软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, [doi: 10.13328/j.cnki.jos.000000]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于频繁模式挖掘的 GCC 编译时能耗演化优化算法(智能化软件新技术专刊)

倪友聪[1,4], 吴瑞[1], 杜欣[1,4], 叶鹏[2], 李汪彪[3],肖如良[1]

[1](福建师范大学 数学与信息学院,福建 福州 350117)

[2](武汉纺织大学 数学与计算机学院,湖北 武汉 430200)

[3](福建师范大学 光电与信息工程学院, 福建 福州 350117)

[4](福建省公共服务大数据挖掘与应用工程技术研究中心, 福建 福州 350117)

通讯作者: 杜欣, E-mail: xindu79@126.com; 叶鹏,E-mail: whuyp@126.com

摘 要: 演化算法通过搜寻 GCC 编译器最优编译选项集对可执行代码的能耗进行改进,以达到编译时优化嵌入式软件能耗的目的.但这类算法未考虑多个编译选项之间可能存在相互影响,导致了其解质量不高且收敛速度慢的问题.针对这一不足,本文设计了一种基于频繁模式挖掘的遗传算法 GA-FP. 该算法在演化过程中利用频繁模式挖掘得到出现频度高且能耗改进大的一组编译选项,并以此作为启发式信息设计了"增添"和"删减"两种变异算子帮助提高解质量和加快收敛速度.通过与 Tree-EDA 算法在 5 个不同领域的 8 个典型案例下进行对比实验,结果表明:本文的 GA-FP 算法不仅能更有效地降低软件能耗(平均降低 2.5%,最高降低 21.1%),而且还能在获得不劣于 Tree-EDA 能耗优化效果的前提下更快地收敛(平均加快 34.5%,最高加快 83.3%),最优解中编译选项的相关性分析进一步验证了所设计变异算子的有效性.

关键词: 软件能耗; 编译优化; 嵌入式软件; 演化算法

An evolutionary algorithm for optimization of energy consumption at GCC compile time based on frequent pattern mining

Abstract: The evolutionary algorithms have been used to improve the energy consumption of executable code of embedded software by searching the optimal compilation options of GCC compiler. However, such algorithms do not consider the possible interaction between multiple compilation options so that the quality of their solutions is not high, and their convergence speed is slow. To solve this problem, this paper designs an evolutionary algorithm based on frequent pattern mining, called GA-FP. In the process of evolution, GA-FP uses frequent pattern mining to obtain a set of compilation options which are of high-frequency and contribute to significant improvement on energy consumption. The derived options are used as the heuristic information and two mutation operators of ADD and DELETE are designed to increase the quality of solution and accelerate the convergence speed. The comparative experiments are done on 8 typical cases in 5 different fields between Tree-EDA and GA-FP. The experimental results indicate that the GA-FP can not only reduce the energy consumption of software more effectively (the average and maximal reduction ratios are 2.5% and 21.1% respectively), but also converge faster (the average of 34.5% faster and up to 83.3% faster) when the energy optimization effect obtained by GA-FP is no less than that of Tree-EDA. The correlation analysis of compilation options in the optimal solution further validates the effectiveness of the designed mutation operators.

Key words: Software Energy; Compilation Optimization; Embedded Software; Evolutionary Algorithm

能耗是嵌入式软件的关键质量属性. 特别是在电量受限的执行环境中,降低嵌入式软件的能耗具有更为重要的价值和意义[1].与嵌入式软件源代码级的能耗优化相比,编译时能耗优化具有无需改动源代码,并且可保证功能语义一致性的优点. 作为一款开源编译器,GCC[2]已广泛应用于嵌入式软件源代码的编译. GCC 提供常用的几种优化等级,利用每种优化等级所预设的一组编译选项对软件源代码进行编译,能实现可执行代

基金项目:福建省新世纪优秀人才项目(2017 年); 福建省自然科学基金项目(2015J01235, 2017J01498); 福建省教育厅 JK 类项目(JK2015006); 湖北省自然科学基金项目 (2018CFB689)

码的质量优化.然而,对于特定的软件源代码、特定的执行平台和特定的优化目标,GCC 的优化等级往往难以获得最佳的优化效果.此外,GCC 编译选项数目众多,选择空间十分庞大.例如:GCC4.9.2 提供了 188 个编译选项,其选择空间高达 2^{188}.依靠程序员人工选择编译选项不仅十分困难,而且也难以保证优化质量.更为重要的是,GCC 提供的优化等级多集中于执行时间和目标代码大小的优化,而未针对能耗优化的场景. Pallister 的研究成果[3] 已表明使用 GCC 的某些优化等级对嵌入式软件进行编译时,甚至出现能耗增加的情况. 近年来,用于能耗优化的 GCC 编译选项的选择问题已经成为了一个研究热点[4].基于 Hoste 等人[10]提出的 58 个常用于能耗优化的编译选项,已涌现出基于统计的方法、机器学习方法和演化算法三类主要的优化方法[5].

基于统计的方法[6] 运用 Mann-Whitney 测试为特定领域嵌入式软件确定一组有改进效果的编译选项.具体地,首先将一组预设的编译选项应用于多个同类型嵌入式软件,考察在这组选项中去掉某一编译选项后的能耗变化情况, 再根据 Mann-Whitney 测试的结果判断去掉该编译选项前后是否存在显著的差异,从而确定该编译选项是否对能耗有显著影响. 最后经过多组统计实验可找出一组对某一类型嵌入式软件有能耗改进效果的编译选项. 由于 GCC 编译选项众多并且它们之间还存在复杂的影响关系,而基于统计的方法一次仅考查一个选项的模式难以最终获取对能耗改进效果最佳的编译选项集合.

机器学习方法[7-9] 先通过对训练样本集使用机器学习算法训练得到模型,再利用模型预测与训练样本集同属一类的嵌入式软件的最优编译选项集合.训练样本集中的每个样本以某一嵌入式软件作为输入,并使用迭代编译的方法找到的最优编译选项集合作为输出.同类型的多个嵌入式软件及它们的最优编译选项集合组成了机器学习方法的训练样本集. 然而,一些研究工作[10-11]已经实证了迭代编译方法不能在庞大的空间中找到最优编译选项集合.受制于训练样本自身的质量,使得机器学习方法得到的模型难以准确预测出真正的最优编译选项集合.

基于演化算法的方法[10-14] 将编译时能耗优化问题抽象成一个编译选项选择优化问题,并针对特定的嵌入式软件和特定的执行平台搜索更大的编译选项选择空间,为进一步降低能耗提供了有力支持.Hoste 等人运用传统遗传算法(Genetic Algorithm,GA) [10-11]获取了较迭代编译和编译器预设的最高优化等级更好的编译选项集合.为了进一步提高解质量和加快算法收敛速度,Lin、Nagiub 和 Garciarena 又提出了一些新的演化算法. Lin [12]设计了一种基因加权的遗传算法. Lin 的算法通过对上一代种群中适应度值高的个体所选择的编译选项加权以影响变异概率,从而加快了收敛速度. Nagiub [13]通过设计新的 poss-over 算子,将上一代种群中适应度优的个体直接加入到下一代种群,利用保留优势解的策略进一步加快了算法的收敛速度.但 Lin 和 Nagiub 的算法不仅存在过早收敛、易陷入局部最优的问题, 而且也未考虑编译选项之间存在相互影响的情况.为了解决这些不足, Garciarena[14]提出了双变量相关分布评估算法 (Tree estimation of distribution algorithm, Tree-EDA). Tree-EDA 算法考虑了任意两个编译选项之间可能存在的相互影响, 并在多个案例下与 GA 和单变量分布评估算法(Univariate Marginal Distribution Algorithm,UMDA)[15]进行了对比实验, 实验结果表明 Tree-EDA 算法能获取较 GA 和 UMDA 更好的解质量和收敛速度. 然而 Tree-EDA 仅考虑了任意两个编译选项之间的相互影响,而未涉及多个选项之间可能存在的相互作用.

为了探究多个编译选项之间的相互作用对解质量和收敛速度的影响, 本文提出了一种基于频繁模式挖掘(Frequent Pattern,FP)的遗传算法(GA) 并将其命名为 GA-FP. GA-FP 算法记录演化过程中有能耗改进效果的个体，并建立带能耗标注的事务表.GA-FP 算法利用带能耗标注的事务表并通过扩展传统的频繁模式树挖掘算法,挖掘出现频度高且能耗改进大的一组编译选项并将其作为启发式信息,引入了“增添”和“删减”两种新的变异算子帮助提高解质量和加快收敛速度.通过与 Tree-EDA 在 5 个不同领域的 8 个典型案例下实验对比的结果表明:本文的 GA-FP 算法不仅能更有效降低软件能耗(平均降低 2.5%,最高降低 21.1%),而且还能在获得不劣于 Tree-EDA 能耗优化效果的前提下更快地收敛(平均加快 34.5%,最高加快 83.3%);最优解中编译选项的相关性分析进一步验证了所设计变异算子的有效性.

本文的组织结构如下:第 1 节给出问题描述;第 2 节阐述 GA-FP 算法的总体流程;第 3 节详细介绍了挖掘带能耗改进标注的频繁编译模式集;第 4 节提出了“增添”和“删减”两个变异算子; 第 5 节给出案例研究及实验结果;第 6 节总结全文并给出未来工作.

1 问题描述

下面给出用于嵌入式软件能耗优化的 GCC 编译选项选择问题的形式化描述.

定义 1 (优化空间 Ω) 若对 GCC 编译器支持的编译选项从 1 至 l 进行编号,则优化空间 Ω 可定义为式(1)所示的 l 元序偶集. 在式(1)中, $x_i = 0$ 或 1 分别表示选用或不选用编译选项 i.

$$\Omega = \{ X \mid X = < x_1, x_2, \cdots, x_i \cdots, x_l > \wedge \ x_i \in \{0,1\} \} \tag{1}$$

定义 2 (选用和未选用的编译选项集) 对于优化空间 Ω 中一个元素 X,其所定义的选用和未选用的编译选项集分别用 $S_S(X)$ 和 $S_U(X)$ 表示.它们分别由式(2)和式(3)定义.

$$S_S(X) = \{ i \mid x_i \in X \ \wedge x_i = 1 \} \tag{2}$$

$$S_U(X) = \{ i \mid x_i \in X \ \wedge x_i = 0 \} \tag{3}$$

定义 3 (能耗评估) 能耗评估函数 $EvE(Sft_{exe})$用于计算一个嵌入式软件可执行代码 Sft_{exe} 在某一特定输入下,从开始运行至结束所消耗的电能. $EvE(Sft_{exe})$的定义如式(4)所示.

$$EvE(Sft_{exe}) = \int_{T_0}^{T_n} P(t)dt \approx \frac{1}{2}\sum_{j=0}^{n-1}(T_{j+1} - T_j) \times (P_j + P_{j+1}) \tag{4}$$

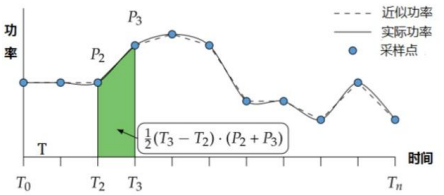

Fig.1 Energy consumption calculation
图 1 能耗计算示意图

在式(4)中, T_j 和 T_{j+1} 分别表示 Sft_{exe} 在特定输入下,执行过程中的第 j 和 j+1 功率测量的采样点; T_0 和 T_n 分别为 Sft_{exe} 执行的开始时刻和结束时刻; P_j 和 P_{j+1} 分别表示第 j 和 j+1 采样点测得的瞬时功率.通过累加相邻两个采样时刻点和对应功率所形成的梯形的面积, 可计算 Sft_{exe} 执行过程中实际能耗的近似值.基于 STM32F4 开发板,我们研发了一套能耗评估系统,实现了 $EvE(Sft_{exe})$函数的功能.

