Formalização e Verificação em Coq do Algoritmo de Dijkstra

João Vitor Fröhlich[1], **Karina Girardi Roggia**[2], **Paulo Henrique Torrens**[3]

[1]Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brazil

[2]Universidade do Estado de Santa Catarina (UDESC)
Joinville – SC – Brazil

[3]University of Kent
Canterbury – Kent – United Kingdom

jvfrohlich@ufmg.br, karina.roggia@udesc.br, paulotorrens@gnu.org

Abstract. *A proof assistant is a software tool that assists in the development of formal proofs. Among the capabilities of a proof assistant, such as Coq, it is possible to model complex structures, in particular software, that may be represented through proofs. Among such structures, we can highlight graphs, but little research has been carried out on formalizing the implementation for the different algorithms that solve problems in graph theory. Thus, contributing to the expansion of the use of proof assistants in verifying the correctness of algorithms in graph theory, this work implements and shows a partial proof of correctness for Dijkstra's algorithm in Coq.*

Resumo. *Um assistente de provas é uma ferramenta de software que auxilia no desenvolvimento de provas formais. Dentre as capacidades de um assistente de provas, como o Coq, é possível modelar estruturas complexas, em particular aplicações, que podem ser representadas por meio de provas. Entre essas estruturas, podemos destacar os grafos, contudo não foram encontradas muitas pesquisas sobre a formalização da implementação dos diversos algoritmos que resolvem problemas da teoria de grafos. Assim, para contribuir com a expansão do uso de assistentes de provas na verificação da corretude de algoritmos da teoria de grafos, este trabalho implementa em Coq o algoritmo de Dijkstra e apresenta uma prova parcial de corretude dessa implementação.*

1. Introdução

Em computação, é importante que algoritmos tenham a garantia que os programas que os implementam produzam o resultado correto e que terminem a sua execução. Para garantir estas propriedades, é possível definir uma especificação formal acerca delas, as quais, então, podem ser provadas utilizando métodos matemáticos rigorosos. Como as provas de corretude de algumas aplicações podem ser muito extensas e complexas, se justifica o uso de um assistente de provas para a verificação de tais por um computador.

Grafos são estruturas matemáticas que podem ser utilizadas para modelar e resolver diversos problemas. Alguns exemplos de uso de assistentes de provas em grafos podem ser encontrados na literatura, sendo um dos mais famosos a prova do teorema das quatro cores [Gonthier 2008], que foi provado inicialmente por [Appel and Haken 1976],

utilizando diversos assistentes de provas separados e, também, por [Gonthier 2023], usando o assistente *Coq*. Porém, pouca pesquisa em grafos foi desenvolvida utilizando os assistentes de provas, tanto na área teórica quanto na área prática. Desta forma, este trabalho visa ampliar o uso do assistente de provas *Coq* nesta área, por meio da implementação e prova do algoritmo de Dijkstra.

Alguns trabalhos exploraram a utilização de assistentes de prova para realizar a implementação e prova do algoritmo de Dijkstra, como em [Mohan et al. 2021], [Nordhoff and Lammich 2012], [Mange and Kuhn 2007] e [Moore and Zhang 2005], porém, ao conhecimento dos autores, nenhum destes trabalhos realizou tanto a prova quanto a implementação em uma forma construtiva do algoritmo no assistente de provas *Coq*, o que permitiria a extração deste código para outras linguagens, como *Haskell*, *Scheme* ou *OCaml*.

Utilizando uma abordagem de representação de código imperativo por meio de um grafo de fluxo de controle, este trabalho realiza a extração de um código funcional que foi implementado em *Gallina*, a linguagem de implementação do provador *Coq*. A partir disso, a especificação formal em *Coq* do algoritmo foi elaborada e parcialmente provada. Este trabalho foi parcialmente financiado pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq (Processo 409707/2022-8) e pela Fundação de Amparo à Pesquisa e Inovação do Estado de Santa Catarina – FAPESC.

2. Assistente de Provas *Coq*

Para auxiliar no processo de provas matemáticas, incluindo a prova da corretude de algoritmos, é possível utilizar assistentes de provas para organizar e mecanizar as etapas das provas. Dentre estes assistentes, um dos mais conhecidos é o *Coq*, cujo núcleo implementa o Cálculo de Construções (Co)Indutivas [Bertot and Castéran 2013], e conta com muitas contribuições da sua comunidade de usuários. Uma de suas principais características é o uso da linguagem de programação e especificação *Gallina*, de paradigma puramente funcional, cujas propriedades garantem que todo programa ou prova escrito nesta linguagem sempre termine [Barras et al. 1999].

