Article

Not peer-reviewed version

# The Clique Width of Two Classes of Series-Parallel Digraphs

Ruzayn Quaddoura *

Posted Date: 26 March 2024

doi: 10.20944/preprints202403.1498.v1

Keywords: Series parallel digraphs; Clique-width; Complexity

*Article*

# The Clique Width of Two Classes of Series-Parallel Digraphs

**Ruzayn Quaddoura**

Department of Computer Science, Zarqa University, Jordan; ruzayn@zu.edu.jo

**Abstract:** The concept of graphs with clique-width at most $k$ was first introduced by Courcelle et al.to be the graphs that can be characterized using k-expressions derived from graph operations that use k labels of vertices. If the clique-width for some graph is bounded then a grand number of algorithmic problems, in general NP-hard, can be solved in polynomial time when restricted to this graph. This important fact motivated the researchers to prove that the clique-width of certain graphs is bounded. Following this research direction, we prove in this paper that the clique-width of series-parallel digraphs is at most 6 and we present an $O(n)$ time algorithm to construct a 6-expression for this class of digraphs. In another part, we present a linear time recognition algorithm for a similar class of series-parallel digraphs and prove that the clique-width of this class is at most 3.

**Keywords:** series parallel digraphs; clique width; complexity

## 1. Introduction

Courcelle et al. in [1] characterized graphs with clique-width at most k as those graphs that can be generated through expressions involving graph operations utilizing $k$ vertex labels. Clique width generalizes another significant graph parameter, tree-width, in the sense that any graph has a bounded tree-width also has a bounded clique width [6]. The concept of clique width was established in 1993 [1], and since then a large number of studies concerning this concept have been published. The subjects in the literature that concern clique width have a theoretical or an algorithmic standpoint and can be summarized by one of the following questions:

- What is the complexity of computing the clique width?
- When the clique width of a graph is unbounded?
- Is there a general algorithm to recognize a graph of bounded clique width?
- Are there general algorithms to solve some optimization problems in a graph of bounded clique width?
- How to construct the clique width of a given graph?

The subject of complexity of computing the clique width has been resolved by Fellows et. al. in [14]. They proved that the problem of determining the clique width of a graph is NP-complete in general. The subject of determining the necessary and /or sufficient conditions to be the clique width of a graph unbounded is till now an open question. This question is motivated from the fact that the clique width for certain class of graphs can be unbounded, for example the class of $(4k_1, C_4, C_5, C_7)$-free graphs and not chordal [8]. The subject of recognizing a class of graphs of bounded clique width is resolved partially; for example, it is proved in [5] that any graph of clique width at most 3 can be recognized in $O(n^2m)$ time. The subject of solving some optimization problems in a graph of bounded clique width is the significance of these graph invariants. It stems from the fact that many problems that are NP-hard in general admit polynomial-time solutions when restricted to graphs with bounded clique width (see for example [2,3,15]). Among these problems are Hamiltonian path or Hamiltonian cycle problems, computing the minimum size of a maximal matching and several partition problems (partition into cliques or triangles or complete bipartite sub-graphs or perfect matching's or forests) [16], computing the chromatic number and computing the minimum size of a dominating set of vertices or edges [4], computing the maximum size of a cut [7], finding vertex-disjoint Paths [13]. The construction of the clique width for a given graph has been studied extensively. For example, in [9], it is proved that the clique width of Cactus graphs is at most 4 and the construction of 4-expression of these graphs can be done in polynomial time. Also in [10] it is proved that the clique width of polygonal tree graphs can be done by constructing a 5-expression in a polynomial time. Recently, the clique width for series parallel undirected graphs is studied in [12], where it is proved that the clique width of these graphs is at most 5 and an $O(n^2)$ algorithm is presented to construct a 5-expression. Following this last direction of research, we prove that the clique width of the famous class of series parallel directed graphs (digraphs) is at most 6 and we propose an $O(n)$ time algorithm to compute a 6-expression of this class. From other part, we introduce a new definition of a similar class of series parallel digraphs and present for which a linear time recognition algorithm. We prove that the clique width of this new class of digraphs is at most 3.

## 2. Preliminaries

A directed graph (or a digraph for short) $G = (V, E)$ is defined by two sets, $V(G)$ or simply $V$ is the vertices set and $E(G)$ or simply $E$ is the arcs set. Every arc of $E$ is an ordered pairs of vertices of $V$. The number $n$ indicates to the number of vertices of $G$ and the number $m$ indicates to the number of edges of $G$.If $(x, y) \in E$ then $x$ is called a predecessor of $y$ and $y$ is called a successor of $x$. The set of all predecessors of a vertex $x$ is denoted by $N^+(x)$, and the set of all successors of $x$ is denoted by $N^-(x)$). The set of neighbors of $x$ is the set $N(x) = N^+(x) \cup N^-(x)$. The number $|N^+(x)|$ is called the positive degree of $x$ and denoted by $d^+(x)$, and the number $|N^-(x)|$ is called the negative degree of $x$ and denoted by $d^-(x)$, the degree of a vertex x is the number $d(x) = |N(x)|$. A vertex $x$ is called a source if $|N^+(x)| = 0$ and is called a sink if $|N^-(x)| = 0$. Given a subset $X$ of the vertices set $V$, the sub-graph induced by $X$ will be denoted by $G[X]$. A path of length $k$ is a sequence of vertices $x_1, x_2, \dots, x_k$ such that any two consecutive vertices form an arc. A path $x_1, x_2, \dots, x_k$ is called a circuit if $x_1, = x_k$ and $k \geq 2$. A directed acyclic graph denoted by DAG is a digraph with no circuit. A chain of length $k$ is a sequence of vertices $x_1, x_2, \dots, x_k$ such that $(x_i, x_{i+1})$ or $(x_{i+1}, x_i)$ is an arc. If $x_1 = x_k$ and $k \geq 2$ the chain is called a cycle. An arc $(x, y)$ is called transitive arc if there is a path from $x$ to $y$ of length at least 2. A bipartite graph $G = (B \cup W, E)$ is given by a set of black vertices $B$ and a set of white vertices $W$ and a set of edges $E \subseteq B \times W$. A graph $G$ is called $F$-free where $F$ is a set of graphs when $G$ does not contain an induced sub-graph isomorphic to a graph of $F$.

The clique width of a graph G is the smallest number of labels required to construct $G$ using the four operations listed below:

- The operation $i(v)$) to create a new vertex $v$ has the label $i$.
- The operation $G \oplus H$ to make a union of two disjoint labeled graphs $G$ and $H$.
- The operation $\eta_{i,j}(G)$ to add in the labeled graph $G$ an edge (or an arc in case of digraphs) from each vertex with label $i$ to each vertex with label $j$ $(i \neq j)$.
- The operation $\rho_{i \to j}(G)$ to change in the labeled graph $G$ every label $i$ to label$j$.

