Dynamic Process Graphs and the Complexity of Scheduling*

Andreas Jakoby[1,2] Maciej Liśkiewicz[2] Rüdiger Reischuk[2]
[1]University of Toronto [2]Med. Universität zu Lübeck

Abstract

In parallel and distributed computing scheduling low level tasks on the available hardware is a fundamental problem. Traditionally, one has assumed that the set of tasks to be executed is known beforehand. Then the scheduling constraints are given by a *precedence graph.* Nodes represent the elementary tasks and edges the dependencies among tasks. This static approach is not appropriate in situations where the set of tasks is not known exactly in advance, for example, when different options how to continue a program may be granted.

In this paper a new model for parallel and distributed programs, the *dynamic process graph,* will be introduced, which represents all possible executions of a program in a compact way. The size of this representation is small – in many cases only logarithmically with respect to the size of any execution. An important feature of our model is that the encoded executions are directed acyclic graphs having a "regular" structure that is typical of parallel programs.

Dynamic process graphs embed constructors for parallel programs, synchronization mechanisms as well as conditional branches. With respect to such a compact representation we investigate the complexity of different aspects of the scheduling problem: the question whether a legal schedule exists at all and how to find an optimal schedule. Our analysis takes into account communication delays between processors exchanging data.

Precise characterization of the computational complexity of various variants of this compact scheduling problem will be given in this paper. The results range from easy, that is $\mathcal{NL}$-complete, to very hard, namely $\mathcal{NEXPTIME}$-complete.

Keywords: scheduling, precedence graph, communication delay, succint graph representation, program constructors

1 Introduction

Scheduling tasks efficiently is crucial for fast executions of parallel and distributed programs. An intensive study of this scheduling problem has led to the development of a number of algorithms that cover a wide spectrum of strategies: from fully static, where the compiler completely precomputes the schedule, i.e. when and where each task will be executed, to fully dynamic, where tasks are scheduled at run-time only. Changing from static to dynamic strategies one gets the potential of reducing the total execution time of a program because the resources are better used, but in general there will be more effort necessary at run-time. Therefore, existing parallel systems fix most details of the schedule already at compile time (see e.g. [8]). For a more dynamic and also fault tolerant approach see for example the MAFT project [13, 12].

In many cases, the set of tasks that have to be executed is not precisely known at compile time. El-Rewini and Ali have introduced a parallel program model that allows a suitable data representation for static scheduling algorithms [4]. The representation is based on two directed graphs: the *branch graph* and the *precedence graph.* This approach models conditional branchings quite well, but it is unsuited for parallel program constructors or synchronization mechanisms, for example the channel concept as implemented in the parallel programming language OCCAM.

*Supported by DFG Research Grant Re 672/2. An extended abstract of this paper has been presented at 16. Symposium on Theoretical Aspects of Computer Science STACS'99.

[1]Department of Computer Science, `email: jakoby@cs.toronto.edu`
[2]Institut für Theoretische Informatik, `email: jakoby/liskiewi/reischuk@informatik.mu-luebeck.de`

1.1 Extension to a Dynamic Environment

In this paper we introduce a new model, the *dynamic process graph*, **DPG**, which allows a natural representation for parallel and distributed programs. In particular, it gives a concise description of static scheduling problems for highly concurrent programs. An important feature of DPGs is that they resemble the characteristics of executing typical parallel or distributed programs, and this in a space-efficient way. This representation can provide an exponential compaction compared to the length of execution sequences.

Our main technical contribution is to analyze the complexity of finding optimal schedules with respect to this compact program representation. We will concentrate on scheduling elementary tasks where each task has unit execution time. No bound will be given on the number of processors available.

1.2 Communication Delays

Papadimitriou and Yannakakis have argued in [18] that scheduling policies should communication delays take into account occurring when one processor sends a piece of data to another one. Thus, it will be faster to schedule dependend tasks on the same processor. For further results concerning scheduling with communication delays in the standard static setting see [11, 10, 19]. Here, we will extend the complexity analysis to the dynamic case.

Communication delays will be specified by a function $\delta : E \to \mathbb{N}$, which defines the time necessary to send the data from one processor to another one. For simplification we assume that this delay is independent of the particular pair of processors (alternatively that $\delta(e)$ gives an upper bound on the maximal delay). Scheduling with communication delays requires the following condition to be fulfilled:

- if a task v is executed on processor p at time t then for each direct predecessor u of v holds: u has been finished either on p by time $t-1$, or on some other processor p' by time $t - 1 - \delta(u, v)$.

1.3 Dynamic Dependencies

In many systems, dependencies among particular task-instances are determined by a scheduling policy, not by the program itself. Consider the following situation. A program contains a part P where several processes concurrently generate data. The program can continue as soon as one of these results is available. Such a situation can compactly be described using the ALT-constructor of OCCAM [2]. In the piece of code given in the left part of Figure 1 one process sends data down channel C1, while a second one behaves similarly using channel C2.

```
P
ALT
  C1 ? X
     P1
  C2 ? X
     P2
```

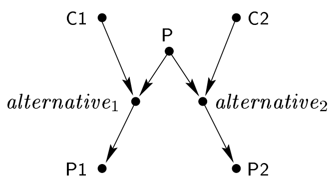

Figure 1: *A dynamic precedence graph representing* P*: the output mode of* P *is* ALT *and the input mode of* $alternative_1$ *and* $alternative_2$ *is* PAR.

A scheduling policy has to select one (and only one) of the alternatives Pi. Hence, either P1 or P2 will be a successor of P.

If both channels Ci get ready at the same time then a scheduler can choose arbitrarily. However, to minimize the total execution time it is helpful to know which process can be executed faster, P1 or P2? Even if the two channels do not get ready simultaneously executing a ready alternative immediately, may overall lead to a longer schedule than waiting until the other channel is ready.

We will consider different degrees of concurrency expressed by the ALT and the PAR constructor to create parallel processes, and analyze the complexity of the corresponding scheduling problems with respect to the amount of concurrency.

To represent such parallel programs in a natural way, we introduce dynamic process graphs, which are generalizations of standard precedence graphs. A dynamic process graph is an directed acyclic graph

$G = (V, E)$ with two sets of labels $I(v), O(v) \in \{\mathsf{PAR}, \mathsf{ALT}\}$ attached to the nodes $v \in V$. Nodes represent tasks, edges dependencies between tasks. A complete formal definition will be given in the next section.

The label $I(v)$ describes the *input mode* of task v. If $I(v) = \mathsf{ALT}$ then to execute v one of the predecessor tasks u with $(u, v) \in E$ has to be completed. $I(v) = \mathsf{PAR}$ requires that executions of all predecessors of v have to be completed before v can start. If task v has been completed then according to the *output mode* $O(v)$ one of v's successors in case $O(v) = \mathsf{ALT}$ (resp. all of them in case $O(v) = \mathsf{PAR}$) has to be initiated. Figure 1 gives an example of such a representation.

Dynamic process graphs are a compact way to illustrate data dependencies of parallel programs. In particular, it will be easy for programs written in a parallel programming language like OCCAM or Ada. Note that a standard precedence graph cannot represent such programs in a simple way. We should note that dynamic process graphs can also be modeled by a certain class of Petri nets and their reachability problem. However, this class does not seem to correspond to those subclasses that have been considered in more detail in the literature. We will therefore stick to the DPG model.

1.4 New results

The scheduling problem for dynamic process graphs is a natural generalization of the static scheduling problem. In particular, the delay scheduling for a precedence graph G is equivalent to the scheduling problem for the dynamic process graph (G, I, O) with $I, O \equiv \mathsf{PAR}$. In the static case, the scheduling problem with communication delays is already computationally difficult. In [18] it has been shown that this problem is $\mathcal{NP}$-complete even if for each graph the communication delay takes only a single value, but this value has to increase with the size of the graph. We have improved this result in [10] showing that the problem remains $\mathcal{NP}$-complete even if we restrict the class of precedence graphs to (1,2)-trees. On the other hand, in [11] it has been shown that for fixed delay $\delta \equiv c$ independent of the precedence graphs the problem can be solved in polynomial time where the degree of the polynomial grows with c.

Due to the compact representation in our dynamic model it is no longer obvious that the dynamic scheduling problem can be solved in $\mathcal{NP}$ at all. In fact, one of our main results is that even restricted to constant communication delay the scheduling problem for dynamic process graphs is $\mathcal{NEXPTIME}$-complete. To prove this we construct a reduction of the SUCCINCT-3SAT problem. However, if we restrict the input mode I to ALT then the problem becomes $\mathcal{P}$-complete. Even more, also fixing the output mode the problem becomes $\mathcal{NL}$-complete. A similar complexity jump has been observed for classical graph problems in [5, 17, 14].

There it is shown that simple graph properties become $\mathcal{NP}$-complete when the graph is represented in a particular succinct way using generating circuits or a hierarchical decomposition. Under the same representation graph properties that are ordinarily $\mathcal{NP}$-complete, like HAMILTON CYCLE, 3-COLORABILITY, CLIQUE (of size $|V|/2$), etc., become $\mathcal{NEXPTIME}$-complete.

On the other hand, some restricted variants of this scheduling problem, which may seem to be easy at first glance, remain hard, namely $\mathcal{NP}$-complete. Figure 2 summarizes our results about the complexity of the dynamic delay scheduling problem with respect to the input and output modes that may occur in the graphs. Below the arbitrary mode means that three cases can occur: either the mode is restricted only to ALT or only to PAR or the mode is unrestricted, i.e. ALT and PAR may occur in the graphs.

input mode	output mode	complexity
ALT	ALT	$\mathcal{NL}$-complete
ALT	PAR	$\mathcal{NL}$-complete
ALT	ALT, PAR	$\mathcal{P}$-complete
PAR	arbitrary	$\mathcal{NP}$-complete
ALT, PAR	ALT	$\mathcal{NP}$-complete
ALT, PAR	PAR	$\mathcal{BH}_2$-hard
unrestricted		$\mathcal{NEXPTIME}$-complete

Figure 2: *The complexity of dynamic scheduling with communication delays with respect to input and output modes.*

We will also consider the question whether for a given dynamic process graph there exists a run for a program represended by the graph at all. It will be shown that the problem is $\mathcal{NP}$-complete even if the input mode is fixed to PAR and the output mode to ALT.

The remaining part of this paper is organized as follows. In Section 2 we give some examples and a formal definition of DPGs and their scheduling problem. Section 3 studies the complexity of the existence problem. The last section deals with the problem to find optimal schedules.

2 Dynamic Process Graphs and Runs

For illustration, consider the following parallel program P written in OCCAM (Fig. 3). This program contains branches, but the situation is not as bad as it could be since the branching does not depend on the current values of variables. Still there is the problem to determine the set of tasks that have to be executed at run time. It will turn out that even for such restricted programs the scheduling problem is quite hard. Depending on the ALT branches chosen the execution of this program is represented by one of the four possible runs shown in Fig. 3. The following definition tries to capture this dichotomy between parallel/distributed programs and their executions.

```
SEQ
 ALT
   in1 ?  x
     P1
   in2 ?  x
     P2
 PAR
    P3
    P4
 ALT
   in1 ?  y
     P5
   in2 ?  y
     P6
```

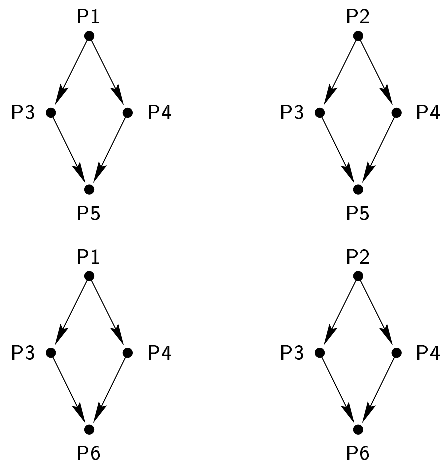

Figure 3: *OCCAM program* P *and its possible runs.*

Definition 1 *A* **dynamic process graph** $\mathcal{G} = (V, E, I, O)$ *is a directed acyclic graph (DAG)* $G = (V, E)$*, with node labellings* $I, O : V \to \{\mathsf{ALT}, \mathsf{PAR}\}$*.* $V = \{v_1, v_2, \ldots, v_n\}$ *represents a set of* **processes** *and* E *dependencies among them.* I *and* O *describe* **input modes,** *(resp.* **output modes***) of the processes.*

A finite DAG $H_{\mathcal{G}} = (W, F)$ *is a* **run** *of* $\mathcal{G}$ *iff the following conditions are fulfilled:*

1. *The set* W *is partitioned into subsets* $W(v_1) \mathbin{\dot\cup} W(v_2) \mathbin{\dot\cup} \ldots \mathbin{\dot\cup} W(v_n)$*. The nodes in* $W(v_i)$ *are* **execution instances** *of the process* v_i*, and will be called* **tasks***.*

2. *Each source node of* $\mathcal{G}$*, which represents a starting operation of the program modelled by* $\mathcal{G}$*, has exactly one execution instance in* $H_{\mathcal{G}}$*.*

3. *For a process* $v \in V$ *let pred*$(v) := \{u_1, u_2, \ldots, u_p\}$ *denote the set of all predecessors of* v *and succ*$(v) := \{w_1, w_2, \ldots, w_r\}$ *its successors. For any execution instance* x *of* v *in* $W(v)$ *it has to hold*

   - *if* $I(v) = \mathsf{ALT}$ *then* x *has a unique predecessor* y *belonging to* $W(u_i)$ *for some* $i \in \{1, \ldots, p\}$*;*
   - *if* $I(v) = \mathsf{PAR}$ *then pred*$(x) = \{y_1, y_2, \ldots, y_p\}$ *with* $y_i \in W(u_i)$ *for each* $i \in \{1, \ldots, p\}$*;*
   - *if* $O(v) = \mathsf{ALT}$ *then* x *has a unique successor* z *belonging to* $W(w_j)$ *for some* $j \in \{1, \ldots, r\}$*;*
   - *if* $O(v) = \mathsf{PAR}$ *then succ*$(x) = \{z_1, z_2, \ldots, z_r\}$ *with* $z_j \in W(w_j)$ *for each* $j \in \{1, \ldots, r\}$*.*

For a DPG $\mathcal{G}$ with run $H_{\mathcal{G}} = (W, F)$ we extend the definition of W as follows. For $A \subseteq V$ let $W(A) := \bigcup_{v \in A} W(v)$ and for $(v, v') \in E$ let $W(v, v') := \{(y, z) \in F \mid y \in W(v)$ and $z \in W(v')\}$.

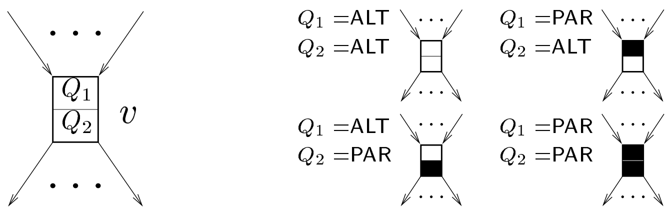

Figure 4: *A node v with input label Q_1 and output label Q_2 and the schematic representation.*

Fig. 4 shows a node v of a DPG with input mode Q_1 and output mode Q_2. Through the paper we will illustrate the ALT-mode by a white box, the PAR-mode by a black box. For a source or a node with indegree 1 the input mode is obviously inessential. Hence we will ignore such a label, and similarly the output label in case of a sink or a node with outdegree 1. A dynamic process graph corresponding to the program of Fig. 3 is shown in Fig. 5 (a).

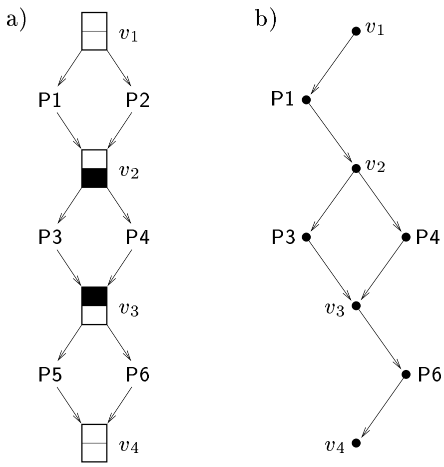

Figure 5: *A dynamic process graph for program* P *(a) and a run of this dynamic process graph (b).*

Observe that a run can be smaller than its defining dynamic process graph, e.g. the graph in Fig. 5 (a) has 10 nodes while its run (b) only 8 since certain tasks are not executed at all. More typically, however, a run will be larger than the dynamic process graph itself since the PAR-constructor allows process duplications. If, for example, the output mode of vertex v_1 in Fig. 5 is changed from ALT into PAR then both processes P_1 and P_2 have to be executed. Hence, process v_2 with input mode ALT has to be duplicated in order to consume both processes. This in turn implies that P_3 and P_4 will have 2 execution instances each. Therefore, each run of the modified graph consists of 15 nodes. The following lemma gives an upper bound on this blow-up, resp. the possible compaction ratio of dynamic process graphs.

Lemma 1 *Let $\mathcal{G} = (V, E, I, O)$ be a dynamic process graph and $H_{\mathcal{G}} = (W, F)$ be a corresponding run. Then it holds $|W| \leq 2^{|V|-1}$. Moreover, this general bound is tight.*

Proof. Call a node v of $G = (V, E)$ to be in level ℓ if the length of the longest path between a source and v is $\ell - 1$, and let r_ℓ denote the number of nodes on level ℓ. Let $H_{\mathcal{G}}$ be any run of $\mathcal{G}$ and let $R(\ell)$ denote the total number of execution instances in $H_{\mathcal{G}}$ for processes $v \in G$ in levels with index at most ℓ. We claim that

$$R(\ell) \leq (r_\ell + 1)(r_{\ell-1} + 1) \ldots (r_2 + 1) r_1 \ . \tag{1}$$

For $\ell = 1$ it holds $R(1) = r_1$ since according to the definition of a run each source of G has exactly one execution instance in $H_{\mathcal{G}}$. Moreover, if we assume that the inequality (1) holds for $\ell < i$ then $R(i)$ can be bounded by

$$R(i) \;\leq\; R(i-1) \;+\; \sum_{v \text{ of level } i} \sum_{u \in \mathrm{pred}(v)} |W(u)| \;\leq\; R(i-1) + r_i \cdot R(i-1)\ ,$$

since each node v in level i has its predecessors in levels $i-1, \ldots, 2, 1$ (in the worst case, all nodes from this levels can be predecessors of v). Hence, the total number of execution instances of v does not exceed $R(i-1)$.

It is easy to check that among all sequences of positive integers $r_1, r_2, \ldots, r_\ell$, fulfiling the condition $r_1 + r_2 + \ldots + r_\ell = |V|$ the value $(r_\ell + 1)(r_{\ell-1} + 1) \ldots (r_2 + 1) r_1$ in inequality (1) is maximal when all numbers r_i are equal to 1, which gives $|W| \leq 2^{|V|-1}$.

To see that the bound is tight consider the complete DAG on n nodes $v_1, \ldots, v_n$ with edges (v_i, v_j) for all $1 \leq i < j \leq n$. All input labels are ALT, while all output labels are PAR. It is easy to see that this dynamic process graph has only one run, and this has size 2^{n-1}. ∎

Hence, there are dynamic process graphs where processes have exponential many execution instances. Note that a similar effect occurs by using the replicated PAR- and ALT-constructor of OCCAM, which allows an exponential blow up of the number of active tasks.

Lemma 2 *Any dynamic process graph $\mathcal{G}$ has at most double exponential many different runs and this bound can actually occur.*

Proof: To give the upper bound we consider the maximal length of a run (Lemma 1). Let $\mathcal{G} = (V, E, I, O)$ be a dynamic process graph and $H_{\mathcal{G}} = (W, F)$ be a corresponding run. Then it holds $|W| \leq 2^{|V|-1}$. Furthermore, we can bound the degree of $H_{\mathcal{G}}$ by $|V|$. Note that each graph with n nodes with maximum degree d can be represented by an binary string of length $O(d \cdot n \cdot \log n)$. Since there are at most $2^{O(d \cdot n \cdot \log n)}$ binary strings of such length $\mathcal{G}$ has at most $2^{O(|V|^2 2^{|V|})} = 2^{2^{|V|+O(\log|V|)}}$ different runs.

For the lower bound we consider the following DPG with vertex set $V = \{v_1, \ldots, v_{n-3}, u_1, u_2, v\}$, where

- the nodes $v_1, \ldots, v_{n-3}, u_1, u_2$ have input mode ALT and output mode PAR and
- v has input mode PAR and output mode ALT.