Uma vez que a linguagem *Gallina* é definida dentro do paradigma puramente funcional, a implementação convencional do algoritmo de Dijkstra, descrita de forma imperativa, não pode ser utilizada sem ser adaptada de alguma forma. Para isso, foi possível se utilizar como inspiração a correspondência entre as representações intermediárias SSA (do inglês, *Static Single-Assignment*) e ANF (do inglês, *A-Normal Form*), conforme descrito por [Appel 1998]. A ideia consiste em separar o algoritmo imperativo em blocos básicos, que podem ser representados por meio de funções puras definidas separadamente. A representação intermediária de SSA representa o código como um grafo de fluxo de controle no qual uma variável pode ter um valor atribuído a ela apenas uma vez.

3. Implementação e Especificação Formal

Para a implementação do algoritmo de Dijkstra, será considerado o pseudocódigo da Figura 1. Neste pseudocódigo, Q representa a fila de vértices a serem visitados, enquanto d representa uma lista que associa um vértice v a um peso, que é o limite superior do peso do menor caminho entre a origem e v. A função de relaxamento de arestas é realizada entre as linhas 4 e 7, enquanto o percorrimento da fila é realizado entre as linhas 1 e 3.

1: **Enquanto** Q não está vazia **Faça**
2: $u \leftarrow$ vértice em Q com menor $d(u)$
3: remove(Q, u)
4: **Para todo** vizinho v de u que esteja presente em Q **Faça**
5: $alt \leftarrow d(u) + weight(u, v)$
6: **Se** $alt < d(v)$ **Então**
7: $d(v) \leftarrow alt$
8: **Retorne** d, p

Figura 1. Algoritmo de Dijkstra

A partir deste algoritmo, é possível fazer a representação intermediária utilizando o grafo de fluxo de controle mostrado na Figura 2. Neste grafo, os blocos b_0 e b_1 representam o percorrimento da fila, enquanto os blocos b_2, b_3 e b_4 representam o processo de relaxamento de arestas e o bloco b_5 representa o retorno da função. Apesar de não estar na forma de SSA, a representação demonstrada é suficiente para os fins deste trabalho.

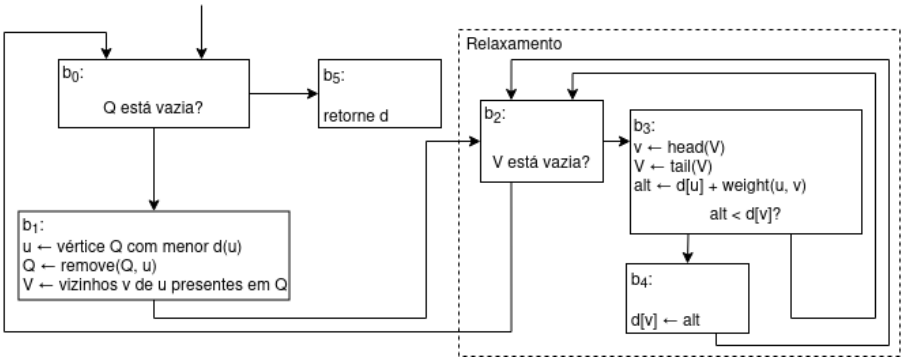

Figura 2. Grafo de Fluxo de Controle do Algoritmo de Dijkstra

Além de ter um código representado em paradigma funcional, é preciso que a execução deste código sempre termine. Para a implementação da função de relaxamento, o *Coq* é capaz de inferir o argumento que converge para o caso base na recursão. Contudo, para a implementação do percorrimento da fila, é preciso deixar explícito que a função remove(Q,u) sempre remove o vértice u da fila, fazendo com que o tamanho da lista Q convirja para o caso base. Para deixar o parâmetro decrescente explícito ao declarar a função b_0, é utilizado o comando *Program Fixpoint*, juntamente com um argumento *measure*, que indica que o parâmetro decrescente é o tamanho da fila Q. Após a declaração da função, será necessário provar que a fila Q decresce a cada chamada de b_0.

Um caminho é definido indutivamente como uma conexão entre dois vértices com um dado peso. Dessa forma, dizemos que existe um caminho em três casos distintos: de um vértice para ele mesmo com peso 0; entre dois vértices, caso exista uma aresta ligando estes vértices diretamente, com o peso dessa aresta; ou por meio de transitividade entre dois caminhos, onde se existe um caminho de um vértice t a um vértice u com peso w_1, e do vértice u a um vértice v com peso w_2, então existe um caminho do vértice t ao vértice v com peso $w_1 + w_2$. Outras definições importantes são as definições de menor caminho, tal que existe um menor caminho de u pra v com peso w se não existe nenhum outro caminho entre estes vértices com peso menor que w, e a definição de alcançabilidade, onde um vértice u alcança v se existe um menor caminho com um peso w entre eles.

Além dos caminhos, são definidos outros dois conceitos, estes relacionados à lista associativa de distâncias: a sanidade, que diz que se existe uma associação de um peso w a um vértice v, então existe um caminho da origem para v com peso w; e a definição da invariante do algoritmo, que diz que para todo vértice do grafo, ou esse vértice não foi visitado pela busca, ou se ele é alcançável a partir da origem, então existe uma distância associada a este vértice que é igual à distância do menor caminho entre a origem e este vértice.

