Algorithmic aspects of reachability in temporal graphs

Filippo Brunelli

Université Paris Cité

École Doctorale Sciences Mathématiques de Paris Centre (ED 386)
Institut de Recherche en Informatique Fondamentale
Institut national de recherche en informatique et en automatique

Thèse de Doctorat en Informatique

Algorithmic aspects of reachability in temporal graphs

---

présentée par

Filippo Brunelli

dirigée par:

Directeur de thèse: Laurent Viennot, co-Directeur: Pierluigi Crescenzi

Soutenue publiquement le 11, Septembre 2023 devant un jury composé de:

Laurent Viennot	Directeur de recherche, Inria	Directeur de thèse
Pierluigi Crescenzi	Professeur, Gran Sasso Science Institute	Co-Directeur de thèse
Ana Silva	Professeure, Universidade Federal do Ceará	Rapporteuse
Matthieu Latapy	Directeur de recherche, CNRS	Rapporteur
Claire Mathieu	Directrice de recherche, CNRS	Présidente du jury
Tiphaine Viard	Maîtresse de conférences, Télécom Paris	Examinatrice

Abstract

Temporal graphs are an extension of graphs which represent networks that evolve and change over time. In this context, the edges can be available at certain times and unavailable at others and they are called *temporal edges*. The length, or another property associated to an edge, that could classically be captured by the weight of an arc, can now have different values based on the time the edge is available. The classical notions from graph theory require novel definitions for temporal graph, taking into account the temporal dimension. A *temporal walk*, for instance, correspond to a sequence of adjacent temporal edges that are available one are after the other. A node v is *temporally reachable* from a node u if there exists a temporal walk from u to v.

In this dissertation we explore problems that can be grouped into two main topics: temporal walk computation and temporalisation of a static graph.

We study the problem of computing *minimum cost* temporal walks, where "cost" is to be intended in a very wide sense. Indeed, we introduce an algebraic cost structure that can be instantiated to model all the classical criteria of walk optimisation in temporal graphs such as arrival time or duration, as well as linear combination of them or lexicographic compositions. Moreover, we study this problem in temporal graphs that are subject to waiting time constraints. Our main result on this topic is a temporal-edge scanning algorithm for single-source minimum-cost walks taking as input an acyclic time-expanded representation of the temporal graph and running in linear time. We also show that the setting in which we obtain linear-time is the widest possible: an additional logarithmic factor is needed both when the acyclicity assumption is dropped, or when a weaker temporal graph representation is used.

When we speak about temporalisation we are referring to the network design problem of turning a static graph into a temporal graph while optimising a certain criteria. In particular, we study a problem inspired by the optimisation of bus/metro/tramway schedules in a public transport network where each trajectory of a vehicle is modelled by a walk in the directed graph representing the map of the network. We consider the problem of turning a collection of such walks (called trips) in a directed graph into a temporal graph by assigning a starting time to each trip so as to maximise the reachability among pairs of nodes. We obtain several complexity results. Among them, we show that maximising reachability via trip temporalisation is hard to approximate within a factor $\sqrt{n}/12$ in an n-vertex digraph, even if we assume that for each pair of nodes, there exists a trip temporalisation connecting them. On the positive side, we show that there must exist a trip temporalisation connecting a constant fraction of all pairs if we additionally assume symmetry, that is, when the collection of trips to be scheduled is such that, for each trip, there is a symmetric trip visiting the same nodes in reverse order. Notice that symmetry is a fair assumption in the context of public transit networks, where a bus or metro line usually consists in trips in both directions.

Keywords: Temporal graph, temporal reachability, temporal path, temporal walk, waiting-time constraints, temporalisation, time assignment, edge scheduled network, public transit network.

Résumé

Les graphes temporels sont une extension des graphes et représentent des réseaux évoluant au fil du temps. Dans ce contexte, les arêtes peuvent être disponibles ou non à certains moments, et sont appelées *arêtes temporelles*. La longueur, ou une autre propriété associée à une arête, qui est classiquement capturée par le poids d'un arc, peut maintenant avoir différentes valeurs en fonction du moment où l'arête est disponible. Les notions classiques de la théorie des graphes nécessitent maintenant de nouvelles définitions tenant compte de la dimension temporelle. Une *marche temporelle*, par exemple, correspond à une séquence d'arêtes temporelles qui sont adjacentes l'une avec la suivante et qui sont disponibles l'une après la suivante. Un nœud est *temporellement accessible* à partir d'un autre s'il existe une marche temporelle allant de l'un à l'autre.

Dans cette thèse, nous explorons des problèmes qui peuvent être regroupés en deux thèmes principaux : le calcul de la marche temporelle et la temporisation d'un graphe statique.

Nous étudions le problème du calcul des marches temporelles à *coût minimal*, le terme "coût" a ici un sens très large. En effet, nous introduisons une structure de coût algébrique qui peut être instanciée afin de modéliser tous les critères classiques d'optimisation de marche dans les graphes temporels, tels que le temps d'arrivée ou la durée, ainsi que leur combinaison linéaire, ou leur composition lexicographique. De plus, nous étudions ce problème dans des graphes temporels soumis à des contraintes sur le temps d'attente. Notre principal résultat sur ce sujet est un algorithme scannant les arêtes temporelles pour calculer les marches de coût minimal depuis une source donnée. Il prend en entrée la représentation classique de graphe étendu dans le temps sous l'hypothèse de son acyclicité, et s'exécute en temps linéaire. Nous montrons également que le cadre dans lequel nous obtenons un temps linéaire est le plus large possible : un facteur logarithmique supplémentaire est nécessaire lorsque l'hypothèse d'acyclicité est abandonnée ou lorsqu'une représentation plus faible du graphe temporel est utilisée.

Lorsque nous parlons de temporisation, nous nous référons au problème de conception de réseau qui consiste à transformer un graphe statique en un graphe temporel tout en optimisant un certain critère. En particulier, nous étudions un problème inspiré par l'optimisation des horaires de bus, métro ou tramway, dans un réseau de transport public où chaque trajectoire d'un véhicule est modélisée par une marche dans le graphe orienté représentant la carte du réseau. Nous considérons le problème de la transformation d'une collection de telles marches (appelées trajets) dans un graphe orienté en un graphe temporel en assignant une heure de départ à chaque trajet de manière à maximiser l'accessibilité entre les paires de nœuds. Nous obtenons plusieurs résultats de complexité. Nous montrons notamment que la maximisation de l'accessibilité via la temporisation des trajets est difficile à approximer avec un facteur meilleur que $\sqrt{n}/12$ dans un digraphe à n sommets, et ceci,

même si nous supposons que pour chaque paire de nœuds, il existe une temporisation des trajets qui les relie. En revanche, en ajoutant une notion de symétrie sur les trajets, c'est-à-dire, que pour chaque trajet il existe un trajet symétrique visitant les mêmes nœuds dans l'ordre inverse, nous montrons qu'il doit exister une temporisation des trajets reliant une fraction constante de toutes les paires. Notons que la symétrie est une hypothèse raisonnable dans le contexte des réseaux de transport public, où une ligne de bus ou de métro comporte généralement des trajets dans les deux sens.

Mots clés: Graphe temporel, connectivité temporelle, chemin temporel, marche temporelle, temps d'attente contraint, temporalisation, assignation de temps, réseau avec ordonnancement des arêtes, réseau de transport public.

Introduction en français

Les graphes sont un modèle qui représente les relations entre les entités d'un ensemble donné. Ces entités peuvent être des personnes, des ordinateurs ou même des lieux. Habituellement, lorsque nous considérons de telles entités existant dans un système réel, leurs relations, et donc la manière dont elles sont liées, changent au fil du temps. Pour capturer ces comportements dynamiques, nous utilisons les graphes temporels. Les graphes temporels sont une extension des graphes et dans ce modèle, les liens peuvent être disponibles à certains moments et indisponibles à d'autres. La longueur ou tout autre propriété d'un lien, qui pourrait classiquement être capturée par le poids d'un arc, peut maintenant changer en tant que fonction arbitraire du temps. Différents modèles de graphes temporels peuvent être introduits en fonction de l'application considérée. Prenons par exemple, un graphe temporel qui modélise les interactions au sein d'un groupe de personnes, en utilisant une capture des moments où les individus sont en contact. Les personnes sont représentées par des sommets et les interactions sont représentées par des arêtes dotées d'étiquettes temporelles. Un autre modèle pourrait décrire un réseau routier, où chaque croisement est représenté par un sommet, et chaque segment de route reliant deux croisements est une arête dotée d'une fonction décrivant le temps nécessaire pour parcourir cette portion de route à différents moments de la journée. Les graphes temporels peuvent trouver une application dans une grande variété de domaines, car de nombreux réseaux présentent un comportement dynamique. Par exemple, on peut les utiliser pour modéliser les réseaux de transport déjà mentionnés, mais aussi les réseaux sociaux, les réseaux mobiles et distribués, les réseaux d'interaction avec les protéines, et bien d'autres encore. Comme nous l'avons mentionné précédemment, différents modèles peuvent être plus appropriés pour traiter différentes applications. Cela conduit malheureusement à une littérature fragmentée sur le sujet.