We connect the nodes by

$$E \quad := \quad \{\ (v_i, v_j) \mid 1 \leq i < j \leq n-3\} \;\; \cup \;\; \{\ (v_{n-3}, u_1), (v_{n-3}, u_2), (u_1, v), (u_2, v)\}\ .$$

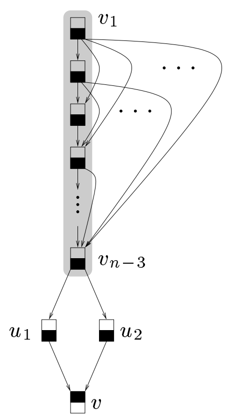

Figure 6: *A dynamic process graph with double exponential different runs.*

It follows from the Proof of Lemma 1 that for any run $H_{\mathcal{G}} = (W, F)$ holds:

$$|W(v_{n-3})| = 2^{n-4} \qquad \text{and therefore} \qquad |W(u_1)| = |W(u_2)| = |W(v)| = 2^{n-4}\ .$$

Note that there are $2^{n-4}!$ different ways to compute the 2^{n-4} execution instances of v. Hence, for a constant c there are $2^{cn2^n} = 2^{2^{n+\Omega(\log n)}}$ different runs for $\mathcal{G}$. ∎

Definition 2 *Let $\mathcal{G} = (V, E, I, O)$ be a dynamic process graph with communication delay $\delta : E \to I\!N$ between its processes. A schedule for $\mathcal{G}, \delta$ is a schedule of a run $H = (W, F)$, where the communication delay between each execution instance $x \in W(u)$ and $y \in W(v)$ is given by $\delta(u, v)$.*

If S is a schedule of $\mathcal{G}, \delta$ then let $T(S)$ denote the duration of S, i.e. the amount of time necessary to complete all tasks in S. Define $\boldsymbol{T_{opt}(\mathcal{G}, \delta)} := \min_{S \text{ for } \mathcal{G},\delta} T(S)$.

This leads to the following decision problem:

Definition 3 DYNAMIC PROCESS GRAPH SCHEDULE (DPGS)
Given a DPG $\mathcal{G} = (V, E, I, O)$, a communication delay δ, and a deadline T^*, does $T_{opt}(\mathcal{G}, \delta) \leq T^*$ hold?
Let DPGS[IN | OUT] *denote the DPGS Problem for the class of DPGs with input and output modes restricted to* IN, *resp.* OUT, *where* IN, OUT $\subseteq \{\mathsf{ALT}, \mathsf{PAR}\}$.

For short, we will write "$\cdot$" if the mode is $\{\mathsf{ALT}, \mathsf{PAR}\}$. Hence, e.g. DPGS[$\cdot$ | ALT] will denote DPGS Problem for DPGs with unrestricted input mode and output mode restricted to ALT.

3 The Execution Problem

3.1 General Properties

The number of different runs of a DPG can be huge according to Lemma 2. On the other hand, it is not obvious that for any DPG an appropriate run exists at all. It is easy to see that dynamic process graphs with either only PAR labels, or with all input labels equal to ALT can always be executed. The first case corresponds to standard static precedence graphs. However, this is no longer true for arbitrary DPGs. For a simple example of a graph which has no run see Figure 7. In this case the input mode of all nodes is PAR and their output mode ALT. This section studies the problem whether a given dynamic process graph has a legal run. Obviously, a schedule for a given DPG can only be constructed if an appropriate run exists.

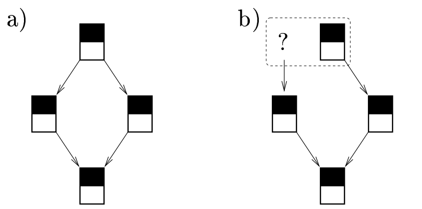

Figure 7: *Example of a dynamic process graphs with $O \equiv$ ALT and $I \equiv$ PAR that has no run.*

Definition 4 Execution Problem for DPGs (ExDPG): *Given a DPG $\mathcal{G}$, decide whether it can be executed, that means whether it has run.*
ExDPG[IN | OUT] *denotes the ExDPG Problem for the class of DPGs with input and output modes restricted to* IN, *resp.* OUT, *where* IN, OUT $\subseteq \{\mathsf{ALT}, \mathsf{PAR}\}$.

Similarly as for DPGS Problem we will write "$\cdot$" if the mode is $\{\mathsf{ALT}, \mathsf{PAR}\}$.

As we have seen above for some restricted DPGs this question has a trivial answer. In general, a decision procedure may be complex. Of course, if a DPG has a run then it has also a schedule, and since we have a bound on the size of the run, one can also compute an upper bound on the maximal schedule length given the maximal communication delay. Thus, the execution problem for DPGs could be solved by a reduction to the scheduling problem with a huge enough deadline. However, we want to capture the complexity of the execution problem more precisely and thus will investigate it directly.

In this section we show that for graphs with arbitrary input and output modes the execution problem is $\mathcal{NP}$-complete. Then we will prove that the problem becames feasible for graphs with the output labels restricted to PAR but it remains $\mathcal{NP}$-complete even if one restricts the input mode to PAR and the output modes to ALT. We will investigate the feasible case first. Figure 8 below summarizes our results about the complexity of ExDPG Problem with respect to the input and output modes that may occur in the graphs, where arbitrary means that three cases can occur: either the mode is restricted only to ALT or only to PAR or the mode is unrestricted, i.e. ALT and PAR may occur in the graphs.

input mode	output mode	complexity
ALT	arbitrary	trivial
PAR	PAR	trivial
ALT, PAR	PAR	$C_=\mathcal{L}$-complete
PAR	ALT	$\mathcal{NP}$-complete
ALT, PAR	ALT	$\mathcal{NP}$-complete
unrestricted		$\mathcal{NP}$-complete

Figure 8: *The complexity of Execution Problem for DPGs with respect to input and output modes.*

Definition 5 *Assume that $H_\mathcal{G} = (W, F)$ is a run of a DPG $\mathcal{G}$ with processes $V = \{v_1, \dots, v_n\}$. The sequence $(|W(v_1)|, |W(v_2)|, \dots, |W(v_n)|)$ will be called the* **characteristic vector** *of $H_\mathcal{G}$.*

Lemma 3 *If $\mathcal{G}$ is a DPG with output mode restricted to* PAR *then all runs have exactly the same characteristic vector.*

Proof: Assume that $\mathcal{G}$ can be executed, that means it has at least one run. Then the restriction on the output mode implies that each process of $\mathcal{G}$ has to be executed at lest once because it is either a source or a successor of a process with output mode PAR.

The lemma follows by induction on the depth of $\mathcal{G}$. Assume first that the claim holds for the subgraph $\mathcal{G}'$ of $\mathcal{G}$ obtained by removing all nodes of maximal depth. By definition, for all runs $H_\mathcal{G} = (W, F)$ and for all nodes v of maximal depth in $\mathcal{G}$ it holds: if u is a predecessor of v and $t_u \in W(u)$ is an execution instance of u then there exists a unique task $t_v \in W(v)$ such that $(t_u, t_v) \in F$. If $I(v) = \mathsf{ALT}$ then the in-degree of t_v is 1, thus there are as many execution instances of v as executions of nodes $u \in \mathrm{pred}(v)$. Since, by inductive assumption, the size of the $W(u)$ does not depend on the particular run the same holds for the size of $W(v)$, too.

On the other hand, if $I(v) = \mathsf{PAR}$ then the in-degree of each execution instance of v is equal to the number of predecessors of v. Hence, in this case there can only be a run for $\mathcal{G}$ if each predecessor u of v has the same number of execution instances. ∎

3.2 The Complexity of Restricted Versions

The proof of Lemma 3 suggests the following procedure which computes the characteristic vector of a DPG $\mathcal{G}$ if the graph can be executed at all; otherwise the procedure returns `false`. Therefore, it solves problem ExDPG[$\cdot$ | PAR].

`procedure` *characteristic-vector* $(\mathcal{G})$
`compute a topological ordering` $(v_1, v_2, \dots, v_n)$ `of` $\mathcal{G}$
`for` $i := 1$ `to` n `do`
  `if` v_i `is a source then` $A(v_i) := 1$
  `elseif` $I(v_i) = \mathsf{ALT}$ `then` $A(v_i) := \sum_{v_j \in \mathrm{pred}(v_i)} A(v_j)$
  `elseif` $A(u) = A(w)$ `for every` $u, w \in \mathrm{pred}(v_i)$ `then` $A(v_i) := A(u)$
  `else return false and halt`
`return` A

Fact 1 *For a given dynamic process graph (V, E, I, O) the problem* ExDPG[$\cdot$ | PAR] *can be solved in time $O(|V| + |E|)$ assuming uniform time measure.*

Hence we obtain: ExDPG[$\cdot$ | PAR] $\in \mathcal{P}$. Further we will show that the problem can be solved efficiently in parallel, too, namely we will construct an $\mathcal{NC}^2$ algorithm for ExDPG[$\cdot$ | PAR] what implies the inclusion above. The algorithm works in two stages. In the first stage, a given DPG $\mathcal{G}$ is reduced to an instance $\mathcal{G}^*$ of a ExDPG[ALT, 1-PAR | PAR] problem, where the prefix 1 denotes that we consider the common Execution Problem ExDPG[$\cdot$ | PAR] restricted to such graphs where nodes with PAR input mode occur only as sinks. In the second stage, decide whether $\mathcal{G}^*$ can be executed. The lemma below shows that the reduction of the first stage can be done efficiently.

Lemma 4 ExDPG[$\cdot$ | PAR] $\leq_{\log}$ ExDPG[ALT, 1-PAR | PAR].

Proof: Let $\mathcal{G} = (V, E, I, O)$ be a given input of ExDPG[$\cdot$ | PAR]-problem. Denote by V_o the nodes of V which cause that $\mathcal{G}$ is not an instace of ExDPG[ALT, 1-PAR | PAR], i.e. let V_o be the set of all non-sink nodes $v \in V$ with $I(v) = \mathsf{PAR}$. Furthermore, let $\mathrm{pred}(v) := (u_1, \ldots, u_d)$ be the ordered sequence of predecessors of $v \in V$ and let $\mathrm{pred}(v, i) := u_i$.

We construct $\mathcal{G}^* = (V^*, E^*, I^*, O^*)$ as follows. Let $I^*(v) = I(v)$ for any $v \in V \setminus V_o$ and $I^*(v) = \mathsf{ALT}$ for $v \in V_o$. Moreover let E^* contain at the beginning the edges: $E \setminus \{(\mathrm{pred}(v, i), v) \mid v \in V_o \text{ and } i > 1\}$.

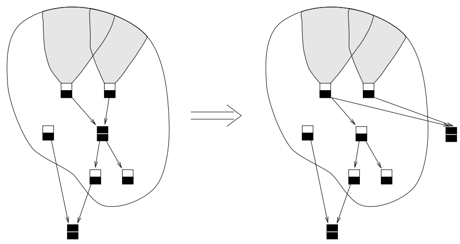

Figure 9: *Construction of an instance of the* ExDPG[ALT, 1-PAR | PAR]*-problem.*

Next, for any node $v \in V_o$ we add one additional node v' to $\mathcal{G}^*$. Define for these new nodes $I^*(v') = O^*(v') = \mathsf{PAR}$ and additionally we add (u, v') to E^* if and only if $(u, v) \in E$. Note that there exists a run $H_{\mathcal{G}^*}$ for $\mathcal{G}^*$ iff for any sink-node $v \in V^*$ and any pair u_1, u_2 of predecessors of v the equality $|W(u_1)| = |W(u_2)|$ holds. Hence, there exists a run for $\mathcal{G}^*$ if and only if there exists a run for $\mathcal{G}$. The construction of $\mathcal{G}^*$ is illustrated in Figure 9. ∎

To complete the algorithm it is sufficient to give an $\mathcal{NC}^2$ procedure which solves an appropriate ExDPG-problem. Below we will show even more. Namely, we describe how to compute in $\mathcal{NC}^2$ the characteristic vector $\mathcal{C}$ for a given instance graph of ExDPG[ALT, 1-PAR | PAR] if such a vector exists. To present the computation more intuitively we will express the executions for nodes of $\mathcal{G}$ in terms of paths connecting $\mathcal{G}$'s vertices.

Definition 6 *Let $G = (V, E)$ be a directed acyclic graph and let $u, v \in V$. Then define* $\#\mathbf{Path}[\boldsymbol{u} \to \boldsymbol{v}]$ *as the number of different paths from node u to v.*

Below we give a relationship between the number of paths and executions. The result can be shown easily by induction hence we omit a proof here.

Lemma 5 *Let $\mathcal{G}$ be an instance of the* ExDPG[ALT, 1-PAR | PAR]*-problem. Then for any run $H_{\mathcal{G}} = (F, W)$ of $\mathcal{G}$ and for all non-sink nodes v it holds that*

$$|W(v)| = \sum_{q \text{ - source of } \mathcal{G}} \#\mathrm{Path}[q \to v] \ .$$

Assuming that $q_1, \ldots, q_k$ are the sources of a given graph $\mathcal{G}$, a procedure which computes the characteristic vector $\mathcal{C}$ of $\mathcal{G}$ have the following shape

`for any non-sink node` v `let` $\mathcal{C}(v) := \sum_{t=1}^{k} \#\mathrm{Path}[q_t \to v]$;
`for any sink node` v `with` $I(v) = \mathsf{PAR}$ `do`
    `if` $\mathcal{C}(u) = \mathcal{C}(w)$ `for any` $u, w \in \mathrm{pred}(v)$ `then` $\mathcal{C}(v) := \mathcal{C}(u)$
    `else return false.`

Obviously, step 2 can be done in a straightforward way. To compute the number of paths in step 1, we use a simple arithmetic $\mathcal{NC}^2$ algorithm described in [15]. Let us recall it below.

We say that a vertex v is at layer i in $\mathcal{G} = (V, E)$ if the longest path from a source node to v is of length i. Hence, e.g. any source node of $\mathcal{G}$ is at layer 0. Let (v, i) denote a vertex v at layer i.

`in parallel for each` u, v, i `with` $u, v \in V$, `and` $0 \le i < n-1$ `do`
    `if` $(u, v) \in E$ `and` u `is at layer` i `and` v `is at layer` $i+1$
        `then` $\#\text{Path}[(u,i) \to (v,i+1)] := 1$ `else` $\#\text{Path}[(u,i) \to (v,i+1)] := 0$;
`for` $k := 1$ `to` $\log n$ `do`
    `in parallel for each` u, v, i, ℓ `where` $2^{k-1} < \ell \le 2^k$ `and` $i + \ell < n$ `do`
        $\#\text{Path}[(u,i) \to (v,i+\ell)] :=$
            $\sum_w \#\text{Path}[(u,i) \to (w,i+\ell/2)] \cdot \#\text{Path}[(w,i+\ell/2) \to (v,i+\ell)]$;

Hence, we conclude

Theorem 1 *The characteristic vector of a DPG with $O \equiv \mathsf{PAR}$ can be computed in $\mathcal{NC}^2$.*

Furthermore, we can prove that ExDPG[$\cdot$ | PAR] is complete for $\mathrm{C}_=\mathcal{L}$.

For a nondeterministic Turing machine M and an input string x define $\#\mathrm{acc}_M(x)$ as the number of accepting computations of M on input x. $\#\mathcal{L}$ denotes the class of functions f with $f \equiv \#\mathrm{acc}_M$ for a nondeterministic logarithmic space bounded machine M. Let Gap$\mathcal{L}$ be the class of functions that are the difference of two $\#\mathcal{L}$-functions and

$$\mathrm{C}_=\mathcal{L} \ := \ \{ \ A \mid \exists f \in \mathrm{Gap}\mathcal{L} \ \forall x \ [\ x \in A \ \Leftrightarrow \ \ f(x) = 0 \ ] \ \} \ .$$

A canonical complete problem for $\mathrm{C}_=\mathcal{L}$ is the ENoP-problem [1]:

Definition 7 Equal Number of Paths Problem [ENoP]
Given two DAGs $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$ and four nodes $a_1, b_1 \in V_1$ and $a_2, b_2 \in V_2$. Decide, whether the number of path from a_1 to b_1 in G_1 is equal to the number of path from a_2 to b_2 in G_2.

In the following we will consider the following variant of the ENoP-problem:

> Given two DAGs $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$ and two sinks $b_1 \in V_1$ and $b_2 \in V_2$. Decide, whether the total number of paths from a source of G_1 to b_1 in G_1 is equal to the total number of paths from a source of G_2 to b_2 in G_2. We will denote this problem by ENoP*.

Lemma 6 ENoP $\le_{\log}$ ENoP* $\le_{\log}$ ENoP.

Proof: Let $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$, $a_1, b_1 \in V_1$ and $a_2, b_2 \in V_2$ be an instance of the ENoP-probelm. We construct an instance of the ENoP*-probelm $\langle (G_1', x_1), (G_2', x_2) \rangle$ as follows: each graph G_i' consist of two copies of each of the graphs G_1, G_2.

Let us first consider the graph G_1' (see Figure 10). To distinguish between the copies of G_1 and G_2 we will index the copies of G_i by $G_{i,1}, G_{i,2}$ and the copies of a_i, b_i by $a_{i,1}, a_{i,2}, b_{i,1}, b_{i,2}$. Furthermore, we add 8 new nodes to each graph G_1': $a_1', a_2', b_1', b_2', c_1, c_2, d_1, d_2$ and draw edges from a_i' to all sources of $G_{i,1}, G_{i,2}$ and from the sinks of $G_{i,1}$ and $G_{i,2}$ to b_i'. Finally we add the edges

$$(a_1', a_2'), (b_2', b_1'), (c_1, a_{1,1}), (b_{1,1}, d_1), (c_2, a_{2,1}), (b_{2,2}, d_2), (a_1', c_1), (d_1, b_1'), (a_2', c_2), (d_2, b_2').$$

Let $x_1 := b_1'$. Note that the number of paths from the unique source a_1' of G_1' to b_1' is:

$$\begin{aligned}\#\text{Path}[a_1' \to b_1'] \ = \ & 2 \cdot \sum_{u \in \mathrm{source}(G_1),\ v \in \mathrm{sink}(G_1)} \#\text{Path}[u \to v] \ + \ 2 \cdot \sum_{u \in \mathrm{source}(G_2),\ v \in \mathrm{sink}(G_2)} \#\text{Path}[u \to v] \\ & + \sum_{u \in \mathrm{source}(G_1)} \#\text{Path}[u \to b_1] \ + \sum_{v \in \mathrm{sink}(G_1)} \#\text{Path}[a_1 \to v] \\ & + \sum_{u \in \mathrm{source}(G_2)} \#\text{Path}[u \to b_2] \ + \sum_{v \in \mathrm{sink}(G_2)} \#\text{Path}[a_2 \to v] \ + \ \#\text{Path}[a_1 \to b_1] \ .\end{aligned}$$

The construction of G'_2 is similar to the construction of G'_1. We change the last 8 edges as follows: $(a'_2, a'_1), (b'_1, b'_2), (c_1, a_{1,1}), (b_{1,2}, d_1), (c_2, a_{2,1}), (b_{2,1}, d_2), (a'_1, c_1), (d_1, b'_1), (a'_2, c_2), (d_2, b'_2)$. Finally, let $x_2 := b'_2$. Note that the number of paths from the unique source a'_2 of G'_2 to b'_2 is:

$$\begin{aligned}\#\mathrm{Path}[a'_2 \to b'_2] \;=\; & 2 \cdot \sum_{u \in \mathrm{source}(G_1),\ v \in \mathrm{sink}(G_1)} \#\mathrm{Path}[u \to v] \;+\; 2 \cdot \sum_{u \in \mathrm{source}(G_2),\ v \in \mathrm{sink}(G_2)} \#\mathrm{Path}[u \to v] \\ & + \sum_{u \in \mathrm{source}(G_1)} \#\mathrm{Path}[u \to b_1] \;+\; \sum_{v \in \mathrm{sink}(G_1)} \#\mathrm{Path}[a_1 \to v] \\ & + \sum_{u \in \mathrm{source}(G_2)} \#\mathrm{Path}[u \to b_2] \;+\; \sum_{v \in \mathrm{sink}(G_2)} \#\mathrm{Path}[a_2 \to v] \;+\; \#\mathrm{Path}[a_2 \to b_2]\,.\end{aligned}$$

Hence, $\langle (G'_1, x_1), (G'_2, x_2) \rangle \in$ ENoP* iff $\langle (G_1, a_1, b_1), (G_2, a_2, b_2) \rangle \in$ ENoP. Therefore it follows that ENoP $\leq_{\log}$ ENoP*.

