南京大学

研究生毕业论文
（申请硕士学位）

论文题目 基于概率推断的算法编程作业自动评价技术研究

作者姓名 王智楷

专业方向 计算机科学与技术

研究方向 分析测试

指导教师 许蕾 副教授

2022 年 4 月 8 日

学　　号: MG1933064

论文答辩日期: 2022 年 4 月 8 日

指 导 教 师:　　(签字)

Probabilistic Inference based Feedback Generation for Online Judge Student Assignments

by

Zhikai Wang

Supervised by

Associate Professor Lei Xu

A dissertation submitted to

the graduate school of Nanjing University

in partial fulfilment of the requirements for the degree of

Master

in

Computer Science and Technology

department of Computer Science and Technology

Nanjing University

April 8, 2022

南京大学研究生毕业论文中文摘要首页用纸

毕业论文题目：基于概率推断的算法编程作业自动评价技术研究

计算机科学与技术 专业 2019 级硕士生姓名：王智楷

指导教师（姓名、职称）：许蕾 副教授

随着计算机专业招生人数扩大，高校教学压力激增，计算机教学形式的数字化转型迫在眉睫。通过代码在线评判系统（Online Judge, OJ）训练并测试学生的编码能力成为计算机教学数字化过程中不可或缺的一环。代码在线评判系统通过基础的程序分析技术来评估学生的提交代码，然而仅仅告知错误或无法通过的测试用例无法帮助学生理解错误的根本原因；现有反馈生成工作通过比较错误学生代码和正确教师版本，寻找错误导致的差异来提供更细节的反馈，然而他们常常无法区分语义差异和错误导致的差异，导致反馈生成失败；现有对齐工作 Apex 通过符号分析区分两种差异，但是依赖于精确度不足的序列对齐结果，当序列对齐错误时会连锁地导致后续迭代对齐的错误，进而生成低质量的错误分析反馈。

本文工作致力于改进大课堂编程教学背景下的学生错误代码反馈信息不充分、现有工作的对齐精确度低和反馈生成质量低下等问题，提高反馈质量，帮助学生理解错误代码的因果链，并且给予修复意见。针对 C 语言 OJ 代码中错误代码和正确版本对齐和反馈生成问题，本文的主要贡献有：

（1）针对错误代码和正确版本的对齐问题，考虑到对齐是一个不确定的过程，提出了一种基于信念传播的代码自动对齐算法，并实现了工具 BpAlign（**B**elief **P**ropagation based **Align**ment）。首先通过动态分析提取错误代码和正确版本的跟踪信息，并基于符号分析和序列对齐算法将错误代码和正确版本的跟踪信息进行初始对齐；随后在优化对齐中，通过依赖扩充将符号表达式、动态运行值、依赖关系和结构信息建模为概率约束，利用信念传播算法获得边缘概率分布，其结果是最大化概率约束满足的后验对齐；最后通过最终对齐算法获得对齐结果。

（2）针对反馈生成问题，提出了一种基于信念传播的代码自动反馈生成算法，并实现工具 Bpex（**B**elief **P**ropagation based **E**rror E**x**planation）。首先利用群策群力的优势，对正确代码聚类并为错误代码选取相似正确版本；对于 BpAlign

的对齐结果，从错误代码和正确版本的输出实例出发寻找关键实例，基于关键实例构建动态切片；然后以对齐结果为参考将动态切片构建为动态切片对齐图；最后利用启发式规则生成反馈文档。

（3）为了评估方法有效性，从 *CODECHEF* 上收集了 5679 个 C 语言提交代码，包括 3076 个错误学生代码，并人工标注了具有代表性的 28 组错误代码和正确版本的对齐实例对，与反馈生成基线方法 Apex 和 Clara 比较。实验显示，BpAlign 的精确率达到 97%，远高于 Apex 的 87%，并且实验充分验证了搜索范围、概率推断规则和概率值选取对结果的影响；Bpex 对所有错误提交的平均反馈生成率为 95.45%，对于标记数据集，Clara 无法为 28 组数据中的 20 组生成反馈，对于其余 8 组学生代码，Bpex 的平均用户评价高于 Clara 和 Apex。实验表明，Bpex 具有高反馈生成率，比现有反馈生成方法更能帮助学生理解错误。

关键词：编程教学；在线评价；动态分析；反馈生成；概率推断

南京大学研究生毕业论文英文摘要首页用纸

THESIS: Probabilistic Inference based Feedback Generation for Online Judge Student Assignments

SPECIALIZATION: Computer Science and Technology

POSTGRADUATE: Zhikai Wang

MENTOR: Associate Professor Lei Xu

With the surge in teaching pressure caused by the expansion of computer-related majors, the digital transformation of computer teaching is imminent. Training and testing students' coding ability through the Online Judge(OJ) system has become an indispensable part of computer science teaching. They typically evaluate the student implementations by basic program analysis techniques. However, simple feedback such as "wrong answer" or failed test cases often fails to help students understand the root cause of the fault. Existing methods aim to provide informative feedback by comparing buggy and correct implementations. However, they have difficulty handling the buggy and semantic differences between versions, preventing them from generating feedback. APEX, a state-of-the-art technique that tolerates differences, generates low-quality explanations due to deterministic alignments.

This paper aims at solving the problems of insufficient OJ feedback, low alignment accuracy, and low feedback quality, improving the feedback, assisting students in understanding the causality of faults, and giving repair suggestions. This paper focuses on the alignment of correct and buggy submissions and the automatic feedback generation. And the main contributions are as follows:

(1) Aiming to solve the problem of alignment of correct and buggy submissions, we propose a belief propagation based algorithm, to tolerate differences and uncertainty. We implement BPALIGN(**B**elief **P**ropagation based **Align**ment). Firstly, we collected execution traces from the buggy and correct programs, and we performed the initial alignment of two symbolic traces with a sequence alignment algorithm. Secondly, in refine alignment, we modeled the information of symbolic expression equivalence, data and control dependence relations, and structures into probabilistic constraints, and

solved it with belief propagation. The solution is a posterior alignment that maximizes the satisfaction of the probabilistic constraints. Finally, we got the results with the final alignment algorithm.

(2) Aiming to solve the problem of automatic generation of feedback, we propose a generation algorithm based on belief propagation, and implemented BPEX(**B**elief **P**ropagation based **E**rror E**x**planation). To use the wisdom of the crowd, we clustered the correct submissions and pick a most similar one for buggy submission. Then, given the result of alignment, we performed slice from the critical instances of output instances to construct dynamic slices. We built DSAG with these slices based on the result of alignment. Finally, we performed feedback generation with heuristic algorithm.

(3) To evaluate the effectiveness of the method, the paper evaluated the 5679 submissions from *CODECHEF*, including 3076 buggy submissions. We selected 28 pairs of representative submissions and marked the alignments manually. We compared with an existing baseline method of APEX and CLARA. The result shows that BPALIGN has an average precision of 97%, whereas APEX has only 87%. The evaluation of search space and probabilities revealed its influence on performance and accuracy, inference rules revealed their effectiveness. BPEX has an average generation rate of 95.45%. In the 28 pairs of submissions, CLARA failed to generate feedback for 20 buggy submissions. For the 8 rest submissions, BPEX received higher grades than CLARA and APEX. Compared with the baseline method, BPEX can better help students understand the causal of faults.

Keywords: Computer Science Education; Online Judge; Dynamic Analysis; Feedback Generation; Probabilistic Inference

目　　录

插图目录

表格目录

第一章　引言

本章概述全文。首先阐明在计算机教学过程中为算法编程作业自动生成反馈的研究背景与意义；接着介绍本文的主要内容，包括自动反馈生成领域中三种常见的生成技术的国内外研究现状；最后给出本文的组织结构。

1.1　研究背景与意义

随着计算机专业招生人数扩大[1]，教学者面临的教学压力激增，2017 年的一篇调研将计算机线上编程教学中最常出现的难点划分为 3 个类别，分别与教师、学生和教学资源相关。教师的难点在于由教学资源缺乏导致的耗时耗力活动，包括分析学生提交代码中的各类错误并给出反馈、与学生进行一对一的解题指导、对学生代码给出公正的评分等。与之相对应，学生关心如何理解代码中的错误、如何提升自身的解题编程能力和自己的课程分数是否与自身努力和表现一致等。教学资源的难点在于缺少编程教学相关的硬件设备、线上平台等。其中 54.6%（411/752）的难点可以通过数字化的编程教学辅助工具来解决[2]。

编程教学辅助工具通常用于协助教学者对学生作业给予反馈或修复学生作业错误，适用于高校大课堂教学和 MOOCs（Massive Open Online Courses）[3]平台。传统的辅助工具以在线评判系统[1]（Online Judge）[4-8]为代表，通常采取黑盒测试[9]，即事先准备好一些测试数据，学生在系统中编写代码并运行，当学生代码无法通过测试用例时，相应的系统反馈会提供给学生。在线评判系统的简单反馈需要学生花费大量时间定位错误原因。此外，有些系统会提供无法通过的测试用例信息来尝试引导学生发现错误原因[10]。这种做法可能会使学生“过拟合”，即面向测试用例编程，违背了编程教学的初衷。

因此，为了提高教学效率，许多方法致力于研究如何自动分析学生代码的错误或自动帮助学生修复错误代码[10-22]。大多数的方法都是基于“错误代码和正确版本仅在错误位置存在差异”的假设，而忽略正确位置也可能存在差异[23]。实际应用中，学生代码无论在正确位置或错误位置，在句法、语法和语义上与正确版本都可能存在差异，这些方法无法区分这些差异，Apex[23]通过动态分析和

[1]https://en.wikipedia.org/wiki/Competitive_programming

符号分析来区分语义差异和错误差异，然而由于学生代码的具体实现差异，序列对齐易于出错（3.1节），从而连锁地导致后续对齐出错。基于上述观察，现有技术可能产生低质量的反馈，是因为它们无法从正确版本中准确地定位出错误代码中的相关错误片段。

本文提出了一种基于概率推断[24]的方法来解决确定化序列对齐所带来的各种问题。在现有工作的基础上，本文充分结合了程序分析和概率推断技术，利用代码的动态、静态和符号特征，构建因子图模型（Factor Graph）[25-26]，并通过信念传播（Belief Propagation）[27]解算获得对齐结果，基于对齐结果构建动态切片对齐图，在此基础上利用正确提交集为错误学生代码生成准确的反馈文档，提高教师教学效率，帮助学生理解错误原因，提升编程能力。

1.2 国内外研究现状

国内外针对编程辅助教学工具的研究主要分为三个方向，分别是错误分析、错误修复和自动评分。

1.2.1 自动错误分析技术

自动错误分析技术通过指出学生提交代码中的错误位置来引导或协助学生解决算法问题，是最早被提出用于分析学生代码的方法，通常分为三类，即程序分析类、测试驱动类和数据驱动类。

第一类方法为程序分析类，需要一个或多个正确范例为参考，通过动态分析、静态分析或符号分析等方法对错误学生代码和正确教师版本比较分析。LAURA[11]将代码转换为图，然后使用启发式策略将学生代码图与正确代码图比较，如果比较结果不同，则根据不同位置进行错误诊断；Banerjee 等人[28]提出将错误代码和正确版本分别进行切片并获取符号信息，然后分析指定后条件的最弱前条件，寻找两个版本之间不一致的约束条件，从而生成错误报告；Singh 等人[13]提出了一种基于语言模型的技术，将正确代码翻译为特定语言，然后生成错误模型，错误模型包括学生代码中错误的潜在更正，错误分析则是通过将学生代码与错误模型进行约束求解生成；Lahiri 等人[29]将等效性检查[30]扩展为寻找可以部分修复错误的单值替换，被替换的单值被报告为根源错误；Apex[23]是

一种基于对齐的自动错误分析方法，它通过动态分析和符号分析[31]来区分语义差异和错误差异，即将错误代码和正确版本中正确实现的部分对齐，无法对齐的部分视为错误相关。为了解释错误，Apex 将错误代码和正确版本的动态跟踪信息对齐[32]来进行比较。在 Apex 中，初始序列对齐总是被认为是正确的，并使用它们作为基线来查找更多对齐。然而，由于学生代码的具体实现差异，序列对齐易于出错（3.1节），当序列对齐结果错误时，会连锁地导致后续对齐出错；Marin 等人[33]通过拓展程序依赖图[34]对学生代码建模，同时将带有反馈信息的模式编码为子图模式，然后通过子图搜索技术[35]在拓展程序依赖图上进行模式匹配，被匹配的子图和相应的反馈被返回给学生；TraceDiff[36]从工程实现的角度介绍了将错误代码和正确版本之间的执行路径比较分析，并通过分析分歧点给出错误分析意见。

第二类方法为测试驱动类，测试驱动的自动错误分析技术需要提供测试用例集。OJ 系统为常见的测试驱动的错误分析平台，其不足在于仅能提示正确与否，不具备引导学生分析错误的能力；为了弥补不足，Pex4Fun[10]是一种以网站为前端的反馈生成技术，该方法通过 Pex[12]生成测试用例，当提交的代码错误时，Pex4Fun 通过分析测试用例下的动态运行信息，将未通过和已通过的测试用例返回给学生，作为对错误给出解释；除了 Pex 以外，Pathgrind[37]等技术也常被用做测试驱动类方法的测试用例生成组件。

第三类是数据驱动类，这类方法应用了数据挖掘[38]、知识图谱[39]、机器学习[40]等技术。Rolim 等人[41]提出分别通过 MistakeBrowser 来挖掘并复用过去学生修复代码的经验和 FixPropagator 来学习教学者修复学生代码的经验；Piech 等人[42]提出基于神经网络的方法将代码的前置和后置条件嵌入特征空间中，然后为教学者提供示例程序的子集，教学者根据程序特征编写反馈，最后将反馈在同一特征空间中传播；CoderAssist[43]提出通过先聚类[44]后反馈的方式，充分利用 MOOCs 具有大规模提交的特点，首先通过静态模型将学生提交按算法进行聚类，然后教学者在每个聚类中选取或自行添加范例代码，最后通过等效性检查将错误版本中不等效的部分替换为正确版本，替换意见即为错误分析反馈。

程序分析类的错误分析技术利用正确版本对错误代码进行分析，他们的可行性建立在“正确版本与错误代码极其相似，仅在错误位置存在差异”的假设，或依赖于教学者手动编写正确版本，不能显著减少教学压力；测试驱动类方法

仅能从是否通过测试用例的角度出发给出错误分析，一方面不利于学生理解错误原因，另一方面会导致学生面向测试用例编程；数据驱动类方法利用其他代码的经验分析错误代码，对数据集的要求高。本文方法结合程序分析和数据驱动的优点，利用但不依赖于数据集，对测试生成技术具有可拓展性。

1.2.2 自动错误修复技术

自动错误修复技术（Automated Program Repair，APR）旨在帮助学生自动修改代码中的错误，通常分为三类，即测试驱动类、数据驱动类和深度学习类。

第一类为测试驱动类。Yi 等人[18]提出利用现成的软件自动修复工具来生成学生代码的部分修复，用户调研发现学生无法很好利用修复意见，而教学者能够从中提高效率；FixML[45]通过正确代码、测试用例来修复错误代码，首先通过运行测试用例来生成与错误代码相关子表达式，然后根据是否涉及正确执行来选择最相关的子表达式，最后通过类型生成技术[46-48]和符号执行推导潜在修复，实验表明该方法能够达到 43% 的修复率，存在一定局限性；为了解决 FixML 关于测试用例可靠性的问题，Song 等人提出了一种测试用例生成技术 TestML[49]，该方法结合枚举搜索[50-51]和符号验证技术，通过枚举搜索寻找能够触发错误代码和正确版本行为差异的测试用例，并结合符号执行和 SMT 求解器得到验证条件来生成具体数值（整数、字符串变量等）。

第二类为数据驱动类，这类方法利用正确代码的经验来修复错误学生代码，这种经验包括正确的数据流、控制流、程序结构、修复方式等。Refazer[52]是一种通过示例代码来生成转换的归纳编程（Inductive Programming，IP）[53]技术，Refazer 基于 PROSE[54]归纳编程框架，并定义了一种领域特定语言（Domain Specific Language，DSL），将程序转换定义为基于抽象语法树（Abstract Syntax Tree，AST）的重写规则，然后将重写规则应用于待修复的学生提交，最后根据排序结果生成修复；Sarfgen[15]和 Clara[16]都需要对正确代码库和错误代码库进行聚类，其中 Sarfgen 通过将代码向量化的方式进行对齐和修复错误，Clara 比较的是控制结构和变量顺序；CAFE[55]面向的是函数型语言 Ocaml，以方法级进行数据搜索，从而间接提升了错误代码的修复率，然而 Ocaml 并不是常规编程教学所用语言，相对非函数型语言的提升则有限；*Refactory*[56]在对齐部分与 Clara 相似，不同之处在于 *Refactory* 通过重构规则对正确代码进行重构，扩充

正确代码的数量，提高匹配率；Marin 等人[57]提出了一种基于静态分析的学生代码对齐方法，提取学生代码中的语义信息和逻辑信息来构成相似度评分，然后根据相似度决定是否匹配，不匹配的实例则被用于生成反馈。

第三类为深度学习类。sk_p[17]是基于神经网络的自动修复技术，从相似的修复分布中生成错误代码的候选程序，然后利用给定的测试套件来检验正确性；DeepFix[58]通过加入注意力机制的多层序列到序列（Sequence To Sequence）[59]模型和循环神经网络（Recurrent Neural Network, RNN）[60]来预测学生代码中的错误位置并生成修复意见，该工具针对的是语法错误，完全修复率达到 27%；Bhatia 等人[14]提出使用循环神经网络来学习正确代码的经验，通过编辑语句的方式修复错误代码；TRACER[61]提出了一种结合程序语言理论和深度学习的方法来修复学生代码中的编译错误，该方法通过编译器的报错信息定位错误范围，然后使用循环神经网络来预测给定错误位置文本的抽象修复，最后通过 Edlib[62]从抽象推荐中根据编辑距离选取具体代码，从而修复编译错误。

自动错误修复技术依赖于大量的正确代码数据集，即使在数据集充足的情况下，仍然无法达到理想的修复率，在现实场景下的有效性低，大多数技术在不具备大量正确版本的情况中无法投入使用；即使能够帮助学生修复错误代码，积极向上的学生能够从中提升学习效率，懒惰的学生只会机械地使用工具而不理解错误原因。本文方法不考虑自动错误修复，仅利用正确版本提供修复意见。

1.2.3 自动代码评分技术

学术界研究自动评分信息的动机在于提高教学者的工作效率，侧重于评估提交代码的正确性，给予评分和判断提交代码是否与指定算法相符。

Drummond 等人[19]提出了一系列的用于计算提交代码之间的语义和句法距离的方法，寻找与已评分提交代码相近的其他提交代码，为这些代码赋予同样的分数，已达到减少教学者需评分代码数量的目的；对于当前提交错误，AutoGrader[21]分析其动态执行路径，并与正确版本的动态执行路径相比较，如果两者之间发生路径的分歧，则判定当前提交代码为错误；Gulwani 等人[20]提出了一种基于模板的动态分析方法，教学者需要手动编写特定算法的框架，该方法通过动态分析判断是否给定提交代码符合教学者给定的算法框架。Paprika[63]提出通过静态分析结合动态分析的方式进行自动评分，Paprika 的实验证明，即使在

弱输出的学生代码中，它也能产生正确的分数和准确的反馈；Zeus[64]通过识别学生提交代码之间的算法等效性来提高人工评分效率。

自动代码评分技术旨在提高教学者的工作效率，其评分指标具有主观能动性，往往无法达到绝对公平。本文方法在自动评分方面具有可拓展性，即根据错误代码和正确版本的相似度（4.3节）对错误代码进行评分。

1.3 本文主要工作

本文的主要研究工作是面向 C 语言算法题的学生提交代码，结合程序分析技术和概率推断技术，自动生成错误学生代码的反馈文档：

1. 提出了一种基于信念传播的自动对齐算法并实现了工具 BpAlign。通过程序分析技术提取错误代码和正确版本的跟踪信息，基于符号分析和序列对齐算法将跟踪信息对齐并赋予初始概率。随后将代码信息建模为因子图，赋予先验概率，通过信念传播算法解算后获得后验概率。通过最终对齐获得对齐结果。

2. 提出了一种基于信念传播的反馈生成算法并实现了工具 Bpex。通过聚类和提取算法，为错误代码选择相似的正确版本。通过错误代码和正确版本的输出实例寻找关键实例并分别构建切片，根据基于信念传播的对齐算法结果合并为动态切片对齐图。随后根据启发式规则生成反馈文档。

3. 从 *CODECHEF* 上收集数据集并进行人工标注和实验评价。在对齐方面，与相关工具 Apex，从搜索范围变体、应用不同概率推断规则和概率值变体进行比较，实验结果显示 BpAlign 在对齐的精确度上相对 Apex 有显著提升，搜索范围能够兼顾时间和精确度，并且具备有效的推断规则；在反馈生成方面，与相关工具 Clara 比较，实验结果显示 Bpex 具有高反馈生成率，相比现有方法能更好地辅助学生理解错误原因。

1.4 论文结构

全文分为六章，如图 1.1所示，主要结构安排如下：

第一章为引言部分，对本文的研究背景与研究意义进行概述，接着介绍国内外相关研究现状，之后引出本文的主要研究工作，最后介绍论文的结构。

第二章对本文使用的相关技术进行概述，主要对本文研究内容相关的概念

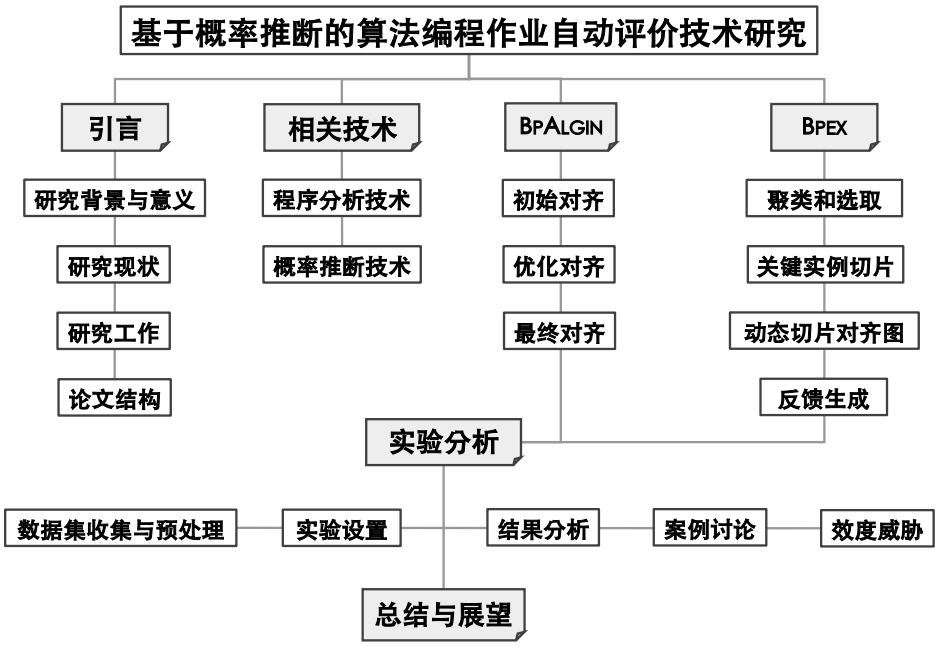


图 1.1 论文框架图

和技术以及用到的工具和评估指标进行介绍。

第三章提出一种基于信念传播的自动对齐技术，并实现了工具 BpAlign。首先介绍研究动机，随后对方法进行整体概述。接着介绍对齐算法的详细设计。

第四章提出一种基于信念传播的反馈生成技术，并实现了工具 Bpex。首先介绍研究动机，随后对方法进行整体概述。接着介绍方法的详细设计，包括聚类与选取、关键实例切片、动态切片对齐图和反馈生成。

