框架。3.2 节介绍遗传算法中使用的程序依赖融合图表示方法。3.3 节介绍基于本文提出的编码方案的交叉运算设计。3.4 节介绍变异运算设计。3.5 节程序依赖图节点相似度计算方法及基于广义熵的适应度函数设计。3.6 节介绍遗传算法中其他步骤的设计。3.7 节对本章进行总结。

3.1　程序依赖图融合基本框架

程序依赖图融合问题是一个针对带标签有向图的多图融合问题。而多图融合是一个复杂的组合优化问题，随着图的数量规模的扩大与每个程序依赖图内节点数量的增多，问题的状态空间呈指数增长，在问题规模较大的情况下，简单的枚举遍历方式无法在可接受时间内找到最优解或较优解。因此，本文设计了一种基于遗传算法的程序依赖图融合算法，算法以熵最小化为目标，寻找一个最优的融合方案。

遗传算法是一种用于解决最优化问题的启发式搜索算法。遗传算法借鉴了自然界中生物的自然选择与遗传进化过程，通过模拟生物的进化实现对最优解的搜索 [27]。在遗传算法中，目标问题的解称为“个体”，而解的特征通过个体的“染色体”表现，对于不同问题，染色体有不同的表达形式。一个解的质量通过“适应度”度量。遗传算法维持一组用于搜索更多解的现有解，这一组现有解，或现有个体，称为“种群”。在遗传

算法的过程中，通常包含如下几个关键操作

1) 选择：模拟自然界中的自然选择过程，以个体适应度为基准，选择种群中参与交叉过程的个体，并对低适应度个体进行淘汰。
2) 交叉：模拟自然界中的生物交配过程，以两个或多个亲本个体为基础，通过染色体操作产生新的个体。
3) 变异：模拟自然界中的基因突变过程，对个体的染色体进行随机的修改，使个体的形状发生改变。
4) 种群多样性调整：模拟自然界中种群的基因多样性，对种群进行筛选，去除有重复基因型的个体并以新的个体补充。

一个一般的遗传算法流程如图 3.1 所示。

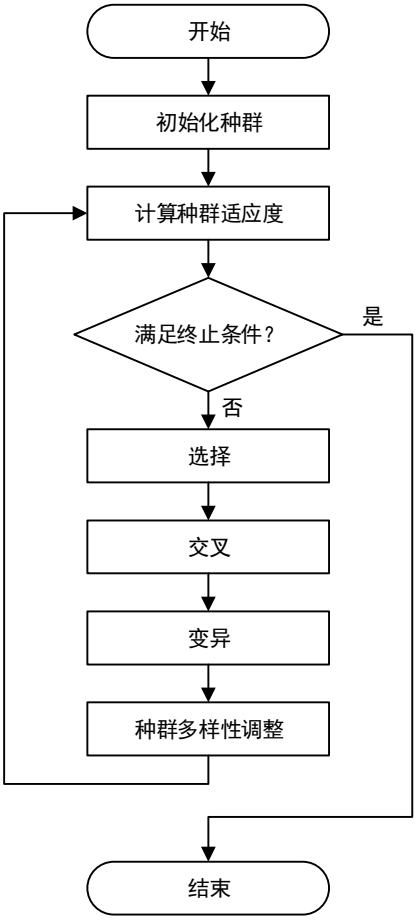


图 3.1　遗传算法流程图

本文采用遗传算法对一组程序依赖图的最佳融合方案进行搜索。算法以一组程序依赖图为输入，以融合方案、即程序依赖融合图作为遗传算法中种群的个体。算法设计用于遗传算法程序依赖融合图的表示方法、交叉与变异操作。基于遗传算法框架，算法以

广义熵为指导，通过演化寻找一个“足够优”的程序依赖融合图，作为最优融合方案输出。遗传算法具有很强的可并行性。本文在实现中对遗传算法进行了并行化处理，以提高算法的运行效率。

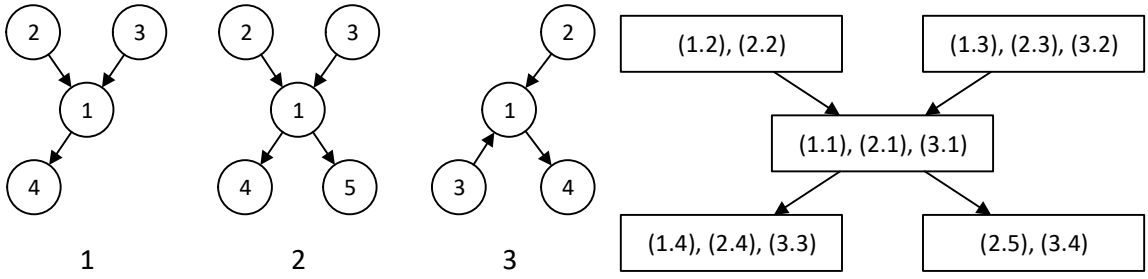
3.2 程序依赖融合图表示方法

在遗传算法中，通常使用某种编码方式对个体的染色体进行表示，进行编码的目的在于方便交叉与变异运算、方便编程实现等。现有常见的遗传算法编码方案有二进制编码、序列编码、值编码、矩阵编码、树形编码等。然而，由于程序依赖融合图结构的复杂性，二进制编码、序列编码、值编码等一维编码方式不足以表述程序依赖融合图的结构；树形编码适合用于描述以抽象语法树形式表示的程序，而不适用于程序依赖图结构；矩阵编码可以用于表示图结构，但由于程序依赖融合图需要满足的诸多约束条件的存在，基于矩阵编码的遗传操作会产生不符合约束条件的无效个体，因此必须采用额外的操作对无效个体进行筛查，增大了算法的复杂性。因此，本文提出了一种针对程序依赖融合图的编码方式，称为节点集编码。直观上，节点集编码的结构与融合图的节点集合相对应，具有较强的表达能力。同时，后文介绍的基于节点集编码的交叉与变异操作与程序依赖融合图中的节点操作相对应，简化了交叉与变异操作，并能够保持操作不影响程序依赖融合图约束条件的满足。

在节点集编码中，融合图与被融合图的节点顺序对编码方式不产生影响。但为表述方便，本章为每个被融合图与被融合图中每个节点赋予一个自然数编号，并以 $i.j$ 或 $(i.j)$ 的形式表示编号为 i 的被融合图中，编号为 j 的节点。

节点集编码使用集合作为一个融合节点的编码，集合中的元素是该融合节点中的被融合节点，这与融合节点的定义相对应。融合图编码为融合节点构成的集合，这与融合图的节点集的定义相对应。节点集编码描述了融合图中融合节点集合的构成，根据第二章介绍的程序依赖融合图性质，融合图的边集合构成能够根据节点集构成计算获得，因此节点集编码能够描述程序依赖融合图结构。

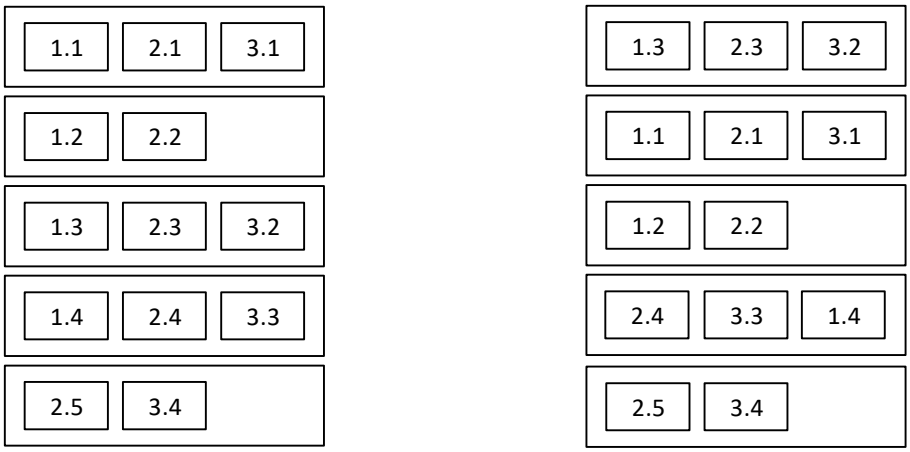
由于程序依赖融合图的约束保证了只有相同类型的节点才能够进行融合，因此不同类型的节点的融合过程是相互分离的，在构造融合方案时，可以按照节点的类型分别进行构造。



(a) 被融合图　　(b) 融合图

图 3.2　融合图表示示例

以图 3.2 为例，图(a)为三个被融合图示例，图中省略了节点内容，只以编号表示节点。图(b)是图(a)中的被融合图进行融合后得到的一个融合图。则，图(b)中所示融合图可以以图 3.3(a)所示的节点集合表示：



(a) 节点集表示 1　　(b) 节点集表示 2

图 3.3　节点集表示示例

值得注意的是，节点集编码中，集合的元素是顺序无关的，即使交换表示中的元素顺序，其仍表示同一融合图。故此编码形式不与一维编码等同。例如，图 3.3(b)表示与图 3.3(a)表示等价，均表示图 3.2 (b)所示的融合图：

可以看到，节点集编码形式与融合图的节点集合直接对应，这使得后文所述的交叉与变异操作更加简洁与直观。

3.3 程序依赖融合图的交叉运算

遗传算法中，交叉运算是以两个个体为基础，产生新的个体的过程。交叉的目标是产生同时具有亲代个体特征的子代。在传统的交叉运算中，这一目标通过模拟遗传学中染色体交叉互换的过程实现，即对两个亲代个体的染色体进行重新组合分配。在程序依赖图融合问题中，由于程序依赖融合图结构的复杂性及其需要的特殊编码方式，通常的交叉方法无法保证产生符合程序依赖融合图结构约束条件的子代个体。因此，本文提出了一种基于节点结构变换的融合图交叉方法。该方法以两个亲代融合图为基础，产生一个结构介于两个亲代个体中间的子代融合图。

对于两个融合图，交叉运算的目标是确定二者之间的一条最短编辑路径，并以编辑路径上的一个“中间状态”作为交叉运算的结果，即子代融合图。融合图之间的编辑路径是将一个融合图变换为另一个融合图所需要进行的操作的集合。本节首先给出对融合图的编辑操作——节点迁移的定义，并说明最短编辑路径的计算方法，最后给出交叉运算的过程描述。

3.3.1 节点迁移

本文采用的交叉运算中，计算两个融合图之间的编辑路径所采用的编辑操作是节点迁移。节点迁移操作是将一个被融合节点从一个融合节点移动到另一个融合节点的过程，具体地：

定义（节点迁移）：对于融合节点集合$\mathcal{V}$，$v_1^m, v_2^m \in \mathcal{V}$，取$v \in v_1^m$，令$v_1^m = v_1^m - \{v\}$，$v_2^m = v_2^m \cup \{v\}$，该过程称为将节点 v 从 v_1^m 移至 v_2^m ，记为$Move(v, v_1^m, v_2^m)$。

节点迁移过程修改了两个融合节点的结构，这一操作可能使得修改后的程序依赖融合图不再满足结构约束。对于融合节点v_1^m, v_2^m，若v_2^m中包含与将要移动的节点 v 来自同一被融合图的另一节点v'，即$\exists v' \in v_2^m \ \ s.t. \ graph(v) = graph(v')$，则节点迁移操作$Move(v, v_1^m, v_2^m)$会使得节点 v_2^m 中包含两个来自同一被融合图的节点，这使得修改后的融合图不满足“任意融合节点中，不存在两个来自同一被融合图的被融合节点”的约束。对于这种情况，特别规定，对于融合节点v_1^m, v_2^m，被融合节点$v \in v_1^m, v' \in v_2^m, graph(v) = graph(v')$，在执行节点迁移操作$Move(v, v_1^m, v_2^m)$后，必须执行另一节点迁移操作$Move(v', v_2^m, v_1^m)$。

3.3.2 程序依赖融合图的编辑距离与编辑路径

在节点迁移操作的基础上，可以定义以节点迁移为基本编辑操作的程序依赖融合图的编辑距离与编辑路径。

定义（融合节点集的编辑距离与最短编辑路径）：对于融合节点集$\mathcal{V}_1$和$\mathcal{V}_2$，对$\mathcal{V}_1$进行一组节点迁移操作后，使得$\mathcal{V}_1$与$\mathcal{V}_2$结构相同，所需的最少操作数，称为融合节点集

$\mathcal{V}_1$与$\mathcal{V}_2$的编辑距离。能够使操作数最少的一组节点迁移操作称为$\mathcal{V}_1$到$\mathcal{V}_2$的一条最短编辑路径。

除了上文所述为满足融合图约束而规定的成对操作外，节点迁移操作是相互独立的。因此，在融合节点集的编辑路径中，交换两个节点迁移操作不影响整体操作结果。因此融合节点集的编辑路径中的节点迁移操作是顺序无关的。

根据融合节点集的编辑距离和编辑路径，可以定义程序依赖融合图的编辑距离与编辑路径。对于两个程序依赖融合图，其编辑距离是各个类型融合节点集和间编辑距离之和，其编辑路径是各个类型融合节点集之间编辑路径的并。

直观上，程序依赖融合图之间的编辑距离与编辑路径描述了程序融合问题的解空间中，两个解之间的最短距离与最短路径。本文即是以最短路径上的中点作为两个融合方案的交叉结果。

融合节点集之间的编辑距离与编辑路径通过寻找两个集合间的最大匹配实现。

定义（融合节点集的匹配）：对于融合节点集$\mathcal{V}_1$和$\mathcal{V}_2$，其匹配$M = \{\langle v_s^m, v_t^m\rangle | v_s^m \in \mathcal{V}_1 \cup \{\emptyset\},\ v_t^m \in \mathcal{V}_2 \cup \{\emptyset\}\}$，满足：

- $v_s^m \cup v_t^m \neq \emptyset$
- $\forall \langle v_s^m, v_t^m\rangle \neq \langle v_s^{m\prime}, v_t^{m\prime}\rangle \in M, v_s^m \neq v_s^{m\prime}, v_t^m \neq v_t^{m\prime}$
- $\forall v_s^m \in \mathcal{V}_1, \exists v_t^m \in \mathcal{V}_2 \cup \{\emptyset\}\ \ s.t. \langle v_s^m, v_t^m\rangle \in M,$
- $\forall v_t^m \in \mathcal{V}_2, \exists v_s^m \in \mathcal{V}_1 \cup \{\emptyset\}\ \ s.t. \langle v_s^m, v_t^m\rangle \in M$。

根据融合节点集的匹配，可以定义基于匹配的融合节点集的距离：

对于融合节点v_s^m和v_t^m，定义$dist(v_s^m, v_t^m) = |v_s^m \triangle v_t^m|$，其中△是集合的对称差运算，$v_s^m \triangle v_t^m = (v_s^m - v_t^m) \cup (v_t^m - v_s^m)$。

