Інтелектуальні алгоритми для побудови арифметики монолітних кодів

К.т.н., доц. Ю. Цимбал, к.ф.-м.н., доц. Ю. Кинаш, маг. А. Медвідь, маг. Ю. Бурда

Lviv Polytechnic National University, 28a/804 Bandera St., Lviv-13, 79013, Ukraine
E-mail: yurij.tsymbal@gmail.com

Abstract. An algorithm for arithmetic operations (addition) in monolithic code has been developed using genetic programming. The results of its application for monolithic code with 5-10 bits have been shown in this paper.

Key words: arithmetic operations, monolithic code, genetic programming.

Вступ

Стрімкий розвиток інформаційних технологій пов'язаний із низкою проблем, що потребують нових, якісно кращих рішень. До таких проблем, зокрема, належать передача даних та їх опрацювання з використанням завадостійкого кодування.

Серед новітніх ідей для реалізації завадостійкого кодування є використання монолітних кодів, заснованих на комбінаторній структурі – ідеальній кільцевій в'язанці (ІКВ) [1, 2].

Великою перевагою у впровадженні монолітних кодів для передачі інформації була б можливість проведення арифметичних операцій безпосередньо над закодованими даними. На даний час такі ефективні алгоритми реалізації цих операцій ще не розроблені, що обумовлено складною, невпорядкованою основою коду.

Постановка задачі

Ідеальна кільцева в'язанка (ІКВ) – це циклічна послідовність n чисел $\{k_1, k_2, \ldots, k_n\}$ на якій всі можливі кільцеві суми вичерпують значення чисел натурального ряду від 1 до $S_n = n * (n–1) + 1$ [1].

Наприклад, циклічна послідовність {1, 2, 6, 4}, є ІКВ з $n = 4$, оскільки її елементи перелічують всі числа натурального ряду від 1 до $n * (n–1) + 1 = 13$, утворених рівно одним способом обрання початкового та останнього елементів цієї послідовності.

Монолітний код на ІКВ – це множина кодових комбінацій, у яких всі одиничні та нульові символи розділені між собою щонайбільше на дві групи різнойменних символів на кільцевій послідовності розрядів, ваги яких пропорційні числам обраної ІКВ.

Для прикладу візьмемо за основу ІКВ довжиною 4: {1, 2, 6, 4}. Тоді число 0 можна зобразити у монолітному коді як "0000", число 3 – "1100", число 4 – "0001", число 5 – "1001", число 13 – "1111". Якщо ж за основу взяти в'язанку {1, 3, 2, 7}, то числу 0 так само відповідатиме код "0000" і числу 13 – "1111", тоді як числу 3 – "0100", числу 4 –"1100" і числу 5 – "0110". Проведення арифметичних операцій у таких кодах, на відміну від класичного двійкового коду, не є очевидним і потребує докладного дослідження. В першу чергу слід розглянути операцію додавання як базову для інших арифметичних операцій.

Основні властивості монолітних та схожих на них кодів

Представлення чисел у монолітних кодах полягає у зображенні числа як суми ваг послідовного набору розрядів. Виходячи із властивостей простих ІКВ, що лежать в основі монолітного коду кожне натуральне число на проміжку від 1 до $n * (n–1) + 1$ можна зобразити рівно одним монолітним кодом (без розривів у ланцюжку одиниць або нулів). Тобто натуральні числа однозначно кодуються і декодуються монолітним кодом.

Монолітний двійковий код має певні переваги перед іншими кодами. Одна з них – простота виявлення та виправлення помилок на приймальній стороні. Поява хоча б одного символу "1" серед нулів або символу "0" серед одиниць у прийнятій кодовій комбінації свідчить про помилку. Помилка не виявляється лише коли хибний сигнал виникає в першому або останньому символах пакета (на межі між ланцюжками нулів та одиниць). Якщо в монолітному коді з'являються хибні символи, то всі вони або частина з них одразу ж виявляються, що спрощує виявлення помилок і забезпечує високу завадостійкість монолітного коду [1].

Монолітні коди як засіб представлення числової інформації є схожими на класичні двійкові коди і т.зв. коди Фібоначчі [3, 4], для яких ґрунтовно розроблені правила виконання базових арифметичних операцій, у першу чергу, додавання.