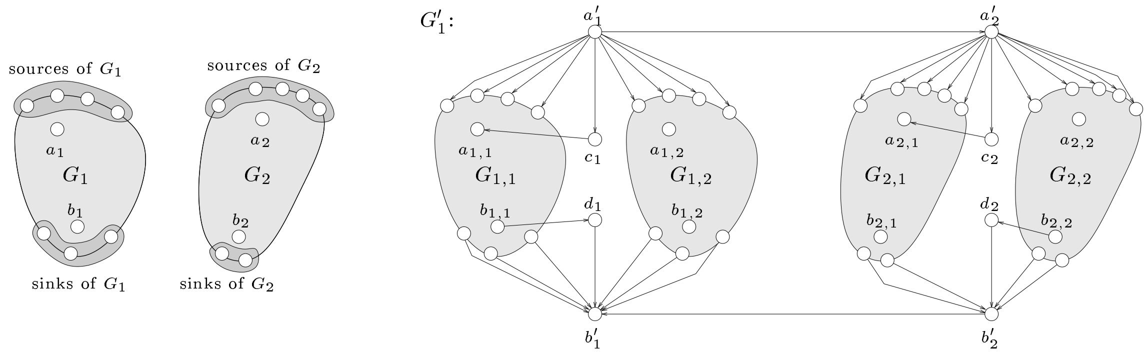

Figure 10: *Construction of an instance of the ENoP*-problem.*

To show that ENoP* $\leq_{\log}$ ENoP we simply add a new source u_i to each of the graphs G_i, $i \in \{1, 2\}$, and connect these node to the sources of G_i. Hence, u_i is the unique source of G_i. The claim follows directly. ∎

Hence, ENoP* is $\mathrm{C}_=\mathcal{L}$-complete. To prove that ExDPG[$\cdot$ | PAR] is $\mathrm{C}_=\mathcal{L}$-complete we will show that ENoP* is log-space reducible ExDPG[$\cdot$ | PAR] and vice versa.

Lemma 7 ENoP* $\leq_{\log}$ ExDPG[$\cdot$ | PAR].

Proof: Let $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$, $b_1 \in V_1$ and $b_2 \in V_2$ be an instance of the ENoP*-problem. Then we generate the DPG $\mathcal{G} = (V, E, I, O)$ with $V := V_1 \cup V_2 \cup \{u\}$ for $u \notin V_1 \cup V_2$, $E := E_1 \cup E_2 \cup \{(b_1, u), (b_2, u)\}$, $O \equiv$ PAR, $I(v) =$ ALT for $v \neq u$, and $I(u) =$ PAR. From Lemma 5 we can conclude, that there exists a run for $\mathcal{G}$ iff the number of paths to b_1 equals the number of paths to b_2. The claim follows directly. ∎

Since ExDPG[$\cdot$ | PAR] is log-space reducible to ExDPG[ALT, 1-PAR | PAR] hence to complete the proof that ExDPG[$\cdot$ | PAR] is $\mathrm{C}_=\mathcal{L}$-complete it is sufficient to show the following fact.

Lemma 8 ExDPG[ALT, 1-PAR | PAR] $\leq_{\log}$ ENoP*.

Proof: For the reduction we will use so called chain graphs D_t which are defined as follows: A **chain graph $\boldsymbol{D_t}$** of length t for $t \geq 0$, consists of $3 \cdot t + 1$ vertices $p_0, p_1, \ldots, p_t, l_1, \ldots, l_n, r_1, \ldots, r_t$, and edges $(p_i, l_i), (p_i, r_i), (l_i, p_{i-1}), (r_i, p_{i-1})$ for $i = t, \ldots, 1$. Note, that for $n = 0$, the chain contains just one node. A recursive construction of the **chain DPG $\boldsymbol{\mathcal{D}_t}$** is shown in figure 11.

Given a value t a chain D_t resp. $\mathcal{D}_t$ can be generated in space $O(\log(t))$.

Let us now consider an instance $\mathcal{G} := (V, E, I, O)$ of the ExDPG[ALT, 1-PAR | PAR]-problem. Let $\boldsymbol{V_{sink}}$ denotes the set of sinks with input mode PAR, $\boldsymbol{E_{sink}}$ be the subset of edges $E \cap (V \times V_{sink})$ from V to V_{sink} in $\mathcal{G}$ and $\boldsymbol{V_{source}}$ the set of sources of $\mathcal{G}$. Note that all these sets can be enumerated in logarithmic space in the size of $\mathcal{G}$. Let $\boldsymbol{\phi_{sink}}$: $\{1, \ldots, |E_{sink}|\} \to E_{sink}$ be such a function that enumerates E_{sink}. Then define for $i \leq |E_{sink}|$

$$\boldsymbol{\sigma(i)} \;:=\; \mathrm{start}(\phi_{sink}(\min\{\, j \mid \mathrm{end}(\phi_{sink}(j)) = \mathrm{end}(\phi_{sink}(i))\}))$$

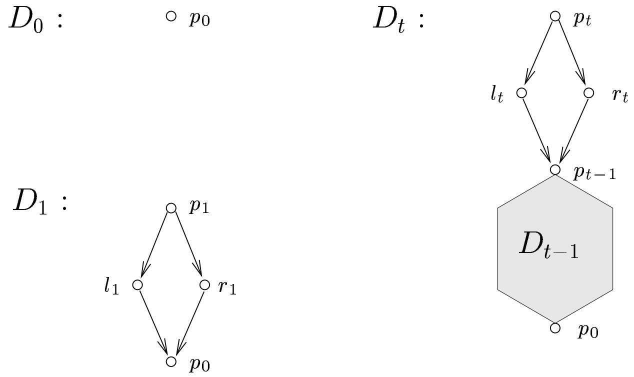

Figure 11: *The chain* D_t.

where $\mathbf{end}((u,v)) := v$ and $\mathbf{start}((u,v)) := v$. It is easy to see that $\sigma(i)$ is computable in space $O(\log|V|)$. Furthermore, define $G_{alt} := (V_{alt}, E_{alt})$ with $V_{alt} := V \setminus V_{sink}$ and $E_{alt} := E \cap (V_{alt} \times V_{alt})$.

We will now construct two graphs G_1 and G_2 where each of these graphs $G_i := (V_i, E_i)$ contains a copy of subgraph G_{alt} and a chain graph $D_{|V|^3} = (V_{|V|^3\text{−chain}}, E_{|V|^3\text{−chain}})$:

$$V_i \;:=\; V_{alt} \cup V_{|V|^3-\text{chain}} \qquad \text{and} \qquad E_i \;:=\; E_{alt} \cup E_{|V|^3-\text{chain}} \cup E_{i,\text{connect}}$$

with
$$E_{1,\text{connect}} := \{\, (\text{start}(\phi_{sink}(i)),\ p_{i\cdot|V|}) \mid p_{i\cdot|V|} \in V_{|V|^3-\text{chain}} \text{ and } 1 \le i \le |E_{sink}| \,\}$$
$$E_{2,\text{connect}} := \{\, (\sigma(i),\ p_{i\cdot|V|}) \mid p_{i\cdot|V|} \in V_{|V|^3-\text{chain}} \text{ and } 1 \le i \le |E_{sink}| \,\}\ .$$

The construction of both graphs is illustrated in figure 12.

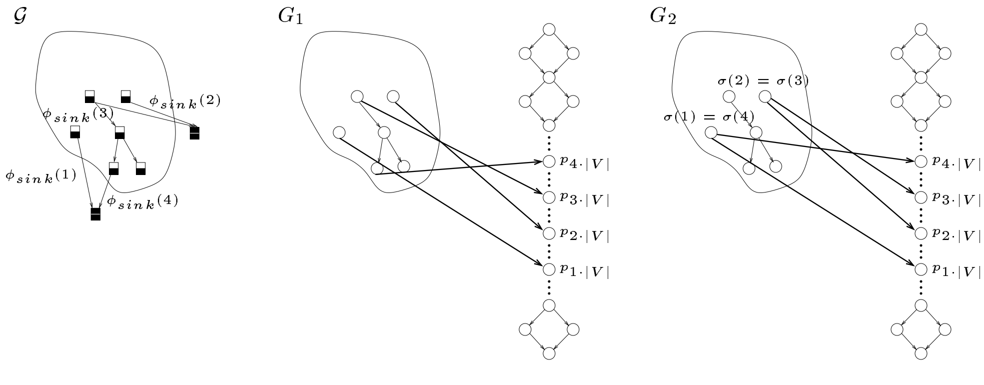

Figure 12: *The resulting graphs* G_1 *and* G_2 *from the reduction* ExDPG[ALT, 1-PAR | PAR] $\le_{\log}$ ENoP*.

Let us now consider the number of paths from a source of G_1 and G_2 to the sink p_0 of the chain graph in both graphs. For a subset of nodes V' of a graph G let $\#\mathbf{Path}_G[V' \to v]$ denotes the number of paths from a node $u \in V$ to v in G. Then it holds:

$$\begin{aligned}
\#\mathrm{Path}_{G_1}[V_1' \to p_0] &= \#\mathrm{Path}_{D_{|V|^3}}[p_{|V|^3} \to p_0] \\
&\quad + \sum_{i=1}^{|E_{sink}|} \#\mathrm{Path}_{G_1}[V_1' \to \mathrm{start}(\phi_{sink}(i))] \cdot \#\mathrm{Path}_{D_{|V|^3}}[p_{i\cdot|V|} \to p_0] \\
&= 2^{|V|^3} + \sum_{i=1}^{|E_{sink}|} 2^{i|V|} \cdot \#\mathrm{Path}_{G_1}[V_1' \setminus \{p_{|V|^3}\} \to \mathrm{start}(\phi_{sink}(i))] \\
\#\mathrm{Path}_{G_2}[V_2' \to p_0] &= \#\mathrm{Path}_{D_{|V|^3}}[p_{|V|^3} \to p_0] \\
&\quad + \sum_{i=1}^{|E_{sink}|} \#\mathrm{Path}_{G_2}[V_2' \to \sigma(i)] \cdot \#\mathrm{Path}_{D_{|V|^3}}[p_{i\cdot|V|} \to p_0] \\
&= 2^{|V|^3} + \sum_{i=1}^{|E_{sink}|} 2^{i|V|} \cdot \#\mathrm{Path}_{G_1}[V_1' \setminus \{p_{|V|^3}\} \to \sigma(i)]
\end{aligned}$$

where V_i' denotes the set of sources of G_i. Since $|E_{sink}| \leq |V|^2$ and $\#\mathrm{Path}_{G_i}[V_i' \setminus \{p_{|V|^3}\} \to u] < 2^{|V|}$ for all nodes $u \in V_i$, $i \in \{1,2\}$ (see Lemma 1) we can conclude that

$$\begin{aligned}
&\#\mathrm{Path}_{G_1}[V_1' \to p_0] = \#\mathrm{Path}_{G_2}[V_2' \to p_0] \\
\iff\quad & \forall i \in \{1,\ldots,|E_{sink}|\} \;:\; \#\mathrm{Path}_{G_1}[V_1' \to \mathrm{start}(\phi_{sink}(i))] = \#\mathrm{Path}_{G_2}[V_2' \to \sigma(i)] \\
\iff\quad & \forall u \in V_{sink}\ \forall v_1, v_2 \in \mathrm{pred}(u) \;:\; |W(v_1)| = |W(v_2)| \\
\iff\quad & \text{there exists a run for } \mathcal{G}
\end{aligned}$$

∎

Theorem 2 ExDPG[$\cdot$ | PAR] *is $C_=\mathcal{L}$-complete.*

3.3 NP-Hard Execution Problems

We consider first DPGs with PAR input and ALT output modes. Note first that for their symmetrical counterparts, the problem ExDPG[ALT | PAR] is trivial – any instance of the problem has a positive answer. Below we show that the complexity of the [PAR | ALT] variant is huge. Namely, we prove

Theorem 3 ExDPG[PAR | ALT] *is $\mathcal{NP}$-hard, even if the underlying DPG is planar and bipartite.*

Proof: The hardness follows by an reduction of 3DM to SE-DPG[PAR | ALT].

Definition 8 3-Dimensional Matching [3DM]
Let $M \subset W \times X \times Y$ where W, X, Y are disjoint sets of size q. Decide whether M contains a matching, i.e., a subset $M' \subseteq M$ such that $|M'| = q$ and no two elements of M' agree in any coordinate.

Note that the 3DM problem is $\mathcal{NP}$-complete even in the case of the planar version (see [3]).

Let $V_W := \{v_w \mid w \in W\}$, $V_X := \{v_x \mid x \in X\}$, $V_Y := \{v_y \mid y \in Y\}$, and $V_M := \{v_m \mid m \in M\}$. Further let

$$E := \{(v_w, v_m), (v_x, v_m), (v_y, v_m) | m = (w,x,y) \in M\}$$

and $V := V_W \cup V_X \cup V_Y \cup V_M$. The input and output modes are defined by $I \equiv$ PAR and $O \equiv$ ALT, respectively. The claim follows directly from the fact that if $H_\mathcal{G}$ is a run of $\mathcal{G}$ then for any node $v \in V_W \cup V_X \cup V_Y$, the set $W(v)$ has exactly one elemnt. ∎

The hardness of ExDPG[$\cdot$ | ALT] and ExDPG follows directly by inclusion. The same proof works also for the following restricted version of the executin problem: for a given DPGs $\mathcal{G}$ decide if there exists a run $H_\mathcal{G} = (W, F)$ of $\mathcal{G}$ with $|W(v)| \leq 1$ for any node v of $\mathcal{G}$. Let **ExDPG$_{unique}$[IN | OUT]** denote such a problem for DPGs with input and output mode functions described by IN, resp. OUT. Then we have

$$\text{ExDPG}_{unique}[\mathsf{PAR} \mid \mathsf{ALT}] \text{ remains } \mathcal{NP}\text{-hard.}$$

Because the size of a run for a given dynamic process graph can be even exponentially larger than the size of the graph (see Lemma 1) hence it is not obvious that the execution problem for unrestricted DPGs can be solved in $\mathcal{NP}$ at all.

Considering the characteristic vector of a run and the number of edges between execution instances of different processes a legal run can be characterized as follows:

Lemma 9 *Let $\mathcal{G}$ be a DPG and let $A(v_i), A(v_i, v_j) \in \mathbb{N}$ be values for all nodes v_i and edges (v_i, v_j) of $\mathcal{G}$. Then there exists a run $H_\mathcal{G}$ for $\mathcal{G}$ with $|W(v_i)| = A(v_i)$ and $|W(v_i, v_j)| = A(v_i, v_j)$ if and only if*

1. *$A(v_i) = 1$ for all sources v_i,*
2. *for all non-sources v_i with $I(v_i) = \mathsf{ALT}$: $A(v_i) = \sum_{(v_j,v_i)\in E} A(v_j, v_i)$,*
3. *for all non-sources v_i with $I(v_i) = \mathsf{PAR}$: $\forall (v_j, v_i) \in E \ : \ A(v_i) = A(v_j, v_i)$,*
4. *for all non-sinks v_i with $O(v_i) = \mathsf{ALT}$: $A(v_i) = \sum_{(v_i,v_j)\in E} A(v_i, v_j)$,*
5. *for all non-sinks v_i with $O(v_i) = \mathsf{PAR}$: $\forall (v_i, v_j) \in E \ : \ A(v_i) = A(v_i, v_j)$.*

Proof: The *only if*-case follows directly from the definition of the sets $W(v_i)$ and $W(v_i, v_j)$.

For the *if*-case we have to notice that we can easily construct the sets $W(v_i)$ and $W(v_i, v_j)$ which fulfill the properties of a run for given values $A(v_i)$ and $A(v_i, v_j)$. ∎

This lemma provides the basis for the following nondeterministic procedure that solves the most general problem ExDPG in polynomial time.

`procedure` *solve-ExDPG*($\mathcal{G}$)
1 `for all sources` v_i `of` $\mathcal{G}$ `set` $A(v_i) := 1$
2 `for all non-sources` v_i `of` $\mathcal{G}$ `choose nondeterministically` $A(v_i) \in \{0, \ldots, 2^{|V|-1}\}$
3 `for all edges` (v_i, v_j) `of` $\mathcal{G}$ `choose nondeterministically` $A(v_i, v_j) \in \{0, \ldots, 2^{|V|-1}\}$
4 `for all non-sinks` v_i `of` $\mathcal{G}$
5 `if` $O(v_i) = \mathsf{ALT}$ `then verify that` $A(v_i) = \sum_{(v_i,v_j)\in E} A(v_i, v_j)$ `and reject if not`
6 `else verify that` $A(v_i) = A(v_i, v_j)$ `for all edges` $(v_i, v_j) \in E$ `and reject if not`
7 `for all non-sources` v_i `of` $\mathcal{G}$
8 `if` $I(v_i) = \mathsf{ALT}$ `then verify that` $A(v_i) = \sum_{(v_j,v_i)\in E} A(v_i, v_j)$ `and reject if not`
9 `else verify that` $A(v_i) = A(v_j, v_i)$ `for all edges` $(v_j, v_i) \in E$ `and reject if not`
10 `accept`

The correctness of this procedure follows easily from Lemma 9: line 1 guarantees that condition (1) of the lemma is fullfield and in lines from 4 to 10 the procedure verifies whether conditions (2)–(5) are fullfield for values A guessed nondeterministically in line 1 and 2. Obviously, *solve-ExDPG* works in polynomial time. Then we obtain:

Theorem 4 ExDPG, $\text{ExDPG}_{unique} \ \in \ \mathcal{NP}$.

The inclusion for ExDPG follows strightforward. To show that ExDPG_{unique} problem lies in $\mathcal{NP}$ one can change in line 2 of the procedure abowe inclusion $A(v_i) \in \{0, \ldots, 2^{|V|-1}\}$ to the following one: $A(v_i) \in \{0, 1\}$.

It is easy to prove that ExDPG[$\cdot$ | ALT] and ExDPG_{unique}[$\cdot$ | ALT] remain $\mathcal{NP}$-complete even for DPGs with only one sink. On the other hand, these problems become easy if we consider DPGs with only one source. In this case, ExDPG with $O \equiv \mathsf{ALT}$ is equivalent to the decision problem whether there exists a path from the source to a sink where all nodes on the path with in-degree at least two have input mode ALT. Hence, a run of ExDPG[PAR | ALT] consists of chains only and is therefore solvable by a finite automata. By a reduction of the *DAG Graph Accessibility Problem [DAG-GAP]* we can show that the single source ExDPG[$\cdot$ | ALT]-problem is $\mathcal{NL}$-complete.

Theorem 5 *Restricted to DPG with one source the* ExDPG[$\cdot$ | ALT]*-problem is $\mathcal{NL}$-complete.*

Proof: For proving the $\mathcal{NL}$-hardness we will reduce the following variant of the GAP-problem to ExDPG[$\cdot$ | ALT].

Definition 9 DAG Graph Accessibility Problem [DAG-GAP]
Given a DAG G and two nodes x, y of G. Decide: Is there a path from x to y?

It is well known that DAG-GAP is $\mathcal{NL}$-complete.

Let $G = (V, E)$ and $x, y \in V$ be an instance of the DAG-GAP problem. Further let $v_1, \ldots, v_n$ be the sources and $w_1, \ldots, w_m$ be the sinks of G. W.l.o.g. assume that w_i are not isolated and that $m \geq 2$.