对于一个匹配M，定义$dist(M) = \sum_{\langle v_s^m, v_t^m\rangle \in M} dist(v_s^m, v_t^m)$。

定义（融合节点集的最大匹配）：对于融合节点集$\mathcal{V}_1$和$\mathcal{V}_2$，最大匹配M_{max}是一个匹配，满足任取$\mathcal{V}_1, \mathcal{V}_2$的一个匹配$M$，$dist(M) \geq dist(M_{max})$。

在上述定义中，对于融合节点v_s^m和v_t^m，$dist(v_s^m, v_t^m)$实际上描述的是为将v_1^m变为v_2^m，所需的从v_1^m中移除的节点数与向v_1^m中加入的节点数之和。进一步地，对于融合节点集$\mathcal{V}_1$和$\mathcal{V}_2$及其一个最大匹配M_{max}，$dist(M_{max})$描述的则是将$\mathcal{V}_1$变为$\mathcal{V}_2$最少所需的对$\mathcal{V}_1$中融合节点进行节点移除和加入的数目总和。不难发现，这个数目即为$\mathcal{V}_1$与$\mathcal{V}_2$的编辑距离的 2 倍，因此$dist(M_{max})/2$ 即为$\mathcal{V}_1$与$\mathcal{V}_2$的编辑距离。

融合节点集最大匹配的计算实际上是一个带权二部图最大匹配问题，该问题已有较成熟的求解方案。本文利用匈牙利算法计算融合节点集的最大匹配。

3.3.3 基于编辑路径的交叉运算

基于上述定义，下面给出本文使用的程序依赖融合图交叉算法。对于两个程序依赖融合图$\mathcal{G}_1$与$\mathcal{G}_2$，按照节点类型划分的每一对融合节点集$\mathcal{V}_1$与$\mathcal{V}_2$，对于每一对融合节点集分别进行交叉运算，其步骤如下：

1) 寻找两个节点集的一个最大匹配。
2) 根据最大匹配中每一对匹配的融合节点之间的对称差，计算出将它们变为相同结构所需修改的节点集合，
3) 根据这个集合构造出$\mathcal{V}_1$到$\mathcal{V}_2$的一个最短编辑路径，即一组节点迁移操作的集合。
4) 在这组节点迁移操作中，随机选取一半的节点迁移操作对$\mathcal{V}_1$执行，即可得到状态介于$\mathcal{V}_1$与$\mathcal{V}_2$中间的一个融合节点集。

根据每个节点类型的交叉融合节点集可以构成完整的交叉融合图的节点集。根据程序依赖融合图边集能够被节点集唯一确定的性质，算法在获得交叉融合节点集后，即可构造出完整的交叉程序依赖融合图。

上述步骤的具体实现如算法 3.1 所示。

其中，$\mathrm{MaximumMatching}(\mathcal{V}_1, \mathcal{V}_2)$函数寻找$\mathcal{V}_1$与$\mathcal{V}_2$的一个最大匹配并返回，$\mathrm{RandomSelect}(S_{move})$函数在$S_{move}$集合中随机选取 1 个元素并返回。第 16 行至 25 行对为满足融合图结构约束而必须执行的成对节点迁移操作进行了检查。第 17 行中，根据融合图结构约束，$\exists Move(v', v_t^m, v_t^{m\prime}) \in S_{move}$一定成立，因此该 Find 操作一定成功。

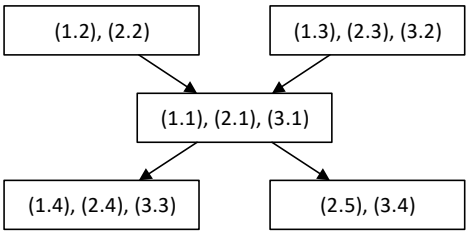
算法 3.1 融合节点集的交叉运算

输入：融合节点集$\mathcal{V}_1, \mathcal{V}_2$
输出：交叉融合节点集$\mathcal{V}_{crossover}$

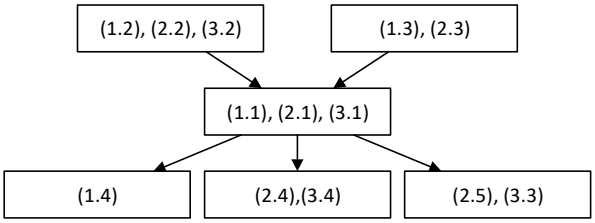
Function Crossover($\mathcal{V}_1, \mathcal{V}_2$)
  $M \leftarrow$ MaximumMatching($\mathcal{V}_1, \mathcal{V}_2$)
  $S_{move} \leftarrow \{\}$
  for each $\langle v_1^m, v_2^m \rangle \in M$ **do**
    for each $v \in v_1^m - v_2^m$ **do**
      Find $\langle v_3^m, v_4^m \rangle \in M$ such that $v \in v_4^m$
      $S_{move} \leftarrow S_{move} \cup \{Move(v, v_1^m, v_3^m)\}$
    end for
  end for
  $movecount \leftarrow \lfloor |S_{move}|/2 \rfloor$
  while $movecount > 0$ **do**
    $Move(v, v_s^m, v_t^m) \leftarrow$ RandomSelect(S_{move})
    Execute $Move(v, v_s^m, v_t^m)$ on $\mathcal{V}_1$
    $S_{move} \leftarrow S_{move} - Move(v, v_s^m, v_t^m)$
    $movecount \leftarrow movecount - 1$
    if $\exists v' \in v_t^m \ s.t. \ graph(v) = graph(v')$ **then**
      Find $Move(v', v_t^m, v_t^{m\prime}) \in S_{move}$
      Execute $Move(v', v_t^m, v_t^{m\prime})$ on $\mathcal{V}_1$
      $S_{move} \leftarrow S_{move} - Move(v', v_t^m, v_t^{m\prime})$
      if $v_t^{m\prime} = v_s^m$ **then**
        $movecount \leftarrow movecount - 1$
      else then
        $S_{move} \leftarrow S_{move} \cup Move(v', v_s^m, v_t^{m\prime})$
      end if
    end if
  end while
  $\mathcal{V}_{crossover} = \mathcal{V}_1$
  Remove empty vertex in $\mathcal{V}_{crossover}$
  return $\mathcal{V}_{crossover}$
end Function

以图 3.2(a)中所示的三个被融合图为基础，图 3.4 展示了一个交叉运算的例子。图(a)与图(b)是由图 3.2 (a)中被融合图融合而成的两个融合图，分别记为$\mathcal{G}_1$与$\mathcal{G}_2$。图(c)是$\mathcal{G}_1$与$\mathcal{G}_2$节点集合之间的一个最大匹配M，图中左右连接的虚线表示被匹配的一对融合节点；左侧的灰色节点表示为使$\mathcal{G}_1$与$\mathcal{G}_2$结构相同，图$\mathcal{G}_1$中需要被移动的节点；右侧的虚线边框节点表示这些需要移动的节点在图$\mathcal{G}_2$“应当所属”的位置。根据上述定义，

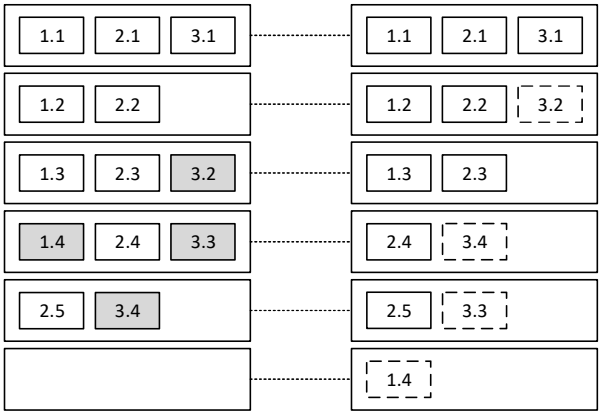
$dist(M)=8$，$\mathcal{G}_1$与$\mathcal{G}_2$节点集之间的编辑距离为 4，即为使$\mathcal{G}_1$与$\mathcal{G}_2$结构相同，最少需要进行的节点迁移操作数为 4。并且，最大匹配找出了需要进行迁移的 4 个节点，即(3.2)，(1.4)，(3.3)，(3.4)。图(d)是在$\mathcal{G}_1$基础上，在上述 4 个节点中，选择了(3.2)，(1.4)两个节点进行迁移后得到的结果，(3.2)与(1.4)两个节点被迁移到了与$\mathcal{G}_2$中相应位置对应的节点。图(e)是根据图(d)的节点集结构构造的融合图，该图即可作为$\mathcal{G}_1$与$\mathcal{G}_2$的一个交叉子代。可以看到，该交叉图同时具有$\mathcal{G}_1$与$\mathcal{G}_2$的特征，即同时拥有与$\mathcal{G}_1$、$\mathcal{G}_2$相同结构的融合节点，是一个状态介于$\mathcal{G}_1$与$\mathcal{G}_2$之间的融合图。



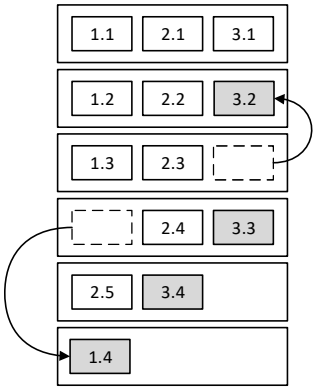
(a) 融合图$\mathcal{G}_1$



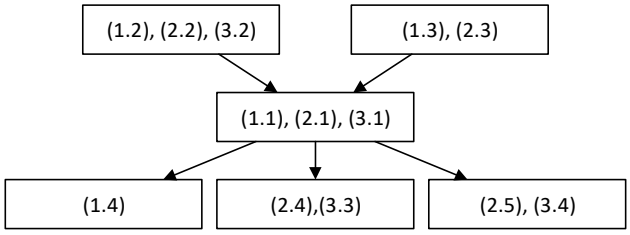
(b) 融合图$\mathcal{G}_2$



(c) $\mathcal{G}_1$与$\mathcal{G}_2$节点集和间的一个最大匹配



(d) 对$\mathcal{G}_1$执行一半的节点迁移操作结果



(e) 交叉融合图

图 3.4　融合图交叉运算示例

3.4 程序依赖融合图的变异运算

遗传算法中，变异运算是对自然界中的基因突变的一个模拟。变异运算对一个个体进行少量修改，使其产生出新的基因型与性状。与交叉运算相同，程序依赖融合图的变异运算同样通过节点迁移操作修改融合节点结构实现。

与交叉类似的，对于一个程序依赖融合图$\mathcal{G}$，对于按照节点类型划分的每一个融合节点集$\mathcal{V}$，算法尝试进行一次或多次变异运算，每次变异运算步骤如下：

1) 向$\mathcal{V}$中添加一个空融合节点，之后在$\mathcal{V}$中随机选择两个融合节点v_1^m与v_2^m。
2) 计算v_1^m与v_2^m中包含的被融合节点所属图的集合G。
3) 随机选择G中一半的元素作为G'。
4) 交换v_1^m与v_2^m中，来自于G'中被融合图的节点。
5) 删除空节点。

在上述过程中，当步骤 1 选择的两个融合节点均非空时，变异运算的作用是将两个融合节点中的部分被融合节点交换；当步骤 1 选择了一个空融合节点时，变异运算的作用是将一个融合节点分裂为两个融合节点。

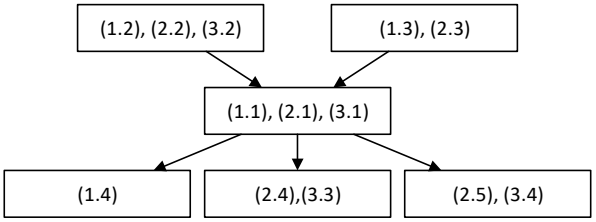
上述变异步骤的具体实现如算法 3.2 所示：

算法 3.2 融合节点集的变异运算

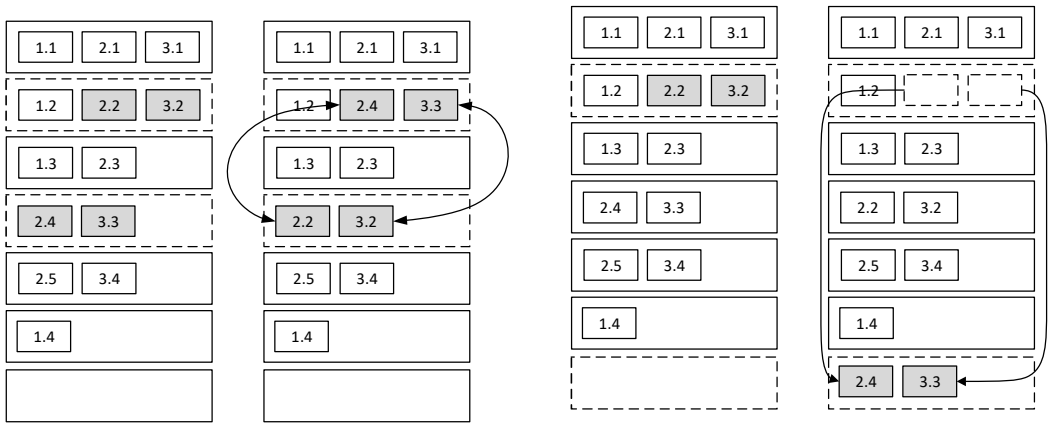
输入：融合节点集合$\mathcal{V}$
输出：变异后融合节点集$\mathcal{V}_{mutation}$

Function Mutation($\mathcal{V}$)
　$\mathcal{V} \leftarrow \mathcal{V} \cup \emptyset$
　$v_1^m, v_2^m = \text{RandomSelect}(\mathcal{V}, 2)$
　$G = \{\ g \mid \exists v \in v_1^m \cup v_2^m\ \ s.t.\ graph(v) = g\}$
　$G' = \text{RandomSelect}(G, |G|/2)$
　$V_1 \leftarrow \{v\ |v \in v_1^m \wedge graph(v) \in G'\}$
　$V_2 \leftarrow \{v\ |v \in v_2^m \wedge graph(v) \in G'\}$
　$v_1^{m\prime} = v_1^m \cup V_2 - V_1$
　$v_2^{m\prime} = v_2^m \cup V_1 - V_2$
　$\mathcal{V}_{mutation} = \mathcal{V} \cup \{v_1^{m\prime}, v_2^{m\prime}\} - \{v_1^m, v_2^m\} - \{\emptyset\}$
　return $\mathcal{V}_{mutation}$
end Function