第五章分别评估了工具 BpAlign 和 Bpex。对于 BpAlign，评估准确性、搜索范围的影响以及概率推断规则的有效性；对于 Bpex，评估聚类算法的有效性、反馈生成率以及生成效果。

第六章是本文的总结与展望，对本文基于概率推断的算法编程作业自动评价技术研究进行总结，同时也提出本文工作存在的不足，并给出未来的工作方向。

第二章　相关技术概述

本章首先概述相关技术，对本文中用到的程序分析技术进行介绍，包括程序依赖、抽象语法树、程序跟踪和符号执行；之后说明了概率推断技术的基本概念，介绍了概率图模型的相关知识和因子图的结构，随后介绍了信念传播算法的主要原理；接着对本文所使用的程序分析工具和概率推断工具进行简要介绍；最后介绍本文中评估代码对齐准确度的三个常用指标。

2.1　程序分析技术

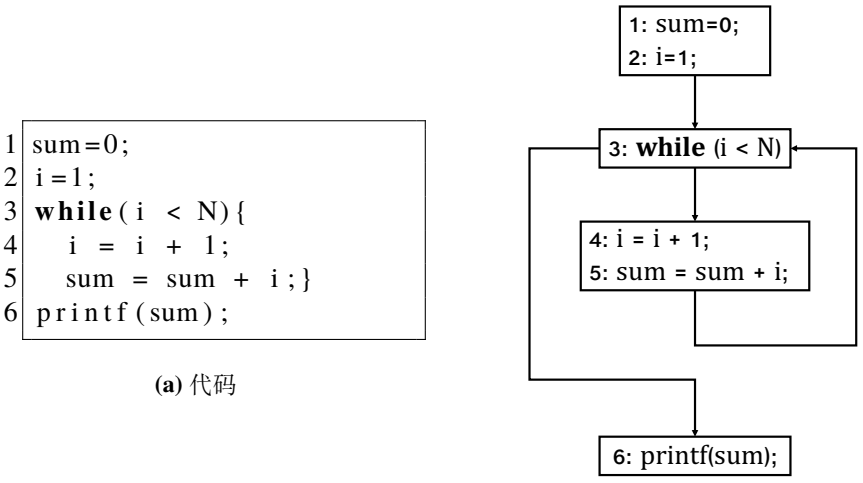
程序分析技术通常分为动态程序分析、静态程序分析和符号分析。动态程序分析通过执行软件来解决软件依赖相关的问题，静态程序分析和符号分析不执行程序。本文的分析对象为解决特定算法问题的代码，通常为学生提交或教师范本。下文将介绍本文使用到的若干程序分析技术。

2.1.1　程序依赖

在程序分析中有两种常用的依赖关系，分别是控制依赖（Control Dependence）和数据依赖（Data Dependence）。程序依赖图（Program Dependence Graph，PDG）[34]将控制依赖关系和数据依赖关系建模为数学中图的表示[65]。通过程序依赖图能够清晰地表达出程序中语句、变量之间的逻辑结构关系，通过语句之间的依赖关系能够获取切片，在 BpAlign 和 Bpex 建模中能够提供每个变量、语句实例的数据依赖以及控制依赖关系，起到重要的信息支撑作用。

(1) 控制依赖

控制依赖用于表示控制流引起的程序实体间的关系，控制流（Control Flow）指程序中操作执行的顺序。控制流图（Control-Flow Graph，CFG）[66]利用数学中图的表示方式，标示计算机程序执行过程中所经过的所有路径，其中控制流图的每个顶点对应一段没有分支的代码，即基本块，有向边表示分支，即基本块之间的流向。设 n_1，n_2 为控制流图中的两个节点，若 n_2 能否被执行取决于 n_1 的



(a) 代码

(b) 控制流图

图 2.1　控制流图示例

执行状态，则称 n_2 控制依赖于 n_1。

图 2.1a是一段求 2 到 N 之和的代码片段，图 2.1b为该段代码对应的控制流图示例，其中语句 3 控制依赖于语句 2，而不是语句 1。语句 2 控制依赖于语句 1，所以语句 3 传递依赖于语句 1，依赖距离为 2；语句 4 和语句 6 能否执行取决于语句 3 的执行状态，因此语句 4 和语句 6 控制依赖于语句 3，同时语句 3 能否执行取决于语句 5，因此语句 3 控制依赖于语句 5。

(2) 数据依赖

数据依赖用于表示由于数据的定义和使用形成的实体之间的关系。设 n_1，n_2 为控制流图中的节点，v 其中的一个变量。若满足下列三个条件，则称 n_1 关于变量 v 数据依赖于 n_2:

1. n_2 对变量 v 进行了定义，即 $v \in def[n_2]$;
2. n_1 执行时使用了 v 的值，即 $v \in use[n_1]$;
3. n_2 与 n_1 间存在一条可执行路径，并且路径上没有语句对 v 进行重新定义。

如图 2.2中，语句 5 关于变量 *sum* 数据依赖于语句 1，因为 1 对变量 *sum* 进行了定义，5 执行时使用了 *sum* 的值；语句 6 数据依赖于 5 和 1，其中 5 在使用 1 的定义同时也定义了 *sum*；语句 6 传递依赖于 *i=1*，依赖距离为 3。

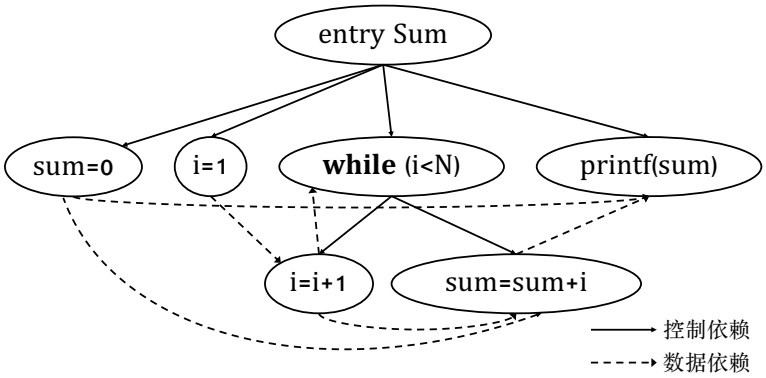


图 2.2　程序依赖图示例

(3) 程序依赖图

程序依赖图由控制依赖关系和数据依赖关系组成，其中控制依赖关系包含了程序中的控制依赖；数据依赖关系是一个程序中语句之间数据依赖的集合。程序依赖图一般包含过程内依赖图（Procedure Dependence Graph，PrDG）和过程间依赖图 (System Dependence Graph, SDG)，在本文中主要使用过程内依赖图。

程序依赖图定义为 $G = (V, E)$，是一个有向图。其中 V 是节点集合，表示过程中的语句或谓词，E 是边集合，表示节点间的数据依赖或控制依赖关系。

图 2.2是图 2.1a中代码片段的程序依赖图，其中实线表示控制依赖关系，虚线表示数据依赖关系，过程内依赖图还包括一个入口节点（entry）用于表示过程的入口。语句 5 关于变量 *sum* 数据依赖于语句 1，因为 1 对变量 *sum* 进行了定义，5 执行时使用了 *sum* 的值；语句 4 直接控制依赖于语句 3，间接控制依赖于 *entry*，依赖距离为 2。

2.1.2　抽象语法树

抽象语法树[1]（Abstract Syntax Tree，AST）是源代码语法结构的一种树状的抽象表示，树上的每个节点都表示源代码中的一种结构。抽象语法树包含了代码中层次分明的结构信息，能够帮助我们了解语句的结构特征。本文利用抽象语法树提取代码的结构特征，以进行后续的概率建模。

图 2.3为图 2.1a转化为的抽象语法树。其中根节点为方法定义（Method Declaration，MD），其四个子节点分别定义了两个参数声明（Variable Declaration，

[1]https://en.wikipedia.org/wiki/Abstract_syntax_tree

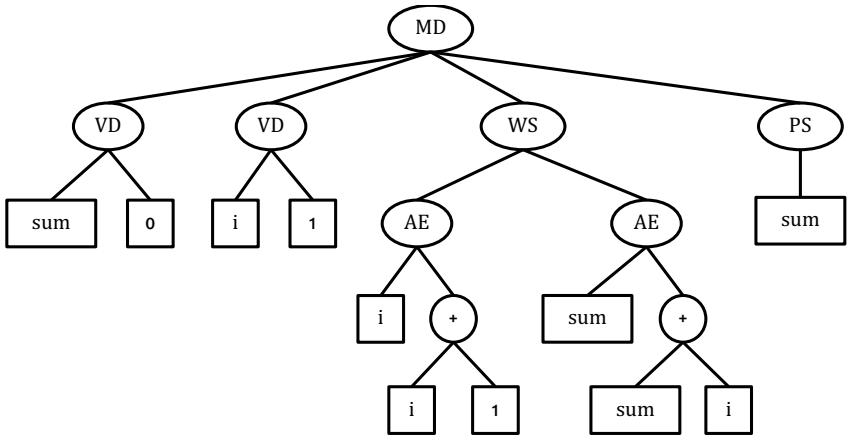


图 2.3　抽象语法树示例

VD)，对应代码 1-2 行；循环语句（While Statement，WS)，对应第 3 行；输出语句（Print Statement，PS)，对应第 6 行；WS 中两个子节点为赋值表达式（Assignment Expression，AE)，对应 4-5 行。

2.1.3　程序跟踪

程序插桩[67]是一种基本的动态程序分析方法，通过向代码中添加一些语句实现对程序的执行、变量的变化等情况的检查来获取程序的控制流和数据流信息。程序插桩常被用于度量软件性能、诊断软件错误、获取跟踪信息。本文使用程序插桩获取学生代码的跟踪信息，即程序跟踪。跟踪是一个忠实记录程序执行细节信息的过程，包括控制流跟踪、依赖跟踪、值跟踪等。

图 2.4给出了一个包含了控制流跟踪、依赖跟踪、值跟踪的程序跟踪示例。其中图 2.4a为一段计算偶数斐波拉契数字之和的代码，给定输入参数 *N0=1*，*N1=2*，*N=20*，表 2.4b为动态插桩后获得的部分跟踪信息，其中序号用于标识每个语句实例，E（Entry）表示代码的起点。在这个跟踪信息中包含左侧代码动态执行的信息，包括执行语句，如 *n2=n0+n1*；控制依赖关系，如序号 6 的语句直接控制依赖于语句 5，并且传递控制依赖于语句 2 和 E，依赖距离分别为 2 和 3；数据依赖关系，如序号 8 的语句直接数据依赖于语句 7，并且传递依赖于语句 1 和 4；符号表达式见第2.1.4节；运行值，如当 *n2=3* 而 *n=20* 时，*n2>n* 的值为 False。

由此可见，基于程序插桩的程序跟踪技术能够牢牢抓住程序运行的轨迹，对代码执行有更清晰的认知。获取程序跟踪通常作为动态程序分析最基础的工作。

```
1  int sum_of_even_fibonacci(int N0, int N1, int N){
2    int n0 = N0,n1 = N1,n2 = 0,n = N;
3    int sum = 2;
4    for(;;){
5      n2 = n0 + n1;
6      if( n2 > n )
7        break;
8      if ( n2 % 2 == 0 )
9        sum += n2;
10     n0 = n1; n1 = n2; }
11   return sum; }
```

(a) 代码

序号	语句	控制依赖	数据依赖	符号表达式	值
1	n2=n0+n1	E	-	N0+N1	3
2	n2>n	E	1	-	False
3	n2%2==0	E-2	1	-	False
4	n2=n0+n1	E-2	1	N1+N0+N1	5
5	n2>n	E-2	1-4	-	False
6	n2%2==0	E-2-5	1-4	-	False
7	n2=n0+n1	E-2-5	1-4	3*N1+2*N0	8
8	n2>n	E-2-5	1-4-7	-	False
…	…	…	...	…	…
9	n2%2==0	E-2-5-8	1-4-7	-	True
10	sum+=n2	E-2-5-9	1-4-7	3*N1+2*N0+2	10

(b) 跟踪

图 2.4 跟踪信息示例

2.1.4 符号执行

符号执行（Symbolic Execution）[31]是一种程序分析技术，本文利用符号执行获取学生代码语句中的符号表达式，来模糊语义差异，辅助判断是否对齐。

使用符号执行分析一个程序的特定代码区域时，该程序会使用符号值作为输入，而非一般执行程序时使用的具体值，程序变量的值表示为符号表达式，程序计算的输出表达为输入符号值的函数。在达到目标代码时，分析器可以得到相应的路径约束，然后通过约束求解器来得到可以触发目标代码的具体值。

通常约束会编码为 SAT（Satisfiablility Problem）/ SMT（Satisfiablity Modulo Theories）[68]符号约束。SAT 用于解决命题逻辑公式问题，指由布尔变量集合所构成的布尔函数，是否存在这些变量的一种分布，使得该函数的取值为 1，许多实际应用无法直接转换为 SAT 问题求解；SMT 指可满足性模理论，将 SAT 求解从只能解决命题逻辑公式扩展为可以解决一阶逻辑所表达的公式。

符号执行的基本思想是首先将程序建模，如控制流图；然后遍历程序中的每条路径，对条件和一条路径上的符号约束对应的符号值建模，获得符号表达式。相比于运行值，符号表达式能够更精确地表达语句的语义信息。

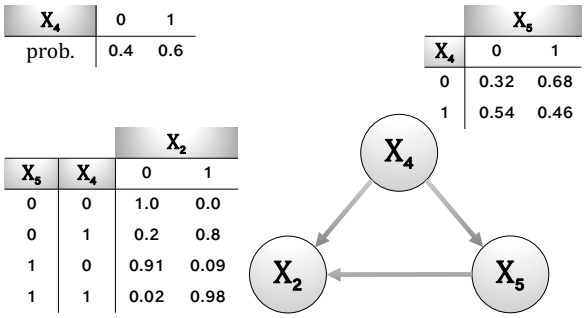


图 2.5 贝叶斯网络结构图示例

图 2.4b中符号表达式一列展示了对图 2.4a中示例程序进行符号执行所得结果。为了方便表达，此处用 *N0* 和 *N1* 表示输入参数，在每次计算中获得符号表达式，如 *n2=n0+n1* 的符号表达式为 *N0+N1*，其中 *n0* 用 *N0* 表示，*n1* 用 *N1* 表示。此后的第一个涉及 *n2* 的计算，用当前 *n2* 的符号表达式表示，以此类推。最后在语句 10 中获得 *sum* 的符号表达式。

2.2 概率推断技术

概率推断技术广泛应用于机器学习，包括概率图模型[69]的构建以及对概率图模型定义的联合概率分布进行推断。本文将程序的关键信息建模为因子图（Factor Graph）[25-26]，然后使用信念传播（Belief Propagation）[27]进行概率推断。

2.2.1 概率图模型

在概率论及其应用中, 因子图是一种无向概率图模型，在贝叶斯推理中得到广泛应用。本节从贝叶斯网络（Bayesian network）[70]出发介绍因子图。

(1) 贝叶斯网络

贝叶斯网络是经典的有向概率图模型，其拓扑结构为有向无环图（Directed Acyclic Graph，DAG）。DAG 定义为 $G = (I, E)$，其中 I 代表图形中所有的节点的集合；E 代表有向连接线段的集合；$X = (X_i), i \in I$ 为其 DAG 中的某一节点 i 所代表的随机变量，若节点 X 的联合概率可以表示成：

$$p(X) = \prod_{i \in I} p(X_i | X_{pa(i)}) \tag{2.1}$$

则称 X 为相对于 DAG 的贝叶斯网络。其中，$pa(i)$ 表示节点 i 之因（parents），被 i 指向的节点表示节点 i 之果（children），以 $c(i)$ 表示，两节点相连会产生一个条件概率值。图 2.5是一种典型的贝叶斯网络结构图，根据定义可知，$P(2) = \{X_4, X_5\}$，$C(2) = \varnothing$，$P(4) = \varnothing$，$C(4) = \{X_2, X_5\}$，$P(5) = \{X_4\}$，以及 $C(5) = \{X_2\}$。

对任意的随机变量，其联合分布可由各自的局部条件概率分布相乘而得出（对每个相对于 X_i 的“因”变量 X_j 而言）：

$$P(X_1 = x_1, ..., X_n = x_n) = \prod_{i=1}^{n} P(X_i = x_i | X_j = x_j) \tag{2.2}$$

(2) 因子图

因子图是对函数进行因式分解为多个局部函数的乘积而得到的一种概率图，其联合质量函数写成：

$$p(\mathbb{X}) = \prod_{i=1}^{n} f_i(\mathbb{X}_i) \quad , \mathbb{X}_i \subset \mathbb{X} \tag{2.3}$$

因子图内含两种节点，分别为随机变量和函数变量，边线表示它们之间的函数关系，函数变量和对应的随机变量的关系就体现在因子图上。在概率图中，求某个变量的边缘分布的解决方式之一是将贝叶斯网络转换成因子图，然后用信念传播算法求解。如图 2.5，根据各个变量对应的关系可得：

$$P(X_2, X_4, X_5) = P(X_4)P(X_5|X_4)P(X_2|X_4, X_5) \tag{2.4}$$

其对应的因子式表示为：

$$P(\mathbb{X}) = f_A(X_4)f_B(X_4, X_5)f_C(X_2, X_4, X_5) \tag{2.5}$$

图 2.6为图 2.5转换为因子图的结果，图中 f_A，f_B 和 f_C 由图 2.5的条件概率表转化而来，并与公式2.5对应，为函数变量；X_4，X_5 和 X_2 即随机变量，对应于贝叶斯网络中的随机变量。

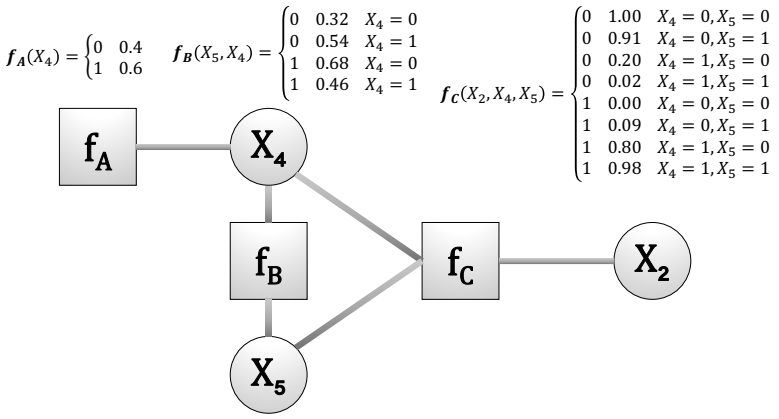


图 2.6　因子图示例

2.2.2　信念传播

信念传播算法又称消息传播算法（message passing），用于精确推断贝叶斯网络中任意节点的边缘分布或观测条件分布，后续推广到无向图上，由因子图表示统一成更泛化的和积算法（sum-product algorithm）。

在2.2.1节中，因子图中包含两种节点，分别是随机变量（v）和函数变量（a）。根据不同的节点类型，因子图有两种消息传递方向，分别是 $m_{v\to a}$ 和 $m_{a\to v}$：

$$m_{v\to a}(X_v)=\prod_{i\in N(v)\backslash a} m_{i\to v}(X_v) \tag{2.6}$$

公式2.6指随机变量 v 传递给函数变量a 的消息，$N(v)$ 指 v 的邻接函数变量；

$$m_{a\to v}(X_v)=\sum_{X'_a:X'_v=X_v} f_a(X'_a)\prod_{i\in N(a)\backslash v} m_{i\to a}(X'_i) \tag{2.7}$$

公式2.7指函数变量a 传递给随机变量 v 的消息，$N(a)$ 指函数变量a 的邻接随机变量，该表达式定义为函数变量与来自所有其他随机变量的消息的乘积，除与 v 相关的变量外，所有变量都被边缘化（marginalized）。

公式2.6-2.7中，信念传播将完全边缘化问题简化为简单项的乘积之和。

在运行循环信念传播（loopy belief propagation）时，每条消息都将从相邻消息的先前值迭代更新，直到没有消息再改变为止，则视为收敛。当收敛时，每个

随机变量的估计边缘概率分布与来自相邻函数变量的所有消息的乘积成正比：

$$P(X_v) \propto \prod_{a \in N(v)} m_{a \to v}(X_v) \tag{2.8}$$

本文方法中将随机变量的边缘概率分布做归一化处理，从而获得后验概率。

2.3 相关工具介绍

本节介绍在论文方法实现过程中使用的工具，主要包括编译器框架 LLVM[2]（Low Level Virtual Machine）[71]、可满足性模理论 (SMT) 求解器 Z3、基于 Python 的概率推断工具，包括 pgmpy[3][72]和 py-factorgraph[4]。

LLVM 是构架编译器的框架系统，以 C++ 编写而成，用于优化以任意程序语言编写的程序，对开发者保持开放并兼容已有脚本。和传统的编译器架构类似，LLVM 分为三个阶段，即前端（Frontend）、优化器（Optimizer）和后端（Backend），不同的前端后端使用统一的中间代码（Intermediate Representation，IR)。本文面向的是 C 语言程序代码，因此选择 Clang 作为前端，在 IR 阶段对代码进行优化处理，将测试代码转化为语言无关的中间代码，以获取程序跟踪信息。

Z3[73]，也称为 Z3 Theorem Prover，是 Microsoft 的跨平台可满足性模理论求解器，它能够检查逻辑表达式的可满足性。Z3 支持算术、位向量、扩展数组等。它的主要应用是扩展静态检查、测试用例生成和谓词抽象（predicate abstraction）。本文使用 Z3 来判断错误代码和正确版本的符号表达式是否相等。

pgmpy 是贝叶斯网络的 Python 实现，专注于模块化和可扩展性。实现了结构学习（Structure Learning）[74]、参数估计（Parameter Estimation）[75]、近似推断（Approximate inference）[76]和精确推断（Exact inference）[77]以及概率推断（Probabilistic Inference ）[24]等各种算法，缺点是时间开销大。py-factorgraph 是更为轻量化的 python 实现的概率推断工具，支持因子图的构建和（循环）信念传播算法。本文起初用 pgmpy 实现，后期为了兼顾性能转向更为轻量的 py-factorgraph。