定义 4 (编译和链接) 定义函数 $cmpLnk_o(Sft_{\text{src}})$和函数 $cmpLnk(Sft_{src}, X)$分别表示运用 GCC 编译器缺省的-O0 等级(不选用任何编译选项)、 $S_S(X)$ 选用的编译选项集,对一个嵌入式软件源代码 Sft_{src} 进行编译和链接后所得到的可执行代码.

定义 5 (目标函数) 定义函数 $f(Sft_{\text{src}}, X)$ 用于计算相对于-O0 等级,使用 $S_S(X)$ 编译选项集获得 Sft_{src} 对应的可执行代码在运行时能耗所降低的百分比. $f(Sft_{\text{src}}, X)$ 的定义如式(5)所示.

$$f(Sft_{\text{src}}, X) = \frac{EvE(cmpLnk_0(Sft_{\text{src}})) - EvE(cmpLnk(Sft_{\text{src}}, X))}{EvE(cmpLnk_0(Sft_{\text{src}}))} \times 100\% \tag{5}$$

定义 6 (优化问题) 用于嵌入式软件能耗优化的编译选项选择问题可描述为:搜索满足式(6)的最优解 X^*.

$$X^* = \max_{X \in \Omega} f(Sft_{src}, X) \tag{6}$$

2 GA-FP 算法流程

GA-FP 算法用于求解式(6)定义的优化问题.它在传统遗传算法[16]中融入了频繁模式挖掘和启发式变异的方法,其流程如表 1 所示. GA-FP 算法主要包括初始化随机种群、计算种群中个体的适应度值、更新带能耗改进标注的编译选项事务表、挖掘带能耗改进标注的频繁编译选项模式集、种群中个体做交叉操作、种群中个体做启发式变异操作、按轮盘赌选择出下一代种群等主要步骤.下面仅对与频繁模式挖掘和启发式变异相关的步骤作简要介绍.

更新带能耗改进标注的编译选项事务表发生在每个个体的适应度值计算后.具体地,通过表 1 中的第 7

至第 13 步将有能耗改进效果的个体 X_k 的信息作为一条事务收集到形如表 2 的事务表 TT_E (the **T**ransaction **T**able with **E**nerge annotations of compilation options)中每条事务 T_E (a **T**ransaction with **E**nerge annotations)是三元组(编译选项编号、出现次数和能耗改进标注)的有序列表.它用于依次保存 X_k 选用的一组编译选项及能耗改进效果的信息.为了便于挖掘,在 T_E 的每个三元组中,出现次数固定为 1、能耗改进标注值为 X_k 较 GCC 的-O0 等级降低能耗的百分比(对应于 $f(Sft_{src}, X_k)$). 能耗改进标注表达了一个编译选项参与了多大能耗改进幅度的事务.

挖掘带能耗改进标注的频繁编译选项模式集(表 1 中的第 15 步),以及“增添”和“删减”两种启发式变异操作(表 1 中的第 19 步至第 23 步)将分别在下面的第 3 章和第 4 章给予详细的阐述.

Table 1 The flow of GA-FP algorithm

表 1 GA-FP 算法的流程

输入: 种群大小 N, 变异概率 p_m, 交叉概率 p_c, 演化代数 t, 最大演化代数 t_{max}
一个嵌入式软件源代码 Sft_{src}
输出: 最优解 X^*

```
1   演化代数 t←1;
2   产生大小为 N 的随机种群 Pop(t) ={X1,X2,...,Xk,...,XN},其中 1 ≤ ∀k ≤ N, Xk ∈ Ω;
3   事务标识 TID ← 0;
4   While (t ≤ tmax ) Do
5     For ( Pop(t) 的每个个体 Xk ) Do //计算种群中每个个体的适应度值
6       按函数 f(Sftsrc, Xk) 求解 Xk 的适应度值 fitVal; // 较 GCC 的-O0 等级降低能耗的百分比
7       If (fitVal > 0) //有能耗改进效果,则更新编译选项事务表
8         置编译选项事务 TE 为空;
9         For ( i in SS(Xk) ) //Xk 所选用的每个编译选项 i
10          TE ←TE ⋃{ (i,1, fitVal) };//i,1 和 fitVal 分别为编译选项编号, 出现次数和能耗降低百分比的值
11        EndFor
12        TID ←TID+1, 并将(TID, TE ) 插入表 2 示意的带能耗改进标注的编译选项事务表 TTE 中;
13      EndIf
14    End For
15    基于表 2,挖掘带能耗改进标注的频繁编译选项模式集; //频繁编译选项模式挖掘
16    对种群 Pop(t) 中个体按概率 pc 执行交叉操作生成临时种群 Q(t); //交叉操作
17    For (Q(t)的每个个体 X ) Do // 启发式变异操作
18      j ← 随机生成整数 0 或 1;
19      If (j ==0)
20        按概率 pm 并基于第 15 步得到的模式集,在 X 上执行删减操作; // 删减操作
21      Else
22        按概率 pm 并基于第 15 步得到的模式集,在 X 上执行增添操作; // 增添操作
23      End If
24    End For
25    基于轮盘赌策略在 Pop(t) ⋃Q(t) 中选择生成下一代种群 Pop(t+1) ;//轮盘赌选择操作
26    t ←t+1;
27  End While
28  X*← Pop(t) 中的最优解;
29  输出最优解 X*;
30  结束算法运行;
```

Table 2 An example for transaction table TT_E
with energy consumption annotations of compilation options

表 2 带能耗改进标注的编译选项事务表 TT_E 的例子

事务标识 TID	带能耗改进标注的编译选项事务 T_E T_E ={<*cmpOptInfo*>|*cmpOptInfo* =(编译选项编号 *cmpOptNm*,出现次数 *count*,能耗改进标注 *engAno*)}
1	< (**6**,1,12%), (1,1,12%), (**3**,1,12%), (4,1,12%), (7,1,12%), (9,1,12%), (13,1,12%), (16,1,12%) >
2	< (1,1,10%), (2,1,10%), (**3**,1,10%), (**6**,1,10%), (12,1,10%), (13,1,10%), (15,1,10%) >
3	< (2,1,11%), (6,1,11%), (8,1,11%), (10,1,11%), (15,1,11%) >
4	< (2,1,10%), (3,1,10%), (11,1,10%), (17,1,10%), (16,1,10%) >
5	< (1,1,8%), (**6**,1,8%), (**3**,1,8%), (5,1,8%), (12,1,8%), (16,1,8%), (13,1,8%), (14,1,8%) >

3 挖掘带能耗改进标注的频繁编译选项模式集

与一般的事务不同,表 2 每个事务 T_E 中的每个编译选项被标注了所参与事务的能耗改进信息.因而在挖掘过程中,不仅需要考虑各个编译选项在事务表中出现的频度,而且还要体现各个编译选项所参与事务的能耗改进情况.文中通过扩展经典的频繁模式(FP)树挖掘算法[17],挖掘出带能耗改进标注的频繁编译选项模式集.下面先给出与频繁编译选项模式相关的定义、带能耗改进标注的频繁模式树(简记为 FP_E 树)的数据结构后,再阐述 FP_E 树的构造算法和基于 FP_E 树的挖掘算法.

3.1 与频繁编译选项模式相关的定义

定义 7 (编译选项的支持计数) i 号编译选项的支持计数 $cnt(i)$ 由式(7)和(8)定义. 式(7)中, $T_E^j(k).cmpOptNm$ 表示事务表 TT_E 第 j 条事务中第 k 个元素的编译选项编号.而式 (8)中, m 和 $|T_E^j|$ 分别表示事务表 TT_E 的事务总数和第 j 条事务中元素的个数.

$$g(i,\ k,\ j)=\begin{cases}1 & \text{if } T_E^j(k).cmpOptNm = i\\ 0 & else\end{cases} \tag{7}$$

$$cnt(i)=\sum_{j=1}^{m}\sum_{k=1}^{|T_E^j|}g(i,k,j) \tag{8}$$

从定义 7 和定义 8 不难看出: $cnt(i)$ 是 i 号编译选项在整个事务表 TT_E 中出现的次数.例如:在表 2 的例子事务表 TT_E 中, $cnt(1)=3$.

定义 8 (频繁编译选项) 称 i 号编译选项是一个频繁编译选项,当且仅当 $cnt(i)$ 大于或等于预设的最小支持计数 $C_{\min}$.

定义 9 (编译选项集的支持计数) 若 $S_{cmpOptNm}$ 是编译选项编号的集合,则 $S_{cmpOptNm}$ 对应的编译选项集的支持计数用 $cnt_S(S_{cmpOptNm})$ 进行表示. $cnt_S(S_{cmpOptNm})$ 由式(9)和(10)定义. 式(9)中, $\Pi_{cmpOptNm}(T_E^j)$ 表示投影出事务表 TT_E 第 j 条事务中所有的编译选项编号.

$$g_S(S_{cmpOptNm},\ j)=\begin{cases}1 & if\ \Pi_{cmpOptNm}(T_E^j)\supseteq S_{cmpOptNm}\\ 0 & else\end{cases} \tag{9}$$

$$cnt_S(S_{cmpOptNm})=\sum_{j=1}^{m}g_S(S_{cmpOptNm},\ j) \tag{10}$$

从公式 9 和公式 10 不难看出: $cnt_S(S_{cmpOptNm})$ 是 $S_{cmpOptNm}$ 的各个编译选项在事务表 TT_E 各条事务中同时出现的次数.例如:在表 2 的例子事务表 TT_E 中, 6 号和 3 号组成的编译选项集{6,3}的支持计数 $cnt_S(\{6,3\})=3$.

定义 10 (频繁编译选项模式) 若 $S_{cmpOptNm}$ 和 $C_{\min}$ 分别是编译选项编号的集合和预设的最小支持计数, 则 $S_{cmpOptNm}$ 对应的编译选项集是频繁编译选项模式,当且仅当 $cnt_S(S_{cmpOptNm})\ge C_{\min}$.

本文基于带能耗改进标注的频繁模式树 FP_E, 挖掘频繁编译选项模式集,下面介绍 FP_E 树的数据结构.

3.2 FP_E树的数据结构

与传统 FP 树[17]的数据结构类似,FP_E 树也是由前缀项树 PT 和项头表 HL 所构成.但 FP_E 树融入了能耗标注,如图 2 所示.

前缀项树 PT 包含一个根结点 *root* 和若干棵前缀项子树. PT 树的每个结点用 *cmpOptNm*、*count*、*engAno*、*parNode* 和 *nextNode* 五个属性描述.它们分别表示编译选项编号、编译选项出现次数、能耗改进标注、指向父结点的指针和指向下一个具有相同编译选项编号的结点的指针.在图 2 椭圆形表示的结点中,逗号分隔的三个数值分别是 *cmpOptNm*、*count* 和 *engAno* 的值,而根结点 *root* 中的这三个数值为空 null. 图 2 中实线和虚线弧间接定义了每个结点的 *parNode* 和 *nextNode* 的值.没有虚线弧的结点,其 *nextNode* 值为空 null.与传统 FP 树不同之处在于 FP_E 树的结点引入了能耗改进标注的描述.