Using these four operations, any graph may be defined by an algebraic expression. For example, the graph $G = (V, E)$ where $V = \{a, b, c, d\}$ and $E = \{ab, , bc, cd, da, db\}$ can be defined by the following expression:

$$\eta_{3,1}(d(3) \oplus (\rho_{3\to 2}(\rho_{2\to 1}(\eta_{2,3}(3(c)(\eta_{1,2}(2(b) \oplus 1(a)))))))$$

If an expression uses at most k different labels, it is referred to as a k-expression.

## 3. Clique Width of Series Parallel Digraphs (SP DAGs)

Series-parallel digraphs, or SP DAGs for short, are a class of directed graphs that play a significant role in graph theory and find extensive applications in various real-world scenarios. It has been defined by Tarjan et al. in [11] to be the digraph that can be constructed starting of the vertices set and recursively using two fundamental operations: series composition and parallel composition. More precisely:

**Definition 1:** *A SP DAG is defined recursively as follows:*

*A DAG containing only one vertex is a SP DAG.*

*If $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are two SP DAGs then the DAG constructed by each of the following operations is also a SP DAG :*

*Parallel composition: $G = G_1PG_2 = (V_1 \cup V_2, E_1 \cup E_2)$.*

*Series composition: $G = G_1SG_2 = (V_1 \cup V_2, E_1 \cup E_2 \cup T_1 \times S_2)$ where $T_1$ is the set of sink vertices of $G_1$ and $S_2$ is the set of source vertices of $G_2$.*

A SP DAG $G$ can be represented by a binary decomposition tree $T(G)$ that reflects the construction of $G$ starting of its vertices using series and parallel operations as following:

- The leaves correspond to the vertices of $G$.
- Let $\alpha$ be An internal node and $\alpha_1, \alpha_1$ are respectively the left and right child of $\alpha$, then $\alpha$ is labeled by $P$ (resp. $S$) if $G[\alpha] = G[\alpha_1]PG[\alpha_2]$ (resp. $G[\alpha] = G[\alpha_1]SG[\alpha_2]$) where $G[\alpha_i], i = 1,2$ is the sub-graph of $G$ induced by the set of vertices having $\alpha_i$ as their least common ancestor.

Figure 1 illustrates an example of a SP DAG and its binary decomposition tree. It is worthy mentioned that the two children of a $S$-node are ordered according to the series operation of that node. Tarjan et. al. in [11] proved that the construction of $T(G)$ when $G$ is a SP DAG can be done in $O(n + m)$ time complexity. This binary decomposition tree of a SP DAG is the key of our computing to its clique width.

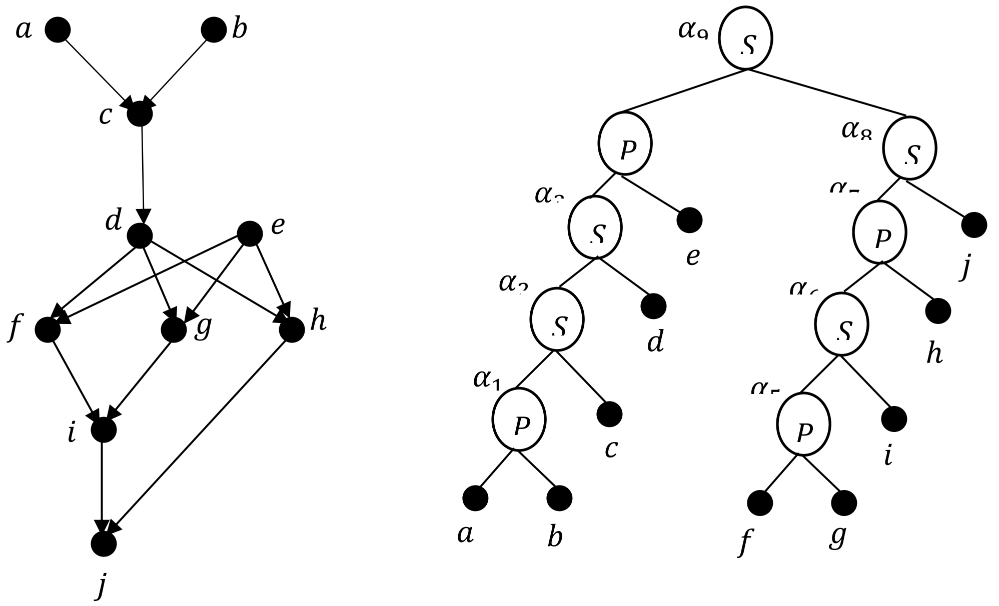


**Figure 1.** A SP DAG and its decomposition binary tree.

**Theorem 1: The clique width of a SP DAG $G$ is at most 6.**

**Proof:** Let $\alpha$ be an internal node of $T(G)$ and $\alpha_1$, $\alpha_2$ are respectively the left and right child of $\alpha$. We will construct a 6-expression of $G[\alpha]$ starting of a similar 6-expression of $G[\alpha_1]$ and 6-expression of $G[\alpha_2]$ For $i = 1,2$, we can partition the vertices set of $G[\alpha_i]$ into at most four sets $S(\alpha_i), T(\alpha_i), X(\alpha_i)$ and $Y(\alpha_i)$, where $S(\alpha_i)$ is the set of non-isolated sources of $G[\alpha_i]$, , $T(\alpha_i)$ is the set of non-isolated sinks of $G[\alpha_i]$, $X(\alpha_i)$ contains the set of isolated vertices of $G[\alpha_i]$, and $Y(\alpha_i)$ is the set of vertices that are not sources, not sinks, and not isolated in $G[\alpha_i]$. We suppose that the label of every vertex of $S(\alpha_1)$ is 1, the label of every vertex of $T(\alpha_1)$ is 2, the label of every vertex of $X(\alpha_1)$ is 1, and the label of every vertex of $Y(\alpha_1)$ is 5.We express this in notation by $1(S(\alpha_1)), 2(T(\alpha_1)), 1(X(\alpha_1))$ and $5(Y(\alpha_1))$. Similarly, we suppose that the label of every vertex of

$\mathrm{S}(\alpha_2)$ is 3, the label of every vertex of $\mathrm{T}(\alpha_2)$ is 4, the label of every vertex of $\mathrm{X}(\alpha_2)$ is also 4, and the label of every vertex of $\mathrm{Y}(\alpha_2)$ is 6. We can express the sub-graph $\mathrm{G}[\alpha_1]$ by $\mathrm{H}_1 \oplus \mathrm{H}_2$ where:

$$\mathrm{H}_1 = \mathrm{G}[1(\mathrm{S}(\alpha_1)) \oplus 5(\mathrm{Y}(\alpha_1)) \oplus 2(\mathrm{T}(\alpha_1))]$$
$$\mathrm{H}_2 = \mathrm{G}[1(\mathrm{X}(\alpha_1))]$$

