Dijkstra Graphs*

Lucila M. S. Bento[1], Davidson R. Boccardo[3], Raphael C. S. Machado[3], Vinícius G. Pereira de Sá[2], and Jayme L. Szwarcfiter[1,2,3]

1 **COPPE-Sistemas**
Universidade Federal do Rio de Janeiro, Brasil

2 **Instituto de Matemática**
Universidade Federal do Rio de Janeiro, Brasil

3 **Instituto Nacional de Metrologia, Qualidade e Tecnologia, Brasil**
E-mails: lucilabento@ppgi.ufrj.br, {drboccardo,rcmachado}@inmetro.gov.br, vigusmao@dcc.ufrj.br, jayme@nce.ufrj.br

Abstract

We revisit a concept that has been central in some early stages of computer science, that of *structured programming*: a set of rules that an algorithm must follow in order to acquire a structure that is desirable in many aspects. While much has been written about structured programming, an important issue has been left unanswered: given an arbitrary, compiled program, decide whether it is *structured*, that is, whether it conforms to the stated principles of structured programming. By employing graph-theoretic tools, we formulate an efficient algorithm for answering this question. To do so, we first introduce the class of graphs which correspond to structured programs, which we call *Dijkstra Graphs*. Our problem then becomes the recognition of such graphs, for which we present an $O(n^2)$-time algorithm. Furthermore, we describe an isomorphism algorithm for Dijkstra graphs presenting the same quadratic complexity.

1998 ACM Subject Classification D.2.3, D.2.4, D.2.8, G.2.2

Keywords and phrases algorithms, directed graphs, isomorphism, structured programming

1 Introduction

Structured programming was one of the main topics in computer science in the years around 1970. It can be viewed as a method for the development and description of algorithms and programs. Basically, it consists of a top-down formulation of the algorithm, breaking it into blocks or modules. The blocks are stepwise refined, possibly generating new, smaller blocks, until refinements no longer exist. The technique constraints the description of the modules to contain only three basic control structures: *sequence*, *selection* and *iteration*. The first of them corresponds to sequential statements of the algorithm; the second refers to comparisons leading to different outcomes; the last one corresponds to sets of actions performed repeatedly in the algorithm.

One of the early papers about structured programming was the seminal article by Dijkstra "Go-to statement considered harmful" [6], which brought the idea that the unrestricted use of go-to statements is incompatible with well structured algorithms. That paper was soon followed by an extensive discussion in the literature about go-to's, as in the papers by Knuth and Floyd [15], Wulf [24] and Knuth [14]. Other classical papers with this aim are those by Dahl and Hoare [7], Hoare [12] and Wirth [21]. The main guidelines of structured

* Authors 1 and 3 have been supported by CNPq, Brazil. Author 5 has been supported by CNPq and CAPES, Brazil.

Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

programming were established in the article by Dijkstra [8], and the early development of programming languages containing blocks—such as ALGOL (Wirth [22]) and PASCAL (Naur [17])—was an important reason for structured programming's widespread adoption. The actual influence of the concept of structured programming in the development of algorithms for solving various problems in different areas occurred right from the start, either explicitly, as in the papers by Henderson and Snowdon [11], and Knuth and Szwarcfiter [16], or implicitly as in the various graph algorithms by Tarjan, e.g. [18, 19].

A natural, important question regarding structured programming is to recognize whether a given program is structured. To our knowledge, such a question has not been solved neither in the early stages of structured programming, nor later. That is the main purpose of the present paper. We formulate an algorithm for recognizing whether a given program is structured, according to Dijkstra's concept of structured programming. Note that the input comprises the binary code, not the source code, otherwise the task would be made trivial by checking the high-level instructions of the chosen programming language. A well-known representation that comes in handy is that of the *control graph* (CFG) of a program, employed by the majority of reverse-engineering tools to perform data-flow analyses and optimizations. A CFG represents the intraprocedural computation of a function by depicting the existing links across its basic blocks. Each basic block represents a straight line in the program's instructions, ending (possibly) with a branch. An edge $A \to B$ (from the exit of block A to the start of block B) represents the program flowing from A to B at runtime.

We are then interested in the version of the recognition problem which takes as input a control flow graph of the program [1, 3]: a directed graph representing the possible sequences of basic blocks along the execution of the program. Our problem thus becomes a graph-theoretic problem: given a control flow graph, decide whether there exists a structured program that produces it.

In this paper, we first define the class of graphs which correspond to structured programs. Such a class has then been named as *Dijkstra graphs*. We then describe a characterization that leads to a greedy polynomial-time recognition algorithm for the graphs of that class, which runs in $O(n^2)$ time for flow graphs with n vertices. Among the potential direct applications, we can mention software watermarking via control flow graph modifications [2, 4].

Additionally, we formulate an isomorphism algorithm for the class of Dijkstra graphs. The method consists of defining a convenient encoding for a graph of the class, which consists of a string of integers. Such an encoding uniquely identifies the graph, and it is shown that two Dijkstra graphs are isomorphic if and only if their encodings are one and the same. The encoding itself has size $O(n)$ and the time complexity of the isomorphism algorithm is $O(n^2)$. In case the given graphs are isomorphic, the algorithm exhibits the isomorphism function between the graphs. Applications of isomorphisms include code similarity analysis [5], since the method can determine whether apparently distinct control flow graphs (of structured programs) are actually structurally identical, with potential implications in digital rights management.

We remark that this is paper is entirely graph theoretic, being motivated by computer programming. Most of the proofs have been omitted owing to space restrictions.

2 Preliminaries

In this paper, all graphs are finite and directed. For a graph G, we denote its vertex and edge sets by $V(G)$ and $E(G)$, respectively, with $|V(G)| = n$, $|E(G)| = m$. For $v, w \in V(G)$, an edge from v to w is written as vw. We say vw is an *out-edge* of v and an *in-edge* of w, with w an *out-neighbor* of v, and v an *in-neighbor* of w. We denote by $N^+_G(v)$ and $N^-_G(v)$ the sets of out-neighbors and in-neighbors of v, respectively. We may drop the subscript when the graph is clear from the context. Also, we write $N^{2+}(v)$ meaning $N^+(N^+(v))$. For $v, w \in V(G)$, v *reaches* w when there is a path in G from v to w. A *source* of G is a vertex that reaches all other vertices in G, while a *sink* is one with no out-neighbors. Denote by $s(G)$ and $t(G)$, respectively, a source and a sink of G. A *(control) flow* graph G is one which contains a distinguished source $s(G)$. A *source-sink* graph contains both a distinguished source $s(G)$ and distinguished sink $t(G)$. A *trivial* graph contains a single vertex.

Let G be a flow graph with source $s(G)$, and C a cycle of G. The cycle C is called a *single-entry cycle* if it contains a vertex $v \in C$ that separates $s(G)$ from the vertices of $C \setminus \{v\}$. A flow graph in which each of its cycles is a single-entry cycle is called *reducible*. Reducible graphs were characterized by Hecht and Ullman [9, 10]. An efficient recognition algorithm for this class has been described by Tarjan [20].

A *topological sort* of a graph G is a sequence $v_1, \ldots, v_n$ of its vertices, such that $v_i v_j \in E(G)$ implies $i < j$. It is well known that G admits a topological sort if and only if G is acyclic. Finally, two graphs G_1, G_2 are *isomorphic* when there is a one-to-one correspondence $f : V(G_1) \cong V(G_2)$ such that $vw \in E(G_1)$ if and only if $f(v)f(w) \in E(G_2)$. In this case, we write $G_1 \cong G_2$, and we call f an *isomorphism function* between G_1, G_2, with $f(v)$ being the *image* of v under f.