[2]https://llvm.org/

[3]https://github.com/pgmpy/pgmpy

[4]https://github.com/mbforbes/py-factorgraph

2.4 评估指标

为了评估对齐的准确性，需要将对齐结果与标注样本进行对比。本文选取精确率 (Precision)、召回率 (Recall) 和综合评价指标 (F-Score) 来提供客观的评价指标。其中 TP（True Positives）指真正例，即成功对齐的实例对；TP（True Negatives）指真负例，即未能对齐的实例对；FP（False Positives）指假正例，即算法输出为对齐，但样本标注中不对齐的实例；FN（False Negatives）指假负例，即算法输出为不对齐，样本标注中也为不对齐的实例。

精确率为解决在被识别为正类别的样本中正类别的比例。公式定义如下：

$$Precision = \frac{TP}{TP + FP} \tag{2.9}$$

召回率为解决在正类别样本中被正确识别为正类别的比例。公式定义如下：

$$Recall = \frac{TP}{TP + FN} \tag{2.10}$$

F-Score 是一种统计量，是一种量测方法精确度常用的指标，经常用来判断算法的精确度。公式定义如下：

$$F\text{-}Score = \frac{(1 + \beta^2) \cdot Precision \times Recall}{(\beta^2 \cdot Precision) + Recall} \tag{2.11}$$

一般来说，提到 F-score 且没有特别的定义时，是指 $\beta = 1$ 时的 F-score，写作 F_1 score（$F_1 = 2 * \frac{precision * recall}{precision + recall}$），代表使用者同样注重 Precision 和 Recall 这两个指标。

2.5 本章小结

本章对全文涉及到的相关技术进行概述。首先介绍了程序分析技术的基本概念，并对本文中用到的程序分析技术进行了介绍；之后说明了概率推断技术的基本概念，介绍了概率图模型的相关知识和因子图的结构，随后介绍了信念传播算法的主要原理；接着对本文所使用的程序分析工具和概率推断工具进行简要介绍；最后介绍了本文中评估代码对齐准确度的三个常用指标。

第三章　基于信念传播的代码自动对齐

本章提出一种基于信念传播的代码自动对齐技术，并实现工具 BpAlign。首先结合错误学生代码示例阐述研究动机；随后对方法进行概述；接下来对方法中的对齐算法展开详细介绍。

3.1　研究动机

大量现有工作基于错误学生代码和正确教师版本仅在错误位置存在差异的假设，而忽略正确位置也可能存在差异（语义差异）[23]。Apex 是一种基于对齐的自动错误分析方法，即将错误学生代码和正确教师版本中正确实现的部分对齐，无法对齐的部分视为错误相关，它通过动态分析和符号分析来解决对存在语义差异语句的对齐，然而 Apex 在对齐的准确度上存在缺陷。

图 3.1展示了来自 *CODECHEF*[1]（一个面向初学者的在线编码练习平台）中的两段解决 *DISHOWN*[2]问题的学生代码，其中图 3.1a是能编译通过但无法通过特定测试用例的错误代码，图 3.1b是正确代码。

问题描述： *DISHOWN* 问题模拟厨师之间的竞赛。输入数据包含厨师的数量和每个厨师最拿手菜肴的得分；然后输入 Q 组查询，查询有两种类型：

1. 0 x y：拥有菜肴编号 x 的厨师和菜肴编号 y 的厨师之间进行一场对决。如果是同一个人，则输出 “`invalid query`”，否则拥有得分高菜肴的厨师获胜，并获得败者所有的菜肴。
2. 1 x：询问并输出当前拥有菜肴编号 x 的厨师编号。

图 3.2展示了这两段代码执行的输入、输出和状态。输入 1 初始化了 3 个厨师的菜肴得分，起初第 i 个菜肴 Di 属于第 i 个厨师 Ci（从 1 编号），并且得分为 i；输入 2 表示 D1 和 D2 的对决，由于 D2 得分高于 D1，C2 胜出并且获得 D1；输入 3 查询了 D1 的拥有者，两段代码的输出都为 “`2`”；输入 4 表示 D1 和 D2 的对决，由于 C2 同时拥有了 D1 和 D2，故是同一个人，输出 “`invalid query`”；

[1] https://www.codechef.com

[2] https://www.codechef.com/problems/DISHOWN

```
1  for (i = 1 to N)
2    read(D[i].score);
3    D[i].chef = i;
4
5  for (i = 1 to Q)
6    read(query);
7    if (query == C)
8      read(x, y);
9      if D[x].chef == D[y].chef
10       print("invalid query");
11     elif D[x].score > D[y].score
12       D[y].chef=D[x].chef; // BUG
13     else
14       D[x].chef=D[y].chef; // BUG
15   else if (query == Q)
16     read(x);
17     print(D[x].chef)
```

(a) 错误代码

```
1  for (i = 1 to N)
2    chef[i] = i;
3
4  for (i = 1 to N)
5    read(score[i]);
6
7  for (i = 1 to Q)
8    read(query);
9    if (query == C)
10     read(x, y);
11     a = FindChef(x); // CORRECT
12     b = FindChef(y); // CORRECT
13     if a == b
14       print("invalid query");
15     elif score[a] > score[b]
16       chef[b] = a;
17     else
18       chef[a] = b;
19   else if (query == Q)
20     read(x);
21     print(FindChef(x));
22
23 function FindChef(c)
24   if c != chef[c]
25     chef[c] = FindChef(chef[c]);
26   return chef[c];
```

(b) 正确代码

图 3.1　*DISHOWN* 的代码示例

输入 5 比较了 D1 和 D3，D1 的所有者为 C2，得分为 2，D3 的所有者为 C3，得分为 3，因此，C3 胜出并赢得所有 3 个菜肴。

预期执行：在输入 6 中，正确代码采用 *FindChef()*（11 和 12 行）来寻找 D1 的拥有者（C2），因为 C3 拥有更高得分 D3，故 C3 胜出并获得 D2（此时为 C2 所有），而不仅仅是将 D1 给予 C3。通过方法 *FindChef()*，此后的查询中，正确代码能够获知 D2 的拥有者为 C3。

错误执行：相对而言，错误代码只将 D1 的拥有者更新为 C3（14 行），而错

#	输入	输出	状态
1	1 2 3		初始化得分
2	C 1 2		C2 胜出，并且 C2 获得 D1
3	Q 1	“2”	C2 拥有 D1
4	C 1 2	“Invalid query”	C2 拥有 D1 和 D2
5	C 1 3		C3 胜出，并且 C3 获得 D1
6	C 2 3		C2 拥有 D2，C3 拥有 D3

(a) 错误执行

#	输入	输出	状态
1	1 2 3		初始化得分
2	C 1 2		C2 胜出，并且 C2 获得 D1
3	Q 1	“2”	C2 拥有 D1
4	C 1 2	“Invalid query”	C2 拥有 D1 和 D2
5	C 1 3		C3 胜出，并且 C3 获得 D1 和 D2
6	C 2 3	“Invalid query”	C3 拥有 D2 和 D3

(b) 预期执行

图 3.2　错误执行和预期执行

6_1	read(query)	**query=C**	8_1	read(query)	**query=C**
7_1	if query==C	**BR=T**	9_1	if query==C	**BR=T**
8_1	read(x,y)	**x=1,y=2**	10_1	read(x,y)	**x=1,y=2**
9_1	if D[x].chef==D[y].chef	D[x].chef=1,D[y].chef=2,**BR=F**	11_1	a=FindChef(x)	**a=1**
11_1	else if D[x].score>D[y].score	D[x].score=1,D[y].score=2,**BR=F**	12_1	b=FindChef(y)	**b=2**
14_1	D[x].chef=D[y].chef	**D[x].chef=2**	13_1	if a==b	**BR=F**
			15_1	if score[a]>score[b]	score[a]=1,score[b]=2,**BR=F**
			18_1	chef[a]=b	**chef[a]=2**

图 3.3　Apex 的对齐

误地忽略 D2 的更新，12 行同理。输入 6 比较了 D2 和 D3，由于在预期执行中 C3 同时拥有 D2 和 D3，故应当输出“`invalid query`”，然后由于错误的拥有关系而无法输出。

Apex 的错误对齐： Apex 将错误代码和正确版本的符号表达式跟踪信息进行对齐，主要分为三个步骤。首先，Apex 为学生代码的跟踪信息符号执行并获取符号表达式，跟踪信息通过错误代码无法通过的测试输入触发；然后，Apex 通过序列对齐算法和符号表达式等价检查来构建初始对齐；最后，Apex 进一步将其他未生成符号表达式的语句实例（如判断语句）或未在序列对齐中的其他语句实例对齐，主要以已经对齐的语句实例的依赖关系为参考。

图 3.3展示了 Apex 对图 3.1的两个代码在图 3.2中输入 #2 时的对齐结果，其中每一行都是被执行的语句实例，左边三列表示错误代码的语句实例执行情况，分别表示语句 i 的第 j 次执行（如 6_1 表示语句 6 的第一次执行）、源代码文本和动态执行信息，其中加粗部分表示被用于序列对齐的值。右边三列表示正确代码的执行情况，灰色框表示 Apex 对齐的语句实例。为了简化表达，图中展示的是实际值而不是符号值。在执行过程中，两段执行分别通过 *D[y].chef* 和 *chef[y]* 来更改 *D[x].chef* 和 *chef[x]* 的值，因此，14_1^{ℓ}，12_1^{t} 和 18_1^{t} 都有具体值 2。Apex 的序列对齐算法将 14_1^{ℓ} 和 12_1^{t} 进行对齐，而 18_1^{t} 没有对齐的实例。然而这样的对齐是错误的，因为 14_1^{ℓ} 的功能是更新第 x 个菜肴的拥有者，而 12_1^{t} 的功能是寻找第 y 个菜肴的拥有者。Apex 的算法会基于序列对齐的结果进行后续对齐，错误的序列对齐结果会对后面的对齐产生灾难性的连锁反应：在示例中，Apex 无法正确地将 9_1^{ℓ} 和 13_1^{t} 对齐，因为在 Apex 的规则中会产生依赖冲突：在 14_1^{ℓ} 和 12_1^{t} 已经对齐的前提下，如果 14_1^{ℓ} 依赖于 9_1^{ℓ}，而 12_1^{t} 被 13_1^{t} 所依赖，则产生冲突。

Apex 错误对齐的根本原因是其过度依赖于判断符号表达式，无法处理对齐的不确定性，而类似上述情况下的不确定性在编程实践中很常见。因此我们提出了一种基于概率推断的方法来处理不确定性。

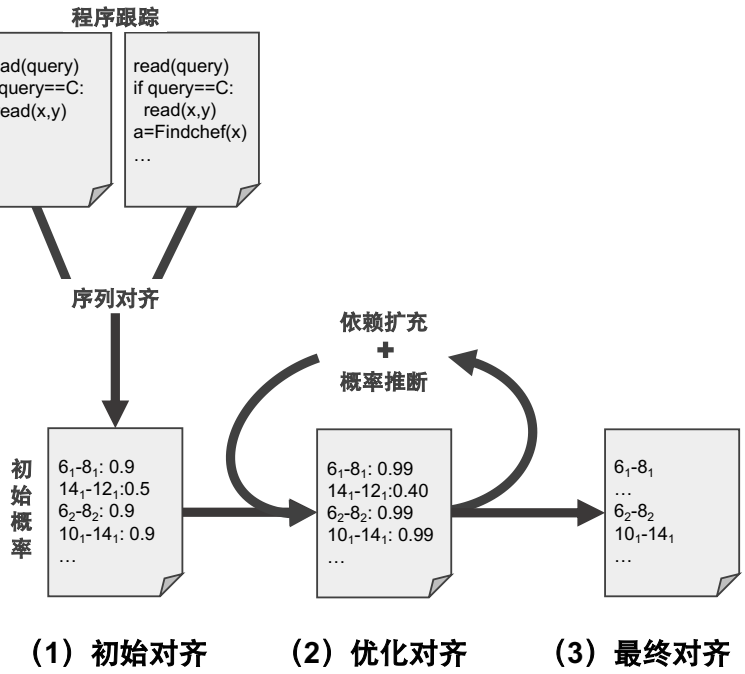


图 3.4　BpAlign 方法概览图

3.2　方法概述

我们设计实现的工具 BpAlign 的总体框架如图 3.4所示，主要包括三个阶段，即初始对齐阶段、优化对齐阶段和最终对齐阶段。

初始对齐阶段：首先使用 LLVM 收集错误代码和正确版本的程序跟踪，然后 BpAlign 通过序列对齐算法将错误代码和正确版本的程序跟踪进行对齐，序列对齐只考虑符号表达式，初始对齐为对齐结果根据其特征赋予初始概率值，考虑符号表达式和输出实例信息。

优化对齐阶段：BpAlign 通过优化对齐算法来将对齐的不确定性建模为概率约束并解算，包括依赖扩充算法和概率推断算法，优化对齐可以迭代执行 n 轮。其中依赖扩充算法将输入实例对构建为概率约束，并根据搜索距离迭代地为已建模为概率约束的实例对所依赖的其他实例对构建概率约束；概率推断算法规定了概率约束的构建方法，包括概率约束和函数变量的定义。

最终对齐阶段：同一个实例可能与多个实例之间对齐概率高于阈值，最终对齐阶段从随机变量中选取对齐概率最高的实例对并最大化对齐数量，最终对齐结果将作为反馈生成方法的输入。

3.3 初始对齐

与 Apex 相似，BpAlign 以初始序列对齐为出发点进行后续对齐；不同之处在于 Apex 在序列对齐的基础上进行迭代对齐，而我们仅以序列对齐结果作为先验概率，而非确定性的对齐结果。

算法 3.1: 序列对齐算法

```
1:  procedure sequenceAlign(E_1, E_2)
2:      for i_1 = 1 to E_1.length do
3:          for i_2 = 1 to E_2.length do
4:              Align[i_1][i_2] ← max(
5:                  Align[i_1 − 1][i_2 − 1] + score(E_1[i_1], E_2[i_2]),
6:                  Align[i_1 − 1][i_2], Align[i_1][i_2 − 1])
7:          end for
8:      end for
9:      alignments ← {}
10:     i_1 ← E_1.length
11:     i_2 ← E_2.length
12:     while i_1 ≥ 1 ∩ i_2 ≥ 1 do
13:         if Align[i_1][i_2] == Align[i_1 − 1][i_2 − 1] + score(E_1[i_1], E_2[i_2]) then
14:             alignments ← alignments ∪ (E_1[i_1], E_2[i_2])
15:             i_1 ← i_1 − 1
16:             i_2 ← i_2 − 1
17:         else if Align[i_1][i_2] == Align[i_1 − 1][i_2] then
18:             i_1 ← i_1 − 1
19:         else if Align[i_1][i_2] == Align[i_1][i_2 − 1] then
20:             i_2 ← i_2 − 1
21:         end if
22:     end while
23:     return alignments
24: end procedure
25: procedure score(ℓ_i, t_j)
26:     if symb(ℓ_i) ≡ symb(t_j) then
27:         return 1
28:     else
29:         return 0
30:     end if
31: end procedure
```

3.3.1 程序跟踪

BpAlign 使用 LLVM 获取跟踪信息，跟踪信息以语句实例为单位，语句实例包括代码文本、控制和数据依赖、运行值、操作数等信息。在本文中，具体实

例通过 i_j^{ℓ} 和 i_j^{t} 表示，分别是错误语句 i 的第 j 次执行和正确语句 i 的第 j 次执行。

本文用于对齐和生成反馈的实例包括三种类型，分别是赋值类型（Store Instance，STIns），如图 3.3中的 14_1^{ℓ}、11_1^{t}、12_1^{t} 和 18_1^{t}；判断类型（Branch Instance，BRIns），如 7_1^{ℓ}、9_1^{ℓ}、9_1^{t} 等；输出类型（Print Instance，PRINT），由输出串和操作数组成，输出串即为实际输出流中的字符串，操作数为输出时引用的变量值，如图 3.1a中第 17 行中的 D[x].chef。

在此基础上进行符号分析，以获得赋值类型语句实例的符号表达式。将以上信息结合，得到所需的程序跟踪。

3.3.2 序列对齐

算法 3.1 描述了本文所使用的尼德曼-翁施序列对齐算法[78]，算法旨在寻找两个序列中能够最大化对齐数量的匹配对，其匹配过程由评分机制所驱动，通常有匹配、不匹配和错位三种赋分方式。本文场景下，当两个实例具有等同的符号表达式时得 1 分，否则得 0 分（24-30 行）。

算法输入为两个跟踪序列 E_1 和 E_2 并初始化算法分数矩阵 $Align$，其中 i_1 表示表示 E_1 的第 i_1 个实例，i_2 同理；2-8 行计算矩阵中的每一项 $Align[i_1][i_2]$，若 $\textsc{score}(E_1[i_1], E_2[i_2])$ 为 1，则取左上角值加 1，否则按从左上、左、上的顺序取最大值；12-22 行对矩阵进行回溯，从矩阵右下角单元格开始，若 $\textsc{score}(E_1[i_1], E_2[i_2])$ 为 1，则回溯到左上角单元格，并将 $(E_1[i_1], E_2[i_2])$ 加入对齐序列 $alignments$，否则按从左上、左、上的顺序回溯到值最大的单元格，直到超出矩阵范围。$alignments$ 作为序列对齐结果返回。

3.3.3 初始概率

序列对齐的结果还无法被直接应用到优化对齐，因为概率推断需要先验概率。初始概率阶段为序列对齐中的实例对赋予初始概率，在优化对齐阶段将初始概率作为先验概率建模为概率约束。

如图 3.5所示，I_{ℓ_i,t_j} 代表当前实例对，如果 ℓ_i 和 t_j 具有等同的符号表达式或是等同的输出实例（如图 3.3中的 17_1^{ℓ} 和 21_1^{t}，他们具有相等的输出串，具体见4.4.1节），则赋予概率值 0.9，表示对齐可能性高，否则赋予概率值 0.5，表示

$$p(I_{\ell_i,t_j}) = \begin{cases} 0.9 & \text{if } symb(\ell_i) \wedge symb(t_j) \wedge symb(\ell_i) \equiv symb(t_j) \\ 0.9 & \text{if } \ell_i \text{ and } t_j \text{ are identical print statements} \\ 0.5 & \text{otherwise} \end{cases}$$

图 3.5　初始概率

无法确定。其中输出实例通过其操作数是否相等来判断是否等同，操作数为输出实例引用的变量构成的序列。

之所以认为具有等同的符号表达式的实例对对齐可能性高，是因为如果两个实例的符号表达式相同，则说明它们在语义上等价，相较于符号表达式不同的实例对更可能对齐。对于其他类型的实例对，如判断类型实例或无符号表达式实例（如循环语句中的计数语句），为它们赋予中立的概率值 0.5，表示它们可能对齐也可能不对齐。

之所以认为等同的输出类型语句实例对对齐可能性高，一方面是因为它们在语义上等价；另一方面，考虑到等同输出实例依赖的其他实例构成的实例对也很可能对齐，3.4.2节所定义的函数变量能够利用等同输出实例与其他依赖实例间的概率约束，提高其他依赖实例的对齐概率。

3.4　优化对齐

基于序列对齐算法获得初始概率后，我们对初始对齐进行优化对齐，优化对齐支持多次迭代，此后每次迭代会对上次优化对齐的结果构建概率约束，并进行依赖扩充，扩充的实例对同样构建为概率约束。

算法 3.2: 优化算法

```
1:  procedure REFINE(oldAlignments, i)
2:      factorGraph ← PROPAGATE(oldAlignments, i)
3:      bpResult ← solve(factorGraph)
4:      newAlignments ← {}
5:      for align ∈ bpResult do
6:          if align.probability > θ then
7:              newAlignments ← newAlignments ∪ {align}
8:          end if
9:      end for
10:     return newAlignments
11: end procedure
```

算法 3.2描述了优化对齐过程。输入变量 *oldAlignments* 初次迭代时为初始对

齐结果，后续迭代时为上次迭代的输出结果；输入变量 i 具有两种作用，一是控制迭代次数，从 1 为起点，到最大迭代次数 n，二是作为依赖扩充的输入来规定依赖距离大小；第 2 行根据 *oldAlignments* 进行依赖扩充（3.4.1节），扩充的实例对构建为因子图（$factorGraph$）；第 3 行将因子图通过信念传播算法解算，获得实例对对齐的边缘概率；第 5-9 行，设置了阈值 θ，如果对齐概率值超过 θ，则将其加入新的对齐结果 *newAlignments*，否则不予考虑。

多次迭代一方面能够提高依赖距离近的实例对的优先级，因为 i 规定了初次优化算法只考虑直接依赖，而后才考虑传递依赖。根据观察，如果两个语句实例对齐，那么在近依赖距离中很可能存在应该对齐的语句实例，具体见3.4.1节。

另一方面，由于 BpAlign 算法的时间瓶颈在于因子图的解算，多次迭代能够有效降低因子图的规模，直观来说即少量多次。在因子图中，我们会为在初始对齐中和被依赖扩充到的语句实例对构建随机变量，初次迭代的随机变量包括初始对齐和直接依赖的实例对，规模较小，后续迭代虽然考虑传递依赖的实例对，但是每次通过阈值 θ 对结果进行剪枝，这样既能够扩大搜索空间，又能有效降低因子图的规模，提高运行速率。

3.4.1　依赖扩充

算法 3.3: 依赖扩充算法

```
1:  procedure PROPAGATE(alignments, n)
2:      worklist ← (alignments, 0)
3:      factorGraph ← initFactorGraph()
4:      while worklist is not empty do
5:          (I_{ℓ_i,t_j}, distance) ← pop(worklist)
6:          ApplyRules(factorGraph, I_{ℓ_i,t_j})
7:          if distance <= threshold then
8:              for I_{ℓ_p,t_q} ∈ dep(ℓ_i, n) × dep(t_j, n) do
9:                  worklist ← worklist ∪ {(I_{ℓ_p,t_q}, distance + 1) }
10:             end for
11:         end if
12:     end while
13:     return factorGraph
14: end procedure
```

算法 3.3描述了依赖扩充算法，对应优化算法 3.2的第 2 行。

依赖扩充算法由 *worklist* 算法所驱动。第 2 行将输入的对齐序列初始化为

worklist 队列，其中 0 表示初始扩充距离（不同于依赖距离）；第 3 行初始化因子图 *factorGraph*；第 4-12 行循环执行，直到队列为空；第 5 行从队头弹出一个实例对 I_{ℓ_i,t_j} 和 I_{ℓ_i,t_j} 的扩充距离 *distance*；第 6 行将 I_{ℓ_i,t_j} 建模为概率约束（3.4.2节），即作为随机变量和相应函数变量加入因子图中；为了防止搜索空间爆炸，设置了阈值 *threshold*（第 7 行）来限制搜索空间大小，扩充距离超过 *threshold* 的实例对被剔除；第 8-10 行基于当前的实例对 I_{ℓ_i,t_j} 进行依赖扩充，第 8 行中，$dep(I,n)$ 表示与实例 I 的控制依赖距离和数据依赖距离在 n 以内的不重复实例序列，n 的值与迭代次序有关，为 1 表示直接依赖，$dep(\ell_i,n)\times dep(t_j,n)$ 表示实例 ℓ_i 和 t_j 的依赖序列间的笛卡尔积；对于笛卡尔积中的每个实例对 I_{ℓ_p,t_q}，第 9 行将其与 *distance+1* 构成对，加入到 *worklist* 队列尾中。