Para provar a corretude do algoritmo de Dijkstra tem-se o teorema principal de acordo com o qual para todo vértice v e peso w, se existe um menor caminho da origem para v com este peso w, então w está associado a v na lista associativa retornada pelo algoritmo. Para auxiliar na prova deste teorema, são enunciados alguns lemas, nos quais se destacam: a preservação da invariante, tanto pela chamada da função b_0 quanto pela chamada da função b_2, e a preservação da sanidade pela chamada da função b_2. Se estes teoremas e lemas forem provados, então é provada a corretude do algoritmo de Dijkstra, visto que é garantido que o algoritmo sempre para, por conta das propriedades da própria linguagem do assistente *Coq*, e que ele respeita uma especificação formal que atesta que ele produz o menor caminho.

4. Prova de Corretude

As ideias apresentadas para a implementação do algoritmo foram replicadas em *Coq*[1], e o resultado foi um algoritmo executável e, pelas exigências da linguagem *Gallina*, é certo que as execuções deste algoritmo sempre terminam. As provas dos lemas da preservação da sanidade e invariância pela aplicação da função de relaxamento ainda não foram concluídas e estão definidas parcialmente. Contudo, pelos avanços realizados na prova destes resultados, há evidências para se argumentar a favor da corretude destas propriedades. Por outro lado, as provas sobre a função b_0 e a função de Dijkstra, utilizando os lemas não provados acerca do relaxamento, foram concluídas. Assim, a prova da corretude do algoritmo, ao momento da escrita, só depende da conclusão da prova da corretude da função de relaxamento. Ressalta-se que quando uma prova é concluída em um assistente de provas, esta prova está correta (conforme a teoria especificada pelo tal).

5. Considerações Finais

Nesse trabalho foi implementado o algoritmo de Dijkstra no assistente de provas *Coq*, bem como foi definida uma especificação formal e implementada uma prova parcial sobre esta especificação. Entende-se que a abordagem utilizada para modelagem e para a implementação do algoritmo pode ser replicada em outros algoritmos e, dessa forma, novas implementações e provas poderão ser realizadas. Quanto à prova parcial, ainda que ela não tenha sido concluída, foi possível obter evidências de sua corretude, que podem ser formalmente provadas ao verificar que a função de relaxamento está correta.

Para a finalização deste trabalho, é necessária a conclusão das provas sobre a função de relaxamento. Além disso, acredita-se na importância do uso do *Coq* para a implementação e prova de corretude de outros algoritmos que resolvem problemas da teoria de grafos, gerando uma biblioteca sólida para grafos e abrindo a possibilidade de desenvolvimento de novos algoritmos com provas consistentes de seu funcionamento.

[1] A implementação pode ser encontradas no repositório: https://github.com/joao-frohlich/dijkstra-spec

Referências

Appel, A. W. (1998). SSA is functional programming. *ACM SIGPLAN Notices*, 33(4):17–20.

Appel, K. and Haken, W. (1976). Every planar map is four colorable. *Bull. Amer. Math. Soc.*, 82:711–712.

Barras, B., Boutin, S., Cornes, C., Courant, J., Coscoy, Y., Delahaye, D., de Rauglaudre, D., Filliâtre, J.-C., Giménez, E., Herbelin, H., et al. (1999). The Coq proof assistant reference manual. *INRIA, version*, 6(11).

Bertot, Y. and Castéran, P. (2013). *Interactive Theorem Proving and Program Development: Coq'Art: The Calculus of Inductive Constructions*. Springer Science & Business Media.

Gonthier, G. (2008). Formal proof–the four-color theorem. *Notices of the AMS*, 55(11):1382–1393.

Gonthier, G. (2023). A computer-checked proof of the Four Color Theorem. Technical report, INRIA.

Mange, R. and Kuhn, J. (2007). Verifying Dijkstra's algorithm in Jahob. Technical report, EPFL.

Mohan, A., Leow, W. X., and Hobor, A. (2021). Functional Correctness of C Implementations of Dijkstra's, Kruskal's, and Prim's Algorithms. In Silva, A. and Leino, K. R. M., editors, *Computer Aided Verification*, pages 801–826, Cham. Springer International Publishing.

Moore, J. S. and Zhang, Q. (2005). Proof pearl: Dijkstra's shortest path algorithm verified with ACL2. In *Theorem Proving in Higher Order Logics: 18th International Conference, TPHOLs 2005, Oxford, UK, August 22-25, 2005. Proceedings 18*, pages 373–384. Springer.

Nordhoff, B. and Lammich, P. (2012). Dijkstra's Shortest Path Algorithm. *Archive of Formal Proofs*. `https://isa-afp.org/entries/Dijkstra_Shortest_Path.html`, Formal proof development.