3 The Graphs of Structured Programming

In this section, we describe the graphs of structured programming, as established by Dijkstra in [8], leading to the definition of *Dijkstra graphs.* First, we introduce a family of graphs directly related to Dijkstra's concepts of structured programming.

A *statement graph* is defined as being one of the following:

(a) *trivial* graph
(b) *sequence* graph
(c) *if* graph
(d) *if-then-else* graph
(e) *p-case* graph, $p \geq 3$
(f) *while* graph
(g) *repeat* graph

For our purposes, it is convenient to assign labels to the vertices of statement graphs as follows. Each vertex is either an *expansible vertex*, labelled X, or a *regular vertex*, labelled R. See Figures 1 and 2, where the statement graphs are depicted with the corresponding vertex labels. All statement graphs are source-sink. Vertex v denotes the source of the graph in each case.

Let H be a subgraph of G, having source $s(H)$ and sink $t(H)$. We say H is *closed* when

- $v \in V(H) \setminus s(H) \Rightarrow N^-(v) \subseteq V(H)$;
- $v \in V(H) \setminus t(H) \Rightarrow N^+(v) \subseteq V(H)$; and
- $vs(H)$ is a cycle edge $\Rightarrow v \in N^+(s(H))$.

In this case, $s(H)$ is the only vertex of H having possible in-neighbors outside H, and $t(H)$ the only one possibly having out-neighbors outside H.

The following concepts are central to our purposes.

Let H be an induced subgraph of G. We say H is *prime* when

- H is isomorphic to some non-trivial statement graph, and

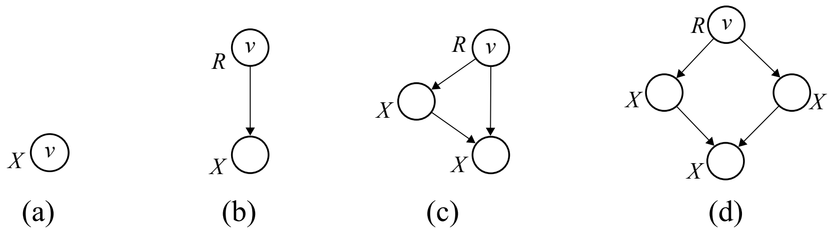

Figure 1 Statement graphs (a)-(d)

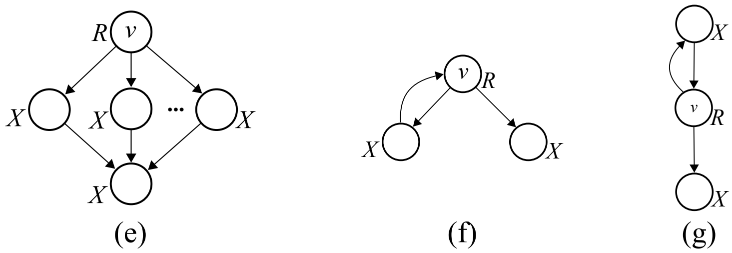

Figure 2 Statement graphs (e)-(g)

- H is closed.

Next, let G, H be two graphs, $V(G) \cap V(H) = \emptyset$, H source-sink, $v \in V(G)$.

The *expansion* of v into a source-sink graph H (Figure 3) consists of replacing v by H, in G, such that

- $N_G^-(s(H)) := N_G^-(v)$;
- $N_G^+(t(H)) := N_G^+(v)$; and
- the remaining adjacencies are unchanged.

Now let G be a graph, and H a prime subgraph of G. The *contraction* of H into a single vertex (Figure 4) is the operation defined by the following steps:

1. Identify (coalesce) the vertices of H into the source $s(H)$ of H.
2. Remove all parallel edges and loops.

We finally have the elements to define the class of Dijkstra graphs. We observe that Dijkstra's concept of structured programming leads naturally to labelled statement graphs.

A *Dijkstra graph (DG)* is a graph with vertices labelled X or R recursively defined as:

1. A trivial statement graph is a DG.

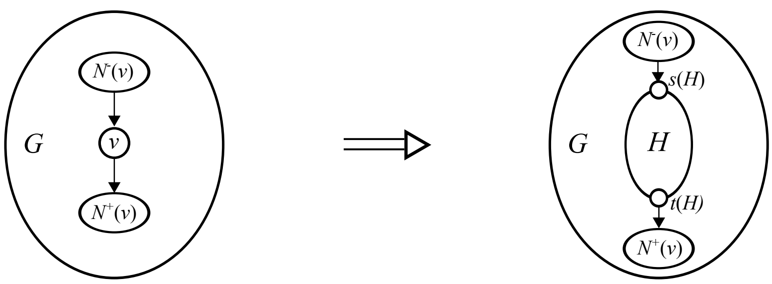

Figure 3 Expansion operation

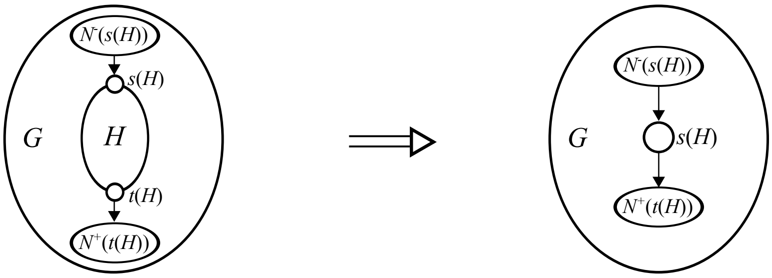

Figure 4 Contraction operation

2. Any graph obtained from a DG by expanding some X-vertex into a non-trivial statement graph is also a DG. Furthermore, after expanding an X-labelled vertex v into a statement graph H, vertex $s(H)$ is labelled as R.

   An example is given in Figure 5.

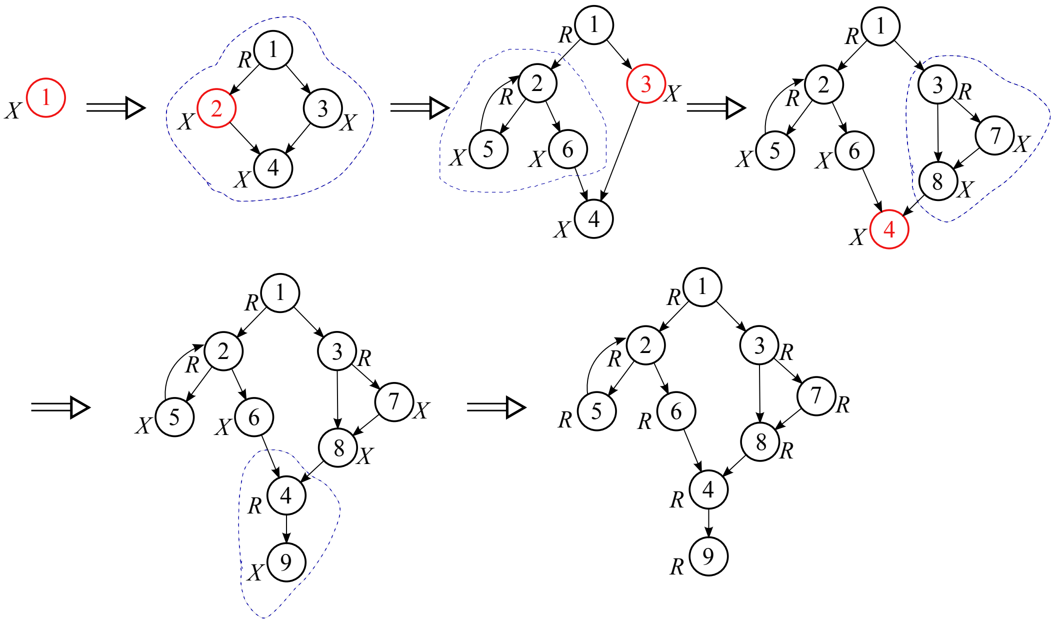

Figure 5 Obtaining a Dijkstra graph via vertex expansions

The above definition leads directly to a constructive characterization of Dijkstra graphs.

