31° Encontro Anual de Iniciação Científica
11° Encontro Anual de Iniciação Científica Júnior
31° EAIC 11° EAIC JR
10 e 11 de novembro de 2022

GRAFOS DA REPRESENTAÇÃO INTERMEDIÁRIA COMO REPRESENTAÇÃO DE PROGRAMAS

Nathalia Mesquita Carnevalli (PIBIC/CNPq/FA/Uem), Anderson Faustino da Silva (Orientador), e-mail: ra116086@uem.br.

Universidade Estadual de Maringá / Centro de Tecnologia / Maringá, PR.

Ciência da Computação / Sistema de Computação

Palavras-chave: grafos, aprendizagem de máquina, representação intermediária

Resumo:

A aprendizagem de máquina tem sido utilizada em diversos contextos, tais como: sistemas inteligentes de geração de código, descoberta de clones de programas, classificação de algoritmos entre outros. Neste contexto, uma questão essencial é a modelagem do conhecimento, mais especificamente, a modelagem – ou representação – de programas. Este trabalho tem por objetivo analisar diferentes representações de programas, baseadas em grafo, as quais são provenientes da representação intermediária do programa. Desta forma, se tem por objetivo final identificar qual a melhor representação na solução do problema de escolha do dispositivo de execução de um programa. Os resultados demonstraram que a representação programl é uma boa opção.

Introdução

Uma área que nos últimos anos aumentou o uso de aprendizagem de máquina é a área de linguagens de programação. Nesta área problemas como: geração de código, detecção de clones, classificação de algoritmos e escolha do dispositivo de execução empregam aprendizagem de máquina. Tais sistemas utilizam diferentes mecanismos para representar programas, como: histograma de instruções, sequências de instruções, vetores numéricos que fornecem propriedades do programa, ou ainda grafos.

O grafo é uma representação que indica a relação entre propriedades, e é formado por vértices e arestas. No contexto em questão, os vértices indicam as instruções do programa e as arestas as relações entre elas, ou seja, o fluxo de controle e/ou o fluxo de dados. Neste trabalho, o interesse é representar programas utilizando a

representação intermediária da infraestrutura LLVM (Lattner e Adve, 2004). A escolha desta infraestrutura de compilação se dá pelo fato de sua larga utilização, na geração de código, tanto no meio comercial como no meio acadêmico. Basicamente, um programa escrito em Linguagem C é transformado em *bitcode*, representação intermediária LLVM, o qual é otimizado e então transformado em código Assembly.

Neste contexto, este trabalho investiga o uso de grafo, como representação de programas, na solução do problema da escolha do dispositivo de execução. Tal problema é formulado da seguinte forma: dado um programa em qual dispositivo este terá o menor tempo de execução, mais precisamente, um programa deve ser executado na CPU ou na GPU para obter o menor tempo de execução. Para alcançar tal objetivo três passos são realizados: (1) extração de diferentes grafos, para representar o mesmo programa; (2) criação de um modelo de redes neurais para aprender escolher o dispositivo; e (3) avaliação do modelo mediante o uso de diferentes grafos.

Materiais e métodos

Os materiais da pesquisa constituem-se da infraestrutura ComPy-Learn (Brauckmann e Outros, 2020) e um conjunto de programas extraídos de diferentes benchmarks. ComPy-Learn foi idealizada como uma infraestrutura para a exploração de representações de programas e o aprendizado de máquina em tarefas relacionadas à programas. Portanto é possível utilizar tal infraestrutura para modelar diferentes problemas, e diferentes representações de programas.

Como mencionado anteriormente, utilizamos o problema da seleção do dispositivo de execução e representação baseada em grafo. Neste último quesito avaliamos três grafos distintos: (1) cfg+call, (2) cdfg+call e (3) programl. O primeiro representa o programa utilizando suas instruções e o fluxo de controle entre elas. Adicionalmente, tal grafo possui arestas de chamadas de funções, ocasionando a união dos grafos que compõem as funções do programa. O segundo adiciona ao primeiro, vértices que representam dados, além de arestas que representam fluxo de dados. Por fim, pode-se dizer que o último grafo é um cdfg+call com mais detalhes sobre os dados do programa. O segundo não faz distinção entre tipos de dados, por exemplo variável e constante, enquanto o terceiro faz tal distinção.