本算法通过依赖距离 n 和扩充距离 *distance* 限制搜索空间大小来降低因子图的规模。n 在迭代中初始为 1，随着迭代次数递增，直观来说即先近后远。因为前序迭代的依赖距离小于后序迭代，因此通过 n 能够在前序迭代中先行为近依赖距离的实例对构建概率约束，在后续迭代中为远依赖距离的实例对构建约束，当 $n=1$ 时，$dep(I,n)$ 只搜索直接依赖项。依赖距离 n 结合扩充距离 *distance* 能够灵活地限制搜索空间的大小，具体见实验分析（第五章）。

3.4.2　概率推断

在每次的优化对齐过程中，我们通过概率推断计算实例对对齐的可能性，在最终对齐过程中，我们通过概率推断结果寻找最大后验概率对齐数量。初始对齐和依赖扩充的实例对被构建为概率约束，在概率约束中根据相关信息赋予先验概率，概率约束在因子图中以函数变量的形式存在，实例对以随机变量的形式存在；而后通过信念传播算法解算得出后验对齐概率；如果迭代未结束，则对齐概率达到阈值的实例对及其后验对齐概率会作为新的实例对和先验概率在下次迭代中参与构建因子图。本节详细说明概率约束和函数变量的定义。

(1) 概率约束

图 3.6展示了概率约束的定义和约束规则，其中 $\mathbb{P}$ 值的计算方式见下一小节。

定义：
ℓ, t：语句标签
ℓ_i, t_j：语句 ℓ/t 的第 i/j 个实例
I_{ℓ_i,t_j}：如果ℓ_i 和t_j 对齐则为真
$symb(\ell_i)$：ℓ_i 的符号表达式
$cdep(\ell_i, n)$：ℓ_i 距离为 n 内的直接控制依赖项的集合
$ddep(\ell_i, n)$：ℓ_i 距离为 n 内的直接数据依赖项的集合
$loop(\ell_i)$：如果ℓ_i 在循环结构中则为真

约束规则：
1. $symb(\ell_i) \equiv symb(t_j) \overset{\mathbb{P}}{\Rightarrow} I_{\ell_i,t_j}$
2. $I_{\ell_i,t_j} \overset{\mathbb{P}}{\Leftrightarrow} \forall \ell_p \in cdep(\ell_i, n),\ \exists t_q \in cdep(t_j, n),\ I_{\ell_p,t_q}$
3. $I_{\ell_i,t_j} \overset{\mathbb{P}}{\Leftrightarrow} \forall \ell_p \in ddep(\ell_i, n),\ \exists t_q \in ddep(t_j, n),\ I_{\ell_p,t_q}$
4. $loop(\ell_i) \wedge loop(t_j) \overset{\mathbb{P}}{\Rightarrow} I_{\ell_i,t_j}$

图 3.6　概率约束定义

约束规则 1 表示：如果有两个实例 ℓ_i 和 t_j 具有等同的符号表达式，那么它们对齐的概率为 $\mathbb{P}$。

规则 2 和规则 3 表示：假设有两个实例 ℓ_i 和 t_j 对齐，那么它们的控制依赖项和数据依赖项的对齐概率为 $\mathbb{P}$。注意此处的约束是双向的，即依赖项的概率约束会反向影响 I_{ℓ_i,t_j}，因为在对齐中并不会区分执行的先后次序，如果只考虑单向，则会导致在信念传播中后执行实例与先执行实例之间的消息不一致。

规则 4 是基于一个观察得出，即如果两个实例 ℓ_i 和 t_j 同在循环结构中，那么它们很有可能对齐，其中循环结构指 for 循环（for（单次表达式; 条件表达式; 末尾循环体）中间循环体；）的单次表达式、条件表达式或末尾循环体和 while（while（表达式）循环体）的表达式。本文通过抽象语法树判断是否符合条件。

```
mid = (low + high) / 2;
```

```
diff = high - low;
mid = low + diff / 2;
```

另外，规则 2 和规则 3 既考虑直接依赖项也考虑传递依赖项，因为由于同类功能的不同实现方式，两个等同的依赖项可能出现在一个版本的直接依赖和另一个版本的传递依赖中。上图所示代码片段展示了两种计算 *low* 和 *high* 的中间值的方式，左边的代码直接通过相加取平均值获得中间值 *mid*，*mid* 直接依赖

于 *low* 和 *high*；右边的代码通过先取 *high* 和 *low* 的差值再将差值的一半与 *low* 相加获得中间值，*mid* 直接依赖于 *low*，但传递依赖于 *high*，如果不考虑传递依赖，则会导致ℓ_{high} 和t_{high} 之间失去概率约束联系。

然而，我们不能考虑所有传递依赖项，否则所有与 I_{ℓ_i,t_j} 具有依赖关系的实例对都将具有高对齐概率，并且会因此构建大量的随机变量，导致因子图空间爆炸。因此，我们通过控制依赖距离的方式来提高近依赖距离依赖项的优先级，并限制因子图大小（3.4.1节）。

通过以上四个规则，能够在每次迭代过程中把代码中的符号表达式、运行值、依赖关系和结构特征构建为概率约束。

(2) 函数变量

我们将图 3.6的概率约束定义和规则构建为因子图。因子图包含两种类型的节点，即随机变量和函数变量。在因子图中，随机变量的边缘概率通过函数变量计算得出，函数变量是一个概率函数，其变量为一列随机变量。I_{ℓ_i,t_j} 对应于因子图中的随机变量，如果 ℓ_i 和 t_j 对齐，那么 I_{ℓ_i,t_j} 为真，否则为假。

图 3.7展示了本文所定义的函数变量。其中概率值对应上一小节的对齐概率 $\mathbb{P}$, 我们用 $H(HIGH) = 0.9$ 表示很有可能对齐；$P(POSITIVE) = 0.6$ 表示倾向于对齐；$U(UNCERTAIN) = 0.5$ 表示无法确定是否对齐；$N(NEGATIVE) = 0.4$ 表示倾向于不对齐；$L(LOW) = 0.1$ 表示对齐可能性很小。这样的设置方式是概率推断中的常用手段[79]。其中概率值 H 的含义是如果实例对具备该特征，则很可能对齐，L 同理；倾向性的概率值 P 的含义是如果实例对具备该特征，则有一定可能对齐，如果需要超过阈值，则需要叠加多个倾向性的概率值的信念，N 同理。

$f_S(I_{\ell_i,t_j})$ 将图 3.6的规则 1 建模为函数变量，当 ℓ_i 和 t_j 的符号表达式等同时，赋予随机变量高概率值 H；根据直觉，我们认为值相等但符号表达式不同或不具有符号表达式的实例对对应的随机变量赋予无法确定的概率值 U；其他情况下，当运行值不同时，赋予低概率值 L 表示对齐可能性低。

$f_{CD}(I_{\ell_i,t_j}, I_{\ell_{d_1},t_{d_1}}, \cdots, I_{\ell_{d_p},t_{d_q}})$ 和 $f_{DD}(I_{\ell_i,t_j}, I_{\ell_{d_1},t_{d_1}}, \cdots, I_{\ell_{d_p},t_{d_q}})$ 将规则 2 和规则 3 建模为函数变量，当 ℓ_i 和 t_j 对齐，并且 ℓ_{d_i} 和 t_{d_j}，也就是 ℓ_i 和 t_j 的依赖项对齐时，赋予高概率值 H，否则赋予低概率值 L。

$$
\begin{aligned}
&f_S\left(I_{\ell_i,t_j}\right)\\
&=\begin{cases}H & \text{if } symb(\ell_i)\equiv symb(t_j)\\ L & \text{if } value(\ell_i)\neq value(t_j)\\ U & \text{otherwise}\end{cases}\\
&f_{CD}\left(I_{\ell_i,t_j},I_{\ell_{d_1},t_{d_1}},\cdots,I_{\ell_{d_p},t_{d_q}}\right)\\
&=\begin{cases}P & \text{if } I_{\ell_i,t_j}\wedge\{(\ell_{d_i},t_{d_j})\in cdep(\ell_i)\times cdep(t_j)\mid I_{\ell_{d_1},t_{d_1}}\vee\cdots\vee I_{\ell_{d_p},t_{d_q}}\}\\ N & \text{otherwise}\end{cases}\\
&f_{DD}\left(I_{\ell_i,t_j},I_{\ell_{d_1},t_{d_1}},\cdots,I_{\ell_{d_p},t_{d_q}}\right)\\
&=\begin{cases}P & \text{if } I_{\ell_i,t_j}\wedge\{(\ell_{d_i},t_{d_j})\in ddep(\ell_i)\times ddep(t_j)\mid I_{\ell_{d_1},t_{d_1}}\vee\cdots\vee I_{\ell_{d_p},t_{d_q}}\}\\ N & \text{otherwise}\end{cases}\\
&f_L\left(I_{\ell_i,t_j}\right)\\
&=\begin{cases}H & \text{if } loop(\ell_i)\wedge loop(t_j)\\ U & \text{otherwise}\end{cases}
\end{aligned}
$$

图 3.7　函数变量定义

$f_L(I_{\ell_i,t_j})$ 将规则 4 建模为概率约束，当 ℓ_i 和 t_j 在循环结构中时，则赋予高概率值。这样的做法虽然简单，但十分有效地避免了 Apex 中常见的错误对齐。例如在以下代码片段中，Apex 会错误地将左边的 $i=0$ 和右边的 $count=0$ 对齐，通过 $f_L(I_{\ell_i,t_j})$，我们能够解决这一问题。

```
for(i = 0; ...; ...)
```

```
int count = 0;
for(i = 0; ...; ...)
```

除了以上四种函数变量外，初始对齐赋予的概率值也以函数变量表达，用 f_P 表示。通过函数变量，能够在每次迭代过程中把概率约束建模为因子图，而后通过概率分析引擎，即信念传播，解算因子图获得边缘概率分布。

3.5　最终对齐

优化对齐阶段计算了实例对之间是否对齐的后验概率，并且将超过阈值 θ 的实例对返回。在返回结果中，同一个实例可能与多个实例之间对齐（后验概率超过阈值），而实际对齐寻求的是一对一对齐，并且最大化对齐数量。

算法 3.4: 最终对齐算法

```
1:  procedure FinalAlign(alignments)
2:      alignments ← sort(alignments, key = lambda I_{ℓ_i,t_j} : (i, j))
3:      results ← {}
4:      residues ← {}
5:      for I_{ℓ_i,t_j} in alignments do
6:          while SAT_1(I_{ℓ_i,t_j}, results[−1]) ∧ SAT_2(I_{ℓ_i,t_j}, results[−1]) do
7:              residues ← residue ∪ {results[−1]}
8:              results ← results\{results[−1]}
9:          end while
10:         if SAT_2(I_{ℓ_i,t_j}, results[−1]) || ¬SAT_1(I_{ℓ_i,t_j}, results[−1]) then
11:             results ← results ∪ {I_{ℓ_i,t_j}}
12:         end if
13:     end for
14:     for I_{ℓ_i,t_j} in residues do
15:         if SAT_3(I_{ℓ_i,t_j}, results) then
16:             results ← results ∪_insert {I_{ℓ_i,t_j}}
17:         end if
18:     end for
19: end procedure
```

算法 3.4展示了最终对齐的方法，输入变量 *alignments* 代表优化对齐的实例对序列，序列中实例对的对齐概率均达到阈值 θ。

算法中考虑三个满足约束，如图 3.8所示，SAT_1 表示如果当前实例对 I_{ℓ_i,t_j} 中 ℓ_i 和 t_j 的次序均后于 I_{ℓ_x,t_y}，则返回假，否则返回真；SAT_2 表示当输入实例对 I 对齐概率高于实例对 R 时，则返回真，否则返回假；SAT_3 表示如果集合 $\mathbb{R}$ 中存在两个实例对，其中一个实例对的实例次序均先于 I_{ℓ_i,t_j}，另一个实例对的实例次序均后于 I_{ℓ_i,t_j}，则返回真，否则返回假。

算法第 2 行首先将优化对齐的结果根据实例对 I_{ℓ_i,t_j} 的先后次序由小到大排序，如果 i 相等，则根据 j 的大小排序；第 3-4 行分别初始化了两个数组，分别是结果集 *results* 和剩余集 *residues*；对于实例对序列中的每一个实例对（第 5-13 行），如果满足 SAT_1 并且结果集的最后一个实例对的后验概率低于当前实例对的后验概率，则将其加入剩余集并删除该实例对，该操作循环进行直到不满足条件（6-9 行），如果当前实例对的后验概率高于最后一个实例对，或不满足 SAT_1，则将其加入结果集中（10-12 行），其目的是避免重复序号的实例对出现在对齐中（SAT_1），并且当出现重复时保留后验概率最高的实例对（SAT_2）。

目前为止并不能保证对齐序列的最大化，存在如下情况：假设实例对 I_{ℓ_2,t_1}

定义：
ℓ, t：语句标签
ℓ_i, t_j：语句 ℓ/t 的第 i/j 个实例
I_{ℓ_i,t_j}：ℓ_i 和t_j 组成的实例对
$I.probability$/$R.probability$：I/R 的对齐概率
$\mathbb{R}$：实例对集合
$\ell_i \rightsquigarrow t_j$：实例$\ell_i$ 直接/传递依赖于t_j

满足约束：

$$SAT_1(I_{\ell_i,t_j}, I_{\ell_x,t_y}) = \begin{cases} false & i > x \wedge j > y \\ true & otherwise \end{cases}$$

$$SAT_2(I_{\ell_i,t_j}, I_{\ell_x,t_y}) = \begin{cases} true & I_{\ell_i,t_j}.probability > I_{\ell_x,t_y}.probability \\ false & otherwise \end{cases}$$

$$SAT_3(I_{\ell_i,t_j}, \mathbb{R}) = \begin{cases} true & \neg\exists I_{\ell_x,t_y} \in \mathbb{R} \wedge (\ell_i \rightsquigarrow \ell_x \wedge \ell_y \rightsquigarrow t_j) \\ & \wedge(\ell_x \rightsquigarrow \ell_i \wedge t_j \rightsquigarrow \ell_y) \\ false & otherwise \end{cases}$$

图 3.8　满足约束定义

的对齐概率高于 I_{ℓ_1,t_1}，根据规则将 I_{ℓ_1,t_1} 从结果集中删除，而后有 I_{ℓ_2,t_2} 的对齐概率高于 I_{ℓ_2,t_1}，根据规则将 I_{ℓ_2,t_1} 从结果集中删除，此时结果集中只有 I_{ℓ_2,t_2}，而 I_{ℓ_1,t_1} 也应当包含在结果集中。因此在 14-18 行进行剩余对齐，对于剩余集中的每个实例对 I_{ℓ_i,t_j}，如果满足插入条件（SAT_3），则将该实例对插入结果集中。剩余对齐保证了对齐实例之间在依赖关系上不存在交叉情况，即如果实例 ℓ_i 和 t_j 对齐，那么不存在两个对齐实例 ℓ_x 和 t_y，其中 ℓ_i 直接依赖或传递依赖于 ℓ_x，t_y 直接依赖或传递依赖于 t_j，反之亦然。

3.6　本章小结

针对错误学生代码和正确教师版本的对齐问题，本章提出了一种基于对齐算法和概率推断的代码自动对齐方法，并设计实现了工具 BpAlign。首先通过动态分析和符号分析提取错误代码和正确版本的跟踪信息，通过序列对齐将两段跟踪信息根据符号表达式等特征赋予初始概率值，然后在优化对齐阶段设计了一系列的概率约束以将代码信息建模，通过信念传播获得后验概率，最后通过最终对齐获得对齐结果。

第四章　基于信念传播的代码自动反馈生成

立足于现有方法在对齐精确度的不足、如何获取与错误代码相似的正确版本以及如何根据对齐结果生成反馈的三大挑战，为了获得质量更高的反馈文档，从而帮助学生提高学习效率，本章提出了一种基于基于信念传播的代码自动反馈生成技术，并实现了工具 Bpex。首先阐述研究动机；随后对方法进行概述；接下来对方法中聚类算法、关键实例切片和动态切片对齐图展开详细介绍。

4.1　研究动机

大部分自动错误分析和自动修复工具通过将错误学生代码和正确教师版本进行部分匹配来将正确位置排除，从而在无法匹配的错误位置生成反馈。反馈的生成方式多种多样，然而反馈是否合理取决于匹配是否准确。

现有工作基于仅在错误位置存在差异的假设，而忽略正确位置也可能存在差异[23]，而正确位置的差异很可能被当作错误部分而参与生成反馈，这样的反馈是不准确的。

Apex 通过动态分析和符号分析将语义差异对齐，然而在3.1节中，Apex 在对齐的准确度上存在缺陷，同样会导致反馈内容不准确。

BpAlign 将错误学生代码和正确教师版本的进行对齐，通过概率推断来容忍不确定性，提高对齐的准确度。然而在反馈生成方面面临着三个挑战：

1. 如果对齐算法的精确度低，则会误将错误位置与正确版本的某个位置对齐或未将正确位置与正确版本的对等位置对齐，而根据错误的对齐结果生成的错误解释，显然是错误的。
2. 算法题具有多种解法，并且学生提交的实现存在多种差异，教师仅能提供数量有限的正确版本，然而错误代码和正确版本之间的算法不同或者语义差异过大会导致反馈生成的结果不具备参考价值，为所有错误提交代码批量定制正确版本难度大。
3. 仅仅将对齐结果反馈给学生，学生会被海量的切片数据所困扰，很难发现与错误相关的重点，从而无法达到提高学习效率的目的。

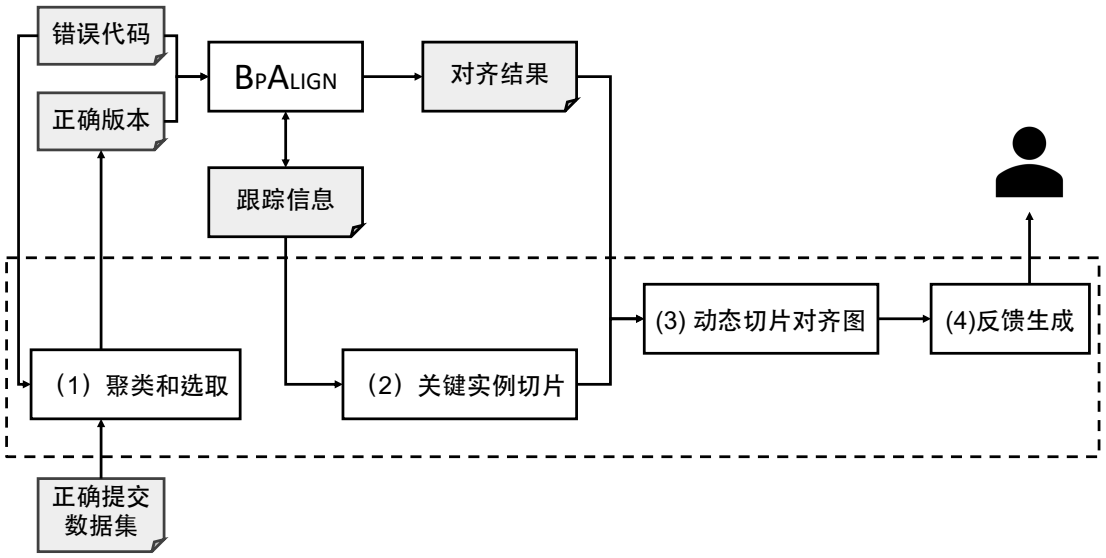


图 4.1　Bpex 方法概览图

为了解决挑战 1，BpAlign 通过引入概率化措施来提高对齐的准确度。为了解决挑战 2，本章提出了基于序列对齐的代码聚类方法。为了解决挑战 3，本章提出了在 BpAlign 基础上的动态切片对齐图构建方法，并基于动态切片对齐图生成反馈文档。

4.2　方法概述

我们设计实现的工具 Bpex 的总体框架如图 4.1所示，其中对齐阶段调用 BpAlign 方法获取对齐结果，跟踪信息沿用 BpAlign 中的程序跟踪输出。Bpex 包括 4 步处理流程，分别是正确提交数据集聚类和选取、关键实例切片、动态切片对齐图构建和反馈生成，最终反馈输出给学生或老师。

聚类和选取阶段：本章通过将正确提交的数据集进行聚类的方式复用正确学生代码，将其作为错误学生代码的正确版本。我们基于序列对齐的相似度将向量化的正确版本数据集聚类，并以该相似度为基准在聚类中选取与错误代码相似的正确版本。

关键实例切片阶段：本章以不符合预期的输出结果为出发点进行反馈生成。与不符合预期输出直接相关的实例即为关键实例，关键实例的所在位置可能同时在错误代码和正确版本中，也可能仅在正确版本或错误代码中。为提取准确的关键实例信息，我们基于错误代码和正确版本的输出实例信息来确定关键实

例类型和位置。我们对关键实例序列进行切片构成动态切片，其中切片考虑控制依赖和数据依赖，切片结果被用于构建动态切片对齐图。

动态切片对齐图阶段：我们给出动态切片对齐图的定义和构建方式。定义主要包括实例结点、依赖关系和对齐关系；构建所需输入为 BpAlign 的对齐结果和动态切片，构建过程根据对齐结果将动态切片进行对齐并且对图中基本块进行分类。

反馈生成阶段：基于动态切片对齐图和图中基本块类型的划分，将图转换为自然语言表述的文档。

4.3 聚类和选取

在现实场景中，针对特定算法题，教学者需要花费大量时间编写多种算法的正确范例。此外，考虑到学生编写风格方式不同，同一种算法经过不同的重构可能产生截然不同的句法结构。因此，仅靠教学者为错误学生代码编写相似的正确版本效率低下，违背本方法的初衷。在工业界难以为错误代码寻找合适的正确版本，但是在教学领域，一道算法题的提交中存在大量的正确代码可供利用[80]。因此，我们将以聚类的方法为群策群力的过程建模，利用学生提交中能够通过测试的所有正确代码作为错误代码的反馈生成范本，为错误提交选择尽可能相似的正确版本。

4.3.1 跟踪信息向量化

为了降低时间开支并尽可能提高匹配率，我们从代码的跟踪信息提取特征跟踪信息，并将其转化为数值向量，即数值感知特征向量（Value-aware Characteristic Vector）。

定义 4.1 (特征跟踪信息) 给定代码的跟踪信息T，特征跟踪信息$\mathbb{T} =< t_1, ..., t_n >$为 T 中实例类型为赋值语句实例的集合，其中 t_i 指动态执行中第 i 个被执行的赋值语句实例。