Depuis que les graphes temporels ont été envisagés pour la première fois il y a plusieurs décennies, de nombreux chercheurs ont travaillé sur les définitions temporelles qui correspondant aux notions standard de la théorie des graphes. L'une des notions fondamentales des graphes temporels que nous explorerons dans cette thèse est celle de la *connectivité temporelle*. Illustrons un exemple simple de graphe temporel qui modélise un réseau de vols aériens. Dans la Figure 1 (a), nous avons cinq villes et six vols les reliant. Les villes sont représentées par des nœuds, et un vol qui relie une ville à une autre est représenté par un arc étiqueté avec l'heure de départ du vol et sa durée.

Une question qui se pose est de savoir s'il est possible de voyager de Paris à Berlin à l'intérieur de ce réseau. Comme le vol Paris-Vienne atterrit à 17 heures, le vol Vienne-Berlin est déjà parti. Il n'est donc pas possible de se rendre à Berlin depuis Paris dans ce réseau, car les deux villes ne sont pas reliées par une séquence de vols pouvant être embarqués l'un après l'autre. Nous allons utiliser le terme *bord temporel* pour décrire ce qui est représenté ici comme un vol régulier, et *chemin temporel* pour désigner une séquence

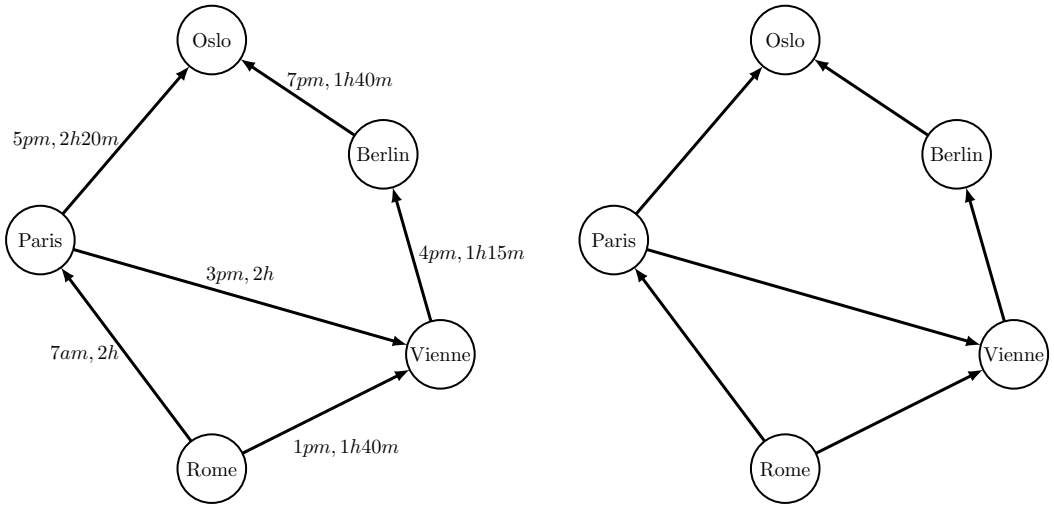

Figure 1: (a) : Représentation d'un réseau de vols par un graphe temporel. Il y a six vols programmés. Par exemple, l'un d'entre eux relie Rome à Paris : le décollage a lieu à 7 heures du matin et le vol a une durée de 2 heures, ce qui signifie que l'avion atterrira à 9 heures du matin. (b) : Un graphe orienté qui représente les connexions du réseau de vols illustré en (a), en ignorant les informations temporelles.

de vols qui permettent d'aller d'une ville de départ à une destination. Dans ce cas, nous dirions que Paris n'est pas *connecté temporellement* à Berlin car aucun chemin temporel ne relie Paris à Berlin.

Considérons maintenant ce qui se passe si nous ne tenons pas en compte l'information temporelle et si nous essayons de représenter le réseau de vols comme dans la Figure 1 (b). Dans ce cas, la quantité partielle d'informations n'est pas suffisante pour répondre à notre question initiale, et pourrait même être trompeuse. Cela met en évidence le fait que si nous essayons de comprimer les données disponibles d'un système intrinsèquement dynamique dans une représentation statique, série de problèmes se posent et ne peuvent pas être abordés correctement. L'information temporelle ne peut être ignorée et doit être traitée correctement. L'introduction de cette dimension temporelle rend inefficace ou inutile de nombreux outils algorithmiques de la théorie classique des graphes. Pour résoudre les problèmes dans ce contexte plus général, il est donc nécessaire de développer et de concevoir de nouvelles procédures efficaces.

Une autre notion clé dans cette thèse est la définition temporelle correspondant à la définition standard du "plus court chemin" dans la théorie des graphes. En effet, alors qu'il est assez intuitif de savoir quel est le critère à optimiser dans un graphe lorsqu'on parle de plus court chemin, il existe plusieurs options intéressantes dans les graphes temporels. Minimiser l'heure d'arrivée, maximiser l'heure de départ, minimiser le temps de voyage, minimiser la durée totale ou le nombre d'arêtes sont quelques-uns des critères les plus couramment étudiés dans le domaine des graphes temporels. Pour revenir à l'exemple du réseau de vols, ces concepts correspondent à un utilisateur qui pourrait être intéressé par le fait d'arriver le plus tôt possible à sa destination, de quitter son domicile le plus tard

possible, de minimiser le temps passé dans les avions, de minimiser le temps total passé à voyager ou le nombre de vols pris.

Supposons par exemple qu'un utilisateur souhaite prendre un vol de Rome à Oslo. Il a le choix entre deux séquences de vols : l'une avec une escale à Paris et l'autre avec deux escales, d'abord à Vienne puis à Berlin. La séquence de vols offrant l'heure d'arrivée la plus précoce de Rome à Oslo est celle qui passe par Paris, puisqu'elle arrive à 19h20, tandis que l'autre option arrive à 20h40. Le choix de faire l'arrêt intermédiaire à Paris est également optimal en ce qui concerne le nombre de vols pris, puisqu'il s'agit de deux vols pour cette option contre trois pour l'autre. D'autre part, la séquence de vols dont la durée totale est la plus courte est celle qui passe par Vienne et Berlin, puisque le temps écoulé entre le départ à 13 heures et l'arrivée à 20 h 40 est de 7 heures et 40 minutes, tandis que pour le choix passant par Paris, cette durée est de 12 heures et 20 minutes. De plus, c'est aussi le choix qui a le départ le plus tardif, puisque dans ce cas le décollage a lieu à 13 heures, alors qu'en passant par Paris, il aurait lieu à 7 heures du matin.

Les graphes temporels peuvent représenter un comportement dynamique qui s'est produit et a été détecté dans un réseau, mais aussi des connexions programmées entre des entités qui n'ont pas encore été observées. Cela conduit à tout un champ de problèmes de conception de réseaux temporels. Prenons un exemple simplifié, tel que le réseau de vols mentionné précédemment. Une compagnie aérienne est confrontée au problème du choix des heures de départ et des trajectoires de ses vols. Elle peut fonder ce choix sur différents critères tels qu'une mesure de joignabilité, par exemple en essayant de maximiser le nombre de correspondances possibles.Supposons que l'on nous donne un digraphe statique, comme celui de la figure 1 (b). Notre tâche est de programmer les vols entre les villes de manière à maximiser le nombre de paires de villes qui seront connectées temporellement par une séquence de vols. Par exemple, un horaire possible serait celui représenté dans la figure 1 (a). Toutefois, il ne s'agirait pas d'un horaire optimal, car en choisissant un décollage plus tôt pour le vol Paris-Vienne, par exemple à 13 heures, nous relierions une paire de villes supplémentaire, c'est-à-dire Paris à Berlin. Dans un contexte similaire, plutôt que de concevoir le réseau de vols à partir de zéro, nous pourrions avoir la possibilité d'apporter quelques modifications, comme retarder l'heure de départ d'un vol programmé.

Organisation

Dans le chapitre 1, nous commençons par rappeler et donner la notation des notions classiques de la théorie des graphes. Ensuite, nous introduisons les graphes temporels en utilisant une nouvelle définition qui peut être affinée pour décrire plusieurs modèles de graphes temporels présents dans la littérature. Nous définissons également les problèmes de chemins temporels les plus classiques qui ont été étudiés au cours des dernières décennies. Le modèle que nous introduisons nous permet d'avoir un aperçu général de l'état de l'art de ces problèmes de chemins temporels. Nous concluons le chapitre en décrivant quelques résultats algorithmiques classiques.

Le reste de la thèse est divisé en deux parties principales : le calcul des marches temporelles dans les chapitres 2 et 3, et la temporalisation d'un graphe statique dans le chapitre 4.