▶ **Theorem 1.** *A graph G is a DG if and only if there is a sequence of graphs $G_0, \ldots, G_k$ each of them having the vertices labelled as X or R, such that*

- *G_0 is the trivial graph, with the vertex labelled X;*
- *$G_k \cong G$;*
- *G_i is obtained from G_{i-1}, $i \geq 1$, by expanding some X-vertex v of it into a statement graph H, whose source $s(H)$ receives label R.*

We remark that the above characterization does not imply a polynomial-time algorithm for recognizing graphs of the class. In the next section, we describe another characterization which leads to such an algorithm.

4 Recognition of Dijkstra Graphs

In this section, we describe an algorithm for recognizing Dijkstra graphs. For the recognition process, the hypothesis is that we are given an arbitrary directed graph G, with no labels, and the aim is to decide whether or not G is a DG. First, we introduce some notation and describe the propositions which form the basis of the algorithm.

4.1 Basic Lemmas

The following lemma states some basic properties of Dijkstra graphs.

▶ **Lemma 2.** *If G is a Dijkstra graph, then*

(i) G contains some prime subgraph;
(ii) G is a source-sink graph; and
(iii) G is reducible.

Denote by $\mathcal{H}(G)$ the set of non-trivial prime graphs of G. Let $H, H' \in \mathcal{H}(G)$. Call H, H' *independent* when

- $V(H) \cap V(H') = \emptyset$, or
- $V(H) \cap V(H') = \{v\}$, where $v = s(H) = t(H')$ or $v = t(H) = s(H')$.

The following lemma assures that any pair of distinct, non-trivial prime subgraphs of a graph consists of independent subgraphs.

▶ **Lemma 3.** *Let $H, H' \in \mathcal{H}$. It holds that H, H' are independent.*

Next, we introduce several concepts which are central for the algorithmic characterization that follows.

Let G be a graph, $\mathcal{H}(G)$ the set of non-trivial prime subgraphs of G, and $H \in \mathcal{H}(G)$. Denote by $G \downarrow H$ the graph obtained from G by contracting H. For $v \in V(G)$, the *image* of v in $G \downarrow H$, denoted $I_{G\downarrow H}(v)$, is

$$I_{G\downarrow H}(v) = \begin{cases} v, & \text{if } v \notin V(H) \\ s(H), & \text{otherwise.} \end{cases}$$

For $V' \subseteq V(G)$, define the (*subset*) *image* of V' in $G \downarrow H$, as $I_{G\downarrow H}(V') = \cup_{v \in V'} I_{G\downarrow H}(v)$.

Similarly, for $H' \subseteq G$, the (*subgraph*) *image* of H' in $G \downarrow H$, denoted by $I_{G\downarrow H}(H')$, is the subgraph induced in $G \downarrow H$ by the subset of vertices $I_{G\downarrow H}(V(H'))$.

Next, we formulate two lemmas which are employed in the ensuing characterization. The first of them shows that any prime subgraph $H \in \mathcal{G}$ is preserved under contractions of different primes. Let G be an arbitrary graph, $H, H' \in \mathcal{H}(G)$, $H \neq H'$.

▶ **Lemma 4.** *$I_{G\downarrow H}(H') \in \mathcal{H}(G \downarrow H)$.*

The next lemma proves a commutative law for the order of contractions.

▶ **Lemma 5.** *If $H, H' \in \mathcal{H}(G)$, then $(G \downarrow H) \downarrow (I_{G\downarrow H}(H')) \cong (G \downarrow H') \downarrow (I_{G\downarrow H'}(H))$.*

4.2 Contractile Sequences

A sequence of graphs $G_0, \ldots, G_k$ is a *contractile sequence* for a graph G, when

- $G \cong G_0$, and
- $G_{i+1} \cong (G_i \downarrow H_i)$, for some $H_i \in \mathcal{H}(G_i)$, $i < k$. Call H_i the *contracting prime* of G_i.

We say $G_0, \ldots, G_k$ is *maximal* when $\mathcal{H}(G_k) = \emptyset$. In particular, if G_k is the trivial graph then $G_0, \ldots, G_k$ is maximal.

Let $G_0, \ldots, G_k$, be a contractile sequence of G, and H_j the contracting prime of G_j. That is, $G_{j+1} \cong (G_j \downarrow H_j)$, $0 \leq j < k$. For $H'_j \subseteq G_j$ and $q \geq j$, the *iterated image* of H'_j in G_q is recursively defined as

$$I_{G_q}(H'_j) = \begin{cases} H'_j, & \text{if } q = j \\ I_{G_q}(I_{G_{j+1}}(H'_j)), & \text{otherwise.} \end{cases}$$

Finally, we describe the characterization in which the recognition algorithm for Dijkstra graphs is based.

▶ **Theorem 6.** *Let G be an arbitrary graph, with $G_0, \ldots, G_k$ and $G'_0, \ldots, G'_{k'}$ two contractile sequences of G. Then $G_k \cong G'_{k'}$. Furthermore, $k = k'$.*

Proof: Let $G_0, \ldots, G_k$ and $G'_0, \ldots, G'_{k'}$ be two contractile sequences, denoted respectively by S and S' of a graph G. Let H_j and H'_j be the contracting primes of G_j and G'_j, respectively. That is, $G_{j+1} \cong (G_j \downarrow H_j)$ and $G'_{j+1} \cong (G'_j \downarrow H'_j)$, $j < k$ and $j < k'$. Without loss of generality, assume $k \leq k'$. Let i be the least index, such that $H_j \cong H'_j$, $j \leq i$. Then $G_j \cong G'_j$, $j \leq i$. Such an index exists since $G \cong G_0 \cong G'_0$. If $i = k$ then $G_k \cong G'_{k'}$, implying $k = k'$ and the theorem holds. Otherwise, $i < k$, $G_i \cong G'_i$ and $H_i \not\cong H'_i$. However $H_i \in \mathcal{H}(G'_i)$. By Lemma 4, the iterated image H_{i_q}, of H_i in G'_q is preserved as a prime subgraph for all G'_q, as long as it does not become the contracting prime of G'_{q-1}. Since $G'_{k'}$ has no prime subgraph, it follows there exists some index p, $i < p < k'$, such that $G'_{p+1} \cong (G_p \downarrow H_{i_p})$, where H_{i_p} represents the iterated image of H_i in G'_p. Let $H_{i_{p-1}}$ be the iterated image of H_i in G'_{p-1}. Clearly, $H'_{p-1}, H_{i_{p-1}} \in \mathcal{H}(G'_{p-1})$, and by Lemma 7, H'_{p-1} and $H_{i_{p-1}}$ are independent in G'_{p-1}. Since $((G'_{p-1} \downarrow H'_{p-1}) \downarrow H_{i_p}) \cong G'_{p+1}$, by Lemma 4, it follows that $((G'_{p-1} \downarrow H_{i_{p-1}}) \downarrow H''_{p-1}) \cong G'_{p+1}$, where H''_{p-1} represents the image of H'_{p-1} in $G'_{p-1} \downarrow H_{i_{p-1}}$. Consequently, we have exchanged the positions in S' of two contracting primes, respectively at indices $p-1$ and p, while preserving all graphs G'_q, for $q < p-1$ and $q > p$. In particular, preserving the graph G'_{p+1} and all graphs lying after G'_{p+1} in S', together with their corresponding contracting primes.

