Hipergrafos de Dijkstra: Reconhecimento e Isomorfismo

Leonardo de Almeida Cavadas[1], Luerbio Faria[1], Lucila Maria de Souza Bento[1], Jayme Luiz Szwarcfiter[1,3], André Luiz Pires Guedes[2], Lilian Markenzon[3]

[1]Instituto de Matemática e Estatística – Universidade do Estado do Rio de Janeiro, Rio de Janeiro, RJ, Brasil

[2]Departamento de Informática – Universidade Federal do Paraná, Paraná, Brasil

[3]PPGI – Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brasil

leonardo.cavadas@pos.ime.uerj.br, [luerbio, lucila.bento]@ime.uerj.br

jayme@nce.ufrj.br, andre@inf.ufpr.br, markenzon@nce.ufrj.br

Abstract. *This paper introduces Dijkstra hypergraphs – directed hypergraphs modelling the execution of structured parallel programs through flow hypergraphs. Dijkstra hypergraphs arised from the concept of Dijkstra structured programming, the concept of multiprogramming of Hansen and Dijkstra et al. and the Dijkstra graphs of Bento et al. Linear time algorithms are provided to recognize and to check isomorphism of Dijkstra hypergraphs.*

Resumo. *Este artigo introduz os hipergrafos de Dijkstra – hipergrafos direcionados que modelam a execução de programas paralelos estruturados, através de hipergrafos de fluxo. Hipergrafos de Dijkstra foram criados a partir do conceito de programação estruturada de Dijkstra, da multiprogramação de Hansen e de Dijkstra et al. e dos grafos de Dijkstra de Bento et al. Algoritmos de tempo linear são fornecidos para reconhecer e verificar o isomorfismo dos hipergrafos de Dijkstra.*

1. Introdução

Em 1972, independentemente, Hansen (1972) e Dijkstra et al. (1972) definiram os fundamentos da programação estruturada. Os autores ofereceram uma profunda visão sobre o assunto, destacando os princípios do paradigma através do uso de comandos de controle com laços condicionais e testes lógicos, resultando em um código mais claro, fácil de depurar e menos propenso a erros.

Com base no modelo proposto por Dijkstra et al. (1972), Bento et al. (2019) definiram os *grafos de Dijkstra*, uma classe de grafos que modela os grafos de fluxo de controle de programas sequenciais estruturados. Bento et al. provaram o reconhecimento linear dessa classe, além de exibirem um algoritmo linear para verificar o isomorfismo entre grafos de Dijkstra.

Em 1968, Dijkstra propôs o multiprocessamento, a comunicação entre processos, a concorrência e a sincronização com o gerenciamento de processos e o sistema de arquivos. A estrutura paralela usada por Dijkstra foi o PARBEGIN/PAREND. Quatro anos depois, em 1972, Hansen revisita a ideia do multiprocessamento, propondo o multiprocessamento estruturado baseado na associação explícita de uma estrutura de dados compartilhada com processos concorrentes e na definição de operações sobre ela. A mesma estrutura paralela de Dijkstra foi nomeada por Hansen de COBEGIN/COEND.

Baseado nos trabalhos mencionados sobre como um programa paralelo funciona, surgiu o interesse em uma nova estrutura para modelar o fluxo de programas paralelos de acordo com os princípios de programação estruturada.

2. Hipergrafos de Dijkstra

Um *hipergrafo direcionado* $H = (V, E)$ é uma estrutura composta por um conjunto de vértices V e um conjunto de hiper-arcos E. Cada hiper-arco é representado por um par ordenado de conjuntos disjuntos de vértices, denotado por (X, Y), onde X é o conjunto de vértices de origem e Y é o conjunto de vértices de destino.

Um hipergrafo *fonte-sumidouro* G tem as seguintes propriedades: (1) possui dois vértices distintos especiais, *fonte* $s(G)$ e *sumidouro* $t(G)$, (2) existe um caminho de $s(G)$ para todos os outros vértices de G, (3) existe um caminho de todos os vértices de G para $t(G)$ e (4) não há caminho de $t(G)$ para outros vértices de G. Os grafos de declaração introduzidos por Bento et al. (2019) representam um tipo específico de grafo fonte-sumidouro. Esses grafos de declaração são categorizados em sete tipos distintos, identificados pelos itens (a)-(g) na Figura 1.