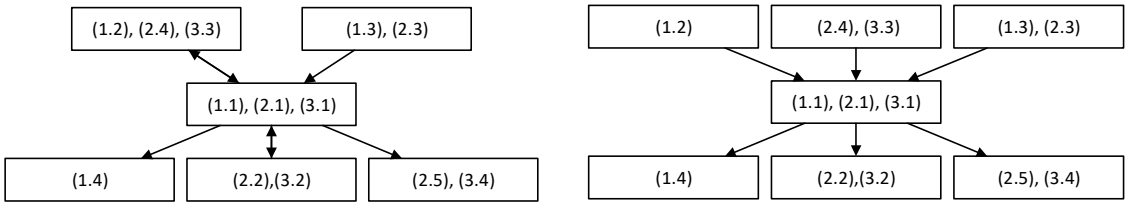
以图 3.4 (e)所示的融合图为例，图 3.5 展示了两个变异运算的例子。图(a)是变异前的融合图。图(b)与图(c)是变异运算中的两种情况，即节点交换的情况与节点分裂的情况。其中虚线标记了被选中的融合节点，灰色标记了其中需要交换的被融合节点。图中给出了进行交换或分裂前后节点集的构成情况，并展示了进行变异操作后的融合图结构。



(a) 变异前融合图



$G = \{1,\ 2,\ 3\},\quad G' = \{2,\ 3\}$ $G = \{1,\ 2,\ 3\},\quad G' = \{2,\ 3\}$



(b) 节点交换的情况　　(c) 节点分裂的情况

图 3.5　变异运算示例

3.5　节点相似度的计算与基于广义熵的适应度函数

程序依赖图融合的基本目标是将相似的被融合节点融合在一起。节点的相似度体现于节点内容的相似度与节点上下文关系的相似度两方面。对于节点上下文关系相似度，第二章描述了基于广义熵的度量方法。对于节点内容相似度，本节将介绍基于抽象语法树（AST）相似度的度量方法。之后结合基于广义熵的节点上下文关系相似度计算，给出本文采用的融合图适应度函数计算方法。

3.5.1 基于抽象语法树的节点相似度计算

节点内容的相似度的度量本质上是对节点对应的程序语句相似度的度量。一个朴素的方案是，将程序语句视为一般的字符串，并利用字符串相似度度量算法计算程序语句相似度。基于字符串的程序语句相似度度量受到程序语句表示形式的影响。例如，语句中变量名会极大地影响相似度度量结果。在群体协同开发的场景中，不同的开发者往往拥有的不同的程序书写习惯，使得不同的开发者对相同语义的程序语句会采用不同的表示方法。因此，在计算程序语句相似度时，应排除表示形式的影响。因此，本文使用基于抽象语法树的程序语句相似度度量方法，通过比较程序语句的结构对相似度进行计算。

在节点相似度的计算中，两个节点（或程序语句）之间的相似度应当为一个 0 至 1 闭区间内的浮点值，对于同一语句形式上的差异（如变量名不同）不应对相似度产生影响。依据程序语句类型的不同，其相似度计算也采用不同的计算方法。

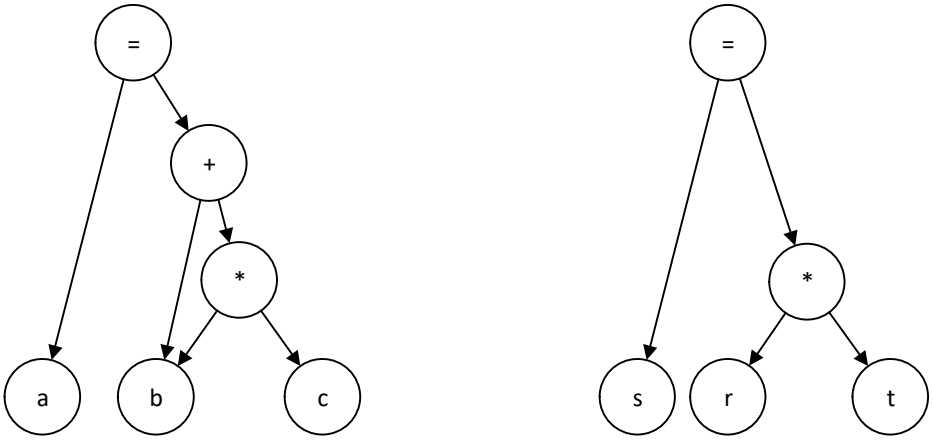
对于程序入口节点，或 break、continue 语句节点等拥有固定形式的节点，其相同类型的节点间相似度为 1，不同类型节点间相似度为 0。

对于变量声明节点，相似度计算时不考虑变量的名称，只根据该变量的类型判断相似度。在本文的实现中，认为相同的类型相似度为 1，类型相同维度不同的数组间相似度为 0.75，基本类型与该类型数组之间相似度为 0.5，其余情况为不相同类型，相似度为 0。

对于其他类型的语句节点，则根据语句的抽象语法树结构计算相似度。具体地，两条语句的语法树结构越相近，则他们的相似度应该越高。对于语法树结构相似度的判断，本文采用寻找两个语法树之间一个最优匹配的方法。为此，首先对本文使用的程序语句抽象语法树结构进行描述。

1) 抽象语法树结构

抽象语法树以节点表示程序语句中的元素，如变量、运算符等，以有向边表示语句元素之间的层级关系，如运算符到其运算数之间存在一条有向边。对于一条程序语句，将其抽象语法树中相同变量对应的节点合并为同一节点后，该语法树成为一个有向无环图。本文即采用这种有向无环图表示的语法树进行结构比较。图 3.6 展示了 a = b + b * c 与 s = r * t + t 两条语句的抽象语法树。



(a) a = b + b * c 的抽象语法树　　(b) s = r * t 的抽象语法树

图 3.6　抽象语法树示例

每个节点拥有唯一的类型，即节点对应语句元素的种类；每条边也拥有唯一的类型，边类型表示这条边所表述的是语句元素间何种关系。典型的语法树节点类型包括运算符、赋值符、变量、常量、参数列表等，典型的边类型包括运算符到运算数、数组变量到数组下标、赋值符到被赋值变量、赋值符到表达式等。

除节点内容、节点与边的类型定义不同外，抽象语法树的结构与本文使用的程序依赖图结构有着很高的相似性。因此，在定义了抽象语法树节点的相似度计算方法后，本章所述的图融合算法同样适用于语法树融合。而图融合的作用正是寻找两个或多个图之间节点的对应关系，因此可以用于对两个语法树结构进行比较。由此，两条程序语句相似度的比较转化为两个抽象语法树结构的比较，而语法树结构比较则通过语法树融合的方法进行。

2) 语法树节点相似度与语法树融合

在程序依赖图融合过程中，需要进行程序语句节点的相似度计算，为此上文提出了基于抽象语法树融合的语句相似度计算方法。在抽象语法树融合的过程中，则需要计算语法树节点的相似度。在抽象语法树中，节点对应的是程序语句中最基本的单元，如变量、运算符等。其相似度计算相较于程序语句相似度计算也简单许多。

- 对于运算符、常量、库函数调用的函数名节点，可直接根据其内容判断相似度。内容相同则相似度为 1，否则为 0。
- 对于变量类的节点，其内容是对应的变量名。对于程序语句而言，变量名不对结构产生影响，因此变量节点的相似度无视变量名，认为为 1。

依据语法树节点的相似度，即可利用与本章所述的程序依赖图融合相同的算法进行抽象语法树融合。与程序融合问题不同的是，抽象语法树融合只有两个被融合图，是一

个二图融合问题，同时由于程序语句复杂性远低于整个程序，语法树规模远小于程序依赖图，因此语法树融合问题规模远小于程序依赖图融合问题。

3) 基于语法树融合的语句相似度计算

基于语法树的融合结果，可以定义两个抽象语法树的相似度。

对于两个抽象语法树T_1与T_2，它们的融合抽象语法树为M，则可分别根据其节点与边的融合情况定义T_1与T_2的节点相似度与边相似度。

节点相似度定义为：

$$S_v(T_1,T_2)=2\frac{\sum_{v_1\neq v_2\in m,m\in V(M)\wedge|m|>1}Sim(v_1,v_2)}{|V(T_1)|+|V(T_2)|} \tag{3.1}$$

其中$Sim(v_1,v_2)$表示语法树节点v_1与v_2的相似度。

边相似度定义为：

$$S_e(T_1,T_2)=2\frac{|E(T_1)|+|E(T_2)|-|E(M)|}{|E(T_1)|+|E(T_2)|} \tag{3.2}$$

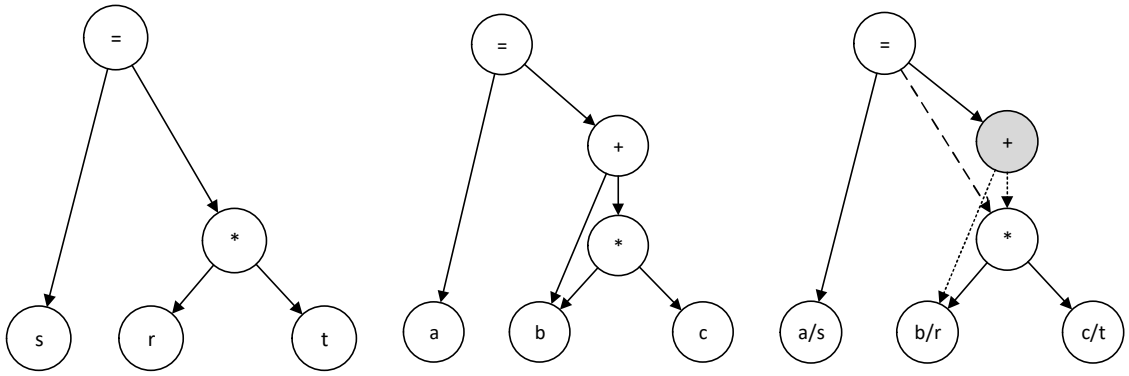
直观上，节点相似度描述的是所有被融合在一起的语法树节点对之间的相似度之和与两个语法树节点数之和的比。当两个语法树的节点完全融合，且每一对融合的语法树节点相似度均为 1 时，两个语法树节点数相同，且融合语法树也具有相同的节点数，且等于 $\sum_{v_1\neq v_2\in m,m\in V(M)\wedge|m|>1}Sim(v_1,v_2)$的值，因此$S_v(T_1,T_2)=1$；而当语法树完全不同，所有节点均未融合时，融合语法树中每个融合节点只包含 1 个被融合节点，$\sum_{v_1\neq v_2\in m,m\in V(M)\wedge|m|>1}Sim(v_1,v_2)=0$，因此$S_v(T_1,T_2)=0$。

类似的，边相似度描述的是被融合在一起的边数与两个语法树总边数的比。当两个语法树的边完全融合时，$|E(T_1)|=|E(T_2)|=|E(M)|$，因此$S_e(T_1,T_2)=1$；当语法树的边完全未融合时，$|E(T_1)|+|E(T_2)|=|E(M)|$，因此$S_e(T_1,T_2)=0$。

在上述基础上，定义两个抽象语法树整体相似度如下：

$$Sim(T_1,T_2)=\theta S_v(T_1,T_2)+(1-\theta)S_e(T_1,T_2) \tag{3.3}$$

即两个语法树节点相似度与边相似度的加权和，其中$\theta\in(0,1)$为权值。



(a) s = r * t 的语法树　　(b) a = b + b * c 的语法树　　(c) 融合语法树

图 3.7　语法树融合示例

图 3.7 展示了一个语法树融合的例子。图(a)与图(b)是两个被融合语法树，图(c)是两个语法树的融合结果。图(c)中标记了所有未融合的节点与边：灰色节点（“+”节点）表示只来自图(b)的节点；虚线边（“=”节点到“*”节点）表示只来自图(a)的边；点线（“+”节点到“*”节点）表示只来自图(b)的边。其余节点与边则由双方融合而成。图(c)中，每一对被融合的节点相似度都为 1。根据上述定义，图(a)与图(b)的节点相似度为：

$$S_v(T_1,T_2)=2\frac{\sum_{v_1\neq v_2\in m,m\in V(M)\wedge|m|>1}Sim(v_1,v_2)}{|V(T_1)|+|V(T_2)|}=2\times\frac{5}{5+6}=\frac{10}{11} \tag{3.4}$$

边相似度为：

$$S_e(T_1,T_2)=2\frac{|E(T_1)|+|E(T_2)|-|E(M)|}{|E(T_1)|+|E(T_2)|}=2\times\frac{4+6-7}{4+6}=\frac{3}{5} \tag{3.5}$$

取权值$\theta=0.5$，则图(a)与图(b)所示语法树的相似度为:

$$Sim(T_1,T_2)=\theta S_v(T_1,T_2)+(1-\theta)S_e(T_1,T_2)=0.5\times\frac{10}{11}+0.5\times\frac{3}{5}\approx 0.755 \tag{3.6}$$

因此，程序语句 s = r * t 与 a = b + b * c 的相似度为 0.755。

3.5.2 基于广义熵的适应度函数

基于第二章介绍的程序依赖融合图广义熵的定义与上文介绍的程序语句相似度的计算方法，对于一个程序依赖融合图，可以计算其广义熵。程序依赖融合图的广义熵是一个非负的浮点值。广义熵越高，则被融合图的相似度越低，融合到一起的节点不相似或融合到一起的节点较少；广义熵越低，则被融合图越相似，有更多的相似节点被融合

到一起；广义熵为 0，则被融合图两两之间结构相同，且相同节点完全融合到一起。因此，广义熵直接刻画了程序依赖图的融合程度，能够反映程序依赖融合图在多大程度上达到本文对于程序融合的“将相似的程序语句尽可能融合”的目标。本文以广义熵加 1 的倒数作为遗传算法中用于个体评估的适应度函数，即：

$$fitness(\mathcal{G}) = \frac{1}{\mathcal{H}(\mathcal{G}) + 1} \tag{3.7}$$

个体的广义熵越低的个体适应度越高。当个体的广义熵达到 0 时，其适应度达到最大值 1。在遗传过程中，算法的目标是寻找广义熵尽可能低的个体。

3.6 遗传算法中其他步骤的设计

本节将对基于遗传算法的程序依赖图融合过程中其他步骤中的一些策略或设计进行描述，包括种群初始化过程、选择过程、种群多样性调整与终止条件的设定。

3.6.1 种群初始化

在遗传算法开始阶段，需要对种群进行初始化。通常，种群初始化即为随机生成个体构成种群的过程。对于程序依赖融合图，随机生成个体即根据一组被融合图进行随机融合。而由于程序依赖融合图的边构成可以根据节点构成确定，故程序依赖图的随机融合过程即为融合节点集的随机构成过程。具体地，在融合图随机生成过程中，对于所有被融合图的所有被融合节点，将其随机融合到一个合法的现有融合节点，或将其单独融合到一个新的融合节点。一个合法的融合节点是指融合后不违反第二章中介绍的程序依赖融合图约束条件的融合节点。当所有被融合节点均融合后，即得到一组合法的融合节点。以这组节点作为的融合节点集，即可构成一个程序依赖融合图。

3.6.2 选择

程序融合的遗传算法中，选择过程分为 2 部分：一是维持种群规模，将种群中低适应度（即高广义熵）的个体删除的过程；二是基于个体的适应度，选择能够参与交叉过程中的每一对亲本个体。