Calcul de la marche temporelle. Le calcul des plus courts chemins est sans aucun doute l'un des problèmes les plus fondamentaux de la théorie algorithmique des graphes, et

également l'un des sous-routines les plus importants pour une grande diversité d'applications dans les réseaux. Bien que sa complexité ait été largement étudiée dans le contexte des graphes statiques, des améliorations restent possibles dans le cas des graphes temporels. Des algorithmes en temps linéaire pour de nombreux critères différents de marches temporelles ont été conçus par différents chercheurs utilisant divers algorithmes. Notre objectif était de fournir un outil algorithmique unique et efficace capable de calculer des marches temporelles génériques optimales. Nous étudions le problème du calcul des marches temporelles *coût minimum*, le terme "coût" étant pris dans un sens très large. En effet, nous introduisons une structure de coût algébrique qui peut être instanciée pour modéliser tous les critères classiques d'optimisation des marches dans les graphes temporels, ainsi que leurs combinaisons linéaires ou leurs compositions lexicographiques. De plus, nous étudions ce problème dans des graphes temporels soumis à des contraintes de temps d'attente. Cela signifie que lors de la navigation dans le graphe temporel, il n'est pas possible de rester stationnaire à un nœud plus d'un certain temps avant de passer au sommet suivant. Il convient de noter que de telles contraintes d'attente sont naturelles dans plusieurs contextes, tels que les réseaux de transport mentionnés précédemment. Considérons le réseau de vols de la figure 1 (a). Un utilisateur peut ne pas vouloir passer plus de quelques heures dans un arrêt intermédiaire. Dans ce cas, la séquence de vols de Rome à Oslo avec escale à Paris ne serait plus un choix possible. En effet, le temps écoulé entre l'arrivée du premier vol et le départ du second est de 8 heures. D'autre part, un utilisateur peut demander que le délai entre un atterrissage et le décollage suivant soit d'au moins 3 heures, afin d'avoir le temps d'utiliser la correspondance aérienne en toute sécurité. Dans ce cas, la séquence de vols de Rome à Oslo avec escale à Vienne et à Berlin ne serait plus réalisable. En effet, le temps écoulé entre l'arrivée à Vienne et le départ du vol suivant vers Berlin est seulement de 1 heure et 35 minutes.

Notre principal résultat sur ce sujet est un algorithme de balayage des arêtes temporelles pour les marches à coût minimal à source unique prenant comme entrée une *représentation temporelle acyclique* du graphe temporel et s'exécutant en temps linéaire. Une représentation étendue dans le temps est une représentation classique d'un graphe temporel à travers un graphe statique. L'hypothèse d'acyclicité signifie que le graphe temporel ne contient pas de cycle d'arêtes temporelles avec un temps de parcours nul à chaque instant. Nous montrons également que le cadre dans lequel nous obtenons un temps linéaire est le plus large possible : un facteur logarithmique supplémentaire est nécessaire lorsque l'hypothèse d'acyclicité est abandonnée ou lorsqu'une représentation plus faible du graphe temporel est utilisée. De plus, nous étendons l'algorithme aux réseaux de transport public, un modèle de graphe temporel où différents types d'arêtes temporelles représentent soit des véhicules programmés se déplaçant dans le réseau, soit la possibilité de marcher à tout moment d'un arrêt à l'autre.

Temporalisation. Inspirés par l'optimisation des horaires de bus/métro/tramway dans un réseau de transport public, nous considérons le problème de conception de réseau consistant à transformer une collection de promenades (appelées trajets) dans un graphe orienté en un graphe temporel en assignant une heure de départ à chaque trajet de manière à maximiser la joignabilité entre les paires de nœuds. Chaque trajet représente la trajectoire d'un véhicule et ses arêtes doivent être programmées l'une après l'autre. L'attribution d'une heure de départ au trajet force donc l'heure de départ de toutes ses arêtes. Nous appelons cette assignation d'une heure de départ une temporisation de voyage. Le problème que nous étudions consiste à trouver une temporisation de voyage qui maximise le nombre de

paires de nœuds qui sont connectés par un chemin temporel. Le problème de la transformation d'un graphe non orienté en un graphe temporel en ordonnant chaque arc de manière indépendante a déjà été étudié dans le cadre du bavardage. Dans ce contexte, l'objectif était de comprendre quand il est possible de connecter temporellement toutes les paires [39]. Il est surprenant de constater que l'approximation de la joignabilité temporelle maximale semble avoir reçu peu d'attention dans la littérature. Notre problème de temporisation des trajets prend en compte un nouveau type de dépendance temporelle où les arcs sont regroupés en promenades qui doivent être programmées de manière séquentielle. A notre connaissance, cette idée est nouvelle bien qu'elle semble naturelle dans des contextes tels que les réseaux de transport en commun. De plus, nous étudions également le problème sans tenir compte de la contrainte de déplacement, ce qui signifie que chaque arc d'un digraphe peut être programmé indépendamment des autres arcs.

Nous obtenons plusieurs résultats de complexité. Parmi ceux-ci, nous montrons que la maximisation de la joignabilité via la temporisation de voyage est difficile à approximer avec un facteur $\sqrt{n}/12$ dans un digraphe de n-vertex, même si nous supposons que pour chaque paire de nœuds, il existe une temporisation de voyage les reliant. En revanche, nous montrons qu'il doit exister une temporisation de trajet reliant une fraction constante de toutes les paires si nous supposons en plus une symétrie, c'est-à-dire lorsque pour chaque trajet de la collection, il existe un trajet symétrique visitant les mêmes nœuds dans l'ordre inverse. Il est à noter que la symétrie est une hypothèse raisonnable dans le contexte des réseaux de transport public, où une ligne de bus ou de métro se compose généralement de trajets dans les deux sens.

Contents

Introduction

Graphs are a model that represents the way entities of a certain set are linked together. For instance, these entities can be persons, computers or even places. Most of the time, when we consider such entities that exist in an actual system, their relations, and thus the way they are linked, change over time. Temporal graphs are an extension of graphs that capture this dynamic behaviour. In this context, the links can be available at certain times and unavailable at others. The length, or another property of a link, that could classically be captured by the weight of an arc, can now change as an arbitrary function of time. Different models of temporal graphs can be introduced depending on the application that is considered. For example, a temporal graph can model the interactions in a group of people, using a record of the time instants at which individuals have been in contact. People are represented by vertices, and interactions are represented by edges equipped with time labels. A different model can describe a road network. Each crossing is represented by a vertex, and each segment of road connecting two crossings is an edge equipped with a function that describes the amount of time it takes to traverse that portion of the road at different times of the day. Temporal graphs can find application in a huge variety of domains, since many networks have inherent dynamic behaviour. For example the already mentioned transportation networks, but also social networks, mobile and distributed networks, protein interaction networks, and several others. As we hinted before, different models can be more suitable to deal with different applications. This unfortunately leads to a fragmented literature on the topic.

Since temporal graphs have been considered for the first time decades ago, authors have worked on defining the counterpart of standard notions from graph theory. One of the fundamental notions in temporal graph that we will explore in this dissertation is *temporal connectivity*. Let us present a small example of a temporal graph that models a flight network. In Figure 2 (a) we have five cities and six flights between them. The cities are represented as nodes, and a flight that connects a city to another is represented through an arc labelled with the departure time of the flight and its duration.

A possible question would be whether it is possible to travel from Paris to Berlin inside this network. As the flight from Paris to Vienna lands at 5pm, the flight from Vienna to Berlin has already left. It is thus not possible to reach Berlin from Paris in this network, because the two cities are not connected by a sequence of flights that can be boarded one after the other. We are going to call *temporal edge* what here is represented as a scheduled flight, and *temporal path* a sequence of flights that permit to go from a starting city to a destination. In this case, we would say that Paris is not *temporally connected* to Berlin as not temporal path connects Paris to Berlin.

Let us now take a look at what happens if we disregarded the temporal information and tried to represent the flight network as in Figure 2 (b). In this case, the partial amount

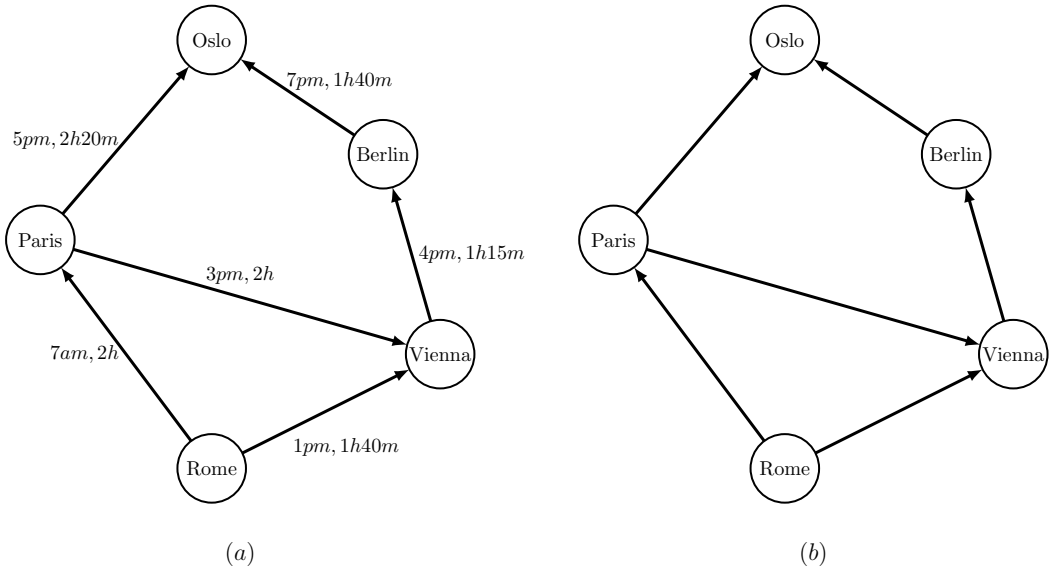

Figure 2: (a): The representation of a flight network through a temporal graph. There are six scheduled flights. For example, one of them is from Rome to Paris: the take off is at 7am and the flight has a duration of 2 hours, meaning the plane will land at 9am.
(b): A directed graph that represents the connections of the flight network illustrated in (a), ignoring the temporal information.

of information is not enough to answer our first question, and might even be misleading. This to understand that, if we tried to compress available data from a system that is inherently dynamic into a static representation, there is a whole set of problems that cannot be approached correctly. The temporal information cannot be ignored and has to be handled properly. The introduction of this temporal dimension makes many algorithmic tools of classical graph theory inefficient or useless. In order to solve problems in this more general context it is thus needed to develop and design novel efficient procedures.