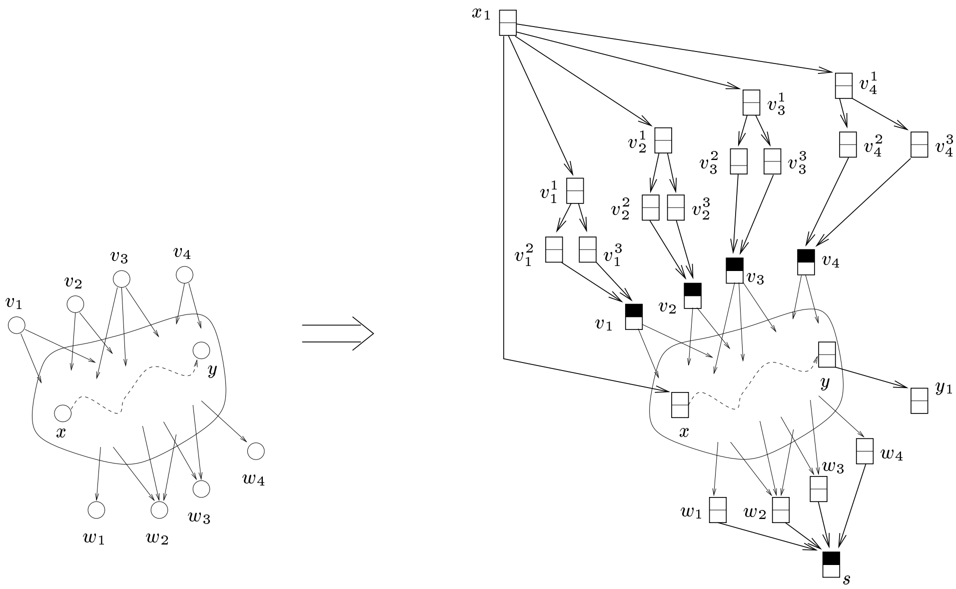

Figure 13: A reduction of DAG Graph Accessibility Problem to ExDPG[$\cdot$ | ALT] for DPGs with one source.

The dynamic process graph is generated as follows: For each source v_i generate a subgraph $A_i := (V_{\text{in}}^i, E_{\text{in}}^i)$ with $V_{\text{in}}^i := \{v_i, v_i^1, v_i^2, v_i^3\}$ and $E_{\text{in}}^i := \{(v_i^1, v_i), (v_i^2, v_i), (v_i^3, v_i^1), (v_i^3, v_i^2)\}$. Let s, x_1, and y_1 be three additional nodes. Finally let $G' = (V', E')$ (see figure 13) with

$$V' := V \cup \{s, x_1, y_1\} \cup \bigcup_{i=1}^{n} V_{\text{in}}^i$$

and $E' := E \cup \{(x_1, x), (y, y_1)\} \cup \bigcup_{i=1}^{n} E_{\text{in}}^i \cup \{(w_i, s) \mid i \in \{1, \ldots, m\}\} \cup \{(x_1, v_i^3) \mid i \in \{1, \ldots, n\}\}$ and $I(v_i) = \mathsf{PAR}$ for all $v_i \in \{s, v_1, \ldots, v_n\}$ and $I(v_i) = \mathsf{ALT}$ elsewhere. It is easy to see that the dynamic process graph (V', E', I, O) has a path from the source to a sink if and only if the sink of the path is y_1 and the second node on the path is x and therefore there exists a path from x to y. Moreover the whole construction is easy enough that it can be done in logaritmic space. ∎

4 Scheduling Dynamic Process Graphs

Let us now consider the problem to construct optimal schedules for dynamic process graphs. Since the execution problem is already hard in the less restricted cases one has to expect similar negative results for the scheduling problem. Our main result below, however, implies that the compaction provided by dynamic process graphs is quite efficient. The complexity of scheduling general DPGs increases to $\mathcal{NEXPTIME}$-complete. The hardness proofs will be the topic of this section. First, however we will analyze the complexity for restricted classes of DPGs.

4.1 The Tractable Cases

One of the main problems of a parallel system is to distributie data to the processes of a program efficiently. This concerns e.g. routing, broadcasting and scattering. Modeling such a job by a dynamic process graph only a restricted kind of input respectively output modes are needed. For example the routing problem needs only DPG with $I \equiv \mathsf{ALT}$ and $O \equiv \mathsf{ALT}$ whereas for the broadcasting $O \equiv \mathsf{PAR}$ is necessary. In the following we will investigate the complexity of the DPGS-problem where $I \equiv \mathsf{ALT}$. Furthermore we will present efficient (i.e. polynomial time bounded) algorithms for these problems.

Theorem 6 DPGS[ALT | ALT] *is $\mathcal{NL}$-complete.*

Proof: First note that any run of an instance of the DPGS[ALT | ALT] problem consists of a set of disjoint chains. Let (G, I, O) be a dynamic process graph with $O \equiv I \equiv \mathsf{ALT}$. Further let δ be the communication delay and T^* the deadline for (G, I, O). Then the DPGS[ALT | ALT]-problem can be solved as follows:

1. Let $v_1, \ldots, v_k$ be the ordered sequence of sources of G.
2. For all $i = 1, 2, \ldots, k$ verify nondeterministically that there is a path from v_i to a sink of G of length smaller than T^*.

Obviously the algorithm can be done in logarithmic space. To prove the hardness we will reduce the DAG-GAP-problem to DPGS[ALT | ALT]. Let $G = (V, E)$ and $x, y \in V$ be an instance of the DAG-GAP problem. Further let $v_1, \ldots, v_n$ be the sources and $w_1, \ldots, w_m$ be the sinks of G.

For $\ell := |V| + 2$ the dynamic process graph G' is constructed as follows: For each source v_i generate a chain $A_i := (V_{\text{in}}^i, E_{\text{in}}^i)$ with $V_{\text{in}}^i := \{v_i, v_i^1, \ldots, v_i^\ell\}$ and $E_{\text{in}}^i := \{(v_i^j, v_i^{j+1}) \mid 1 \le j < \ell\} \cup \{(v_i^\ell, v_i)\}\}$. Further define for each sink w_i a chain $B_i := (V_{\text{out}}^i, E_{\text{out}}^i)$ with $V_{\text{out}}^i := \{w_i, w_i^1, \ldots, w_i^\ell\}$ and $E_{\text{out}}^i := \{(w_i^j, w_i^{j+1}) \mid 1 \le j < \ell\} \cup \{(w_i, w_i^\ell)\}\}$. Let x_1 and y_1 be two additional nodes. Finally let $G' = (V', E')$ (see figure 14) with

$$V' := V \cup \{x_1, y_1\} \cup \bigcup_{i=1}^{n} V_{\text{in}}^i \cup \bigcup_{i=1}^{m} V_{\text{out}}^i$$

and

$$E' := E \cup \{(x_1, x), (y, y_1)\} \cup \bigcup_{i=1}^{n} E_{\text{in}}^i \cup \bigcup_{i=1}^{m} E_{\text{out}}^i \cup \{(x_1, v_i^1) \mid 1 \le i \le n\} .$$

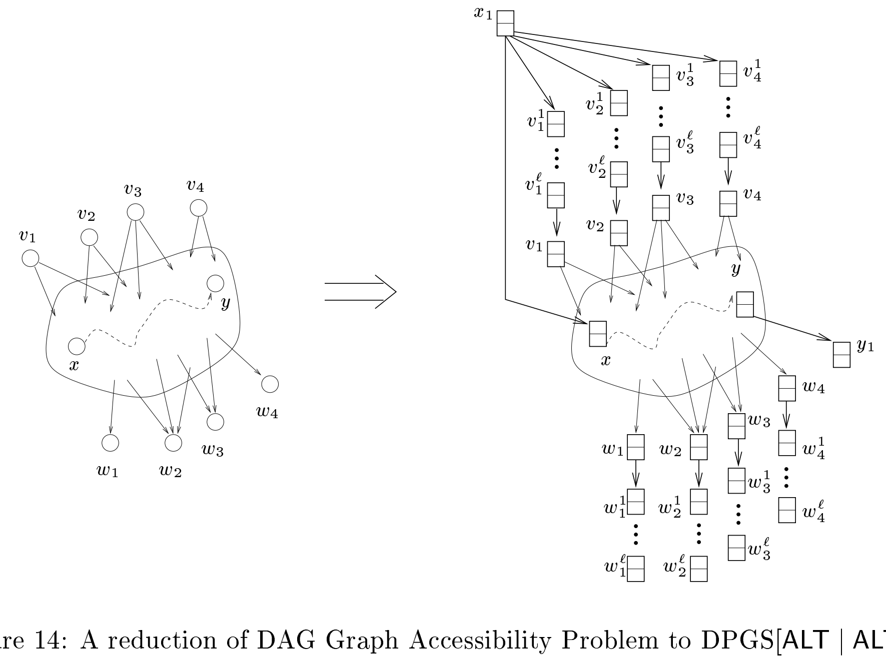

Figure 14: A reduction of DAG Graph Accessibility Problem to DPGS[ALT | ALT].

Define the deadline $T^* := |V| + 2$. It is easy to see that the dynamic process graph (G', I, O) with $I \equiv O \equiv \mathsf{ALT}$ and an arbitrary positive communication delay function has only a schedule of length at most T^* iff there exists a path from x to y. ∎

Theorem 7 DPGS[ALT | PAR] *is* $\mathcal{NL}$*-complete.*

Proof: Note that a run of an instance of the DPGS[ALT | PAR] problem is just a set of binary out-trees. Let (G, I, O) be a dynamic process graph with $I \equiv \mathsf{ALT}$ and $O \equiv \mathsf{PAR}$. Further let δ be a given communication delay and T^* a given deadline for (G, I, O). Then the DPGS[ALT | PAR]-problem can be solved as follows:

1. Let $v_1, \ldots, v_k$ be the ordered sequence of sources of G.
2. For all $i = 1, 2, \ldots, k$ verify universally (that means co-nondeterministically) that there is no path from v_i to a sink of G of length greater then T^*.

Since co$-\mathcal{NL} = \mathcal{NL}$ it remains to prove that the problem is hard for this class. To show this we will reduce the co-DAG-GAP-problem to DPGS[ALT | PAR].

Let $G = (V, E)$ and $x, y \in V$ be an instance of the co-DAG-GAP problem. For $\ell := |V| + 2$ define the dynamic process graph G' as follows: For the nodes x and y generate the chains $X := (V_x, E_x)$ with $V_x := \{x, x_1, \ldots, v_\ell\}$ and $E_x := \{(x_i, x_{i+1}) \mid 1 \leq i < \ell\} \cup \{(x_\ell, x)\})$ resp. $Y := (V_y, E_y)$ with $V_y := \{y, y_1, \ldots, y_\ell\}$ and $E_y := \{(y_i, y_{i+1}) \mid 1 \leq i < \ell\}\} \cup \{(y, y_1)\})$. Finally let $G' = (V', E')$ with $V' := V \cup V_x \cup V_y$ and $E' := E \cup E_x \cup E_y$ (see figure 15).

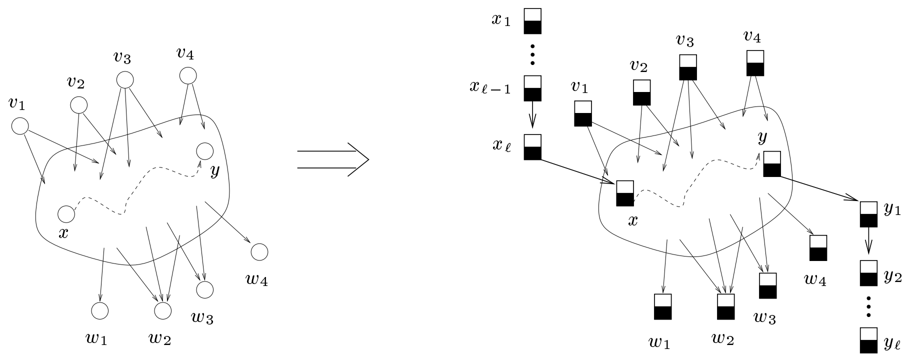

Figure 15: A reduction of DAG Graph Accessibility Problem to DPGS[ALT | PAR].

Define the deadline $T^* := |V| + \ell$. It is easy to see that the dynamic process graph (G', I, O) with $I \equiv \mathsf{ALT}$ and $O \equiv \mathsf{PAR}$ and an arbitrary positive communication delay function has a schedule of length at most T^* iff there exists no path from x to y. Otherwise any strategy $H_\mathcal{G}$ for $\mathcal{G}$ contains a path from x_1 to y_ℓ of length at least $2 \cdot \ell + 1 > |V| + \ell$. ∎

Theorem 8 DPGS[ALT | ·] *is* $\mathcal{P}$*-complete*

Proof: Analogously to DPGS[ALT | PAR] a run for an instance of the DPGS[ALT | ·] problem is a set of out-trees. The only difference is that for nodes with ALT output mode, each appropriate execution instance has exactly one predecessor and the problem arises to choose a predecessor that minimize the time of the whole schedule. A schedule of the minimal length can be constructed by the following recursive function: Let v be a node of a given dynamic process graph (G, I, O). Then define

$$T_{min}(v) := \begin{cases} 1 & v \text{ is a sink in } G = (V, E) \\ 1 + \max_{(v,u) \in E} T_{min}(u) & O(v) = \mathsf{PAR} \\ 1 + \min_{(v,u) \in E} T_{min}(u) & O(v) = \mathsf{ALT} \ . \end{cases}$$

It follows by an induction that for any communication delay δ

$$\max_{v- \text{ source of } G} \{T_{min}(v)\}$$

is the minimal length of schedule for the given Dynamic Process Graph. for short To prove the hardness we will reduce a special version of the Circuit Value Problem to DPGS[ALT | $\cdot$].

Definition 10 Circuit Value Problem [CVP]
Given a Circuit C and an assignment α of the input variables of C. Decide whether $C(\alpha) = 1$.

It is well known that the CVP-problem is $\mathcal{P}$-complete even if we restrict the problem to circuits (SAM2CVP see [7])

- which consists only of AND and OR gates and
- each path from an input to an output has the same length.

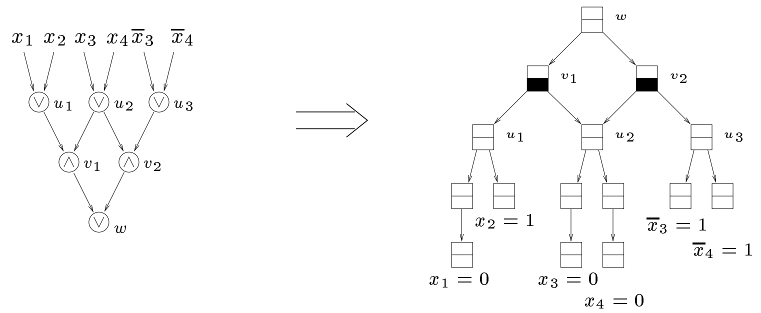

Figure 16: A reduction of CVP to DPGS[ALT | $\cdot$].

Let $G = (V, E)$, $\beta : V \to \{\wedge, \vee, x_1, \overline{x}_1, \ldots, x_n, \overline{x}_n\}$, and $\alpha : \{x_1, \ldots, x_n\} \to \{0, 1\}$ be an instance of the SAM2CVP-problem. Then define the dynamic process graph $\mathcal{G}_{G,\beta,\alpha} := (G', I, O)$ with $I \equiv \mathsf{ALT}$ as follows: For each input gate x_i of G generate a graph $X_i := (V_i, E_i)$ with

$$V_i := \begin{cases} \{x_i\} & \text{if } \alpha(x_i) = 1 \\ \{x_i, x_i'\} & \text{if } \alpha(x_i) = 0 \end{cases} \quad \text{and} \quad E_i := \begin{cases} \emptyset & \text{if } \alpha(x_i) = 1 \\ \{(x_i, x_i')\} & \text{if } \alpha(x_i) = 0 \end{cases},$$

respectively for $\overline{x}_i$ generate $\overline{X}_i := (\overline{V}_i, \overline{E}_i)$ with

$$\overline{V}_i := \begin{cases} \{x_i\} & \text{if } \alpha(x_i) = 0 \\ \{x_i, x_i'\} & \text{if } \alpha(x_i) = 1 \end{cases} \quad \text{and} \quad \overline{E}_i := \begin{cases} \emptyset & \text{if } \alpha(x_i) = 0 \\ \{(x_i, x_i')\} & \text{if } \alpha(x_i) = 1 \end{cases}.$$

Finally, let $G' := (V', E')$ with $V' := V \cup \bigcup_{i=1}^{n}(V_i \cup \overline{V}_i)$ and $E' := \bigcup_{i=1}^{n}(E_i \cup \overline{E}_i) \cup \{(x, y) \mid (y, x) \in E\}$. For the output mode of the Dynamic Process Graph define

$$O(v) := \begin{cases} \mathsf{ALT} & \text{if } \beta(v) = \vee \\ \mathsf{PAR} & \text{else.} \end{cases}$$

An example for such an instance of the SAM2CVP problem and the corresponding dynamic process graph is given in figure 16.

Let depth(G) denotes the depth of G. By induction it can easily be seen that a dynamic process graph $\mathcal{G}_{G,\beta,\alpha}$ as described above has a schedule of length depth$(G) + 1$ if and only if α is an satisfying assignment for the circuit (G, β). ∎

4.2 The $\mathcal{NP}$-Hard Versions

Below we show that some specific variants of the DPGS problem are $\mathcal{NP}$-complete. If we restrict our attention to problems for the dynamic process graph (V, E, I, O) with $I \equiv \mathsf{PAR}$ and arbitrary output modes O the $\mathcal{NP}$-hardness follows directly from the hardness of the scheduling problem with communication delay for in-trees (see [10]).

Theorem 9 *The DPGS problem is $\mathcal{NP}$-hard. This even holds if $I \equiv$ PAR.*

To give an $\mathcal{NP}$ algorithm for the DPGS problem restricted to dynamic process graphs (V, E, I, O) with $I \equiv$ PAR and arbitrary output function O we will use the following basic lemma:

Lemma 10 *Let $\mathcal{G} = (V, E, I, O)$ be a dynamic process graphs with $I \equiv$ PAR. Then it holds for all runs $H_{\mathcal{G}}$ and all nodes $v \in V$ that $|W(v)| \leq 1$.*

Proof: Assume that there exists a run $H_{\mathcal{G}}$ and a node $v \in G$ with $|W(v)| > 1$. By definition it holds that $|W(u)| = 1$ for all sources u. Hence there exists a node v_0 of G with $|W(v_0)| > 1$ and $|W(u)| \leq 1$ for all predecessors of v_0.

Let $\mathcal{G} = (V, E, I, O)$ be a dynamic process graphs with $I \equiv$ PAR then it follows from the definition of a run that $|W(v_0)| \leq \min_{u \text{ pred}(v_0)} |W(u)|$. So $|W(v_0)| \leq 1$ – a contradiction. ∎

It follows directly from Theorem 9 and Lemma 10 that the problems

$$\text{DPGS[PAR | ALT]}, \quad \text{DPGS[PAR | PAR]}, \quad \text{and} \quad \text{DPGS[PAR} \mid \cdot \text{]}$$

are $\mathcal{NP}$-complete.

A lower bound of $|W(v)|$ for dynamic process graphs $\mathcal{G} = (V, E, I, O)$ with $O \equiv$ ALT can be shown as follows:

Lemma 11 *Let $\mathcal{G} = (V, E, I, O)$ be a dynamic process graphs with $O \equiv$ ALT. Then it holds for all runs $H_{\mathcal{G}}$ and all nodes $v \in V$ that $|W(v)| \leq |V|$.*

Proof: Note that a run $H_{\mathcal{G}}$ is a forest where the leaves are given by the sources of $G = (V, E)$. Furthermore, it is easy to see that each source of G occurs exactly in one tree and each inner node in each tree of $H_{\mathcal{G}}$ at most once. Hence $|W(v)|$ is bounded by the number of sources of G. ∎

An $\mathcal{NP}$ algorithm for this problem follows from the small size and from the tree-like shape of the run $H_{\mathcal{G}}$ for a graph $\mathcal{G}$ with $O \equiv$ ALT.

Through the rest of this section we will focus on the DPGSproblem for dynamic process graphs with $O \equiv$ PAR and arbitrary input mode.

Theorem 10 *Restricted to dynamic process graphs with $O \equiv$ PAR the DPGSproblem is $\mathcal{BH}_2$-hard.*

To prove this theorem we introduce first some specific DPGs that will be useful also in the next section to prove our man result that the complexity of scheduling general dynamic graphs is $\mathcal{NEXPTIME}$-hard.

Line $\mathcal{L}_\ell$ of length $\ell \geq 1$, consists of ℓ vertices $v_1, v_2, \ldots, v_\ell$ and $\ell - 1$ edges: $(v_1, v_2) \ldots (v_{\ell-1}, v_\ell)$. Their input and output modes are inessential. However, for the completeness we set this modes as follows $I \equiv O \equiv$ PAR. The second DPG, called a **chain $\mathcal{D}_t$** of length t for $t \geq 0$, is an extension of chain for common graph that we defined in section 3 (see also figure 11). We simply choose for this graph $I \equiv$ ALT and $O(x) :=$ PAR for all its nodes.