Resultados e Discussão

Para a execução do experimento, a rede neural (Mou e Outros, 2016) é executada 10 vezes em 100 épocas, para cada tipo de grafo. O objetivo é observar a

acurácia da rede, quando a representação do programa é modificada. Desta forma, quanto maior a acurácia melhor a representação se mostra. O conjunto de dados é formado por 170 programas, os quais são divididos em treino e teste utilizando validação cruzada com 10 agrupamentos.

A Figura 1 apresenta os resultados obtidos por cfg+call. Este grafo apresentou variação média da acurácia no treinamento de 60%, no teste de 62%. Além disso, a acurácia de teste estagnada na 51ª época.

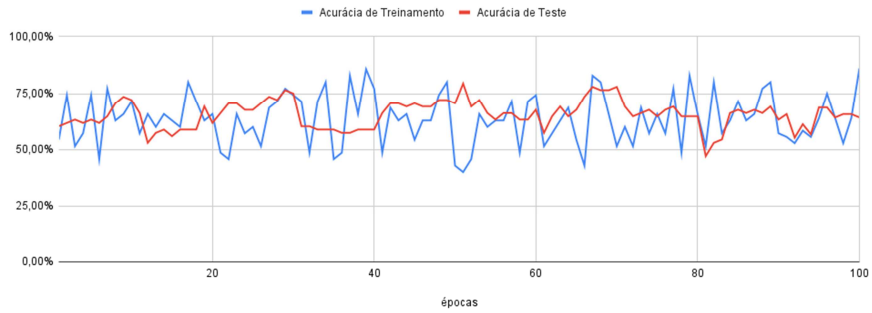

Figura 1 - grafo cfg+call

A Figura 2 apresenta os resultados obtidos por cdfg+call. Este grafo apresentou variação média da acurácia no treinamento de 62%, no teste de 65%. Além disso, a acurácia de teste estagnada na 51ª época.

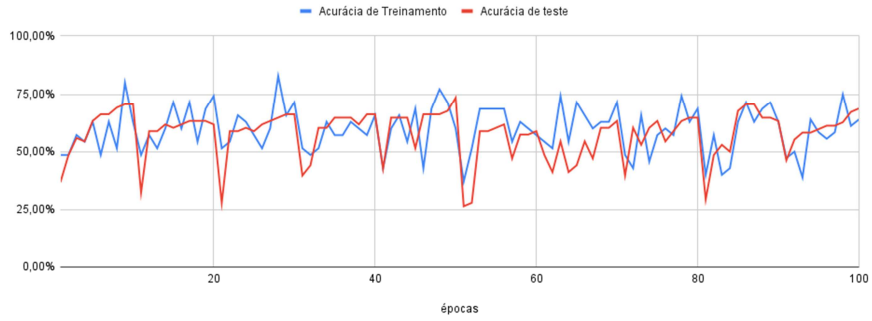

Figura 2 - grafo cdfg+call

A Figura 3 apresenta os resultados obtidos por programl. Este grafo apresentou variação média da acurácia no treinamento de 67%, no teste de 68%. Além disso, a acurácia de teste estagnada na 50ª época.

Figura 3 - grafo programl

Conclusões

Este trabalho apresentou uma avaliação de diferentes tipos de grafo, no contexto do problema de seleção do dispositivo de execução. Mediante experimentos ficou evidenciado que a representação programl foi a que apresentou a melhor acurácia.

Agradecimentos

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) pelo financiamento concedido para a realização deste estudo.

Referências

Alexander Brauckmann; Andrés Goens; Jeronimo Castrillon, "ComPy-Learn: A toolbox for exploring machine learning representations for compilers," 2020 Forum for Specification and Design Languages (FDL), 2020.

Chris Lattner e Vikram S. Adve, LLVM: a compilation framework for lifelong program analysis & transformation, International Symposium on Code Generation and Optimization, 2004. CGO 2004.

Lili Mou, Ge Li, Lu Zhang, Tao Wang, Zhi Jin. Convolutional Neural Networks over Tree Structures for Programming Language Processing. In AAAI, 2016.