Якщо порівняти принцип організації монолітного коду з двійковим кодом, коди є схожі у тому, що вага кожного розряду строго визначена наперед. Проте у двійковому коді є сталим співвідношення між вагами сусідніх розрядів. У монолітному коді це співвідношення є різним для кожної пари сусідніх розрядів. У цьому монолітний код є схожим до коду Фібоначчі.

Код Фібоначчі – це код, у якому ваги розрядів рівні відповідним значенням послідовності Фібоначчі {1, 2, 3, 5, 8, ...}.

За теоремою Цекендорфа [3, 4] доведено, що будь-яке невід'ємне ціле число єдиним чином можна зобразити як суму деякого набору чисел Фібоначчі, притому, що набір не міститиме сусідніх чисел Фібоначчі.

Арифметичні операції у двійкових кодах та у коді Фібоначчі можна зобразити у вигляді послідовності перенесень чисел з одних розрядів у інші. Коли у певному розряді двійкового коду в результаті додавання значення перевищує одиницю, відбувається перенесення "двійки" із цього розряду в старший, поки у коді не залишаться лише "нулі" та "одиниці".

Наприклад, додавання десяткових чисел 21 і 27 у двійковому коді можна умовно зобразити як

$$21_{10} + 27_{10} = 010101_2 + 011011_2$$

$$= 021112 = 101120 = 101200 = 102000 = 110000 = 48_{10}$$

Код Фібоначчі має свої правила перенесення. Зокрема, двійка у одному з розрядів розбивається на одиницю в наступному старшому розряді і одиницю в молодшому на 2 розряді: 0200 = 1001, тобто потрібно перенесення як вправо так і вліво. А дві одиниці в сусідніх розрядах перетворюються в одиницю у наступному старшому розряді: 011 = 100.

Для монолітних кодів вводяться свої правила перенесення. Якщо розглянути в'язанку {1, 3, 10, 2, 5} і пронумерувати розряди від 1, починаючи з крайнього лівого, можна описати такі перетворення: з 1-го розряду двійку перенести як одиницю у 4-й розряд, одиниці з 1-го та 4-го розряду перенести як одну одиницю у другий розряд і т.д.

Отже, код Фібоначчі можна вважати узагальненням двійкових кодів [3, 4]. В свою чергу коди Фібоначчі належать до великого класу кодів, у яких кожен розряд має свою, визначену наперед вагу. Одними з представників таких кодів є також монолітні коди, засновані на ІКВ. Їхня структура є більш загальною, оскільки не існує чіткого зв'язку між вагами сусідніх розрядів. З огляду на це для різних ідеальних кільцевих в'язанок правила арифметики відповідних монолітних кодів будуть різними. Виконання арифметичних операцій у монолітних кодах загалом є складнішим порівняно з двійковими кодами та кодом Фібоначчі.

Послідовність дій для формування алгоритму додавання двох чисел у монолітному коді поділимо на 2 етапи.

1. Формування набору елементарних перетворень (переходів або мікроопераций) для обраної ІКВ.

2. Генерування послідовності елементарних перетворень, які дозволять для приведення до монолітного вигляду суми двох чисел, поданих у монолітному коді.

Аналіз методів досліджень

Для розв'язання поставленої задачі можна використати детерміновані підходи, зокрема, перебір з поверненням і метод гілок та границь, або стохастичні підходи на основі певних евристик.

Ідея досліджень полягає у використанні евристичних, зокрема, генетичних алгоритмів [5] для відтворення послідовності виконання елементарних перенесень або перетворень, які дозволять привести суму двох чисел у монолітних кодах до монолітного вигляду.

Попередньо потрібно сформувати набір елементарних перетворень для обраного монолітного коду. Надалі генетичний алгоритм оперує з об'єктами, що містять послідовність таких елементарних перетворень, результатом його роботи повинна бути така послідовність, що перетворює суму у монолітний код.

Оцінити ефективність генетичного алгоритму для розв'язання цієї задачі можна у порівнянні з результатами застосування детермінованих алгоритмів (повного перебору або методу гілок та границь).