Lemma 12 *Let $\mathcal{D}_t$ be a chain, with $t \geq 0$. Then there exists exactly one run $\mathcal{H}_t$ for $\mathcal{D}_t$. Furthermore, it holds that $T_{opt}(\mathcal{D}_t, \delta) = 2t + 1$ for any function $\delta \geq 0$.*

Proof: From the construction of $\mathcal{D}_t$ we can conclude that any run for $\mathcal{D}_t$ is a binary tree where any path from a source $s \in W(p_t)$ to a sink $t \in W(p_0)$ has exactly length $2t + 1$. Hence $T_{opt}(\mathcal{D}_t, \delta) = 2t + 1$ for all functions $\delta \geq 0$. ∎

To define **delayed chain $\mathcal{DD}_{n,\ell}$** we modify the chain $\mathcal{D}_t$ of length $t \geq 0$, replacing every node l_i, respectively every node r_i, by a line $l_{i,1}, \ldots, l_{i,\ell}$, resp. $r_{i,1}, \ldots, r_{i,\ell}$. The input connections of l_i, resp. r_i, are driven now to the source of the line graph (i.e. to $l_{i,1}$, resp. to $r_{i,1}$), and output connections run from its sinks: $l_{i,\ell}$ and $r_{i,\ell}$. Furthermore, for $h, \ell \geq 1$ we define the **counting graph $\mathcal{C}_{h,\ell}$** of height h consisting of one delayed chain $\mathcal{DD}_{h,\ell}$ of length h, called the **main** chain and for every $t \in \{h-1, \ldots, 0\}$ of pairs of chains, a **left** and a **right** one, each of length t. For every such t we add one edge from $l_{t+1,\ell}$ of the main chain to the source of the left chain of length t, and one edge from $r_{t+1,\ell}$ of the main chain to the source of the right chain of length t. An example of a counting graph is given in Fig. 17.

Lemma 13 *Let $\mathcal{C}_{h,\ell}$ be a counting graph, with $h, \ell \geq 1$. Then there exists exactly one run $\mathcal{H}$ for $\mathcal{C}_{h,\ell}$. Furthermore, if a_t and b_t, with $t \in [0..h-1]$ denote the sink of the left, respectively the right chain $\mathcal{D}_t$, then it holds that*

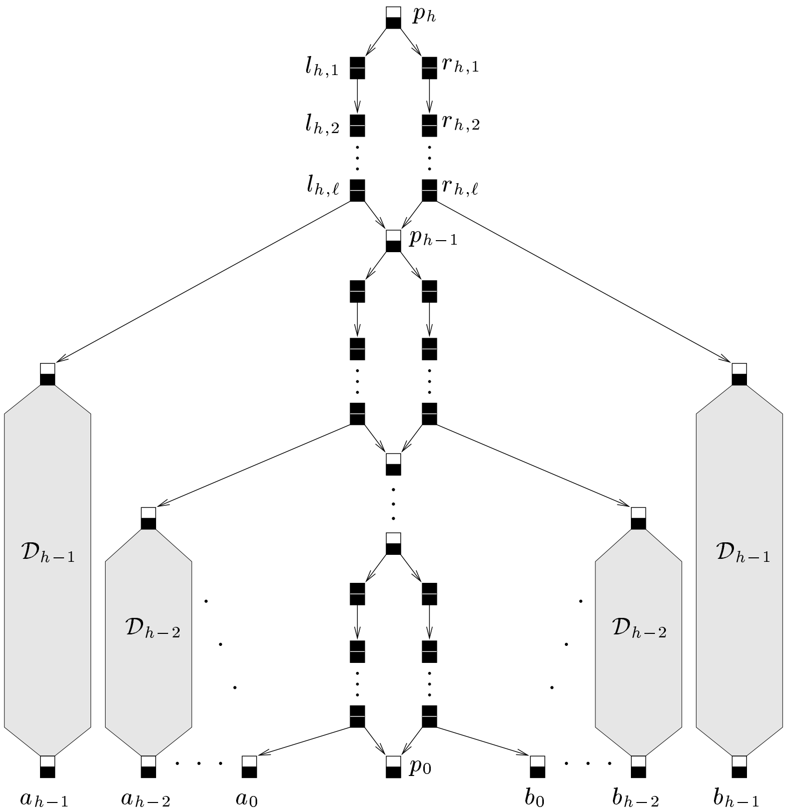

Figure 17: *A counting graph $\mathcal{C}_{h,\ell}$.*

- $|W(p_0)| = 2^h$ *and*
- $|W(a_t)| = |W(b_t)| = 2^{h-1}$*, for any $t \in [0..h-1]$.*

Proof: Note first that if we restrict our attention to the main chain of $\mathcal{C}_{h,\ell}$ then, similarly as in the proof of Lemma 12 one can show that there exist a unique run $\mathcal{H}_1$ of this subgraph and that for any node $p_t \in \{p_h, \dots, p_0\}$ of the main chain it holds $|W(p_t)| = 2^{h-t}$. Since the output mode of each p_t is PAR it implys that

$$|W(l_{t,\ell})| = |W(r_{t,\ell})| = |W(p_t)| = 2^{h-t}.$$

Now using Lemma 12 we obtain that for each chain $\mathcal{D}_t$, with $t \in [0..h-1]$, there exist a unique run. Next we combine (in an obvious way) these runs together with $\mathcal{H}_1$ to obtain a complete run $\mathcal{H}$ for $\mathcal{C}_{h,\ell}$. From Lemma 12 we conclude for any $t \in [0..h-1]$:

$$|W(a_t)| = |W(l_{t+1,\ell})| \cdot 2^t = 2^{h-t-1} \cdot 2^t = 2^{h-1}.$$

Similarly one obtains $|W(b_t)| = 2^{h-1}$. ∎

Now we are ready to give the Proof of Theorem 10.

Proof: The hardness follows by a reduction of SAT-UNSAT to DPGS[$\cdot$ | PAR].

Definition 11 SAT-UNSAT

Given two Boolean formulae F_1, F_2. Decide whether F_1 is satisfiable whereas F_2 is not?

It is shown in [16] that SAT-UNSAT is $\mathcal{BH}_2$-complete. It is easy to see that SAT-UNSAT is still $\mathcal{BH}_2$-complete if we restrict the formulas F_1, F_2 to Boolean formulas in 3-CNF.

Let F_1, F_2 be an instance of the SAT-UNSAT problem. We construct first two DPGs $\mathcal{G}_1$ and $\mathcal{G}_2$ such that $\mathcal{G}_1$ represents the formula F_1 and $\mathcal{G}_2$ the formula F_2. Then we combine these two graphs to form the resulting DPG $\mathcal{G}$ and we give such values T^* and δ that F_1 is satisfiable and F_2 is not if and only if $T_{opt}(\mathcal{G}, \delta) \leq T^*$.

We construct DPG $\mathcal{G}_1$ as an in-tree and we define appropriate communication delay δ in a very similar way as described in [10]. We can summarize the properties of this construction as follows.

Lemma 14 *Let F_1 be a boolean formula in 3-CNF and $\alpha, \beta \in I\!N$. Then there exists a logarithmic space bounded TM M_1 that generates on input $F_1\#1^\alpha\#1^\beta$ a dynamic process tree $\mathcal{G}_1$ with $I \equiv O \equiv$ PAR, a constant delay function δ (depending on the input length), and a deadline T_1^* such that*

1. *$\delta \geq \alpha$ and $T_1^* \geq 2 \cdot \delta + \beta$,*
2. *$T_{opt}(\mathcal{G}_1, \delta) \geq T_1^*$, and*
3. *$T_{opt}(\mathcal{G}_1, \delta) = T_1^*$ iff there exists a satisfying assignment for F_1.*

The proof is analogous with the proof of Theorem 1 in [10] therefore we omit it here.

We choose the values $\alpha, \beta \in \mathbb{N}$ depending on the complete DPG $\mathcal{G}$ and the formula F_2 in the following way $\alpha := |\mathcal{G}| - |\mathcal{G}_1|$ and $\beta := 6 \cdot n + 1$ where $\boldsymbol{n}$ denotes be the number of variables of F_2. Let

$$F_2(x_1, \ldots, x_n) := \bigwedge_{i=1,\ldots,m} (y_{i,1} \vee y_{i,2} \vee y_{i,3})$$

be an UNSAT formula, where $y_{i,j} \in \{x_1, \overline{x}_1, \ldots, x_n, \overline{x}_n\}$ be the j-th literal of the i-th clause. To construct the representation of an UNSAT-formula F_2 some additional structures will be needed. Note that the problem to decide whether $F_2(x_1, \ldots, x_n)$ is not satisfiable is equivalent to the problem to decide whether

$$F_2' \ := \ \left(\ \forall x_1 \ \ldots \forall x_n \ : \ \bigvee_{i=1,\ldots,m} (\overline{y}_{i,1} \wedge \overline{y}_{i,2} \wedge \overline{y}_{i,3}) \ \right)$$

it true. In the following we will construct in fact a dynamic process graph $\mathcal{G}_2$ that encodes this formula. We start with $\boldsymbol{\mathcal{U}_n}$ – a DPG that represents the quantifiers $\forall x_1 \ \ldots \forall x_n$ and then we introduce m DPGs $\boldsymbol{\mathcal{T}_i}$ representing the conjunctions $C_i := (\overline{y}_{i,1} \wedge \overline{y}_{i,2} \wedge \overline{y}_{i,3})$.

$\mathcal{U}_n$ will be constructed as follows. Let $\mathcal{C}_{n,6m+1}$ be a counting graph. Recall n is the number of variables of F_2' and m is the number of conjunctions of this formula. The source, respectively the sink of the main chain of $\mathcal{C}_{n,6m+1}$ will be denoted by r_2, resp. $u = p_0$. The sinks of the right chains $\mathcal{D}_i$ we denote by x_{i+1} and the sinks of the left ones by $\overline{x}_{i+1}$. To complete the construction of $\mathcal{U}_n$ add $n+1$ nodes to $\mathcal{U}_n$: $s_1, \ldots, s_n$, and s with $I(s_i) =$ ALT for all $i \in \{1, \ldots, n\}$ and $I(s) =$ PAR. Furthermore draw the edges $(x_i, s_i), (\overline{x}_i, s_i), (s_i, s)$ for all $i \in \{1, \ldots, n\}$ and finally the edge (p_0, s) (see Figure 18).

Now we introduce the DPGs $\mathcal{T}_i$ to represent conjunctions $C_i = (\overline{y}_{i,1} \wedge \overline{y}_{i,2} \wedge \overline{y}_{i,3})$. Each $\mathcal{T}_i$ consists of six chains $\mathcal{D}_{n-1}$ and eight additional nodes $f_{i,1}, f_{i,2}, f_{i,3}, t_{i,1}, t_{i,2}, t_{i,3}, f$, and c_i with $I(f_j) = I(t_j) =$ ALT for $j \in \{1,2,3\}$ and $I(f) = I(c_i) =$ PAR. The nodes of $\mathcal{T}_i$ are connected as shown in Figure 19. Furthermore we combine thiese graphs with $\mathcal{U}_n$ together connecting $f_{i,1}, f_{i,2}, f_{i,3}, t_{i,1}, t_{i,2}$ and $t_{i,3}$ with such nodes of $x_1, \overline{x}_1, \ldots, x_n, \overline{x}_n$ which correspond to literals $y_{i,1}, y_{i,2}, y_{i,3}$, $\overline{y}_{i,1}, \overline{y}_{i,2}$, and $\overline{y}_{i,3}$ respectively. Speaking more formally for $i = 1, 2, \ldots, m$ and $j = 1, 2, 3$ we add $6 \times m$ edges: $(x_k, f_{i,j})$ and $(\overline{x}_k, t_{i,j})$ if $y_{i,j} = x_k$ and edges: $(\overline{x}_k, f_{i,j})$ and $(x_k, t_{i,j})$ if $y_{i,j} = \overline{x}_k$. Through the rest of the proof we will identify $y_{i,j}$ and $\overline{y}_{i,j}$ with the corresponding variable nodes. Finally, for each i we draw edge (c_i, s). This completes the construction of $\mathcal{G}_2$.

Let us now investigate basic properties of runs for $\mathcal{G}_2$.

Lemma 15 *Let $\mathcal{G}_2$ be a DPG as defined above. Then there exists a run for $\mathcal{G}_2$. Further for any run $\mathcal{H}$ of $\mathcal{G}_2$ it holds that*

- *$|W(r_2)| = 1$ and $|W(s)| = 2^n$,*
- *$|W(x_i)| = |W(\overline{x}_i)| = 2^{n-1}$, for any $i \in [1..n]$, and*
- *$|W(c_i)| = 2^n$, for any $i \in [1..m]$.*

Proof: We constuct a run $\mathcal{H}$ for $\mathcal{G}_2$ as follows. Let us consider first the counting subgraph $\mathcal{C}_{n,6m+1}$ of graph $\mathcal{U}_n$. From Lemma 13 it follows that there exiest a run $\mathcal{H}_1$ for $\mathcal{C}_{n,6m+1}$. Furthermore, from the same lemma

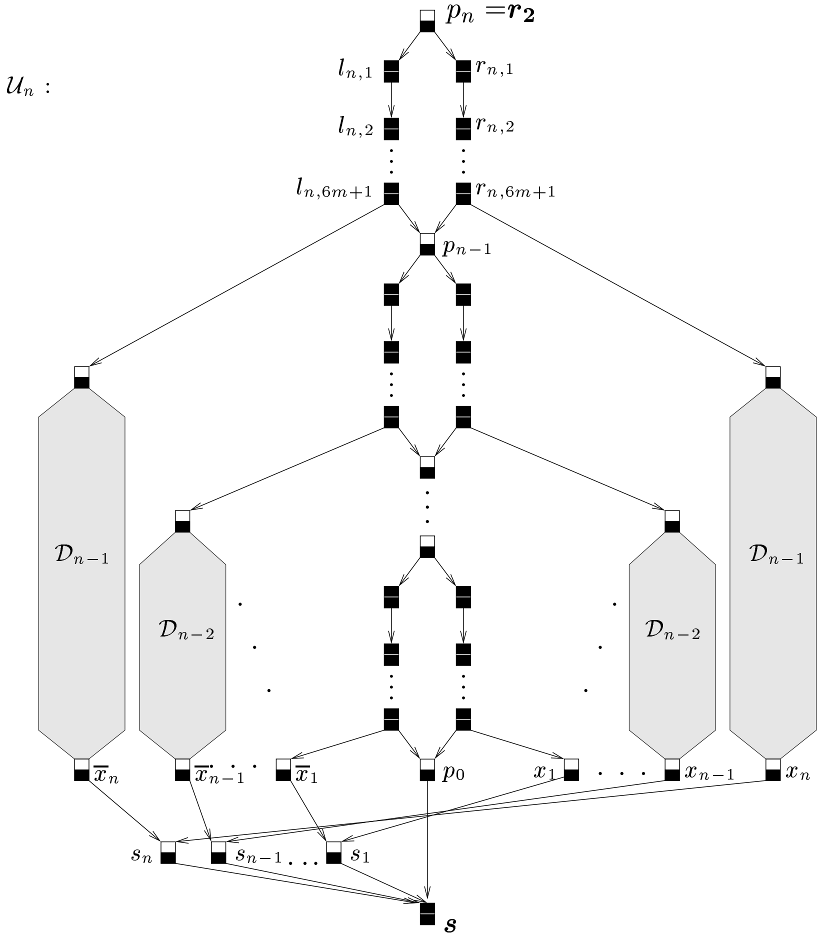

Figure 18: *A subgraph of the DPG $\mathcal{G}_2$ which represents the quantifiers $\forall x_1 \dots \forall x_n$ of UNSAT.*

we conclude that $|W(p_0)| = |W(s)| = 2^n$ and that for any $i \in [1..n]$, $|W(x_i)| = |W(\overline{x}_i)| = 2^{n-1}$. Since the input mode of nodes $f_{i,j}$ and $t_{i,j}$ of $\mathcal{T}_i$ graph is ALT hence we obtain additionally 2^{n-1} execution nodes for each $f_{i,j}$ and $t_{i,j}$. These executions are connected with appropriate nodes in $\bigcup_{i=1}^{n} W(x_i) \cup W(\overline{x}_i)$.

By Lemma 12 each chain $\mathcal{D}_{n-1}$ of $\mathcal{T}_i$ has a run and the number of execution nodes for the sink of $\mathcal{D}_{n-1}$ is 2^{n-1}. Hence we obtain 2^{n-1} additional execution nodes for each $f_{i,j}$ and $t_{i,j}$. In this case each such execution node is connected with an appropriate sink of a run graph for $\mathcal{D}_{n-1}$. Therefore $|W(f_{i,j})| = |W(t_{i,j})| = 2^n$, for all $i \in [1..n]$ and $j \in [1,2,3]$.

To complete the construction of the resulting run $\mathcal{H}$, for any i we add a set of nodes $W(f_i)$ and $W(c_i)$, with $|W(f_i)| = |W(c_i)| = 2^n$ then we connect each node of $W(f_i)$ with one node in $W(c_i)$ and finnaly we connect these nodes with the rest in an obvious way. ∎

Now we are ready to give the complete construction of our resulting DPG $\mathcal{G}$. Let $\mathcal{L}_K$ be a line graph of length

$$K \;:=\; n(6m+2) + n^2 + n + 5m + 2 - 1.$$

Then connect the source r_1 of $\mathcal{G}_1$ with $\mathcal{L}_K$ and the sink r_2 of $\mathcal{G}_2$ (see Figure 20) and define the communication delay and the deadline of $\mathcal{G}$ as follows

$$\boldsymbol{\delta} \;:=\; |\mathcal{G}| - |\mathcal{G}_1| \qquad \boldsymbol{T}^* \;:=\; T_1^* + K.$$

From Lemma 14 it follows directly that T^* is a lower bound of the minimal schedule length of $\mathcal{G}$. More precisely, $\mathcal{G}$ has a schedule of length T^* only if $\mathcal{G}_1$ has a schedule of length T_1^*, i.e. F_1 is satisfiable. Now assume that F_1 is satisfiable and that S is a schedule of length T^* for $\mathcal{G}$ and communication delay δ. For a processor P let $P(S)$ denote the set of tasks computed by P. Then a correctnes of our whole construction follows from Lemma 16 and Lemma 17 presented below.