维持种群规模的过程是对自然界中自然选择过程的模拟，对环境适应度低的个体将被“淘汰”。在算法中，当种群经过交叉产生了子代后，种群中个体数量超过了设定的规模。此时，对种群中全部个体按照适应度排序，将超出种群规模设定值的低适应度（高广义熵）个体删除，以达到维持种群规模的目的。

交叉亲本选择的过程是对自然界中，对“较强适应度的个体拥有优先交配权”的模拟。令种群规模为D，则遗传算法中每一代会通过交叉产生$D/2$个子代。对于这$D/2$个

子代的双亲，本文通过基于适应度比例的选择方法，即轮盘赌选择方法。其思想是一个个体被选为交叉亲本的概率与其适应度成正比。令$P(\mathcal{G})$表示$\mathcal{G}$被选为交叉亲本的概率，则$\forall \mathcal{G}_i \in \{\mathcal{G}_1, \mathcal{G}_2, \dots, \mathcal{G}_n\}$，有：

$$P(\mathcal{G}_i) = \frac{fitness(\mathcal{G}_i)}{\sum_{k\in 1,2,\dots,n} fitness(\mathcal{G}_k)} \tag{3.8}$$

基于轮盘赌选择方法，高适应度的个体可能在一代中参与多次交叉，使得“优势基因”得到巩固，同时低适应度的个体仍有产生子代的可能。

3.6.3 种群多样性调整

种群多样性调整过程的目的是避免在遗传过程中种群陷入同一化。同一化即种群中个体拥有完全一样的基因型，即程序依赖融合图拥有完全相同的结构。这使得算法难以进一步对解空间进行搜索，进而过早收敛在一个局部最优解。

本文采用的种群多样性调整方法为：在每一代结束时对种群中重复个体进行剔除，并以随机生成的个体进行补充。具体地，令种群规模为N，算法设定一个个体重复度阈值γ，并限定相同结构的融合图不能出现$\lceil \gamma N \rceil$次以上。对于程序依赖融合图，结构相同定义为拥有完全相同的融合节点集构成。当一组结构相同的融合图数目超过$\lceil \gamma N \rceil$时，则保留其中$\lceil \gamma N \rceil$个，将其余的融合图从种群中删除。当对整个种群的重复个体删除操作完成后，采用 3.6.1 节所述的融合图随机生成方法，随机生成与删除个体数目相同的新个体加入种群。

3.6.4 终止条件设定

本文采用三个条件作为遗传算法的终止判断：

1) 目标熵达到：在程序融合问题中，广义熵拥有确定的下界 0，本文以其作为融合算法的目标。当种群中出现熵为 0 的个体时，即代表存在了一个完美的融合方案。此时，算法终止，并以该个体作为程序融合的最终结果。
2) 连续收敛代数达到设定值：当种群中最优个体的广义熵连续多代未发生明显变化时，则认为算法已经找到了一个足够优解。在实现中，算法设定了一个最大收敛代数d，若种群连续d代的最优个体广义熵的最大值与最小值之差不超过10^{-6}，则认为种群已收敛。此时，算法终止，并以最新一代的最优个体作为程序融合的结果。
3) 遗传代数达到设定值：当上述 2 个条件无法在可接受时间内达成时，算法限定遗传过程不能超过一个最大代数。当算法经过该代数的遗传，仍无法满足上述 2 项条件时，则令算法终止，并以最新一代的最优个体作为程序融合的结果。

3.7　本章小结

本章介绍了一种基于遗传算法的程序依赖图融合算法。算法采用程序依赖融合图的节点集作为遗传算法中的个体表示方法。基于该表示法，本章介绍了用于遗传算法的交叉与变异运算的设计。之后本章介绍了算法中使用的程序语句相似度计算方法，并结合第二章提出的广义熵概念，给出了遗传算法中采用的适应度函数设计。最后，本章介绍了遗传算法中种群初始化、选择、种群多样性调整与算法终止判断过程的设计。

第四章　基于程序依赖融合图的错误定位

在程序融合与反馈的过程中，程序融合的目标是尽可能地提取群体对目标问题解决方案的共性部分，并分离个体解决方案之间的差异，程序融合的结果以程序依赖融合图表示。而错误定位过程则是在程序依赖融合图的基础上，识别每个个体程序中的错误。与一般的错误定位方法不同，在群体协同开发的环境中，当群体对于目标问题的解决方案达成一定程度的共识时，这一共识将能够为针对每个个体的错误定位提供参考。因此，错误定位过程可以对群体中其他个体的信息加以利用。本章介绍的错误定位方法，即是利用此群体共识思想，发现群体中每个个体程序中可能存在的错误。

基于群体共识的错误定位建立在程序融合的基础上，程序融合过程对群体在目标问题的解决方案上的共识进行了提取，以程序依赖融合图的形式表示；而个体与群体共识相异的部分则体现为程序依赖融合图中的局部“异常”。基于群体共识思想，与群体共识相异的个体程序片段可能存在程序错误。由此，基于群体共识的错误定位即转化为基于程序依赖融合图的共识发现与异常提取过程。程序融合的目标是将各个程序依赖图中相似的节点融合在一起，且相异的节点内分离。对应于程序，程序融合过程可以发现一组程序之间的结构上的相同与相异之处。当群体中一个个体程序在某个程序片段局部与其他个体存在较大差异时，在程序依赖图中，该程序将在某个或某几个节点上与其他程序依赖图中的节点具有较低的相似度。因而，在理想的程序融合过程中，这些节点将无法与其他节点较完美地融合。根据这一特性，本文设计了三种基于群体共识的错误定位策略。这些方法在程序依赖融合图的基础上对每个个体程序进行错误定位并形成反馈信息。

本章各节安排如下：4.1 节介绍基于群体共识的错误定位的基本思想，该节描述了错误定位方法所基于的核心思想——群体共识，以及在群体共识思想的基础上，程序依赖融合图的意义。4.2 节介绍本文提出的三种基于群体共识的错误定位方法，分别为基于节点融合度的错误定位、基于边融合度的错误定位、基于节点内部相似度的错误定位。4.3 节对本章进行总结。

4.1　基于群体共识的错误定位基本思想

在群体协同开发的环境下，多个开发者针对同一问题进行程序开发。尽管每个开发者的程序都可能存在错误，但在群体中，个体的程序之间可能能够相互弥补。具体地，在针对目标问题的开发过程中，一个开发者遇到的问题或出现的错误，可能能够被群体中另一个开发者解决或避免。在群体规模较大时，群体会对目标问题的解决方案或解决方案片段，即正确的程序或程序片段的形式产生一定程度的共识。这一共识是由

群体中的个体共同“贡献”而成的，相较于其他“独特”的程序片段，达成共识的程序形式更有可能代表着正确的问题解法。这一思想对于群体及群体程序有着如下假设与依赖：

1) 群体中的程序应当存在足够的共性：针对目标问题，群体中的每个个体应当使用相同或相近的解决思路。思路与方法相异的程序之间难以提供有效的信息参考，而群体中不同的解决方法之间也难以形成共识。
2) 个体的程序之间应当存在足够的互补性：群体中的错误应当具有足够的多样性，即一个个体所出现的错误，在其他程序中不一定存在，或在其他程序中包含了相同程序片段的正确形式。群体中应当没有共性的错误，这使得不借助外部信息，依赖群体的共识识别错误成为可能。

相比于现有的大多自动化错误定位方法，基于群体共识的错误定位方法对于能够定位的错误类型或造成错误的原因没有限制。特别地，基于群体共识的错误定位方法不依赖于目标问题的测试用例。当程序在与目标问题相关的位置出现程序逻辑错误时，可能代表着开发者自身对于目标问题理解存在偏差。在不借助测试用例的情况下，自动化的错误定位方法将因为无法理解目标问题而难以发现相关错误。但基于群体共识的错误定位方法可以利用融合后的程序中蕴含的其他开发者对目标问题的理解对出错的开发者进行错误定位与反馈。直观上，基于群体共识的错误定位方法利用群体对目标问题进行了理解。

4.1.1 程序依赖融合图的意义

在上述基本思想与假设的下，程序依赖融合图中的融合节点与融合边代表着更具体的意义。当群体中的程序全部基于相同或相近的解决思路时，这些程序存在着高度相似的逻辑结构与语句间关系。另一方面，程序语句的形式代表了语句在程序中执行的功能。当两条程序语句在语句形式与上下文关系两方面均有较高相似度的情况下，可以判断它们在各自程序中起到相同或相似的作用。在程序依赖图中，程序的逻辑结构与语句间关系对应于节点之间的依赖边，程序语句的形式对应节点的内容。因此，在程序融合过程中，被融合的一组节点对应的语句在语句形式与上下文关系上均具有较高的相似度，从而可以判断它们在各自原程序中起到相同或相似的作用。即，通过程序融合过程，一组节点被融合在一起时，不仅代表它们在形式与上下文关系拥有较高的相似度，还代表这些节点在程序中的位置与作用具有较高的相似度。

程序依赖融合图的上述意义表示，根据程序依赖图的融合情况，可以判断被融合节点对应语句在程序中位置与作用的对应关系。当一个程序中的某一程序语句出现错误时，在程序融合中将表现为这一语句对应的节点与其他程序中相同位置与作用的节点存在形式或关系上的差异，这一差异将在程序依赖融合图中表现为节点或边的未融合，

或节点内容的异常。下文介绍的错误定位方法均基于这一观察，通过发现程序中的错误在程序依赖融合图中表现出的异常对出错语句进行判断。

4.1.2 错误定位的形式

在本章介绍的错误定位方法中，程序错误的识别以程序语句为单位。当一个程序语句中某个局部存在微小错误（如赋值表达式中某一运算符出错），算法将对该语句整体进行识别并反馈；当多个语句同时存在错误并相互关联时，算法将对这些语句分别进行识别并反馈。在程序依赖图中，原程序中的每一个语句对应于一个节点。因此在以程序依赖融合图为基础的错误定位过程中，存在错误的语句即对应存在异常的节点。因此，在后文介绍的错误定位方法中，将对可能存在错误的程序依赖图节点进行定位。

4.2 错误定位方法

基于上述思想，在程序融合的过程中，存在错误的程序片段在程序融合过程中，将程序依赖融合图中表现出偏离群体共识的情况，成为一处“异常”。对于程序依赖融合图而言，异常可能体现于程序依赖图结构的异常或节点内容的异常。根据异常的形式不同，本文提出如下三种错误定位方法：基于节点融合度的错误定位方法、基于边融合度的错误定位方法、基于节点内部相似度的错误定位方法。

4.2.1 基于节点融合度的错误定位

第二章中，本文介绍了基于广义熵的融合质量度量；第三章中，本文介绍了用于程序融合的使用基于广义熵的适应度函数的遗传算法。融合节点的广义熵刻画了被融合在一起的一组节点之间，节点内容与节点依赖关系两方面的分歧程度。广义熵越高，则这一组节点的分歧程度越大。由于程序融合过程是以尽可能低的广义熵为引导目标，因此直观上，当一个程序依赖图中的某个节点与其他程序依赖图中的节点均具有较大分歧的时候，将该节点与其他任何节点融合均会导致广义熵的增大，融合算法会倾向于选择“孤立”该点。这一倾向表现在融合结果中，即为该节点单独形成一个融合节点。在融合程序依赖图中，节点融合度描述一个融合节点中包含的被融合点个数，则被“孤立”的节点所在的融合节点拥有较低的融合度。

对应于原程序而言，当一个程序中某个程序语句在语句内容与上下文关系两方面均有较高的特殊性，使得其他程序中不包含与其相似的语句时，则在程序依赖融合图中，该语句对应的被融合节点所在的融合节点具有较低的融合度，基于群体中存在正确共识的思想，该程序在该语句处可能存在错误。另一方面，当程序中存在部分冗余语句，程序中其他语句与群体中其他程序对应的相似语句成功融合时，即使这些语句未对程

序正确性造成影响，它们在融合图中仍成为孤立的未融合节点。对于冗余语句，本文将其等同视为一种程序错误进行识别并反馈。

具体地，令群体中程序总数为N，算法设置一个低融合度阈值$\theta_v \in [0,1]$。在程序依赖融合图中，当一个融合节点v^m的融合度$|v^m| < \theta_v N$时，则认为融合节点v^m的融合度过低。此时，对于v^m中包含的所有被融合节点，算法认为其对应的程序语句存在错误。

4.2.2 基于边融合度的错误定位

在程序融合过程中，融合节点的广义熵包含节点内容的广义熵与节点依赖关系的广义熵两部分。其中，节点依赖关系的广义熵刻画了被融合在一起的一组程序依赖图节点与上下文节点间依赖关系的相似性。在本文使用的程序依赖图结构中，节点间依赖关系包含控制依赖、数据依赖、声明依赖三种。在程序中，这三种依赖关系分别描述了节点对应语句的如下信息：

- 控制依赖：为了执行该语句所需满足的控制条件，体现为程序中该语句的层级结构。
- 数据依赖：该语句所使用或赋值的变量在其他语句中被赋值或使用的情况。
- 声明依赖：该语句所赋值的变量的声明位置，体现为该语句在程序中为哪个变量进行了赋值。

当一组被融合在一起的程序依赖图节点对应的语句在上述三类关系上存在分歧时，在程序融合结果中则体现为只有少数的融合边被融合在一起。与融合节点的融合度类似，边融合度表示一条融合边中包含的被融合边数量。

对于一个融合节点，当以其为源节点或目标节点的某条融合边具有较低的融合度时，则表示与该融合边中的被融合边关联的被融合节点在上下文关系上与群体中相同位置的其他被融合节点存在差异，则这些被融合节点对应的程序语句可能存在错误，这些错误的表现为该语句与上下文之间产生了异常的关系。

具体地，令群体中程序总数为N，对于融合节点v^m，算法设定一个低融合度阈值θ_e，若存在融合边e^m，满足$src(e^m) = v^m$或$tgt(e^m) = v^m$，$|e^m| < \theta_e N$，则认为e^m的融合度过低。此时，对于v^m中的一个被融合节点v，若其满足$\exists e \in e^m, src(e) = v$或$tgt(e) = v$，即该被融合节点关联的一条边被融合到了具有低边融合度的融合边中，则认为被融合节点v对应的程序语句可能存在错误。

根据异常的依赖类型不同，上述方法所识别的错误类型也不同：异常的控制依赖可能代表着错误的控制结构，如分支、循环结构中的语句位置错误；异常的数据依赖或声明依赖可能代表着语句中使用或赋值了错误的变量。当拥有上述错误的程序语句在形式上与群体中其他程序中的语句相似或相同时，根据相似程度，在程序融合过程中，该语句对应的节点仍可能被融合到一个具有较高节点融合度的融合节点。此时，该语句