Another notion that plays a key role in this dissertation is the counterpart of the standard definition of "shortest path" from graph theory. Indeed, while it is fairly intuitive what is the criteria to optimize in a graph when speaking about shortest paths, there are several different interesting options in temporal graphs. Minimising the arrival time, maximising the departure time, minimising the time spent travelling, minimising the overall duration or the number of edges are some of the most common criteria studied in the field of temporal graphs. Going back to the example of the flight network these concepts correspond to a user that might be interested in respectively arriving as early as possible to the destination, leaving their house as late as possible, minimising the time spent on planes, minimising the overall time spent traveling or the number of flights taken.

For example, let us suppose that a user wants to fly from Rome to Oslo. There are two sequences of flights they can choose from: one with an intermediate stop in Paris and the other one with two intermediate stops, first in Vienna and then in Berlin. The sequence of flights that has the earliest arrival time from Rome to Oslo is the one going through Paris, as it arrives at time 7:20pm, while the other one arrives at 8:40pm. The choice to make the intermediate stop in Paris is also optimal concerning the number of flights taken, as they are

two for this option and three for the other one. On the other hand, the sequence of flights that has minimum overall duration is the journey going through Vienna and Berlin, as the time elapsed between departure at 1pm and arrival at 8:40pm, is 7 hours and 40 minutes, while for the choice through Paris this quantity is 12 hours and 20 minutes. Moreover, this is also the choice that has the latest departure, as in this case the take off is at 1pm, while going through Paris it would be at 7am.

Temporal graphs can represent dynamic behaviour that occurred and was detected in a network, but also scheduled connections between entities that have yet to be observed. This leads to a whole field of temporal network design problems. As a simplified example we could think about the flight network discussed earlier. A flight company faces the problem of choosing departure times and trajectories of their flights. They could base this choice on different criteria as some reachability measure, for example trying to maximise the number of possible connections. Let us imagine that we are given a static digraph, like the one in Figure 2 (b). We are given the task to schedule flights between the cities in a way that maximises the number of cities that will be temporally connected by a sequence of flights. For example, a possible schedule would be the one represented in Figure 2 (a). However, this would not be an optimal schedule, as by choosing an earlier take off for the flight from Paris to Vienna, for example 1pm, we would connect one more pair of cities, that is Paris to Berlin. In a similar setting, rather than designing the flight network from scratch, we might only have the option to make some changes, like delaying the departure time of a scheduled flight.

Organisation

In Chapter 1 we first recall and give the notation for classical graph theory notions. We then introduce temporal graphs using a novel definition that can be refined to describe several models of temporal graphs present in the literature. We also define the most classical temporal path problems that have been studied in the last decades. The model we introduce allows us to provide a wide picture about the state of the art of such temporal path problems. We conclude the chapter by describing some classical algorithmic results.

The remaining of the thesis is divided into two main parts: temporal walk computation in Chapter 2 and 3, and temporalisation of a static graph in Chapter 4.

Temporal walk computation. Computing shortest paths is certainly one of the most fundamental problems within algorithmic graph theory, as well as one of the most important subroutines for a large diversity of applications in networks. While its complexity has been extensively covered in the context of static graphs, there is still room for improvement in temporal graphs. Linear time algorithms for many different criteria of temporal walks were designed by different authors using diverse algorithms. Our goal was to provide a single efficient algorithmic tool capable to compute generic optimal temporal walks. We study the problem of computing *minimum cost* temporal walks, where "cost" is to be intended in a very wide sense. Indeed, we introduce an algebraic cost structure that can be instantiated to model all the classical criteria of walk optimisation in temporal graphs, as well as linear combinations of them or lexicographic compositions. Moreover, we study this problem in temporal graphs that are subjected to waiting time constraints. This means that when navigating the temporal graph it is not possible to stay stationary at a node more than a certain amount of time before moving onto the next vertex. Note that such waiting

constraints are natural in several contexts, such as the transportation networks mentioned before. Let us consider the flight network in Figure 2 (a). A user might not want to spend more than a few hours in an intermediate stop. In this case, the sequence of flights from Rome to Oslo stopping in Paris would not be a possible choice anymore. This is because the time elapsed between the arrival of the first flight and the departure of the second one is 8 hours. On the other hand, a user might request that the time between a landing and the following take off is at least 3 hours, to have time to safely use the plane connection. In this case, the sequence of flights from Rome to Oslo stopping in Vienna and Berlin would not be feasible anymore. Indeed, the time elapsed between the arrival in Vienna and the departure of the next flight to Berlin is just 1 hour and 35 minutes.

Our main result on this topic is a temporal-edge scanning algorithm for single-source minimum-cost walks taking as input an *acyclic time-expanded representation* of the temporal graph and running in linear time. A time-expanded representation is a classical representation for a temporal graph through a static graph. The acyclic assumption means that the temporal graph does not contain a cycle of temporal edges with zero-travel time at any time instant. We also show that the setting in which we obtain linear-time is the widest possible: an additional logarithmic factor is needed both when the acyclicity assumption is dropped, or when a weaker temporal graph representation is used. Moreover, we extend the algorithm to the setting of public transit networks, a model of temporal graph where different type of temporal edges represent either scheduled vehicles moving in the network or the possibility to walk at any time from a stop to another.

Temporalisation. Inspired by the optimisation of bus/metro/tramway schedules in a public transport network, we consider the network design problem of turning a collection of walks (called trips) in a directed graph into a temporal graph by assigning a starting time to each trip so as to maximise the reachability among pairs of nodes. Each trip represents the trajectory of a vehicle and its edges must be scheduled one right after another. Setting a starting time to the trip thus forces the departure time of all its edges. We call such a starting time assignment a trip temporalisation. The problem we study consists in finding a trip temporalisation that maximises the number of pairs of nodes that get connected by a temporal path. The problem of turning an undirected graph into a temporal graph by scheduling each arc independently has already been considered in the gossip setting. The focus in this context was to understand when it is possible to temporally connect all pairs [39]. Surprisingly, approximating maximum temporal reachability seems to have received little attention in the literature. Our trip temporalisation problem takes into account a novel type of temporal dependency where arcs are grouped into walks that must be sequentially scheduled. To the best of our knowledge, this idea is new although it seems natural in contexts such as transit networks. Moreover, we also study the problem disregarding the trip constraint, meaning that each arc of a digraph can be scheduled independently from the other arcs.

We obtain several complexity results. Among them, we show that maximising reachability via trip temporalisation is hard to approximate within a factor $\sqrt{n}/12$ in an n-vertex digraph, even if we assume that for each pair of nodes, there exists a trip temporalisation connecting them. On the positive side, we show that there must exist a trip temporalisation connecting a constant fraction of all pairs if we additionally assume symmetry, that is, when for each trip in the collection, there is a symmetric trip visiting the same nodes in reverse order. Notice that symmetry is a fair assumption in the context of public transit networks, where a bus or metro line usually consists in trips in both directions.

Publications

Parts of this dissertation appeared in the following publications:

Chapter 2:

[**9**] : Filippo Brunelli, Pierluigi Crescenzi, and Laurent Viennot. On computing pareto optimal paths in weighted time-dependent networks. *Information Processing Letters*, 2021.

[**11**] : Filippo Brunelli and Laurent Viennot. Computing temporal reachability under waiting-time constraints in linear time. To appear: *2nd Symposium on Algorithmic Foundations of Dynamic Networks, SAND 2023*, June 19-20, 2023, Pisa, Italy.

[**12**] : Filippo Brunelli and Laurent Viennot. Minimum-cost temporal walks under waiting-time constraints in linear time. *CoRR*, 2022.

Chapter 4:

[**10**] : Filippo Brunelli, Pierluigi Crescenzi, and Laurent Viennot. Maximizing reachability in a temporal graph obtained by assigning starting times to a collection of walks. *Networks*, 2023.

[**3**] : Alkida Balliu, Filippo Brunelli, Pierluigi Crescenzi, Dennis Olivetti, and Laurent Viennot. A note on the complexity of maximizing temporal reachability via edge temporalisation of directed graphs. *CoRR*, 2023.

Chapter 1

Preliminaries and State of the Art

1.1 Preliminaries

1.1.1 Basic graphs notions

Definition 1 (Directed graph). *A* directed graph *(or* digraph*) D is defined as a pair (V, F), where:*

- *V is a set of elements called* vertices *or* nodes*,*
- *$F \subseteq V \times V$ is a set of ordered pairs of nodes, called* arcs *(or* edges*).*

In an *undirected graph* the edges are unordered pairs; since most of the time we deal with directed graphs rather than undirected, we give all the basic definitions for the directed model, and we are careful to mention that a graph is undirected when it is going to be the case. Moreover, to emphasize the difference with temporal graphs, when we refer to a directed or an undirected graph we might add the adjective *static*. We will prefer the term *arcs* rather than *edges* when speaking about static graphs, in order to emphasize the difference with *temporal edges* that are introduced later.