Очікуваним результатом роботи детермінованого алгоритму буде оптимальна послідовність, яка приведе код до монолітного вигляду. Але, з огляду на експоненційну складність такого алгоритму, час роботи буде зростати надто швидко і стане неефективним на великих вхідних даних (при ІКВ великої розмірності).

У свою чергу генетичний алгоритм буде знаходити шукану послідовність операцій лише з деякою ймовірністю, причому зі збільшенням розмірності ІКВ ймовірність поступово падатиме. Але час роботи алгоритму зростатиме повільніше, ніж час роботи алгоритму повного перебору оскільки залежність часу роботи генетичного алгоритму від розміру вхідних даних має поліноміальний характер. Тобто генетичний алгоритм може виявитись більш ефективним за перебір варіантів на великих вхідних даних.

Опис результатів досліджень

Для реалізації генетичного алгоритму для додавання чисел у монолітному коді слід розв'язати низку підзадач.

1. Формування набору правил перетворення (мікроопераций) для заданої ІКВ. Будь-яка послідовність мікроопераций вважатиметься кандидатом у розв'язок задачі.

2. Генерування початкового набору кандидатів у розв'язок задачі (генетичних кодів для індивідів початкової популяції).

3. Повторення для заданої кількості поколінь або генерацій:

а) вибору індивідів з поточної популяції (селекції);

б) змін у популяції (мутації та схрещення);

в) сортування популяції за значенням цільової функції (фітнесом);

г) відбору індивідів для формування нової популяції.

Оскільки розв'язком задачі додавання чисел у монолітному коді є послідовність мікроопераций, задача може бути вирішена засобами генетичного програмування [6].

Суть мутації полягає у тому, що у генетичному коді обраного індивіда у одній чи кількох позиціях значення комірки заміняється на будь-яке інше. Схрещенням називають процес обміну двох індивідів ділянками генетичного коду. Застосування цих операцій має стимулювати оновлення генетичного коду популяції і запобігати виродженню (витісненню кількома індивідами решти індивідів у наступних поколіннях).

Для відбору найкращих індивідів при переході до наступних поколінь слід розробити цільову функцію, значення якої використовується для порівняння індивідів між собою. В якості цільової функції запропоновано обрати різницю між відстанню за Хеммінгом від початкового коду до монолітного та середнім значенням відстані до монолітного коду після кожного кроку перетворення. Від'ємне значення цільової функції означатиме, що під час перетворень код віддаляється від монолітного, додатне значення буде вказувати на наближення до монолітності.

На якість роботи генетичного алгоритму впливає багато факторів. Зокрема, це розмір популяції, вибір індивідів для мутації та схрещення, реалізація схре-

щення, співвідношення між кількістю мутацій та схрещень для одного покоління, вигляд цільової функції кількість нових індивідів, які переходять у наступне покоління і т. д.

Щоб запобігти виділенню надвеликих обсягів пам'яті та значного сповільнення роботи алгоритму вирішено не змінювати розмір популяції від покоління до покоління.

Через велику кількість параметрів у налаштуваннях алгоритму та недостатню інформацію про залежність параметрів один від одного, задача оптимізації генетичного алгоритму потребує окремого ґрунтовного дослідження, можливо теж за допомогою евристичних алгоритмів.

Крім того можливо доцільним є введення додаткових параметрів до генетичного алгоритму, наприклад, довжини т.зв. префіксу послідовності (початкової підпослідовності, що складається лише з коректних перетворень). При введенні таких параметрів слід проводити додатковий аналіз того, як ці параметри у поєднанні з уже існуючими будуть впливати на перебіг роботи генетичного алгоритму. Введення нового параметру може з одного боку додавати нові знання про об'єкти, якими ми оперуємо, а з іншого боку приховано збільшити ймовірність виродження популяції чи призвести до ігнорування перспективних індивідів.