项头表 HL 的每个表项用 *cmpOptNm* 和 *hdLnk* 两个属性进行描述.这两个属性分别表示编译选项编号和结点链(虚线弧构成的链)的头指针.通过结点链可将同一个编译选项链接起来.例如:图 2 中项头表 HL 最后一行的头指针 *hdLnk* 将前缀树 PT 中所有出现 16 号编译选项的结点(灰色背景指示)链接起来.

3.3 FP_E树的构造算法

基于 FP_E 树的数据结构, 表 3 所示的 FP_E-Create 算法和表 4 所示的 Insert_tree 算法给出了 FP_E 树的构造过程. 整个过程可分为 3 个主要步骤:构造初始的 FP_E 树、生成频繁编译选项事务表 TTF_E (the **T**ransaction **T**able of **F**requent compilation options with **E**nerge annotation)、将 TTF_E 表各条事务中的频繁编译选项插入到 FP_E 树中.前两个步骤由 FP_E-Create 算法完成,而第 3 个步骤由 FP_E-Create 算法调用 Insert_tree 算法实现. 与传统 FP 树构造不同之处在于 FP_E-Create 算法对能耗改进标注的处理.下面以表 2 的有能耗改进效果的事务表 TT_E 和最小支持计数 $C_{\min}=3$ 为例,简要解释 FP_E 树的构造过程.

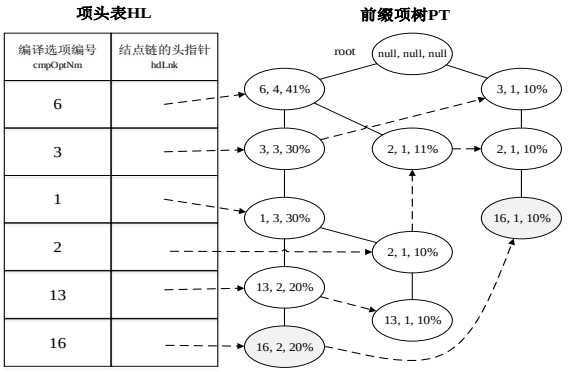

Fig.2 FP_E tree derived from the example transaction table TT_E shown in table 2

图 2 基于表 2 例子事务表 TT_E 生成的 FP_E 树

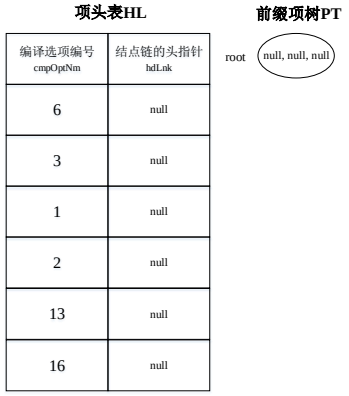

Fig.3 The initial FP_E tree in the example

图 3 例子初始的 FP_E 树

Table 3 FP_E-Create algorithm

表 3 FP_E-Create 算法

输入:有能耗改进效果的事务表 TT_E, 最小支持计数 $C_{\min}$	
输出: FP_E 树	
1	扫描一次事务表 TT_E 收集频繁编译选项的支持计数, 并按支持计数的降序进行排序,建立排序后的频繁编译选项序列 Seq_{FCOpt} ; // Seq_{FCOpt} 的每个元素为二元组(编译选项编号 *cmpOptNm*,支持计数 *count*)
2	将 Seq_{FCOpt} 中各元素的编译选项编号依次赋值给项头表 HL 各表项的编译选项编号 *cmpOptNm* 属性, 并令 HL 中各表项的指针域 *hdLnk* 为 null; //构造初始的项头表
3	创建根结点 *root*,并令 *root* 中各个属性的值为 null; //构造初始的前缀项树
4	扫描一次事务表 TT_E 筛选出频繁编译选项,并按 Seq_{FCOpt} 中频繁编译选项编号的顺序进行排序, 生成排序后的频繁编译选项事务表 TTF_E ; //此步骤不修改各频繁编译选项的能耗改进标注值
5	For (频繁编译选项事务表 TTF_E 的每一行事务 TF_E) Do
6	带能耗改进标注的频繁编译选项列表 $List_{FCOpt} \leftarrow TF_E$;
7	调用 Insert_tree($List_{FCOpt}$, *root*);
8	End For
9	输出 FP_E 树;
10	结束算法运行;

Table 4 Insert_tree algorithm

表 4 Insert_tree 算法

输入: 带能耗改进标注的频繁编译选项列表 $List_{FCOpt}$, FP_E 树的根 *root*	
输出: void	
1	If ($List_{FCOpt}.size \geq 1$)
2	If (*root* 有孩子结点 *node* 且 *node.cmpOptNm* == $List_{FCOpt}$ [0].*cmpOptNm*) //有匹配的孩子结点
3	*node.count* ← *node.count*+1; //修改支持计数
4	*node.engAno* ←*node.engAno*+ $List_{FCOpt}$ [0].*engAno*; //修改能耗改进标注值
5	Else
6	创建一个新结点 *node*;

```
7          node. cmpOptNm ← List_FCOpt [0].cmpOptNm; // 赋编译选项编号的值
8          node.count←1; //赋支持计数的值
9          node.engAno= List_FCOpt [0].engAno; //赋能耗改进标注的值
10         node.parNode ← root, node.nextNode ← null; // 插入到 FP_E 树中
11         在项头表 HL 中找名为 node.cmpOptNm 的表项 TblItem; //将结点 node 链接到结点链中
12         If (TblItem.hdLnk ==null)
13           TblItem.hdLnk ← node;
14         Else
15           沿着 TblItem.hdLnk 找到尾结点 Tail;
16           Tail.nextNode ← node;
17         End If
18       End If
19       复制 List_FCOpt 中除第 1 个元素外的所有元素,得到一个新列表 newList_FCOpt ;
20       Insert_tree( newList_FCOpt , node);
21     End If
```

(1) 构造初始的 FP_E 树

基于表 2 的有能耗改进效果的事务表 TT_E,并利用 FP_E-Create 算法的第 1 步先得到频繁编译选项序列 Seq_{FCOpt}= <(1,3), (2,3), (3,4), (6,4), (13,3), (16,3)>. 在 Seq_{FCOpt} 中,每个二元组的第 1 个和 2 个元素分别是编译选项的编号和支持计数.由于 Seq_{FCOpt}存放的是各个频繁编译选项, Seq_{FCOpt} 中各个二元组的支持计数应大于等于最小支持计数 $C_{\min}=3$.对 Seq_{FCOpt} 按照支持计数降序排列后, Seq_{FCOpt} = <(**6**,4), (**3**,4), (**1**,3), (**2**,3), (**13**,3), (**16**,3)>.

FP_E-Create 算法的第 2 步根据 Seq_{FCOpt} 中各频繁编译选项的顺序<6, 3, 1, 2, 13, 16>构造出项头表 HL,并经第 3 步生成仅包含根结点 *root* 的一棵 FP_E 树,如图 3 所示.

(2) 生成排序后的频繁编译选项事务表 TTF_E

FP_E-Create 算法的第 4 步扫描事务表 TT_E 按最小支持计数 $C_{\min}=3$ 筛选出频繁编译选项, 并按 Seq_{FCOpt} 中频繁编译选项的顺序进行排序, 生成排序后的频繁编译选项事务表 TTF_E,如表 5 所示.

Table 5 The transaction table TTF_E of frequent compilation options obtained by selecting and sorting the transaction table TT_E

表 5 对事务表 TT_E 进行筛选和排序后获取的频繁编译选项事务表 TTF_E

事务标识 TID	频繁编译选项事务 TF_E TF_E ={<*FCOptInfo*>| *FCOptInfo* =(编译选项编号 *cmpOptNm*,出现次数 *count*,能耗改进标注 *engAno*)}
1	< (6,1,12%), (3,1,12%), (1,1,12%), (13,1,12%), (16,1,12%) >
2	< (6,1,10%), (3,1,10%), (1,1,10%), (2,1,10%), (13,1,10%)>
3	< (6,1,11%), (2,1,11%) >
4	< (3,1,10%), (2,1,10%), (16,1,10%) >
5	< (6,1,8%), (3,1,8%), (1,1,8%), (13,1,8%), (16,1,8%) >

(3) 将频繁编译选项事务表 TTF_E 各事务中的频繁编译选项插入到 FP_E 树

利用 FP_E-Create 算法的第 5 至第 8 步并以表 5 频繁编译选项事务表 TTF_E 的每条事务为单位,通过调用 Insert_tree 算法将各事务中的每个频繁编译选项依次插入到 FP_E 树中.

Insert_tree 算法在将 *i* 号频繁编译选项插入到 FP_E 树时,既要考虑其出现次数,又要体现所参于事务的能耗改进幅度.依据根结点有无相同编译选项编号的孩子结点,分为两种情况: 若没有匹配的孩子结点,则新增结点,并将新增结点的出现次数属性 count 和能耗标注属性 *engAno* 的值分别置为 1 和 *i* 号编译选项所关联的能耗改进标注值; 否则,将匹配的孩子结点的 count 属性和 *engAno* 属性分别累加 1 和累加 *i* 号编译选项所关联的能耗改进标注值. 下面以插入表 5 频繁编译选项事务表 TTF_E 的前 2 行事务为例进行解释和说明.

① 插入表 5 事务表 TTF_E 第 1 行事务中的各频繁编译选项

在图 3 初始 FP_E 树的基础上,将表 5 事务表 TTF_E 中的第 1 行事务中各频繁编译选项依次插入到 FP_E 树中.由于 Insert_tree 算法每次递归插入时,根结点 root 没有匹配的孩子结点,故依次生成 5 个新结点,建立 FP_E

树的第 1 个分支,如图 4 所示.

② 插入表 5 事务表 TTF_E 第 2 行事务中各频繁编译选项

在图 4 的 FP_E 树基础上,将表 5 事务表 TTF_E 第 2 行事务的各频繁编译选项依次插入到 FP_E 树时,由于插入第 1 个频繁编译选项(其编号、支持计数和能耗改进标注分别为 6、1 和 10%)时,*root* 结点有一个匹配的孩子结点(如图 4 灰色背景的结点,其支持计数和能耗改进标注分别为 1 和 12%),需要将这个孩子结点的计数和能耗改进标注分别与表 5 第 2 行事务中 6 号编译选项的支持计数和能耗改进标注进行累加(见 Insert_tree 算法的第 3 步和第 4 步)并更新为 *count*=2 和 *engAno*=22%.

类似地,插入表 5 第 2 行的第 2 和第 3 个频繁编译选项.而当插入第 4 个频繁编译选项(其编号、支持计数和能耗改进标注分别为 2、1 和 10%)时,此时的根结点(1, 2, 22%)下没有匹配的孩子结点,根据 Insert_tree 算法的第 6 至第 17 步,在根结点(1, 2,22%)下新建一个孩子结点并将表 5 第 2 行事务中 2 号编译选项的支持计数和能耗改进标注赋值给该孩子结点的对应属性,从而得到 FP_E 树的第 2 个分支. 同理将第 5 个频繁编译选项插入到 FP_E 树得到图 5 所示的 FP_E 树.