Let $e = (u, v) \in F$ be an arc. We call u the *tail* of e and v the *head* of e, and we denote them respectively with $tail(e)$ and $head(e)$. We define the *out-neighbourhood* of a node v as the set of nodes $N_{out}(v) = \{w : (v, w) \in F\}$, and if $w \in N_{out}(v)$ we say that w is an *out-neighbour* of v. We define the *out-degree* of v as the cardinality of $N_{out}(v)$. Similarly, we define the *in-neighbourhood* of a node v as the set of nodes $N_{in}(v) = \{u : (u, v) \in F\}$, and if $u \in N_{in}(v)$ we say that u is an *in-neighbour* of v. We define the *in-degree* of v as the cardinality of $N_{in}(v)$. In the following we will denote by $[k]$ the set of natural numbers $\{1, 2, \ldots, k\}$.

Definition 2 (Walk). *Given a directed graph $D = (V, F)$, a* walk *from a vertex u to a vertex v is defined as a sequence of arcs $\langle e_i, e_2, \ldots, e_k \rangle$, where: $tail(e_1) = u$, $head(e_k) = v$, and $tail(e_{i+1}) = head(e_i)$ for each $i \in [k-1]$.*

A *path* is a walk through distinct vertices. A node v is said to be *reachable* from a node u if there exists a walk from u to v (in the following, we will assume that a node is reachable from itself).

An *out-arborescence* rooted in r is a digraph $D = (V, F)$ such that for any vertex $v \in V$ there exists exactly one walk from r to v. Similarly, an *in-arborescence* rooted in r is a directed graph $D = (V, F)$ such that for any vertex $v \in V$ there exists exactly one walk from v to r.

We call a *multi directed graph* (or *multidigraph*) a digraph $D = (V, F)$ that is allowed to have multiple arcs with same tail and head, namely, F is a multiset. Here, the term *multiset* or *collection* is used to denote a set in which order is ignored but multiplicity is significant. The cardinality $|A|$ of a multiset A denotes the sum of the multiplicities of the distinct elements in A: in this case, $|F|$ denotes the number of (not necessarily distinct) arcs in F. We call a *weighted multidigraph* a multidigraph where each arc is associated to a weight. We denote a weighted multidigraph D as a pair (V, F) where V is the set of vertices and F is the set of *(weighted) arcs*, and each weighted arc is represented as a triple (u, v, μ), where u and v are vertices, respectively tail and head of the arc, and $\mu \in \mathbb{R}_{\geq 0}$ is the weight of the arc.

We define the *length* of a path P in a digraph as the number of its arcs, or the sum of the weights of its arcs if it is a weighted digraph. We say that a path P from u to v is a *shortest path* if there is no other path from u to v with inferior length. We call *distance* of v from u the length of a shortest path from u to v.

1.1.2 Basic algorithmic notions

A *breadth-first search (BFS)* algorithm can be used to explore a digraph (V, F) starting from a vertex u and visiting all the nodes reachable from u by increasing distance. To be more precise, the algorithm works in the following way. In the beginning, all vertices are marked as non-visited. A queue, that follows the *first-in-first-out* rule, is instantiated and the input vertex u is added to the queue. At each step, until the queue is empty, the algorithm extracts a node from the queue, marks it as visited, and adds to the queue the out-neighbours of the extracted node that have not been marked as visited yet. The algorithm has a complexity of $O(|V|+|F|)$. The order in which the vertices are visited is called a *BFS ordering*. Moreover, a BFS execution also induces an out-arborescence rooted in the input vertex.

A fundamental algorithmic result that we will use in the context of shortest path computation is Dijkstra's algorithm [28]. Given a weighted digraph with non-negative weights $D = (V, F)$ and a source node s, it computes the shortest path towards every other node in the graph. The algorithm performs the following instructions. It starts by marking all nodes as non-visited and setting a parent pointer to a default value for each node. It assigns to each node u a tentative distance $d(u)$: assigns zero to the source and plus infinity to every other node. This tentative distance represents the length of the shortest path found so far by the algorithm and it is going to be updated during the execution. A plus infinity distance means that no path has been found until that point. At each iteration we select a node u that is non-visited with the lowest tentative distance. For each of the non-visited out-neighbours v of u, if the tentative distance $d(v)$ is greater than $d(u)$ plus the weight of (u, v), then $d(v)$ is updated to the lower value and the parent pointer of v is set to u. After this operation u is marked as visited. At the end of the execution $d(v)$ is the distance from s to v, and following the parent pointers we can retrieve a shortest path.

The complexity of the algorithm depends on the data structure used to store the non-visited nodes, which affects the time needed to pick one with minimum distance. When the algorithm is implemented using a Fibonacci Heap as priority queue the complexity is $O(|F| + |V| \log |V|)$.

1.1.3 Basic complexity notions

We denote with P the set of decision problems that can be solved in polynomial time with respect to the input size by a deterministic Turing machine. We denote with NP the set of decision problems that can be verified in polynomial time by a deterministic Turing machine, where verify means check if a given solution is a feasible solution. We say that a problem is NP-hard if every problem in NP can be reduced in polynomial time to it; which means finding a polynomial time algorithm for any NP-hard problem would lead to polynomial algorithms for all problems in NP. In different words we can say that a problem is NP-hard if it is at least as hard as the hardest problem in NP. Finally, we say that a decision problem is NP-complete if it is in NP and it is NP-hard.

Let us now present an example of NP-complete problem that we work with in the next chapters. In order to define the problem we first introduce some additional notions.

A *Booloean formula* is a logical formula built from Boolean variables and logical operators. In particular we will consider the three following logical operators: the logical *and* called *conjunction* and denoted by $\wedge$, the logical *or* called *disjunction* and denoted by $\vee$, and finally the logical *not* called *negation* and denoted by $\neg$. A *literal* is given by a variable or the negation of a variable. In the following, a *clause* is a disjunction of literals. Finally, a *formula* (in *conjunctive normal form*) is a conjunction of clauses. An *assignment* is a function that associate to each variable a Boolean value, and a formula is *satisfied* by an assignment if the expression obtained replacing the variables with the corresponding values from the assignment evaluates to TRUE.

The *Boolean satisfiability problem (*SAT*)* consists in determining whether a given formula is satisfiable or not, meaning to establish if there exists an assignment that satisfies the formula. We consider a particular variant of the SAT problem which is called 3-SAT. In this case each clause in the formula contains exactly three literals. For example an instance of the 3-SAT problem could be the formula $\Phi = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$. The assignment α such that $\alpha(x_1) =$ FALSE, $\alpha(x_2) =$ TRUE and $\alpha(x_3) =$ FALSE satisfies Φ, which implies that Φ is satisfiable. The 3-SAT problem is NP-complete, and in Chapter 4 several results are obtained through reductions from 3-SAT. Notice that given a decision problem $\mathcal{P}$, since 3-SAT is NP-complete, by showing a polynomial time reduction from 3-SAT to $\mathcal{P}$ we deduce that there exists a polynomial time reduction from any problem in NP to $\mathcal{P}$, that is $\mathcal{P}$ is NP-hard. If, on top of that, we show that it is possible to verify that a given solution for $\mathcal{P}$ is a feasible solution, then $\mathcal{P} \in$ NP and we can conclude that $\mathcal{P}$ is NP-complete.

1.1.4 Temporal graphs

Generally speaking, a *temporal graph* is a graph where the availability or the travel time of the arcs changes over time. Following this general concept it is possible to provide a definition which establishes a hierarchy among most models of temporal graphs that are present in the literature. The following definition is not classical, but it helps to gather and speak about state of the art results in a coherent way.

Definition 3 (Temporal graph). *A* temporal graph *G is given by a set of vertices V and a set of temporal edges E. A temporal edge is defined as a tuple $e = (u, v, I, \lambda)$, where:*

- *$u \in V$ is the* tail *of e,*
- *$v \in V$ is the* head *of e,*

- *I is an interval in $\mathbb{R}$ which is called the* availability interval *of e,*
- *$\lambda : I \to \mathbb{R}_{\geq 0}$ is a* travel time *(or* delay*) function.*

The availability interval of a temporal edge could be a closed, open or mixed interval. A temporal edge $e = (u, v, I, \lambda)$ denotes the fact that it is possible to go from node u to node v departing at any time instant $\tau \in I$ and taking $\lambda(\tau)$ amount of time to traverse the edge. Multiple temporal edges with the same tail and head are allowed, and among such temporal edges, overlapping availability intervals are allowed. The travel time function λ of a temporal edge e induces an *arrival time* function $arr_e : I \to \mathbb{R}$ that associates to each possible departure time $\tau \in I$ the arrival time to the head of the edge, namely $arr_e(\tau) = \tau + \lambda(\tau)$. The arrival time function and the travel time function can be easily computed from one another, yielding to equivalent models if one or the other is given.

By assuming different properties on the travel time functions of the temporal edges we obtain different temporal graph models. In particular, by considering stronger and stronger assumptions we obtain a hierarchy of models where each model encompasses the next one.

Definition 4 (Piecewise linear temporal graph). *A* piecewise linear temporal graph *is a temporal graph such that, for each temporal edge, its travel time function λ is affine.*

Definition 5 (Piecewise constant temporal graph). *A* piecewise constant temporal graph *is a piecewise linear temporal graph such that, for each temporal edge, its travel time function λ is constant.*

Given a temporal edge $e = (u, v, I, \lambda)$, since λ is a constant function over the availability interval I in the case of piecewise constant temporal graphs, with a slight abuse of notation, we may represent the temporal edge using a scalar $\lambda \in \mathbb{R}$ which is the image of the travel time function λ, instead of the travel time function itself.

