Réécriture de workflows scientifiques et provenance

Sarah Cohen-Boulakia, Christine Froidevaux, Jiuqiang Chen

Réécriture de workflows scientifiques et provenance

Sarah Cohen-Boulakia*, Christine Froidevaux*, Jiuqiang Chen*,**
* Laboratoire de Recherche en Informatique, CNRS UMR 8623
Université Paris Sud, 91405 Orsay France
Groupe AMIB, INRIA Saclay, France
** School of Information Science and Engineering
Lanzhou University, China

Abstract

Les systèmes de workflow sont nombreux et disposent de modules de gestion de provenance qui collectent les informations relatives aux exécutions (données consommées et produites) permettant d'assurer la reproductibilité d'une expérience. Un grand nombre d'approches s'est développé pour aider à la gestion de ces masses de données de provenance. Un certain nombre de ces approches ont une bonne complexité parce qu'elles sont dédiées à des structures de workflows série-parallèles. Réécrire un workflow en un workflow série-parallèle permettrait donc de mieux exploiter l'ensemble des outils de provenance existants. Nos contributions sont : (i) introduction de la notion de réécriture de workflow provenance-equivalence, (ii) revue de transformations de graphes, (iii) conception de l'algorithme de réécriture SPFlow préservant la provenance (iv) évaluation de notre approche sur un millier de workflows.
Mots clés : workflows scientifiques; provenance; graphes série-parallèle.

1 Introduction

Les systèmes de gestion des workflows scientifiques (e.g. [1], [2], [3], [4], [5], [6]) sont de plus en plus utilisés pour spécifier et gérer les expériences bioinformatiques. Dans ces systèmes, une expérience est représentée par un workflow dans lequel un grand nombre de tâches bioinformatiques (outils bioinformatiques, accès aux données) s'enchaînent les unes aux autres. Un workflow se représente alors naturellement sous la forme d'un graphe orienté où les sommets sont les tâches bioinformatiques à effectuer et les arcs représentent l'ordre dans lequel les tâches seront exécutées. La plupart des systèmes de workflows considèrent des graphes acycliques orientés : c'est ce que nous choisirons aussi dans cet article. Chaque workflow peut être exécuté plusieurs fois en faisant varier les données prises en entrée ou le paramétrage des outils (sans perte de généralité, les paramètres d'outils sont considérés comme des données). Nous considérerons ici le cas où une exécution se représente à son tour sous forme de graphe qui a la même structure que la spécification, où chaque

sommet représente l'exécution d'un outil et où les arcs sont étiquetés par les données consommées et produites à chaque étape. Aussi, nous considérons le cas de tâches reproductibles : deux exécutions d'une même tâche avec les mêmes données d'entrée fournira toujours le même résultat (en particulier, le comportement d'une tâche ne doit pas dépendre de variables aléatoires).

La Figure 1 propose (a) un exemple de workflow réel issu du système Taverna, (b) sa représentation sous forme de graphe et (c) un exemple de graphe d'exécution de ce workflow. Tout graphe (représentant un workflow ou une exécution) a une source unique s et un puits unique t, tels que tout sommet soit sur un chemin de s à t (accessibilité).

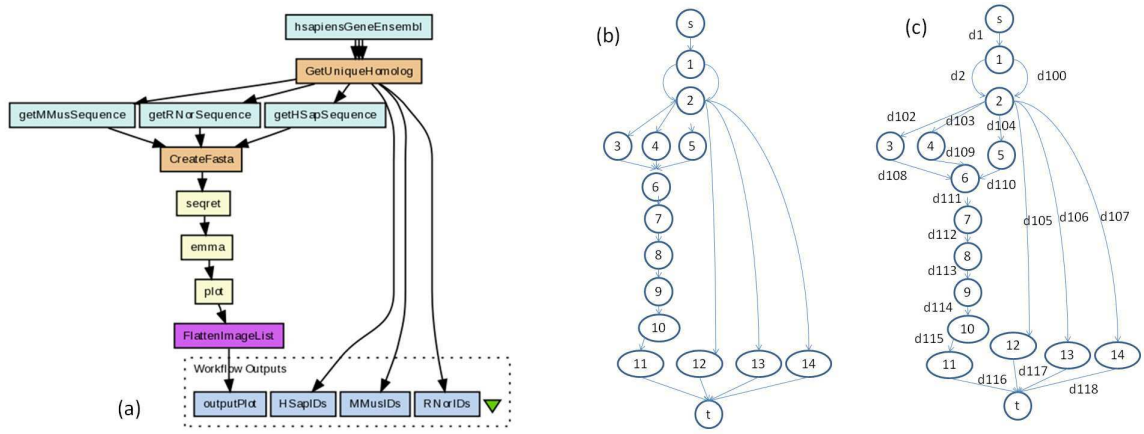

Figure 1: Exemple de workflow simple issu du système Taverna; (b) représentation sous forme de graphe; (c) exemple de graphe d'exécution

Face à la complexité croissante des exécutions de workflows, au besoin de reproductibilité des résultats et au besoin d'associer une qualité aux données obtenues, la gestion de la provenance dans les workflows scientifiques est devenue un domaine de recherche en pleine expansion, en particulier dans la communauté Bases de Données. Un nombre important d'outils d'aide à la gestion des masses de données de provenance ont été conçus : certains proposent d'aider au stockage des données de provenance (réduction au maximum du volume de données nécessaire à la reproduction d'une expérience, indexation), d'autres permettent d'interroger ces données (pour mieux comprendre une différence entre deux exécutions, pour rechercher des motifs dans un ensemble (d'exécutions de) workflows disponibles), d'autres encore permettent de visualiser la provenance ou encore de (re)planifier les exécutions, un domaine particulièrement d'actualité avec l'avènement du *cloud computing*. Ces outils font tous intrinsèquement des opérations complexes sur des structures de graphes (recherche de sous-graphe dans un graphe, comparaison de graphes, ...), qui, si elles sont effectuées sur des graphes acycliques (DAGs) sans autre restriction de structure conduisent à des problèmes NP-difficiles. Au contraire, ces problèmes peuvent être résolus en temps polynomial lorsque certaines restrictions sont imposées aux graphes, comme par exemple qu'ils aient une structure série-parallèle (SP) [7]. Certaines approches de gestion de provenance [8, 9, 10] ont donc choisi de se restreindre à des graphes de workflows ayant des structures SP. Néanmoins, en général, les workflows générés par les systèmes de workflows sont des DAGs de structure quelconque.

Proposer une procédure de réécriture de tout graphe de workflow en un graphe série-parallèle tout en assurant que la provenance du graphe transformé soit la même que dans le graphe original permettrait de mieux exploiter l'ensemble des outils de gestion de provenance existants. C'est l'objectif de cet article.

Nos contributions sont les suivantes : (i) nous introduisons la notion de réécriture de workflow préservant la provenance (section 3), (ii) nous déterminons parmi les stratégies de transformations de graphes existantes celles qui préservent la provenance et proposons de nouvelles stratégies (section 4), (iii) nous introduisons l'algorithme de réécriture de graphes SPFlow (section 5), (iv) nous fournissons des chiffres clés sur les résultats obtenus par notre approche sur un ensemble de workflows scientifiques réels et nous démontrons la faisabilité de notre approche et en particulier montrons que le coût de la réécriture est raisonnable (section 6). Nous discutons nos résultats et concluons en section 7.

2 Structures SP et workflows

Dans cette section, nous introduisons une formalisation des workflows et des exécutions sous forme de graphes et donnons les fondements sur les structures série-parallèle. Nous présentons ensuite trois scénarios motivant l'intérêt de l'utilisation de ces structures pour les workflows scientifiques.

2.1 Workflows série-parallèle

2.1.1 Modèle de Workflow

Intuitivement, une spécification de workflow est un cadre pour des exécutions de workflows, qui spécifie l'ensemble des tâches qui sont exécutées et l'ordre qui doit être respecté entre les différentes exécutions des tâches. Formellement, nous modélisons une spécification de workflow comme un multigraphe orienté étiqueté, dont les sommets représentent les tâches du workflow et les arcs représentent le *flot des données* entre les tâches.

Considérons un multi-graphe acyclique $G = (V, E)$, où V est un ensemble de sommets et E un ensemble d'arcs avec $E \subseteq V \times V$.

On utilise le terme **st-dag** [7] pour désigner un multi-graphe acyclique ayant deux sommets particuliers, s la source, et t le puits. On note $s(G)$ (resp. $t(G)$) la source (resp. le puits) du st-dag G.