Similarly, we can express the sub-graph $\mathrm{G}[\alpha_2]$ by $\mathrm{H}_3 \oplus \mathrm{H}_4$ where:

$$\mathrm{H}_3 = \mathrm{G}[3(\mathrm{S}(\alpha_2)) \oplus 6(\mathrm{Y}(\alpha_2)) \oplus 4(\mathrm{T}(\alpha_2))]$$
$$\mathrm{H}_4 = \mathrm{G}[4(\mathrm{X}(\alpha_2))]$$

Figure 2.a illustrates this decomposition of $G[\alpha_i]$. Without loss of generality, and for simplification of reading, we represented in this Figure each set of $S(\alpha_i), T(\alpha_i), X(\alpha_i)$ and $Y(\alpha_i), i = 1,2$, by a single vertex. To construct a 6-expression for $G[\alpha]$ we distinguish two cases according to the type of $\alpha$:

Suppose that$\alpha$ is a $P$-node. In this case, the set $S(\alpha_1) \cup S(\alpha_2)$ is the set of non-isolated sources of $G[\alpha]$, the set $T(\alpha_1) \cup T(\alpha_2)$ is the set of non-isolated sinks of $G[\alpha]$, the set $X(\alpha_1) \cup X(\alpha_2)$ is the set of isolated vertices of $G[\alpha]$, and the set $Y(\alpha_1) \cup Y(\alpha_2)$ is the set of vertices that are not sources, not sinks, and not isolated in $G[\alpha]$. Hence, the expression $t_1 \oplus t_2$constructs $G[\alpha]$ using 6 labels where:

$$t_1 = H_1 \oplus \rho_{3\to1}(\rho_{6\to5}(\rho_{4\to2}(H_3)))$$
$$t_2 = H_2 \oplus \rho_{4\to1}(H_4)$$

Figure 2.b, illustrate the construction of $G[\alpha]$ using the expressions $t_1$and $t_2$. Suppose that $\alpha$ is a $S$-node. Since $G[\alpha]$ is connected, it does not contain isolated vertices. Since $\alpha_1, \alpha_2$ are respectively the left child and right child of $\alpha$ then, every vertex of $X(\alpha_1)$ must be a sink in $G[\alpha_1]$ and every vertex of $X(\alpha_2)$ must be a source in $G[\alpha_2]$, thus every vertex of $X(\alpha_1)$ would be a source in $G[\alpha]$ and every vertex of $X(\alpha_2)$ would be a sink in $G[\alpha]$. Therefore, $S(\alpha_1) \cup X(\alpha_1)$ is the set of sources of $G[\alpha]$, the set $X(\alpha_2) \cup T(\alpha_2)$ is the set of sinks of $G[\alpha]$, and the set $Y(\alpha_1) \cup T(\alpha_1) \cup S(\alpha_2) \cup Y(\alpha_2)$ is the set of vertices that are not sources, not sinks, and not isolated in $G[\alpha]$. Hence, the following ordered series of expressions constructs $G[\alpha]$ using 6 labels:

$$t_1 = \eta_{3,4}(\rho_{1\to3}(H_2) \oplus H_4), t_2 = \eta_{2,4}(t_1 + H_1), t_3 = \rho_{3\to6}(\rho_{6\to5}(H_3)),$$
$$t_4 = \eta_{2,6}\left(\eta_{3,6}(t_3 \oplus t_2)\right), t_5 = \rho_{3\to1}(\rho_{6\to5}(\rho_{2\to5}(\rho_{4\to2}(t_4))))$$

Figure 2.c illustrate the construction of $G[\alpha]$ using the expressions $t_1, t_2, t_3, t_4$ and $t_5$. Now, by traversing $T(G)$ in post order and calculating a 6-expression for every internal node of $T(G)$, we conclude that the whole digraph $G$ represented by the root of $T(G)$ can be constructed by a 6-expression. ◎