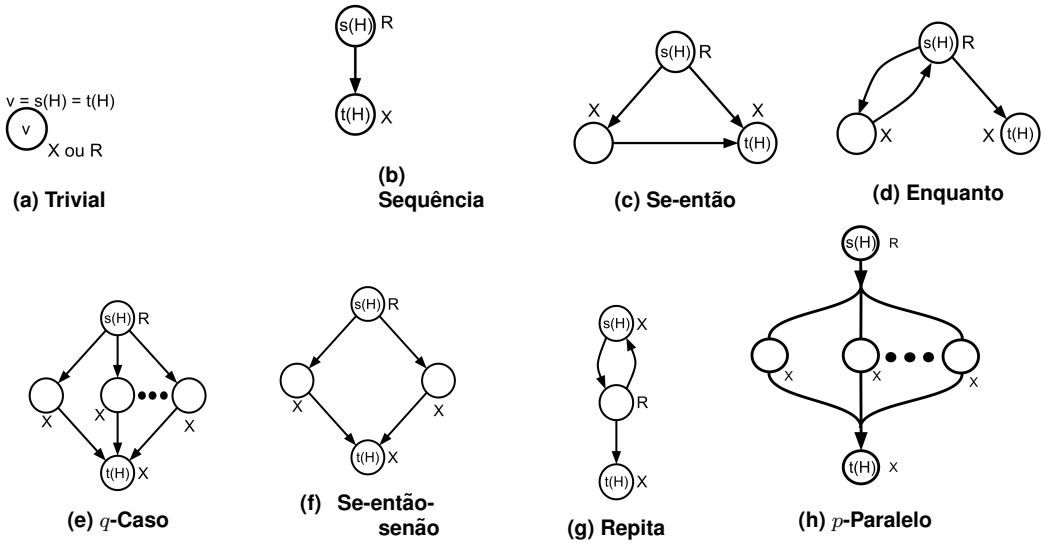

Figura 1. Hipergrafos de declaração.

Uma tripla $G = (V, E, s)$ é um hipergrafo de *fluxo* se (V, E) é um hipergrafo direcionado, $s \in V$ é o vértice de origem e existe um caminho de s para qualquer outro vértice em V. Grafos de fluxo em que uma busca em profundidade iniciada no vértice fonte retorna sempre o mesmo conjunto de arcos de retorno, estes grafos são chamados de grafos de fluxo redutíveis se, e somente se, seus ciclos possuem uma única entrada [Hecht and Ullman 1972, Hecht and Ullman 1974]. Esta definição pode ser estendida para os hipergrafos de fluxo [Guedes et al. 2011], por exemplo, no hipergrafo Enquanto (Figura 1d) só é possível entrar no ciclo pelo vértice $s(H)$ do hipergrafo.

No modelo PARBEGIN/PAREND de Dijkstra, a execução inicia com um processo S_0, após a conclusão desse processo, a estrutura PARBEGIN identifica um conjunto de processos paralelos, representados por $S_1, \ldots, S_p$, $p \geq 2$, que são executados simultaneamente. Depois do término dos processos $S_1, \ldots, S_p$, a estrutura PAREND marca o

fim do paralelismo e o início da execução do processo S_{p+1}, que é então executado sequencialmente. O hipergrafo de declaração p-paralelo (Figura 1(h)) modela a estrutura do PARBEGIN/PAREND. Um hipergrafo de *declaração* é um tipo específico de hipergrafo fonte-sumidouro que pode assumir oito formas distintas, identificadas pelos itens (a)-(h) na Figura 1, sendo rotulados como X (*expansível*) e R (*não expansível*).