在程序结构或变量上的异常则通过低融合度的融合边体现。

4.2.3 基于节点内部相似度的错误定位

基于节点融合度与基于边融合度的错误定位方法所基于的思想是：存在错误的语句对应的程序依赖图节点由于语句内容或上下文关系异常，无法与其他节点相融合。因此，通过计算融合节点或融合边的融合度可以发现存在上述异常的程序语句。然而，当程序语句的错误对于语句形式与语句上下文关系影响都较小，在程序融合过程中，该语句对应节点及其边与其他节点具有相同或相近的融合度。此时，上述两种方法将无法识别该错误语句。基于节点内部相似度的错误定位方法将根据节点内容，即程序语句的形式识别可能存在错误的语句。

一组在原程序中具有相同作用的程序语句应当具有较高的形式上的相似性。当其中一个或部分语句存在错误时，将会体现出与其他程序语句之间的差异。具体地，对于一组应当具有相同功能的程序语句，错误语句与正确语句之间的相似度应当低于正确语句与正确语句之间的相似度。在程序依赖融合图中，对于一个融合节点，如果其中包含的部分被融合节点与其他节点间存在较明显的相似度差异，则这些节点对应的程序语句可能存在错误。

1) 节点的内部相似度

融合节点的内部相似度表示在一个融合节点中，某一被融合节点与所有被融合节点的总体相似度。具体地，对于融合节点v^m中的一个被融合节点v，节点v在v^m中的内部相似度$Sim_{inner}(v, v^m)$定义为：

$$Sim_{inner}(v, v^m) = \frac{\sum_{v' \in v^m} Sim(v, v')}{|v^m|} \tag{4.1}$$

其中，$Sim(v, v')$表示节点v与v'的相似度，3.5 节介绍了其基于抽象语法树结构的计算方法。节点内部相似度表示了一个被融合节点与其所在融合节点中其他被融合节点的相似度情况。内部相似度越大，则该节点与其他节点越相似；内部相似度越小，则该节点相比于其他节点越“异常”。在一个融合节点中，如果存在部分被融合节点，其内部相似度明显低于其他节点，则基于群体对程序的正确形式达成共识的思想，这些被融合节点可能代表着程序错误。

上述内部相似度的“异常”判断针对的是被融合节点内部相似度之间的相对关系，而其具体值则根据程序语句的不同可能存在不同的取值范围，因此无法作为异常的判断依据。以如下情况为例：融合节点v_1^m与v_2^m分别包含 5 个被融合节点，各自以编号 1 至 5 表示。在v_1^m与v_2^m中，编号 5 的节点均存在异常。这些节点两两之间的相似度如表 4.1 与表 4.2 所示。

表 4.1　v_1^m的内部相似度

节点编号	1	2	3	4	5
1	1.00	1.00	1.00	1.00	0.80
2	1.00	1.00	1.00	1.00	0.80
3	1.00	1.00	1.00	1.00	0.80
4	1.00	1.00	1.00	1.00	0.80
5	0.80	0.80	0.80	0.80	1.00
内部相似度	0.96	0.96	0.96	0.96	0.84

表 4.2　v_2^m的内部相似度

节点编号	1	2	3	4	5
1	1.00	0.80	0.80	0.80	0.50
2	0.80	1.00	0.80	0.80	0.50
3	0.80	0.80	1.00	0.80	0.50
4	0.80	0.80	0.80	1.00	0.50
5	0.50	0.50	0.50	0.50	1.00
内部相似度	0.78	0.78	0.78	0.78	0.60

在v_1^m中，节点 1 至 4 两两之间相似度为 1，即其程序语句相同。与其相比，节点 5 存在部分相异之处，使得节点 5 与其他节点之间相似度均为 0.8。在错误定位过程中，根据上述节点内部相似度公式，节点 1 至 4 在v_1^m中内部相似度为 0.96，节点 5 为 0.84。因此推断节点 5 存在错误。

在v_2^m中，节点 1 至 4 间存在部分差异，其两两之间相似度为 0.8。与其相比，节点 5 存在更大差异，其与其他节点之间相似度均为 0.5。在错误定位过程中，根据上述公式可计算，节点 1 至 4 在v_2^m中内部相似度为 0.78，而节点 5 为 0.6。尽管节点 1 至 4 不完全相同，但节点 5 存在更明显的异常，因此推断节点 5 存在错误。

对比v_1^m与v_2^m，v_1^m中的节点 5 具有 0.84 的内部相似度，而这一数值高于v_2^m中所有节点的内部相似度。但由于其仍与v_1^m中其他节点的内部相似度 0.96 存在较大差距，因此仍可推断v_1^m中节点 5 存在错误；相对的，v_2^m中节点 1 至 4 不会因为 0.78 的内部相似度被判断为错误。

2) 基于相似度聚类的差异识别

基于节点内部相似度的错误定位根据内部相似度之间的相对差异对可能的错误进行推断。对于一个融合节点，当其中包含的部分被融合节点的内部相似度明显低于其他被融合节点时，这些节点对应的程序语句则可能存在错误。为了在一组被融合节点中寻找内部相似度偏低的节点，本文采用基于相似度的聚类方法。具体地，对于被融合在某个融合节点内的一组被融合节点，算法尝试对这些节点依据其内部相似度归为 2 类，即内部相似度“正常”的节点类与内部相似度“偏低”的节点类。当算法成功将这些节点分类，且两类节点的平均内部相似度之间有足够明显差距时，则认为内部相似度“偏低”的节点类中节点存在异常。

本文采用 K-means 算法 [28]对节点依据内部相似度进行聚类。K-means 算法将一组元素划分为 k 个聚类，使得每个元素到其聚类中心的距离最小。K-means 算法需要以目标聚类数 k 作为输入参数。在本问题中，目标的聚类数 k 值为 2，对应“正常”与“异常”2 类节点。被聚类元素，即被融合节点之间的距离以内部相似度差的绝对值计算。算法初始生成 2 个聚类均值点，即两个内部相似度的值，在本文中设定为一组被融合节点的内部相似度的最小值与最大值。之后算法将每个被聚类元素，即每个被融合节点分配到距离最近的均值点所代表的聚类。对于每个聚类，算法随后依据每个聚类中被融合节点的内部相似度均值重新设定聚类的均值点。算法重复迭代上述过程，直至聚类均值点不再发生变化，此时算法收敛于某一（局部）最优解。

算法结束后，两个聚类均值点即分别代表可能的“正常”与“异常”分类。此时，若两个聚类存在较大的区分度，则认为这一组被融合节点中存在内部相似度明显偏低的节点，对应于内部相似度均值较小的聚类中的节点。对于“较大区分度”，算法设定为当两个聚类均值点的均值比值大于一定阈值时，认为其具有较大区分度。此时，被融合节点被分为两类，其中内部相似度均值较小的一类节点对应的程序语句可能存在错误；相反地，若区分度不足，则认为所有被融合节点都相似，不存在异常节点。

基于节点内部相似度的错误定位算法的伪代码如算法 4.1 所示：

算法 4.1 基于节点内部相似度的错误定位

输入：融合节点v^m，区分度阈值θ

输出：异常节点集合V_{fault}

Function FaultLocalization(v^m, θ)
  $SimSet \leftarrow \{Sim_{inner}(v, v^m) | v \in v^m\}$
  $mean_1 = Min(SimSet), mean_2 = Max(SimSet)$
  $V_1 = \{\}, V_2 = \{\}$
  $flag_{changed} = True$
  while $flag_{changed}$ **do**
    for each $v \in v^m$ **do**
      $dist_1 = |Sim_{inner}(v, v^m) - mean_1|$
      $dist_2 = |Sim_{inner}(v, v^m) - mean_2|$
      if $dist_1 < dist_2$ **do**
        $V_1 \leftarrow V_1 \cup \{v\}, V_2 \leftarrow V_2 - \{v\}$
      else
        $V_1 \leftarrow V_1 - \{v\}, V_2 \leftarrow V_2 \cup \{v\}$
      end if
    end for
    $mean'_1 = \sum_{v \in V_1} Sim_{inner}(v, v^m) / |V_1|$
    $mean'_2 = \sum_{v \in V_2} Sim_{inner}(v, v^m) / |V_2|$
    if $mean'_1 \neq mean_1$ **or** $mean'_2 \neq mean_2$ **do**
      $flag_{changed} =$ **True**
    else
      $flag_{changed} =$ **False**
    end if
    $mean_1 \leftarrow mean'_1, mean_2 \leftarrow mean'_2$
  end while
  if $mean_2 / mean_1 > \theta$ **do**
    $V_{fault} \leftarrow V_1$
  else
    $V_{fault} \leftarrow V_2$
  end if
  return V_{fault}
end Function

4.3 本章小结

本章对本文提出的程序错误定位方法进行了介绍。该错误定位方法基于群体共识思

想，以群体对目标问题的正确解法大体达成共识为前提，利用程序依赖融合图，对每个被融合图结构与内容进行分析，从而发现每个程序中可能存在的异常。本章首先对基于群体共识的错误定位的基本思想进行了介绍。之后本章介绍了基于三种不同分析角度的错误定位方法，即：基于节点融合度的错误定位，基于边融合度的错误定位，基于节点内部相似度的错误定位。

第五章　实验与评估

对于本文第三章提出的基于程序依赖图的程序融合算法与第四章提出的基于群体共识的错误定位算法，本章分别设计了三类实验以验证它们的有效性。对于程序融合算法，本章设计了基于同构程序依赖图的融合实验。实验选定一个程序并复制生成一定规模的程序依赖图，对这些程序依赖图进行融合并评估融合结果。对于错误定位三算法，本章设计了基于变异程序的仿真群体错误定位实验与基于实际群体的错误定位实验。在基于变异程序的仿真群体错误定位实验中，实验选定一个程序并在其基础上产生一组包含程序错误的变异程序，之后对这一组程序进行仿真实验；在基于实际群体的错误定位实验中，实验对求解同一问题的由多名开发者提供的一组包含错误的程序进行错误定位，并对结果进行分析评估。

本章各节安排如下：5.1 节介绍基于同构程序依赖图的融合实验方案与实验结果；5.2 节介绍基于变异程序的错误定位实验方案与实验结果；5.3 节介绍基于群体程序的错误定位实验方案与实验结果。5.4 节对实验结果进行总结。

5.1　实验程序的选取

本章进行的实验中，所有实验程序来源于编程网格[①]。编程网格是一个由北京大学信息科学技术学院开发的面向编程教学环境的在线评测系统。编程网格中包含大量不同难度的编程问题，支持学生使用 C/C++/Java/Pascal 语言进行编程提交。学生提交程序后，系统根据问题的测试用例对学生程序进行测试，并反馈是否通过，若未通过，则会提示未通过原因与未通过的测试用例。在编程网格中，对于一个编程问题，系统记录了对该问题进行尝试的所有学生的全部程序提交记录。在本文指导教师与编程网格相关人员的支持下，本文获取了编程网格中部分问题的部分学生提交记录，并以此为实验程序进行程序融合与错误定位的实验。

用于实验的编程问题与学生程序的选取遵循以下条件：

1) 编程问题有至少 10 名学生曾经进行尝试，且提交过错误代码（系统反馈 Wrong Answer）并最终通过（系统反馈 Passed）。对于尝试人数过少的程序则由于难以构成符合条件的群体，无法用于实验。
2) 编程问题有一定的复杂性，但不包含复杂的输入或输出处理。对于过于简单的程序，不同学生程序中存在的错误也较为单一，不符合群体程序存在多样性的条件。而对于复杂的输入或输出处理往往与问题的核心算法无关，并非错误定位问题的重点。

① http://www.pkupc.cn/programming/

3) 程序使用 C 语言编写。本文提出的程序依赖图融合算法与基于程序融合的错误定位算法对于程序语言并无特定要求。本文的程序依赖图生成工具实现与错误定位实验均针对 C 语言程序。
4) 程序为单函数程序。为简化问题，本文的程序依赖图生成与融合算法均针对单个函数或单函数程序。对于包含多个函数的程序，则可以依据函数进行划分后分别进行程序依赖图生成与程序融合。

5.2 基于同构程序依赖图的融合实验

为了验证程序融合算法的有效性，本文设计了基于同构程序依赖图的融合实验。实验选取不同规模的三个程序，并对这些程序进行复制与变异，各自生成不同规模的同构程序。之后采用程序融合算法对三组程序分别进行融合以实验融合效果。

5.2.1 实验方案

1) 实验设置

本实验设置了 3 组程序的融合实验，分别以编程网格中不同难度的三个问题作为目标问题。在每一组中，选择一个学生提交程序作为基准程序。实验选取了三个长度不同的程序，使得三组程序的程序依赖图具有不同的节点规模。表 5.1 展示了三组程序选择的问题信息与基准程序信息，包括问题程序名称，基准程序生成的程序依赖图节点数，与目标问题的简单描述。作为基准程序的一个示例，图 5.1 展示了统计药费程序的程序依赖图。

表 5.1 程序依赖图融合实验的实验程序信息

组别	程序名	节点数	程序描述
1	细胞分裂	15	对于输入 n，计算 2 的 n 次幂
2	统计药费	20	依据给定分段计费规则计算总费用
3	前 k 大数	25	输出一个数组的前 k 大的元素