Finally, apply the above operation iteratively, until eventually the iterated image of H_i becomes the contracting prime of G'_i. In the latter situation, the two sequences coincide up to index $i+1$, while preserving the original graphs G_k and $G'_{k'}$. Again, applying iteratively such an argument, we eventually obtain that the two sequences turned coincident, preserving the original graphs G_k and $G'_{k'}$. Consequently, $G_k \cong G'_{k'}$, which also means $k = k'$. □

4.3 The Recognition Algorithm

We start this section with a bound for the number m of edges of Dijkstra graphs.

▶ **Lemma 7.** *Let G be a DG graph. Then $m \leq 2n - 2$.*

We can describe an algorithm for recognizing Dijkstra graphs based on Theorem 6.

Let G be a graph. The basic idea is to find iteratively a non-trivial prime subgraph of the graph and contract it, until either the graph becomes trivial or otherwise no such subgraph exists. In the first case the graph is a DG, while in the second it is not.

Algorithm 1: Dijkstra graphs recognition algorithm

Initial Step: Let G be a directed graph. First, count the number of edges of G, and terminate the algorithm if $m > 2n - 2$. Next, verify if G is a reducible flow graph. If negative, then according to Lemma 2, G is again not a DG. Define $G_0 \cong G$, and $k := 0$. Perform a DFS of G, starting from $s(G)$. Let T be the tree obtained by the DFS. Find a topological sort $v_1, \ldots, v_{V(T)}$ of the vertices of T. Set $\ell = 1$.

General Step: If $|V(T)| = 1$ terminate the algorithm: G_0 is a DG graph, and $G_0, \ldots, G_k$ a contractile sequence of it. Otherwise, verify if there is a non-trivial prime subgraph H of G, whose source is v_ℓ. If negative, and additionally $\ell = |V(T)|$ then terminate the algorithm: G_0 is not a DG. However, when $\ell < |V(T)|$, set $\ell : \ell + 1$ and repeat the General Step. In the situation where G contains a non-trivial prime subgraph H, compute $k := k + 1$, $G_k \cong (G_{k-1} \downarrow H)$, find a DFS tree of G_k, starting from $s(G_k)$, find a topological sort $v_1, \ldots, v_{V(T)}$ of the vertices of T, set $\ell = 1$ and repeat the General Step. $\square$

The correctness of Algorithm 1 follows basically from Theorem 6. As for the complexity, first observe that to decide whether the graph contains a non-trivial prime subgraph whose source is a given vertex $v \in V(G)$, we need $O|(N^+(v)|$ steps. Therefore, when considering all vertices of G we require $O(m)$ time. Since there can be $O(n)$ searches for prime graphs, we require overall $O(nm)$ time. Since there can be $O(n)$ prime subgraphs altogether, we require $O(n^2)$ time for termination. Finding a topological sort of a graph can be done in $O(m)$ time [13], while the remaining operations, as contractions, can all be implemented in overall $O(m)$ time. By Lemma 7, $m = O(n)$. Thus, the overall time complexity is $O(n^2)$.

5 Isomorphism of Dijkstra Graphs

We now describe a polynomial-time algorithm for the isomorphism of Dijkstra graphs.

5.1 Breadth-first Contractile Sequences

Let $\mathcal{C} = G_0, \ldots, G_k$ be a contractile sequence of a graph G, and H_i the contracting prime of G_i, i.e., $G_{i+1} \cong (G_i \downarrow H_i)$, $i < k$. Let $G_0^* \cong G_0 \cong G$ and $|\mathcal{H}(G_0^*)| = q_0$, $H_j \in \mathcal{H}(G_0^*)$, $0 \le j < q_0$. For $i = 0, \ldots, \ell - 1$, iteratively denote $G_{i+1}^* \cong G_{i+q_i}^*$ and $|\mathcal{H}(G_{i+1}^*)| = q_{i+1}$, and for $0 \le j < q_{i+1}$, write $H_{j+q_0+\ldots+q_i} \in \mathcal{H}(G_{i+1}^*)$, where ℓ is the least i, such that $\mathcal{H}(G_i^*) = \emptyset$, $i \ge 0$. Denote also $G_{i+1}^* \cong (G_i^* \downarrow \mathcal{H}(G_i^*))$, and write $\mathcal{H}^*(G) = \{H | H \in \mathcal{H}(G_i^*), i \ge 0\}$.

A sequence $\mathcal{C}$ satisfying the above conditions is called a *breadth-first contractile sequence*. In other words, such a contractile sequence is one whose contracting primes are chosen as follows. Let $|\mathcal{H}(G_0^*)| = q_0$. The first q_0 contracting primes are those of $\mathcal{H}(G)$. Let $G_1^* = G_0^* \downarrow \mathcal{H}(G_0^*)$ denote the graph obtained after the contraction of the all primes of $\mathcal{H}(G_0^*)$. Let $|\mathcal{H}(G_1^*)| = q_1$, choose as the next q_1 contracting primes those of $\mathcal{H}(G_1^*)$. Let $G_2^* = G_1^* \downarrow \mathcal{H}(G_1^*)$ and so on. The last graph of the sequence is G_ℓ^*, which contains no primes.

The *dependency* graph $D(G_i^*)$ of G_i^*, relative to some contractile sequence $\mathcal{C}$ of G, is the directed graph whose vertices are the primes of G_i^*, and for primes $H, H' \in \mathcal{H}(G_i^*)$, there is an edge directed from H to H', precisely when $t(H) = s(H')$.

▶ **Lemma 8.** *Let $D(G_i^*)$ be the dependency graph of G_i^*, relative to a breadth-first contractile sequence of it. It holds that $D(G_i^*)$ is a union of vertex disjoint induced paths.*

▶ **Lemma 9.** *For each $i = 0, \ldots, \ell - 1$, the graph G_{i+1}^* is invariant, not depending on the chosen ordering for the contracting primes of $\mathcal{H}(G_i^*)$.*

5.2 Encoding a Dijkstra Graph

We describe below a convenient Dijkstra graph encoding which is the basis of our isomorphism algorithm for Dijkstra graphs. It will be shown two DGs are isomorphic if and only their corresponding encodings coincide. The encodings refer explicitly to the statement graphs having source v as depicted in Figures 1 and 2, and consist of (linear) strings. The following operation on strings is employed. Let, A, B be a pair of strings. The concatenation of A and B, denoted $A||B$, is the string formed by A, immediately followed by B.

For a Dijkstra graph G, the string that will be coding G is constructed over an alphabet of symbols containing integers in the range $\{1, \ldots, \Delta^+(G) + 4\}$, where $\Delta^+(G)$ is the maximum cardinality among the out-neighborhoods of G.

In order to define an encoding $C(G)$ for a Dijkstra graph G, we assign an integer, named *type*(H), for each statement graph H, an encoding $C(v)$ for each vertex $v \in V(G)$, and an encoding $C(H)$ for each prime subgraph H of a breadth-first contractile sequence of G. The encoding $C(G)$ of the graph G is defined as being that of the source of G. For a subset $V' \subseteq V(G)$, the encoding $C(V')$ of V' is the set of strings $C(V') = \{C(v_i)|v_i \in V'\}$. Write $lex(C(V')) = C(v_1)||...||C(v_r)$ whenever $V' = \{v_1, \ldots, v_r\}$ and $C(v_i)$ is lexicographically not greater than $C(v_{i+1})$.