定义 4.2 (数值感知特征向量) 给定代码的特征跟踪信息$\mathbb{T}$，$\mathbb{T}$ 的数值感知特征向量 $\mathbb{V}$ 为 $< v_1, ..., v_n >$，其中 v_i 指在 $\mathbb{T}$ 中第 i 个实例的运行值。

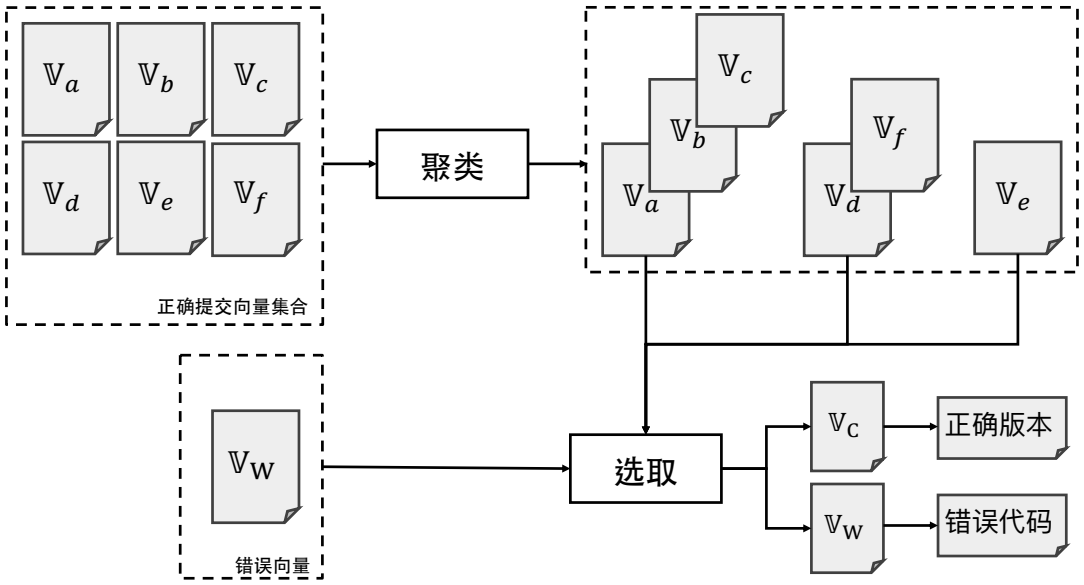


图 4.2　聚类选取算法概览图

以图 2.4为例，该跟踪信息的特征跟踪信息为 $< I_1, I_4, I_7, ..., I_{10} >$，其中 I_i 表示序号 i 的实例，其数值感知特征向量 $\mathbb{V}$ 为 $< 3, 5, 8, ..., 10 >$。

在特征跟踪信息中，我们不考虑符号表达式是因为：需要通过约束求解器判断符号表达式是否等同，与数值比较相比花费时间更多；我们不考虑其他语句实例如判断类型语句实例是因为：与数值相比，判断类型语句实例只具有两种运行值（True 和 False），并且即使两个判断类型语句实例的运行值相同，其实际语义很可能不同，即使运行值不同，也可能有相同的语义，考虑判断类型语句实例会增加聚类的误报率（False Positive）。

4.3.2　聚类选取算法

对错误代码和正确版本要求为相似，但非寻求最优解。因此，为了兼顾效率并利用已有算法，我们以序列对齐为引导对正确版本聚类，并为错误代码在聚类中寻找最匹配的正确版本，其中相似度的计算方式如下：

$$sim(\mathbb{V}_l, \mathbb{V}_t) = \frac{|lcv(\mathbb{V}_l, \mathbb{V}_t)|}{min(|\mathbb{V}_l|, |\mathbb{V}_t|)} \tag{4.1}$$

其中 l 和 t 分别代表两个代码，分子 $lcv(\mathbb{V}_l, \mathbb{V}_t)$ 表示向量 $\mathbb{V}_l$ 和 $\mathbb{V}_t$ 的最长公共子向量（Longest Common Subvector, LCV），由序列对齐算法解出（算法 3.1），得分函数替换为判断特征向量 u_i/v_j 的第 i/j 个运行值是否相等，相等返回 1，否则

返回 0；分母 $min(|\mathbb{V}_l|, |\mathbb{V}_t|)$ 取两个特征向量长度的最小值。

图 4.2描述了聚类选取的过程。算法将正确提交向量集合和错误代码对应向量作为输入，将错误代码和其相似度最高的正确版本作为输出。

(1) 聚类算法

聚类算法可以不定期在服务器维护，而不必每次分析错误代码时调用。首先将正确提交代码向量化后的集合进行聚类；对于当前错误代码的向量，从聚类中选取相似度最高的聚类；最后将错误向量与正确向量对应代码输出，作为 BpAlign 的输入。

算法 4.1: 聚类算法

```
1:  procedure CLUSTER(𝕍)
2:      flag ← false
3:      for 𝕍_i in 𝕍 do
4:          for cluster in clusters do
5:              if sim(𝕍_i, min(cluster, key = lambda c : c.length)) > γ then
6:                  flag ← true
7:                  cluster ← cluster ∪ {𝕍_i}
8:              end if
9:          end for
10:         if not flag then
11:             clusters ← clusters ∪ {𝕍_i}
12:         end if
13:     end for
14: end procedure
```

算法 4.1描述了聚类算法的具体流程，在 3-13 行中，对于当前数据集中的每个正确提交代码的特征向量，在聚类集合中遍历寻找相似度达到阈值 γ 的聚类，如果有，则将其加入聚类中，否则以当前特征向量创建新的聚类。第 5 行中，我们在聚类中选取长度最小的特征向量是基于两个直觉观察，首先教学者在拟定正确范本时，对于一道算法题的多种实现，往往倾向于最精简的表达；其次，在反馈生成中（4.6节），精简的正确代码鼓励算法输出”You should not have done”，即更多的错误因果链分析、更少的修复意见，因为过多的修复意见可能导致学生“过拟合”（1.1节）。

(2) 选取算法

选取算法在每次的评估时被调用。根据给定的错误学生代码，选取算法从对应的正确版本聚类中遍历，并与错误学生代码的特征向量比较相似度，最终相似度最高的正确版本特征向量对应的代码被返回，与错误代码组合为错误代码和正确版本，作为 BpAlign 的输入。

算法 4.2: 选取算法

```
1:  procedure PICK(clusters, 𝕍_W)
2:      minSim ← 1
3:      𝕍_C ← ∅
4:      for cluster in clusters do
5:          𝕍_candidate ← min(cluster, key = lambda c : c.length)
6:          curSim ← sim(𝕍_i, 𝕍_candidate)
7:          if curSim > minSim then
8:              minSim ← curSim
9:              𝕍_C ← 𝕍_candidate
10:         end if
11:     end for
12:     return 𝕍_W, 𝕍_C
13: end procedure
```

算法 4.2描述了选取算法的具体流程，在 4-11 行中，对于当前聚类中的每个正确提交向量，在聚类集合中选取与错误代码特征向量 $\mathbb{V}_W$ 最相似的正确版本向量，同样选取长度最小的向量作为候选向量，第 12 行将错误代码特征向量和正确版本特征向量返回。

4.4　关键实例切片

错误代码的动态跟踪信息中既包含了错误相关的语句实例，又包含了错误无关但无法对齐的语句实例，本节通过关键实例切片算法从错误代码和正确版本的跟踪信息筛选出与错误有关的实例并基于依赖关系构建切片。

4.4.1　关键实例提取

我们以输出语句实例为引导提取关键实例。输出语句实例为输出语句在某次执行的动态跟踪信息，主要包含两部分，分别为输出串和操作数，输出串即标准输出结果，操作数即输出串中的引用变量，均通过动态分析获得。如输出

语句 *printf("%d+%d",a,b)*，在某次执行中 *a=1,b=2*，那么该语句实例的输出串为*"1+2"*，操作数分别为 1 和 2。此外动态跟踪还会记录实例以及操作数的控制和数据依赖关系（图 2.4b）。

算法 4.3: 输出实例对齐算法

```
1:  procedure AlignPrints(L, T)
2:      alignments ← {}
3:      minLen ← min(L.length, T.length)
4:      for i in range(0, minLen) do
5:          alignments ← alignments ∪ {(l_i, t_i)}    // l_i/t_i = L[i]/T[i]
6:      end for
7:      if minLen < L.length then
8:          for i in range(minLen + 1, L.length) do
9:              alignments ← alignments ∪ {(l_i, ∅)}
10:         end for
11:     else if minLen < T.length then
12:         for i in range(minLen + 1, T.length) do
13:             alignments ← alignments ∪ {(∅, t_i)}
14:         end for
15:     end if
16:     return alignments
17: end procedure
```

对于错误代码和正确版本的所有输出实例，首先将它们合并。算法 4.3描述了合并过程，给定两段输出实例，其中 L 表示错误代码的输出实例序列，T 表示正确版本的输出实例序列，按照先后执行次序排列。第 3 行取两段序列中最短长度 *minLen*；第 3-6 行将两个序列前 *minLen* 项结对并存入 *alignments* 集合，为了方便表示，用 l_i 表示序列 L 的第 i 个实例，t_i 同理[1]；第 7-15 行将较长的序列的剩余实例与空实例 ∅ 结对存入集合；第 16 行返回 *alignments* 集合。

对于 *alignments* 中的每个实例对，按先后次序进行比较，关键实例存入关键实例集合 $\mathbb{C}$。首先比较输出串以定位不一致的输出实例，公式4.2定义了输出串的比较方式，当 l_i 的输出串 $S(l_i)$ 与 t_i 的输出串 $S(t_i)$ 不相等，或 l_i 和 t_i 其中之一为空时，对于给定序列 *alignments*，第一个满足公式4.2的实例对被存入 $\mathbb{C}$。

$$\frac{S(l_i) \neq S(t_i) \vee l_i = \varnothing \vee t_i = \varnothing, \mathbb{C}' \leftarrow \mathbb{C} \cup \{(l_i, t_i)\}}{< l_i, t_i > \cdot \mathbb{C} \rightarrow \mathbb{C}'} \tag{4.2}$$

对于上面提到的 *printf("%d+%d",a,b)*，假设预期输出为*"1+2"*，而实际输出

[1]注：此处的 i 不同于先前所述的执行次序

为*"1+3"*，如果从输出实例的粒度切片，在控制依赖上并无不妥，但是由于 a 的结果正确，在数据依赖上会包含正确结果的切片，从而导致冗余的反馈。Bpex 通过比较输出串定位错误的输出实例，为了精准定位关键实例，我们进一步比较操作数。

算法 4.4: 操作数对齐算法

```
1:  procedure AlignOperands(l_i, t_i)
2:      alignments ← {}
3:      minNum ← min(n(l_i), n(t_i))
4:      for j in range(0, minNum) do
5:          alignments ← alignments ∪ {(l_ij, t_ij)}
6:          // l_ij/t_ij = l_i/t_i 的第 j 个操作数
7:      end for
8:      if minNum < n(l_i) then
9:          for i in range(minNum + 1, n(l_i)) do
10:             alignments ← alignments ∪ {(l_ij, ∅)}
11:         end for
12:     else if minNum < n(t_i) then
13:         for i in range(minNum + 1, n(t_i)) do
14:             alignments ← alignments ∪ {(∅, t_ij)}
15:         end for
16:     end if
17:     return alignments
18: end procedure
```

对于 $\mathbb{C}$ 中的实例对，如果 l_i 为空，则说明错误代码缺少输出；如果 t_i 为空，则说明错误代码冗余输出。这两种情况下结束关键实例提取，否则进一步提取操作数作为关键实例。

算法 4.4描述了操作数提取方式，其中 $n()$ 计算输出语句实例中操作数的个数。算法 4.3将错误代码和正确版本的输出序列构成序列对，然后在序列对中定位第一个不一致输出串的输出实例对，与算法 4.3类似，算法 4.4将不一致输出实例对构成操作数对，然后在操作数对序列中定位第一个不一致的操作数对，由于操作数不是语句实例类型，因此我们将操作数直接数据依赖的语句实例加入关键实例集合 $\mathbb{C}$ 中（公式4.3）。

$$\frac{l_{ij} \neq t_{ij} \vee l_{ij} = \varnothing \vee t_{ij} = \varnothing, \mathbb{C}' \leftarrow \mathbb{C} \cup \{(ddep(l_{ij}, 1), ddep(t_{ij}, 1))\}}{< l_{ij}, t_{ij} > \cdot \mathbb{C} \rightarrow \mathbb{C}'} \tag{4.3}$$

4.4.2　动态切片

对于关键实例集合 $\mathbb{C}$ 中的每个语句实例对，我们通过控制依赖和数据依赖关系分别对实例对中的两个语句实例制作动态切片。因此，动态切片为两段，分别为错误代码的切片和正确版本的切片。

算法 4.5: 切片提取算法

```
1:  procedure makeSlices(ℂ)
2:      L ← {}
3:      T ← {}
4:      for c in ℂ do
5:          L ← L ∪ c[0]
6:          T ← T ∪ c[1]
7:      end for
8:      𝕃 ← makeSlice(L)
9:      𝕋 ← makeSlice(T)
10:     return 𝕃, 𝕋
11: end procedure
12: procedure makeSlice(criticals)
13:     slice ← {}
14:     for critical in criticals do
15:         slice ← slice ∪ ddeps(critical)
16:         slice ← slice ∪ cdeps(critical)
17:     end for
18:     slice ← sort(slice, key = lambda ℓ_i : (i))
19:     return slice
20: end procedure
```

算法 4.5描述了切片提取算法，第 2-7 行将关键实例集合 $\mathbb{C}$ 的语句实例对拆分为两个关键实例序列，L 表示错误代码的关键实例，T 表示正确版本的关键实例；对于两段关键实例序列分别调用函数 makeSlice 来制作切片（第 8-9 行）；第 12-20 行描述了切片制作方法，对于关键实例序列中的每个语句实例 $critical$，$ddeps(critical)$ 从跟踪信息中获取该语句实例的所有数据依赖实例，包括直接数据依赖和传递数据依赖（图 2.4b中的数据依赖），同样地，$cdeps(critical)$ 获取该语句实例的所有控制依赖实例（图 2.4b中的控制依赖）；第 18 行将切片中的语句实例按被执行的先后次序从先往后排列；第 19 行返回结果，两段切片结果将被用于构建动态切片对齐图。

4.5　动态切片对齐图

很多错误分析工具直接将与错误输出有关的动态切片反馈给开发者，它们无法分析切片中的每个实例是否计算错误、是否缺失某些计算或判断语句、是否本不应当被执行。BPEX 以正确版本的切片为指导，以 BPALIGN 为桥梁，将错误代码和正确版本的关键实例动态切片进行对齐，构建动态切片对齐图。

定义4.3为动态切片对齐图的定义，其中 I_{ℓ_i,t_j} 表示实例对 ℓ_i 和 t_j，$\mathbb{L}$，$\mathbb{T}$ 分别表示错误代码和正确版本的动态切片。

定义 4.3 (动态切片对齐图) 动态切片对齐图$G =< N, D, R, W, C >$是一个五元结构，其中 $< N, D >$ 是一个有向图。

N 是语句实例结点的集合，$n_0 \in N$ 是语句实例结点。

$D = D_{data} \cup D_{control}$ 分别是代表数据依赖和控制依赖关系的有向边，如果 $(n, m) \in D$，则 n 数据依赖或控制依赖于 m。

$R = R_A \cup R_U$ 分别代表两种对齐类型，R_A 表示对齐关系，如果 $I_{\ell_i,t_j} \in R_A$，则 ℓ_i 和 t_j 对齐，对齐结果由 *BPALIGN* 给出，R_U 表示对齐但不匹配关系，如果 $(I_{\ell_i,t_j}) \in R_U$，则 ℓ_i 和 t_j 对齐但不匹配（定义*4.4*）。

W 表示在 $\mathbb{L}$ 中非对齐关系 R 的实例集合。

C 表示在 $\mathbb{T}$ 中非对齐关系 R 的实例实例集合。

定义 4.4 (对齐但不匹配关系) 对齐但不匹配关系 R_U 指对于所有 $I_{\ell_i,t_j} \in R_U$，$I_{\ell_i,t_j} \notin R_A$，存在 $\ell_x, t_y \in R_A$，其中 $x < i, y < j$，l_i, l_x 指语句 l 分别在第 i 次和第 x 次执行时的语句实例，t_j, t_y 指语句 t 分别在第 j 次和第 y 次执行时的语句实例。

对于错误代码和正确版本的切片 $\mathbb{L}$，$\mathbb{T}$ 和 BPALIGN 的对齐结果 $\mathbb{A}$，我们首先为切片中的每个语句实例创建结点集 N，并且基于依赖关系创建边集 D；然后对切片的对齐实例进行关于关系 R_A 的标记，如公式4.4所示，在标记过程中通过 $\mathbb{M}$ 保存对齐过的语句，并且记录最后一个非关键实例的对齐实例对 I_{ℓ_x,t_y} 的序号 a 和 b[2]；考虑到对齐实例之间可能存在的未对齐实例，我们将其从切片中删除；为了标记关系 R_U，如公式4.5所示，分别从 $\mathbb{L}$ 的第 $a+1$ 个实例和 $\mathbb{T}$ 的第

[2]注：此处的序号 a, b 分别指 ℓ_x 在 $\mathbb{L}$ 的序号和 t_y 在 $\mathbb{T}$ 的序号

$b+1$ 个实例开始，如果 I_{ℓ_i,t_j} 满足对齐但不匹配关系，则标记为关系 R_U；最后，对于 $\mathbb{L}$ 和 $\mathbb{T}$ 中既不在 R_A 中也不在 R_U 中的实例，我们分别将它们标注为 W 和 C。

$$\frac{\ell_i \in \mathbb{L} \wedge t_j \in \mathbb{T} \wedge I_{\ell_i,t_j} \in \mathbb{A}, {R_A}' \leftarrow R_A \cup I_{\ell_i,t_j}, \mathbb{M}' \leftarrow \mathbb{M} \cup \{(l,t)\}}{< I_{\ell_i,t_j} > \cdot R_A \rightarrow {R_A}', < I_{\ell_i,t_j} > \cdot \mathbb{M} \rightarrow \mathbb{M}'} \tag{4.4}$$

$$\frac{\ell_i \in \mathbb{L} \wedge t_j \in \mathbb{T} \wedge I_{\ell_i,t_j} \notin \mathbb{A} \wedge (\ell_x, t_y) \in \mathbb{M} \wedge x < i \wedge y < j, {R_U}' \leftarrow R_U \cup I_{\ell_i,t_j}}{< I_{\ell_i,t_j} > \cdot R_U \rightarrow {R_U}'} \tag{4.5}$$

4.6　反馈生成

在动态切片对齐图中，结点所对应的实例具有两种类型，分别是赋值语句实例（Store Instance，STIns）和谓词语句实例（Predicate Instance，PRIns）。在关键实例提取部分（4.4.1节），我们有输出实例（Print Instance，PRINT）。基于这三种实例和动态切片对齐图，我们生成反馈。

在动态切片对齐图中，我们定义了关系 R_U 和实例集合 W, C，对于在 R_U 中实例对的实例ℓ_i 和t_j，在 W 中的实例ℓ_i 和在 C 中的实例t_j，我们首先获取每个集合中在跟踪信息里最先执行的实例，构成候选（**C**andidate）集合 $\mathbb{C}$，候选集合中的语句实例即为根源错误相关。

对于 $\mathbb{C}$ 中的每个实例 c，我们获取满足公式4.6的实例ℓ_x，构成起始（**S**tart）集合 $\mathbb{S}$，规则4.6描述了对于 c，I_{ℓ_x,t_y} 是其直接控制依赖项（用 $\mapsto$ 表示），ℓ_x 是 PRIns 类型的语句实例，并且 I_{ℓ_x,t_y} 在动态切片对齐图中具有关系 R_A。这样做的目的是获取错误代码在发生错误前执行的最后一次对齐的谓词语句，通过获取谓词语句为学生指示错误发生的位置。

$$\frac{I_{\ell_x,t_y} \in R_A \wedge c \in \mathbb{C} \wedge type(\ell_x) \equiv \text{PRIns} \wedge (c \mapsto \ell_x \vee c \mapsto t_y), \mathbb{S}' \leftarrow \mathbb{S} \cup \ell_x}{< I_{\ell_x,t_y} > \cdot \mathbb{S} \rightarrow \mathbb{S}'} \tag{4.6}$$

因为动态切片分为两段，所以 $\mathbb{S}$ 中实例的数量最多为 2，即其中一个在错误代码的切片中，一个在正确版本的切片中，最少为 0；我们根据实例的代码文本生成字符串，对于在错误代码切片中的起始实例的代码文本 **T** 有“When **T**, you should not have done”；对于在正确版本切片中的起始实例的代码文本 **T** 有“When **T**, you should have done”；如果未找到起始实例，则根据正确版本或错误代码切片分别生成“You should not have done”或“You should have done”。

此后，我们分别将错误代码和正确版本切片中在关系 R_U 和 W, C 中的实例的代码文本根据依赖关系接在起始文本中打印出来。对于 R_U 中的实例，我们只打印错误代码中实例的代码文本，并附以正确版本中的运行值；对于 W，我们将 W 的实例以依赖关系作为先后次序接在“(..., y/Y)ou should not have done”后；同样，对于 C，我们将 C 的实例以依赖关系作为先后次序接在“(..., y/Y)ou should have done”后。

综上所述，反馈包含了错误发生的起始位置（$\mathbb{S}$ 集合），错误的因果链关系（错误代码切片具有关系 R_U, W 的实例），修复建议（正确版本切片中具有关系 C 的实例），具体示例见5.4节。

4.7　本章小结

针对错误代码的反馈生成问题，本章提出了一种基于动态切片对齐图的代码自动反馈生成技术，并设计实现了工具 BPEX。

本章首先通过聚类方式选取与错误代码相似的正确版本，然后通过 BPALIGN 进行对齐；在获得对齐结果后，以不符合预期的输出结果为出发点生成反馈，我们首先实现了一个关键实例提取方法，从错误代码和正确版本的输出实例中提取关键实例信息，并对关键实例制作动态切片，基于动态切片和 BPALIGN 的对齐结果生成动态切片对齐图，根据动态切片对齐图进行反馈文档生成，反馈文档提供了错误发生的起始位置、错误因果链和修复建议。

第五章　实验评估及结果分析

为了研究 BpAlign 和 Bpex 在真实学生代码反馈生成场景中与现有方法的表现差异、搜索空间、概率约束规则和概率值的影响及实际应用中的用户体验，本章进行一系列的实验，包括数据集的收集与预处理、设置的研究问题以及实验结果分析，并对实验部分数据进行案例讨论来说明有效性，最后对局限性进行分析。