Définition 2.1 *[8] Une* ***spécification de workflow*** *est un st-dag $G_{spec} = (V_{spec}, E_{spec})$ dont les sommets sont étiquetés par une fonction $L_{vs} : V_{spec} \to L_{VS}$, où L_{VS} est un ensemble de valeurs de sommets.*

Une spécification de workflow peut être exécutée plusieurs fois. Ainsi, selon les données en entrée du workflow et les assignations de valeurs aux paramètres des tâches, on obtiendra des **exécutions de workflows** différentes. Chaque exécution de workflow exécute les tâches et fait transiter entre elles des données.

Nous donnons ci-dessous la définition formelle d'une exécution de wokflow, qui est adaptée de celle de [8] :

Définition 2.2 *Une **exécution de workflow** $G_{run} = (V_{run}, E_{run})$ est un st-dag dont les sommets et les arcs sont étiquetés, respectivement par une fonction $L_{vr} : V_{run} \rightarrow L_{VR}$, où L_{VR} est un ensemble de valeurs de sommets, et une fonction $L_{er} : E_{run} \rightarrow L_{ER}$, où L_{ER} est un ensemble de valeurs d'arcs. On notera $\tilde{x}$ l'étiquette du sommet x, i.e. $L_{vr}(x) = \tilde{x}$ et d_i l'étiquette de l'arc e_i, i.e. $L_{er}(d_i) = e_i$.*
*Une exécution de workflow $G_{run} = (V_{run}, E_{run})$ est **valide** par rapport à la spécification de workflow $G_{spec} = (V_{spec}, E_{spec})$ si :*
(i) $V_{run} = V_{spec}$, (ii) $E_{run} = E_{spec}$ et (iii) $L_{vr} = L_{vs}$ avec $L_{VR} = L_{VS}$.

On introduit dans la sous-section suivante la notion de graphe série-parallèle.

2.1.2 Graphes série-parallèle

Les graphes série-parallèle ont été introduits par Duffin [11] comme un modèle mathématique pour les réseaux électriques. Duffin a prouvé qu'un st-DAG est série-parallèle si et seulement si il ne contenait pas un sous-graphe homomorphique au graphe "interdit" représenté en Figure 5(a). Des algorithmes pour détecter qu'un graphe est série-parallèle ont été proposés dont l'algorithme de Valdes *et al.* [12] qui est en temps linéaire.

Un graphe **série-parallèle (graphe SP)** est un st-dag G que l'on obtient itérativement de la façon suivante :

1. Graphe SP Basique : G, le graphe qui contient un unique arc entre s et t, est un graphe SP (appelé "**BSP**" pour "basic SP-graph");
2. **Composition Série** : étant donnés G_1 (avec une source s_1 et un puits t_1) et G_2 (avec une source s_2 et un puits t_2) deux graphes SP, G est obtenu en identifiant $s = s_1$, $s_2 = t_1$ et $t = t_2$;
3. **Composition Parallèle** : étant donnés G_1 (avec une source s_1 et un puits t_1) et G_2 (avec une source s_2 et un puits t_2) deux graphes SP, G est obtenu en identifiant $s = s_1 = s_2$ et $t = t_1 = t_2$.

Un st-dag G, qui n'est pas un graphe SP est appelé un graphe **non-SP**.

Déterminer qu'un graphe a ou non une structure SP se fonde sur un ensemble d'opérations de réduction qui ont été introduites dans [7], reprises ci-après, et illustrées en Figure 2. Nous complétons ces définitions en spécifiant les étiquettes des sommets et des arcs. Les arcs introduits par les opérations de réduction sont étiquetés de sorte à garder la trace des sous-graphes qu'ils remplacent.

Définition 2.3 *Soit $G_1 = (V_1, E_1)$ un st-dag dont les sommets et les arcs sont étiquetés, respectivement par une fonction $L_{1vr} : V_1 \rightarrow L_{VR}$, et une fonction $L_{1er} : E_1 \rightarrow L_{ER}$. L'opération élémentaire op transforme G_1 en un st-dag $op(G_1) = G_2 = (V_2, E_2)$, dont les sommets et les arcs sont étiquetés, respectivement par une fonction $L_{2vr} : V_2 \rightarrow L_{VR}$, et une fonction $L_{2er} : E_2 \rightarrow (L_{VR} \cup L_{ER})^*$, où $(L_{VR} \cup L_{ER})^*$ désigne l'ensemble des expressions régulières construites sur $L_{VR} \cup L_{ER}$.*

1. ***Réduction Série*** *Soient u, v, w trois sommets de V_1 tels que $e = (u, v)$ est l'unique arc entrant de v et $f = (v, w)$ est l'unique arc sortant de v. L'opération op de **réduction en série** en v remplace e et f par $g = (u, w)$.*
Le graphe $G_2 = (V_2, E_2)$ obtenu à partir de G_1 est tel que : $V_2 \subset V_1$, L_{2vr} est la restriction de L_{1vr} sur V_2, L_{2er} est identique à L_{1er} sur les arcs communs aux deux graphes, et $L_{2er}(g) = L_{1er}(f) \cdot L_{1vr}(v) \cdot L_{1er}(e)$.

2. ***Réduction Parallèle*** *Soient v et w deux sommets de V_1 reliés par k arcs e_1 ... e_k. L'opération op de* ***réduction en parallèle*** *en v et w remplace les k arcs par un unique arc $g = (v, w)$ et laisse tous les autres arcs inchangés.*
Le graphe $G_2 = (V_2, E_2)$ obtenu à partir de G_1 est tel que : $V_2 = V_1$, $L_{2vr} = L_{1vr}$, L_{2er} identique à L_{1er} sur $E_1 \cap E_2$, et $L_{2er}(g) = L_{1vr}(e_1) + ... + L_{1vr}(e_k)$.

3. ***Réduction de sommet*** *Soit v un sommet de V_1 ayant un unique arc entrant $e = (u, v)$ et ayant k arcs sortants $f_1 = (v, w_1), ..., f_k = (v, w_k)$. L'opération op de* ***réduction de sommet*** *en v remplace les k arcs $\{e, f_1, ..., f_k\}$ par les k nouveaux arcs $\{g_1, ..., g_k\}$ où $g_i = (u, w_i)$, pour $i \in [1, k]$.*
Le graphe $G_2 = (V_2, E_2)$ obtenu à partir de G_1 est tel que : $V_2 \subset V_1$, L_{2vr} est la restriction de L_{1vr} sur V_2, L_{2er} est identique à L_{1er} sur $E_1 \cap E_2$, et $L_{2er}(g_i) = L_{1er}(f_i) \cdot L_{1vr}(v) \cdot L_{1er}(e)$. Le sommet v est appelé ***sommet de réduction****. Le cas où v a un degré sortant égal à 1 et un degré entrant strictement supérieur à 1 est défini de façon analogue (cf. 2(d)).*

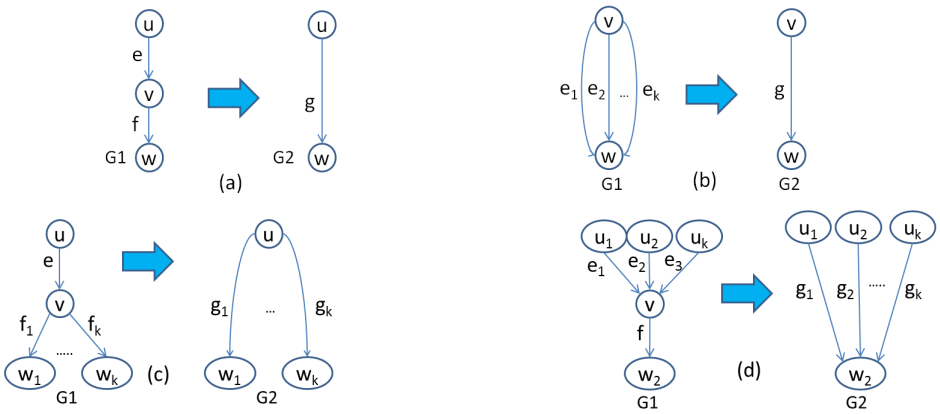

Figure 2: Opérations de réductions : (a) réduction série; (b) réduction parallèle, (c) réduction de sommet fondée sur les sorties; (d) réduction de sommet fondée sur les entrées

Remarque : Si G un st-dag, $op(G)$ est un st-dag de même source et de même puits que G, pour chacune des opérations de réduction ci-dessus.

Définition 2.4 *Soit G un st-dag.*

1. *On appelle* ***MaxRed(G)*** *le graphe obtenu par application successive de toutes les opérations de réductions série et parallèle possibles sur G.*
2. *Un st-dag G est dit* ***maximalement réduit*** *si $G = MaxRed(G)$.*
3. *G est* ***SP*** *si et seulement si $MaxRed(G) = BSP$.*

La Figure 3 décrit les opérations de réductions effectuées depuis le graphe de la Figure 1(b) jusqu'au graphe BSP. Aucune réduction de sommet n'a été utilisée : le graphe initial G_0 est SP.