Next, we describe how to obtain the actual encodings. The types of the the different statement graphs are shown in the second column of Table 1. For a vertex $v \in V(G)$, the encoding $C(v)$ is initially set to 1. Subsequently, if v becomes the source of a prime graph H, $C(v)$ is updated by implicitly assigning $C(v) := C(v)||C(H)$, where the encoding $C(H)$ of H is given by the third column of Table 1. Such an operation is called the *expansion* of v. It follows that $C(H)$ is written in terms of *type*(H) and the encodings of the vertices of H, and so on iteratively. A possible expansion of some other vertex $w \in V(G)$ could imply in an expansion of v, and so iteratively. Observe that when H is an if-then-else or a p-case graph, we have chosen to place the encodings of the out-neighbors of $s(H)$ in lexicographic ordering. For the remaining statement graphs H, the ordering of the encodings of the out-neighbors of $s(H)$ is also unique and implicitly imposed by H. When all primes associated to $C(v)$ have been expanded, $C(v)$ is a string of integers, and is considered *ready*.

Table 1 Statement graph types and encodings $C(H)$ of prime subgraphs H, with $s(H) = v$

statement graphs H	*type*(H)	$C(H)$
trivial	1	
sequence	2	$2\|\|C(N^+(v))$
if-then	3	$3\|\|C(N^+(v) \setminus N^{2+}(v))\|\|C(N^{2+}(v))$
while	4	$4\|\|C(N^+(v) \cap N^-(v))\|\|C(N^+(v) \setminus N^-(v))$
repeat	5	$5\|\|C(N^+(v) \cap N^-(v))\|\|C(N^+(v) \setminus N^-(v))$
if-then-else	6	$6\|\|lex(C(N^+(v)))\|\|C(N^{2+}(v))$
p-case	$p+4$	$p+4\|\|lex(C(N^+(v)))\|\|C(N^{2+}(v))$

Algorithm 2: Dijkstra graphs isomorphism algorithm

Initial Step: Set $C(v) := 1$, for each $v \in V(G)$. Set $i := 0$ and $G_0^* := G$.

General Step: If $i = \ell$ then set $C(G) := C(s(G))$ and terminate the algorithm. Otherwise, find the collection of prime subgraphs $\mathcal{H}(G_i^*)$ and the dependency graph $D(G_i^*)$ of G_i^*. While $V(D(G_i^*)) \neq \emptyset$ perform the following operations. Choose a sink H of $D(G_i^*)$, remove it from $D(G_i^*)$ and let $v := s(H)$. Then expand $C(v)$ as follows:

$$C(v) := \begin{cases} 1||2||C(N^+(v)), \text{ if } H \text{ is a sequence graph;} \\ 1||3||C(N^+(v) \setminus N^{2+}(v))||C(N^{2+}(v)), \text{ if } H \text{ is an if-then graph;} \\ 1||type(H)||C(N^+(v) \cap N^-(v))||C(N^+(v) \setminus N^-(v)), \\ \qquad\qquad \text{if } H \text{ is a while or repeat graph;} \\ 1||type(H)||lex(C(N^+(v)))||C(N^{2+}(v)), \\ \qquad\qquad \text{if } H \text{ is an if-then-else or p-case graph.} \end{cases}$$

When D_i turns empty, construct $G_{i+1}^* \cong (G_i^* \downarrow \mathcal{H}(G_i^*))$, set $i := i + 1$ and repeat the General Step.

5.3 The Isomorphism Algorithm

Let G be a DG. Algorithm 2 constructs the encoding $C(G)$ for G.

As an example, consider the Dijkstra graph G of Figure 6(a). Initially, set $C(v) := 1, \forall v \in V(G)$. Following the breadth-first contractile sequence construction, we obtain $G_0^* \cong G$ and that the set of primes $\mathcal{H}(G_0^*)$ of G_0^* consists of the four primes marked in Figure 6(a). Then the dependency graph $D(G_0^*)$ is formed by the path q, a and two isolated vertices m and u. Following their sink-to-source ordering results in

$C(a) := 16||C(n)||C(c)||C(e) = 16111$,

$C(q) := 12||C(a) = 1216111$,

$C(m) := 12||C(d) = 121$,

$C(u) := 12||C(v) = 121$.

Next, compute $G_1^* \cong (G_0^* \downarrow \mathcal{H}(G_0^*))$. It follows that $\mathcal{H}(G_1^*)$ is formed by the two primes, depicted in Figure 6(b). Then $D(G_1^*)$ contains two isolated vertices y and p leading to

$C(y) := 14||C(a)||C(b) = 141216111$,

$C(p) := 13||C(m)||C(t) = 131211$.

Finally, compute $G_2^* \cong (G_1^* \downarrow \mathcal{H}(G_1^*))$, and $\mathcal{H}(G_2^*)$ contains the unique prime of Figure 6(c), which leads to $D(G_2^*)$ to correspond to the single vertex x. Since the corresponding statement graph is an if-then-else graph, we need to lexicographically order the encodings of the out-neighbors y and p of x. We obtain $C(p) \leq C(y)$, which implies that $C(y)$ must precede $C(p)$ in $C(x)$. Finally, since x is the source of G, it follows that

$C(G) := C(x) := 16||C(p)||C(y)||C(u) = 16131211141216111121$.

5.4 Correctness and Complexity

▶ **Lemma 10.** *During the execution of the encoding algorithm of a Dijkstra graph G, consider the expansion of encoding $C(v)$. Suppose that encodings $C(v_1), C(v_2), \ldots, C(v_p)$ appear on the right hand side of the expression of $C(v)$. Then all encodings $C(v_1), C(v_2), \ldots, C(v_p)$, as well as those obtained by their expansions, are ready at this point.*

The following Theorem 11 is the main result in this isomorphism section.

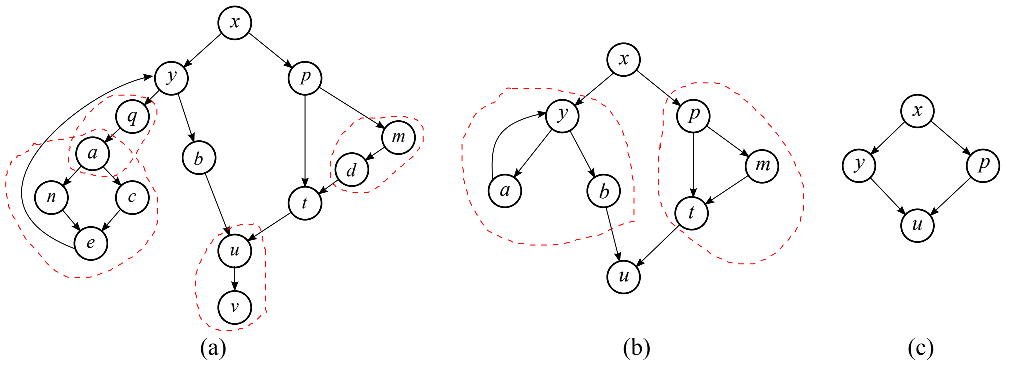

Figure 6 Encoding a Dijkstra Graph

▶ **Theorem 11.** *Let G, G' de Dijkstra graphs, and $C(G), C(G')$ their encodings, respectively. Then G, G' are isomorphic if and only if $C(G) = C(G')$.*

The corollaries below are direct consequences of Theorem 11.

▶ **Corollary 12.** *Let G be a DG and $C(G)$ its encoding.*

1. *There is a one-to-one correspondence between the 1's of $C(G)$ and the vertices of G.*
2. *The encoding $C(G)$ of G is unique and is a representation of G.*

▶ **Corollary 13.** *Let G, G' be DGs and $C(G), C(G')$ their corresponding encodings, satisfying $C(G) = C(G')$. Then an isomorphism function f between G and G' can be determined as follows. Let $v \in V(G)$ and $v' \in V(G')$ correspond to 1's at identical relative positions in $C(G)$ and $C(G')$, respectively. Define $f(v) := v'$.*

Finally, consider the complexity of the isomorphism algorithm.