5.1　数据集收集与预处理

5.1.1　数据收集

我们通过 AutoGrader[1]提供的爬虫工具从 *CODECHEF*[2]收集了 10 道 C 语言实现的算法题的真实错误提交和正确提交作为数据集。考虑到本方法所面向的群体，即大课堂中的 C 语言初学者，10 道算法题难度以简单为主。

表 5.1描述了 10 道算法题的信息，其中 **# 算法题**表示算法题的名称，算法题描述和所有提交均能通过链接访问，如 ALEXTASK[3]；**# 行数**描述了每道算法题错误代码和正确版本的平均行数，从 19 到 70 行不等；**# 提交数**表示所收集的错误代码和正确版本提交数，总提交数为 5679，其中包括 3076 个能通过编译并执行但无法通过所有测试用例的错误提交，2603 个能够通过所有测试用例的正确提交；**# 语句实例数**表示通过动态分析获取错误代码和正确版本的跟踪中语句实例的平均数量，表中描述了所有类型实例数，考虑到部分实例类型，如加载实例（Load Instance，LDIns），通常与错误非直接相关，后续实验中仅考虑其中三种与错误直接相关的实例类型（BRIns、STIns 和 PRINT）。

5.1.2　数据预处理

得益于 OJ 平台具有大规模的正确提交数量，我们通过将正确提交根据相似度进行聚类的方式，来寻找与每一个错误提交相似的正确版本。根据4.3节中

[1]https://github.com/s3team/AutoGrader

[2]https://codechef.com

[3]https://www.codechef.com/problems/ALEXTASK

表 5.1　算法题信息

# 算法题	# 行数		# 提交数		# 语句实例数	
	错误代码	正确版本	错误代码	正确版本	错误代码	正确版本
ALEXTASK	51	48	256	245	1332	2766
ANKTRAIN	38	40	259	293	180	166
CHEFAPAR	34	31	296	246	472	425
CHRL4	35	43	437	45	261	296
DISHOWN	52	70	40	22	522	547
ENTEXAM	53	44	263	282	604	609
KOL16B	33	33	466	416	505	378
NOTINCOM	31	26	357	332	618	650
RGAME	34	36	235	202	372	372
TRISQ	20	19	467	520	414	403
平均值	38	39	308	260	528	661

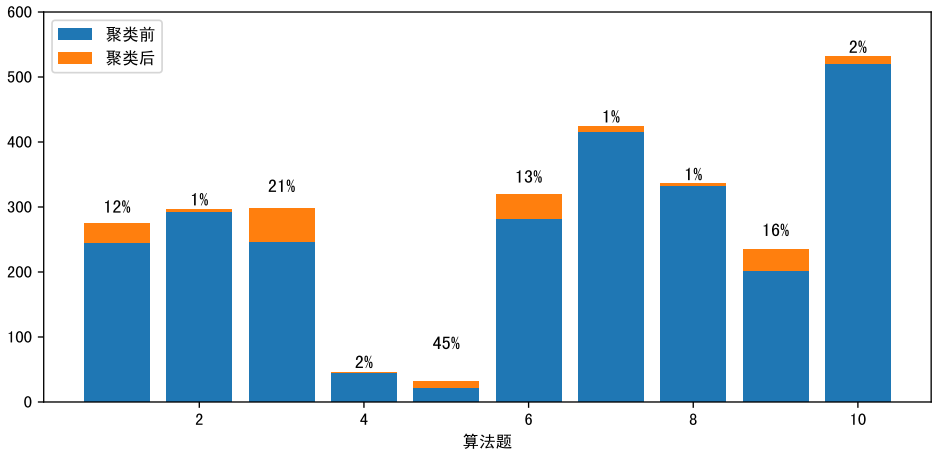


图 5.1　聚类结果

描述的聚类算法，我们设定 $\gamma = 0.9$ 进行聚类，平均每个算法题耗时为 17.15s，图 5.1展示了正确提交经过聚类的结果，其中 **# 聚类前**表示聚类前的提交数量，**# 聚类后**表示聚类后的数量。通过聚类我们能够将正确提交的数量平均压缩至原先的 11.4%。

为了提高评估效率，并且增强结果的说服力，我们基于以下启发式规则从中选取样本：在所有错误提交进行聚类，然后人工选取使用不同算法的错误提交，对于每个错误提交在正确代码聚类中选取相似度最高的版本，最终得到了 28 对错误代码和正确版本，如表 5.2所示，序号 1-10 与表 5.1中的 **# 算法题**一一对应；**# 提交数**展示了每道算法题对应的错误代码和正确版本数量；**# 行数**表示算法题的平均行数，代码行数从 20 到 67 行不等；**# 语句实例数**展示了每个算法题对应提交的平均语句实例数量，其数量远低于表 5.1，是因为在本文方法中只考虑 BRIns、STIns 和 PRINT 三种实例。

表 5.2 标记对齐样本

#	# 样本数	# 行数		# 语句实例数		# 标记对齐数
		错误代码	正确版本	错误代码	正确版本	
1	3	40	37	372	414	312
2	3	30	41	243	327	189
3	3	36	33	459	450	342
4	3	29	49	93	159	78
5	2	47	53	118	154	88
6	3	53	67	273	300	246
7	2	29	34	102	108	80
8	3	36	29	174	183	150
9	2	32	38	44	46	36
10	3	20	24	246	549	168

为了自动化算法评估，我们邀请了两名有程序分析研究背景的开发者（同实验室攻读硕士，与本文工作无关）对对齐实例对分别进行人工标注，标注方法是给定错误代码和正确版本的跟踪信息，两名开发者分别独立地从中标注应当对齐的实例对。标注完成后进行比对，如果标注结果有不一致对齐，则需要进行讨论达成一致结果。表 5.2中 **# 标记对齐数**展示了每个算法题对应提交的总标注数量，合计标注了 1689 个对齐实例对。

5.2 实验设置

本章实验所使用的主要语言为 Python3.7，工具包括 LLVM[71]和 Z3[73]，在数据集收集与预处理等阶段均使用处理器为 2.6 GHz 4-Core Intel Core i7、内存为 8G、操作系统为 Parallel Desktop 下的 Ubuntu 16.04 虚拟机完成。跟踪获取和符号表达式比较沿用了 Apex[23]的方法，在跟踪中动态地记录了语句实例的标签（如ℓ_i, t_j）、执行序号、符号表达式、运行值、控制依赖和数据依赖等信息，符号表达式通过 Z3 求解器比较，对齐和反馈生成均使用 Python3.7 开发，概率图模型通过 py-factorgraph[4]构建和解算，代码可公开获取[5]。

为了验证本文方法的有效性，我们提出了以下研究问题：

RQ1: BpAlign 与现有对齐方法 Apex 相比，对齐效果如何?

RQ2: 不同的搜索空间对 BpAlign 的对齐结果有什么影响?

RQ3: 不同组合的概率推断规则对 BpAlign 的对齐结果有什么影响?

[4]https://github.com/mbforbes/py-factorgraph

[5]https://github.com/BpexRepository/Bpex

RQ4: 函数变量中不同的概率值取值对 BpAlign 的对齐结果有什么影响?

RQ5: 与不聚类相比，聚类后选取正确版本在效率上的提升如何?

RQ6: Bpex 在为错误代码生成反馈的生成率如何?

RQ7: Bpex 与现有反馈生成方法相比，对学生的帮助如何?

为了考察本文提出方法不同方面的表现，需要与现有对齐方法和反馈生成方法进行对比。我们复现了具有代表性的基于对齐的错误分析方法 Apex[23]和基于修复的反馈生成方法 Clara[16]。

对于 **RQ1**，我们将 BpAlign 和 Apex 分别对表 5.2所描述的标注数据集对齐，并根据标注结果评估时间、准确率、回调率和 F_1 Score 四个指标。

对于 **RQ2**，我们通过调整优化对齐迭代次数 n 和依赖扩充搜索距离阈值 $threshold$ 来改变搜索距离的大小，以 Apex 为基线方法，比较时间、准确率、回调率和 F_1 Score 四个指标。

对于 **RQ3**，我们通过改变概率推断规则的组合产生了 5 个 BpAlign 的变体，并对这些变体在 F_1 Score 的表现进行评估。

对于 **RQ4**，我们通过尝试不同函数变量中概率值的取值产生了 8 个 BpAlign 的变体，并对这些变体在时间、准确率、回调率和 F_1 Score 的表现进行评估。

对于 **RQ5**，我们通过分别在所有数据集中选取与错误代码相似的正确版本和在聚类中进行选取来比较两种选取方式在时间和准确度上的指标。

对于 **RQ6**，我们通过 Bpex 对表 5.1中描述的算法题进行反馈生成，观察反馈生成率。

对于 **RQ7**，我们选取了 8 对 Clara 成功生成反馈的错误代码和正确版本，并邀请了 16 名参与者基于 Bpex、Clara 和 Apex 的反馈信息进行错误修正，记录修复时间和人工评价。

5.3 实验结果分析

RQ1. BpAlign 与现有对齐方法 Apex 相比，对齐效果如何?

为了回答 **RQ1**，我们将 BpAlign 与 Apex 在时间、准确率（Precision）、回调率（Recall）和 F_1 score ($F_1 = 2 * \frac{precision*recall}{precision+recall}$) 上进行比较，对齐结果见表 5.3。

序号 1-10 与表 5.1中的 **# 算法题**一一对应；**# 对齐数**表示每道算法题对应提

表 5.3 Apex 和 BpAlign 的对齐结果

#	# 对齐数		# 时间 (s)		# Precision		# Recall		# F_1 Score	
	Apex	BpAlign	Apex	BpAlign	Apex	BpAlign	Apex	BpAlign	Apex	BpAlign
1	110	105	2.89	6.47	0.91	0.98	0.96	1	0.93	0.99
2	66	61	1.45	2.38	0.93	1	1	0.92	0.96	0.96
3	120	103	2.35	5.34	0.87	0.98	0.93	0.89	0.9	0.93
4	28	26	0.23	0.67	0.85	0.97	0.95	0.99	0.89	0.98
5	45	35	1.41	1.37	0.61	0.92	0.64	0.74	0.62	0.82
6	85	75	1.35	3.46	0.91	0.98	0.94	0.9	0.92	0.94
7	41	37	0.84	1.52	0.98	1	1	0.9	0.99	0.95
8	54	49	1.29	2.53	0.86	0.99	0.91	0.98	0.88	0.99
9	18	17	0.72	0.32	0.93	1	0.93	0.93	0.93	0.96
10	61	56	2.83	5.07	0.84	0.93	0.92	0.87	0.87	0.89
平均值	63	56	1.54	2.91	0.87	0.97	0.92	0.91	0.89	0.94

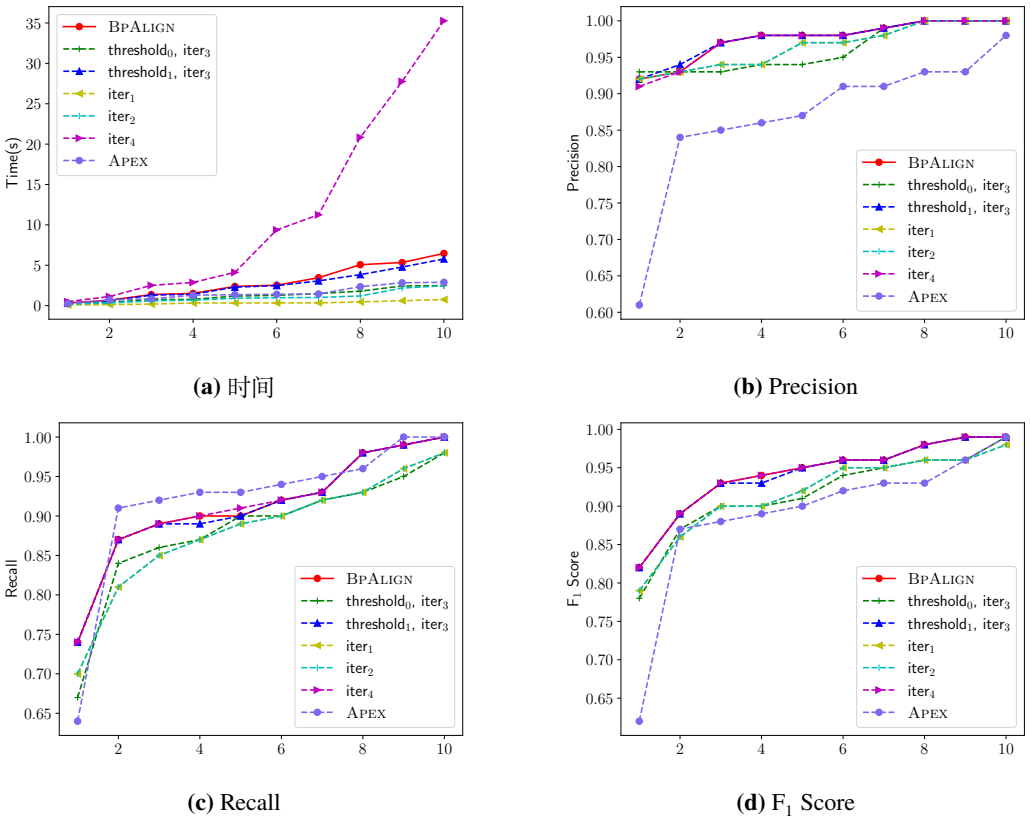
交的平均对齐数，在 **# 对齐数**中，相较于 BpAlign，Apex 输出了更多数量的对齐对，因为 Apex 将对齐问题建模为 MAX-SAT 问题，意在寻找最大数量满足符号表达式等同的实例对，而 BpAlign 的对齐建立在满足概率阈值 θ 前提下，故数量相对偏少。

 # 时间 (s) 表示为每个算法题提交生成反馈所用平均时间，BpAlign 比 Apex 花费更多时间，是因为 BpAlign 的时间瓶颈在于对因子图进行概率推断，因为因子图的大小与其承载信息相关，而承载信息与搜索空间大小相关，我们必须要保证一定的搜索空间才能够提高对齐的准确度。从数据上看，BpAlign 平均花费时间为 2.91 秒，而 Apex 是 1.54 秒。总体来说，BpAlign 比 Apex 花费时间不超过 2 倍，在实际情况中可以接受。在 **# Precision** 中，BpAlign 达到了 97% 的准确率，高于 Apex 的 87%。特别在算法题 DISHOWN 中，BpAlign 达到了 92% 的准确率，而 Apex 仅有 61%。Apex 的回调率（**# Recall**）略高于 BpAlign，因为 Apex 致力于寻找最大满足符号表达式等同条件的对齐数量，较 BpAlign 输出更多的对齐实例对，而在实际场景中，大部分对齐实例对例都具有等同的符号表达式。但是 Apex 无法在提高回调率的同时保证准确率。总体上说，BpAlign 的平均 **# F_1 score** 比 Apex 高 5%，在 DISHOWN 中高 20%。

> **回答 RQ1.** 由表 5.3可见，与现有对齐方法 Apex 相比，虽然牺牲了一些时间，但是 BpAlign 在准确度上更高。

RQ2. 不同的搜索空间对 BpAlign 的对齐结果有什么影响?

在3.4节所阐述的优化对齐算法中，BpAlign 通过多次迭代运行的方式来提高搜索空间，并且通过搜索距离阈值来限制依赖扩充的数量。迭代次数 *iter* 和



(a) 时间　　**(b)** Precision

(c) Recall　　**(d)** F_1 Score

图 5.2　不同搜索空间结果

搜索距离阈值 *threshold* 控制了 BpAlign 搜索空间的大小，在评估中，我们默认设置迭代次数为 3，搜索距离阈值为 2，并且通过设置不同数值产生了 5 个变体。如图 5.2所示，分别考虑迭代次数为 1，2，4 和搜索距离阈值为 0、1 的情况，我们将 Apex 作为基线方法。其中 x 轴表示算法题序号，根据 y 轴的指标从小到大排列。

在准确率和 F_1 Score 上，即使最小的搜索空间（变体 $iter_1$）也比基线方法 Apex 高，然而如前文所说，Apex 的回调率更高。在图 5.2a中，变体 $iter_4$ 花费了最多时间，变体 $iter_1$ 花费最少时间，BpAlign 的运行时间随着迭代次数指数级增加。我们不考虑变体 $iter_4$，是因为其花费大量时间但在准确度上提升不明显，在实际场景中效率低；不考虑变体 $threshold_1, iter_3$ 是因为与默认配置 $threshold_2, iter_3$ 相比，该变体扩充了更少的实例，从而导致回调率偏低（图 5.2c），并且在 F_1 Score 上不如默认配置表现好（图 5.2d）；对于其它变体，如 $iter_1$，可以被用于对时间要求高于准确度的现实场景中，因为其花费了最少的时间但准确度仍然高于基线方法。

表 5.4 概率推断规则结果

#	BpAlign	$-f_S$	$-f_{CD}$	$-f_{DD}$	$-f_{CD}-f_{DD}$	$-f_L$
1	0.99	0.87	**1**	0.99	**1**	0.99
2	**0.96**	0.71	**0.96**	**0.96**	**0.96**	**0.96**
3	**0.93**	0.82	0.9	0.92	0.9	0.92
4	**0.98**	0.84	**0.98**	**0.98**	**0.98**	0.96
5	**0.82**	0.75	0.79	0.81	0.79	0.81
6	**0.94**	0.87	0.93	0.91	0.91	0.92
7	**0.95**	0.89	**0.95**	**0.95**	**0.95**	0.93
8	**0.99**	0.97	**0.99**	0.91	0.91	0.91
9	**0.96**	0.81	**0.96**	**0.96**	**0.96**	**0.96**
10	**0.89**	0.78	**0.89**	0.87	0.86	**0.89**
平均值	**0.94**	0.83	0.93	0.92	0.93	0.93

回答 RQ2. 由图 5.2可见，越大的搜索空间越能提高对齐的准确度，但是花费更多时间。尽管如此，我们能够选取特定的搜索空间来兼顾准确度和时间。

RQ3. 不同组合的概率推断规则对 BpAlign 的对齐结果有什么影响?

在3.4.2节中，我们定义了 4 种概率推断规则，包括概率约束和对应的函数变量的定义。本部分通过探索不同概率规则的应用方式对准确度（F_1 Score）的影响来回答 **RQ3**。如表 5.4所示，我们基于 BpAlign 通过禁用一个或者多个推断规则衍生出了 5 种变体，分别不考虑：符号表达式和运行值（$-f_S$）、控制依赖的对齐信息（$-f_{CD}$）、数据依赖的对齐信息（$-f_{DD}$）、控制和数据依赖的对齐信息（$-f_{CD}-f_{DD}$）和循环结构信息（$-f_L$）。

$-f_S$ 的准确度最低，是因为符号表达式和运行值是驱动本方法对齐的最主要的特征。除了 1# 实例，与其它变体相比，BpAlign 的准确度最高。调查发现，BpAlign 将两个没有标注为对齐的语句输出为对齐，如下所示，BpAlign 将它们对齐，是因为它们分别依赖的两个实例为对齐实例对。

if(gcd(a[j],a[k])>max)	if((a[i]*a[j])/gcd(a[i],a[j])<min)

咨询标注者得知，这两个实例未被标注为对齐，是因为它们有不同的句法结构，然而实际上它们的语义相同，因此应当对齐。如果考虑这两个实例对齐，那么 BpAlign 在 1# 实例中具有最高的 F_1 Score。

$-f_{CD}$、$-f_{DD}$ 和 $-f_{CD}-f_{DD}$ 未将一些语句实例对齐，其中包括少数语义相同但是运行值不同或判断类型的语句实例，通过考虑依赖因素，BpAlign 能够

表 5.5 概率值取值对对齐结果的影响

	# 时间 (s)	# Precision	# Recall	# F1 Score
BpAlign	**2.91**	**0.97**	**0.91**	**0.94**
$H_{0.6}$	3.08	**0.97**	0.9	0.93
$H_{0.7}$	3.26	**0.97**	**0.91**	**0.94**
$H_{0.8}$	3.11	**0.97**	**0.91**	**0.94**
$H_{0.99}$	3.05	0.96	0.9	0.93
$P_{0.7}$	3.58	0.94	**0.91**	0.92
$P_{0.8}$	3.79	0.94	**0.91**	0.92
$P_{0.9}$	3.6	0.93	**0.91**	0.92
$P_{0.99}$	3.15	0.92	**0.91**	0.91

重新将他们对齐。$-f_L$ 未将一些语句实例对齐，语句片段如下，常用于算法题需要多个测试用例的情况，它们语义相同，但是运行值不同（分别为从 0 自增和从 t 自减）。

```
for (i = 0; i < t; i++)
```

```
while (t−−)
```

总体来说，相较于 5 个变体，BpAlign 在 10 道算法题的对齐中表现最优。

回答 RQ3. 由表 5.4可见，组合了 4 种概率推断规则的 BpAlign 在不同情况下能够达到最高的准确度。

RQ4. 函数变量中不同的概率值取值对 BpAlign 的对齐结果有什么影响?

在3.4.2节中,我们以 $H(HIGH) = 0.9$ 表示很有可能对齐,$P(POSITIVE) = 0.6$ 表示倾向于对齐,并且给相反项分别赋 0.1 和 0.4 表示对齐可能性很小和倾向于不对齐。本实验中通过修改 H 和 P（L 和 N 作相应修改）产生了 8 个 BpAlign 的变体，如表 5.5所示，其中 $H_{0.6}$ 表示将很可能对齐的概率值修改为 $H = 0.6$，每列分别展示了不同变体在标注数据集上所获得的时间、准确率、回调率和 F_1 Score 的平均值，加粗数值为该列指标中表现最优。

整体而言，不同概率值的设置对结果存在细微的影响，当 $H > 0.9$ 时，**# Precision** 和 **# Recall** 略有下降，当 $H < 0.9$ 时时间消耗略微增加，$H = 0.6$ 时回调率略微下降，说明对符号表达式相同的情况，需要赋予一个相对较高的概率值；当 $P >= 0.7$ 时，回调率不发生明显变化，但是准确率出现下降，说明在考虑依赖信息时，需要一个高于无法确定（$U(UNCERTAIN) = 0.5$），但不能对

表 5.6 聚类选取的效果

#	聚类 + 选取		直接选取		TOP3
	相似度	时间	相似度	时间	准确度
1	0.89	0.52	0.98	5.57	0.33
2	1	0.01	1	0.28	1
3	0.92	0.27	0.92	1.48	1
4	0.87	0.01	0.94	0.08	1
5	0.82	0.04	0.82	0.06	1
6	0.91	0.18	0.99	1.52	0.67
7	1	0.02	1	0.85	1
8	1	0.01	1	0.94	1
9	0.89	0.06	0.89	0.35	1
10	1	0.04	1	2.44	1
平均值	0.93	0.116	0.954	1.357	0.9

最终的后验概率起决定性作用的概率值。如果与当前实例对存在依赖关系，且仅有一个实例对对齐，信念传播下 $P = 0.6$ 难以使得当前实例对的边缘概率超过阈值，而如果存在依赖关系的对齐实例对越多，其概率值则会越大，从而说明当前实例对对齐概率很高。

回答 RQ4. 由表 5.5可见，综合时间、准确率、回调率和 F_1 Score，BpAlign 的概率值设置达到了近似的局部最优解。

RQ5. 与不聚类相比，聚类后选取正确版本在效率上的提升如何?