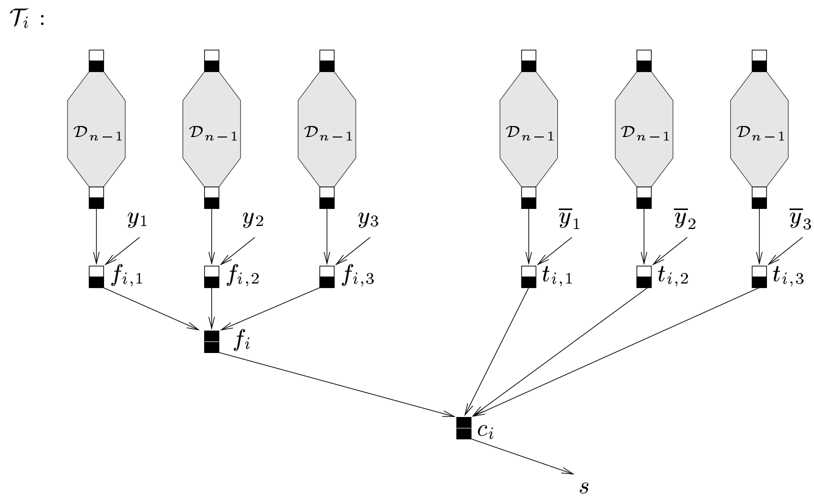

Figure 19: *A subgraph $\mathcal{T}_i$ of the DPG $\mathcal{G}_2$ which represents the i-th conjunction $C_i = (\overline{y}_{i,1} \wedge \overline{y}_{i,2} \wedge \overline{y}_{i,3})$.*

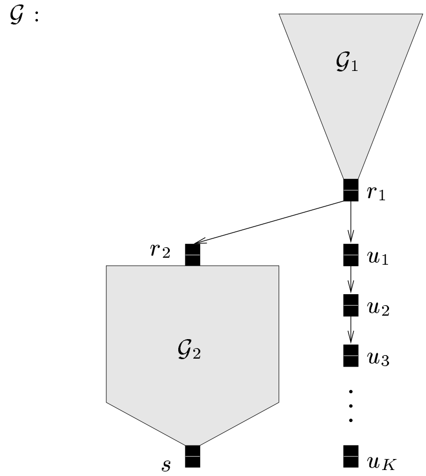

Figure 20: *The resulting DPG with $K := n(6m + 2) + n^2 + n + 5m + 2 - 1$ that represents an instance of the SAT-UNSAT problem.*

Lemma 16 (Universal Quantifier Correctness) *There are 2^n processors $P_1, \ldots, P_{2^n}$ computing tasks of $W(s)$ such that*

(1) for any processor P_j and any $i \in [1..n]$ P_j computes either a task of x_i or a task of $\overline{x}_i$, i.e. it holds

$$W(x_i) \cap P_j(S) \neq \emptyset \iff W(\overline{x}_i) \cap P_j(S) = \emptyset$$

and

(2) for any pair of processors P_i, P_j, with $i \neq j$, the sets of literals for tasks computed in P_i, resp. P_j are disjunct, i.e. it holds:

$$\{z \in \{x_1, \overline{x}_1, \ldots, x_n, \overline{x}_n\} | \, W(z) \cap P_i(S) \neq \emptyset\} \neq \{z \in \{x_1, \overline{x}_1, \ldots, x_n, \overline{x}_n\} | \, W(z) \cap P_j(S) \neq \emptyset\}.$$

Lemma 17 (Satisfiability) *Let P be a processor which computes a task $w_s \in W(s)$. Then there exists $i \in [1..m]$ such that P computes three tasks w_1, w_2, w_3, with*

$$w_1 \in W(\overline{y}_{i,1}), \; w_2 \in W(\overline{y}_{i,2}), \; \textit{and } w_3 \in W(\overline{y}_{i,3}).$$

To prove that F_2' is true when such a schedule S exists one can follow from Lemma 16 that there are 2^n processors $P_1, \ldots, P_{2^n}$ computing tasks of $W(s)$ which fulfil condition (1) and (2) of the Lemma. We will interpret computations of processors P_j as evaluations of the formula

$$\bigvee_{i=1,\ldots,m} (\overline{y}_{i,1} \wedge \overline{y}_{i,2} \wedge \overline{y}_{i,3}) \tag{2}$$

of F_2' for specific values of variables $x_1, \ldots, x_n$ as follows: if P_j computes a task of x_i we say P_j evaluates variable x_i as true; if P_j computes a task of $\overline{x}_i$ then P_j evaluates x_i as false. Next, for any conjunction $C_i = (\overline{y}_{i,1} \wedge \overline{y}_{i,2} \wedge \overline{y}_{i,3})$ we say that P_j evaluates this conjunction as true if and only if P_j computes three tasks w_1, w_2, w_3, with

$$w_1 \in W(\overline{y}_{i,1}),\ w_2 \in W(\overline{y}_{i,2}),\ \text{and } w_3 \in W(\overline{y}_{i,3}).$$

Finally P_j evaluates the formula (2) as true if and only if there exists $i \in [1..m]$, such that P_j evaluates conjunction C_i as true. From Lemma 16(1) we have that P_j evaluates each variable x_i either as true or false. Furthermore this condition implies also that for any $i \in [1..m]$ P_j evaluates the conjunction $C_i = (\overline{y}_{i,1} \wedge \overline{y}_{i,2} \wedge \overline{y}_{i,3})$ as true if and only if for the evaluation of variables of P_j the conjunction C_i is true. Hence our interpretation is correct.

Now from Lemma 16(2) it follows that the processors $P_1, \ldots, P_{2^n}$ test all possible 2^n evaluations for variables $x_1, \ldots, x_n$ and by Lemma 17 follows that any processor P_j evaluates formula (2) as true. Hence F_2' is true.

On the other hand if one assumes that F_1 is satisfiable and F_2' is true then the following schedule of $\mathcal{G}$ has the length T^*. First let processor P_0 compute the sink node r_1 of $\mathcal{G}_1$ and then the line $\mathcal{L}_K$. According to Lemma 14 P_0 coputes these tasks in time $T_1^* + K = T^*$. Next we construct a schedule for $2 \cdot 2^n$ processors which compute the tasks of $\mathcal{G}_2$. With each $j \in [1..2^n]$ we associate two processors P_j and P_j'. The first processor computes (in the same way as P_0) the sink node r_1 of $\mathcal{G}_1$ and then tasks of $\mathcal{U}_n$, $f_{i,k}, t_{i,k}, f_i$, and c_i, with $i \in [1..m]$ and $k = 1, 2, 3$. Processors P_j' compute tasks of appropriate $\mathcal{D}_{n-1}$ graphs and each P_j' computes exactly one task of f_i, for some $i \in [1..m]$. Speaking more formally we define schedule for these processors as follows. Let $(a_1, \ldots, a_n)$, with $a_i \in \{\mathit{true, false}\}$, be the j-th evaluation of variables $x_1, \ldots, x_n$ e.g. with respect to lexicographical order. Since F_2' is true, there exists $i_0 \in [1..m]$ such that for the evaluation $(a_1, \ldots, a_n)$ the conjunction $(\overline{y}_{i_0,1} \wedge \overline{y}_{i_0,2} \wedge \overline{y}_{i_0,3})$ is true.

schedule for P_j

tasks	time
1. compute the sink node r_1 of $\mathcal{G}_1$	T_1^*
2. for $i = n, n-1, \ldots, 1$ compute p_i and if $a_i = \mathit{true}$ then compute $r_{i,1}, r_{i,2}, \ldots, r_{i,6m+1}$, $2(i-1)$ tasks of $\mathcal{D}_{i-1}$, and x_i if $a_i = \mathit{false}$ then compute $l_{i,1}, l_{i,2}, \ldots, l_{i,6m+1}$, $2(i-1)$ tasks of $\mathcal{D}_{i-1}$, and $\overline{x}_i$ compute p_0	$n(6m+2) + \sum_{i=1}^{n}(2i-1) + 1 =$ $n(6m+2) + n^2 + 1$
3. compute tasks of $s_1, s_2, \ldots s_n$	n
4. for $i = 1, \ldots, m$ and $k = 1, 2, 3$ if $y_{i,k} = \mathit{true}$ then *$y_{i,k} = true$ means that the literal $y_{i,k}$* *is true according to $(a_1, \ldots, a_n)$* compute a task of $f_{i,k}$ if $y_{i,k} = \mathit{false}$ then compute a task of $t_{i,k}$	$3m$
5. for $i \in \{1, 2, \ldots, n\} \setminus \{i_0\}$ compute tasks of f_i	$m-1$
6. for $i = 1, \ldots, m$ compute tasks of c_i	m
7. compute a task of s	1

schedule for P'_j

tasks	time
1. for $i = 1, \ldots, m$ and $k = 1, 2, 3$ if $y_{i,k} = \mathit{true}$ then compute $2(n-1)+1$ tasks of $\mathcal{D}_{n-1}$ and a task of $t_{i,k}$ if $y_{i,k} = \mathit{false}$ then compute $2(n-1)+1$ tasks of $\mathcal{D}_{n-1}$ and a task of $f_{i,k}$	$3m \cdot (2(n-1)+2)$
2. compute a task of f_{i_0}	1

It is obvious that P'_j can compute tasks (1.) in time $3m \cdot (2(n-1)+2)$ since they have no predecessors neither in $\mathcal{G}_1$ nor in $\mathcal{U}_n$. Moreover P'_j can compute task (2.) immediately after computing of tasks (1.) because among these tasks P'_j has computed tasks of $f_{i_0,1}, f_{i_0,2}$, and $f_{i_0,3}$ (recall that according to the evaluation $(a_1, \ldots, a_n)$ the conjunction $(\overline{y}_{i_0,1} \wedge \overline{y}_{i_0,2} \wedge \overline{y}_{i_0,3})$ is true). Hence processor P'_j works in time

$$3m \cdot (2(n-1)+2) + 1 \;<\; |\mathcal{G}| - |\mathcal{G}_1| \;=\; \delta.$$

Processor P_j computes tasks (1..4) in time $T_{1..4} = T_1^* + n(6m+2) + n^2 + n + 3m + 1$. To compute tasks (5..7) it needs the results of tasks computed by P'_j. Note that

$$T_{1..4} \;>\; T_1^* \;>\; 2\delta$$

and P'_j finishes its computation in time $< \delta$. Therefore P_j can use the results of P'_j in step $T_{1..4}+1$ since the communication delay is equal to δ. Then the total time of computation of P_j is

$$T_{1..4} + 2m \;=\; T_1^* + n(6m+2) + n^2 + n + 5m + 1 \;=\; T^*.$$

To complete the proof of Theorem 9 we have to prove Universal-Quantifier-Correctness-Lemma and Satisfiability-Lemma.

Proof: [Lemma 16] Let $\mathcal{H}$ be a run of $\mathcal{G}$ for the schedule S. Furthermore let P be a processor computing a task $w_s \in W(s)$. We have that P has to compute r_1 and because the Lemma 14 this can be done in time T_1^*. From the construction of $\mathcal{U}_n$ and the restriction of the communication delay function it follows directly that P has to compute also

- a set of tasks representing at least one path of the main chain of $\mathcal{U}_n$ from the source r_2 of $\mathcal{U}_n$ to s,
- a set of tasks representing at least one path either from $l_{i,m+1}$ to $\overline{x}_i$ or from $r_{i,m+1}$ to x_i for each $i \in \{1, \ldots, n\}$ and
- a task for any node s_i and c_i.

Hence P has to compute at least

$$\begin{aligned} t \;&:=\; n(6m+2) + 1 + \sum_{i=0}^{n-1}(2i+1) + n + 4m + 1 \\ &=\; n(6m+2) + n^2 + n + 4m + 2 \;=\; T^* - T_1^* - m + 1 \end{aligned}$$

tasks of $\mathcal{U}_n$ and the subgraphs $\mathcal{T}_i$.

Assume that there exists a processors P computing tasks $w_s \in W(s)$, $w_1 \in W(x_i)$, and $w_2 \in W(\overline{x}_i)$ for at least one $i \leq n$. From the choice of the delay function δ it follows that P has also to compute a set of tasks representing the nodes $l_{i,1}, \ldots, l_{i,6m+1}$ and a set of tasks representing $r_{i,1}, \ldots, r_{i,6m+1}$. Hence, P has to compute at least $t+m+1$ tasks within $t+m-1$ steps – a contradiction.

From Lemma 15 follows we have at lest 2^n such processors P because $|W(s)| = 2^n$. Hence (1) follows.

To prove (2) we have to consider that each task of p_0 – and therefore of s – represents a unique path of the main chain, that has to be computed by the processor computing the corresponding task of s. Hence (2) follows analogously to (1). ∎

Proof: [Lemma 17] Note that for any schedule of length at most T^* any processor P that computes a task $w_s \in W(s)$ has also to compute all successors of w_s corresponding to nodes of $\mathcal{U}_n$, i.e.

$$\begin{aligned} t &:= n \cdot (6m+1) + 1 + \sum_{i \le n-1} (2 \cdot i + 1) + n + 1 \\ &= n \cdot (6m+1) + n^2 + n + 2 \;=\; T^* - T_1^* - 5m + 1 \end{aligned}$$

tasks of $\mathcal{U}_n$. Furthermore, it follows from the choice of δ that all tasks of the chain DPGs of $\mathcal{T}_i$ can be compute by some separate processors.

From Lemma 16 we can conclude that no task of $W(c_i)$ depends on one task of $W(\overline{x}_j)$ and one task of $W(x_j)$. Now assume, that any task t_{c_i} of $\bigcup_{i \in [1..m]} W(c_i)$ computed by P depends on at least one task $W(x_j)$ where x_j is a variable of clause C_i. Then P has to compute at least 5 tasks of any subgraph $\mathcal{T}_i$ – one task $W(f_{i,b}) \cup W(t_{i,b})$ ($b \in \{1,2,3\}$), one task of $W(f_i)$ and one of $W(c_i)$. Hence, P has to compute $5m$ tasks within $T^* - T_1^* - t = 5m - 1$ steps – a contradiction.

Hence, there exists at least one node c_j where P compute no task of $W(y_{i,1}) \cup W(y_{i,2}) \cup W(y_{i,3})$, but one task of any set $W(\overline{y}_{i,1})$, $W(\overline{y}_{i,2})$, and $W(\overline{y}_{i,3})$. ∎

∎

4.3 The general case

Let $\mathcal{G} = (V, E, I, O)$ be a given DPG, δ be a communication delay, and let T^* be a deadline. Then to solve the problem if $T_{opt}(\mathcal{G}, \delta) \le T^*$ holds one can guess nondeterministically $H_{\mathcal{G}}$ – a run of $\mathcal{G}$ next to guess a schedule S for $H_{\mathcal{G}}$ and finally to verify that $T(S) \le T^*$. To check that $H_{\mathcal{G}}$ in fact is a run of $\mathcal{G}$ we can use Lemma 9. Hence this tast can be done deterministically in polynomial time with respect to the lenght of $H_{\mathcal{G}}$. It is easy to see that the remaining tests can be done also in such time. Hence from Lemma 1 one deduces that the whole procedure can be done nondeterministically in exponential time with respect to the lenght of $\mathcal{G}$. Note that because the bound of Lemma 1 is tight the above exponential time is achieved in the worst case. This give

Lemma 18 *DPGS problem can be solved in $\mathcal{NEXPTIME}$.*

Through the rest of this section we will prove that DPGS problem is hard for this class.

Theorem 11 *Scheduling dynamic process graphs is $\mathcal{NEXPTIME}$-complete, even in case of constant communication delay.*

Proof: From Lemma 18 we have that the problem is in $\mathcal{NEXPTIME}$. To prove that it is hard we will use the reduction to the following problem

Definition 12 SUCCINCT-3SAT: *As input we are given a Boolean circuit over the standard AND, OR, NOT-basis that succinctly codes a Boolean formula in conjunctive normal form with the additional property that each clause has exactly three literals and each literal appears exactly three times. Suppose that the encoded formula consists of n variables and m clauses. On input $(0, i, k)$ with $i \in \{0, \ldots, n-1\}$ and $k \in \{1,2,3\}$ (appropriately coded in binary), the coding circuit returns the index of the clause where the literal $\neg x_i$ appears the k-th time. On input $(1, i, k)$ it returns the index of the clause where x_i appears for the k-th time. On input $(2, j, k)$ with $j \in \{0, \ldots, m-1\}$ and $k \in \{1,2,3\}$, it returns the k-th literal of the j-th clause.*
The problem is to decide whether the encoded formula is satisfiable.

The $\mathcal{NEXPTIME}$-completeness for this problem has been proved by Papadimitriou and Yannakakis in [17].

For the reduction of SUCCINCT-3SAT to the scheduling problem, the first crucial step is a transformation of a Boolean circuit B into a DPG.

Each input gate x of B is encoded by two nodes x_f and x_t with output mode PAR. The meaning of the indices here and later is as follows: x_f codes the value *false* for x, while x_t *true* . A NOT gate x is encoded by a graph of four vertices x_f, x_t, v_f, v_t with two edges: (x_f, v_t) and (x_t, v_f). The nodes x_f, x_t represent the input of this gate, while v_f, v_t the output.

Each AND , resp. OR gate is represented by a graph of ten vertices $x_f, x_t, y_f, y_t, p_1, p_2, p_3, p_4, v_f, v_t$ as follows. In both cases, there are edges $(x_f, p_1), (y_f, p_1), (x_t, p_4), (y_t, p_4), (x_f, p_2), (y_t, p_2), (x_t, p_3), (y_f, p_3)$. Furthermore, for an AND gate we add the edges $(p_1, v_f), (p_2, v_f), (p_3, v_f), (p_4, v_t)$, while for an OR gate $(p_1, v_f), (p_2, v_t), (p_3, v_t), (p_4, v_t)$. p_1, p_2, p_3, p_4 are internal nodes to establish the connection between the x and y inputs to the v outputs.

The input and output modes of these subgraphs are defined as follows: For all sources $x \in \{x_f, x_t, y_f, y_t\}$ choose $I(x) = O(x) := \mathsf{ALT}$; for all sinks $v \in \{v_f, v_t\}$ let $I(v) := \mathsf{ALT}$ and $O(v) := \mathsf{PAR}$; finally, for all inner nodes p_i define $I(p_i) := \mathsf{PAR}$ and $O(p_i) := \mathsf{ALT}$.

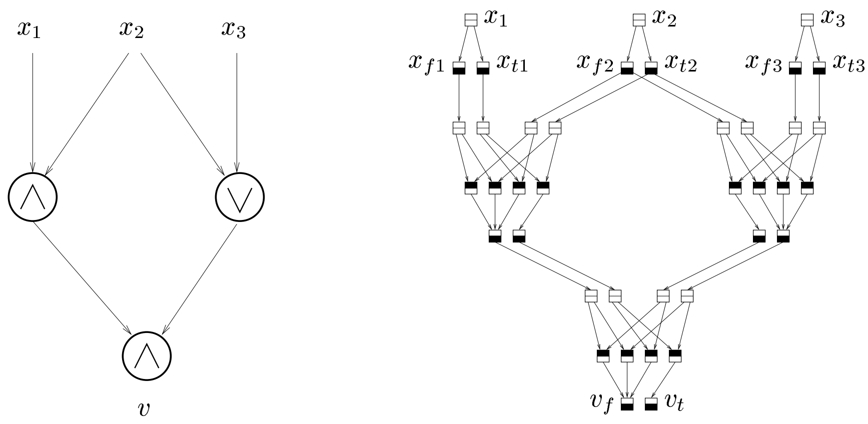

Figure 21: *A Boolean circuit and its DPG with the auxiliary nodes x_i.*

To encode an arbitrary Boolean circuit B by a DPG $\mathcal{G}$ we code each gate of B separately as described above and draw edges from the output nodes of each gate to the appropriate input nodes.

To understand the functionality assume that to the graph $\mathcal{G}$ additional temporary nodes $x_1, x_2, \ldots, x_r$ are added, each with output mode ALT. x_i will be connected to x_{fi} and x_{ti} via (x_i, x_{fi}) and (x_i, x_{ti}). Our encoding achieves the following property. Let $x_1, x_2, \ldots, x_r$ be the input gates of B, and $v_1, v_2, \ldots, v_s$ its output gates. In the coding graph $\mathcal{G}$ there will be nodes x_{fi}, x_{ti} for $1 \leq i \leq r$, and sinks v_{fj}, v_{tj} for $1 \leq j \leq s$. To simplify notation, let $x_i(\mathit{false}) := x_{fi}$ and $x_i(\mathit{true}) := x_{ti}$, etc. Then the following holds: for any input $(b_1, b_2, \ldots, b_r)$, B returns $(c_1, c_2, \ldots, c_s)$ iff there exists a run for $\mathcal{G}$ such that each of the process nodes $x_1(b_1), \ldots, x_r(b_r)$ and $v_1(c_1), \ldots, v_s(c_s)$ has one execution instance, and none of the complementary nodes $x_1(\neg b_1), \ldots, x_r(\neg b_r)$, $v_1(\neg c_1), \ldots, v_s(\neg c_s)$ has any execution instance. As an example see Figure 21.

Now assume that B is a given input circuit of the SUCCINCT-3SAT problem that codes a Boolean 3CNF formula $\mathcal{F}$ as specified above. We may assume that $\mathcal{F}$ has exactly $m = 2^d$ clauses. Then there will be $n = 2^{d-1}$ variables.

The first step of our reduction is to construct four circuits: A, A_1, A_2, and A_3. The first circuit verifies the syntax of the formula encoded by B. Speaking more formally, A with the input j returns 1 if and only if the given circuit B returns j on exactly three inputs $\boldsymbol{(k,\ B(2, j, k'),\ k'')}$ for all $k \in \{0, 1\}$ and $k', k''\{1, 2, 3\}$. Otherwise A returns 0.