L'opération de réduction de sommet permet de donner une idée de la *distance* qui sépare un graphe non SP d'une structure SP. Intuitivement plus cette opération

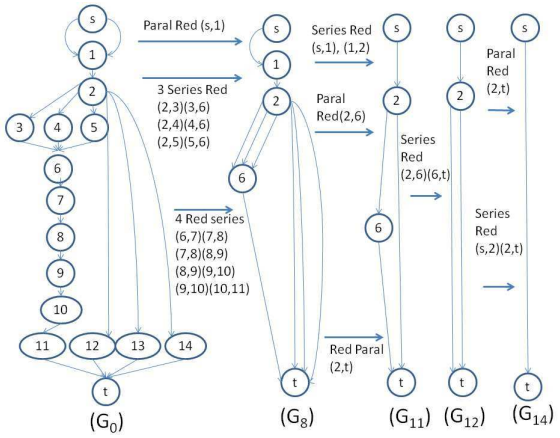

Figure 3: Exemple d'opérations de réduction appliquées successivement à G_0.

est utilisée (*i.e.* plus il y a de sommets de réduction) plus le graphe est loin d'une structure SP.

Dans la suite de cette section, nous présentons trois scénarios pour mettre en évidence l'intérêt de considérer des structures SP dans les workflows scientifiques.

2.2 Scénarios

Dans cette section, nous proposons trois scénarios pour illustrer les avantages à travailler sur des structures SP plutôt que sur des graphes acycliques quelconques.

2.2.1 Conception de Workflows

Alors que les workflows scientifiques ont pour objectif d'aider au partage et à la réutilisation de procédures d'analyses, une étude récente [13] a montré que les auteurs de workflows réutilisaient facilement leurs propres workflow mais très rarement les workflows d'une tierce personne. Une des explications est que la structure de graphe d'un workflow scientifique peut être particulièrement complexe, rendant les principales étapes de l'analyse difficiles à appréhender. Guider les concepteurs de workflows à construire des workflows simples à comprendre est fondamentalement important pour améliorer le partage et la réutilisation de workflows.

Nous pensons que les structures SP peuvent être d'une aide précieuse dans ce contexte. D'abord intuitivement, et d'un point de vue purement visuel, les structures SP sont simples; les graphes SP sont en étage, leurs arcs ne se croisent pas, rendant les phases principales du workflow plus faciles à distinguer. Par exemple, considérons le workflow de la Figure 4(a) dont la structure représentée en Figure 4(b) n'est pas SP. Il est relativement complexe de distinguer visuellement les grandes étapes de ce workflow ou de construire des sous-workflows indépendants. La Figure 4(c) représente le même workflow dans lequel le sommet 2 a été dupliqué en 2' (ce qui revient à proposer d'interroger deux fois le catalogue de données). Le workflow restructuré est SP et déjà plus facile à appréhender alors que sa structure originale n'est pas très éloignée d'un graphe SP puisque seul le sommet 2 est un sommet de réduction.

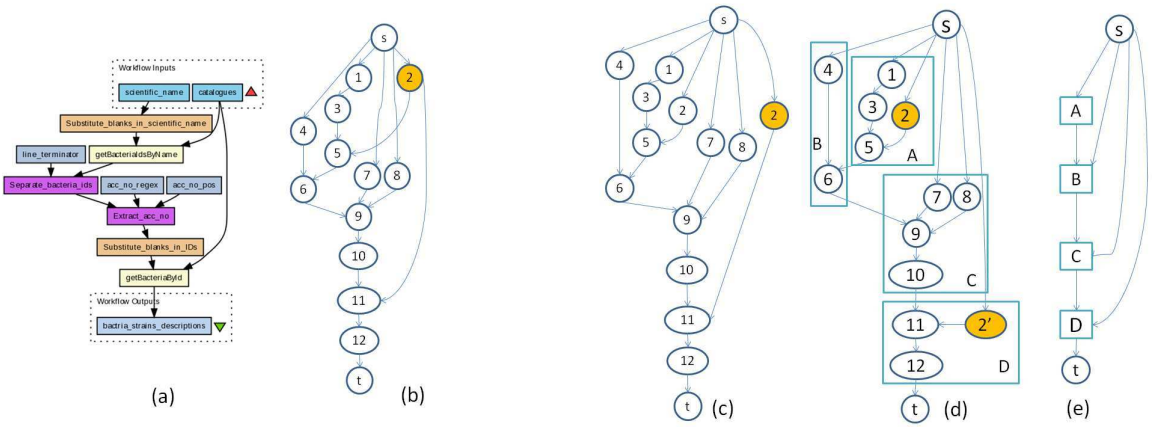

Figure 4: (a) Exemple de workflow simple issu du système Taverna, (b) structure graphe du workflow (non SP); (c) transformation de la structure en graphe SP; (d) proposition de sommets composites; (e) graphe de haut niveau obtenu

De façon plus formelle, il existe des approches pour aider à la conception de sous-workflows dans les workflows. C'est le cas de ZOOM [9] qui construit une vue d'utilisateur : étant données des informations sur les tâches élémentaires dans le workflow qui ont un intérêt pour un utilisateur particulier, une vue d'utilisateur est une représentation concise du workflow qui regroupe ensemble des tâches élémentaires pour créer un petit nombre de tâches composites (ou sous-workflows) telles que (1) chaque tâche composite contient au plus une tâche élémentaire d'intérêt qui devient la tâche centrale du composite; (2) on n'introduit (correction) ni ne supprime (complétude) de dépendance de données entre les tâches d'intérêt. L'objectif est de construire une vue d'utilisateur avec le plus petit nombre de tâches composites. Il a été montré [14] que la solution optimum ne peut pas être atteinte pour des DAGs arbitraires alors qu'un algorithme en temps linéaire existe lorsque le workflow a une structure SP.

Dans l'exemple de la Figure 4, ZOOM peut construire automatiquement les sous-workflows A, B, C et D comme indiqué en Figure 4(d), rendant le workflow plus facile à appréhender et les sous-workflows faciles à réutiliser. Si on regarde ce même workflow en considérant les sous-workflows comme des boîtes noires, l'utilisateur voit alors le workflow de la Figure 4(e). Le même type de construction sur le workflow d'origine imposerait un arc entre A et D, rendant ainsi le partage et la réutilisation des sous workflows moins faciles.

Il est important de bien noter que le workflow choisi ici est simple (12 sommets) et que la structure d'origine n'est pas très éloignée d'une structure SP. L'intérêt d'exploiter les structures SP est d'autant plus grand que la structure initiale du workflow est complexe.

2.2.2 Interrogation de workflows

Une autre façon de concevoir un workflow est de se baser sur des workflows existants. L'utilisateur peut donc être amené à interroger les entrepôts de workflows pour rechercher un workflow ayant une structure particulière ou contenant un motif donné.

Le besoin de pouvoir faire ce type de recherche dans les entrepôts de workflow est exprimé depuis plusieurs années [15, 16, 17] mais n'est aujourd'hui toujours pas pris en compte dans les entrepôts de workflows où les workflows sont interrogeables uniquement par mots clés présents dans leur description textuelle. La raison en est simple : ce type de recherche est directement associé à des problèmes connus pour être NP-difficiles (isomorphisme de sous-graphes) sur des DAGs classiques. Les structures SP présentent un avantage clair : le problème de recherche d'un sous-graphe isomorphe à un graphe se traite en temps polynomial sur de telles structures.

Un autre exemple de type de requêtes faisant appel à des opérations sur les graphes est la recherche de différences entre structures de workflows ayant des points communs. Là encore le problème du calcul de la différence de deux sous-graphes d'un même graphe est NP-difficile dans le cas général et polynomial pour des graphes SP.

2.2.3 Planification de workflows

De façon orthogonale, les structures SP peuvent être aussi particulièrement intéressantes dans le contexte de la planification d'exécutions de workflows. Dans le domaine plus général de la planification de tâches dans les programmes, les structures SP sont exploitées depuis des décennies [18], notamment parce qu'elles ont montré leurs avantages pour l'analyse de programme [19], l'estimation de coût [20], et l'efficacité de la planification [21].

Avec le développement des grilles de calcul et plus récemment du *cloud computing*, la planification de tâches de workflow sur des ressources multiples et distribuées est d'importance croissante. Dans l'exemple simple de la Figure 4 les tâches composites A, B, C, D peuvent être ordonnancées dans cet ordre dans la structure modifiée SP, alors qu'en considérant le graphe d'origine non SP il y a une dépendance en plus à considérer entre les tâches A et D, rendant plus complexe le découpage du workflow pour la planification.