同理再依次插入表 5 第 3、第 4 和第 5 行事务中的各编译选项可输出表 2 事务表 TT_E 所对应的 FP_E 树,如图 2 所示.

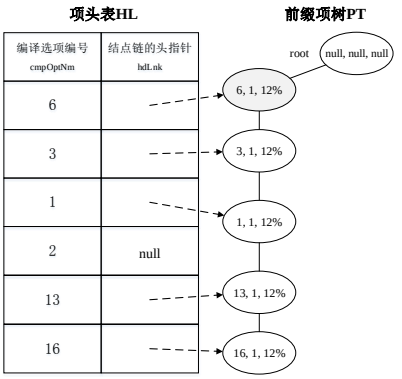

Fig.4 FP_E tree obtained by inserting frequent compilation options of 1st row in Table 5 TTF_E

图 4 将表 5 事务表 TTF_E 第 1 行事务中各频繁编译选项插入后得到的 FP_E 树

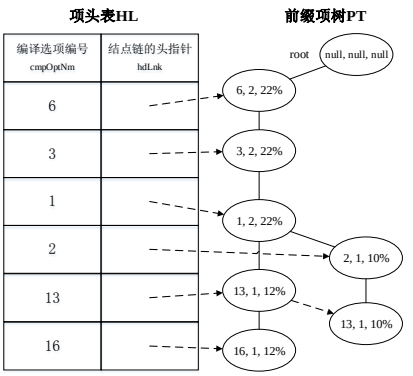

Fig.5 FP_E tree obtained by inserting frequent compilation options of 1st and 2nd rows in Table 5 TTF_E

图 5 将表 5 事务表 TTF_E 第 1 行和第 2 行事务中各频繁编译选项插入后得到的 FP_E 树

3.4 基于FP_E树的挖掘算法

下面先给出与 FP_E 树挖掘相关的概念,再阐述基于 FP_E 树挖掘算法的具体流程.

(1) 与 FP_E 树挖掘相关的概念

定义 11 (前缀路径) 若 *node* 是 FP_E 树的一个非根结点,则 *node* 的前缀路径定义为从根 *root* 到 *node* 之间的一条路径.它对应一个结点序列并简记为 *path*|*node*,又称 *node* 是前缀路径 *path*|*node* 的后缀结点.

例如:图 2 项头表 HL 最后一个表项的头指针指向的结点 *node* 所对应的前缀路径 *path*|*node* 为: $path_1$=<(6,4,41%),(3,3,30%),(1,3,30%),(13,2,20%)>.

定义 12 (条件前缀路径) 若 *node* 的前缀路径为 *path*|*node*,则 *node* 的条件前缀路径定义为:将 *path*|*node* 上各结点的支持计数修改为 *node* 的支持计数并记作 *cndPath*|*node*.

node 的支持计数小于等于其前缀路径上每个结点的支持计数.这一性质由 FP_E-Create 算法保证,文献[17]也给出了相应的证明.支持计数的修改反映了 *node* 指示的编译选项与 *path*|*node* 中各结点的指示编译选项共同出现的次数. 与支持计数不同,能耗改进标注用于刻画频繁编译选项所参与各事务的能耗改进效果,故未进行修改.

例如:设图 2 项头表 HL 最后一个表项的头指针指向的结点为 *node*, 其支持计数和所对应的前缀路径分是 2 和 $path_1$=<(6,4,41%),(3,3,30%),(1,3,30%),(13,2,20%)>.则通过修改支持计数后得到 *node* 的条件前缀路径 *cndPath*|*node* 为:

cndPath$_1$=<(6,**2**,41%),(3,**2**,30%),(1,**2**,30%),(13,**2**,20%)>.

定义 13 (频繁编译选项关联的 FP$_E$ 树结点集) 若 i 号编译选项是频繁编译选项, 则 i 关联的 FP$_E$ 树结点集 $nodes(i)$ 由式(11)定义. 式(11)中的 row, $head(node)$分别表示项头表 HL 的行下标和获取 $node$ 所在结点链的头指针.

$$nodes(i)=\{\, node \mid i=\mathrm{HL}[row].cmpOptNm \wedge head(node)=\mathrm{HL}[row].hdLnk \,\} \tag{11}$$

由式(11)可以看出: i 号频繁编译选项关联的 FP$_E$ 树结点集是一个结点链上各结点构成的集合,且该结点链上各结点的编译选项编号为 i.例如:图 2 中 16 号频繁编译选项关联的 FP$_E$ 树结点集中包含了 2 个灰色背景指示的结点.

定义 14 (频繁编译选项的条件前缀路径集) 若 i 号编译选项是频繁编译选项,则 i 对应的条件前缀路径集 $cndPaths \mid i$ 由式(12)定义.

$$cndPaths \mid i=\{\, (cndPath \mid node) \mid node \in nodes(i) \,\} \tag{12}$$

例如:图 2 中 16 号频繁编译选项关联 2 个灰色背景指示的结点.左边结点的条件前缀路径是 cndPath$_1$=<(6,2,41%),(3,2,30%),(1,2,30%),(13,2,20%)>, 而右边结点的条件前缀路径 cndPath$_2$=<(3,1,10%),(2,1,10%)>.因而,图 2 中 16 号频繁编译选项的条件前缀路径集为:

$cndPaths \mid 16=\{$<(6,2,41%),(3,2,30%),(1,2,30%),(13,2,20%)>, <(3,1,10%),(2,1,10%)>$\}$

定义 15 (频繁编译选项的条件事务表) 若 i 号频繁编译选项的条件前缀路径集为 $cndPaths \mid i$,则以 $cndPaths$ 中每条路径为一条事务,将构成一个以 i 为后缀的频繁编译选项事务表 TTF$_E$ $\mid i$.并称其为频繁编译选项 i 的条件事务表.

例如, 图 2 中 16 号频繁编译选项的条件前缀路径集是 $cndPaths \mid 16$, 则以 16 为后缀的频繁编译选项事务表 TTF$_E$ $\mid 16$ 如表 6 所示.

Table 6 Conditional transaction table corresponding to the frequent compilation option numbered 16 in the example

表 6 例子中 16 号频繁编译选项所对应的条件事务表

事务标识 TID	频繁编译选项事务 TF_E $TF_E=\{<FCOptInfo> \mid FCOptInfo=$(编译选项编号 $cmpOptNm$,出现次数 $count$,能耗改进标注 $engAno$)$\}$
1	< (6,2,41%), (3,2,30%), (1,2,30%), (13,2,20%)>
2	< (3,1,10%), (2,1,10%)>

定义 16 (条件 FP$_E$ 树) 若 TTF$_E$ $\mid i$ 是频繁编译选项 i 的条件事务表, 则以 TTF$_E$ $\mid i$ 为输入所构建的 FP$_E$ 树定义为频繁编译选项 i 的条件 FP$_E$ 树,并简记为 FP$_E$ $\mid i$.

例如: 表 6 是 16 号频繁编译选项的条件事务表,在最小支持计数 $C_{\min}=3$ 下运用 FP$_E$-Create 算法和 Insert_tree 算法可构建条件 FP$_E$ 树,如图 6 所示.

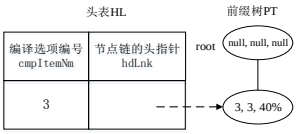

Fig.6 Conditional FP$_E$ tree of the frequent compilation option numbered 16 in the example

图 6 例子中 16 号频繁编译选项的条件 FP$_E$ 树

定义 17 (带能耗改进标注的频繁编译选项模式) 设 fp_{cmpOpt} 是带能耗改进标注的编译选项集合 $\{cmpOptInfo \mid cmpOptInfo=(cmpOptNm, engAno)\}$.其中,$cmpOptNm$ 和 $engAno$ 分别表示编译选项编号和能耗改进标注.若投影 fp_{cmpOpt} 中每个元素的 $cmpOptNm$ 而获得的编译选项集是满足定义 10 的频繁编译选项模式,则

称 fp_{cmpOpt} 是一个带能耗改进标注的频繁编译选项模式. 当 $|fp_{\text{cmpOpt}}|=k$ 称 fp_{cmpOpt} 为带能耗标注的 k 频繁编译选项模式,并简记为 fp^{k}_{cmpOpt} .

(2)算法流程

表 7 给出了带能耗改进标注的频繁编译选项模式的挖掘算法 FP_E-growth 描述.当以 FP_E-Create 算法构造的 FP_E 树和空 null 作为参数调用 FP_E-growth,可输出频繁编译选项模式集表.其包含了所有的 k 频繁编译选项模式集. 对图 2 的 FP_E 树运用 FP_E-growth 算法可输出表 8 所示的频繁编译选项模式集表.

Table 7 FP_E-growth algorithm

表 7 FP_E-growth 算法

输入: FP_E 树, 后缀结点集 *postFixNodes*	
输出: 频繁编译选项模式集表 //全局变量,初始为空	
1	If (FP_E 树中除根结点外只有一个结点 *node*)
2	连接 *node* 与 *postFixNodes*,得到带能耗改进标注的频繁编译选项模式 fp_{cmpOpt} ;
3	根据 fp_{cmpOpt} 的元素个数 *k*,将 fp_{cmpOpt_E} 输出到频繁编译选项模式集表的 *k* 频繁模式集中;
4	Else
5	For (FP_E 树项头表 HL 的每一行 *row*) Do
6	*node* ← *row.hdLink*; //将 *row* 头指针 *hdLnk* 指向的结点赋给结点 *node*
7	根据 *node* 得到带能耗改进标注的频繁编译选项模式 fp_{cmpOpt} , 并基于 fp_{cmpOpt} 中元素的个数 *k*, 将 fp_{cmpOpt} 输出到频繁编译选项模式集表的 *k* 频繁模式集中;
8	构造 FP_E|*row.cmpOptNum*; //构造以 *row.cmpOptNum* 为后缀的条件 FP_E 树
9	*postFixNodes* ← *postFixNodes* ∪ *node*; //加入 *node* 结点到 *postFixNodes*
10	FP_E-growth(FP_E|*row.cmpOptNum*, *postFixNodes*);
11	End For
12	End If
13	结束算法运行;

Table 8 The table for the set of frequent pattern of compilation options with energy annotations in the example

表 8 例子的带能耗改进标注的频繁编译选项模式集表

k 频繁编译选项模式集(记 S^{k}_{fp})	带能耗改进标注的 *k* 频繁编译选项模式(记 fp^{k}_{cmpOpt})
1 频繁编译选项模式集 S^{1}_{fp}	{(16,30%)} {(13,30%)} {(2,31%)} {(1,30%)} {(3,40%)} {(6,41%)}
2 频繁编译选项模式集 S^{2}_{fp}	{(16,30%),(3,40%)} {(13,30%),(6,82%)} {(13,30%),(3,60%)} {(13,30%),(1,60%)} {(1,30%),(6,41%)} {(1,30%),(3,30%)} {(3,30%),(6,41%)}
3 频繁编译选项模式集 S^{3}_{fp}	{(13,30%),(6,82%),(3,60%)} {(13,30%),(6,82%),(1,60%)} {(13,30%),(3,60%),(1,60%)} {(1,30%),(3,30%),(6,41%)}
4 频繁编译选项模式集 S^{4}_{fp}	{(13,30%),(6,82%),(3,60%),(1,60%)}

4 变异操作

GA-FP 算法引入了“增添”和“删减”作为两种启发式变异操作.这两种操作以带能耗改进标注的频繁编译选项模式集为启发式信息,下面给出相关定义后,再给出它们的具体操作步骤.