For $k = 1, 2, 3$, circuits A_k, with the input $\boldsymbol{(j, i)}$, are defined as follows. A_k returns $\mathbf{(1, 1)}$ if $B(2, j, k) = i$, $B(1, i, k') = j$ for some k', and $B(0, i, k'') \neq j$ for any k'', with $k', k'' \in \{1, 2, 3\}$, that means x_i (but not $\neg x_i$) is the k-th literal of the j-th clause. A_k returns $\mathbf{(1, 0)}$ if $B(2, j, k) = i$, $B(0, i, k') = j$ for some k', and $B(1, i, k'') \neq j$ for any k'', in other words $\neg x_i$ (but not x_i) is the k-th literal of the j-th clause. If both literals x_i and $\neg x_i$ appear in the j-th clause, i.e. $B(1, i, k')$ and $B(0, i, k'')$ are equal to j for some k', k'', then there are also at least two values $k_1, k_2 \in \{1, 2, 3\}$ such that $B(2, j, k_1) = B(2, j, k_2) = i$. In this case A_k returns $\mathbf{(1, 0)}$ if $k = \min\{\ell \in \{1, 2, 3\} \mid B(2, j, \ell) = i\}$ and A_k returns $\mathbf{(1, 1)}$ for all other $k \in \{\ell \in \{1, 2, 3\} \mid B(2, j, \ell) = i\}$. Finally, in any other case, A_k returns $\mathbf{(0, 0)}$. Obviously, these circuits can be easily constructed from given B. Let $\mathcal{A}, \mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$ be the DPGs which denote the encode circuits A, A_1, A_2, A_3. We will call $\mathcal{A}$ the **syntax-verifier**, and $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$ the **index-verifier** graphs.

According to our encoding method, $\mathcal{A}$ has $2d$ sources, and each graph $\mathcal{A}_k$ has $2d + 2(d-1)$ sources, that encode integers j and i, with $0 \leq j \leq m-1$ and $0 \leq i \leq n-1$. Let us call the sources of $\mathcal{A}$ $\mathbf{j_{t1}}, \mathbf{j_{f1}}, \mathbf{j_{t2}}, \mathbf{j_{f2}}, \ldots, \mathbf{j_{t\mathit{d}}}, \mathbf{j_{f\mathit{d}}}$. They encode integer j as follows: If the ℓ-th digit of the binary representation of j is 1 then in a run for $\mathcal{A}$ which represent j, the node $\mathbf{j_{t\ell}}$ has exactly one execution instance, while $\mathbf{j_{f\ell}}$ has no execution at all. If the ℓ-th digit of j is 0 then $\mathbf{j_{f\ell}}$ has exactly one execution instance and $\mathbf{j_{t\ell}}$ has none. This unique instance of either $\mathbf{j_{t\ell}}$ or $\mathbf{j_{f\ell}}$ will be denoted by $t(j, \ell)$.

Similarly, we denote the sources of $\mathcal{A}_k$ by $\mathbf{j_{t1}}, \mathbf{j_{f1}}, \mathbf{j_{t2}}, \mathbf{j_{f2}}, \ldots, \mathbf{j_{t\mathit{d}}}, \mathbf{j_{f\mathit{d}}}$ and $\mathbf{i_{t1}}, \mathbf{i_{f1}}, \ldots, \mathbf{i_{t\mathit{d}-1}}, \mathbf{i_{f\mathit{d}-1}}$, and the executions of these nodes represent integers j and i as above with instances $t(j, \ell)$, resp. $t(i, \ell)$. Note also, that according to the construction method, $\mathcal{A}$ has two sink nodes and each of $\mathcal{A}_k$ has four. Let the number $\boldsymbol{\alpha} := |\mathcal{A}| + |\mathcal{A}_1| + |\mathcal{A}_2| + |\mathcal{A}_3|$ count the number of nodes of these graphs.

To generate the representation of the input values i and j we will use the line DPG, the chain DPG, and the counting graph as defined on page 19 to 19. For some technical reasons let $O(p_0) = \mathsf{ALT}$ where p_0 denotes the since of the counting graph $\mathcal{C}_{h,\ell}$. Furthermore, we address $\mathcal{C}_{h,\ell}$ by a parameter $\alpha := \ell - 1$ and its height h. An example of a counting graph with $h = 2$ and $\alpha = 2$ is given in Fig. 22.

As special incarnation, define the **clause-index graph** as a counting graph of height d and the **variable-index graph** as a counting graph of height $d-1$ (recall that $m = 2^d$). For the clause-index graph let us denote the sink of the main chain by $\mathbf{C}$ and the remaining sinks by $\mathbf{j_{t\ell}}, \mathbf{j_{f\ell}}$, with $1 \leq \ell \leq d$. More precisely, let $\mathbf{j_{t\ell}}$ denote the sink of the right chain and $\mathbf{j_{f\ell}}$ the sink of the left chain of length $\ell - 1$. An execution instance of the sink of the main chain $\mathbf{C}$ of the clause-index graph represents the clause index of which is represented by the execution pattern of the sinks of the other chains (for each ℓ only one sink of the two chains of length ℓ will be executed, either the left or the right one). Similarly, we denote by $\mathbf{X}$ the sink of the main chain of the variable-index graph, and for $\ell = 1, \ldots, d-1$ by $\mathbf{i_{t\ell}}$ (resp. $\mathbf{i_{f\ell}}$) the sink of the right (resp. left) chain of length $\ell - 1$.

Finally, we introduce the **clause-index-duplicator graph** and the **variable-index-duplicator graph** that are slightly modified variants of the corresponding counting graphs. Let $\boldsymbol{R_k}$ be a DPG consisting of nodes $u, p_1, \ldots, p_k, v$ and edges $(u, p_1), \ldots, (u, p_k)$ and $(p_1, v), \ldots, (p_k, v)$. The output mode of u is PAR, and the output modes of the nodes $p_1, \ldots, p_k, v$ is ALT. Further we choose $I(x) = \mathsf{ALT}$ for all nodes of R_k. The clause-index-duplicator graph is a clause-index graph in which $2d$ copies of R_3 are added: each of $\mathbf{j_{t1}}, \mathbf{j_{f1}}, \ldots, \mathbf{j_{t\mathit{d}}}, \mathbf{j_{f\mathit{d}}}$ is connected to the source of its own copy of R_3. In this graph rename the vertices by simply "shifting down" the names $\mathbf{j_{t\ell}}$ and $\mathbf{j_{f\ell}}$ to the sinks of the copies of the R_3 graph.

A variable-index-duplicator is obtained in a similar way by connecting each node $\mathbf{i_{t1}}, \mathbf{i_{f1}}, \ldots, \mathbf{i_{t\mathit{d}-1}}, \mathbf{i_{f\mathit{d}-1}}$ to one copy of the R_6 graph. In the graph obtained this way we also "shift down" the names $\mathbf{i_{t1}}, \mathbf{i_{f1}}, \ldots,$ $\mathbf{i_{t\mathit{d}-1}}, \mathbf{i_{f\mathit{d}-1}}$ of the old to the new sinks.

The total construction consists of two disjoint graphs made up of the subgraphs described above. The first one $\mathcal{G}_I$ will be responsible for checking whether the input circuit B generates a Boolean formula according to the syntax of SUCCINCT-3SAT. The second graph $\mathcal{G}_{II}$ checks whether the encoded formula is satisfiable.

Part I. To construct a dynamic process graph $\mathcal{G}_I$ that verifies the syntax of SUCCINCT-3SAT we will use a clause-index graph, a syntax-verifier $\mathcal{A}$, and a line graph of length β, where $\beta \geq 1$ is a parameter of this part (which will be fixed later depending on the size of $\mathcal{G}_{II}$ the graph of the second part). Connecting these subgraphs we draw edges from the nodes $\mathbf{j_{t1}}, \mathbf{j_{f1}}, \ldots, \mathbf{j_{t\mathit{d}}}, \mathbf{j_{f\mathit{d}}}$ of the clause-index graph to the corresponding sources in $\mathcal{A}$. Recall that $\mathcal{A}$ has two sink nodes – call them $\mathbf{b_f}$ and $\mathbf{b_t}$ – that encode the output bit of the circuit A. This bit equals 1 for input j iff circuit B correctly encodes the j-th clause.

To complete the construction of $\mathcal{G}_I$ we draw one edge from $\mathbf{b_t}$ to the source and one from $\mathbf{C}$ to the sink of the line graph. The sink of the line graph will be denoted by $\mathbf{r}$ (see Fig. 23).

To guarantee a correct encoding of the integers $j \in \{0, \ldots, m-1\}$ we like all tasks which correspond to this integer, i.e. $t(j, \ell)$, to be executed by the same processor. These execution instances $\mathbf{j_{t1}}, \mathbf{j_{f1}}, \ldots, \mathbf{j_{t\mathit{d}}}, \mathbf{j_{f\mathit{d}}}$ can be forced appropriately by a suitable communication delay and deadline. Defining

$$\boldsymbol{D_c} \;:=\; 1 + d \cdot (\alpha + 2) + \sum_{\ell=0}^{d-1} (2\ell + 1) \qquad \text{and} \qquad \boldsymbol{T_1^*(\beta)} \;:=\; D_c + |\mathcal{A}| + \beta.$$

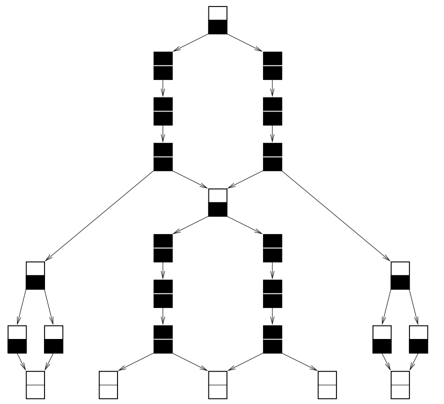

Figure 22: *The counting graph for $h = 2$ and $\alpha = 2$.*

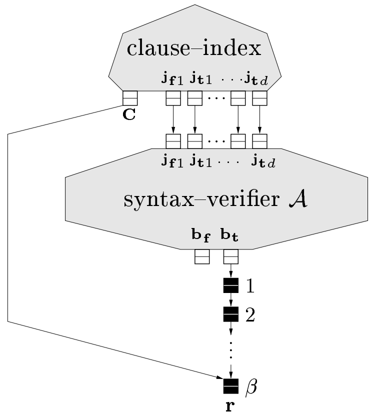

Figure 23: *DPG $\mathcal{G}_I$ that meets the deadline iff B generates a correct encoding.*

for any integer β we can prove the following relation.

Lemma 19 *For $\beta \geq 1$ and any communication delay $\delta_1 \geq D_c + |\mathcal{A}|$, $\mathcal{G}_I$ can be scheduled with deadline $T_1^*(\beta)$ iff B encodes a 3-CNF formula $\mathcal{F}$, where each literal appears exactly three times.*

Proof. Assume first that there exists a schedule S with communication delay δ_1 and deadline $T_1^*(\beta)$ for $\mathcal{G}_I$. Let $H_{\mathcal{G}_I} = (W, F)$ be a run for $\mathcal{G}_I$ such that S schedules $H_{\mathcal{G}_I}$. Then we show that the structure of $H_{\mathcal{G}_I}$ enforces that for all $j \in \{0, \dots, m-1\}$, circuit A with input j returns 1. This means that B encodes expression $\mathcal{F}$ as stated inthe Lemma.

Observe that the characteristic vector of $H_{\mathcal{G}_I}$ has the following property: $|W(\mathbf{C})| = |W(\mathbf{b_t})| = |W(\mathbf{r})| = m$, $W(\mathbf{b_f}) = \emptyset$ and for the sinks $\mathbf{j_{f}}_\ell$,$\mathbf{j_{t}}_\ell$ of the clause-index graph, $|W(\mathbf{j_{f}}_\ell)| = |W(\mathbf{j_{t}}_\ell)| = m/2$, for any $\ell \in \{1, \dots, d\}$. An interesting observation is that this property holds not only for the particular run $H_{\mathcal{G}_I}$ but also for *any* run of $\mathcal{G}_I$. Moreover, this property means that A returns 1 for any input from a multiset of integers $j_p = j_{p,1} j_{p,2} \dots j_{p,d}$ in binary, where $p = 0, 1, \dots, m-1$ and for any $\ell \in \{1, \dots, d\}$ the following holds: $\sum_{p=0}^{m-1} j_{p,\ell} = m/2$. A peculiarity of $H_{\mathcal{G}_I}$ causing that the multiset equals just to $\{0, \dots, m-1\}$ is how the execution nodes for sinks $\mathbf{j_{f1}}$,$\mathbf{j_{t1}}, \dots$,$\mathbf{j_{f}}_d$,$\mathbf{j_{t}}_d$ of the clause-index graph are connected to the executions for sources $\mathbf{j_{f1}}$,$\mathbf{j_{t1}}, \dots$,$\mathbf{j_{f}}_d$,$\mathbf{j_{t}}_d$ of syntax-verifier $\mathcal{A}$. Below we analyze these connections in detail.

Denote the execution nodes for $\mathbf{C}$ and $\mathbf{r}$ by $\mathbf{C}(0), \dots, \mathbf{C}(m-1), \mathbf{r}(0), \dots, \mathbf{r}(m-1)$, respectively. Now let j be an integer, with $0 \leq j \leq m-1$. Then consider a subgraph of $H_{\mathcal{G}_I}$ which is induced by $\mathbf{r}(j)$ and all its predecessors. Partition the vertices of this subgraph into $W_1(j)$ – a set of execution nodes for the clause-index – and $W_2(j)$ – executions for the line and syntax-verifier $\mathcal{A}$.

Because of the value of communication delay δ_1, node $\mathbf{r}(j)$ and all its predecessors have to be executed on the same processor. Hence the following condition must hold

$$|W_1(j)| + |W_2(j)| \ \leq \ T_1^*(\beta) = D_c + |\mathcal{A}| + \beta \ . \tag{3}$$

Any set $W_1(j)$ must contain an execution of $\mathbf{C}$, let us say this is $\mathbf{C}(j)$, together with executions of its predecessors. Denote the $d \cdot (\alpha + 2) + 1$ nodes as follows:

(a) $u_d, v_{d,1}, \dots, v_{d,\alpha}, w_d, \dots, u_1, v_{1,1}, \dots, v_{1,\alpha}, w_1, u_0 = \mathbf{C}(j)$,

where u_ℓ is an execution for p_ℓ, $v_{\ell,t}$ is an execution of the t-th node in the ℓ-th line, and w_ℓ is the execution either of $l_{\ell,\alpha+1}$ or of $r_{\ell,\alpha+1}$. We can assume such an indexing of executions for $\mathbf{C}$ (and hence also for $\mathbf{r}$) that if the ℓ-th bit of integer j is 0 then w_ℓ is the execution of $l_{\ell,\alpha+1}$ and otherwise w_ℓ is the execution for $r_{\ell,\alpha+1}$. Next, $W_1(j)$ must contain also executions for $\mathbf{j}$ nodes and their predecessors. From the construction of $\mathcal{A}$ it follows that for any $\ell \in \{1, \dots, d\}$ either an execution for node $\mathbf{j_{f}}_\ell$ or for $\mathbf{j_{t}}_\ell$ belongs to $W_1(j)$. Now observe that $W_1(j)$ has the minimal number of elements if and only if it contains nodes (a) and for any $\ell \in \{1, \dots, d\}$ either:

(b0) execution of $\mathbf{j_{f\ell}}$ and all its $2(\ell-1)$ predecessors from the left chain of the length $\ell-1$ if ℓ-th bit of j is 0, or

(b1) execution of $\mathbf{j_{t\ell}}$ and all its $2(\ell-1)$ predecessors from the right chain of the length $\ell-1$ if ℓ-th bit of j is 1.

Obviously, the minimal value for $|W_1(j)|$ is D_c. Hence for such $W_1(j)$ and for any $W_2(j)$ condition (3) holds since $|W_2(j)| \leq |\mathcal{A}| + \beta$. On the other hand, any $W_1(j)$ which differs from that one described above has more nodes than $D_c + \alpha \geq D_c + |\mathcal{A}|$ and for any $W_2(j)$, condition (3) does not hold since $|W_2(j)| \geq \beta + d + 1$. Therefore, if $\mathcal{G}_I$ can be scheduled with deadline $T_1^*(\beta)$ then for any $j \in \{0, \dots, m-1\}$, the set $W_1(j)$ contains nodes defined in (a), and for $\ell \in \{1, \dots, d\}$ either nodes described in (b0) or in (b1). This means however, that the executions for nodes $\mathbf{j}$ in $W_1(j)$ encode integer j what implies that for any $j \in \{0, \dots, m-1\}$ circuit A with input j returns 1 (remember that).

Assume now that B encodes expression $\mathcal{F}$ as stated in the Lemma. As we have claimed previously, this means that for any $j \in \{0, \dots, m-1\}$ we have $A(j) = 1$, i.e. that for a run $H_{\mathcal{A}}$ of $\mathcal{A}$ if the executions of $\mathbf{j}$ encode integer j then $H_{\mathcal{A}}$ has an execution for $\mathbf{b_t}$, and hence for $\mathbf{r}$. And this is crucial for a run of $\mathcal{G}_I$ because then a run exists at all (any run of $\mathcal{G}_I$ has exactly m executions for $\mathbf{C}$ and each execution of $\mathbf{C}$ needs a separate execution for $\mathbf{r}$). Now among all runs of the given graph $\mathcal{G}_I$ one should choose such a graph where for any node $\mathbf{r}(j)$ the set $W_1(j)$ has D_c elements. One can easy show that such a graph exists. Obviously such graph can be scheduled with communication delay δ_1 and deadline $T_1^*(\beta)$. This completes the proof. ∎

Part II. The construction of the essential DPG $\mathcal{G}_{II}$ is technically more involved than that for the syntax checker. A clause-index-duplicator and a variable-index-duplicator graph will be used. Moreover, we will add the DPGs $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$, and some auxiliary nodes.

The construction starts by drawing edges between the clause-index-duplicator and the $\mathcal{A}_k$ graphs in a similar way as previously, i.e. connecting each of the nodes $\mathbf{j_{t1}}, \mathbf{j_{f1}}, \dots, \mathbf{j_{t\mathit{d}}}, \mathbf{j_{f\mathit{d}}}$ with its counterparts in $\mathcal{A}_1, \mathcal{A}_2$ and $\mathcal{A}_3$. We connect also each of the nodes $\mathbf{i_{t1}}, \mathbf{i_{f1}}, \dots, \mathbf{i_{t\mathit{d}-1}}, \mathbf{i_{f\mathit{d}-1}}$ of the variable-index-duplicator with appropriate $\mathbf{i_{t\ell}}, \mathbf{i_{f\ell}}$ nodes of $\mathcal{A}_1, \mathcal{A}_2$ and $\mathcal{A}_3$. Recall that $\mathcal{A}_k$ has four sink nodes:

- the first pair, call them $\mathbf{a_f}$ and $\mathbf{a_t}$, encode the first output bit of the circuit A_k that equals to 1 if for the input (j, i), x_i or $\neg x_i$ is the k-th literal of the j-th clause;
- the second pair of $\mathcal{A}_k$, let us call them $\mathbf{b_f}$ and $\mathbf{b_t}$, encode the second output bit of A_k that equals to 1 if x_i is the k-th literal of the j-th clause and 0 elsewhere.

The following fragment will be responsible for assigning *true* or *false* to the particular variable. Let us introduce two new nodes $\mathbf{T}$ and $\mathbf{F}$ which are connected to $\mathbf{X}$ via $(\mathbf{X}, \mathbf{T})$ and $(\mathbf{X}, \mathbf{F})$. Remember that the output mode of $\mathbf{X}$ is ALT which enforces that for each execution instance of $\mathbf{X}$ there exists exactly one edge connecting this task of $\mathbf{X}$ either with an execution of $\mathbf{T}$ or with an execution of $\mathbf{F}$. This guarantees that to each variable x exactly one value – either *true* or *false* – is assigned. To transfer this value to all literals x and $\neg x$ that appear in the formula we define $O(\mathbf{T}) = O(\mathbf{F}) := \mathsf{PAR}$ and connect these nodes to the sources of four copies of the R_3 graph, two for each. One of the sinks of the two R_3 graphs connected with $\mathbf{T}$ will be called $\mathbf{x_t}$, and the other one $\neg\mathbf{x_f}$. Analogously call the sinks of the R_3 graphs connected with $\mathbf{F}$ $\neg\mathbf{x_t}$ and $\mathbf{x_f}$. The output mode of these four sinks is ALT.