3 Provenance

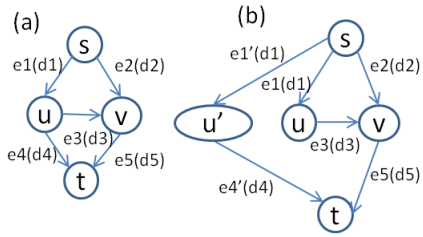

Figure 5: Deux graphes pour illustrer les notions relatives à la provenance

3.1 Modèle de provenance

La notion de provenance d'une donnée produite par une tâche fixée a été introduite dans [22] pour une exécution de workflow. Nous reprenons cette notion et la formalisons ici. Intuitivement, la provenance d'une donnée est la suite ordonnée des tâches

exécutées pour produire cette donnée (c-à-d, les tâches dont elle dépend), et les données d'entrée de ces tâches. En ce sens, notre modèle est naturellement compatible avec des approches suivant OPM (Open Provenance Model[1]). Nous distinguons la provenance immédiate, qui décrit la dernière étape de production d'une donnée et la provenance profonde qui donne la succession des étapes ayant permis de produire la donnée. Rappelons que dans la représentation sous forme de multi-graphe étiqueté d'une exécution de workflow, une donnée produite par une tâche est l'étiquette d'un arc sortant du sommet représentant cette tâche. Formellement, on a la définition suivante :

Définition 3.1 ***Provenance d'une donnée***
Etant donné une exécution de workflow $G_{run} = (V_{run}, E_{run})$ soit :

- *$u \in V_{run}$ qui n'est pas la source de (G_{run}), avec $L_{vr}(u) = \tilde{u}$;*
- *$f \in E_{run}$ un arc sortant de u avec $L_{er}(f) = d$;*
- *$e_i \in E_{run}$, $1 \leq i \leq p$ les arcs entrants de u, avec $L_{er}(e_i) = d_i$.*

*La **Provenance immédiate** de d dans G_{run} (où d est l'étiquette de l'arc f) est définie par la fonction $imProv : E_{run} \rightarrow (L_{VR} \cup L_{ER})^*$, avec :*

$$imProv(f)_{G_{run}} = \tilde{u} \cdot (d_1 + \ldots + d_p) \tag{1}$$

*La **Provenance Profonde** de d dans G_{run} (où d est l'étiquette de l'arc f) est définie comme la suite ordonnée des tâches et des données d'entrée des tâches qui ont été successivement utilisées pour la produire. Elle est formalisée par la fonction $DProv : E_{run} \rightarrow (L_{VR} \cup L_{ER})^*$, qui est définie récursivement comme suit :*

$$DProv(f)_{G_{run}} = \tilde{u} \cdot (d_1 \cdot DProv(e_1)_{G_{run}} + \ldots + d_p \cdot DProv(e_p)_{G_{run}}) \tag{2}$$

Le cas de base est donné lorsque u égal à s, la source de G_{run} (st-dag), avec $f \in E_{run}$ un arc sortant de s, $\tilde{s} = L_{vr}(s)$ et $L_{er}(f) = d$. La provenance de d est alors donnée par :

$$DProv(f)_{G_{run}} = imProv(f)_{G_{run}} = \tilde{s} \tag{3}$$

Les expressions régulières permettent d'exprimer de façon concise le fait qu'une donnée est produite par une tâche à partir de l'utilisation conjointe de l'ensemble de données d'entrée de cette tâche (opération "+") et le fait que les tâches en aval consomment les données produites par les taches en amont (opération "$\cdot$").

Exemple 3.1 *Considérons le graphe G_r de la Figure 5 (a). La provenance immédiate de la donnée d_4 transitant sur l'arc e_4 est donnée par la tâche qui l'a produite, v et les données directement utilisées en entrée de cette tâche pour la produire, soit d_3 et d_1. Cette information s'écrit de la façon suivante : $imProv(e_4)_{G_r} = \tilde{v} \cdot (d_1 + d_3)$. On a aussi : $DProv(e_4)_{G_r} = \tilde{v} \cdot (d_1 \cdot DProv(e_1)_{G_r} + d_3 \cdot DProv(e_3)_{G_r}) =$ $\tilde{v} \cdot (d_1 \cdot \tilde{s} + d_3 \cdot [\tilde{u} \cdot d_2 \cdot \tilde{s}]) = \tilde{v} \cdot [d_1 + (d_3 \cdot \tilde{u} \cdot d_2)] \cdot \tilde{s}$. Cette formule exprime que la donnée transitant sur l'arc e_4 a été obtenue par l'utilisation de la tâche v qui a pris pour entrées d_1 et d_3 dont on donne la provenance : d_1; qui est obtenue directement de la source s alors que la provenance de d_3 nécessite une étape de récursion supplémentaire : elle est obtenue en sortie de la tâche u qui a utilisé d_2, avec d_2 obtenue par la source s.*

[1] http://openprovenance.org/

Remarques: Compte tenu des propriétés de commutativité, associativité, et distributivité de $\cdot$ et $+$, la provenance d'une donnée est définie de façon unique pour une exécution de workflow fixée. Intuitivement, la notation sous forme d'expression régulière permet de représenter l'exécution de façon concise. Que l'on cherche à factoriser ou développer l'expression, on exprime la même provenance. Par exemple, l'expression $d_4 \cdot \tilde{v} \cdot [d_1 + (d_3 \cdot \tilde{u} \cdot d_2)] \cdot \tilde{s}$ peut aussi s'écrire $d_4 \cdot \tilde{v} \cdot d_1 \cdot \tilde{s} + d_4 \cdot \tilde{v} \cdot d_3 \cdot \tilde{u} \cdot d_2 \cdot \tilde{s}$. Dans cette deuxième formulation, on écrit jusqu'à la source du graphe tous les chemins qui ont produit les données d'entrées à la dernière tâche ayant produit d_4 plutôt que d'avoir une formulation factorisée montrant l'aspect récursif du processus.

D'autre part, étant donnée une exécution de workflow G_{run} et un sommet u, la provenance de tous les arcs sortants de u est la même. En d'autres termes, toutes les sorties d'une même tâche ont la même provenance, ce qui exprime bien qu'elles ont été (récursivement) obtenues de la même façon.

Dans certains cas d'étude de la provenance d'une donnée, il peut être intéressant de connaître simplement les données impliquées dans la production d'une donnée sans se soucier de l'ordre dans lequel elles ont été consommées ni des tâches qui ont été exécutées. Pour cela, nous introduisons la définition suivante de $Useful$ qui fournit toutes les données utilisées pour produire une donnée fixée.

Définition 3.2 ***Données utilisées***
Étant donnée une exécution de workflow $G_{run} = (V_{run}, E_{run})$ soit d l'étiquette de l'arc f dans G_{run}. Les ***données utilisées*** *pour produire d sont fournies par $Useful(f)$ qui est l'ensemble de toutes les étiquettes d_i apparaissant dans $DProv(f)_{G_{run}}$. On dira que d* ***dépend*** *de d_i.*

Il est important de noter que $Useful$ ne suffit pas à montrer que deux provenances sont équivalentes (puisqu'on perd - en plus des tâches - à la fois l'ordre d'utilisation des données et le nombre de fois qu'elles peuvent être utilisées). Néanmoins, si les données dépendantes d'une donnée d sont différentes alors les provenances de d ne sont pas équivalentes.
Nous introduisons maintenant la notion d'**histoire** d'une tâche dans une exécution de workflow qui va nous permettre de définir ensuite la notion fondamentale de provenance d'une sortie d'exécution. L'histoire d'une tâche est liée à la notion de provenance, dans la mesure où la provenance d'une donnée produite par une tâche est obtenue par la concaténation de l'étiquette de la tâche et de l'histoire de la tâche. Formellement :

Définition 3.3 ***Histoire***
*Soit $G_{run} = (V_{run}, E_{run})$ une exécution de workflow. L'****histoire*** *de u dans G_{run} est donnée par une fonction $Hist : V_{run} \rightarrow (L_{VR} \cup L_{ER})^*$, qui est définie comme suit :*