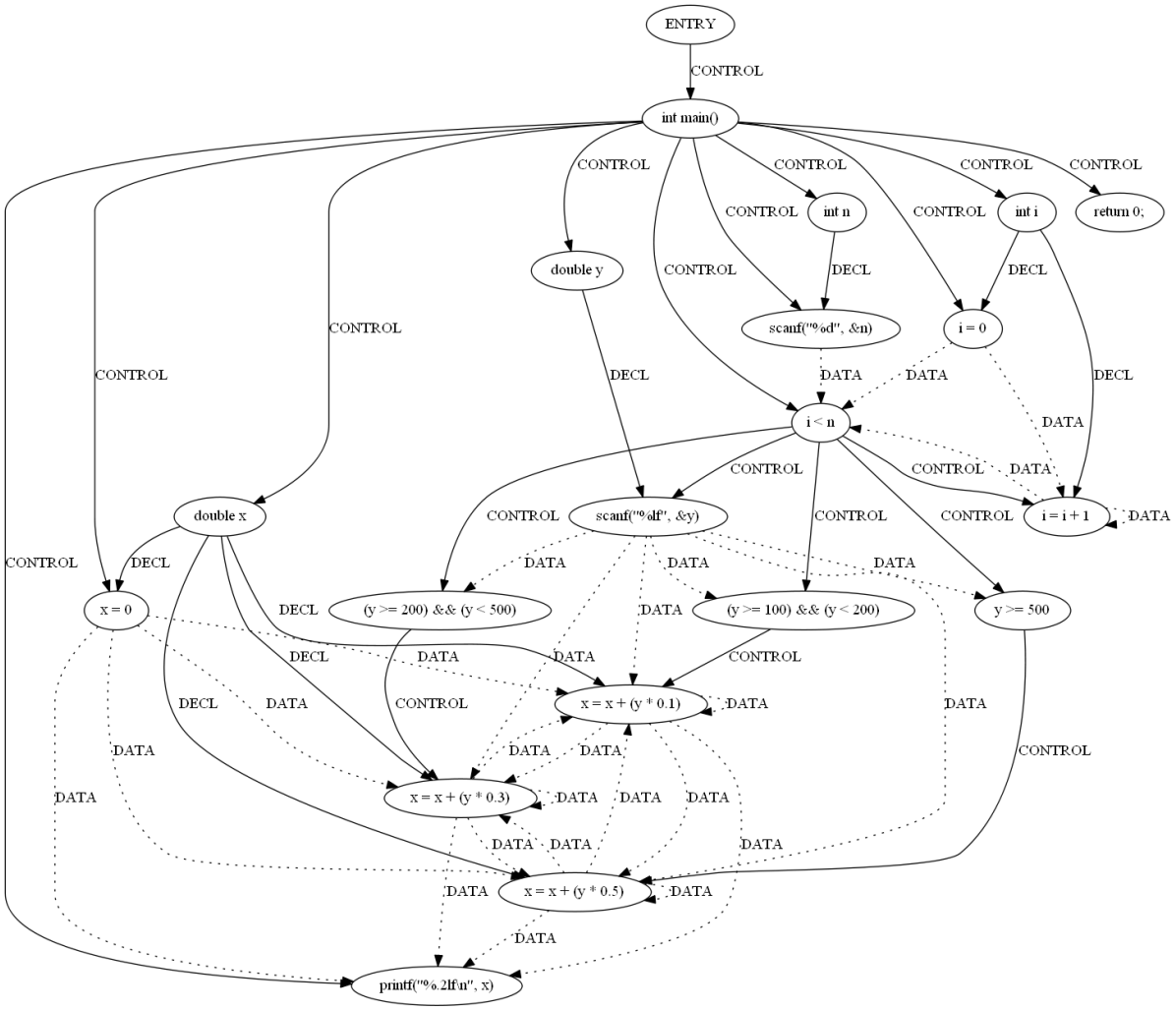


图 5.1　统计药费程序的程序依赖图

每一组内，以基准程序为基础，通过对程序代码进行随机变异生成 4 个变异程序。变异方式包括：代码风格与格式修改、变量名替换、无关语句顺序交换。在每一组内，将基准程序与变异程序进行复制，分别生成 4 个规模不同的群体，群体中程序数分别为 5、10、15、20。由此，实验共设置 3 组，每组 4 个规模不同的群体。实验对此 12 个群体分别进行 5 次融合，记录每次融合所得程序依赖融合图的节点数与广义熵，并生成相应融合图。

2)　结果预期

各组程序中，对基准程序进行的变异操作不影响程序结构与语句语义，因此各组中所有程序的程序依赖图应当两两同构，且对应节点的相似度均为 1，即各个群体中的所有程序实质上完全相同。因此各个群体中，所有程序应当能够完全融合，即融合图与原程序依赖图结构相同，且熵值为 0。

5.2.2 实验结果与分析

表 5.2 展示了每组各个群体的 5 次融合所得程序依赖融合图的平均熵值与平均融合节点数。

表 5.2 同构程序依赖图融合实验结果

程序名	群体规模	平均熵值	平均节点数
细胞分裂	5	0.00	15
	10	0.00	15
	15	0.00	15
	20	0.00	15
统计药费	5	0.00	20
	10	0.00	20
	15	0.00	20
	20	0.00	20
前 k 大数	5	0.00	25
	10	0.00	25
	15	0.00	25
	20	0.00	25

可以看出，对于所有的 12 个群体，本文的程序融合算法能够将群体中的程序稳定地融合为广义熵为 0 的最优程序依赖融合图。由于每个群体中的程序依赖图结构相同，因此程序依赖融合图中的节点数与被融合程序依赖图中节点数相同。通过人工验证，所有的程序依赖融合图均与对应的被融合图结构相同，实验结果与预期完全相符。

图 5.2 展示了对于统计药费问题，群体规模为 20 时，程序融合所得的程序依赖融合图。图中同一节点内的多条语句表示被融合在一起的不同语句，可以看出，被融合在一起的程序语句即使变量名并不完全相同，但其结构是完全一致的。且图 5.2 所示的程序依赖融合图结构与图 5.1 所示被融合图完全相同。

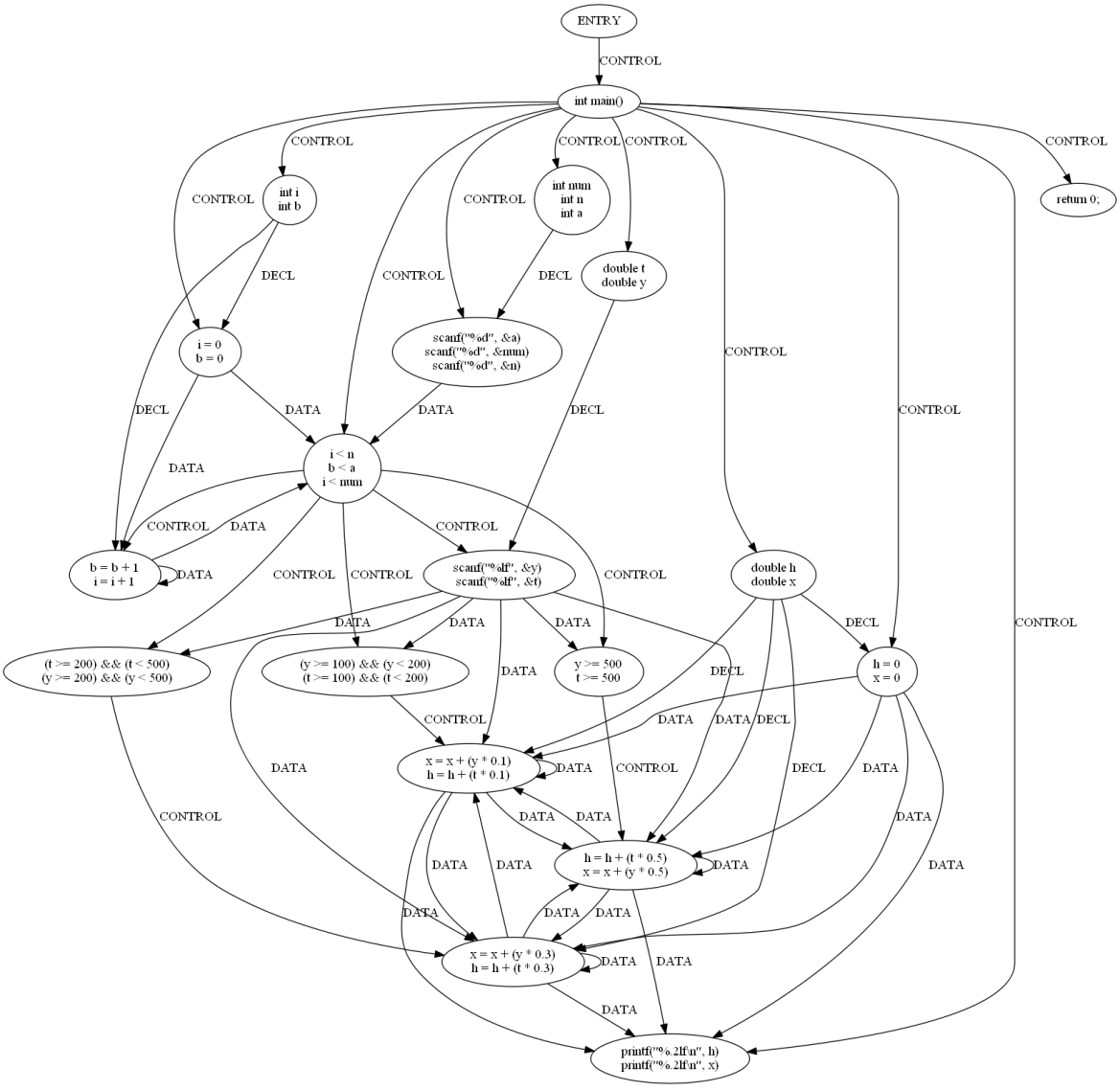


图 5.2　统计药费程序的程序依赖融合图

从上述实验结果可以看出，对于结构相同而表现形式有差异的一组程序，本文提出的程序融合算法能够达到预期的最优效果，即将语义相同的程序语句相融合，并使语义不同的语句分离。

5.3　基于变异程序的仿真群体错误定位实验

为验证错误定位算法的有效性，本文设计了基于变异程序的仿真群体错误定位实验。实验以针对某一问题的一个正确程序为基准，通过程序变异的方式生成多个含有错误的程序构成一个仿真群体，以模拟群体中每个开发者的程序均包含错误的情况。之后实验使用本文的错误定位算法对这一模拟群体中的程序进行错误定位，并分析错误定位的效果。

5.3.1 实验方案

1) 实验程序的生成

实验选取编程网格中 4 个问题作为目标问题，针对这 4 个问题分别在实际学生代码的基础上构建模拟群体。对于每个目标问题，实验从该问题的正确提交中选择一个程序作为基准程序。在基准程序的基础上，通过随机变异产生错误的方式生成 10 个变异程序。变异遵循如下原则：

1) 变异方式模拟实际程序开发中可能出现的错误模式；

2) 变异不对大段程序进行重写；

3) 变异后的程序不存在语法错误，即保证变异程序能够通过编译并执行。

变异主要包含以下方式：程序语句的修改，包括变量、运算符、表达式结构的修改等；程序语句的插入与删除；程序语句的移位。此外，程序变异过程保证生成的 10 个变异程序中至多有 1 个正确程序。实验将 10 个变异程序构成模拟群体，并在该群体中采用本文提出的算法进行错误定位。

```
If (a<100) {
    a=0;
} else if (a>=100 && a<200) {
    a=0.1*a;
} else if (a>=200 && a<500) {
    a=0.3*a;
} else if (a>=500){
    a=0.5*a;
}
```

(a) 变异前

```
If (a<100) {
    a=0;
} else if (a>=100 && a<200) {
    a=0.9*a;
} else if (a>=200 && a<500) {
    a=0.7*a;
} else if (a>=500){
    a=0.5*a;
}
```

(b) 变异后

图 5.3 程序变异示例

图 5.3 展示了对统计药费的一个程序的变异示例。图(a)所示是变异前的程序片段。图(b)所示是对该片段中 a=0.1*a 与 a=0.3*a 两条语句进行变异后所得程序片段，图中以灰色标注了变异语句位置。该变异模拟了学生在开发过程中，对目标问题理解有误而导致的在表达式中使用了错误的常数的情况。在之后进行的实验的程序提取过程中，也的确发现了部分学生程序存在此错误。

2)　错误定位的评估方法

对于每一个变异程序，在变异过程中记录其中能够引起程序错误的变异语句及其位置。直观上，这些导致了错误的变异位置是错误定位的“正确答案”。错误定位算法对可能存在错误的程序语句进行反馈。当反馈的语句与记录的变异位置语句相符时，认为该反馈正确。对于一组程序构成的群体，令S_{fault}表示群体所有程序中导致了程序错误的变异语句的集合，令$S_{feedback}$表示错误定位算法反馈的可能存在错误的程序语句集合。则针对此模拟群体，可以计算错误定位算法的精确率(Precision)与召回率(Recall)。与 F 值。精确率与召回率描述了错误定位算法的差准率与查全率，而 F 值为精确率与召回率的调和平均值。直观上，当精确率与召回率越高，说明错误定位算法的效果越好。具体地，精确率，召回率与 F 值计算方式如下：

$$Precision = \frac{|S_{fault} \cap S_{feedback}|}{|S_{feedback}|} \tag{5.1}$$

$$Recall = \frac{|S_{fault} \cap S_{feedback}|}{|S_{fault}|} \tag{5.2}$$

$$F_1 = 2\ \frac{Precision \times Recall}{Precision + Recall} \tag{5.3}$$

3)　实验设置

本实验共设置 4 组，每组以一个目标问题的一个正确程序为基准，通过变异产生 10 个不同的变异程序并构成群体。每个变异程序包含 0-4 个程序错误，其中至多有 1 个程序不包含程序错误。错误的数量以产生该错误所变异的语句数量计。表 5.3 列举了本实验中选择的 4 个目标问题及其描述。表 5.4 展示了本实验的 4 组实验群体设置，包含基准程序的语句数量，每个变异程序中的错误语句数量，以及群体中合计错误语句数。

表 5.3　仿真群体错误定位实验目标问题

组别	问题名	问题描述
1	前 k 大数	输出一个数组的前 k 大的元素
2	统计药费	依据给定分段计费规则计算总费用
3	出现奇数次的数	输入一个数组，寻找其中唯一一个出现奇数次的数
4	转换 3 进制	输入一个 10 进制数，输出其 3 进制表示

表 5.4　仿真群体错误定位实验设置

组别	问题名	基准语句数	各变异程序错误语句数										合计错误
			1	2	3	4	5	6	7	8	9	10	
1	前 k 大数	28	0	1	1	2	1	2	2	2	3	4	18
2	统计药费	23	2	3	1	1	1	4	2	3	0	1	18
3	出现奇数次的数	29	1	3	2	1	4	2	4	1	2	2	22
4	转换 3 进制	25	1	2	2	1	2	4	2	1	1	3	19

实验对每一组程序采用本文设计的算法进行错误定位，并基于前文所述的评估方法对错误定位结果进行评估。

5.3.2 实验结果与分析

实验对四组程序群体分别进行程序融合与错误定位，并记录错误定位的反馈信息。反馈信息中每一项对应于一个可能含有错误的语句。四组实验的反馈统计结果及根据结果计算的精确率、召回率、F 值如表 5.5 所示。

表 5.5　仿真群体错误定位实验结果

问题名	总错误数	总反馈数	正确反馈数	精确率	召回率	F 值
前 k 大数+	18	19	16	0.84	0.89	0.86
统计药费	18	19	16	0.84	0.89	0.86
出现奇数次的数	22	19	18	0.95	0.82	0.88
转换 3 进制	19	20	16	0.80	0.84	0.82

可以看出，在对基于程序变异的模拟群体进行错误定位时，本文提出的程序融合与错误定位算法拥有较高的准确性，其拥有平均 80%的精确率，并能够发现群体中 84%的错误。