Seja um hipergrafo G e H um subhipergrafo fonte-sumidouro de G. Dizemos que H é *fechado* [Bento et al. 2019], quando todo hiper-arco de fora de H entra em H por $s(H)$ e todo hiper-arco que sai de H sai de $t(H)$. Um hipergrafo H é *primo* quando é fechado e isomorfo a algum hipergrafo de declaração com exceção do trivial. Notamos por $\mathcal{H}(G)$ a coleção dos subhipergrafos primos de um hipergrafo G. A operação de *contração* [Bento et al. 2019] de H consiste em contrair os vértices de H em um único vértice v_H, removendo todas as arestas paralelas e laços, gerando o hipergrafo resultante $G \downarrow H$, tal que $N^-_{G\downarrow H}(v_H) = N^-_G(s(H))$ e $N^+_{G\downarrow H}(v_H) = N^+_G(t(H))$, o que significa que a vizinhança de entrada de v_H é igual à vizinhança de entrada de $s(H)$ e a vizinhança de saída de v_H é igual à vizinhança de saída de $t(H)$. Dado G, H dois hipergrafos, onde $V(G) \cap V(H) = \emptyset$ e $v \in V(G)$. A operação de *expansão* de v em H, obtendo o grafo $G \uparrow H$, consiste na substituição de v por H, em G, tal que $N^-_{G\uparrow H}(s(H)) = N^-_G(v)$, $N^+_{G\uparrow H}(t(H)) = N^+_G(v)$ e as adjacências remanescentes são preservadas.

Baseado nesses conhecimentos, definimos os hipergrafos de *Dijkstra* tendo seus vértices rotulados com X ou R, tal que: (1) Um hipergrafo de declaração trivial é um hipergrafo de *Dijkstra*; (2) Qualquer hipergrafo obtido a partir de um hipergrafo de *Dijkstra* ao expandir um vértice rotulado com X em algum hipergrafo de declaração também é um hipergrafo de *Dijkstra*.

Teorema 1 *Se G é um hipergrafo de Dijkstra, então (1) G possui algum subhipergrafo primo, (2) G é um hipergrafo fonte-sumidouro e (3) G é redutível.*

Sejam $H_1, H_2 \in \mathcal{H}(G)$. Chamamos H_1, H_2 de *independentes* se $V(H_1) \cap V(H_2) = \emptyset$, ou $V(H_1) \cap V(H_2) = \{v\}$, onde $v = s(H_1) = t(H_2)$ ou $v = t(H_1) = s(H_2)$

Teorema 2 *Se $H_1, H_2 \in \mathcal{H}(G)$, $H_1 \neq H_2$, então H_1 e H_2 são independentes.*

A utilidade de verificar se um hipergrafo é um hiperrgafo de *Dijkstra* é fundamental para conhecer a estrutura do programa paralelo, de acordo com os princípios de programação estruturada, que aquele hipergrafo de *Dijkstra* representa.

Dado um hipergrafo G. Uma sequência de grafos $\{G_i | i \in \{0, 1, 2, \ldots, k\}\}$ é uma *sequência de contrações*, quando $G = G_0$, e $G_{i+1} = (G \downarrow H_i)$, para algum $H_i \in \mathcal{H}(G_i), i < k$ [Bento et al. 2019]. Chamamos H_i de *primo da contração* de G_i. Uma sequência de contrações é *maximal* quando G_k não tem primos não triviais. Prova-se que independente da sequência de contrações, toda sequência de contrações maximais é máxima. Mais ainda, dadas duas sequências de contrações maximais $\{G_i | i \in \{0, 1, 2, \ldots, k\}\}$ e $\{G'_i | i \in \{0, 1, 2, \ldots, k'\}\}$ de G. Então, $k = k'$ e $G_k = G'_{k'}$. Dessa forma, prova-se que um hipergrafo é de Dijkstra se e somente se, G_k é o grafo trivial, o que constitui a essência do algoritmo de reconhecimento da classe.

Além disso, demonstra-se que é possível utilizar uma ordenação topológica para selecionar cada grafo primo de um hipergrafo de entrada G em $O(m)$, onde m é o número de hiper-arcos de G. O passo final para alcançar o tempo linear de reconhecimento consiste em verificar se $m \leq 2n - 2$, onde $n = |V|$. Isso permite pré-processar G e testar

se é um hipergrafo de Dijkstra apenas quando a ordenação puder ser feita em tempo $O(m) = O(2n-2) = O(n)$. O valor $2n-2$ se origina do Lema "Se $G = (V, E)$ é um hipergrafo direcionado de Dijkstra, com $n = |V|$ e $m = |E|$, então $m \leq 2n-2$."