- *si u=$s(G_{run})$, $Hist(s)_{G_{run}} = \varepsilon$ (mot vide)*
- *si u=$t(G_{run})$, soient $e_i \in E_{run}$, les arcs entrants de t ($1 \leq i \leq p$), avec $L_{er}(e_i) = d_i$: $Hist(t)_{G_{run}} = d_1 \cdot DProv(e_1)_{G_{run}} + \ldots + d_p \cdot DProv(e_p)_{G_{run}}$*
- *pour tout sommet $u \in V_{run}$ tel que $u \neq s(G_{run})$ et $u \neq t(G_{run})$ avec $f \in E_{run}$ un arc sortant de u, $Hist(u)_{G_{run}}$ est tel que $DProv(f)_{G_{run}} = \tilde{u} \cdot Hist(u)_{G_{run}}$.*

Exemple 3.2 *Considérons à nouveau le graphe G_r de la Figure 5 (a), l'histoire de u est la provenance des données prises en entrée par u, soit uniquement $d2$: $Hist(u)_{G_r} = d_2 \cdot \tilde{s}$.*

Définition 3.4 ***Provenance de sortie d'un workflow***
Etant donnée une exécution de workflow G_{run}, sa ***provenance de sortie****, notée $OutProv(G_{run})$, est définie comme l'histoire du sommet puits du graphe. Formellement : $OutProv(G_{run}) = Hist(t(G_{run}))_{G_{run}}$.*

Exemple 3.3 *Toujours sur la Figure 5 (a), on a $OutProv(G_r) = [d_4 \cdot \tilde{v} \cdot [d_1 + (d_3 \cdot \tilde{u} \cdot d_2)] \cdot \tilde{s}] + [d_5 \cdot \tilde{u} \cdot d_2 \cdot \tilde{s}]$ qui exprime que la provenance de sortie du graphe est l'histoire de t c'est-à-dire dans cet exemple, la provenance des données d_4 et d_5.*

Nous disposons maintenant de l'ensemble des notions nécessaires pour définir la provenance-équivalence de deux exécutions qui fait l'objet de la sous section suivante.

3.2 Conservation de la Provenance

Nous cherchons à transformer une exécution de workflow non SP en une autre qui soit SP et qui soit le plus proche possible du graphe original. Pour cela nous voulons conserver la provenance de sortie du graphe. Nous introduisons donc ici la notion d'**équivalence des provenances de sortie** de deux graphes.

Définition 3.5 ***Provenance-équivalence***
Soit G_{r1}, G_{r2} deux exécutions de workflows. Nous disons que G_{r1} et G_{r2} sont ***provenance-équivalents*** *si et seulement s'ils ont même provenance de sortie, i.e. $OutProv(G_{r1}) = OutProv(G_{r2})$.*
Nous utiliserons la notation $G_{r1} \overset{prov}{\Leftrightarrow} G_{r2}$ dans ce qui suit.

Exemple 3.4 *Reprenons la Figure 5 et considérons les graphes G_r (a) et G'_r (b). Le graphe G'_r a été obtenu à partir de G_r en dupliquant le sommet u en un sommet u' : ils ont donc la même étiquette. De même, les arcs e_2 et e'_2 ont même étiquette, ainsi que les arcs e_5 et e'_5.*
$OutProv(G'_r) = [L_{er}(e_4) \cdot L_{vr}(v) \cdot [L_{er}(e_1) + (L_{er}(e_3) \cdot L_{vr}(u) \cdot L_{er}(e_2))] \cdot L_{vr}(s)] + [L_{er}(e'_5) \cdot L_{vr}(u') \cdot L_{er}(e_2) \cdot L_{vr}(s)]$
Or $L_{vr}(u') = L_{vr}(u)$, $L_{er}(e_5) = L_{er}(e'_5)$ et $L_{er}(e_2)$ $L_{er}(e_2)$. On a donc :
$OutProv(G'_r) = [d_4 \cdot \tilde{v} \cdot [d_1 + (d_3 \cdot \tilde{u} \cdot d_2)] \cdot \tilde{s}] + [d_5 \cdot \tilde{u} \cdot d_2 \cdot \tilde{s}]$ *qui n'est rien d'autre que $OutProv(G_r)$. D'où : $G_r \overset{prov}{\Leftrightarrow} G'_r$.*

Propriété 3.1 ***Conservation de la provenance des opérations de réduction :*** *Les opérations de réduction conservent la provenance. Plus précisément, en reprenant les notations de la définition 2.3 et en considérant la Figure 2 on a :*

1. *Réduction série : $Hist(w)_{G_1} = Hist(w)_{G_2}$;*
2. *Réduction parallèle : $Hist(w)_{G_1} = Hist(w)_{G_2}$;*
3. *Réduction de sommet : $Hist(w_i)_{G_1} = Hist(w_i)_{G_2}$ pour tout $i \in [1, k]$.*

Cette propriété résulte du fait que nous stockons dans l'étiquette des arcs restants,dans l'ordre inverse du flot de données, les étiquettes des arcs et des sommets qui ont été supprimés.

4 Réécriture de graphes

Dans cette section nous passons d'abord en revue les principales stratégies qui existent dans la littérature pour réécrire un graphe non-SP en un graphe SP. Pour chacune d'elles nous déterminons si elles préservent la provenance ou pas. Dans un second temps, nous introduisons notre stratégie de réécriture de graphe, fondée sur la duplication de sommet en précisant dans quel contexte cette stratégie préserve la provenance.

4.1 (Re)synchronisation

Une stratégie pour rendre SP des structures non SP a été proposée par Escribano [23][24] en se fondant sur la notion de (re)synchronisation. De façon informelle, l'idée est "d'étager" le graphe en ajoutant des sommets (et des arcs) artificiels, qui jouent le rôle de *tâches de synchronisation*. Ces algorithmes combinent trois stratégies principales de synchronisation illustrées en Figure 6. À partir du graphe représenté en (a), la stratégie haute (*up-synchronisation*) fournit le graphe (b), la stratégie basse (*down-synchronisation*) fournit le graphe (c), et la stratégie croisée (*across-synchronisation*) le graphe (d). Dans la suite nous donnons une intuition de la raison pour laquelle chacune de ces stratégies transforme un sous-graphe non SP en un sous-graphe SP et préserve ou non la provenance.

Ces transformations rendent-elles le graphe final SP ? Oui. Considérons la synchronisation haute (graphe 2.(b)). Une réduction parallèle supprime les doubles arcs entre u et v d'une part et entre v et t d'autre part. Une réduction série supprime le sommet u qui n'a plus qu'un arc entrant et un arc sortant suite à la réduction précédente; une réduction parallèle et une réduction série permettent enfin d'obtenir le graphe BSP. Le même type de raisonnement peut se faire sur les cas (c) et (d).

Ces transformations conservent-elles la provenance ? Non. Si la provenance était conservée, aucune dépendance de données ne serait ajoutée ou perdue. Or, dans le graphe (b), la donnée d5 dépend de la donnée d_4 ($d_4 \in Useful(d_5)$), ce qui n'est pas le cas dans le graphe (a) ($d_4 \notin Useful(d_5)$). Dans le graphe (c), c'est la donnée d_3 qui dépend de d_2 ce qui n'est pas le cas dans le graphe (a). Enfin, dans le cas (d), les données en sortie du nouveau sommet de synchronisation x dépendent de toutes les données entrantes de ce sommet, en particulier d_5 dépend de d_1, ce qui n'est pas le cas dans le graphe (a).

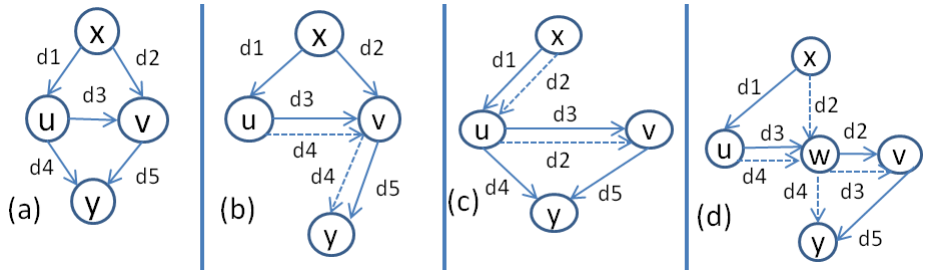

Figure 6: Resynchronization: À partir du graphe interdit en (a), les opérations de up-, down- et across-synchronisation fournissent respectivement les graphes (b), (c) et (d).

4.2 Notre approche : Duplication de sommets

L'approche que nous proposons se fonde sur une opération de duplication de sommet. Nous résolvons le problème des sommets de réduction en proposant de les dupliquer ce qui permet de garder le même ensemble de tâches. Le workflow de la Figure 4 de la section 2 a été transformé en suivant notre procédure. Deux opérations de duplication de sommets sont possibles et représentées en Figure 7, chacune fondée sur l'une des deux opérations de réduction de sommet.