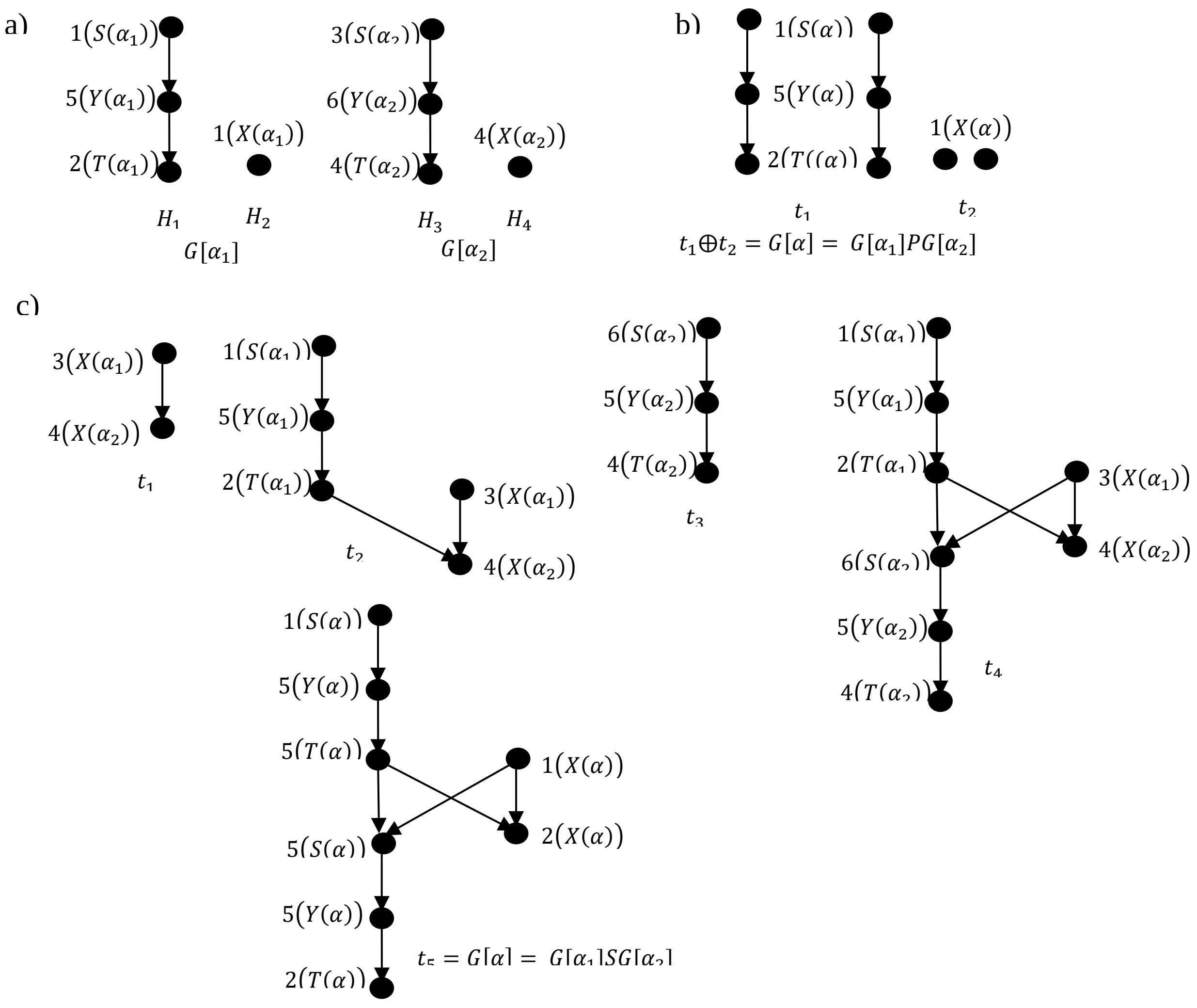


**Figure 2.** a) The sub-graphs $G[\alpha_1]$ and $G[\alpha_2]$, b) the construction of $G[\alpha] = G[\alpha_1]PG[\alpha_2]$, c) the construction of $G[\alpha] = G[\alpha_1]SG[\alpha_2]$

The construction a SP DAG by a 6-expression using the method described in the proof of Theorem 1 can be translated into the Algorithm Construction a 6-expression of a SP DAG. In this Algorithm, we define an empty expression, denoted by $\emptyset$, to be an expression of an empty set of vertices. For any expression $t$, we consider that $t \oplus \emptyset = t$, so $\eta_{i,j}(t \oplus \emptyset) = \eta_{i,j}(t)$ and $\rho_{i \to j}(t \oplus \emptyset) = \rho_{i \to j}(t)$. Our Algorithm associates with every node $\alpha$ of $T(G)$ two expressions $t_1(\alpha)$ and $t_2(\alpha)$, where $t_1(\alpha)$ represents the construction of the sub-graph induced by the non-isolated vertices of $G[\alpha]$ and $t_2(\alpha)$ represents the construction of isolated vertices of $G[\alpha]$. It may be at most one of $t_1(\alpha)$ and $t_2(\alpha)$ empty. The labels in the end of computing $t_1(\alpha)$ or $t_2(\alpha)$ represent implicitly whether the corresponding vertices are non-isolated sources (labeled by 1), non-isolated sinks (labeled by 2), isolated vertices (labeled by 1), or vertices that are not sources, not sinks and not isolated in $G[\alpha]$ (labeled by 5). Initially, in step 1, every vertex is labeled by 1 since the sub-graph induced by a leaf consists only of one isolated vertex that consider as a source. Step 2 determines the left child $\alpha_1$ and the right child $\alpha_2$ of an internal node $\alpha$. Step 3 constructs the expression $t_1(\alpha)$ and $t_2(\alpha)$ for a $P$-node $\alpha$ to be respectively the disjoint union of $t_1(\alpha_1), t_1(\alpha_2)$ and the disjoint union of $t_2(\alpha_1), t_2(\alpha_2)$. Steps 4 to 13 constructs the expressions $t_1(\alpha)$ and $t_2(\alpha)$ for a $S$-node $\alpha$ as following: Step 5 changes the label of non- isolated sources of $G[\alpha_2]$ from 1 to 3, the label of non-isolated sinks from 2 to 4, and the other non-isolated vertices from 5 to 6; Step 6, changes the label of isolated vertices of $G[\alpha_2]$ from 1 to 4 as these vertices are sinks in $G[\alpha_2]$. Step 7 changes the label of isolated vertices of $G[\alpha_1]$ from 1 to 3. The steps from 8 to13 construct the expression $t_1(\alpha)$ in the same way as in Figure 2.c. The expression $t_2(\alpha)$ must be empty because of $G[\alpha]$ is connected. Since the number of nodes in $T(G)$ is $O(n)$ and every step can be done in $O(1)$ time, the total time complexity of this Algorithm is $O(n)$.

**Algorithm Construction a 6-expression of a SP DAG**

```
Input: A binary decomposition tree $T(G)$ of a SP DAG $G$
Output: A 6-expression of $G$.
Let $\alpha$ be a node on a post order traversal of $T(G)$
    1. If $\alpha$ is a leaf then $t_1(\alpha) = \emptyset, t_2(\alpha) = 1(\alpha)$
    2. Else let $\alpha_1$ and $\alpha_2$ be respectively the left and right children of $\alpha$
    3. if $\alpha$ is a $P$-node then $t_1(\alpha) = t_1(\alpha_1) \oplus t_1(\alpha_2)$, $t_2(\alpha) = t_2(\alpha_1) \oplus t_2(\alpha_2)$
    4. else\\$\alpha$ is a $S$-node
    5.         $t_1(\alpha_2) = \rho_{1\to3}(\rho_{5\to6}(\rho_{2\to4}(t_1(\alpha_2))))$
    6.         $t_2(\alpha_2) = \rho_{1\to4}(t_2(\alpha_2))$
    7.         $t_2(\alpha_1) = \rho_{1\to3}(t_2(\alpha_1))$
    8.         $E_1 = \eta_{3,4}(t_2(\alpha_1) \oplus t_2(\alpha_2))$
    9.         $E_2 = \eta_{2,4}(E_1 + t_1(\alpha_1))$
    10.        $E_3 = \rho_{3\to6}\left(\rho_{6\to5}\left(t_1(\alpha_2)\right)\right)$
    11.        $E_4 = \eta_{2,6}\left(\eta_{3,6}(E_3 \oplus E_2)\right)$
    12.        $E_5 = \rho_{3\to1}(\rho_{6\to5}\left(\rho_{2\to5}(\rho_{4\to2}(E_4))\right))$
    13.        $t_1(\alpha) = E_5, t_2(\alpha) = \emptyset$
```

**Figure 3.** illustrates an example of the application of Algorithm Construction a 6-expression of a SP DAG for the digraph $G$ and its binary decomposition tree $T(G)$ shown in Figure 1. The visited nodes according to post order traversal of $T(G)$ is defined by the symbols $\alpha_i, 1 \le i \le 9$, written near every internal node of $T(G)$. Every line in Figure 3 shows the executed steps for constructing $G[\alpha_i], 1 \le i \le 9$. The vertices of left and right children of $\alpha_i, 1 \le i \le 9$ are represented respectively by a black color and a white color. The arrow and the number above it indicates the result of the corresponding step of the Algorithm. The absence of a result for a specific step means that this step has no effect on the construction.