在4.3节中，我们以聚类和选取的方式提高为错误代码选取相似正确版本的效率，本部分探索直接在正确数据集中进行聚类选取的效率。

表 5.6展示了通过聚类选取搜索正确版本的相似度（计算方式见公式4.1）和时间、直接在未聚类的正确提交数据集中选取的相似度和时间、TOP3 准确度。其中 TOP1 相似度指与错误代码相似度的最高值，TOP3 准确度指通过聚类选取，在前 3 个相似正确代码中选中的准确度，其中每个算法题按照标记对齐中的提交数量（表 5.2）取均值。

在相似度上，聚类选取在 7 个算法题上均与直接选取获得正确提交的相似度相同，而其余 3 个算法题相似度差值不超过 10%；在时间消耗上，通过先聚类再选取降低了每次评估学生代码时所需用时，最多降低至全部用时的 1%，平均降低耗时至 8.5%；在 TOP3 准确度上，8 个算法题均获得了 100% 的准确率，而其余 2 个算法题在相似度上与 TOP1 差距小，但是能够大幅降低时间消耗，平

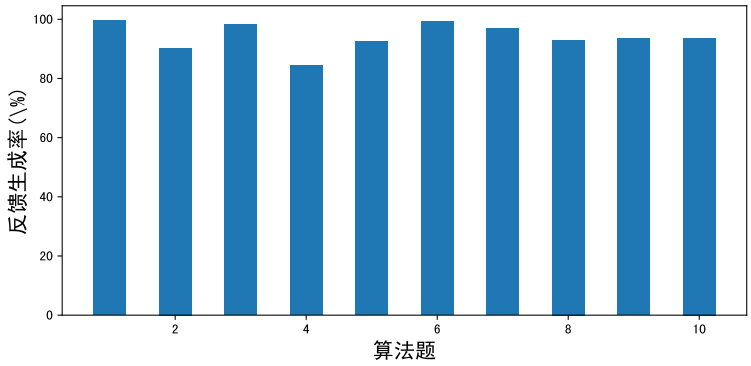


图 5.3　反馈生成率

均的准确度为 0.9。

> **回答 RQ5.** 由表 5.6可见，综合相似度、时间消耗和 TOP3 准确度，聚类后选取正确版本能够大幅提升寻找相似正确代码的效率。

RQ6. Bpex 在为错误代码生成反馈的生成率如何?

反馈生成率（反馈生成率 $=\frac{\text{反馈生成数量}}{\text{错误提交数量}}$）是学生代码自动反馈生成评价中常用的指标，其中分母表示当前问题的所有错误提交数量，分子表示工具成功生成反馈的提交数量。图 5.3展示了 Bpex 在表 5.1所示的数据集中运行的反馈生成率。Bpex 的平均反馈生成率为 95.45%，最高为 99.6%（ALEXTASK），最低为 84.44%（CHRL4）。经过检查发现，无法生成反馈的原因主要包括测试用例未触发错误和编译出错。

由于工具对算法题形式要求和语言存在差异，故未在此数据集中与 Clara 比较。为了与 Clara 比较，我们将表 5.2中的 28 对错误代码和正确版本转换为 Clara 支持的格式。经过运行，Clara 成功为其中的 8 个错误提交生成反馈信息，而其余 20 个错误提交生成失败。

> **回答 RQ6.** 由图 5.3可见，Bpex 具有高反馈生成率，在真实场景下能够为大多数错误学生代码生成反馈。

RQ7. Bpex 与现有反馈生成方法相比，对学生的帮助如何?

为了回答 **RQ7**，我们将 Bpex 与 Apex 和 Clara 进行比较。在相关领域还有一些研究我们没有进行比较，如 Sarfgen[15]，我们尝试过但无法获取源代码;

表 5.7 每组花费时间的平均值和标准差

#	Bpex	Clara	Apex	无
平均值	3715.5	3883	4521.25	5935
标准差	411.01	383.4	657.19	1477.42

表 5.8 学生反馈结果

	# Accuracy		# Comprehensibility	
	得分	p-value	得分	p-value
Bpex	4.16	-	4.25	-
Clara	3.44	0.0319	2.56	0.0175
Apex	2.81	0.0373	4.09	0.7462

Cafe[55]，其所面向的函数型语言 Ocaml 与 C 语言在实现方法存在明显区别，不具备可比性。

在 **RQ6** 中，我们将 Clara 在 28 个错误提交中运行，Clara 成功为其中 8 个提交生成反馈，我们利用这 8 个错误提交和 Clara、Apex 与 Bpex 的反馈信息进行了用户调研。我们邀请了 16 名 C 语言的初学者参与并将他们随机分为 4 组，每组包含 4 名学生。

首先，用户调研要求每组学生给定错误提交和反馈信息对错误提交进行修复（debug）。第一组学生使用 Bpex 的反馈 debug，第二组学生使用 Clara 的反馈 debug，第三组学生使用 Apex 的反馈 debug，第四组学生不使用辅助工具。如果算法题能够通过测试用例即为完成，我们为每组学生记录平均完成时间；然后，对于 8 个错误提交的每个反馈结果，前三组学生分别回答两个问题，每个问题的选项采用五点李克特量表[81]，得分 1、2、3、4、5 分别表示非常不同意、不同意、中立、同意和非常同意，这两个问题是：

准确性（Accuracy）：反馈信息能否准确地提示代码中的根源错误?

可理解性（Comprehensibility：反馈信息是否易于理解?

表 5.7展示了每组花费时间的平均值和标准差，其中 Bpex 平均花费 62 分钟，Clara 平均花费 65 分钟，Apex 平均花费 75 分钟，不使用辅助工具花费了 99 分钟。从时间上看，相比 Clara 和 Apex，Bpex 能帮助学生提高解题效率；从标准差上看，相比不使用辅助工具，辅助工具能降低学生编码用时的离散程度。

表 5.8展示了学生反馈结果，在两个问题上，Bpex 均获得比 Clara 和 Apex 更高的分数。对学生给出的反馈进行了 t 检验，Clara 与 Bpex 在**准确性**和**可理**

```
You should have done:
33^t_7 : if(b<min)
35^t_3 : min=b;
41^f_1 : printf("%lld\n",min);' you should have output 3.0, instead of 6.0.

When while(t--) at 19^f_1, you should not have done:
28^f_1 : min=a[0]*a[1];
```

(a) Bpex 反馈示例

```
* Change 'ret := ite(==(y, 0), x, FuncCall(gcd, y, %(x, y)))' to 'ret := ite(==(y, 0), x
    , FuncCall(gcd, y, %(x, y)))' at the beginning of the function 'gcd' (cost=1.0)
* Change 'min := *([](a, 0), [](a, 1))' to 'min := 1000000000' at line 14 (cost=7.0)
* Change 'lcm := /(*([](a, i), [](a, j)), FuncCall(gcd, [](a, i), [](a, j)))' to 'lcm :=
    /(*([](a, i), [](a, j)), FuncCall(gcd, [](a, i), [](a, j)))' at line 20 (cost=1.0)
* Change 'min := min' to 'min := ite(<(lcm', min), lcm', min)' inside the body of the '
    for' loop beginning at line 19 (cost=5.0)
```

(b) Clara 反馈示例

图 5.4 反馈示例

解性的 p 值分别为 0.0319 和 0.0175，置信区间为 95%，Apex 与 Bpex 在**准确性**上的 p 值为 0.0373（95% 的置信区间）。与 Clara 相比，Bpex 在**准确性**上取得更高的分数是因为 Bpex 提供的是因果链分析反馈，根源错误位于切片前段，在反馈输出中优先输出，而 Clara 提供的是修复反馈，反馈结果与错误代码和正确版本的不一致实现位置相关，根源错误可能出现在反馈中的任何位置，没有优先输出，而 Apex 准确性低是因为对齐的精确度低；Bpex 在**可理解性**上取得更高的分数是因为 Bpex 提供的因果链分析反馈与学生做算法题编译出错输出的 Traceback 相似，易于理解，如图 5.4a，而 Clara 的反馈为非自然语言，较 Bpex 更难理解，如图 5.4b；Apex 和 Bpex 在**可理解性**上具有相似的得分和高重合的置信区间，因为本文复现 Apex 采用的是同样的反馈生成算法（第四章）。

回答 RQ7. 由表 5.7和表 5.8可见，Bpex 帮助学生花费最少的时间 debug 并且在准确性和可理解性得分最高，说明与现有反馈生成方法相比，Bpex 能更有效地帮助学生理解代码错误。

5.4 案例讨论

为了更好地理解我们的方法和现有工作在对齐和反馈生成的问题，我们以图 3.1中的错误代码和正确版本为例进行分析。

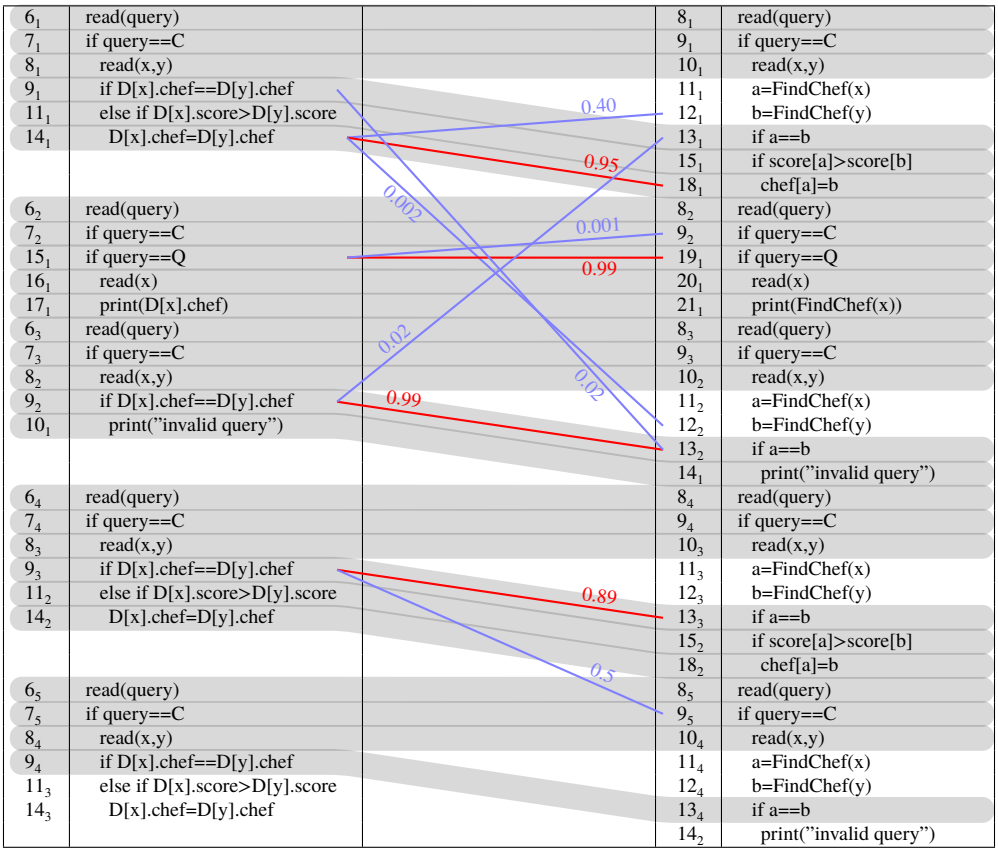


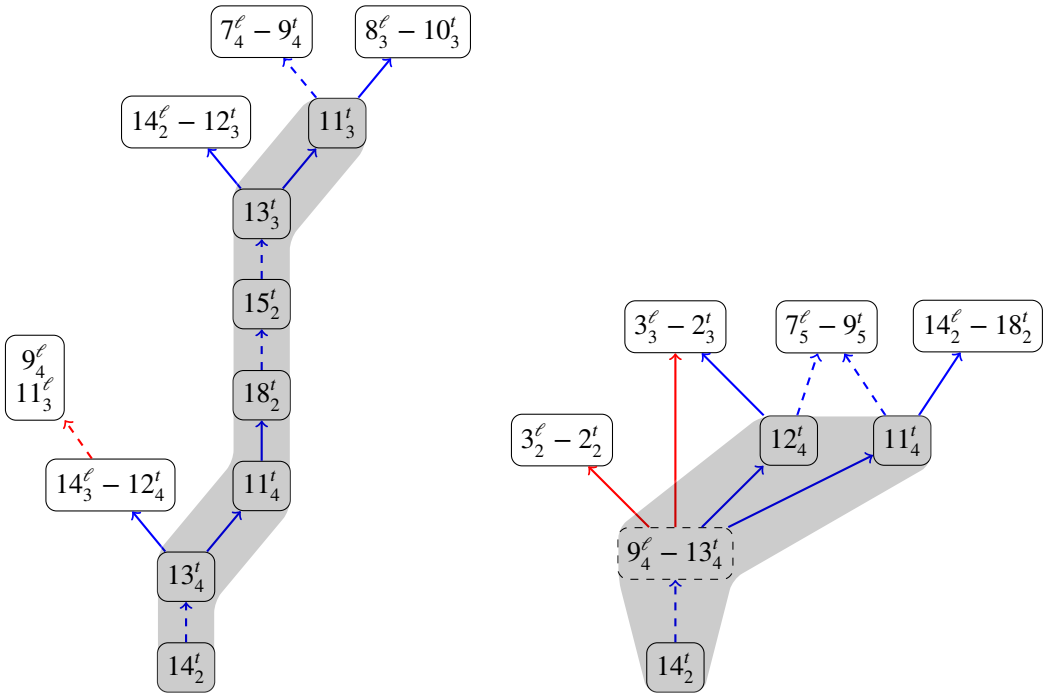
图 5.5　BpAlign 的对齐

5.4.1　自动对齐问题

在3.1节中，我们已经分析了 Apex 错误对齐的原因，并且提出通过概率推断的方法处理不确定性，图 5.5展示了 BpAlign 在输入 #2 到 #6 的动态跟踪信息对齐结果。图中的序列号和代码与图 3.3中一致，红色和紫色线条表示实例对和它们的对齐概率值，灰色格子表示最终的对齐结果。对于 14_1^{ℓ}，存在 12_1^{t}、18_1^{t} 和 12_2^{t} 三种候选对齐实例，由于 18_1^{t} 和 14_1^{ℓ} 的对齐概率最高，因此选取为最终对齐。其原因是 14_1^{ℓ} 和 18_1^{t} 都依赖于高对齐概率的实例对 9_1^{ℓ} 和 13_1^{t}，然而其它候选对齐实例不依赖。通过信念传播，14_1^{ℓ} 和 18_1^{t} 的对齐概率会传播到其它实例对，如 11_1^{ℓ} 和 15_1^{t}，它们被 14_1^{ℓ} 和 18_1^{t} 所依赖，本文利用依赖关系的方式见3.4节。对于其它实例，如 15_1^{ℓ} 和 9_2^{t}，由于它们的后验概率低，因此不纳入最终对齐结果。

5.4.2　反馈生成问题

反馈生成需要在动态切片对齐图（4.5节）的基础上进行，而反馈是否准确依赖于对齐算法的准确性。图 5.6a和图 5.6b展示了 Apex 和 Bpex 根据图 3.1中的



(a) Apex 动态切片对齐图　　**(b)** Bpex 动态切片对齐图

图 5.6　动态切片对齐图示例

错误代码和正确版本生成的动态切片对齐图片段。其中方块中为语句实例，包含两个语句实例的实线方块说明该语句实例对具有关系 R_A，虚线方块说明该语句实例对具有关系 R_U，实现表示数据依赖关系（D_{data}），虚线表示控制依赖关系（$D_{control}$），为了便于观察，我们用红色表示错误代码中的依赖关系，蓝色表示正确代码中的依赖关系。灰色方块表示被用于生成反馈的部分。

以图 5.6b为例，14^{t}_{2} 为关键实例，通过 14^{t}_{2}，我们获取其控制依赖实例 13^{t}_{4}，该实例与 9^{ℓ}_{4} 对齐但不匹配（关系 R_U），说明了错误提交未输出预期结果是因为判断“两个菜肴是否是同一个主人”时结果与正确版本不一致；从正确版本的数据依赖出发，12^{t}_{4} 和 11^{t}_{4} 分别输出 13^{t}_{4} 中的 a 和 b，在错误提交中，9^{ℓ}_{4} 中的 $D[x=2].chef$ 和 $D[y=3].chef$ 在“$3^{\ell}_{2}-2^{t}_{2}$”和“$3^{\ell}_{3}-2^{t}_{3}$”处被定义，这两个实例对为对齐实例对，从 12^{t}_{4} 的数据依赖出发到达“$3^{\ell}_{3}-2^{t}_{3}$”，从 11^{t}_{4} 的数据依赖出发到达“$14^{\ell}_{2}-18^{t}_{2}$”，12^{t}_{4} 和 11^{t}_{4} 共同控制依赖于 9^{t}_{5}，9^{t}_{5} 与 7^{ℓ}_{5} 构成对齐实例对。动态切片对齐图清晰地展示了两段执行之间的不一致，从中能够分析错误提交的原因，基于灰色方块部分的实例，我们生成反馈，如图 5.7b所示，其中方法 FindChef() 省略了函数中的具体执行实例，通过该反馈，学生能够发现缺失的函数和函数的实现方式示例。

```
When query==C at 7^ℓ_4, you should have done
11^t_3 : a = FindChef(x)
13^t_3 : if a != b
  15^t_2 : if score[a] > score[b]
    18^t_2 : chef[a] = b
11^t_4 : a = FindChef(x)
13^t_4 : if a == b
  14^t_2 : print("invalid query");
```

(a) Apex 错误分析

```
When query==C at 7^ℓ_5, you should have done
11^t_4 : a = FindChef(x);
12^t_4 : b = FindChef(y)
13^t_4 : if a == b
  14^t_2 : print("invalid query");
```

(b) Bpex 错误分析

图 5.7　错误分析示例

对于 Apex，在 14^{ℓ}_2, $x = 1$ 将 $D[x].chef$ 赋值为 3，而其本应该对齐的 18^{ℓ}_2, $chef[a = 2]$ 处将 $chef[a]$ 赋值为 3，然而 Apex 错误地将其与 12^{t}_3 对齐，从而 18^{ℓ}_2 被错误地分类为 C，进而导致 Apex 的动态切片对齐图错误，错误的动态切片对齐图导致了错误的反馈，如图 5.7a所示，虽然其中也包括了缺失函数示例，但是夹杂了错误学生代码中的正确实现语句（如 15^{t}_2），具有误导性。

5.5　效度威胁

本节对本章实验过程中的内部效度和外部效度进行分析。其中，内部效度指在完全相同的研究过程中复制研究结果的程度，是表示实验所提供的自变量与因变量之间因果关系明确程度的一种指标；外部效度是指在脱离研究情境后，研究结果还能成立的程度。

内部效度威胁： BpAlign 的动态分析和符号分析阶段沿用 Apex[6]的开源优化代码，其实际插桩结果是否完全准确和符号表达式的生成方式是否可靠可能潜在地影响我们结果的准确性。

在实验的数据标注阶段，由于数据集包含的错误提交数量大，对应跟踪中的实例数量多，我们无法对所有的错误代码和正确版本评估对齐的准确度，因此我们从中以算法为区分选取具有代表性的错误提交，并以序列对齐为指标对正确版本进行聚类并选取与错误代码相似的正确版本。然而是否使用不同算法依赖于主观判断，存在客观性缺失问题；基于序列对齐的选取方式可能因为序列对齐本身的精度问题，使得寻找到的正确版本不是最优解。

获取错误代码和正确版本的数据集后，我们邀请两名开发者进行人工对齐实例对的标注，并就不一致的实例对协商以达成一致。尽管如此，主观性可能潜

[6]http://apexpub.altervista.org

在地影响到我们的结果。

外部效度威胁：我们从 *CODECHEF* 收集了 5679 个 C 语言提交代码，取得了一定的实验效果。尽管 C 语言是大多数高校课堂教学的编程语言，但这并不能代表所有的编程语言，近些年其它语言，如 Python 和 Java 也被广泛用于高校课堂教学，我们将把我们的方法扩展到其它编程语言。

BPEX 关键实例提取阶段通过比较输出串和操作数是否一致收集关键实例，其可行性是建立在学生代码的输出格式统一，即与算法题规定格式相同。当学生提交代码中出现输出串拆分、字符错误等情况时，无法确定工具的有效性。

我们选取了 *CODECHEF* 的 10 道算法题作为评估数据集，虽然这 10 道算法题也被其它面向学生代码评分的工具所选取，如 AutoGrader[21]，但无法确保它们能够充分体现工具的有效性。

BPEX 生效的前提是具备测试用例能够触发学生代码的错误，在实验过程中选取的 28 对错误代码和正确版本均通过人工的方式获取满足条件的测试用例，而在现实场景中如果测试用例未触发错误，则无法让工具生效。

5.6　本章小结

为了验证 BPALIGN 和 BPEX 的有效性，本章进行了若干实验。收集了 *CODECHEF* 上 10 道算法题的 5679 个 C 语言提交代码，并且从中选取具有代表性的 28 对错误代码和正确版本形成数据集，随后复现了两个学生代码反馈生成方法。实验结果显示我们的工具在对齐、反馈生成等方面相对现有方法均有一定的提升。为了进一步探究方法中的搜索空间、概率推断规则和概率值选取的作用，我们对它们生成了一系列变体进行消融实验。研究结果及实验结果表明，结合概率推断技术能使得对齐和反馈生成效果显著提升。最后对实验过程的内部效度威胁和外部效度威胁进行了分析。

第六章　总结与展望

本章对全文的研究工作进行总结，并对当前工作的不足之处进行讨论，进而对未来工作进行展望。

6.1　工作总结

近年来，随着计算机专业招生扩大，高校课堂教学压力激增，越来越多的研究者参与到编程教学数字化转型中。针对学生代码自动生成反馈文档对于缓解教学者工作压力、辅助学生解决问题、提高教学效率具有重要的意义。本文就学生错误代码反馈信息不充分、现有工作对齐精度低和反馈生成质量低下问题，提出了基于概率推断的算法编程作业自动评价技术。

具体而言，本文的主要工作总结如下：

1. 收集了来自 *CODECHEF* 上 10 道算法题的 5679 个 C 语言提交代码，其中包括 3076 个错误代码，通过聚类算法和人工核查形成了 28 对错误代码和正确版本组成的数据集，进行人工标注，总共标注了 1689 个对齐实例对，构建的数据集用于支持重复实验。

2. 提出了一种基于概率推断的自动对齐算法并实现工具 BpAlign。通过程序分析技术提取错误代码和正确版本的跟踪信息，基于符号分析和序列对齐算法将其对齐并赋予初始概率，随后将代码信息包括依赖信息、符号信息、结构信息构建为概率图模型，并赋予先验概率，通过信念传播算法获得后验对齐概率，通过最终对齐算法获得对齐结果。实验结果显示我们方法的对齐准确度优于实验中复现的基于对齐的错误分析算法 Apex，具有灵活的搜索范围和有效的概率推断规则。

3. 提出了一种基于动态切片对齐图的反馈生成生成算法并实现工具 Bpex。通过聚类选取算法，为错误代码选取相似的正确版本，基于关键实例算法为错误代码和正确版本构建动态切片，随后将动态切片和 BpAlign 的对齐结果构建为动态切片对齐图，根据启发式规则生成反馈文档。实验结果显示我们的方法在反馈生成率和效果上优于实验中复现的基于修复的反馈生成算法 Clara，能更好地提高学生编程学习效率。

6.2 未来展望

本文的研究工作主要依赖于程序分析技术和概率推断技术，以 C 语言 OJ 算法题代码为主要研究场景，研究自动对齐技术和反馈生成技术。本文第五章针对设置的研究问题进行了严谨的实验和详尽的分析，但仍存在需要进一步完善和改进的地方。

(1) 数据集的选择：本文选择 C 语言为主要研究场景和数据集的组成成分，为了保证方法有效性和实验结论的可靠性，未来需要在更多开发语言背景下实施本文的方法步骤，对本文结论进行验证。

实验部分通过聚类的方式选取数据集，并进行实验数据标注，聚类以序列对齐为相似度度量指标，本文已经论证过序列对齐的精确度问题，未来考虑选取其他度量指标，比如抽象语法树相似性[82]、语义特征相似性[83]等。

(2) 方法的改进：在概率推断方面，本文提出了 4 种概率推断规则，并且实验说明了每种概率推断规则均在提升准确度上具有积极作用，未来考虑结合更多的程序信息，增加有效的推断规则；对于对齐概率的选择，虽然学术界常采用基于经验的先验概率值，但是不同的先验概率值必然会导致不同的结果，未来考虑利用数据集的结果，研究如何训练先验概率值，进一步提高对齐准确度；通过 py-factorgraph 解算因子图的时间开销仍然不可小觑，未来考虑利用更快速的概率推断工具，如 libDAI[84]，进一步降低算法的时间开销。

在关键实例提取阶段，我们的方法未覆盖所有可能的输出情况，如学生代码对输出串进行拆分、字符错误等。未来我们会完善并改进关键实例提取算法，提高方法的鲁棒性。

在反馈生成阶段，本文的反馈生成中根据正确代码的实现提供修复意见，对于正确代码中未对齐的语句实例未以错误代码的命名方式反馈，未来考虑利用静态分析对参数进行映射以提供更准确的修复意见并实现自动的错误修复。

此外，本文方法生效的前提是测试用例能够触发错误学生代码，未来我们考虑研究或集成测试用例工具，如 Pathgrind[37]，进一步完善方法的生态。

(3) 可拓展性：本文方法的面向对象为错误的学生代码，验证了概率推断方法在错误分析中的可行性，未来考虑将对齐算法和概率推断技术应用于大规模工程项目的错误定位、错误修复等领域。

参考文献

[1] Soper T. Analysis: the exploding demand for computer science education, and why America needs to keep up. Geekwire[Z]. 2014.