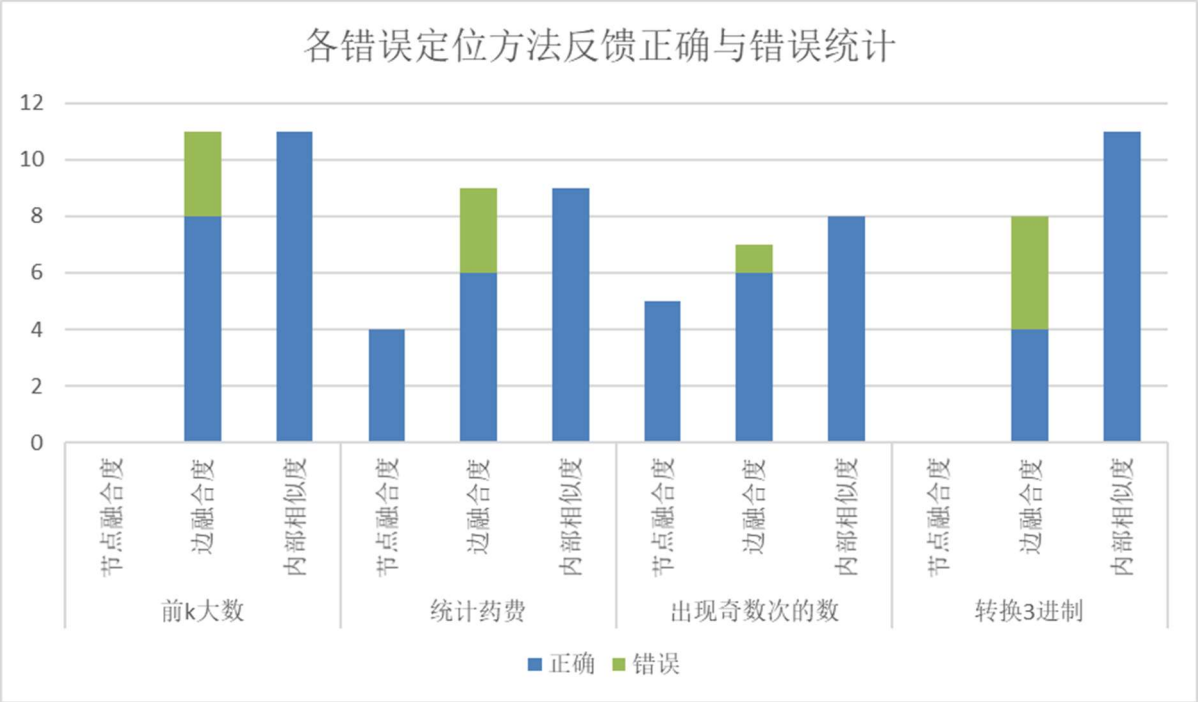


图 5.4　各错误定位方法反馈正确与错误统计

此外，实验对算法中三种不同的错误定位方法，即基于节点融合度、基于边融合度、基于节点内部相似度的错误定位方法所产生的反馈中正确与错误的数量进行了统计，如图 5.4 所示。图中显示了对于每一组程序，实验中由不同方法产生的正确反馈与错误反馈的数量。由于同一个程序错误可能能够被多种错误定位方法同时发现，因此图中各错误定位方法反馈的错误数总和会多于群体中总错误数。

可以看出，在 4 组实验中，基于节点内部相似度的错误定位方法能够发现最多的程序错误，而基于节点融合度的错误定位方法发现的错误较少。直观上，由于程序融合趋向于将程序语句融合在一起，因此只有形式或关系上具有极大异常的程序语句才会被孤立，成为融合度较低的节点；另一方面，在群体中各个程序使用相同解决方案的情况下，融合度较低的节点则有很高可能性代表着程序错误。

在此实验中，所有的错误反馈均产生于基于边融合度的错误定位方法。通过对这些反馈与对应错误程序的分析发现，当部分程序语句出现错误时，这些程序语句中产生的异常控制依赖或数据依赖边往往会对与该边相关的另一节点产生影响。直观上，当一个程序语句存在错误，使得该语句与另一语句之间产生了错误的依赖关系时，对于该依赖关系的另一端节点，这一依赖同样是一个“异常依赖”。部分情况下，基于边融合度的错误定位算法能够识别这一异常依赖，但无法正确判断该依赖的两端节点中哪一方产生了该依赖，因此产生了错误的反馈。

另一方面，基于群体共识的错误定位方法存在着局限星，对于群体中少部分程序错

误无法予以反馈。在 4 组实验中，无法反馈的程序错误分为 2 类。

1) 该程序错误由部分关键代码缺失引起。在实验中，对部分变异程序采取的变异方法是删除部分程序语句。本文提出的三种错误定位方法尚无法识别程序语句的缺失，因此无法对该类错误进行定位。
2) 群体中对于某一程序语句存在 2 种以上不同的错误形式，且错误不对该语句依赖关系产生影响时，其中与正确形式较为相近的错误无法被正确定位。在此情况下，不同形式的错误语句与正确语句在程序融合过程中被融合在一起，而错误不对依赖关系产生影响，故基于节点融合度与边融合度的错误定位方法无法识别这些错误。而基于节点内部相似度的错误定位方法在根据内部相似度进行聚类时，会将正确形式的语句与与正确形式最相近的错误形式划分到同一聚类，使得“略有错误”的程序语句无法被正确定位。

实验结果表明，对于基于变异程序的模拟群体，本文提出的程序融合与错误定位方法能够有效地对由变异产生的错误进行定位。基于群体共识的三种错误定位方法均能够对群体中的程序错误进行识别。同时，对于算法无法识别或错误识别的少量错误，本节对其原因与错误类型进行了分析与总结。

5.4 基于实际群体的错误定位实验

在基于变异程序的错误定位实验中，实验通过对一个基准程序进行变异的方式产生仿真群体，并在仿真群体上进行错误定位实验。在基于群体程序的错误定位实验中，实验选取编程网格中针对某一问题的一组包含错误学生提交程序构成实验群体，并以此群体为基础进行错误定位实验。

5.4.1 实验方案

1) 实验程序的选取

基于实际群体的实验选取与仿真群体实验中相同的 4 个问题作为目标问题，分别进行错误定位实验。对于每个问题，实验从编程网格选择曾经尝试过该问题的 10 个学生组成的群体。实验选取的 10 个学生满足如下条件：

1) 每个学生对该问题曾经提交过包含错误的程序，且在错误程序之后提交过正确通过程序。
2) 对于每个学生，提交的错误程序与之后的正确程序应采用相同的解决方案。即正确程序是该学生通过修改错误程序而得，而非放弃之前思路重新编写的程序。
3) 10 个学生对于目标问题采用了相同的思路与解决方案。

对于选定的 10 个学生，实验选取每个学生的提交记录中，最后一次错误提交程序

作为群体中的个体程序。

通过上述方法选择的 10 个学生及各自的 10 个含有错误的程序构成了一个真实的群体协同开发场景。尽管这 10 个学生的开发过程可能在时空上相异，但他们的开发过程具有高度的一致性：这些学生的开发针对同一问题，面对相同的输入条件与输出要求，使用相同的编程语言。这使得实验选取的学生符合构成协同开发的群体所需满足的条件，能够视为一个真实的群体协同开发场景。

2)　实验设置

不同于基于变异程序的模拟实验，在实际的群体协同开发环境下由开发者编写的含有错误的程序中，造成程序错误的语句往往难以界定。另一方面，对于同一个程序错误，很多时候也会有多种修复方式，这些修复方式所修改的程序语句也不相同。因此，在基于实际群体的错误定位实验中，由于难以对产生错误的程序语句进行判定，因而无法简单地判定针对某一语句的错误定位正确与否。

借助学生对该问题的正确递交程序可以对学生错误程序中存在的错误进行识别，同时正确程序也代表了将该错误程序修复的一种可行方案。在基于实际群体的错误定位实验中，实验对错误定位算法产生的反馈中标识了程序错误的反馈数进行统计。此外，在程序选取过程中发现，许多学生程序中包含局部的冗余代码或过于复杂的实现。冗余代码是指部分程序语句对程序执行没有起到任何作用；复杂实现是指在程序的某功能局部，该学生使用了较为复杂的实现方式，而其他学生在相同局部存在着更优的解法。冗余代码与复杂实现尽管对程序的正确性没有影响，但其体现了该学生在问题解决方案设计上的缺陷或不严谨。另一方面，在面向实际应用的程序开发中，冗余代码与复杂实现可能对程序的执行效率产生影响。本实验对错误定反馈中，标识了冗余代码或复杂实现的反馈一并进行了统计。

具体而言，在基于实际群体的错误定位实验中，对于一个包含错误的学生程序，结合该学生对该问题的正确提交与人工检查，实验对错误程序中存在的错误数量与位置进行了标注。一处程序错误可能涉及多条程序语句。在错误定位算法反馈的可能包含错误的语句中，若一个或一组程序语句与程序中的某个错误相关，且对这些程序语句进行修改能够构成针对该错误的一个修复方案，则认为这一个或一组程序语句标识了该程序错误，其对应的反馈为正确反馈。若将一个或一组反馈中的程序语句删除后，对程序功能不产生影响，则认为这一个或一组程序语句对应的反馈标识了一处冗余代码。若对于一组程序语句，其实现的程序功能能够被群体中其他程序中存在的一组程序语句替代，且进行替代的一组程序语句拥有更少的语句数，则认为被替代的一组程序语句是可能的较复杂实现，其对应的反馈标识了一处复杂实现。对于上述三种情况，实验认为反馈有效；相反，对于不符合上述任何一种的错误定位反馈，则认为是错误反馈。

本实验共设置 4 组，选取的目标问题与仿真群体实验相同，如表 5.3 所示。对于每一组，实验依照上文所属标准选取 10 个不同学生提交的错误程序构成实验群体，并针对该群体使用本文设计的错误定位算法对每个错误程序进行错误定位。对于错误定位算法产生的反馈依据上述标准进行分类，并对各类型的反馈数进行统计与比较，以对错误定位的质量进行评估。

5.4.2 实验结果与分析

表 5.6 实际群体实验正确反馈统计

问题名	群体错误数	反馈识别的错误数	识别率	正确反馈数
前 k 大数	17	11	0.65	17
统计药费	23	18	0.78	20
出现奇数次的数	13	8	0.62	11
转换 3 进制	16	12	0.75	19
合计	69	49	0.71	67

表 5.6 展示了实际群体实验中对群体中错误识别情况。对于每个目标问题，表 5.6 记录了群体中存在的错误数量与通过错误定位算法反馈正确识别的错误数量并计算了识别率，同时记录了标识这些正确识别的反馈数量，即正确的反馈数。由于一个程序错误可能涉及多个程序语句，故正确的反馈数量多于正确识别的错误数。可以看出，基于群体共识的错误定位算法能够对群体中的错误进行较有效的识别，平均能够识别群体中 71%的程序错误。对于不同问题或不同群体，算法的正确识别率会有较大差异，后文将对产生这种差异的原因进行分析。

表 5.7 实际群体实验反馈类型统计

问题名	正确反馈	冗余代码	复杂实现	不正确反馈	反馈合计	反馈正确率	反馈有效率
前 k 大数	17	1	1	7	26	0.65	0.73
统计药费	20	1	1	9	31	0.65	0.71
出现奇数次的数	11	5	5	13	34	0.32	0.62
转换 3 进制	19	2	1	9	31	0.61	0.71
合计	67	9	8	38	122	0.55	0.69

表 5.7 展示了各组实验中，各类型反馈的数量统计，包括正确识别了程序错误的反馈，识别了冗余代码的反馈，识别了复杂实现的反馈与未能正确识别的不正确反馈。对于正确反馈、识别冗余代码的反馈与识别复杂实现的反馈，实验认为反馈实际有效，即能够对程序优化产生帮助。据此，表 5.7 统计了各组实验的反馈正确率与有效率。可以看出，基于群体共识的错误定位算法产生的反馈中平均具有 69%的有效率。但对于“出现奇数次的数”这组群体，反馈的正确率较低，后文将对这一现象进行分析。

表 5.8　各错误定位方法反馈有效与无效统计

问题名	节点融合度		边融合度		内部相似度	
	有效	无效	有效	无效	有效	无效
前 k 大数	3	1	9	4	13	4
统计药费	0	0	4	7	19	2
出现奇数次的数	11	0	2	4	9	9
转换 3 进制	6	0	10	6	10	6
合计	20	1	25	21	51	21

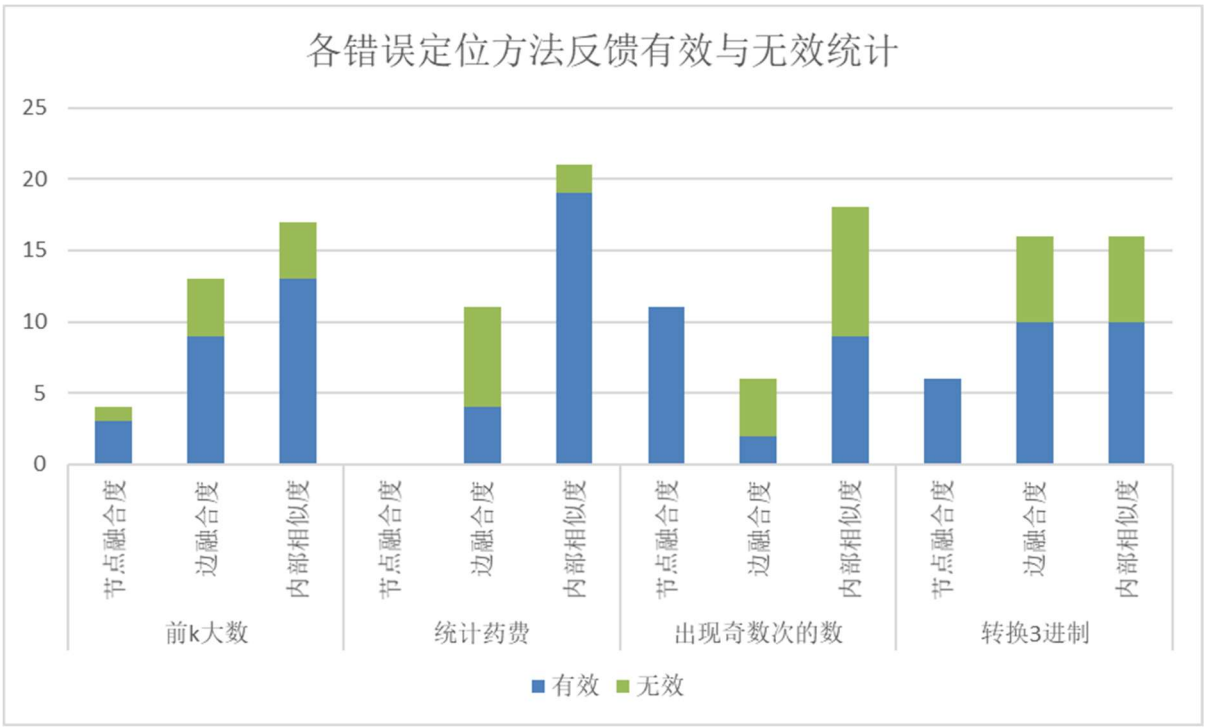


图 5.5　各错误定位方法反馈有效与无效统计

表 5.8 与图 5.5 展示了不同错误定位方法产生的反馈的有效与无效数统计，其表

示了 3 种错误定位方法在 4 组实验中分别产生的有效反馈与无效反馈数。可以看出，在“出现奇数次的数”以外的其他 3 组实验中，基于节点内部相似度的错误定位方法能够产生数量最多的有效反馈；基于节点融合度的错误定位方法产生的反馈较少，但有效率较高；基于边融合度的错误定位算法同时产生了较多的有效与无效反馈。在“出现奇数次的数”的实验中，基于节点融合度的错误定位产生了较多的有效反馈，而另两种错误定位算法产生的有效反馈较少，且产生了较多的无效反馈。