Next, for each graph $\mathcal{A}_k$, with $1 \leq k \leq 3$, new nodes $\mathbf{v}, \neg\mathbf{v}, \mathbf{v_t}, \mathbf{v_f}, \neg\mathbf{v_t}, \neg\mathbf{v_f}, \mathbf{r}, \mathbf{s}$ with input mode PAR are introduced, and a node $\mathbf{val}$ with input mode ALT. The output mode of all these nodes is ALT. We connect these nodes by the edges $(\mathbf{a_t}, \neg\mathbf{v})$, $(\mathbf{a_t}, \mathbf{v})$, $(\mathbf{b_f}, \neg\mathbf{v})$, $(\mathbf{b_t}, \mathbf{v})$ and $(\mathbf{v}, \mathbf{v_f})$, $(\mathbf{x_f}, \mathbf{v_f})$, $(\mathbf{v}, \mathbf{v_t})$, $(\mathbf{x_t}, \mathbf{v_t})$ and symmetrically $(\neg\mathbf{v}, \neg\mathbf{v_f})$, $(\neg\mathbf{x_f}, \neg\mathbf{v_f})$, $(\neg\mathbf{v}, \neg\mathbf{v_t})$, $(\neg\mathbf{x_t}, \neg\mathbf{v_t})$. Finally, the following edges $(\neg\mathbf{v_t}, \mathbf{val})$, $(\mathbf{v_t}, \mathbf{val})$, $(\mathbf{val}, \mathbf{r})$, $(\mathbf{val}, \mathbf{s})$ are added. The whole construction gets completed by drawing edges from the sink $\mathbf{C}$ of the clause-index graph (remember, the output mode of $\mathbf{C}$ is ALT) to the first, second and third copy of the node $\mathbf{r}$ (see Fig. 24).

The communication delay and the deadline of the final graph are defined as

$$\boldsymbol{\delta} \;:=\; D'_c + D'_v + \max_{1 \leq k \leq 3} \{|\mathcal{A}_k|\} \qquad \text{and} \qquad \boldsymbol{T^*} \;:=\; \delta + 8 \,,$$

where $D'_c \;:=\; D_c + 2d$, $D_v \;:=\; 1 + (d-1) \cdot (\alpha + 2) + \sum_{\ell=0}^{d-2} (2\ell + 1)$ and $D'_v \;:=\; D_v + 2(d-1)$.

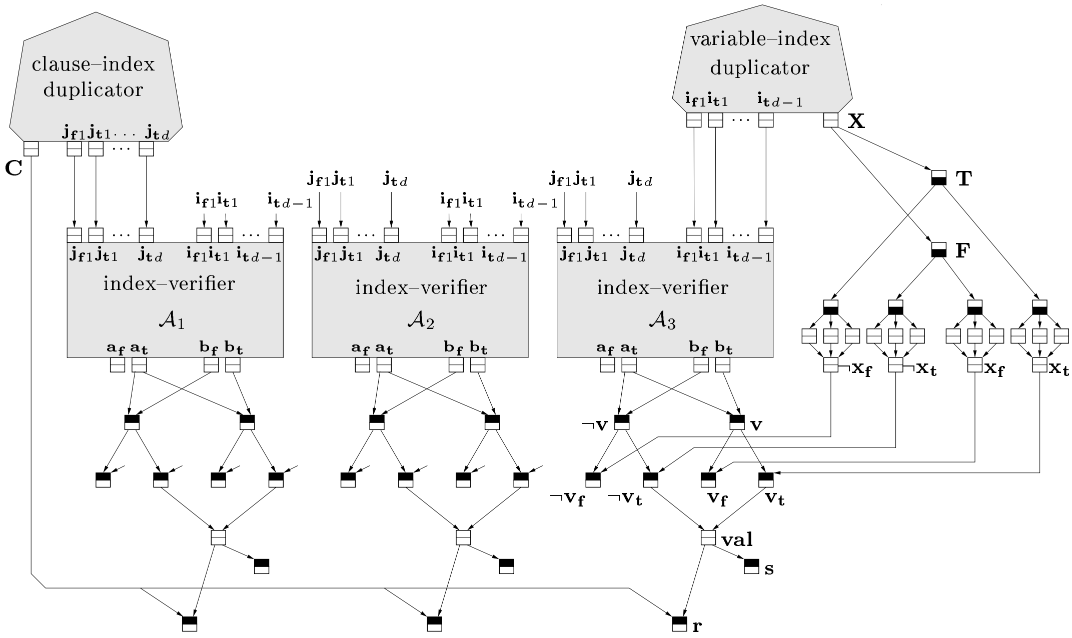

Figure 24: *The construction of $\mathcal{G}_{II}$.*

Lemma 20 *$\mathcal{G}_{II}$ can be scheduled by deadline T^* with respect to communication delay δ iff the formula $\mathcal{F}$ encoded by B is satisfiable.*

Proof. Assume that S is a schedule with parameters δ and T^* and let $H_{\mathcal{G}_{II}} = (W, F)$ be a run for $\mathcal{G}_{II}$ which is scheduled by S. We will show that $H_{\mathcal{G}_{II}}$ enforces that for any $j \in \{0, \ldots, m-1\}$ there is $k_j \in \{1, 2, 3\}$ such that k_j-th literal of the j-th clause is true.

Let us distinguish $\mathbf{v}$ nodes (respectively, $\neg\mathbf{v}, \mathbf{v_t}, \mathbf{v_f}, \ldots$) for different index-verifiers by index $k \in \{1, 2, 3\}$. Therefore, we will use $\mathbf{v}_k, \neg\mathbf{v}_k, \mathbf{v_t}_k, \mathbf{v_f}_k, \ldots$ to indicate that the vertices are successors of $\mathcal{A}_k$.

Now consider the characteristic vector of $H_{\mathcal{G}_{II}}$. Note first that $|W(\mathbf{C})| = m$ and $|W(\mathbf{X})| = n$. Denote the corresponding executions as follows: $\mathbf{C}(0), \ldots, \mathbf{C}(m-1)$, and $\mathbf{X}(0), \ldots, \mathbf{X}(n-1)$. Execution node $\mathbf{C}(j)$ is used to encode the j-th clause and $\mathbf{X}(i)$ the i-th variable. A next observation is that for the sinks $\mathbf{j_{f\ell}}$,$\mathbf{j_{t\ell}}$ of the clause-index duplicator the following holds $|W(\mathbf{j_{f\ell}})| = |W(\mathbf{j_{t\ell}})| = 3m/2$, for any $\ell \in \{1, \ldots, d\}$. The similar condition is true for the sinks of the variable-index duplicator, namely we have $|W(\mathbf{i_{f\ell}})| = |W(\mathbf{i_{t\ell}})| = 6n/2 = 3m/2$, for any $\ell \in \{1, \ldots, d-1\}$. The crucial point of the proof is how these sinks are connected to the execution nodes of the corresponding sources of runs for $\mathcal{A}_k$.

From the evaluation of the numbers of executions for sink nodes of the both duplicator graphs we conclude that for any $k \in \{1, 2, 3\}$ the number of different subgraphs of $H_{\mathcal{G}_{II}}$ which are runs for $\mathcal{A}_k$ is m. Hence for the sinks of $H_{\mathcal{G}_{II}}$ we have that

$$\sum_{k=1}^{3} (|W(\mathbf{r}_k)| + |W(\mathbf{s}_k)| + |W(\neg\mathbf{v_f}_k)| + |W(\mathbf{v_f}_k)|) = 3m \ .$$

Because $|W(\mathbf{C})| = m$ and the output mode of $\mathbf{C}$ is ALT it holds that $|W(\mathbf{r}_1)| + |W(\mathbf{r}_2)| + |W(\mathbf{r}_3)| = m$ for the sinks $\mathbf{r}_k$ and the number of the rest of the sinks in $H_{\mathcal{G}_{II}}$ is $2m$. Let us enumerate the executions for $\mathbf{r}_1, \mathbf{r}_2$, and $\mathbf{r}_3$ by $\mathbf{r}(0), \ldots, \mathbf{r}(m-1)$ in such a way that $\mathbf{r}(j)$ is the node connected with $\mathbf{C}(j)$. We will enumerate the executions of the other nodes in a similar way.

Let $k_j \in \{1, 2, 3\}$ be an integer indicating which node is executed by $\mathbf{r}(j)$: $\mathbf{r}_1, \mathbf{r}_2$, or $\mathbf{r}_3$. We will show that k_j indicates also the literal in the j-th clause of the formula $\mathcal{F}$ which is true.

Consider now schedules for the sinks of $H_{\mathcal{G}_{II}}$ and their predecessors. The crucial ones are schedules for nodes $\mathbf{r}(j)$ and their predecessor hence we will investigate them below in detail. Because of the value of communication delay δ, each $\mathbf{r}(j)$ and all its predecessors have to be executed on the same processor.

Therefore the following condition must hold

$$|W_1(j)| + |W_2(j)| + |W_3(j)| \ \leq\ T^* = D'_c + D'_v + \max_{1\leq k\leq 3}\{|\mathcal{A}_k|\} + 8 \ , \tag{4}$$

where $W_1(j)$ denotes a subset of $\mathbf{r}(j)$'s predecessors which are executions for the clause-index duplicator, $W_2(j)$ denotes $\mathbf{r}(j)$'s predecessors which are executions for the variable-index duplicator, and finally let to $W_3(j)$ belong $\mathbf{r}(j)$ and its remaining predecessors.

Any set $W_1(j)$ must contain $\mathbf{C}(j)$ together with its predecessors. Similarly as in the proof of Lemma 19 denote the nodes as follows:

(a) $u_d, v_{d,1}, \ldots, v_{d,\alpha}, w_d, \ldots, u_1, v_{1,1}, \ldots, v_{1,\alpha}, w_1, u_0 = \mathbf{C}(j)$,

where u_ℓ is an execution for p_ℓ, $v_{\ell,t}$ is an execution of the t-th node in the ℓ-th line, and w_ℓ is the execution either of $l_{\ell,\alpha+1}$ or of $r_{\ell,\alpha+1}$. Also, assume such an indexing of executions for $\mathbf{C}$ that if the ℓ-th bit of integer j is 0 then w_ℓ is the execution of $l_{\ell,\alpha+1}$ and otherwise w_ℓ is the execution for $r_{\ell,\alpha+1}$. $W_1(j)$ must contain also executions for $\mathbf{j}$ nodes and their predecessors. We know that for any $\ell \in \{1, \ldots, d\}$ either an execution for node $\mathbf{j}_{\mathbf{f}\ell}$ or for $\mathbf{j}_{\mathbf{t}\ell}$ belongs to $W_1(j)$. Now observe that $W_1(j)$ has the minimal number of elements if and only if it contains nodes (a) and for any $\ell \in \{1, \ldots, d\}$ either:

(b0) execution of $\mathbf{j}_{\mathbf{f}\ell}$ and all its $2 + 2(\ell - 1)$ predecessors from the graph R_3 and the left chain of the length $\ell - 1$ if ℓ-th bit of j is 0, or

(b1) execution of $\mathbf{j}_{\mathbf{t}\ell}$ and all its $2 + 2(\ell - 1)$ predecessors from the graph R_3 and the right chain of the length $\ell - 1$ if ℓ-th bit of j is 1.

The minimal value for $|W_1(j)|$ is D'_c.

Similarly, we analyze the minimal value for $|W_2(j)|$. Obviously $W_2(j)$ contains $\mathbf{X}(i)$, for some $i \in \{0, \ldots, n-1\}$, together with its predecessors. Let

(a') $u_{d-1}, v_{d-1,1}, \ldots, v_{d-1,\alpha}, w_{d-1}, \ldots, u_1, v_{1,1}, \ldots, v_{1,\alpha}, w_1, u_0 = \mathbf{X}(i)$,

denote these nodes; the meaning for u, v, and w is analogous as in the case of $W_1(j)$. As previously, we can assume such indexing for executions of $\mathbf{X}$ that if the ℓ-th bit of integer i is 0 then w_ℓ is the execution of $l_{\ell,\alpha+1}$ and otherwise w_ℓ is the execution for $r_{\ell,\alpha+1}$. $W_2(j)$ contains also executions for $\mathbf{i}$ and their predecessors. $W_2(j)$ has the minimal number of elements if and only if it contains nodes (a') and for any $\ell \in \{1, \ldots, d-1\}$ either:

(b0') execution of $\mathbf{i}_{\mathbf{f}\ell}$ and all its $2 + 2(\ell - 1)$ predecessors from the graph R_6 and the left chain of the length $\ell - 1$ if ℓ-th bit of j is 0, or

(b1') execution of $\mathbf{j}_{\mathbf{t}\ell}$ and all its $2 + 2(\ell - 1)$ predecessors from the graph R_6 and the right chain of the length $\ell - 1$ if ℓ-th bit of j is 1.

The minimal value for $|W_2(j)|$ is D'_v.

Consider now the third set – $W_3(j)$. It contains $\mathbf{r}(j)$, an execution for $\mathbf{val}_{j_k}$, and vertices of a subgraph of $H_{\mathcal{G}_{II}}$ which is a run for $\mathcal{A}_{k_j}$. Moreover it contains either executions for $\neg\mathbf{v}_{\mathbf{t}k_j}, \neg\mathbf{v}_{k_j}, \neg\mathbf{x}_\mathbf{t}$, its two predecessors from R_3 and $\mathbf{F}$ or executions for $\mathbf{v}_{\mathbf{t}k_j}, \mathbf{v}_{k_j}, \mathbf{x}_\mathbf{t}$, its two predecessors from R_3 and $\mathbf{T}$. In both cases $W_3(j)$ has no more than $\max_{1\leq k\leq 3}\{|\mathcal{A}_k|\} + 8$ elements. Hence for such *minimal* $W_1(j), W_2(j)$, and $W_3(j)$ as described above condition (4) holds. On the other hand, any set $W_1(j)$ which differs from that one described above has more than $D'_c + \alpha \geq D'_c + \max_{1\leq k\leq 3}\{|\mathcal{A}_k|\}$ elements. Similarly, a set $W_2(j)$ which does not fulfills conditions as above has more than $D'_v + \alpha \geq D'_v + \max_{1\leq k\leq 3}\{|\mathcal{A}_k|\}$ elements. Therefore, condition (4) does not hold since $|W_3(j)| \geq 2d + 8$.

We conclude, that $\mathcal{G}_{II}$ can be scheduled with deadline T^* if and only if $W_1(j), W_2(j)$, and $W_3(j)$ are minimal. This means that

- There exists a run such that for any j exists an index i such that $\mathbf{r}(j) \in W_3(j)$ and i is represented by $W_2(j)$.

- This implies that there exists a run such that for any j exists an index i such that i is represented by $W_2(j)$ and $\mathbf{F}(i) \in W_3(j) \Rightarrow \mathbf{b}_\mathbf{f}(i) \in W_3(j)$ as well as $\mathbf{T}(i) \in W_3(j) \Rightarrow \mathbf{b}_\mathbf{t}(i) \in W_3(j)$.

- Hence, there exists an assignment for $x_1, \dots, x_n$ such that for any j exists an index i with $x_i = 0 \Rightarrow \overline{x}_i \in C_j$ and $x_i = 1 \Rightarrow x_i \in C_j$.

We can summarize, that $\mathcal{G}_{II}$ can be scheduled with deadline T^* if and only if there exists a satisfying assignment for the formula given by B, what completes the proof. ∎

To complete the proof of Theorem 11 let $\delta_1 := \delta$ and $\beta := T^* - (D_c + |\mathcal{A}| - 1)$. Note that δ_1 fulfills the inequality as required by Lemma 19. Then Lemma 19 and 20 imply

Lemma 21 *The union of $\mathcal{G}_I$ and $\mathcal{G}_{II}$ can be scheduled within the deadline T^* given communication delay δ iff the corresponding* SUCCINCT-3SAT *problem has a positive solution.*

∎

5 Conclusion

We have defined a dynamic model for scheduling process graphs. These dynamic process graphs allow a compact representation of typical distributed programs written in OCCAM or Ada style. We are not aware of another framework with a similar expressive power. Restricting the input and output mode of nodes different degrees of concurrency can be modeled.

With respect to the degree of concurrency we have analyzed how difficult it is to decide whether a dynamic process graph can be executed and to construct optimal schedules. An almost complete exact characterization could be given. Only for the scheduling problem with output mode restricted to PAR there remains a gap.

input mode	output mode	complexity	problem for reduction
ALT	ALT	$\mathcal{NL}$-complete	Directed Graph Reachability Problem
ALT	PAR	$\mathcal{NL}$-complete	complement of DGR Problem
ALT	ALT, PAR	$\mathcal{P}$-complete	Circuit Value Problem
PAR	arbitrary	$\mathcal{NP}$-complete	Scheduling Problem with Communication Delay for in-Trees
ALT, PAR	ALT	$\mathcal{NP}$-complete	
ALT, PAR	PAR	$\mathcal{BH}_2$-hard	SAT-UNSAT Problem
unrestricted		$\mathcal{NEXPTIME}$-complete	SUCCINCT-3SAT

Figure 25: *The different variants of scheduling DPGs*

For the general case, we have shown an exponential complexity jump when scheduling process graphs in the dynamic setting. This implies that our compact dynamic representation is quite effective.

Finally, note that although constructing optimal schedules of standard graphs cannot be done efficiently, at least the problem can be approximated to a certain factor by simple algorithmic methods. This even holds when one takes communication delays into account.

6 Acknowledgements

We thank Eric Allender for pointing out the complexity of execution problem for DPGs restricted to PAR output mode. The first author thank also Faith Fich for helpful conversations.

References

[1] E. Allender, M. Ogihara, *Relationships among PL, #L, and the determinant,* RAIRO - Theoretical Informatics and Applications Vol. 30, 1996, 1-21.

[2] A. Burns, *Programming in Occam 2,* Addison-Wesley Publishing Company, 1988.

[3] M. Dyer and A. Frieze, *Planar 3DM Is NP-Complete*, J. Algorithms 7, 1986, 174-184.

[4] H. El-Rewini and H. H. Ali, *Static Scheduling of Conditional Branches in Parallel Programs,* J. Par. Distrib. Comput. 24, 1995, 41-54.

[5] H. Galperin, A. Wigderson, Succinct Representations of Graphs, Information and Control, 56, 1983, 183-198.

[6] M. R. Garey and D. S. Johnson, *Computers and Intractability, A Guide To the Theory of NP-Completeness,* Freeman 1979.

[7] R. Greenlaw, H. J. Hoover and W. L. Ruzzo, *Limits to Parallel Computation – P-Completeness Theory,* Oxford University Press, 1994.

[8] S. Ha, E. Lee, *Compile-time Scheduling and Assignment of Data-flow Program Graphs with Data-dependent Iteration*, IEEE Trans. Computers 40, 1991, 1225-1238.

[9] A. Jakoby, M. Liśkiewicz, R. Reischuk, *Scheduling Dynamic Graphs,* in Proc. 16th International Symposium on Theoretical Aspects of Computer Science, LNCS 1563, Springer-Verlag, 1999, pp. 383-392.

[10] A. Jakoby and R. Reischuk, *The Complexity of Scheduling Problems with Communication Delay for Trees,* Proc. 3. Scandinavian Workshop on Algorithm Theory SWAT'92, 1992, 165 - 177.

[11] H. Jung, L. Kirousis, P. Spirakis, *Lower Bounds and Efficient Algorithms for Multiprocessor Scheduling of DAGs with Communication Delays,* Proc. 1. SPAA, 1989, 254 - 264.

[12] R. Kieckhafer, *Fault-Tolerant Real-Time Task Scheduling in the MAFT Distributed System*, 22. Hawaii Int. Conf. on System Science, 1989, 145-151.

[13] R. Kieckhafer, C. Walter, A. Finn, P. Thambidurai, *The MAFT Architecture For Distributed Fault-Tolerance*, IEEE Trans. Computers, April 1988, 398-405.

[14] T. Lengauer, K. Wagner, *The Correlation between the Complexities of the Nonhierarchical and Hierarchical Versions of Graph Problems,* J. CSS 44, 1992, 63-93.

[15] M. Mahajan and V. Vinay, *A combinatorial algorithm for the determinant,* Proc. ACM-SIAM Symp. on Distrib. Alg., 1997, pp. 730-738.

[16] C. Papadimitriou and M. Yannakakis, *The Complexity of Facets*, Proc. 14. STOC, 1982, 255-260.

[17] C. Papadimitriou and M. Yannakakis, *A Note on Succinct Representations of Graphs*, Information and Control, 71, 1986, 181-185.

[18] C. Papadimitriou and M. Yannakakis, *Towards an Architecture-Independent Analysis of Parallel Algorithms,* Proc. 20. STOC, 1988, 510-513, see also SIAM J. Comput. 19, 1990, 322-328.

[19] B. Veltman, *Multiprocessor Scheduling with Communication Delays,* Ph.D. Thesis, University of Technology Eindhoven, Department of Computer Science, Eindhoven, The Netherlands, 1993.