▶ **Lemma 14.** *Let G be a Dijkstra graph, and $C(G)$ its encoding; Then $|C(G)| \leq 2n - 1$.*

▶ **Theorem 15.** *The isomorphism algorithm terminates within $O(n^2)$ time.*

6 Conclusions

The analysis of control flow graphs has been considered in various papers. To our knowledge, no full characterization and no recognition algorithm for control flow graphs of structured programs were known before. Nevertheless, there are some related classes for which characterizations and efficient recognition algorithms do exist, e.g. the class of reducible graphs, a superclass of Dijkstra graphs. The class of reducible graphs, however, is much larger, and its recognition played but a minor role in solving the present problem.

An important question solved in this paper is that of recognizing whether two control flow graphs (of structured programs) are syntactically equivalent, i.e., isomorphic. Such question fits in a recent area of research called *code similarity analysis*, with applications in clone detection, plagiarism and software forensics.

Since the establishment of structured programming, new statements have been added to the original structures, as is the case of *break*/*continue* and the *divergent if-then-else*. In fact, the inclusion of additional control blocks in structured programming was predicted by Knuth [14] himself. It is noteworthy that the basic ideas and techniques in the present paper can be generalized to efficiently recognize graphs that incorporate the above statements. This is also true for the isomorphism algorithm.

References

1. F. E. Allen and J. Cocke, A program data flow analysis procedure, *Comm. ACM* 19 (1976), 137–147.
2. L. M. S. Bento, D. R. Boccardo, R. C. S. Machado, V. G. Pereira de Sá and J. L. Szwarcfiter, Towards a provably resilient scheme for graph-based watermarking, *Proc. 39th Intl. Workshop on Graph-Theoretic Concepts in Comp. Sci. (WG'13), LNCS* 8165 (2013), 50–63.
3. N. Chapin and S. P. Denniston, Characteristics of a structured program, *ACM SIGPLAN Notices* 13 (1978), 36–45.
4. C. Collberg, S. Kobourov, E. Carter and C. Thomborson, Error-correcting graphs for software watermarking, *29th Workshop on Graph-Theoretic Concepts in Computer Science, (WG'03), Lecture Notes in Computer Science* 2880 (2003), 156–167.
5. Collberg, C. and J. Nagra, *Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection*. Addison Wesley (2010).
6. E. W. Dijkstra, Go-to statement considered harmful, *Comm. ACM* 11 (1968), 174–186.
7. O.-J. Dahl and C. A. Hoare, Hierarchical program structures, in *Structured Programming*, Academic Press, 1972, 175-220.
8. E. W. Dijkstra, Notes on Structured Programming, in *Structured Programming* (1972), 1–82, Acad. Press.
9. M. S. Hecht and J. D. Ullman, Flow graph reducibility, *SIAM J. on Comp.* 1 (1972), 188-202.
10. M. S. Hecht and J. D. Ullman, Characterizations of reducible flow graphs, *J. ACM* 21 (1974), 367–374.
11. P. Henderson and R. Snowdon, An experiment in structured programming, *BIT* 12 (1972), 38–53.
12. C. A. R. Hoare, Notes on data structuring, in *Structured Programming*, Academic Press, 1972, 83-174.
13. D. E. Knuth, *The Art of Computer Programming: Fundamental Algorithms (volume 1)* Addision-Wesley, Reading, Ma., 1968 (2nd ed. 1973).
14. D. E. Knuth, Structured programming with go-to statements, *ACM Comp. Surveys* 6 (1974) 261–301.
15. D. E. Knuth and R. W. Floyd, Notes on avoinding 'go to' statements, *Inf. Proc. Let.* 1 (1971), 23–31.
16. D. Knuth and J. L. Szwarcfiter, A structured program to generate all topological sort arrangements, *Inf. Proc. Let.* 2 (1974), 153–157.
17. P. Naur, ed. Report on the algorithmic language ALGOL 60, *Comm. ACM* 3 (1960), 299-314.
18. R. E. Tarjan, Depth-first search and linear graph algorithms, *SIAM J. Comp.* 1 (1972), 146-160.
19. R. E. Tarjan, Finding dominators in directed graphs, *SIAM J. Comp.* 3 (1974), 62–89.
20. R. E. Tarjan, Testing flow graph reducibility, *J. Comp. Sys. Sci.* 9(1974), 355-365
21. N. Wirth, Program development by stepwise refinement, *Comm. ACM* 14 (1971), 221-227.
22. N. Wirth, The programming language PASCAL, *Acta Informatica* 1 (1971), 35-63.
23. N. Wirth, On the composition of well structured programs, *ACM Comp. Surveys* 6 (1974), 247-259.
24. W. A. Wulf, A case against the Go-to, *Proc. of the ACM Ann. Conf.*, 1972, 791-791.

A Appendix: omitted proofs

Theorem 1. *A graph G is a DG if and only if there is a sequence of graphs $G_0, \ldots, G_k$ each of them having the vertices labelled as X or R, such that*

(i) *G_0 is the trivial graph, with the vertex labelled X*
(ii) *$G_k \cong G$*
(iii) *G_i is obtained from G_{i-1}, $i \geq 1$, by expanding some X-vertex v of it into a statement graph. The label of v is then changed to R.*

Proof: Suppose G is a DG graph. It follows from its definition that G can be constructed by starting from a trivial graph G_0, whose vertex is labelled X, and iteratively replacing expansion vertices by statement graphs. Let k be the number of expansions performed in the construction of G. At iteration i, denote by G_i the graph obtained from G_{i-1} by expanding some expansion vertex of G_{i-1} into a statement graph $0 \leq i \leq k$. Then $G_k \cong G$ and the sequence $G_0, \ldots, G_k$ satisfies the conditions of the theorem.

Conversely, suppose that there exists a sequence of graphs $G_0, \ldots, G_k$ satisfying the conditions (i)-(iii) of the theorem. Conditions (i) and (iii) suffice to show that G_i is a DG graph, for all $0 \leq i \leq k$, and condition (ii) implies G is a DG graph. ☐

Lemma 2. *Let G be a Dijkstra graph. Then*

(i) *G contains some prime subgraph*
(ii) *G is a source-sink graph*
(iii) *G is reducible*

Proof: By definition, there is a sequence of graphs $G_0, \ldots, G_k$, where G_0 is trivial, $G_k = G$ and G_i is obtained from G_{i-1} by expanding some X-vertex $v_{i-1} \in V(G_{i-1})$ into a statement graph $H_i \subseteq G_i$. Then no vertex $v_i \in V(H_i)$, except $s(H_i)$ has in-neighbors outside H_i, and also no vertex $v_i \in V(H_i)$, except $t(H_i)$, has out-neighbors outside H_i. Furthermore, if H_i contains any cycle then H_i is necessarily a while graph or a repeat graph. The latter implies that such a cycle is $s(H)v$, where $v \in N^+(s(H))$. Therefore H_i is prime in G_i meaning that (i) holds. To show (ii) and (iii), first observe that any statement graph is single-source and reducible. Next, apply induction. For G_0, there is nothing to prove. Assume it holds for G_i, $i > 1$. Let $v_{i-1} \in V(G_{i-1})$ be the vertex that expanded into the subgraph $H_i \subseteq G_i$. Then the external neighborhoods of H_i coincide with the neighborhoods of v_{i-1}, respectively. Consequently, G_i is single-source. Now, let C_i be any cycle of G_i, if existing. If $C_i \cap H_i = \emptyset$ then C_i is single-entry, since G_{i-1} is reducible. Otherwise, if $C_i \subset V(H_i)$ the same is valid, since any statement graph is reducible. Finally, if $C_i \not\subset V(H_i)$, then v_{i-1} is contained in a single-entry cycle C_{i-1} of G_{i-1}. Then C_i has been formed by C_{i-1}, replacing v_{i-1} by a path contained in H_i. Since C_{i-1} is single-entry, it follows that C_i must be so. ☐