4.1 变异操作相关定义

定义 18 (频繁编译选项模式的加 1 匹配) 若 $S_S(X)$ 和 $S_U(X)$ 分别是个体 X 指示的已选用和未选用的编译选项编号集,并且从 $S_S(X)$ 中随机选取 k 个元素($1\le k\le|S_S(X)|$)构成编译选项编号集合为 $S_{cmpOptNm}$,则 $S_{cmpOptNm}$ 的频繁编译选项模式的加 1 匹配由式(13)定义. 式(13)中, S_{fp}^{k+1} 是挖掘出的所有 k+1 频繁编译选项模式构成的集合, 而 $\Pi_{\mathrm{cmpOptNm}}(fp_{cmpOpt})$ 表示从频繁编译选项模式 fp_{cmpOpt} 中投影出各编译选项编号所构成的集合.

$$fpMatch^{+}(S_{cmpOptNm})=\{fp_{cmpOpt}\mid(k=|S_{cmpOptNm}|)\wedge(fp_{cmpOpt}\in S_{fp}^{k+1})\wedge(S_{cmpOptNm}\subset\Pi_{cmpOptNm}(fp_{cmpOpt}))\wedge((\Pi_{cmpOptNm}(fp_{cmpOpt})-S_{cmpOptNm})\subset S_U(X))\} \tag{13}$$

例如:个体 X={0,0,1,0,1,0,0,0,0,1,1,1,0,1,1,0,1},则 $S_S(X)$={3,5,9,10,11,13,14,16}, $S_U(X)$={1,2,4,6,7,8,12,15}. 挖掘获得的带能耗改进标注的频繁编译选项模式集如表 8 所示.设从 $S_S(X)$ 中随机选取 2 个元素构成的编译选项编号集合为 $S_{cmpOptNm}=\{3,13\}$,从表 8 的频繁编译选项模式集 S_{fp}^{3} 中对应的 4 行可分别投影出 4 个编译选项编号集: {13, **6**, 3}、{13, 6, 1}、{13, 3, **1**}、{1, 3, 6}. 仅第 1 和第 3 个集合包含 $S_{cmpOptNm}=\{3,13\}$,并且这两个集合与 $S_{cmpOptNm}$ 的差集都被 $S_U(X)$ 包含, 故 $fpMatch^{+}(\{3,13\})$ ={{(13,30%), (**6,82%**), (3,60%)}, {(13,30%), (3,60%), (**1,60%**)}}.

定义 19(频繁编译选项的加 1 匹配) 若 $S_{cmpOptNm}$ 是从个体 X 的已选用编译选项编号集中随机抽取大小为 k 的子集, $S_{cmpOptNm}$ 的频繁编译选项模式加 1 匹配为 $fpMatch^{+}(S_{cmpOptNm})$,并且 $fpMatch^{+}(S_{cmpOptNm})$ 不空,则 $S_{cmpOptNm}$ 的频繁编译选项加 1 匹配 $fCOptMatch^{+}(S_{cmpOptNm})$ 由式(14)定义. 式(14)中, *fCOpt.cmpOptNm* 表示频繁编译选项 *fCOpt* 的编译选项编号.

$$fCOptMatch^{+}(S_{cmpOptNm})=\{fCOpt\mid(fp_{cmpOpt}\in fpMatch^{+}(S_{cmpOptNm}))\wedge(fCOpt.cmpOptNm\in(\Pi_{cmpOptNm}(fp_{cmpOpt})-S_{cmpOptNm}))\} \tag{14}$$

例如:在上例的 $fpMatch^{+}(\{3,13\})$ 中, 两个频繁编译选项模式可分别投影出 2 个编译选项编号集{13, **6**, 3}、{13, 3, **1**}.这两个集合与 $S_{cmpOptNm}=\{3,13\}$ 的差集分别为:{**6**}、{**1**}.根据差集,可从 $fpMatch^{+}(\{3,13\})$ 中获取 $fCOptMatch^{+}(\{3,13\})=\{(6,82\%),(1,60\%)\}$.

定义 20 (频繁编译选项模式的减 1 匹配) 若个体 X 的已选用和未选用的编译选项编号集分别为 $S_S(X)$ 和 $S_U(X)$,并且从 $S_S(X)$ 中随机选取 k 个元素($1<k\le|S_S(X)|$)构成编译选项编号集为 $S_{cmpOptNm}$,则 $S_{cmpOptNm}$ 的频繁编译选项模式减 1 匹配 $fpMatch^{-}(S_{cmpOptNm})$ 由式(15)定义. 式(15)中, S_{fp}^{k-1} 是挖掘出的所有 k-1 频繁编译选项模式构成的集合, 而 $\Pi_{cmpOptNm}(fp_{cmpOpt})$ 表示从频繁编译选项模式 fp_{cmpOpt} 中投影出各编译选项编号所构成的集合.

$$fpMatch^{-}(S_{cmpOptNm})=\{fp_{cmpOpt}\mid(k=|S_{cmpOptNm}|)\wedge(fp_{cmpOpt}\in S_{fp}^{k-1})\wedge(S_{cmpOptNum}\supset\Pi_{cmpOptNm}(fp_{cmpOpt}))\} \tag{15}$$

例如:个体 X={0,0,1,0,1,0,0,0,0,1,1,1,0,1,1,0,1},则 $S_S(X)$={3,5,9,10,11,13,14,16}, $S_U(X)$={1,2,4,6,7,8,12,15}. 挖掘获得的带能耗改进标注的频繁编译选项模式集如表 8 所示.设从 $S_S(X)$ 中随机选取 2 个元素构成的编译选项编号集合为 $S_{cmpOptNm}=\{3,16\}$,从表 8 的频繁编译选项模式集 S_{fp}^{1} 包含 6 个元素:{(**16**,30%)}、{(13,30%)}、{(2,31%)}、{(1,30%)}、{(**3**,40%)}和{(6,41%)}. 仅第 1 和第 5 个元素的编译选项编号被包含于 $S_{cmpOptNm}=\{3,16\}$, 故 $fpMatch^{-}(\{3,16\})$ ={{(16,30%)},{(3,40%)}}.

4.2 增添操作

表 9 给出了按增添操作的具体流程.当个体 X 没有选用任何编译选项,则采取随机单点变异方式将 X 的一位由 0 变 1;否则实施启发式增添编译选项.下面沿用定义 18 和定义 19 中使用的例子解释增添操作过程中的启发式变异.

设个体 X={0,0,1,0,1,0,0,0,0,1,1,1,0,1,1,0,1},则 $S_S(X)$={3,5,9,10,11,13,14,16}, $S_U(X)$={1,2,4,6,7,8,12,15}.挖掘获得的带能耗改进标注的频繁编译选项模式集如表 8 所示.假定表 9 第 5 步从 $S_S(X)$ 中随机选取 k=2 个元素组成待变异的编译选项编号集合为 $S_{cmpOptNm}=\{3,13\}$. 根据定义 19 可得 $fCOptMatch^{+}(\{3,13\})=\{(6,82\%),(1,60\%)\}$,其过程分析见定义 19 的例子说明.故表 9 第 6 步得到频繁编译选项集 $S_{fCOpt}=\{(6,82\%),(1,60\%)\}$.由表 9 第 10 步知 $p_{fCOpt}(1)=\frac{82\%}{82\%+60\%}\approx0.58$, $p_{fCOpt}(2)=\frac{60\%}{82\%+60\%}\approx0.42$.因而,分别以 0.58 和 0.42 概率选择 6 号和 1 号编译选项.假设选中 6 号编译选项,则通过表 9 第 13 步将 X 下标为 6 的码值替换成 1,得到了新个体 X'={0,0,1,0,1,**1**,0,0,1,1,1,0,1,1,0,1}.

4.3 删减操作

表 10 给出了按删减操作的具体过程.若个体 X 没有选用任何编译选项,则不做删减操作.当个体 X 仅选用 1 个编译选项,则删减这个编译选项.除此两种情况外,实施启发式删减编译选项.沿用定义 20 所举的例子说明删减操作过程的启发式变异.

假定个体 X={0,0,1,0,1,0,0,0,1,1,1,0,1,1,0,1},则 $S_S(X)$={3,5,9,10,11,13,14,16}, $S_U(X)$={1,2,4,6,7,8,12,15}.挖掘获得的带能耗改进标注的频繁编译选项模式集如表 8 所示.设表 10 第 6 步从 $S_S(X)$ 中随机选取 k=2 个元素构成的编译选项编号集合为 $S_{cmpOptNm}$={3,16}.根据定义 20 可得 $fpMatch^-(\{3,16\})$={{(16,30%)},{(3,40%)}},其过程分析见定义 20 下的例子说明.因而, 表 10 第 7 步得到 s_{fCOpt}={{(16,30%)},{(3,40%)}}.由表 10 第 11 步知: $p_{fCOpt}(1)=\frac{(30\%+40\%)-30\%}{(2-1)\times(30\%+40\%)}\approx 0.57$, $p_{fCOpt}(2)=\frac{(30\%+40\%)-40\%}{(2-1)\times(30\%+40\%)}\approx 0.43$,故分别以 0.57 和 0.43 概率选择 16 号和 3 号编译选项. 假设选中 16 号编译选项进行移除,而保留潜在能耗效果改进更好的 3 号编译选项.通过表 10 第 15 步将 X 下标为 16 的码值替换成 0,得到了新个体 X'={0,0,1,0,1,0,0,0,1,1,1,0,1,1,0,**0**}

Table 9 The flow of "Add" operation
表 9 "增添"操作的流程

输入:个体 X、带能耗改进标注的频繁编译选项模式集表
输出:新个体 X'

1 解析变异个体 X,得到已选用的编译选项编号的集合 $S_S(X)$ 和未选用的编译选项编号集合 $S_U(X)$;
2 If ($|S_S(X)|$ = 0) // 个体没有选用任何编译选项
3 在 $S_U(X)$ 中按等概率方式随机选择一个变异位置;
4 Else
5 在 $S_S(X)$ 中随机选择 k ($1\le k\le|S_S(X)|$) 个元素组成待变异的编译选项编号集 $S_{cmpOptNm}$;
6 频繁编译选项集 $s_{fCOpt}\leftarrow fCOptMatch^+(S_{cmpOptNm})$;
7 If (s_{fCOpt} 为空集)
8 在 $S_U(X)$ 中按等概率方式随机选择一个变异位置;
9 Else
10 对 s_{fCOpt} 中第 j 个频繁编译选项 $fCOpt_j$,以概率 $p_{fCOpt}(j)=\frac{fCOpt_j.engAno}{\sum_{fCOpt\in S_{fCOpt}} fCOpt.engAno}$ 进行选择, 并按 $fCOpt_j$ 的编译选项编号确定变异位置; // $fCOpt_j.engAno$ 表示获取频繁编译选项 $fCOpt_j$ 的能耗改进标注
11 End If
12 End If
13 将变异位置上的码值 0 替换成 1,并得到新的个体 X';
14 输出个体 X';