在针对“出现奇数次的数”问题的实验中，实验结果显示了明显低于其他组实验的反馈质量。对该组的群体程序与由群体程序融合所得的程序依赖融合图分析发现：相对于其他 3 组群体，该组群体的程序之间相似性较低。从而使得程序依赖融合度的融合程度较低。尽管该组群体中各个程序所基于的解决思想与解决方案相同，但其在实现上有较大的自由度。这使得群体中各个程序在一些局部存在较大的多样性。即，群体在这些局部未能达成足够的共识。因此，基于群体共识的错误定位算法对于低共识度的程序局部未能正确识别程序中的错误，从而产生了较多无效反馈。

表 5.9　各群体程序融合程度

问题名	融合图广义熵	融合图节点数	被融合图平均节点数	节点融合比
前 k 大数	5016.26	32	28.8	1.11
统计药费	9915.47	27	22.6	1.19
出现奇数次的数	15255.81	40	25.6	1.56
转换 3 进制	7470.56	33	25.4	1.30

表 5.9 展示了 4 组实验中，各群体中程序的融合情况。对于每个群体，程序的融合程度通过程序依赖融合图的广义熵、程序依赖融合图的节点数与节点的融合比刻画。其中，节点的融合比定义为融合图的节点数与被融合图的平均节点数的比值，是一个大于 1 的值。直观上，节点的融合比描述了程序融合过程中节点的融合程度，融合在一起的节点越多，融合比越低；融合在一起的节点越少，融合比越高。当所有被融合图节点数一致且相互完全融合时，融合比为 1。融合图的广义熵描述了群体程序之间的分歧程度，节点的融合比描述了群体程序间融合的程度。可以看出，对于“前 k 大数”问题，群体中的分歧程度最小；对于“出现奇数次的数”问题，群体分歧程度最高。相应的，结合表 5.7 可以看出，在“前 k 大数”问题中，群体程序的融合程度也较高，节点融合比为 1.11，错误定位算法的反馈有效率也较高，为 0.73；在“出现奇数次的数”问题中，群体程序融合程度较低，节点融合比为 1.56，错误定位算法有效率低，为 0.62。

实验结果表明，基于群体共识的错误定位算法能较有效地发现群体程序中存在的错

误或待改进之处，但其效果受到目标问题与群体程序的影响。对于分歧程度较低，共识度较高的群体，基于群体共识的错误定位算法的反馈有效率更高；相反，对于分歧程度较高的群体，由于群体共识度较低，错误定位算法的反馈有效率较低。

5.5 本章小结

本章通过三个不同的实验分别验证了程序融合算法与错误定位算法的有效性，分别为基于同构程序依赖图的融合实验、基于变异程序的仿真群体错误定位实验、基于实际群体的错误定位实验。

在基于同构程序依赖图的融合实验中，实验对不同规模的程序在不同规模的群体下进行了融合。实验表明本文提出的程序融合算法能够有效而稳定地对程序进行融合，且程序中对结构无影响的形式改变不会影响程序融合结果。在基于变异程序的仿真群体错误定位实验中，实验针对多组问题，以真实程序为基础，通过程序变异构造多个包含程序错误的变异程序，组成了仿真群体，并对该仿真群体进行错误定位实验。实验表明，基于群体共识的错误定位算法可以有效地对仿真群体中的程序错误进行定位与反馈。在基于实际群体的错误定位实验中，实验以在线评测系统编程网格中的真实学生程序组成群体，并对该群体进行错误定位实验。在对实际群体的错误定位中，基于群体共识的错误定位算法在发现程序错误的同时，也能够对程序中存在的冗余代码与待优化的实现进行反馈。实验结果表明，基于群体共识的错误定位算法可以对真实群体中71%的错误进行识别，所有的反馈中 69%的反馈能够对程序提出有效的优化提示。

综上所述，本文提出的程序融合与错误定位算法在仿真群体与实际群体中均可有效地对群体中的程序错误进行识别。使用本文提出的程序融合与基于群体共识的错误定位算法在群体协同开发场景中对开发者进行辅助是可行的。

第六章　总结与展望

6.1　本文总结

针对群体协同开发的场景，以对群体中每个个体的开发过程提供辅助为目标，本文设计了一个基于程序依赖图融合的错误定位算法并进行了实现。该算法首先将群体中各开发者编写的程序转换为程序依赖图表示。之后，基于遗传算法框架，算法对群体的程序依赖图进行融合，生成程序依赖融合图。在程序依赖融合图的基础上，算法基于群体共识思想，对每个个体程序进行错误定位。此外，本文设计了三类实验对算法的有效性进行了验证。

本文主要完成了以下工作：

第一，本文提出了一种以程序依赖图为表示方式的程序融合算法。该算法将程序转化为程序依赖图表示，以广义熵作为融合质量的度量方式，通过遗传算法对最优融合方案进行搜索。其中，本文提出了用于程序融合问题的程序依赖图融合方案定义与表示方式——即程序依赖融合图的概念，并以程序依赖融合图为基础，设计了用于遗传算法的程序依赖融合图编码方法、交叉与变异算法。本文提出了基于抽象语法树融合程序语句相似度计算方法，并将其与广义熵结合，作为遗传算法的适应度函数。

第二，本文设计了以程序依赖融合图为基础的基于群体共识的错误定位算法。针对群体协同开发的场景，该算法基于群体对同一目标问题的解决方案形成共识的基本思想，通过识别群体与个体之间的差异性对个体进行错误定位。本文提出了基于该思想的三种错误定位方法，即基于节点融合读的错误定位，基于边融合度的错误定位，和基于节点内部相似度的错误定位。

第三，本文设计了三种实验以验证程序融合算法与错误定位算法的有效性，分别为：基于同构程序依赖图的融合实验、基于变异程序的仿真群体错误定位实验与基于实际群体的错误定位实验。在基于同构程序依赖图的融合实验中，实验在多种规模的群体中对不同规模的同构程序进行融合，通过融合图的广义熵与结构说明本文的程序融合算法可以有效地对群体中的程序进行融合。在基于变异程序的仿真群体错误定位实验中，实验通过局部程序修改的方式基于基准程序生成多个变异程序组成仿真群体，并在仿真群体中进行错误定位。实验结果表明在仿真群体中，本文提出的错误定位算法能有效地发现群体中程序存在的错误。在基于实际群体的错误定位实验中，实验以在线评测系统编程网格中的学生提交程序构成群体，并在其基础上进行错误定位。实验结果表明，在实际群体开发过程中，采用本文提出的算法对群体中程序进行错误定位是可行的。

6.2　进一步工作设想

针对当前工作的不足之处，进一步工作可以从以下几个方面展开：

第一，融合算法效率的提升。本文提出的程序融合算法的运行时间会随着程序规模与群体规模的扩大而快速增长。算法优化的一个关键方向，即如何在更大规模的程序融合问题上保持可接受的运行时间。进一步地，由于程序融合的目的是应用于群体开发过程的信息反馈，一个理想的状态是程序融合的运行速度能够支持在开发过程中进行快速反馈。

第二，错误定位算法的优化。本文提出了三种不同的基于群体共识的错误定位算法，然而错误定位的效果对于群体达成共识的程度有较强的依赖。错误定位算法的优化目标即为降低算法对群体共识程度的依赖的同时更精准地发现程序中的错误。此外，本文的错误定位算法没有借助程序代码以外的其他信息，若对测试用例等其他辅助信息加以利用，则可以进一步提升错误定位效果。

参考文献

[1] 张伟，梅宏，“基于互联网群体智能的软件开发:可行性、现状与挑战,” *中国科学:信息科学,* 12 2017.

[2] E. Bonabeau, M. Dorigo and G. Theraulaz, Swarm Intelligence: From Natural to Artificial Systems, Oxford university press, 1999.

[3] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," *Proceedings of the Sixth International Symposium on. IEEE,* pp. 39-43, 1995.

[4] D. Karaboga, "An idea based on honey bee swarm for numerical optimization," Erciyes university, engineering faculty, computer engineering department, 2005.

[5] D. M, M. V and C. A, "Ant system: optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics,* vol. 26, no. 1, pp. 29-41, 1996.

[6] V. Ahn, Lui, B. Maurer, C. McMillen, D. Abraham and M. Blum, "recaptcha: Human-based character recognition via web security measures," *Science,* vol. 321, no. 5895, pp. 1465-1468, 2008.

[7] K. Lakhani, D. Garvin and E. Lonstein, *Topcoder (a): Developing software through crowdsourcing,* 2010.

[8] S. Weber, "The success of open source. Harvard University Press," Harvard University Press, 2004.

[9] A. M. S. Laurent, Understanding Open Source and Free Software Licensing, O'Reilly Media, Inc., 2004.

[10] S. S. Levine and M. J. Prietula, "Open Collaboration for Innovation: Principles and Performance," *Organization Science,* pp. 1414-1433, 30 12 2013.

[11] M. Johnson, "What Is A Pull Request?," 8 11 2013. [Online]. Available: http://oss-watch.ac.uk/resources/pullrequest. [Accessed 23 4 2018].

[12] M. J. Harrold, B. Malloy and G. Rothermel, "Efficient construction of program dependence graphs," *ACM SIGSOFT Software Engineering Notes,* vol. 18, pp. 160-170, 1993.

[13] J. Krinke, "Identifying similar code with program dependence graphs," in *Reverse Engineering, 2001. Proceedings. Eighth Working Conference on*, 2001.

[14] C. Liu, C. Chen, J. Han and P. S. Yu, "GPLAG: detection of software plagiarism by program dependence graph analysis," pp. 872-881, 2006.

[15] R. a. Cochran, L. D'Antoni, B. Livshits, D. Molnar and M. Veanes, "Program Boosting: Program Synthesis via Crowd-Sourcing," *ACM SIGPLAN Notices,* vol. 50, no. 1, pp. 677-688, 2015.

[16] Horwitz, Susan, J. Prins and T. Reps, "Integrating noninterfering versions of programs,"

ACM Transactions on Programming Languages and Systems (TOPLAS), pp. 345-387, 1989.

[17] K. O, M. T, M. V, H. W and P. N, "Topological network alignment uncovers biological function and phylogeny," *Journal of the Royal Society Interface,* 2010.

[18] M. Weiser, "Program slicing," in *Proceedings of the 5th international conference on Software engineering*, 1981.

[19] J. W. Laski and B. Korel, "A data flow oriented program testing strategy," *IEEE Transactions on Software Engineering,* pp. 347-354, 1983.

[20] M. Renieres and S. P. Reiss, "Fault localization with nearest neighbor queries," *Automated Software Engineering, 2003. Proceedings. 18th IEEE International Conference on,* pp. 30-39, 2003.

[21] W. W, F. S, L. G. C and N. T, "Automatic program repair with evolutionary computation," *Communications of the ACM,* vol. 53, no. 5, pp. 109-116, 2010.

[22] N. Suguna, "Fault Localization using Probabilistic Program Dependence Graph," *International Journal of Computer Applications,* vol. 66, 2013.

[23] C.-H. Hsiao, M. Cafarella and S. Narayanasamy, "Using web corpus statistics for program analysis," *ACM SIGPLAN Notices,* vol. 49, no. 10, pp. 49-65, 2014.

[24] J. Ferrante, K. J. Ottenstein and J. D. Warren, "The program dependence graph and its use in optimization," *ACM Transactions on Programming Languages and Systems (TOPLAS),* vol. 9, no. 3, pp. 319-349, 1987.

[25] T. Ball, S. Horwitz and T. Reps, *Correctness of an algorithm for reconstituting a program from a dependence graph,* University of Wisconsin-Madison, Computer Sciences Department, 1990.

[26] 王诗君, *基于环境激发效应的概念建模工具的设计与实现,* 北京大学, 2016.

[27] M. Mitchell, An introduction to genetic algorithms, MIT press, 1998.

[28] S. Lloyd, "Least squares quantization in PCM," *IEEE transactions on information theory,* vol. 28, no. 2, pp. 129-137, 1982.

致谢

感谢北京大学信息科学技术学院软件工程研究所，使我有机会在软件工程领域学习并进行研究。软件工程研究所为我提供了优越的学习与研究环境，创造了诸多接触学术前沿与参与开发工作的机会，这些都成为我将来学习与工作的坚实基础与宝贵经验。

感谢我的指导老师张伟老师与赵海燕老师。两位老师为我指明了学习与研究的方向，对我严格要求与悉心指导。两位老师以言传身教授与我看待、思考、解决问题的方法，令我受益匪浅。在毕业设计与论文写作阶段，两位老师对我的工作与论文给予了多方面的细致指导。感谢金芝老师与王千祥老师，我在学习与研究上的成长同样离不开两位老师指导与帮助。

感谢王春晖在本文的研究工作中给予的帮助。感谢赵天琪、蒋逸、赵伟泽、齐荣嵘、代彬丁等各位师兄师姐，感谢张宇霞、申博、李博远、朱霆、王佩、韦宇晗、张馨月、褚文杰、王晓敏等组内的各位同学，与你们一起，使我度过了愉快的学习与生活。感谢黎宣、李烁、王泽瑞、杨楠、杨晓波等各位师兄师姐在我本科与研究生阶段对我的诸多帮助与关照。

感谢我的室友李首扬、倪际楠与倪焱，在三年中相互激励，共同前进，使我度过了快乐而又充实的研究生生活。感谢我的诸位好友多年来的陪伴。

感谢我的父母与各位亲人一直以来对我的信任与支持，在未来的道路上我定会不断前进。

北京大学学位论文原创性声明和使用授权说明

原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不含任何其他个人或集体已经发表或撰写过的作品或成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本声明的法律结果由本人承担。

论文作者签名：　　日期：　　年　　月　　日

学位论文使用授权说明

（必须装订在提交学校图书馆的印刷本）

本人完全了解北京大学关于收集、保存、使用学位论文的规定，即：

- 按照学校要求提交学位论文的印刷本和电子版本；
- 学校有权保存学位论文的印刷本和电子版，并提供目录检索与阅览服务，在校园网上提供服务；
- 学校可以采用影印、缩印、数字化或其它复制手段保存论文；
- 因某种特殊原因需要延迟发布学位论文电子版，授权学校□一年/□两年/□三年以后，在校园网上全文发布。

（保密论文在解密后遵守此规定）

论文作者签名：　　导师签名：

日期：　　年　　月　　日