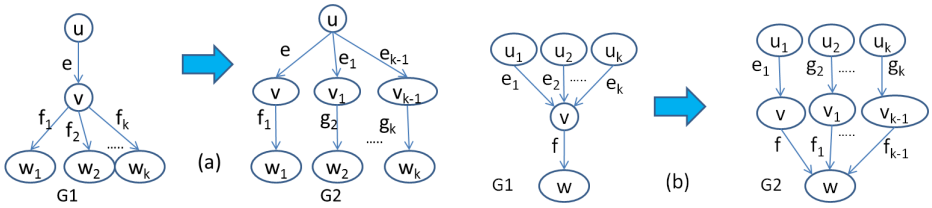

Figure 7: Duplication de sommets (a) à partir des entrées, (b) à partir des sorties.

Ces transformations rendent-elles le graphe final SP ? Oui. La Figure 7 (a) représente bien en G1 un (fragment de) graphe non SP puisqu'il est maximalement réduit et distinct de BSP. Sur G2, l'application de k réductions séries et une réduction parallèle permet d'obtenir un graphe qui n'est plus maximalement réduit et qui après réduction se rapproche de BSP. On peut faire le même raisonnement pour la transformation (b).
Ces transformations conservent-elles la provenance ? Non pour la transformation 7(b). La donnée qui transite sur l'arc f dépend des données transitant sur les arcs $e_1, ..., e_k$ dans G_1. Ce n'est pas le cas dans G_2 : f ne dépend plus que de e_1. En d'autres termes, $e_2...e_k \in Useful(L_{er}(f))$ dans G_1 alors que $e_2...e_k \notin Useful(L_{er}(f))$ dans G_2. La duplication de type 9.(b) ne préserve donc pas la provenance. Le problème majeur vient du fait que l'on a supprimé des entrées de la tâche v qui ne peut plus fournir de résultat s'il lui manque des entrées pour qu'elle soit correctement paramétrée et fonctionnelle.
Oui pour la transformation (a). On ne rencontre pas le problème précédent. La tâche v est dupliquée et chaque copie reçoit la même entrée ($L_{er}(e) = L_{er}(e_1)... = L_{er}(e_{k-1})$), les k sorties de v sont chacune connectées au graphe en aval. La tâche v n'est donc pas modifiée.

Nous définissons formellement l'opération de duplication de type (a) (fondée sur les entrées de la tâche à dupliquer).

Définition 4.1 *Soit $G_1 = (V_1, E_1)$ un st-dag dont les sommets et les arcs sont étiquetés par L_{1vr} et L_{1er}, respectivement. L'opération op transforme G_1 en un st-dag $op(G_1) = G_2 = (V_2, E_2)$, dont les sommets et les arcs sont étiquetés, respectivement par une fonction $L_{2vr} : V_2 \rightarrow L_{VR}$, et une fonction $L_{2er} : E_1 \rightarrow (L_{VR} \cup L_{ER})^*$.*

Duplication de sommet: *Soit v un sommet de V_1 ayant un unique arc entrant $e = (u, v)$ et ayant k arcs sortants $f_1 = (v, w_1), ..., f_k = (v, w_k)$. L'opération op de* ***duplication du sommet*** *v ajoute de nouveaux sommets v_1,..., v_{k-1}, qui sont des copies du sommet v, ajoute les arcs $\{e_1, ..., e_{k-1}\}$, avec $e_i = (u, v_i)$ pour $i \in [1, k-1]$, et remplace les arcs $\{f_2, ..., f_k\}$ par de nouveaux arcs $\{g_2, ..., g_k\}$ avec $g_i = (v_{i-1}, w_i)$*

pour $i \in [2, k]$.
Le graphe $G_2 = (V_2, E_2) = op(G_1)$ *est tel que :* $V_2 \supset V_1$, L_{2vr} *est une extension de* L_{1vr} *sur* V_2, *qui coïncide avec* L_{1vr} *sur* $V_1 \cap V_2$ *avec* $L_{2vr}(v_i) = L_{1vr}(v)$ *pour tout* $i \in [1, k-1]$. *De plus,* L_{2er} *est identique à* L_{1er} *sur* $E_1 \cap E_2$, $L_{2er}(e_{i-1}) = L_{1er}(e)$, *et* $L_{2er}(g_i) = L_{1er}(f_i)$ *pour* $i \in [2, k]$.

Propriété 4.1 ***Conservation de la provenance par l'opération de duplication de sommet.*** *L'opération de duplication de sommet conserve la provenance. Plus précisément, en reprenant les notations de la définition ci-dessus on a :* $Hist(w_i)_{G_1} = Hist(w_i)_{G_2}$ *pour tout* $i \in [1, k]$.

5 Algorithme de transformation

5.1 L'algorithme SPFlow

Dans cette section nous introduisons l'algorithme SPFlow qui réécrit un graphe G non-SP en un nouveau graphe SP, appelé SPG, obtenu à partir de G en dupliquant certains sommets de G, tout en s'assurant que G et SPG sont provenance-équivalents. Nous allons appliquer successivement des réductions série et parallèle, et effectuer des réductions de sommet sur G construisant ainsi un graphe intermédiaire G_{red}, et ce, jusqu'à ce que G_{red} soit le graphe basique BSP. Les sommets à dupliquer sont déterminés par les sommets à réduire dans G_{red}.

Tout d'abord, nous rappelons la notion de **sous-graphe autonome**, présentée dans [7] et utilisée dans notre algorithme. Intuitivement, les sous-graphes autonomes permettent de se limiter à des composantes plus petites du graphe initial pour effectuer les duplications de sommets, sans interaction avec le reste du graphe, puisqu'aucun arc n'entre dans le sous-graphe autonome ou n'en sort.

Définition 5.1 ***Sous-graphe autonome***
Soit G *un st-dag. Nous notons* $G[(v, w)$ *le sous-graphe de* G *de source* v *et de puits* w. *On dit que* $G(v, w)$ *est un* ***sous-graphe autonome*** *de* G *s'il satisfait la propriété suivante : Pour tout chemin* P *de* s *vers* t *dans* G, *l'ensemble des arcs de* $P \cap G[v, w]$ *est vide ou forme un chemin de* v *vers* w. *Notons que* v *peut être* s, *et que* w *peut être* t *ou les deux, mais que* $G(v, w)$ *ne peut pas être un arc unique ni tout le graphe* G.
Si $G(v, w)$ *est un sous-graphe autonome de* G, *nous appelons (v,w) une* ***paire de coupure*** *de* G.

Une décomposition de G en sous-graphes autonomes peut être obtenue en temps linéaire [7]. Dans la suite, nous notons $G[v, w]$ le sous graphe de G qui contient tous les noeuds et arcs de tous les chemins de v) w dans G.

Exemple 5.1 *Le graphe* G *de la Figure 8 a plusieurs sous-graphes autonomes. Par exemple,* $G[s, x]$ *et* $G[x, t]$ *dont les paires de coupure sont* (s, x) *et* (x, t).

Principe de l'algorithme: L'algorithme $SPFlow$ prend en entrée le graphe G et donne en sortie le graphe G_{red} et le graphe SPG. G_{red} est obtenu par réductions

successives de G (y compris des réductions de sommets). Dans SPG certains sommets de G sont dupliqués; ces sommets sont déterminés à partir des réductions de sommet effectuées dans G_{red}.

Initialisation :
(i) $SPG \leftarrow G$; $s \leftarrow s(G)$; $t \leftarrow t(G)$
(ii) $G_{red} \leftarrow MaxRed(G)$
(iii) Décomposer G_{red} en sous-graphes autonomes.
(iv) Appeler $TransfoRec(G, G_{red}, SPG, s, t)$.

Procédure TransfoRec (IN: G; IN/OUT: G_{red}; IN/OUT: SPG; IN: u, p)
Tant que $G_{red} \neq BSP$ ***faire*** :
Etape 1 : Choisir dans G_{red} un sommet v successeur de u dans G_{red} sur lequel on puisse appliquer une réduction de sommet selon Figure 2(c) (cf. Section 2).
Etape 2:
(i) v est la source d'un sous-graphe autonome de puits $w \in G_{red}$.
Rappeler TransfoRec (G, G_{red}, SPG, v,w), en considérant $G_{red}[v, w]$ au lieu de G_{red}, c-a-d, en considérant v comme la nouvelle source du graphe réduit.
(ii) v n'est pas la source d'un sous-graphe autonome.