Table 10 The flow of "Delete" operation
表 10 "删减" 操作的流程

输入:个体 X、带能耗改进标注的频繁编译选项模式集表
输出:新个体 X'

1 解析变异个体 X,得到已选用的编译选项编号的集合 $S_S(X)$ 和未选用的编译选项编号的集合 $S_U(X)$;
2 If ($|S_S(X)|>0$) // 个体有选中的编译选项
3 If ($|S_S(X)|$ =1) //个体只有 1 个选用的编译选项
4 根据 $S_S(X)$ 中唯一的编译选项编号确定变异位置;
5 Else
6 在 $S_S(X)$ 中随机选择 k ($1<k\le|S_S(X)|$) 个元素组成待变异的编译选项编号集 $S_{cmpOptNm}$;
7 频繁编译选项集 $S_{fCOpt}\leftarrow fpMatch^-(S_{cmpOptNm})$;
8 If ($|S_{fCOpt}|\le 1$) // 当 S_{fCOpt} 只有一个元素时,为避免仅有的一个频繁且有改进效果的编译选项 // 被确定选中删减,也采用随机选择变异位置的策略
9 在 $S_{cmpOptNm}$ 中随机选择一个,并确定变异位置;
10 Else
11 //让能耗改进标注值大的编译选项以小概率被"删减"操作选择中, 等价于在 // { ($\sum_{fCOpt\in S_{fCOpt}} fCOpt.engAno$) − $fCOpt_j.engAno$ } 中选择大概率的选项

对 S_{fCOpt} 中第 j 个频繁编译选项 $fCOpt_j$,以概率 $p_{fCOpt}(j)=\frac{(\sum_{fCOpt\in S_{fCOpt}} fCOpt.engAno)-fCOpt_j.engAno}{(|S_{fCOpt}|-1)\times\sum_{fCOpt\in S_{fCOpt}} fCOpt.engAno}$ 进行选择,并按 $fCOpt_j$ 的编译选项编号确定变异位置;

```
        End If
      End If
    End If
    将变异位置上的码值 1 替换成 0,并得到新个体 X';
    输出 X';
```

5 案例研究

本节给出了案例研究. 5.1 节简介了实验案例; 5.2 节提出了要验证的问题及度量标准; 5.3 节说明了实验中使用的统计方法; 5.4 节介绍了实验安装; 5.5 节展示了实验结果并进行了分析.

5.1 案例简介

本文从 BEEBS 平台[18]中选用涵盖安全、网络、通信、汽车和消费五个领域的 8 个案例.表 11 给出了这些案例的应用领域和代码量.

Table 11 The application domains and code size in the experimental cases

表 11 实验案例的应用领域和代码量

案例	应用领域	代码量(行数)
2D FIR	汽车、消费	77
Blowfish	安全	658
CRC32	网络、通信	159
Cubic root solver(Cubic)	汽车	211
Dijkstra	网络	175
FDCT	消费	245
Float Matrix Multiplication(Float_Matrix)	汽车、消费	118
Integer Matrix Multiplication(Int_Matrix)	汽车	114

5.2 研究问题及其度量指标

问题 1(解质量):本文 GA-FP 算法较 Tree-EDA 算法能否得到更优的编译选项组合,使得案例的运行能耗更低?Tree-EDA 是目前以能耗为优化目标并可获取较优编译选项组合的一种算法.通过回答这一问题,可以验证 GA-FP 算法的有效性.

度量指标:将 GA-FP 和 Tree-EDA 最优解对应的能耗相对改进百分比 $I_{\Delta eng\%}$ 作为度量指标. 在案例源代码 Sft_{src} 下, $I_{\Delta eng\%}$ 的定义如式(16)所示,其值越大越好. 在式(16)中, $X^*_{\text{Tree-EDA}}$ 和 $X^*_{\text{GA-FP}}$ 分别表示 Tree-EDA 和 GA-FP 算法获得的最优解. $EvE(cmpLnk(Sft_{src}, X^*_{\text{Tree-EDA}}))$ 和 $EvE(cmpLnk(Sft_{src}, X^*_{\text{GA-FP}}))$ 分别表示在案例源代码 Sft_{src} 下 Tree-EDA 和 GA-FP 算法最优解对应的能耗值.

$$I_{\Delta eng\%}(Sft_{\text{src}})=\frac{EvE(cmpLnk(Sft_{src}, X^*_{\text{Tree-EDA}}))-EvE(cmpLnk(Sft_{src}, X^*_{\text{GA-FP}}))}{EvE(cmpLnk(Sft_{src}, X^*_{\text{Tree-EDA}}))}\times 100\% \tag{16}$$

问题 2(收敛速度):与 Tree-EDA 算法相比,GA-FP 算法能否加快收敛速度? 通过回答这一问题进一步验证 GA-FP 算法的有效性.

度量指标:为了公平比较两种算法的收敛速度,将 GA-FP 达到不劣于 Tree-EDA 最优解对应最小迭代次数的相对减少百分比 $I_{\Delta i\%}$ 作为度量指标. 在案例源代码 Sft_{src} 下, $I_{\Delta i\%}$ 的定义如式(17)所示,其值越大越好. 在式(17)中, $MinI_{Tree-EDA}(Sft_{src}, X^*)$ 和 $MinI_{GA-FP}(Sft_{src}, X^*)$ 分别表示在案例源代码 Sft_{src} 下 Tree-EDA 和 GA-FP 获取最优解 X^* 所需要的最小迭代次数.

$$I_{\Delta i\%}(Sft_{\text{src}})=\frac{MinI_{Tree-EDA}(Sft_{src}, X^*)-MinI_{GA-FP}(Sft_{src}, X^*)}{MinI_{Tree-EDA}(Sft_{src}, X^*)}\times 100\% \tag{17}$$

问题 3 (最优解中编译选项的正相关性):与 Tree-EDA 算法相比,本文 GA-FP 算法所获得的最优解中编译选项之间是否存在更强的正相关性? 依赖于具体的软件源代码和执行平台中与能耗相关的特征,GCC 多

个编译选项之间呈现不相关、负相关和正相关等不同的影响关系. 通过回答这一问题可揭示 GA-FP 较 Tree-EDA 获得更好的解质量和更快的收敛速度的原因. 亦即 GA-FP 在出现频度高且对能耗有显著改进效果的一组编译选项基础上,引入的启发式变异应使得获取的最优解中包括更多正相关的编译选项.

度量指标:一组正相关编译选项是指若从中移除任何一个选项,将使能耗的优化效果显著变差.考虑到 GA-FP 和 Tree-EDA 最优解中包含的编译选项在数目上的差异,将正相关选项在最优解中所占的比例作为度量指标 $I_{\mathrm{PC\%}}(X^*)$, 以公正地比较最优解中各选项间的正相关性. $I_{\mathrm{PC\%}}(X^*)$ 的定义如式(18)所示.式(18)中 X^+ 表示从最优解 X^* 选用的编译选项开始,运用文献[6]提出的正交数组和曼-惠特尼秩和检验相结合的方法,得到的正相关的编译选项集.

$$I_{\mathrm{PC\%}}(X^*)=\frac{|X^+|}{|X^*|} \tag{18}$$

问题 4(编译选项的使用频度): 在 GA-FP 算法对 8 个案例能耗优化结果中,各个编译选项的使用频度如何? 通过对每个编译选项使用频度的分析可以帮助人们在选用 GCC 编译选项时给出一些有用的借鉴和参考.

度量指标: 在 GA-FP 算法 m 次运行各个案例的结果中,每个编译选项 x_i 的使用频度 $I_{fq\%}(x_i)$ 作为度量指标,其定义如式(19)所示. $useNum(x_i, k, j)$ 表示 GA-FP 算法在第 k 次运行第 j 个案例所得最优解中编译选项 x_i 的使用次数. $I_{fq\%}(x_i)$ 的值越大,说明编译选项 x_i 在 8 个案例中的使用频度越高.

$$I_{fq\%}(x_i)=\sum_{j=1}^{8}\left(\frac{1}{m}\sum_{k=1}^{\mathrm{m}}useNum(x_i, k, j)\right) \tag{19}$$

5.3 使用的统计方法

考虑到演化算法的随机性,GA-FP 算法和 Tree-EDA 算法被分别独立运行 20 次,并进行统计检验.本文采用 Wilcoxom 秩和检验[19]方法对实验数据进行统计分析,并将置信水平 α 的值设置为 0.05. 为了进一步观测对比数据的差异程度(effect size),本文还使用 Vargha-Delaney[19]的 $\hat{A}_{12}$ 作为 effect size 的直观度量. $\hat{A}_{12}$ 的值域为[0,1],其值越大说明差异程度越大.

5.4 实验安装

(1) 实验环境

上位机的运行环境:Intel(R) Core(TM)i5-4590, 3.30 GHz 处理器, 8G 内存及 ubuntu-16.04.1 操作系统.

能耗评估系统:基于 STM32F4 板自主研发.

被优化案例软件的运行环境:STM32F103 板.

编译器和编译选项:GCC4.9.2,并选用与 Tree-EDA [14]相同的 58 个编译选项

(2) 算法参数的设定

为了公平起见, GA-FP 尽可能采用与 Tree-EDA 相同的参数设定.表 12 给出了 GA-FP 和 Tree-EDA 的参数设定.表 12 中的符号 NA 表示 GA-FP 或 Tree-EDA 不包含对应的参数.

Table 12 Parameter settings of Tree-EDA and GA-FP

表 12 Tree-EDA 和 GA-FP 的参数设定

参数	Tree-EDA	GA-FP
种群大小	100	100
最大演化代数	50	50
截断比率	0.5	NA
变异概率	NA	0.3
交叉概率	NA	0.6
最小支持计数	NA	10 (种群大小的 10%)

5.5 实验结果及分析

下面具体给出问题 1 和问题 2 的实验结果及分析.

(1) 问题 1(解质量)的实验结果及分析

表 13 给出了 8 个案例下-O0 等级和-O3 最优等级以及 GA-FP 和 Tree-EDA 算法 20 次运行最优解对应的能耗情况.从表 13 可以看出:对于每个案例,GA-FP 和 Tree-EDA 在平均情况和最好情况下都能获取较-O3 等级更优的编译选项集; GA-FP 在最坏情况下对于全部案例也能得到优于-O3 等级的编译选项集, 而 Tree-EDA 在 2D FIR 和 Float_Matrix 两个案例却得到劣于-O3 等级的结果. 并且对于所有 8 个案例，GA-FP 算法在平均情况、最坏情况和最好情况下都获得了较 Tree_EDA 和-O3 等级更优的编译选项集.