Definition 6 (Point availability temporal graph). *A* point availability temporal graph *is a piecewise constant temporal graph such that for each temporal edge, its availability interval I is a point $\tau \in \mathbb{R}$ (i.e. $I = [\tau, \tau]$).*

In this case, a temporal edge $e = (u, v, I, \lambda)$ can be denoted by a quadruple $e = (u, v, \tau, \lambda)$, where $\tau, \lambda \in \mathbb{R}$.

Definition 7 (Uniform temporal graph). *A* uniform temporal graph *is a point availability temporal graph such that the travel time is a constant c over all temporal edges. In particular we say* uniform strict *when $c = 1$ and* uniform non-strict *when $c = 0$.*

In this case a temporal edge $e = (u, v, I, \lambda)$ can be denoted with a lighter notation by a triple (u, v, τ) and the constant c is specified separately.

This hierarchy is useful to identify which results obtained in a model can be translated to another, to understand which properties make a problem become difficult, and also to get a better overview on a fragmented literature.

Whenever we will see a drawing of a temporal graph, the temporal edges are going to be illustrated as in Figure1.1.

1.1.5 Basic definitions in temporal graphs

For what concerns temporal graphs and in particular this work, one of the most important notion is the one of *temporal walk*.Intuitively, a temporal walk consists in moving from node

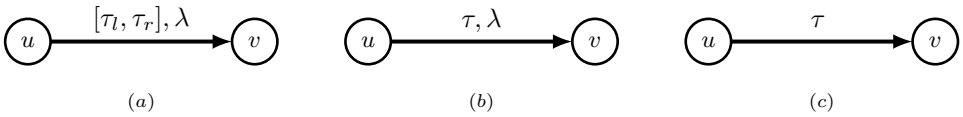

Figure 1.1: An illustration of a temporal edge in a piecewise constant temporal graph (a), in a point availability temporal graph (b) and in a uniform temporal graph (c).

to node by traversing edges one after the other in a classical way, but also taking into account a natural temporal constraint. More formally:

Definition 8 (Temporal walk). *Given a temporal graph $G = (V, E)$, a* temporal walk *from u to v (or a uv-walk for short) is a sequence of pairs of temporal edges and departure times $\langle (e_1, \tau_1), (e_2, \tau_2), \ldots, (e_k, \tau_k) \rangle$ such that:*

- *e_i is a temporal edge in E, for each $i \in [k]$,*
- *$\tau_i \in I_i$, where I_i is the availability interval of the temporal edge e_i, for each $i \in [k]$,*
- *$tail(e_{i+1}) = head(e_i)$, for each $i \in [k-1]$,* *(walk constraint)*
- *$\tau_{i+1} \geq arr_{e_i}(\tau_i)$, for each $i \in [k-1]$.* *(temporal constraint)*

Definition 9 (Temporal path). *A* temporal path *is a temporal walk through distinct nodes.*

Given a temporal walk $Q = \langle (e_1, \tau_1), (e_2, \tau_2), \ldots, (e_k, \tau_k) \rangle$ the *departure time* and *arrival time* of Q are defined as τ_1 and $arr_{e_k}(\tau_k)$, and denoted as $dep(Q)$ and $arr(Q)$, respectively. A temporal walk is said to be *strict* if $\tau_{i+1} > \tau_i$ for each $i \in [k-1]$; notice that temporal walks are strict in case of temporal graphs where the travel times are strictly positive, because $\tau_{i+1} \geq arr_{e_i}(\tau_i) = \tau_i + \lambda(\tau_i) > \tau_i$. We define the *overall waiting time* of Q as $\sum_{i=1}^{k-1} \tau_{i+1} - arr_{e_i}(\tau_i)$. We say that a node v is *temporally reachable* from a node u (or *u-reachable*) when there exists a *temporal walk* from u to v. We also say that the temporal walk *connects* u to v, and that the pair (u, v) is *temporally connected*. Moreover, we say that a temporal edge e with head v is *temporally reachable* from u (or *u-reachable*) when there exists an sv-walk ending with temporal edge e. Given a temporal graph $G = (V, E)$ we define its *underlying graph* as the digraph $D = (V, F)$, where $(u, v) \in F$ if and only if there exists a temporal edge $e \in E$ such that $tail(e) = u$ and $head(e) = v$. Moreover, in a temporal graph, we say that a node v is an *in-neighbour* (resp. out-neighbour) of u, if v is an in-neighbour (resp. out-neighbour) of u in the underlying graph. We define the *temporal in-degree* of a node u as the cardinality of the set of temporal edges that have head u, and the *temporal out-degree* as the cardinality of the set of temporal edges that have tail u. We define the arc activity of an arc (u, v) in the underlying graph as the cardinality of the set of temporal edges that have tail u and head v. Finally, we define the *lifetime* L of a temporal graph as the minimum length of an interval in $\mathbb{R}$ that contains the union of the availability intervals of the temporal edges.

Since the models that we use the most in this work are point availability and uniform strict temporal graphs, we will introduce here some notation that will be useful in the next chapters. In these two models each temporal edge e has a unique available departure time τ, which is denoted by $dep(e)$ and thus also a unique arrival time, $\tau + \lambda$ that is denoted by

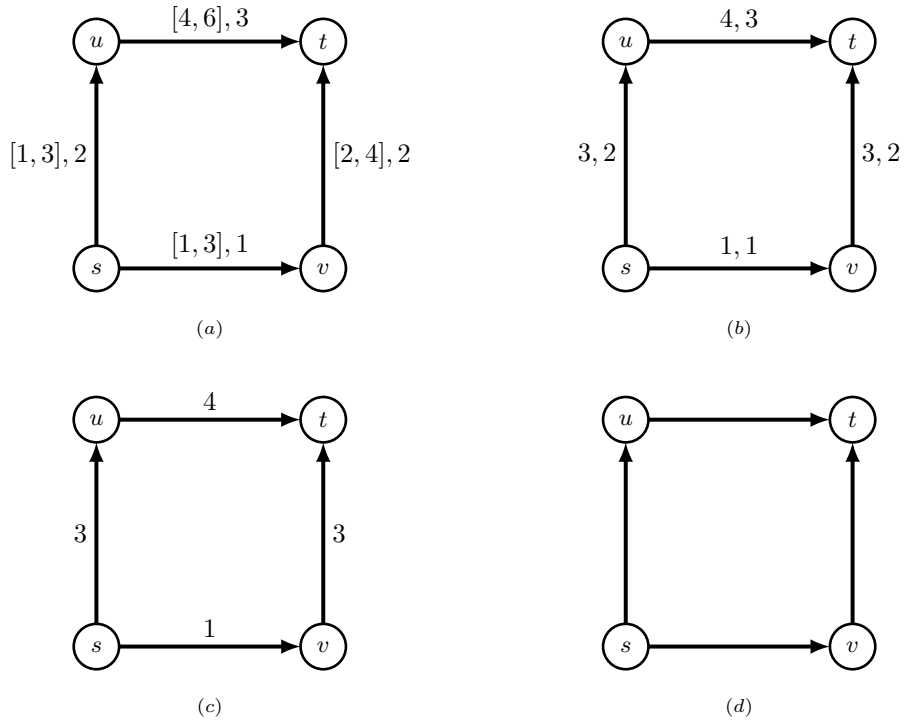

Figure 1.2: An example of piecewise constant temporal graph (a), of point availability temporal graph (b) and of uniform strict temporal graph (c). Each of these temporal graphs has the same underlying graph (d).

$arr(e)$. In the case of strict uniform temporal graphs, in particular, $arr(e) = \tau+1$. Moreover, whenever we consider a temporal walk $\langle (e_1, \tau_1), (e_2, \tau_2), \dots (e_k, \tau_k)\rangle$ in these models we can drop the time labels and use $\langle e_1, e_2, \dots, e_k\rangle$, since each τ_i must be equal to the time at which e_i is available.

Example Figure 1.2 represents three examples of temporal graphs that have the same underlying graph. The pair (s, t) is temporally connected in all the examples. A walk that connects s to t in the piecewise constant temporal graph (a) is $Q_a = \langle((s, u, [1, 3], 2), 1), ((u, t, [4, 6], 3), 5)\rangle$. The departure time of Q_a is 1 and the arrival time is $5 + 3 = 8$. The overall waiting time of Q_a is 2, as the walks arrives in u at time 3 and leaves the node at time 5. In the point availability temporal graph (b) it is not possible to connect s to t using a temporal walk that goes through node u. Indeed, the sequence of temporal edges $\langle(s, u, 3, 2), (u, t, 4, 3)\rangle$ does not respect the temporal constraint in the definition of temporal walk. Notice that the travel times of the corresponding temporal edges of the graphs in (a) and (b) are the same, however the greater availability of the temporal edges in (a) allows more flexibility which makes it possible to reach t from s going through u. Nonetheless, $Q_b = \langle(s, v, 1, 1), (v, t, 3, 2)\rangle$ is a temporal walk that connects s to t in (b). The departure time of Q_b is 1, the arrival time is $3 + 2 = 5$ and the overall waiting time is 1, as the walks arrives in node v at time 2 and leaves towards t at time 3. Finally, in the uniform

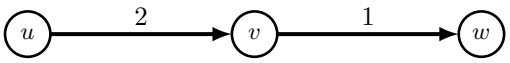

Figure 1.3: A uniform temporal graph where (u, v) and v, w are temporally connected but (u, w) is not.

strict temporal graph (c), unlike (b), it is possible to connect s to t by a temporal walk through node u. Notice that the departure times of the temporal edges in graphs (b) and (c) are the same. However the greater travel time from s to u in (b) is greater than the one in (c), and that informally makes the temporal edge arrive too late in u to *'catch'* the next temporal edge. Thanks to the lower travel time in (c) there exists a temporal walk $Q_c = \langle (s, u, 3), (u, t, 4) \rangle$, with departure time 3, arrival time 4 that connects s to t going through u. Notice that the overall waiting time of Q_c is 0, as it arrives in node u at time 4 and leaves it at the same time.