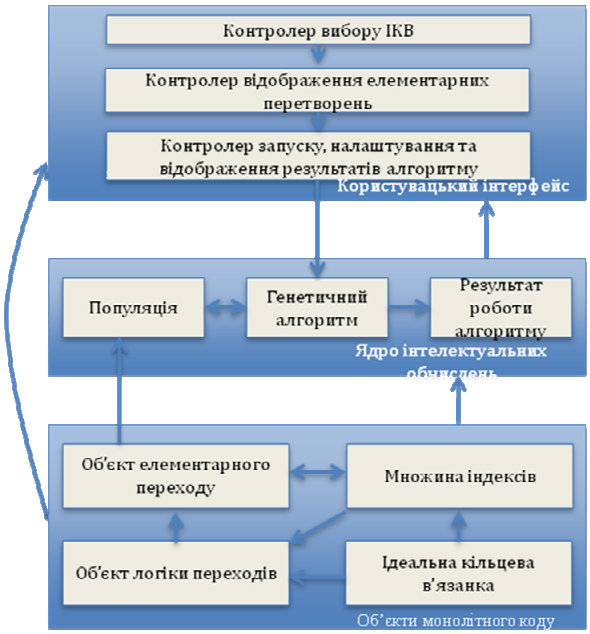

Рис. 1. Структурна схема програмної системи для побудови арифметики монолітних кодів з використанням генетичного програмування

Для реалізації перебору з поверненнями та генетичного алгоритму для додавання у монолітному коді створено програмну систему, різні версії якої працюють у середовищах Silverlight, iOS та MacOS (рис. 1).

Інтерфейс програми є інтуїтивно зрозумілим і подає результати роботи у зручній для користувача формі. Користувач має можливість вибору кільцевої в'язанки та налаштування деяких параметрів генетичного алгоритму (рис. 2).

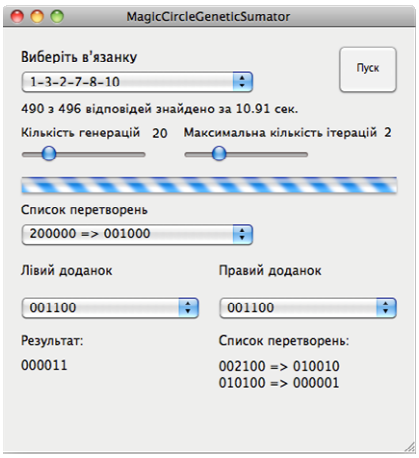

Рис. 2. Головне вікно програми (версія для MacOS)

Висновки

Розроблено алгоритм та створено програмну систему для виконання операції додавання у монолітному коді за допомогою генетичного програмування і методу гілок та границь.

Проведені експерименти показують, що використання генетичного алгоритму для розв'язання задач такого класу є ефективнішим за часом за використання детермінованих алгоритмів. З іншого боку через стохастичну природу генетичний алгоритм не завжди гарантує отримання оптимального розв'язку задачі на відміну від методів перебору варіантів.

Підтверджено, що використання інтелектуальних алгоритмів, зокрема, генетичних, є досить перспективним підходом для отримання за поліноміальний час розв'язку задачі з неполіноміальною складністю (NP-повної задачі).

Зі схожості монолітного коду до двійкових кодів та коду Фібоначчі випливає, що якщо у генетичному алгоритмі використати перетворення притаманні певному коду і задати відповідні умови коректності кодових комбінацій, то алгоритм можна ефективно адаптувати для кожного з названих кодів.

Більш складною задачею є задача проведення на монолітному коді таких операцій як множення та ділення. Її вирішення дозволить повною мірою впроваджувати монолітні коди, як водночас завадостійкі та арифметичні.

[1]. Різник В.В. Синтез оптимальних комбінаторних систем. – Львів: Вища школа. Вид-во при Львів. ун-ті, 1989. – 168 с.

[2]. Наукова школа професора Володимира Різника http://iknit.lp.edu.ua/riznyk.

[3]. Стахов А.П. Коды золотой пропорции. – М.: Радио и свіязь, 1984. – 152 с., ил. – (Кибернетика).

[4]. Stakhov A. The Mathematics of Harmony. From Euclid to Contemporary Mathematics and Computer Science. – World Scientific Publishing Co. Pte. Ltd, 2009. – 738 p.

[5]. Емельянов В.В., Курейчик В.В., Курейчик В.М. Теория и практика эволюционного моделирования. – М: Физматлит, 2003. – 432 с.

[6]. Koza J.R. Genetic programming: On the Programming of Computers by Means of Natural Selection. – A Bradford Book, 1992. – 840 p.