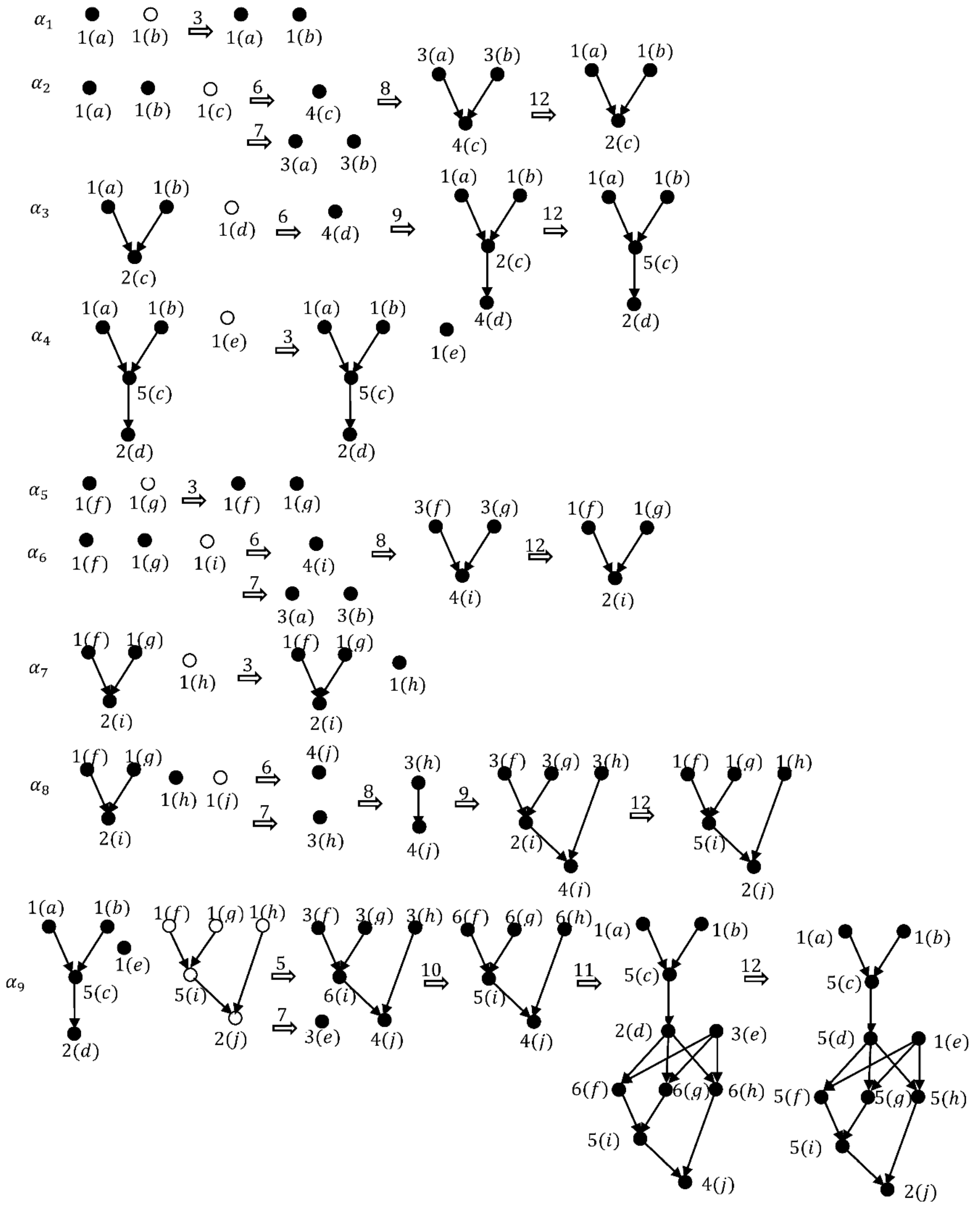


**Figure 3.** application of Algorithm Construction a 6-expression of a SP DAG for the digraph $G$ in Figure 1.

## 3. Similar Series Parallel Digraphs (SSP DAGs)

**Definition 1:** *A SSPDA G is defined recursively as follows:*
*A DAG having a single vertex is a SSP.*
*If $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are two SSP DAGs then the DAG constructed by each of the following operations is also a SSP:*
*Parallel composition:$G = G_1PG_2 = (V_1 \cup V_2, E_1 \cup E_2)$.*
*Series composition:$G = G_1SG_2 = (V_1 \cup V_2, E_1 \cup E_2 \cup S_1 \times S_2)$ where $S_i$ is the set of source vertices of $G_i, i =$ 1,2.*

It is obvious that a SSP DAG is without transitive arcs. In the same way as in the class of SP DAGs, a SSP DAG $G$ can be represented by a binary decomposition tree $T(G)$ that reflects the construction of $G$ starting of its vertices using series and parallel operations. Also as in the binary decomposition tree of a SP DAG, the two children of a $S$-node are ordered according to the series operation of that node. Figure 4 represents a SSP DAG $G$ and its binary decomposition tree $T(G)$.