1.1.6 Differences with static graphs

Working on temporal graph algorithms raises some challenges that are not present in classical digraphs. This is due to the fact that some properties holding in digraphs do not hold in the temporal case anymore. In the following we will discuss a few such properties in the context of paths by providing some simple counter-examples in the case of uniform temporal graphs. Because of the hierarchy defined earlier, this means that such properties do not hold even in the other (more general) models of temporal graphs we defined.

Transitivity of reachability In digraphs it is possible to rely on the transitivity of reachability: given three nodes u, v and w, if v is reachable from u and w is reachable from v then w is reachable from u. The reason for this is that it is possible to concatenate a walk from u to v and a walk from v to w into a single walk from u to w. However, this is not true for temporal walks any more. In the temporal graph displayed in Figure 1.3, there exists a temporal path, made of a single temporal edge, from u to v, and another one from v to w. However, there exists no temporal walk from u to w, since the sequence $\langle (u, v, 2), (v, w, 1) \rangle$ is not a temporal walk.

Prefix optimality of shortest paths Whenever a shortest path from a source to a certain destination is given in a digraph, we can rely on the fact that any prefix of such a path is also a shortest path. If it was not the case we would be able to find a shorter path from the source to the destination.

In temporal graphs the notion of "shortest" has a wide variety of interpretations, that we will discuss in more detail in the following sections and Chapter 2. Without going too

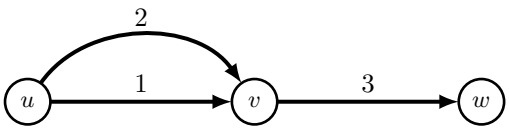

Figure 1.4: An example of a uniform temporal graph showing that the prefix of a temporal path with minimum arrival time does not necessarily have minimum arrival time itself.

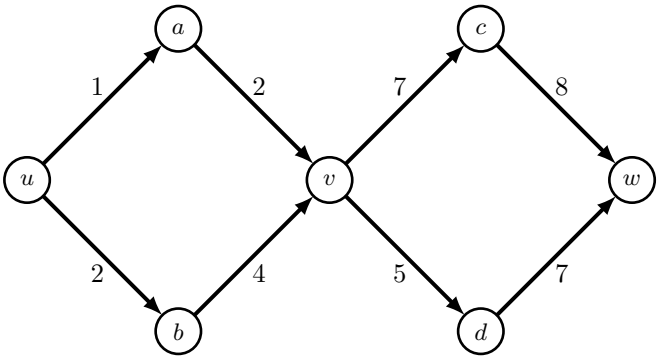

Figure 1.5: An example of a uniform temporal graph showing that the prefix of a temporal path with minimum duration may not have minimum duration itself.

much into details, one of the possible ways to define a "shortest" temporal path is as a temporal path that minimises arrival time to the destination. Let us consider the temporal graph from Figure 1.4. Here the temporal path $\langle (u, v, 2), (v, w, 3) \rangle$ has minimum arrival time among the temporal paths from u to v. However its prefix, i.e. the temporal edge $(u, v, 2)$, does not constitute a shortest temporal path from u to v since the temporal path made of the sole temporal edge $(u, v, 1)$ has an inferior arrival time.

One could argue that in this model, for this definition of "shortest", it would always be possible to choose a shortest temporal path such that its prefixes are also shortest temporal paths. This is indeed true, and in the figure this temporal path would be $\langle (u, v, 1), (v, w, 3) \rangle$. However, this is not valid anymore for other definitions of "shortest" temporal path present in the literature. One other classical criterion to define a "shortest" temporal path P is to minimise the duration, namely $arr(P) - dep(P)$. In the temporal graph represented in Figure 1.5 the temporal path with minimum duration from u to v is $\langle (u, b, 2), (b, v, 4), (v, d, 5), (d, w, 7) \rangle$, with a duration of $8 - 2 = 6$. And, according to this definition, this is the unique shortest temporal path from u to w. Its prefix $\langle (u, b, 2), (b, v, 4) \rangle$, on the other hand, does not have minimum duration among the temporal paths from u to v, as $\langle (u, a, 1), (a, v, 2) \rangle$ has duration $3 - 1 = 2$. Moreover, notice that the temporal paths with minimum duration from u to v and from v to w, $\langle (u, a, 1), (a, v, 2) \rangle$ and $\langle (v, c, 7), (c, w, 8) \rangle$ respectively, could be concatenated into a temporal path from u to w, but this would not lead to a shortest temporal path.

1.2 State of the art

1.2.1 Models

Temporal graphs arose as an extension of graphs from the need to develop a tool that better captures interactions which are dynamic and change over time. This concept of time dependent network finds applications in a variety of domains. This led to a fragmented literature about the topic, which progressed at different points in time, pushed by different applications. In particular there are plenty of models, some of which are closely related, and the notation between those is not always consistent.

One of the first models of temporal graph, and also one of the most flexible, was designed

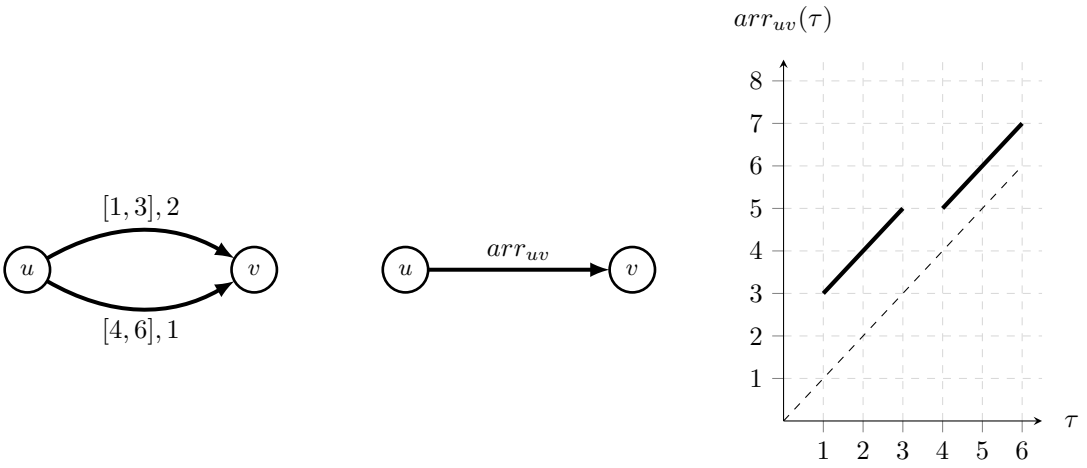

Figure 1.6: A piecewise constant temporal graph on the left, with two temporal edges sharing head and tail. The corresponding arrival time functions are merged into a single arrival time function associated to the arc (u, v).

in the context of road networks [19, 29, 43] under the name of time-dependent network and was studied in [52, 21, 22, 35, 23]. Different models have since been developed in the context of gossiping [14, 40], mobile and distributed networks, as well as social networks, under the name of evolving graphs [13], temporal networks [42, 46], time-varying graphs [17], and also in the setting of public transit networks, where the model is called edge-scheduled network [6], timetable [26, 25] or temporal graph [63]. Let us see how the models that we introduced in Section 1.1.4 relate to the ones present in the literature.

Let us consider the assumption that the availability intervals of temporal edges with the same tail and head are disjoint. In this case, it is then possible to merge the travel time or arrival time functions of such edges into a single function. More precisely, let us consider the temporal edges e such that $tail(e) = u$ and $head(e) = v$, and the respective arrival time functions arr_e. It is then possible to define by cases the function $arr_{(u,v)} : \mathbb{R} \to \mathbb{R}$, by defining $arr_{(u,v)}(\tau) = arr_e(\tau)$ when τ belongs to the availability interval of e and $arr_{(u,v)}(\tau) = +\infty$ for each τ that does not belong to any availability interval of a temporal edge from u to v. In the case of piecewise linear (resp. constant) temporal graphs, the functions $arr_{(u,v)}$ are thus piecewise linear (resp. constant). See Figure 1.6 for an example in the case of a piecewise constant temporal graph.

Dehne, Omran and Sack [23] and Foschini, Hershberger and Suri [35] define a time-dependent network starting from a digraph, that in our case would correspond to the underlying graph, and assigning to each arc (u, v) the arrival time function $arr_{(u,v)}$.