Lemma 4. *$I_{G\downarrow H}(H') \in \mathcal{H}(G \downarrow H)$.*

Proof: Let G be a graph, $H, H' \in \mathcal{H}(G)$, $H \neq H'$. By Lemma 7, H, H' are independent. If H, H' are disjoint the contraction of H does not affect H', and the lemma holds. Otherwise, by the independence condition, it follows that $V(H) \cap V(H') = \{v\}$, where $v = s(H) = t(H')$ or $v = s(H') = t(H)$. Examine the first of these alternatives. By contracting H, all neighborhoods of the vertices of $I_{G\downarrow H}(H')$ remain unchanged, except that of $I_{G\downarrow H}(s(H'))$, since its in-neighborhood becomes equal to $N^-_G(s(H))$. On the other hand, the contraction of H into v cannot introduce new cycles in H'. Consequently, H' preserves in $G \downarrow H$ its property of being a non-trivial and closed statement graph, moreover, prime. Finally, suppose

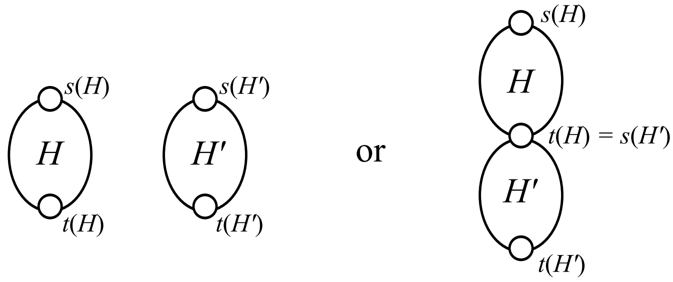

Figure 7 Independent primes

$v = s(H) = t(H')$. Again, the neighborhoods of the vertices of $I_{G\downarrow H}(H')$ are preserved, except possibly the out-neighborhoods of the vertices of $I_{G\downarrow H}(t(H'))$, which become $N^+_G(t(H))$, after possibly removing self-loops. Consequently, $I_{G\downarrow H}(H') \in \mathcal{H}(G \downarrow H)$. ☐

Lemma 5. *Let $H, H' \in \mathcal{H}(G)$. Then*

$$(G \downarrow H) \downarrow (I_{G\downarrow H}(H')) \cong (G \downarrow H') \downarrow (I_{G\downarrow H'}(H)).$$

Proof: Let $A \cong (G \downarrow H) \downarrow (I_{G\downarrow H}(H'))$ and $B \cong (G \downarrow H') \downarrow (I_{G\downarrow H'}(H))$. By Lemma 7, H, H' are independent. First, suppose H, H' are disjoint. Then $I_{G\downarrow H}(H') = H'$ and $I_{G\downarrow H'}(H) = H$. It follows that, in both graphs A and B, the subgraphs H and H' are respectively replaced by a pair of non-adjacent vertices, whose in-neighborhoods are $N^-_G(s(H))$ and $N^-_G(s(H'))$, and out-neighborhoods $N^+_G(t(H))$ and $N^+_G(t(H'))$, respectively. Then $A = B$. In the second alternatives, suppose H, H' are not disjoint. Then $V(H) \cap V(H') = \{v\}$, where $v = s(H) = t(H')$, or $v = t(H) = s(H')$. In both cases, and in both graphs A and B, the subgraphs H and H' are contracted into a common vertex w. When $v = s(H) = t(H')$, it follows $N^-_G(A) = N^-_G(s(H')) = N^-_B(v)$ and $N^+_A(v) = N^+_G(t(H)) = N^+_B(v)$. Finally, when $v = t(H) = s(H')$, we have $N^-_A(v) = N^-_G(s(H)) = N^-_B(v)$, while $N^+_A(v) = N^+_G(t(H')) = N^+_B(v)$. Consequently, $A = B$ in any situation. ☐

Lemma 7. *Let $H, H' \in \mathcal{H}$. Then H, H' are independent.*

Proof: If $V(H) \cap V(H') = \emptyset$ the lemma holds. Otherwise, let $v \in V(H) \cap V(H')$. The alternatives $v = s(H_1) = s(H_2)$, $v = t(H_1) = t(H_2)$, $v \neq s(H_1), t(H_1)$ or $v \neq s(H_2), t(H_2)$ do not occur because they imply H_1 or H_2 not to be closed. Next, let $v_1, v_2 \in V(H_1) \cap V(H_2)$, $v_1 \neq v_2$. In this situation, examine the alternative where $v_1 = s(H_1) = t(H_2)$ and $v = s(H_2) = t(H_1)$. The latter implies that exactly one of H_1 or H_2, say H_2, is a while graph or a repeat graph. Then there is a cycle edge $ws(H_1)$, satisfying $w \in N^-(s(H_1))$ and $w \in V(H_2) \setminus \{t(H_2)\}$. Consequently, $w \notin N^+(s(H_1))$, contradicting H_1 to be closed. The only remaining alternative is $V(H_1) \cap V(H_2) = \{v\}$, with $v = s(H_1) = t(H_2)$ or $v = s(H_2) = t(H_1)$. Then H_1, H_2 are indeed independent. ☐

Lemma 7. *Let G be a DG graph. Then $m \leq 2n - 2$.*

Proof: By Theorem 1, if G is a DG graph there is a sequence of graphs $G_0, \ldots G_k$, where G_0 is the trivial graph, $G_k \cong G$ and G_i is obtained from G_{i-1} by expanding an X-vertex of G_{i-1} into a statement graph. Apply induction on the number of expansions employed in the construction of G. If $k = 0$ then G is a trivial graph, which satisfies the lemma. For $k \geq 0$, Suppose the lemma true for any graph $G' \cong G_i$, $i < k$. In particular, let $G_i \cong G_{k-1}$. Let

n' and m' be the number of vertices and edges of G', respectively. Then $m' \leq 2n' - 2$. We know that G_k has been obtained by expanding a vertex of G_{k-1} into a statement graph H. Discuss the alternatives for H. If H is the trivial graph then $n = n'$ and $m = m'$. If H is a sequence graph then $n = n' + 1$ and $m = m' + 1$. If H is an if graph, a while graph or repeat graph then $n = n' + 2$ and $m = m' + 3$. If H is an if then else graph or a p-case graph then $n = n' + p + 1$ and $m = m' + 2p$, where p is the outdegree of the source of H. In any of these alternatives, a simple calculation implies $m \leq 2n - 2$. □

Lemma 8. *Let $D(G_i^*)$ be the dependency graph of G_i^*, relative to a breadth-first contractile sequence of it. Then $D(G_i^*)$ is a union of vertex disjoint induced paths.*

Proof: Let G be an arbitrary graph. By Lemma 7, for each fixed i, each pair of prime subgraphs $H, H' \in \mathcal{H}(G_i^*)$ is either vertex disjoint or such that the source of one of them, say $s(H)$ coincides with the sink $t(H')$ of the other. In addition, there can be no third prime subgraph $H'' \in \mathcal{H}(G_i^*)$ whose source or sink coincides with the same vertex $t(H) = s(H')$, otherwise one of the subgraphs H, H', H'' would not be closed, a contradiction with being primes. The lemma follows. □