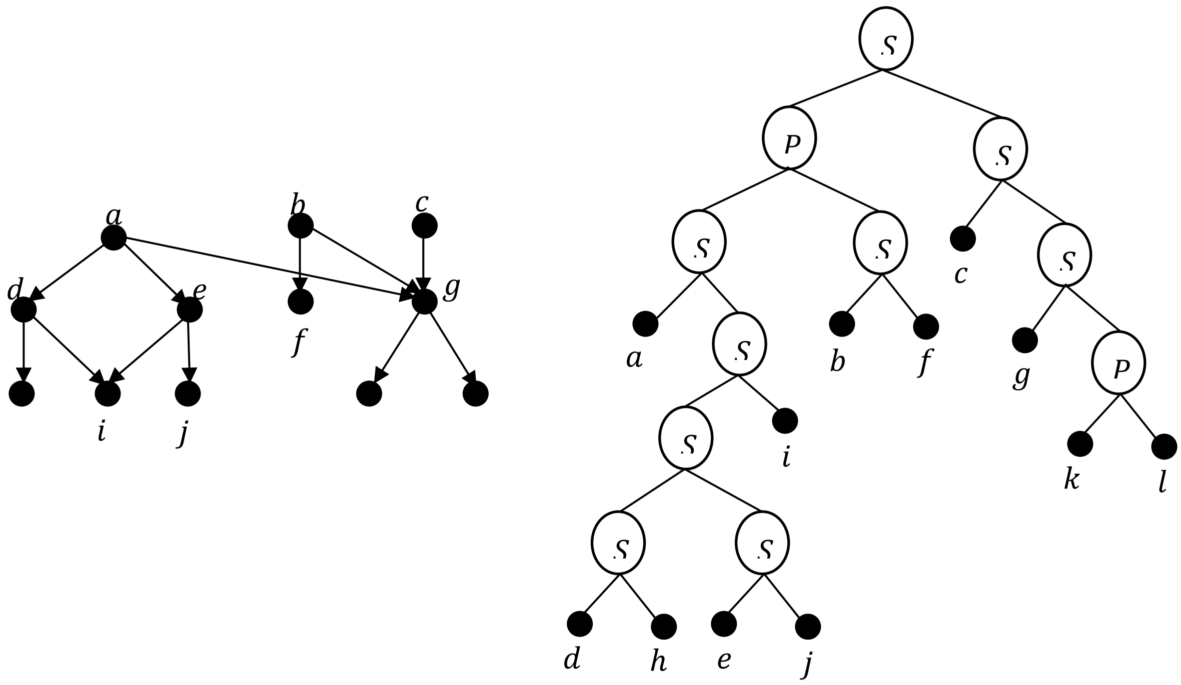
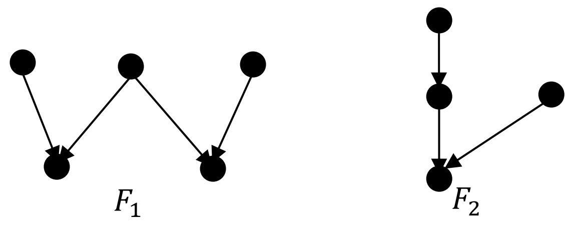


**Figure 4.** A SSP DAG and its binary decomposition tree.

The following theorem is the key of our recognition algorithm of SSP DAGs.

**Theorem 2:** *Let G be a connected DAG without transitive arcs. $G$ is a SSP if and only $G$ is $\{F_1, F_2\}$-free.*



**Proof:** Suppose that Gis a SSPDAG and let's show that Gis $\{F_1, F_2\}$-free.

**Claim 1:** *Let $y_1, y_2 \in V(G)$ such that $N^-(y_1) \cap N^-(y_2) \neq \emptyset$ then $N^-(y_1) \subseteq N^-(y_2)$ or $N^-(y_2) \subseteq N^-(y_1)$.*

**Proof:** Since G is a SSP, every arc in G is created by a series operation. According to the series operation, for any vertex $y$, all the arcs $\{(x, y) | \, x \in N^-(y)\}$ are created by the same series operation. So, if there is two vertices $y_1, y_2$ such that $N^-(y_1) \cap N^-(y_2) \neq \emptyset$ then if the two sets of arcs $\{(x, y_1) | \, x \in N^-(y_1)\}$ and $\{(x, y_2) | \, x \in N^-(y_2)\}$ are created by the same series operation then $N^-(y_1) = N^-(y_2)$.Suppose that the set of arcs $\{(x, y_1) | \, x \in N^-(y_1)\}$ is created by a series operation $S_1$ and the set of arcs $\{(x, y_2) | \, x \in N^-(y_2)\}$ is created by a series operation $S_2$ where $S_1$ precedes $S_2$. Since $N^-(y_1) \cap N^-(y_2) \neq \emptyset$, the vertex $y_2$ was a source during the operation $S_2$, so $N^-(y_1) \subseteq N^-(y_2)$. If $S_2$ precedes $S_1$then $N^-(y_2) \subseteq N^-(y_1)$. ■

By Claim 1, $G$is $F_1$-free.

**Claim 2:** *Let$y \in V(G)$, for every $x_1, x_2 \in N^-(y)$, $N^-(x_1) = N^-(x_2)$*

*Proof: Let $x \in N^-(x_1)$. By the definition of the series operation, the arc $(x_1, y)$ has been created by a series operation $S_1$ that precedes the series operation $S_2$ for which the arc $(x, x_1)$ has been created. Since the arcs*

*$(x_1, y), (x_2, y)$ have been created by the same series operation $S_1$ then during the series operation $S_2$ there was as sources $x_1$ and $x_2$,therefore $(x, x_2) \in E$, this implies that$N^-(x_1) = N^-(x_2)$.* ■

By Claim 2, $G$is $F_2$ -free suppose now $G$ is a connected DAG without transitive arcs and $\{F_1, F_2\}$-free. Let's show that $G$is SSP DAG. Let $S$ be the set of all sources of $G$and $Q = G[V - S]$.

**Claim 3:** *Every vertex of $Q$ that is a successor of a vertex of $S$ is a source of $Q$.*

*Proof: Let $y$ be a vertex of $Q$ that is not a source and a successor to a vertex $x \in S$. Let $z$ be a source in $Q$ such that $z$ is an ancestor of $y$. Since $G$ does not contain transitive arcs,$(x, u) \notin E$ for every vertex $u$ located on the path going from $z$ to $y$. Suppose that $z$ is a predecessor of $y$. Since $S$ is the set of all sources of $G$, there is a source $t \in S$ such that $(t, z) \in E$. Since $G$ does not contain transitive arcs, the set $\{t, z, y, x\}$ induces the configuration $F_2$, a contradiction. Suppose that $z$ is not a predecessor of $y$, let $u_1$ be a predecessor of yin $Q$ and $u_2$ is a predecessor of $u_1$. Since $G$ does not contain transitive arcs then $\{x, y, u_1, u_2\}$ induces the configuration $F_2$, a contradiction.* ■

*Let $C_1, \dots, C_k$ be the connected components of $Q$ and $S'$ is the set of sources of $Q$.*

**Claim 4:** *If a source $x \in S$ is a predecessor to a source $y$ of some connected component $C_i, 1 \le i \le k$ then $x$ is a predecessor to every source of $C_i$.*

*Proof: Suppose the contrary, then there a source$y'$ in $C_i$ such that $(x, y') \notin E$. Since $y$ and $y'$ are sources in $C_i$ and $G[C_i]$ is connected there is a chain in $G[C_i]$ that connects $y$ and $y'$. Without loss of generality, let $z \in C_i$ such that $(y, z), (y', z) \in E$. Now the set $\{x, y', y, , z\}$ induces the configuration $F_2$, a contradiction.* ■

If $k = 1$ then by Claim 4,$G[S \cup S']$ is a bipartite complete. It is clear that $G$ admit a series decomposition into $S$ and $V(G) - S$.