O algoritmo para reconhecer um hipergrafo de Dijkstra verifica se um hipergrafo de fluxo G de entrada é um hipergrafo de Dijkstra retornando Verdadeiro ou Falso, caso contrário. Inicialmente, uma variável booleana é definida como Falso e atualizada com base nos cálculos. O algoritmo calcula o número de hiper-arcos em G e verifica se é menor ou igual a $2n-2$, onde n é o número de vértices. Se sim, continua a verificação; caso contrário, retorna Falso. Em seguida, executa uma busca em profundidade para encontrar um conjunto de hiper-arcos ciclo, começando na raiz $s(G)$ do hipergrafo. Após a busca, gera uma ordenação topológica dos vértices excluindo o conjunto de hiper-arcos ciclo. Para cada vértice na ordenação, verifica se é a raiz de um subhipergrafo primo H de G. Se sim, realiza uma operação de contração em H e continua o processo. Se G se tornar trivial, atualiza a variável booleana para Verdadeiro. Por fim, retorna o valor da variável, indicando se o hipergrafo de entrada é um hipergrafo de Dijkstra ou não.

3. Isomorfismo

A verificação de isomorfismo entre hipergrafos de Dijkstra é feita também por uma ordenação topológica lexicográfica. Essa ordenação estabelece códigos distintos para cada um dos hipergrafos primos selecionados. Nota-se que todos os grafos primos, com exceção do q-caso e do p-paralelo (Figura 1e e 1h), terão códigos entre 1 e 6, enquanto o q-caso e o p-paralelo terão respectivamente um número par ou ímpar maior ou igual a 7. Devido à natureza lexicográfica da busca, resulta que a sequência de primos é única. Assim, dois hipergrafos de Dijkstra são isomorfos se e somente se possuírem a mesma sequência.

O algoritmo recebe como entrada um hipergrafo de Dijkstra G e o conjunto E_c de hiper-arcos ciclos de G e inicia encontrando uma ordenação topológica dos vértices do hipergrafo. Isso permite que o algoritmo processe os vértices na ordem correta iterando sobre a ordenação topológica na ordem reversa, começando pelo último vértice e terminando no primeiro. Para cada vértice v_i o algoritmo atribui um código $C(v_i)$.

Se o vértice v_i for a raiz $s(H)$ de um subgrafo primo H, então é calculado o código de v_i com base em qual hipergrafo de declaração é o grafo primo H. Para cada H temos associado um número dentre $1, 2, 3, 4, 5, 6, 2(p+1)+1, 2(q+1)$, concatenado (em alguns casos de forma lexicográfica) com os códigos dos vértices vizinhos de $s(H)$. Após isso, é realizado uma operação de contração em H. Finalmente, o código $C(G)$ do hipergrafo G é atribuído como o código do primeiro vértice v_1 (que será $s(G)$) e retornado como resposta. Ao verificar o ismorfismo entre hipergrafos de *Dijkstra*, concluímos que a programação daqueles programas, de cada um dos hipergrafos de *Dijkstra*, foi estruturada da mesma forma.

Concluímos que, através das provas para essa nova estrutura, a programação paralela pode ser estruturada e, com isso, a construção da classe hipergrafos de *Dijkstra* e um dos pontos que essa investigação pode prosseguir é encontrar um algoritmo para CONJUNTO DE VÉRTICES DE RETROALIMENTAÇÃO linear para essa classe.

Referências

Bento, L. M. d. S., Boccardo, D. R., Machado, R. C. S., Miyazawa, F. K., de Sá, V. G. P., and Szwarcfiter, J. L. (2019). Dijkstra graphs. *Discrete Applied Mathematics*, 261:52–62.

Dijkstra, E. W. (1968). The structure of"the"-multiprogramming system. *Communications of the ACM*, 11(5):341–346.

Dijkstra, E. W., Dahl, O.-J., and Hoare, C. A. R. (1972). *Structured programming*. Academic Press Ltd.

Guedes, A. L. P., Markenzon, L., and Faria, L. (2011). Flow hypergraph reducibility. *Discrete applied mathematics*, 159(16):1775–1785.

Hansen, P. B. (1972). Structured multiprogramming. *Communications of the ACM*, 15(7):574–578.

Hecht, M. S. and Ullman, J. D. (1972). Flow graph reducibility. In *Proceedings of the fourth annual ACM symposium on Theory of computing*, pages 238–250.

Hecht, M. S. and Ullman, J. D. (1974). Characterizations of reducible flow graphs. *Journal of the ACM (JACM)*, 21(3):367–375.