[2] Sentance S, Csizmadia A. Computing in the Curriculum: Challenges and Strategies from a Teacher's Perspective[J]. Education and Information Technologies, 2017, 22(2): 469-495.

[3] Gaebel M. MOOCs: Massive open online courses[M]. EUA Geneva, 2014.

[4] Hollingsworth J. Automatic Graders for Programming Classes[J]. Commun. ACM, 1960, 3(10): 528-529.

[5] Edwards S H, Perez-Quinones M A. Web-CAT: Automatically Grading Programming Assignments[C]//ITiCSE '08: Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education. Madrid, Spain: Association for Computing Machinery, 2008: 328.

[6] Higgins C, Hegazy T, Symeonidis P, et al. The CourseMarker CBA System: Improvements over Ceilidh[J]. Education and Information Technologies, 2003, 8(3): 287-304.

[7] Wang T, Su X, Ma P, et al. Ability-Training-Oriented Automated Assessment in Introductory Programming Course[J]. Comput. Educ., 2011, 56(1): 220-226.

[8] Queirós R A P, Leal J P. PETCHA: A Programming Exercises Teaching Assistant[C]//ITiCSE '12: Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education. Haifa, Israel: Association for Computing Machinery, 2012: 192-197.

[9] Beizer B. Black-box testing: techniques for functional testing of software and systems[M]. John Wiley & Sons, Inc., 1995.

[10] Tillmann N, De Halleux J, Xie T, et al. Teaching and Learning Programming and Software Engineering via Interactive Gaming[C]//ICSE '13: Proceedings of the 2013 International Conference on Software Engineering. San Francisco, CA, USA: IEEE Press, 2013: 1117-1126.

[11] Adam A, Laurent J P. LAURA, a system to debug student programs[J]. Artificial Intelligence, 1980, 15(1): 75-122.

[12] Tillmann N, de Halleux J. Pex–White Box Test Generation for .NET[C]// Beckert B, Hähnle R. Tests and Proofs. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008: 134-153.

[13] Singh R, Gulwani S, Solar-Lezama A. Automated Feedback Generation for Introductory Programming Assignments[C]//PLDI '13: Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation. Seattle, Washington, USA: Association for Computing Machinery, 2013: 15-26.

[14] Bhatia S, Kohli P, Singh R. Neuro-Symbolic Program Corrector for Introductory Programming Assignments[C]//ICSE '18: Proceedings of the 40th International Conference on Software Engineering. Gothenburg, Sweden: Association for Computing Machinery, 2018: 60-70.

[15] Wang K, Singh R, Su Z. Search, Align, and Repair: Data-Driven Feedback Generation for Introductory Programming Exercises[C]//PLDI 2018: Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation. Philadelphia, PA, USA: Association for Computing Machinery, 2018: 481-495.

[16] Gulwani S, Radiček I, Zuleger F. Automated Clustering and Program Repair for Introductory Programming Assignments[C]//PLDI 2018: Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation. Philadelphia, PA, USA: Association for Computing Machinery, 2018: 465-480.

[17] Pu Y, Narasimhan K, Solar-Lezama A, et al. Sk_p: A Neural Program Corrector for MOOCs[C]//SPLASH Companion 2016: Companion Proceedings of the 2016 ACM SIGPLAN International Conference on Systems, Programming, Languages and Applications: Software for Humanity. Amsterdam, Netherlands: Association for Computing Machinery, 2016: 39-40.

[18] Yi J, Ahmed U Z, Karkare A, et al. A Feasibility Study of Using Automated Program Repair for Introductory Programming Assignments[C]//ESEC/FSE 2017: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. Paderborn, Germany: Association for Computing Machinery, 2017: 740-751.

[19] Drummond A, Lu Y, Chaudhuri S, et al. Learning to Grade Student Programs in a Massive Open Online Course[C]//ICDM '14: Proceedings of the 2014 IEEE International Conference on Data Mining. USA: IEEE Computer Society, 2014: 785-790.

[20] Gulwani S, Radiček I, Zuleger F. Feedback Generation for Performance Problems in Introductory Programming Assignments[C]//FSE 2014: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. Hong Kong, China: Association for Computing Machinery, 2014: 41-51.

[21] Liu X, Wang S, Wang P, et al. Automatic Grading of Programming Assignments: An Approach Based on Formal Semantics[C]//ICSE-SEET '19: Proceedings of the 41st International Conference on Software Engineering: Software Engineering Education and Training. Montreal, Quebec, Canada: IEEE Press, 2019: 126-137.

[22] Ahmed U Z, Srivastava N, Sindhgatta R, et al. Characterizing the Pedagogical Benefits of Adaptive Feedback for Compilation Errors by Novice Programmers[C]//ICSE-SEET '20: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering Education

and Training. Seoul, South Korea: Association for Computing Machinery, 2020: 139-150.

[23] Kim D, Kwon Y, Liu P, et al. Apex: Automatic Programming Assignment Error Explanation[J]. SIGPLAN Not., 2016, 51(10): 311-327.

[24] Cooper G F. The computational complexity of probabilistic inference using Bayesian belief networks[J]. Artificial intelligence, 1990, 42(2-3): 393-405.

[25] Kschischang F R, Frey B J, Loeliger H. Factor Graphs and the Sum-Product Algorithm[J]. IEEE Trans. Inf. Theor., 2006, 47(2): 498-519.

[26] Loeliger H A. An introduction to factor graphs[J]. IEEE Signal Processing Magazine, 2004, 21(1): 28-41.

[27] Pearl J. Reverend Bayes on Inference Engines: A Distributed Hierarchical Approach[C]//AAAI'82: Proceedings of the Second AAAI Conference on Artificial Intelligence. Pittsburgh, Pennsylvania: AAAI Press, 1982: 133-136.

[28] Banerjee A, Roychoudhury A, Harlie J A, et al. Golden Implementation Driven Software Debugging[C]//FSE '10: Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering. Santa Fe, New Mexico, USA: Association for Computing Machinery, 2010: 177-186.

[29] Lahiri S K, Sinha R, Hawblitzel C. Automatic Rootcausing for Program Equivalence Failures in Binaries[C]//Kroening D, Păsăreanu C S. Computer Aided Verification. Cham: Springer International Publishing, 2015: 362-379.

[30] Lahiri S K, Hawblitzel C, Kawaguchi M, et al. SYMDIFF: A Language-Agnostic Semantic Diff Tool for Imperative Programs[C]//Madhusudan P, Seshia S A. Computer Aided Verification. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012: 712-717.

[31] King J C. Symbolic Execution and Program Testing[J]. Commun. ACM, 1976, 19(7): 385-394.

[32] Needleman S B, Wunsch C D. A general method applicable to the search for similarities in the amino acid sequence of two proteins[J]. Journal of Molecular Biology, 1970, 48(3): 443-453.

[33] Marin V, Pereira T, Sridharan S, et al. Automated Personalized Feedback in Introductory Java Programming MOOCs[C]//. 2017: 1259-1270.

[34] Ferrante J, Ottenstein K J, Warren J D. The Program Dependence Graph and Its Use in Optimization[J]. ACM Trans. Program. Lang. Syst., 1987, 9(3): 319-349.

[35] Lee J, Han W S, Kasperovics R, et al. An In-Depth Comparison of Subgraph Isomorphism Algorithms in Graph Databases[J]. Proc. VLDB Endow., 2012, 6(2): 133-144.

[36] Suzuki R, Soares G, Head A, et al. TraceDiff: Debugging unexpected code behavior using trace divergences[C]//2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). 2017: 107-115.

[37] Sharma A. Exploiting Undefined Behaviors for Efficient Symbolic Execution[C]//ICSE Companion 2014: Companion Proceedings of the 36th International Conference on Software Engineering. Hyderabad, India: Association for Computing Machinery, 2014: 727-729.

[38] Adriaans P, Zantinge D. Data mining[M]. Addison-Wesley Longman Publishing Co., Inc., 1997.

[39] Ehrlinger L, Wöß W. Towards a definition of knowledge graphs.[J]. SEMANTiCS (Posters, Demos, SuCCESS), 2016, 48(1-4): 2.

[40] 机器学习[M]. 清华大学出版社, 2016.

[41] Head A, Glassman E, Soares G, et al. Writing Reusable Code Feedback at Scale with Mixed-Initiative Program Synthesis[C]//L@S '17: Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale. Cambridge, Massachusetts, USA: Association for Computing Machinery, 2017: 89-98.

[42] Piech C, Huang J, Nguyen A, et al. Learning Program Embeddings to Propagate Feedback on Student Code[C]//ICML'15: Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37. Lille, France: JMLR.org, 2015: 1093-1102.

[43] Kaleeswaran S, Santhiar A, Kanade A, et al. Semi-Supervised Verified Feedback Generation[C]//FSE 2016: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. Seattle, WA, USA: Association for Computing Machinery, 2016: 739-750.

[44] Milligan G W, Cooper M C. Methodology review: Clustering methods[J]. Applied psychological measurement, 1987, 11(4): 329-354.

[45] Lee J, Song D, So S, et al. Automatic Diagnosis and Correction of Logical Errors for Functional Programming Assignments[J]. Proc. ACM Program. Lang., 2018, 2(OOPSLA).

[46] Feser J K, Chaudhuri S, Dillig I. Synthesizing Data Structure Transformations from Input-Output Examples[C]//PLDI '15: Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation. Portland, OR, USA: Association for Computing Machinery, 2015: 229-239.

[47] Osera P M, Zdancewic S. Type-and-Example-Directed Program Synthesis[J]. SIGPLAN Not., 2015, 50(6): 619-630.

[48] Polikarpova N, Kuraj I, Solar-Lezama A. Program Synthesis from Polymorphic Refinement Types[J]. SIGPLAN Not., 2016, 51(6): 522-538.

[49] Song D, Lee M, Oh H. Automatic and Scalable Detection of Logical Errors in Functional Programming Assignments[J]. Proc. ACM Program. Lang., 2019, 3(OOPSLA).

[50] Feser J K, Chaudhuri S, Dillig I. Synthesizing Data Structure Transformations from Input-Output Examples[J]. SIGPLAN Not., 2015, 50(6): 229-239.

[51] Lee W, Heo K, Alur R, et al. Accelerating Search-Based Program Synthesis Using Learned Probabilistic Models[J]. SIGPLAN Not., 2018, 53(4): 436-449.

[52] Rolim R, Soares G, D'Antoni L, et al. Learning Syntactic Program Transformations from Examples[C]//ICSE '17: Proceedings of the 39th International Conference on Software Engineering. Buenos Aires, Argentina: IEEE Press, 2017: 404-415.

[53] Gulwani S, Hernández-Orallo J, Kitzelmann E, et al. Inductive Programming Meets the Real World[J]. Commun. ACM, 2015, 58(11): 90-99.

[54] Polozov O, Gulwani S. FlashMeta: A Framework for Inductive Program Synthesis[J]. SIGPLAN Not., 2015, 50(10): 107-126.

[55] Song D, Lee W, Oh H. Context-Aware and Data-Driven Feedback Generation for Programming Assignments[M]//Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. New York, NY, USA: Association for Computing Machinery, 2021: 328-340.

[56] Hu Y, Ahmed U Z, Mechtaev S, et al. Re-Factoring Based Program Repair Applied to Programming Assignments[C]//ASE '19: Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering. San Diego, California: IEEE Press, 2019: 388-398.

[57] Marin V J, Contractor M R, Rivero C R. Flexible Program Alignment to Deliver Data-Driven Feedback to Novice Programmers[C]//Cristea A I, Troussas C. Intelligent Tutoring Systems. Cham: Springer International Publishing, 2021: 247-258.

[58] Gupta R, Pal S, Kanade A, et al. DeepFix: Fixing Common C Language Errors by Deep Learning[C]//AAAI'17: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence. San Francisco, California, USA: AAAI Press, 2017: 1345-1351.

[59] Bahdanau D, Cho K, Bengio Y. Neural Machine Translation by Jointly Learning to Align and Translate[J]. CoRR, 2015, abs/1409.0473.

[60] Zaremba W, Sutskever I, Vinyals O. Recurrent Neural Network Regularization[EB/OL]. 2014. https://arxiv.org/abs/1409.2329.

[61] Ahmed U Z, Kumar P, Karkare A, et al. Compilation Error Repair: For the Student Programs, from the Student Programs[C]//ICSE-SEET '18: Proceedings of the 40th International Conference on Software Engineering: Software Engineering Education and Training. Gothenburg, Sweden: Association for Computing Machinery, 2018: 78-87.

[62] Šošić M, Šikić M. Edlib: a C/C++ library for fast, exact sequence alignment using edit distance[J]. Bioinformatics, 2017, 33(9): 1394-1395.

[63] Wang Z, Xu L. Grading Programs Based on Hybrid Analysis[C]//International Conference on Web Information Systems and Applications. 2019: 626-637.

[64] Clune J, Ramamurthy V, Martins R, et al. Program Equivalence for Assisted Grading of Functional Programs[J]. Proc. ACM Program. Lang., 2020, 4(OOPSLA).

[65] 李必信. 程序切片技术及其应用[M]. 北京: 科学出版社, 2006.

[66] Allen F E. Control Flow Analysis[C]//Proceedings of a Symposium on Compiler Optimization. Urbana-Champaign, Illinois: Association for Computing Machinery, 1970: 1-19.

[67] Huang J. Program Instrumentation and Software Testing[J]. Computer, 1978, 11(4): 25-32.

[68] Barrett C, Tinelli C. Satisfiability modulo theories[G]//Handbook of model checking. Springer, 2018: 305-343.

[69] Koller D, Friedman N. Probabilistic graphical models: principles and techniques[M]. 2009.

[70] Jensen F V, Nielsen T D. Bayesian networks and decision graphs[M]. Springer, 2007.

[71] Lattner C, Adve V. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation[C]//CGO '04: Proceedings of the International Symposium on Code Generation and Optimization: Feedback-Directed and Runtime Optimization. Palo Alto, California: IEEE Computer Society, 2004: 75.

[72] Ankan A, Panda A. Pgmpy: Probabilistic graphical models using python[C]//Proceedings of the 14th Python in Science Conference (SCIPY 2015). 2015.

[73] De Moura L, Bjørner N. Z3: An Efficient SMT Solver[C]//TACAS'08/ETAPS'08: Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Budapest, Hungary: Springer-Verlag, 2008: 337-340.

[74] Tsamardinos I, Brown L E, Aliferis C F. The max-min hill-climbing Bayesian network structure learning algorithm[J]. Machine learning, 2006, 65(1): 31-78.

[75] Aster R C, Borchers B, Thurber C H. Parameter estimation and inverse problems[M]. Elsevier, 2018.

[76] Breslow N E, Clayton D G. Approximate inference in generalized linear mixed models[J]. Journal of the American statistical Association, 1993, 88(421): 9-25.

[77] Agresti A. A survey of exact inference for contingency tables[J]. Statistical science, 1992, 7(1): 131-153.

[78] Needleman S B, Wunsch C D. A general method applicable to the search for similarities in the amino acid sequence of two proteins[J]. Journal of Molecular Biology, 1970, 48(3): 443-453.

[79] Beckman N E, Nori A V. Probabilistic, Modular and Scalable Inference of Typestate Specifications[J]. SIGPLAN Not., 2011, 46(6): 211-221.

[80] Fu Y, Osei-Owusu J, Astorga A, et al. PaCon: A Symbolic Analysis Approach for Tactic-Oriented Clustering of Programming Submissions[C]//SPLASH-E 2021: Proceedings of the 2021 ACM SIGPLAN International Symposium on SPLASH-E. Chicago, IL, USA: Association for Computing Machinery, 2021: 32-42.

[81] Likert R. A technique for the measurement of attitudes.[J]. Archives of psychology, 1932.

[82] Jiang L, Misherghi G, Su Z, et al. DECKARD: Scalable and Accurate Tree-Based Detection of Code Clones[C]//ICSE '07: Proceedings of the 29th International Conference on Software Engineering. USA: IEEE Computer Society, 2007: 96-105.

[83] Perry D M, Kim D, Samanta R, et al. SemCluster: Clustering of Imperative Programming Assignments Based on Quantitative Semantic Features[C]//PLDI 2019: Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation. Phoenix, AZ, USA: ACM, 2019: 860-873.

[84] Mooij J M. LibDAI: A Free and Open Source C++ Library for Discrete Approximate Inference in Graphical Models[J]. Journal of Machine Learning Research, 2010, 11: 2169-2173.

致　　谢

感谢我的导师许蕾老师。在研究生的三年里，许老师在学术上给予我精心的指导，在生活上给予我细致的关怀。科研中，许老师在疲惫时给予我精神上的鼓励，在怠慢时及时地给予我鞭策，在论文的选题、撰写、实验和修改等方面均倾注了大量的心血；项目中，给予我提升自己实际动手能力和合作能力的机会，让我在象牙塔中就能学习到工业界的运作方式，并且偶尔的出差机会让我能够领略到重庆郭家沱、内蒙古阿拉善、山东青岛等地的风土人情和自然风光，终身难忘；生活中，许老师和蔼可亲，平易近人，在枯燥的科研生活里给我带来了色彩斑斓的注脚，让我领略到南京的美景和美食。在疫情时，许老师不忘我独自在异乡的孤独，给我带来了亲手制作的甜点和小吃。能够师从许老师，是我莫大的荣幸，谨在此向许蕾老师表示诚挚的敬意和感谢。

感谢徐宝文教授和软件质量研究所的所有老师。在他们带领下的软件质量研究所为我提供了优越的科研平台和学术环境，感谢周毓明老师、陈林老师、李言辉老师在这三年里的学习生活中对我的帮助和教导，在学术上的交流。

感谢软件质量研究所的同学。感谢何欣程师姐、张强师兄、周航师兄和陶英师兄对我的科研项目进行交流与探讨，感谢林君宇同学在项目和科研中给予我的支持和合作，感谢我的同学周慧聪和王洋在我的学习和生活中给予的帮助与陪伴，也要感谢刘笑今、朱泓全、周敬尧和杨钧尹等师弟师妹们给我们实验室带来的朝气。

感谢我在微软实习阶段给我过我帮助的同事，mentor 陈文博、manager 陈硕还有薛源、郝潇正、刘晟昊等，感谢你们给我一个充满技术氛围又轻松愉悦的实习体验，也感谢你们对我的支持与肯定。

感谢曹迎春老师、尹存燕老师、李淑环老师、张定老师、方言老师在日常学习生活中给我们提供的帮助和付出。

感谢 NJU Linux User Group 提供的模板，让我轻松应对论文的格式与排版。

最后我要感谢我的女朋友、父母和家人们在我的学习生涯中为我提供的无微不至的关怀和帮助，支持我渡过无数难关从学校走向社会。在此，谨对各位无私的帮助表示衷心的感谢，未来我会继续拼搏，向全新的人生迈进！

科研成果

成果 A　攻读硕士学位期间申请的专利

[1] 许蕾，王智楷，刘翔宇，田晓滨，王瀚霖，颜同路，一种基于概率对齐的代码错误定位方法。专利申请号：202010576945.3，申请时间：2020 年 06 月 23 日

成果 B　攻读硕士学位期间发表的论文

[1] Zhikai Wang, Lei Xu. Grading Programs Based on Hybrid Analysis. WISA 2019. LNCS 11817. Springer, Cham. https://doi.org/10.1007/978-3-030-30952-7_63

[2] Xintang Lin, Haibo Zhang, Hui Xia, Liangjiang Yu, Xiangyan Fang, Xuan Chen, Zhikai Wang. Test Case Minimization for Regression Testing of Composite Service Based on Modification Impact Analysis. WISA 2020. LNCS 12432. Springer, Cham. https://doi.org/10.1007/978-3-030-60029-7_2

成果 C　攻读硕士学位期间参加的项目

[1] 国家自然科学基金重点项目，面向安全攸关深度学习系统的软件测试技术（61832009），2019.01-2023.12

[2] 横向项目，服务化软件测试用例生成方法论证，2020.01-2020.12