Suppose $k \ge 2$ and $G[S \cup S']$ is not abipartite complete. Since $G$ is connected, $G[S \cup S']$ must be also connected. We claim that there is a vertex $y \in S'$ such that for every $x \in S$, $(x, y) \in E$. Suppose the contrary, then for every vertex $y \in S'$ there is avertex $x \in S$ such that $(x, y) \notin E$. Let $y_1, y_2 \in S'$ and $x_1, x_2 \in S$ such that $(x_1, y_1), (x_2, y_2) \in E$ and $(x_1, y_2), (x_2, y_1) \notin E$. Since $G[S \cup S']$ is connected, there is a chain in $G[S \cup S']$ that connects $(x_1, y_1)$ and $(x_2, y_2)$. Without loss of generality, let $x \in S$ such that $(x, y_1), (x, y_2) \in E$, then $\{x, x_1, y_1, x_2, y_2\}$ induces the configuration $F_1$, a contradiction. Let $Y = \{y \in S' : \forall\, x \in S, (x, y) \in E$ and $C_1, \dots, C_r$ be the connected components of $Q$ that contain the vertices of $Y$. It is proved in Claim 4 that every source of every connected component $C_i$ $(1 \le i \le r)$ is a successor of every source in $S$. Therefore $G$ admit a series decomposition into $V(G) - (C_1, \dots, C_r)$ and $C_1, \dots, C_r$. It follows that we can always reduce $G$ to its vertices set by a parallel decomposition and a series decomposition, this implies that $G$ is a SSP DAG. ◎

## 4. Recognition of SSP DAGs

We present in this section a linear Algorithm to recognize if an arbitrary DAG is SSP or not. We will take into account the following sort of the vertex set for a DAG $G$.

**Definition 3:** *Let $G$ be a DAG and $S$ is the set of sources of $G$, let $A_1 = S, A_2 = N^+(A_1), \dots, A_p = N^+(A_{p-1})$. The sort $\rho = \{A_1, \dots, A_p\}$ is called a topologically sort of $V(G)$.*

Our Algorithm uses the following result:

**Lemma 4:** *Let $G$ be a DAG and let $\rho = \{A_1, \dots, A_p\}$ be the topologically sort of $V(G)$. Then $G$ is a SSP DAG if and only if the following conditions are verified:*

*a) For every $(x, y) \in E(G)$ there is $1 \le i \le p - 1$ such that $x \in A_i, y \in A_{i+1}$;*

*b) For every $2 \le i \le p$, $G[A_{i-1} \cup A_i]$ is a bipartite $F_1$-free graph;*

*c) Let $C = (C_i, C_{i+1})$ be a connected component of $G[A_i \cup A_{i+1}]$, $2 \le i \le p - 1$ then for every $x, y \in C_i, N^-(x) = N^-(y)$.*

**Proof:** We can remark that $y \in Aj$ ,$j > i + 1$ if and only if $G$contains a transitive arc or $G$ contains the configuration $F_2$. The conditions a and b assure that $G$ is $F_1$-free, the conditions a and c assure that $G$is $F_2$-free. ◎

The following Lemma provides a simple method for verifying the condition b of Lemma 4.

doi:10.20944/preprints202403.1498.v1

**Lemma 5:** *Let $G = (B \cup W, E)$ be a bipartite DAG of depth one. $G$ is $F_1$-free if and only if for every $x, y \in W$, $N^-(x) \subseteq N^-(y)$ or $N^-(x) \cap N^-(y) = \emptyset$.*

**Proof**: Obviously if $G$ is $F_1$-free then the *conditions* of Lemma must be verified. On the contrary, if one of these conditions is verified then every connected component of $G$ contains an isolated vertex or a universal vertex. Therefore we can reduce $G$ to its vertices set by a parallel and series decomposition.◎

The following Algorithm contains the procedures for detecting the conditions of Lemma 4. Step 1 tests whether $G$ contains a transitive arc or the configuration $F_1$. Step 2 tests whether $G$ contains the configuration $F_2$ or not. Step 3 tests whether $G$ contains between two consecutive levels of $\rho$ the configuration $F_1$ or not. Let's show that the time complexity of this algorithm is $O(n+m)$. The determination of $\rho$ and testing whether every arc of $G$ is located between two consecutive levels of $\rho$ requires $O(n+m)$ time. The success of first step guarantees that the sets of edges $E(G_i), i = 1,\dots,p-1$, constitute a partition of $E(G)$. Therefore, testing the inclusion relation of the vertices $N^-(y_j), j = 1,\dots,r$ for every $A_{i+1} = \{y_1,\dots,y_r\}, i = 1,\dots,p-1$, using the mark procedure described in step 2, can be executed in time $O(V(G_i) + E(G_i))$, so the second step also requires $O(n+m)$ time. The non-empty sets $C_1, \dots C_k$ produced in step 2 are the input of step 3 for every $G_i, i = 1,\dots,p - 1$. Indeed $C_i \cup \{N^+(x): x \in C_i\}$ are the connected components of $G_i, i = 1,\dots,p - 1$. Now, to test the condition c of Lemma 4, it is enough to compare for every $x \in C_i$, the set $N^-(x)$ with $N^-(x_1)$ where $x_1$ is an arbitrary vertex of $C_i$. This can be done in $O(V(G_i) + E(G_i))$ time. Hence the total time complexity of this Algorithm is $O(n+m)$.

---

**Algorithm Recognition of SSP DAG**

```
Input: A DAG $G = (V, E)$.
Output: The message “Success” if$G$ is SSP, otherwise“’ Failure”
Let $\rho = \{A_1,\dots,A_p\}$be the topologically sort of $V(G)$.
Step 1
For every $(x, y) \in E(G)$ do
If $x \in A_i$ and $y \in Aj$ with $j > i + 1$ then
        Exit with the message “Failure”
End If
End For
Step 2
Let $G_1 = G[A_1 \cup A_2], \dots, G_{p-1} = G[A_{p-1} \cup A_p]$
For $1 \le i \le p - 1$
  Let $A_{i+1} = \{y_1, \dots, y_r\}$ such that $d^-(y_1) \ge \dots \ge d^-(y_r)$
  Let $C_1 = \dots = C_r = \emptyset$
  For $1 \le j \le r$
    If there is a vertex $x \in N^-(y_j)$ that is marked by $k \ne j$ then
      If there is a vertex $x \in N^-(y_j)$ that is not marked by $k$ then
           Exit with failure message
      End If
    Else Mark every vertex in $N^-(y_j)$ by $j$; $C_j = C_j \cup N^-(y_j)$
    End If
    End For
    Step 3
    Let $C_1, \dots C_k$ be the non-empty sets produced in step 2
    For $1 \le i \le k$
        Let $C_i = \{x_1, \dots, x_s\}$
        For $2 \le j \le s$
            If $N^-(x_j) \ne N^-(x_1)$ then Exit with failure message
        End For
    End For
Return Success
```

## 5. Clique Width of SSP DAGs

The computation of clique width of SSP DAGs can be done in a similar way to the one we did for computing the clique width of SP DAGs. The following theorem shows this computation.