1. Dupliquer dans SPG le sommet v $k-1$ fois, si v a k successeurs dans G_{red}. Lors de cette duplication, au lieu de simplement dupliquer l'arc e de G_{red}, dupliquer le sous-graphe $SPG[u, v]$ en $SPG[u, v_1], ..., SPG[u, v_{k-1}]$. De même, au lieu de simplement considérer les arcs f_i de G_{red}, considérer les sous-graphes $SPG[v, w_i]$, qui deviennent $SPG[v_1, w_2], ..., SPG[v_{k-1}, w_k]$.
2. Appliquer dans G_{red} l'opération de réduction de sommet à v.
3. $G_{red} \leftarrow MaxRed(G_{red})$

Fin Tant que **Fin TransfoRec**

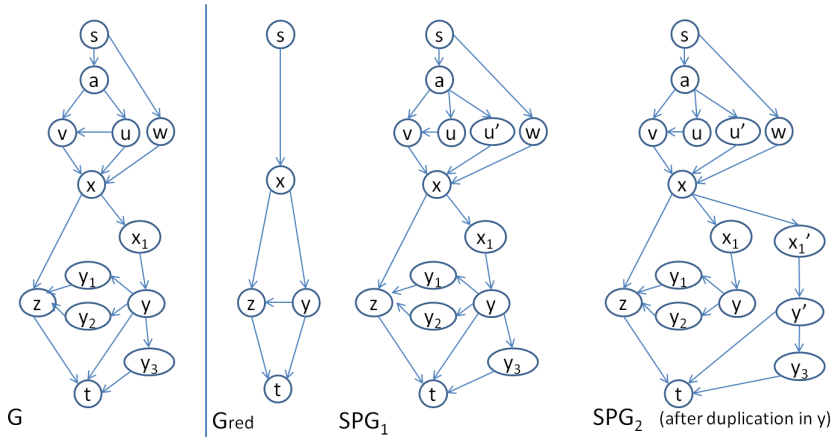

Figure 8: Exemple d'une étape d'exécution de TransfoRec

Exemple 5.2 *La Figure 8 illustre une étape de l'algorithme où $G[s, x]$ a déjà été traité, donnant l'arc (s, x) dans G_{red} après réduction du sommet u et donnant $SPG_1[s, x]$ dans SPG_1, après duplication en u.*

L'algorithme considère alors x comme successeur de s dans G_{red}. Comme (x,t) est une paire de coupure, il rappelle $TransfoRec$ en considérant x comme source. Le sommet y est le successeur de x dans G_{red} sur lequel on effectue une réduction de sommet (dans G_{red}). La duplication de y dans SPG_1 donne alors SPG_2.

5.2 Propriétés de SPFlow

Les propriétés suivantes se montrent par récurrence sur le nombre d'étapes de l'algorithme.

Propriété 5.1 *A toute étape de l'algorithme, on a : $MaxRed(G_{red}) = MaxRed(SPG)$.*

En effet, à un arc (u,v) de G_{red} correspond un sous-graphe $SPG[u,v]$ qui est SP et à la réduction de sommet dans G_{red} correspond une duplication dans SPG qui conduit à la même réduction maximale.

Propriété 5.2 *Pour tout sommet w de G_{red} présent dans SPG et dans G on a : $Hist(w)_{SPG} = Hist(w)_G$=$Hist(w)_{G_{red}}$.*

Cette propriété découle des propriétés des opérations de réductions des sections 3 et 4.

Théorème 5.1 *À la fin de l'exécution de l'algorithme SPFlow on a*

1. *SPG est un graphe SP*
2. *$OutProv(G) = OutProv(SPG)$*

Sketch de preuve:

1. À la fin de l'algorithme, $G_{red} = BSP$ et $G_{red} = MaxRed(G_{red})$. Or, $MaxRed(G_{red}) = MaxRed(SPG)$ (propriété 5.1). Donc $MaxRed(SPG) = BSP$.
2. $OutProv(G) = Hist(t)_G$; or $Hist(t)_G = Hist(t)_{G_{red}}$=$Hist(t)_{G_{SPG}}$ (propriété 5.2). D'où $Hist(t)_G = Hist(t)_{SPG} = OutProv(SPG)$.

6 Expérimentations

Les données utilisées dans cette section ont été collectées dans myExperiment[2], l'entrepôt de workflows scientifique publique le plus important. Nous avons focalisé notre étude sur les workflows conçus dans le cadre du système Taverna, qui est l'un des systèmes de workflows les plus utilisés aujourd'hui et qui est le principal système dont les workflows de myExperiment sont issus. En Mai 2012, 1421 workflows de Taverna étaient présents dans la base. Nous avons retiré de cet ensemble les workflows mal formés ainsi que les doublons; le nombre total de workflows ainsi sélectionnés est alors de 1014 wokflows.

La présente section a deux objectifs : (1) présenter l'étude que nous avons faite sur la structure de workflows réels, (2) fournir des chiffres clés relatifs à l'application de notre approche sur ces workflows.

[2]http://www.myexperiment.org

Table 1: Évolution de la proportion des structures SP parmi les workflows scientifiques de myExperiment

Date	Nombre de workflows	graphes SP (proportion)	graphes SP (proportion)
2010	681	429 (63%)	252 (37%)
2011	879	554 (63%)	325 (37%)
2012	1014	624 (61,5%)	390 (38,5%)

Table 2: Proportion des structures Non-SP vs SP par famille

Famille	#workflows	% de structures non-SP
Simple	92	100 %
Intermédiaire	480	87 %
Complexe	214	45 %
Très complexe	228	6,7 %

6.1 Structure des workflows

Chaque workflow a d'abord été représenté par un st-DAG, *i.e.* un DAG dans lequel nous avons ajouté une unique source pointant vers tous les sommets n'ayant pas de prédécesseurs et un unique puits sur lequel pointent tous les sommets n'ayant pas de successeurs [25]. Nous avons implémenté l'algorithme de détection de structures SP dont nous avons donné un exemple de déroulement dans la section 2, Figure 4. Nous indiquons ci-après des informations sur (i) la proportion de structures non-SP vs SP dans les workflows, (ii) l'évolution de cette proportion et du nombre de workflows dans les dernières années, (iii) la *distance* qui sépare les structures non-SP des structures SP et (iv) des caractéristiques des workflows non-SP.

6.1.1 Proportion de workflows SP et non-SP

Notre premier résultat donné en Table 1 montre qu'il y avait une majorité de structures SP dans l'ensemble des workflows disponibles lorsque myExperiment a débuté et que la proportion de graphes SP est stable.

Plus précisément nous avons étudié la répartition des workflows SP et non-SP sur quatre familles de workflows définies en fonction de leur nombre de sommets : les workflows simples ont de 1 à 3 sommets, les intermédiaires de 4 à 10, les complexes de 11 à 20, et les très complexes strictement plus de 20 sommets.

Sans surprise, on trouve moins de structures SP parmi les workflows les plus complexes. Les chiffres obtenus sont particulièrement tranchés : alors que les workflows simples ou intermédiaires sont quasiment tous SP, la proportion de structures de workflows SP chute pour les workflows de plus de 10 sommets : 55% de structures non-SP pour les workflows complexes et 93% pour les workflows ayant plus de 20 sommets.

Dans la suite, nous étudions la *distance* qui sépare les structures non-SP des structures SP en nous basant sur le nombre d'opérations de réduction de sommet qu'il faut appliquer au graphe de départ pour obtenir le graphe basique BSP [7].

6.1.2 Caractéristiques des workflows non-SP

Dans ce second ensemble d'expériences, notre objectif est d'évaluer à quel point les structures non SP sont éloignées des structures SP. La Figure 9 présente le pourcentage de workflows ayant un nombre donné de sommets de réduction dans les structures non-SP. Par exemple, on peut lire que 31% des workflows non SP ont un sommet de réduction alors que 27% en ont deux.

Le principal résultat donné par la Figure 9 est que 67% des workflows non SP ont seulement de 1 à 3 sommets de réduction.

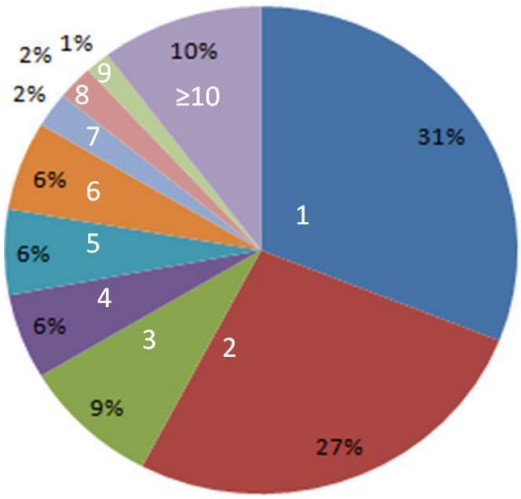

Figure 9: Pourcentage des workflows ayant un nombre fixé de sommets de réductions

6.2 Evaluation de SPFlow

Dans cette sous-section, nous évaluons le comportement de l'algorithme SPFlow sur les données réelles. Nous travaillons ici sur un ensemble de 390 workflows, puisque nous ne considérons que les graphes non-SP que nous réécrivons en graphes SP.