Table 13 The energy consumption (Unit: Joule) of the optimal solutions obtained by -O0 level, -O3 level, Tree-EDA and GA-FP by 20 runs in 8 cases

表 13 8 个案例下-O0 和-O3 最优等级以及 GA-FP 和 Tree-EDA 算法 20 次运行最优解对应的能耗 (单位:焦耳)

案例	-O0 等级	-O3 等级	TreeEDA			GA-FP		
			平均值 (较-O3 等级降低值)	最大值 (较-O3 等级降低值)	最小值 (较-O3 等级降低值)	平均值 (较-O3 等级降低值)	最大值 (较-O3 等级降低值)	最小值 (较-O3 等级降低值)
2D FIR	22.3	**18.1**	17.39 (↓0.71)	**20.4 (↑2.3)**	16.2 (↓1.9)	16.5 (↓1.6)	**16.8 (↓1.3)**	16.1 (↓2.0)
Blowfish	503.1	273.3	261 (↓12.3)	265.0 (↓8.3)	**256.1 (↓17.2)**	257.4 (↓15.9)	258.1 (↓15.2)	**254.7 (↓18.6)**
CRC32	39.7	28.2	25.95 (↓2.25)	26.2 (↓2.0)	25.6 (↓2.6)	25.2 (↓3.0)	25.6 (↓2.6)	24.9 (↓3.3)
Cubic	171.9	119.1	116.5 (↓2.6)	116.9 (↓2.2)	116.3 (↓2.8)	113.8 (↓5.3)	116.7 (↓2.4)	102.6 (↓16.5)
Dijstra	184.1	158.9	153.6 (↓5.3)	154.8 (↓4.1)	153.0 (↓5.9)	151.9 (↓7.0)	152.9 (↓6.0)	151.3 (↓7.6)
FDCT	32.4	25.1	24.1 (↓1.0)	24.5 (↓0.6)	23.8 (↓1.3)	23.75 (↓1.35)	23.8 (↓1.3)	23.6 (↓1.5)
Float_Matrix	200.1	**165.5**	162 (↓3.5)	**166.9 (↑1.4)**	155.1 (↓10.4)	153.9 (↓11.6)	**161.7 (↓3.8)**	152.7 (↓12.8)
Int_Matrix	68.9	19.3	17.47 (↓1.83)	17.6 (↓1.7)	17.4 (↓1.9)	17.35 (↓1.95)	17.5 (↓1.8)	17.1 (↓2.2)

表 14 进一步给出了 GA-FP 和 Tree-EDA 在 8 个案例下能耗改进百分比($I_{\Delta eng\%}$)的秩和检验结果. 表 14 中第 2 列 p-value 值均小于置信水平 0.05.这表明在 8 个案例下 GA-FP 的 $I_{\Delta eng\%}$ 指标在统计意义上显著优于 Tree-EDA. 从表 14 中第 3 列的 effect size 值可以看出 GA-FP 算法在 Crc32、和 Dijstra 两个案例下以概率 1 优于 Tree-EDA, 在 2D FIR、Blowfish、Cubic、FDCT、Float_Matrix 和 Int_Matrix 六个案例下以较大概率在解质量上优于 Tree-EDA. 图 7 所给出的统计盒图也从直观上得到一致结论. 从表 15 的统计量结果可知:8 个案例下的 $I_{\Delta eng\%}$ 指标平均值为 2.5%,最大 $I_{\Delta eng\%}$ 指标值可达 21.1%.

Table 14 Rank sum test results of energy consumption improvement percentage $I_{\Delta eng\%}$ of GA-FP compared with Tree-EDA under 8 cases

表 14 GA-FP 较 Tree-EDA 在 8 个案例下能耗改进百分比($I_{\Delta eng\%}$)的秩和检验结果

案例	p-value	effect size
2D FIR	2.093e-07	0.88
Blowfish	7.963e-09	0.97
CRC32	7.933e-09	**1**
Cubic	2.096e-07	0.94
Dijstra	7.992e-09	**1**
FDCT	7.846e-09	0.91
Float_Matrix	3.96e-06	0.75
Int_Matrix	0.004549	0.82

Table 15 Statistical results of energy consumption improvement percentage $I_{\Delta eng\%}$ of GA-FP compared with Tree-EDA under 8 cases

表 15 GA-FP 较 Tree-EDA 在 8 个案例下能耗改进百分比($I_{\Delta eng\%}$)的统计量结果

案例	$I_{\Delta eng\%}$ 的平均值	$I_{\Delta eng\%}$ 的最大值	$I_{\Delta eng\%}$ 的最小值	$I_{\Delta eng\%}$ 的方差
2D FIR	5.1%	**21.1%**	-5.3%	3.84e-3
Blowfish	1.4%	4.3%	-0.8%	1.2e-4

CRC32	2.9%	6.1%	0.1%	2.2 e-4
Cubic	2.3%	6.5%	-0.3%	2.9 e-4
Dijstra	1.1%	1.6%	0.4%	3.0 e-6
FDCT	1.5%	3.2%	-0.8%	1.0 e-4
Float_Matrix	5.0%	13.7%	-4.9%	2.3 e-3
Int_Matrix	0.7%	2.4%	-1.6%	1.2 e-4
平均值	**2.5%**	7.36%	-1.65%	8.74e-4

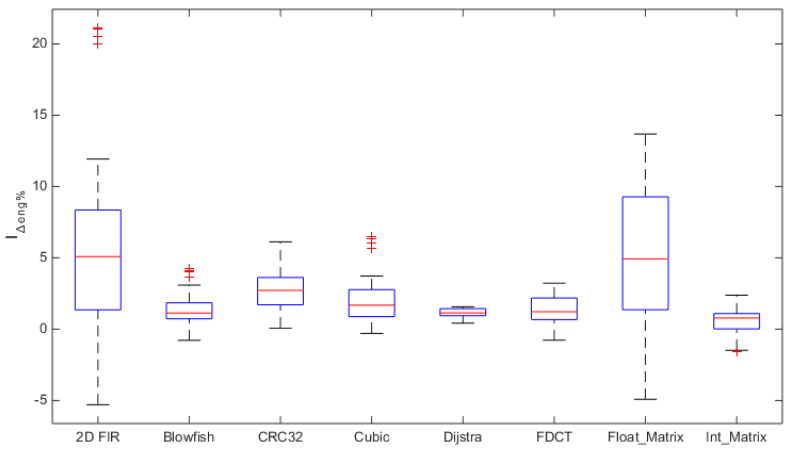

Fig.7 Boxplot using energy consumption improvement percentage $I_{\Delta eng\%}$ of GA-FP compared with Tree-EDA under 8 cases

图 7 GA-FP 较 Tree-EDA 在 8 个案例下能耗改进百分比($I_{\Delta eng\%}$)的统计盒图

(2) 问题 2(收敛速度)的结果及分析

表 16 给出了在 8 个案例下收敛速度指标 $I_{\Delta i\%}$ (GA-FP 达到不劣于 Tree-EDA 最优解对应最小迭代次数的相对减少百分比)的秩和检验结果. 表 16 中第 2 列 p-value 值均小于置信水平 0.05, 表明在 8 个案例下 GA-FP 的 $I_{\Delta i\%}$ 指标在统计意义上显著优于 Tree-EDA. 表 16 中第 3 列的 effect size 值均大于等于 0.8, 说明 GA-FP 算法较 Tree-EDA 算法在大概率上具有更快的收敛速度. 表 17 可看出:8 个案例下的 $I_{\Delta i\%}$ 指标平均值为 34.5%,最大达到了 83.3%.图 8 所给出的统计盒图也直观地得到一致结论.

Table 16 Rank sum test results of the convergence speed metric $I_{\Delta i\%}$ in 8 cases

表 16 在 8 个案例下收敛速度指标 $I_{\Delta i\%}$ (GA-FP 达到不劣于 Tree-EDA 最优解对应最小迭代次数的相对减少百分比)的秩和检验结果

案例	p-value	effect size
2D FIR	0.006799	0.8
Blowfish	7.253e-09	1
CRC32	6.71e-09	1
Cubic	7.294e-09	1
Dijstra	7.227e-09	1
FDCT	7.28e-09	1
Float_Matrix	7.294e-09	1
Int_Matrix	3.262e-05	0.9

Table 17 Statistical results of convergence speed metric $I_{\Delta i\%}$ in 8 cases

表 17 在 8 个案例下收敛速度指标 $I_{\Delta i\%}$ (GA-FP 达到不劣于 Tree-EDA 最优解对应最小迭代次数的相对减少百分比)的统计量结果

案例	$I_{\Delta i\%}$ 的平均值	$I_{\Delta i\%}$ 的最大值	$I_{\Delta i\%}$ 的最小值	$I_{\Delta i\%}$ 的方差
2D FIR	7.4%	21.2%	-9.1%	7.4e-3
Blowfish	39.7%	75.0%	20.5%	2.47e-2
CRC32	38.6%	73.8%	21.4%	2.16 e-2

Cubic	54.0%	**83.3%**	25.0%	3.53 e-2
Dijstra	34.6%	62.9%	11.4%	3.35 e-2
FDCT	42.9%	76.9%	17.9%	3.80 e-2
Float_Matrix	49.7%	76.9%	15.4%	3.54 e-2
Int_Matrix	8.9%	20.6%	-5.9%	4.76 e-3
平均值	**34.5%**	61.325%	12.075%	2.51 e-2

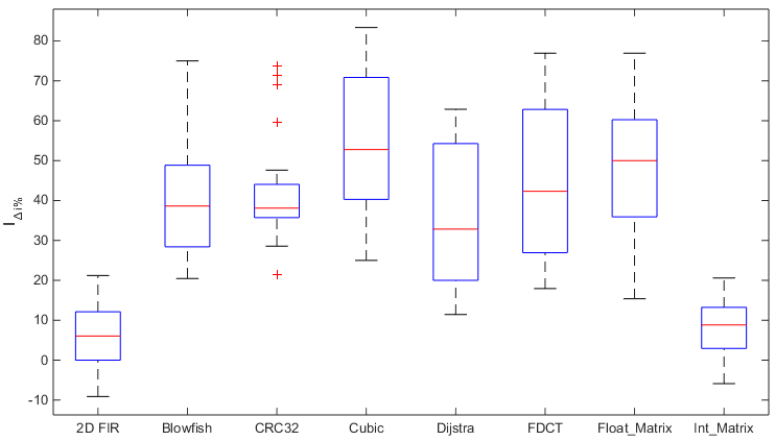

Fig.8 Boxplot using convergence speed metric $I_{\Delta i\%}$ under 8 cases

图 8 在 8 个案例下收敛速度指标 $I_{\Delta i\%}$ (GA-FP 达到不劣于 Tree-EDA 最优解对应最小迭代次数的相对减少百分比)的统计盒图

(3) 问题 3(最优解中编译选项的正相关性)的结果及分析

表 18 给出了在 8 个案例下 Tree-EDA 和 GA-FP 算法获得的最优解中编译选项正相关性指标 $I_{PC\%}$ (正相关编译选项在最优解中所占的比例)的秩和检验结果,表 18 中第 2 列 p-value 值均小于置信水平 0.05, 表明在 8 个案例下 GA-FP 的 $I_{PC\%}$ 指标在统计意义上显著优于 Tree-EDA. 除了 Blowfish、Cubic 和 Dijstra 三个案例外,表 18 中第 3 列的 effect size 值均大于 0.8. 这表明:与 Tree-EDA 算法相比,GA-FP 算法获得最优解中的各编译选项之间在大概率上具有更强的正相关性.