**Theorem 1:** *The clique width of a SSP DAG $G$ is at most 3.*

**Proof:** Let $T(G)$ be a binary decomposition tree of a SSP DAG $G$. Let $\alpha$ be an internal node of $T(G)$ and $\alpha_1$, $\alpha_2$ are respectively the left and right children of $\alpha$. We will construct a 3-expression of $G[\alpha]$ starting of a similar 3-expression of $G[\alpha_1]$ and 3-expression of $G[\alpha_2]$ For $i = 1,2$, we can partition the vertices set of $G[\alpha_i]$ into at most two sets $S(\alpha_i)$ and $Y(\alpha_i)$, where $S(\alpha_i)$ is the set of sources of $G[\alpha_i]$ and $Y(\alpha_i)$ is the set of remaining vertices of $G[\alpha_i]$. We must point out here that if $G[\alpha_i]$ contains an isolated vertex then this vertex is considered as a source in $G[\alpha_i]$. We suppose that the label of every vertex of $S(\alpha_1)$ is 1 and the label of every vertex of $Y(\alpha_1)$ is 2. Similarly, we suppose that the label of every vertex of $S(\alpha_2)$ is 3 and the label of every vertex of $Y(\alpha_2)$ is 2. So we can express the sub-graph $G[\alpha_1]$ and $G[\alpha_2]$ as:

$$G[\alpha_1] = G[1(S(\alpha_1)\oplus 2(Y(\alpha_1))]$$
$$G[\alpha_2] = G[3(S(\alpha_1)\oplus 2(Y(\alpha_1))]$$

Suppose that $\alpha$ is a $P$-node. The set $S(\alpha_1) \cup S(\alpha_2)$ is the set of sources of $G[\alpha]$, and the set $Y(\alpha_1) \cup Y(\alpha_2)$ is the set of remaining vertices of $G[\alpha]$. Hence, the expression $t$ constructs $G[\alpha]$ using 3 labels where:

$$t = G[\alpha_1]\oplus\rho_{3\to 1}(G[\alpha_2])$$

Suppose that $\alpha$ is a $S$-node. By the definition of series operation, the set of sources of $G[\alpha]$ is the set $S(\alpha_1)$, and the set of remaining vertices of $G[\alpha]$ is the set $S(\alpha_2) \cup Y(\alpha_1) \cup Y(\alpha_2)$. Hence the expression $t$constructs $G[\alpha]$ using 3 labels where:

$$t = \rho_{3\to 1}(\eta_{1,3}(G[\alpha_1]\oplus G[\alpha_2]))$$

## 6. Conclusion

We show in this paper that the clique width of a SP DAG $G$ is at most 6 and the construction of a 6-expression of $G$can be done in $O(n)$ time complexity using a binary decomposition tree$T(G)$. On other hand, we defined the class of digraphs SSP as a similar class of SP DAGs and proved that this new class can be recognized in linear time complexity. We proved that the clique width of a SSP DAG is at most 3. The construction of a 3-expression of a SSP DAG $G$ requires to construct a binary decomposition tree$TG)$ that we believe to be done in linear time.

## References

1. B. Courcelle, J. Engelfriet, G. Rozenberg. Handle-rewriting hypergraph grammars, Journal of Computer and System Sciences 46 (1993) 218-270.
2. B. Courcelle, J.A. Makowsky, U. Rotics. Linear time solvable optimization problems on graphs of bounded clique width, Theory of Computing Systems 33 (2000) 125-150.
3. B. Courcelle. The expression of graph properties and graph transformations in monadic second-order logic, in: Handbook of Graph Grammars and Computing by Graph Transformation, vol. 1, World Sci. Publishing, River Edge, NJ, 1997, pp. 313_400.
4. D. Kobler, U. Rotics. Edge dominating set and colorings on graphs with fixed clique width, Discrete Applied Mathematics 126 (2003) 197-221.
5. D.G. Corneil, M. Habibb , J.M. Lanlignel, B. Reedd, U. Rotics. Polynomial-time recognition of clique width ≤ 3 graphs, Discrete Applied MathematicsVolume 160, Issue 6, April 2012, pp 834-865.
6. D.G. Corneil, U. Rotics. On the relationship between clique-width and treewidth, SIAM Journal on Computing 34 (2005) 825_847.
7. E. Wanke. k-NLC graphs and polynomial algorithms, Discrete Applied Mathematics 54 (1994) 251_266.
8. I. Penev. On the clique width of (4k1,C4,C5,C7)-free graphs. Discrete Applied Mathematics, Vol. 285, (2020) pp. 688–690.
9. J. L. Gonzalez-Ruiz, J. R Marcial-Romero, J. Hernandez-Servın. Computing the clique width of cactus graphs. Electronic Notes in Theoretical Computer Science, Vol. 328, (2016)pp. 47–57.
10. J. L. Gonzalez-Ruiz, J. R. Marcial-Romero, J. A. Hernandez, G. De Ita. Computing the clique width of polygonal tree graphs, Advances in Soft Computing, Springer International Publishing, Cham, (2017) pp. 449–459.
11. J. Valdes, R. E. Tarjan, and E. L. Lawler. The Recognition of Series Parallel Digraphs, SIAM Journal on ComputingVol. 11, Iss. 2 (1982), 298-313.
12. M. A. López-Medina, J. Leonardo González-Ruiz, J. Raymundo Marcial-Romero, J. A. Hernández. Computing the Clique-Width on Series-Parallel Graphs, Computación y Sistemas, Vol 26, No 2 (2022)
13. M. Kurt, M. Berberler, and O. Ugurlu. A New Algorithm for Finding Vertex-Disjoint Paths. The International Arab Journal of Information Technology, Vol. 12, No. 6, November 2015.
14. M. R. Fellows, F. A. Rosamond, U. Rotics, S. Szeider. Clique-width is NP-complete, SIAM Journal on Discrete Mathematics, Vol. 23, No. 2, (2009)pp. 909–939.
15. S.i. Oum, P. Seymour. Approximating clique width and branch-width, Journal of Combinatorial Theory Series B 94 (2006) 514-528.
16. W. Espelage, F. Gurski, E. Wanke. How to solve NP-hard graph problems on clique width bounded graphs in polynomial time, Lecture Notes in Computer Science 2204 (2001) 117_128.