Lemma 9. *For each $i = 0, \ldots, \ell - 1$, the graph G_{i+1}^* is invariant, not depending on the chosen ordering for the contracting primes of $\mathcal{H}(G_i^*)$.*

Proof: Apply Lemma 5 iteratively, in the interval of $\mathcal{C}$, starting at the first contractile prime of the graph corresponding to G_i^* and ending at the last contractile prime which generated G_{i+1}^*. This preserves the graph G_{i+1}^*. □

Lemma 10. *During the execution of the encoding algorithm of a Dijkstra graph G, consider the expansion of encoding $C(v)$. Suppose that encodings $C(v_1), C(v_2), \ldots, C(v_p)$ appear on the right hand side of the expression of $C(v)$. Then all encodings $C(v_1), C(v_2), \ldots, C(v_p)$, as well as those obtained by their expansions, are ready at this point.*

Proof: If v is not a source of some prime subgraph of $\mathcal{H}^*(G)$ then $C(v) = 1$ and there is nothing to prove. Otherwise, v is the source of some $H \in \mathcal{H}(G_i^*)$, for some $i \geq 0$. Let v_j be a vertex which lies on the right hand side of the expression of $C(v)$. If v_j is not a source of some prime subgraph of $\mathcal{H}^*(G)$ then $C(v_j) = 1$ and the lemma holds. Consider the situation where $v_j = s(H')$, for $H' \in \mathcal{H}^*(G)$. Let $H' \in \mathcal{H}(G_q^*)$. Following the ordering of the contractions imposed by the breadth-first contractile sequences, we know that $q \leq i$. If $q < i$, by a inductive argument, we may assume that the encodings of the sources of the primes os $\mathcal{H}(G_q^*)$ are ready, by the time the sources of $\mathcal{H}(G_i^*)$ are considered. In particular, $C(v_j)$ is ready when it appears at the right hand side of $C(v)$. Next, consider te alternative $q = i$, that is, v and v_j are sources of primes of $\mathcal{H}(G_i^*)$. Since $C(v_j)$ is at the right of $C(v)$, there is a path in the dependency graph $D(G_i^*)$, where v is an ancestor of v_j. The algorithm considers the vertices of $D(G_i^*)$ in some order from, sinks to sources. In consequence, $C(v_j)$ is obtained before $C(v)$, meaning that it is ready at the time $C(v)$ is considered. □

Theorem 11. *Let G, G' de Dijkstra graphs, and $C(G), C(G')$ their encodings, respectively. Then G, G' are isomorphic if and only if $C(G) = C(G')$.*

Proof: By hypothesis, G, G' are isomorphic. We show that it implies $C(G) = C(G')$. First, recall that $C(G) = C(s(G))$ and $C(G') = C(s(G'))$. Let $\mathcal{C}, \mathcal{C}'$ be breadth-first contractile sequences of G, G', respectively, and consider the graphs $G_0^*, \ldots, G_\ell^*$ and $G_0'^*, \ldots, G_{\ell'}'^*$. Since G, G' are isomorphic, by Lemma 9 and Theorem 6, we may assume that $\ell = \ell'$, and for

$0 \leq i \leq \ell$, $\mathcal{H}(G_i^*) = \mathcal{H}(G_i'^*)$, and that $G_i^*, G_i'^*$ are isomorphic. Without loss of generality, we can also assume that, for some isomorphism function, if H, H' are primes of $G_i^*, G_i'^*$, respectively such that $s(H')$ is the image of $s(H)$ under the isomorphism, then the remaining vertices of $V(H')$ are respectively the images of those of $V(H)$. Next, let us show that indeed $C(G) = C(G')$. In fact, we prove that if $s(H')$ is the image of $s(H)$ under the considered isomorphism function, then $C(s(H)) = C(s(H'))$, for $H \in \mathcal{H}^*(G)$ and $H' \in \mathcal{H}^*(G')$. The proof proceeds by induction on the index i of $G_i^*, G_i'^*$. For $i = 0$, let H, H' be primes of $G_0^*, G_0'^*$, respectively, such that $s(H_0)$ is the image of $s(H)$, and both $s(H), s(H)$ are ready. Then H, H' must be sinks of $D(G_i^*), D(G_i'^*)$, respectively. Since any $v \in V(H)$ and $v' \in V(H')$ satisfies $C(v) = C(v') = 1$, it follows that $C(s(H)) = C(s(H'))$, as required. The induction hypothesis is that if v' is the image of v, and $C(v), C(v')$ are ready at some iteration $\leq i$ then $C(v) = C(v')$. Similarly, for $i > 0$, we choose primes H, H' of $G_i^*, G_i'^*$, respectively, such that they are sinks in their corresponding dependency graphs. Applying the induction hypothesis for the vertices $v \in V(H)$ and $v' \in V(H')$ would lead to $C(s(H)) = C(s(H'))$. Eventually we obtain $C(s(G_{\ell-1}^*)) = C(s(G_{\ell-1}'^*))$, implying $C(G) = C(G')$.

Conversely, assume that G, G' are Dijkstra graphs such that $C(G) = C(G')$. We show that G, G' are isomorphic. The proof is by induction on the length of the encodings. If $|C(G)| = |C(G')| = 1$ then G, G' are trivial graphs and the theorem holds. Otherwise, assume the result correct for graphs whose encodings have lengths $< |C(G)|$. By Lemma 2, G contains a prime subgraph H. Consequently, $C(G)$ contains a substring $C(H)$, corresponding to H, starting at the 1 corresponding to $s(H) \in V(G)$ and ending at the 1 corresponding to $t(H) \in V(G)$. Since $C(G) = C(G')$, the encoding $C(G')$ contains a substring X which coincides with $C(H)$, inclusive in their relative positions in $C(G)$ and $C(G')$, respectively. Then $X = C(H)$, which implies that X corresponds to a prime subgraph $H' \in \mathcal{H}(G')$. Let $s(H')$ and $t(H')$ be the vertices of G' corresponding to the 1's at the beginning and ending of substring X in $C(G')$, respectively. Next, compress substring $C(H)$ into its head corresponding to $s(H)$, and $C(H')$ into its head $s(H')$. The resulting strings correspond to the encodings of the graphs $G \downarrow H$ and $G' \downarrow H'$, respectively. By the induction hypothesis, $G \downarrow H$ and $G' \downarrow H'$ are isomorphic. Since H and H' are isomorphic and $s(H') = f(s(H))$ it follows that G and G' are isomorphic. □

Lemma 14. *Let G be a Dijkstra graph, and $C(G)$ its encoding; Then $|C(G)| \leq 2n - 1$. The bound is tight.*

Proof: The encoding $C(G)$ consists of exactly n 1's, together with elements of a multiset $U \subseteq \{1, \ldots, \Delta^+(G) + 4\}$. We know that $C(G)$ starts and ends with an 1, and it contain no two consecutive elements of U. Therefore $C(G) \leq 2n - 1$. When G consists of the induced patn P_n, it follows $|C(P_n)| = 2n - 1$, attaining the bound. □

Theorem 15. *The isomorphism algorithm terminates within $O(n^2)$ time.*

Proof: Recall that $m = O(n)$, by Lemma 7. The construction of a breadth-first contractile sequence and the determination of all primes of $H \in \mathcal{H}(G_i^*), 0 \leq i < \ell$ requires $O(n^2)$ time. There are at most $O(n)$ such prime subgraphs altogether. Lexicographic ordering takes linear time on the total length of the strings to be sorted. For each $v \in V(G)$, following the isomorphism algorithm, $C(v)$ can be constructed in time $|C(v)|$. Consequently, each of the operations, finding a breadth-first contractile sequence and determining the encoding of the graph, terminates within $O(n^2)$ time. □