表 19 给出了在 8 个案例下 Tree-EDA 和 GA-FP 算法获得的最优解中编译选项正相关性指标 $I_{PC\%}$ (正相关编译选项在最优解中所占的比例)的统计结果.从表 19 可以看出:GA-FP 算法的 $I_{PC\%}$ 指标在 8 个案例下的平均值和最小值均优于 Tree-EDA 算法;仅在 Cubic 和 Blowfish 案例下, $I_{PC\%}$ 指标在最大值上 GA-FP 算分别劣于和等于 TreeEDA 算法.图 9 所给出的统计盒图也直观地得到一致结论.该实验结果说明:相比于 Tree-EDA 算法,在 GA-FP 算法获得最优解中,各编译选项之间存在更强的正相关性.进而实证了文中在频繁模式集基础上引入"增添"和"删减"两种变异算子的有效性.

Table 18 Rank sum test results of positive correlation metric $I_{PC\%}$ in the optimal solutions under 8 cases

表 18 在 8 个案例下最优解中编译选项正相关性指标 $I_{PC\%}$ 的秩和检验结果

案例	p-value	effect size
2D FIR	2.338e-06	0.821
Blowfish	**0.0001609**	**0.6969**
CRC32	1.489e-07	0.9439
Cubic	1.037e-05	**0.7999**
Dijstra	**0.0001215**	**0.6964**
FDCT	1.597e-05	0.8077
Float_Matrix	7.366e-05	0.8079
Int_Matrix	5.535e-06	0.8293

Table 19 Statistical results of positive correlation metric $I_{PC\%}$ in the optimal solutions under 8 cases

表 19 在 8 个案例下最优解中编译选项正相关性指标 $I_{PC\%}$ 的统计量结果

案例	$I_{PC\%}$ 的平均值		$I_{PC\%}$ 的最大值		$I_{PC\%}$ 的最小值		$I_{PC\%}$ 的方差	
	Tree-EDA	GA-FP	Tree-EDA	GA-FP	Tree-EDA	GA-FP	Tree-EDA	GA-FP
2D FIR	0.346	0.694	0.863	0.931	**0**	0.438	0.214	0.132
Blowfish	0.376	0.636	**0.875**	**0.875**	0.094	0.355	0.199	0.147
CRC32	0.403	0.757	0.7	0.913	0.211	0.515	0.093	0.129
Cubic	0.365	0.597	**1**	0.880	0.125	0.384	0.181	0.124
Dijstra	0.425	0.677	0.806	0.920	**0**	0.323	0.185	0.152
FDCT	0.400	0.720	0.792	0.884	0.04	0.552	0.205	0.094
Float_Matrix	0.527	0.739	0.84	0.960	0.241	0.516	0.164	0.107
Int_Matrix	0.447	0.612	0.643	0.733	0.219	0.44	0.119	0.082
平均值	0.411	0.679	0.815	0.887	0.116	0.44	0.17	0.121

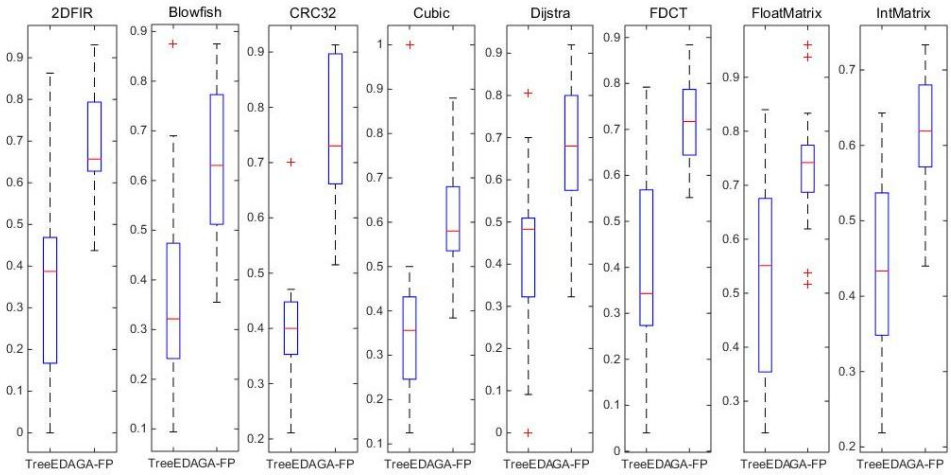

Fig.9 Boxplot using positive correlation metric $I_{PC\%}$ in the optimal solutions under 8 cases

图 9 在 8 个案例下最优解中编译选项正相关性指标 $I_{PC\%}$ 的统计盒图

(4) 问题 4 (编译选项的使用频度)的结果及分析

图 10 给出了 8 个案例在以能耗为优化目标下,每个 GCC 编译选项的使用频度情况.图 10 横轴中的数字表示编译选项的编号.而纵轴则通过不同颜色标识出不同案例下各编译选项的使用频度,并通过累加各案例中每个编译选项的使用频度得出度量指标 $I_{fq\%}(x_i)$ 的值.横轴中所有编译选项编号按照 $I_{fq\%}(x_i)$ 的值从左向右降序排列.从图 10 可以看出: 在 8 个案例中, -freorder-blocks (43 号)、-fschedule-insns2(47 号)以及-fgcse(39 号)依次为使用频度最高的 3 个编译选项;而-falign-functions\=0 (29 号)、-fno-reorder-blocks-and-partition(34 号)和-fexpensive-optimizations(38 号)依次为使用频度最低的 3 个编译选项.

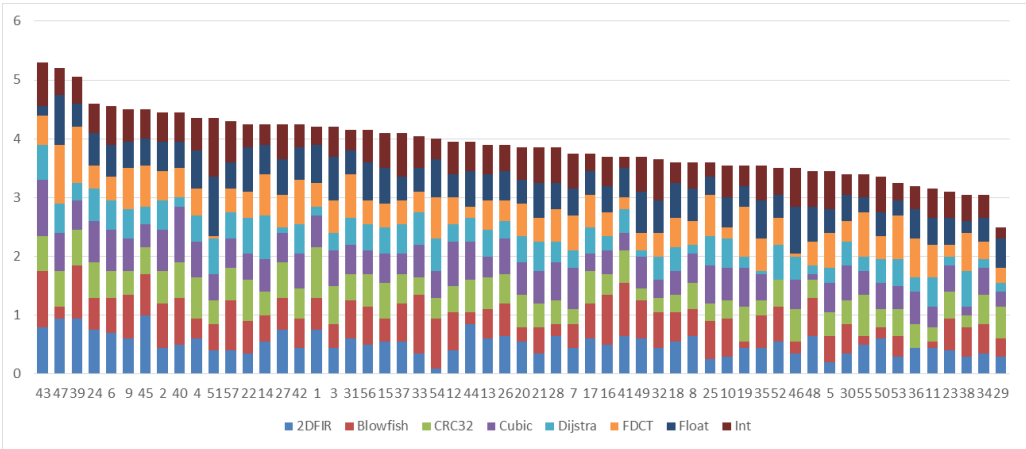

Fig.10 Histogram using the frequency metric $I_{fq\%}(x_i)$ of each compilation option in the optimal solutions under 8 cases

图 10 在 8 个案例下各编译选项使用频度指标 $I_{fq\%}(x_i)$ 的柱状图

6 总结和未来工作

本文将频繁模式挖掘和启发式变异引入到传统遗传算法,提出了一种用于 GCC 编译时能耗优化的算法 GA-FP.该算法通过考虑多个编译选项之间可能存在的相互影响,并在演化过程中使用频繁模式挖掘方法找出一组出现频度高且能耗改进幅度大的编译选项.在此基础上进一步设计了“增添”和“删减”两种新的变异算子.通过案例研究实证了 GA-FP 的解质量和收敛速度在统计意义上显著优于 Tree-EDA,最优解中编译选的相关性分析进一步验证了所设计变异算子的有效性.

本文在利用频繁模式挖掘时仅考虑对能耗有改进效果的编译选项组合, 未涉及那些对能耗改进有负影响的编译选项集合,未来工作将综合考虑这些编译选项集合进一步优化现有的算法. 另外,当前的优化算法在进行能耗评估时仍存在耗时长的问题,未来拟引入代理模型帮助解决这一问题.

References:

[1] Guo RZ, Guo J, Li M. Green Com puting and Green Embedded Systems.Computer Engineering, 2015,42(08):13 21 (in Chinese with English abstract).

[2] https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html

[3] Pallister J. Exploring the fundamental differences between compiler optimisations for energy and for performance. University of Bristol, 2016.

[4] Ashouri A H, Killian W, Cavazos J, et al. A survey on compiler autotuning using machine learning. arXiv preprint arXiv:1801.04405, 2018.

[5] Pallister J, Hollis S J, Bennett J. Identifying compiler options to minimize energy consumption for embedded platforms. The Computer Journal, 2013, 58(1): 95-109

[6] Patyk T, Hannula H, Kellomaki P, et al. Energy consumption reduction by automatic selection of compiler options. International Symposium on Signals, Circuits and Systems. IEEE, 2009:1-4.

[7] Boussaa M, Barais O, Baudry B, et al. Notice: A framework for non-functional testing of compilers. Software Quality, Reliability and Security (QRS), 2016 IEEE International Conference on. IEEE, 2016: 335-346.

[8] A. M. Malik, “Spatial based feature generation for machine learning based optimization compilation,” in 2010 Ninth International Conference on Machine Learning and Applications, 2010, pp. 925–930.

[9] Li F, Tang F, Shen Y. Feature Mining for Machine Learning Based Compilation Optimization. Eighth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing. IEEE, 2014:207-214.

[10] K. Hoste and L. Eeckhout. COLE: compiler optimization level exploration. In Proceedings of the 6th annual IEEE/ACM international symposium on Code generation and optimization, pages 165–174. ACM, 2008.

[11] Sandran T, Zakaria N, Pal A J. An optimized tuning of genetic algorithm parameters in compiler flag selection based on compilation and execution duration. Proceedings of the International Conference on Soft Computing for Problem Solving (SocProS 2011) December 20-22, 2011. Springer, India, 2012: 599-610.

[12] Lin S C, Chang C K, Lin N W. Automatic selection of GCC optimization options using a gene weighted genetic algorithm. Computer Systems Architecture Conference, 2008. ACSAC 2008. Asia-Pacific. IEEE, 2008:1-8.

[13] Nagiub M, Farag W. Automatic selection of compiler options using genetic techniques for embedded software design. Computational Intelligence and Informatics (CINTI), 2013 IEEE 14th International Symposium on. IEEE, 2013: 69-74.

[14] Garciarena U, Santana R. Evolutionary optimization of compiler flag selection by learning and exploiting flags interactions. Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion. ACM, 2016: 1159-1166.

[15] Mühlenbein H, Paass G. From recombination of genes to the estimation of distributions I. Binary parameters. International conference on parallel problem solving from nature. Springer, Berlin, Heidelberg, 1996: 178-187.

[16] Holland J H. Genetic algorithms[J]. Scientific american, 1992, 267(1): 66-73.

[17] Han J, Pei J, Yin Y. Mining frequent patterns without candidate generation. ACM sigmod record. ACM, 2000, 29(2): 1-12.

[18] Pallister J, Hollis S, Bennett J. BEEBS: Open benchmarks for energy measurements on embedded platforms. arXiv preprint arXiv:1308.5174, 2013.

[19] Arcuri A, Briand L. A practical guide for using statistical tests to assess randomized algorithms in software engineering. Software Engineering (ICSE), 2011 33rd International Conference on. IEEE, 2011: 1-10.

附中文参考文献:

[1] 郭荣佐,郭进,黎明.绿色计算与绿色嵌入式系统.计算机科学,2015,42(08):13-21.