La première information est celle du rapport entre la taille du graphe initial (en nombre de sommets) et la taille du graphe réécrit. La Figure 10 fournit cette information. Bien que le nombre de sommets de réduction ne soit pas directement associé à la taille (en nombre de sommets) d'un graphe, on peut constater que les workflows de moins de 30 sommets ont au pire une taille qui triple lors de l'opération de réécriture. Même si trois graphes ont un nombre de sommets dupliqués qui peut être très important, d'autres très gros graphes (plus de 300 sommets) peuvent avoir un ratio faible.

Enfin, on peut noter que le temps de réécriture est négligeable pour les structures actuelles de workflows (qui ont relativement peu de sommets de réduction). Sur une machine dual core @2.2GHz et 2GB de RAM, le temps minimal d'exécution est de 1 ms, le temps maximum de 434 ms. Même dans le cas de quelques workflows très complexes, ce temps est très raisonnable.

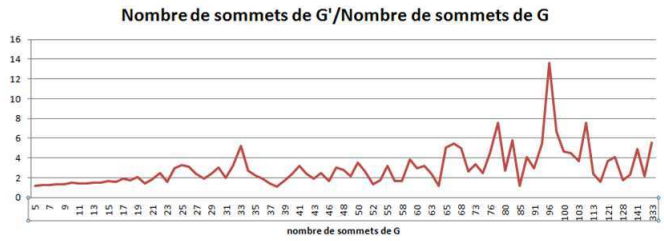

Figure 10: Nombre de sommets dupliqués dans le graphe réécrit par rapport au nombre de sommets dans le graphe initial

7 Discussion

Les workflows scientifiques forment des graphes complexes qu'il faut pouvoir concevoir, visualiser, interroger, exécuter, planifier... Ces actions sont intrinsèquement complexes (impliquant la recherche de sous-graphes dans un graphe, la comparaison de graphes, ...) et conduisent à des problèmes NP-difficiles lorsqu'elles sont effectuées sur des graphes acycliques (DAGs) classiques comme le sont les workflows scientifiques. Au contraire, ces problèmes peuvent être résolus en temps polynomial lorsque la structure est série-parallèle (SP). La réécriture d'un workflow non SP en un workflow SP est donc particulièrement utile, si de plus la nouvelle réécriture ne permet pas de distinguer (pour un observateur extérieur) une exécution du workflow initial d'une exécution du workflow réécrit. En d'autres termes, la provenance des données est conservée. La contribution majeure de cet article est l'introduction d'un algorithme original de réécriture de workflows préservant la provenance. Plus précisément, nous avons : (1) introduit un modèle pour représenter les workflows scientifiques et la provenance des données générées par leurs exécutions, (2) défini la notion de réécriture provenance-équivalente et montré que les stratégies de réécriture existantes ne sont pas provenance-équivalentes, (3) conçu l'algorithme $SPFlow$ qui est provenance-équivalent, (4) évalué notre approche sur un millier de workflows scientifiques.

Nous travaillons actuellement sur le développement d'un outil prenant en entrée un workflow Taverna de structure non SP et fournissant une version SP du workflow qui soit exécutable à son tour dans Taverna. Nous souhaitons étendre notre étude expérimentale et comparer le temps d'exécution de workflows réels avant et après réécriture avec $SPFlow$. Une seconde direction de recherche vise à mieux comprendre les raisons pour lesquelles certains workflows ne sont pas SP: Notre objectif à long terme est de fournir des guides pour concevoir des workflows permettant aux utilisateurs de créer des workflows dont la structure est proche d'une structure de SP.

References

[1] D. Hull, K. Wolstencroft, R. Stevens, C. A. Goble, M. R. Pocock, P. Li, T. Oinn, Taverna: a tool for building and running workflows of services, Nucleic Acids Research 34 (Web-Server-Issue) (2006) 729–732.

[2] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. B. Jones, E. A. Lee, J. Tao, Y. Zhao, Scientific workflow management and the kepler system, Concurrency and Computation: Practice and Experience 18 (10) (2006) 1039–1065.

[3] I. Foster, J. Vockler, M. Woilde, Y. Zhao, Chimera: A virtual data system for representing, querying, and automating data derivation, in: SSDBM, 2002, pp. 37–46.

[4] J. Goecks, A. Nekrutenko, J. Taylor, Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences, in: Genome Biology, 2011, pp. 438–462.

[5] Y. Gil, V. Ratnakar, E. Deelman, G. Mehta, J. Kim, Wings for pegasus: Creating large-scale scientific applications using semantic representations of computational workflows, in: Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada, 2007, pp. 1767–1774.

[6] S. Cohen-Boulakia, S. Lair, N. Stransky, S. Graziani, F. Radvanyi, E. Barillot, C. Froidevaux, Selecting biomedical data sources according to user preferences, Bioinformatics 20 (2004) i86–i93.

[7] W. W. Bein, J. Kamburowski, M. F. M. Stallmann, Optimal reductions of two-terminal directed acyclic graphs, SIAM J. Comput. 21 (6) (1992) 1112–1129.

[8] Z. Bao, S. C. Boulakia, S. B. Davidson, A. Eyal, S. Khanna, Differencing provenance in scientific workflows, in: Proc. of the Int. Conf. on Data Engineering, ICDE 2009, 2009, pp. 808–819.

[9] O. Biton, S. C. Boulakia, S. B. Davidson, C. S. Hara, Querying and managing provenance through user views in scientific workflows, in: Proc. of the Int. Conf. on Data Engineering, ICDE, 2008, pp. 1072–1081.

[10] S. P. Callahan, J. Freire, E. Santos, C. E. Scheidegger, C. T. Silva, H. T. Vo, Vistrails: visualization meets data management, in: SIGMOD Conference, 2006, pp. 745–747.

[11] R. Duffin, Topology of series-parallel networks, Journal of Mathematical Analysis and Applications 10 (1965) 303¨C313.

[12] J. Valdes, R. E. Tarjan, E. L. Lawler, The recognition of series parallel digraphs, in: STOC, 1979, pp. 1–12.

[13] J. Starlinger, S. Cohen-Boulakia, U. Leser, (re)use in public scientific workflow repositories, Proceedings of the 24th International Conference on Scientific and Statistical Database Management (SSDBM).

[14] O. Biton, S. B. Davidson, S. Khanna, S. Roy, Optimizing user views for workflows, in: ICDT, 2009, pp. 310–323.

[15] A. Goderis, P. Fisher, A. Gibson, F. Tanoh, K. Wolstencroft, D. D. Roure, C. A. Goble, Benchmarking workflow discovery: a case study from bioinformatics, Concurrency and Computation: Practice and Experience 21 (16) (2009) 2052–2069.

[16] S. Cohen-Boulakia, U. Leser, Search, adapt, and reuse: the future of scientific workflows, SIGMOD Record 40 (2) (2011) 6–16.

[17] A. Gater, D. Grigori, M. Bouzeghoub, A graph-based approach for semantic process model discovery, in: Graph Data Management, 2011, pp. 438–462.

[18] A. González-Escribano, A. J. C. van Gemund, V. Cardeñoso-Payo, Performance implications of synchronization structure in parallel programming, Parallel Computing 35 (8-9) (2009) 455–474.

[19] K. Lodaya, P. Weil, Series-parallel posets: Algebra, automata and languages, in: STACS, 1998, pp. 555–565.

[20] A. J. C. van Gemund, The importance of synchronization structure in parallel program optimization, in: International Conference on Supercomputing, 1997, pp. 164–171.

[21] L. Finta, Z. Liu, I. Milis, E. Bampis, Scheduling uet-uct series-parallel graphs on two processors, Theor. Comput. Sci. 162 (2) (1996) 323–340.

[22] S. Cohen, S. Cohen-Boulakia, S. Davidson, Towards a model of provenance and user views in scientific workflows, in: Data Integration in the Life Sciences, Vol. 4075 of LNBI, Springer, 2006, pp. 264–279.

[23] A. González-Escribano, Synchronization architecture in parallel programming models, Ph.D. thesis, University of Valladolid (2003).

[24] A. G. Escribano, V. C. Payo, A. van Gemund, Conversion from nsp to sp graphs, Tech. rep., Departamento de Informática, Universidad de Valladolid (1997).

[25] W. Wang, Study of the managment of the provenance information in taverna, in: Internship Master Report (Polytech Paris-Sud) supervised by S. Davidson and S. Cohen-Boulakia, 2009.