Xuan, Ferreira and Jarry [13] define an evolving graph as a sequence of subgraphs $G_1, \ldots, G_T$ of a digraph G, their union corresponds to the underlying graph in our case. An arc (u, v) in G_τ would correspond in our model to a temporal edge from u to v that is available at time τ. In a variant of their model they group in intervals arcs appearing in consecutive subgraphs, such intervals would correspond to our availability intervals.

In the link stream model by Latapy, Viard and Magnien [45], they define the temporal graph as a set of times T, a set of nodes V and a set of links $E \subseteq T \times V \times V$. Two nodes u and v are linked at time τ if $(\tau, \{u, v\})$ belongs to E. When describing an instance,

they represent the links in E as a union of time intervals $I_1 \cup \cdots \cup I_k$ together with a (unordered) pair of nodes $\{u, v\}$. Moreover, as there is no notion of travel time, and based on the definition they give of paths, such a model is close to our piecewise constant temporal graphs where the travel time is zero for all temporal edges. They also define a more general model, called stream graph, in which nodes can be unavailable during some interval of time.

The edge-scheduled network model considered by Berman [6], up to slight differences in notation, corresponds to the point availability temporal graph model presented here, while the temporal graph model considered by Wu, Cheng, Huan, Ke, Lu and Xu [63] corresponds exactly to the point availability temporal graph model as we defined it. The timetable model from Dibbelt, Pajor, Strasser and Wagner [26] and Delling, Pajor and Werneck [25] encompasses the point availability model, and is closer to the application of public transport. Indeed, there are connections corresponding to temporal edges available at a single point in time, that are grouped into trips, which represent a scheduled vehicle moving in the underlying graph. There are also footpaths, which correspond to temporal edges that are available at any time instant with a constant travel time.

The temporal networks model from Mertzios, Michail and Spirakis [46] is a static graph $G = (V, E)$ labelled with a function $\lambda : E \to 2^{\mathbb{N}}$ that assigns a set of time labels to each arc. Given their definition of temporal path that requires strictly increasing time labels, this model is equivalent to the strict uniform temporal graphs defined before.

1.2.2 Temporal graph representations

When it comes to developing algorithms on temporal graphs, the way the graph is represented and given as input plays a key role. Different representations of the same model could lead to algorithms with different time and space complexities. Let us see some of the representations used in the literature.

Piecewise linear temporal graphs:

Arc arrival functions. A temporal graph $G = (V, E)$ is given through a classical representation (e.g. adjacency list) of the underlying graph $D = (V, F)$. On top of that, each arc (u, v) is associated with a piecewise linear arrival time function $arr_{(u,v)}$ that associates to each available time instant τ of a temporal edge e from u to v the corresponding arrival time $arr_e(\tau)$. This representation was used for example by Dehne, Omran and Sack [23].

Piecewise constant temporal graphs:

Adjacency lists. For each node u there is a list storing the nodes v such that there exists at least a temporal edge from u to v. Each such neighbour v is associated to the sorted list of availability time intervals of temporal edges from u to v, with the corresponding travel times. This representation was introduced by Xuan, Ferreira and Jarry [13].

Point availability temporal graphs:

List of quadruples. The whole temporal graph $G = (V, E)$ is simply stored as a list containing a quadruple $e = (u, v, \tau, \lambda)$ for each temporal edge $e \in E$. This list can additionally be sorted either by departure time or arrival time of the temporal edges, in this case we say that it is *pre-sorted*. Among others, this representation was used by Dibbelt, Pajor, Strasser and Wagner [26] and Wu, Cheng, Huan, Ke, Lu and Xu [63].

Time-expanded graph. The idea behind a time-expanded representation tracks back to the work of Étienne-Jules Marey [32] in 1885. In the context of temporal graphs it was first fromalised by Ford and Fulkerson [34] and it consists in transforming the temporal graph into a static graph by introducing a copy of each node for each possible time instant. Each temporal edge is then turned into an arc from the two corresponding copies of its tail and head. In this work we will consider a variant where we introduce copies of a node only for time instants corresponding to a departure time of an edge from that node, or an arrival time of an edge to that node, following the approach of Schulz, Wagner and Weihe [56]. This allows to use a similar amount of space as the previous list representation up to a constant factor. It is directly available in applications such as contact tracing where the temporal graph is obtained by gathering traces from all nodes, and where each trace is a recording of all edge events at the node. Although additional sorting is required to obtain a global ordering of temporal edges according to time, it does provide a local ordering at each node.

Since we are going to use this representation in the next chapters we will also give a formal definition here. Given a temporal graph $G = (V, E)$, its *time-expanded representation* is a directed graph $D = (W, F^c \cup F^w)$, where:

- The nodes in W are labelled nodes v_τ, where $v \in V$ refers to a node of G and τ is a time label. More precisely, $v_\tau \in W$ if and only if there exists a temporal edge in E with tail v and departure time τ or a temporal edge with head v and arrival time τ. We will also refer to such nodes as *copies of* v. Let us denote with $Pred^w(v_\tau)$ the copy of v in W with maximum time label less than τ, if it exists.

- We distinguish two types of arcs F^c and F^w called connection arcs and waiting arcs respectively. The set F^c contains an arc $(u_\tau, v_{\tau+\lambda})$ for each temporal edge $e = (u, v, \tau, \lambda) \in E$. These arcs represent a temporal connection between the nodes in V and are called *connection arcs*. Note that each arc (v_τ, w_ν) in F^c satisfies $\tau \leq \nu$, since travel times are non-negative. The set F^w is defined to contain an arc $(Pred^w(v_\tau), v_\tau)$ for each $v \in V$ and for each copy v_τ of v such that $Pred^w(v_\tau)$ is defined. These arcs represent the possibility to wait at a node $v \in V$ during a walk in G and are called *waiting arcs*. Note that each arc (v_τ, v_ν) in F^w satisfies $\tau < \nu$. As we allow temporal edges which are self loops (i.e. edges (v, v, τ, λ)), there might exist two copies of an arc in D, one in F^c and one in F^w. Formally, D is thus a directed multidigraph as we distinguish arcs in F^c from those in F^w and assume $F^c \cap F^w = \emptyset$.

Example The time-expanded representation of the temporal graph in Figure 1.5 is displayed in Figure 1.7. Notice that it is possible to associate to each path in the time-expanded representation, which is a static graph, a temporal walk in the temporal graph and viceversa. For example, let us consider the path $\langle (u_1, a_2), (a_2, v_3), (v_3, v_5), (v_5, d_6), (d_6, d_7), (d_7, w_8) \rangle$ in the static graph. We can associate it to the temporal walk from u to w $\langle (u, a, 1), (a, v, 2), (v, d, 5), (d, w, 7) \rangle$, in a way that each connection arc in the path corresponds to a temporal edge in the temporal walk, and each waiting arc between consecutive copies of a node corresponds to time spent waiting in that node. With a similar reasoning we can associate any temporal walk in the temporal graph to a walk in the time-expanded static digraph. Note that when travel times are positive, the static walk is necessarily a path.

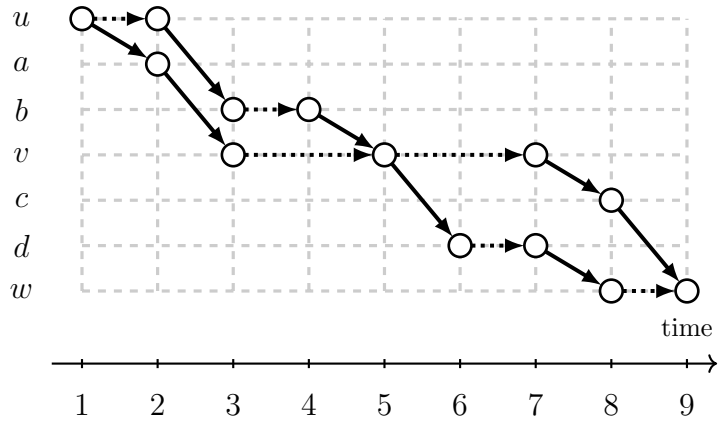

Figure 1.7: The time-expanded representation of the uniform strict temporal graph in Figure 1.5.

Uniform temporal graphs:

List of triples. The whole temporal graph $G = (V, E)$ is stored simply through a sorted list containing a triple $e = (u, v, \tau)$ for each temporal edge $e \in E$. Again, such a list can additionally be sorted by the time labels of the temporal edges. This representation is particularly suitable for algorithms that need to perform a linear scan of the temporal edges, for example the one designed by Kempte, Kleinberg and Kumar [42].

Snapshots. Let us consider the set T of time labels τ such that there exists at least one temporal edge with label τ. It is possible to consider for each node u an array A_u of length T, such that every entry $A_u[\tau]$ stores a pointer to a linked list of the nodes v such that there exists (u, v, τ). This representation was used by Mertzios, Michail and Spirakis [46].

Adjacency lists. For each node u there is a list storing the nodes v such that there exists at least one temporal edge from u to v. Each such neighbour v is associated to the sorted list of availability time instants of temporal edges from u to v. This representation was introduced by Ferreira and Viennot [33].

1.2.3 FIFO property and waiting policies

A property of temporal graphs which is well discussed in the literature, and most of the time is assumed, is the so called *FIFO (First In First Out) property*. Informally, it states that a later departure from a node u heading to a node v using a temporal edge results in a later (or equal) arrival time. Let us assume the availability intervals of the temporal edges are disjoint, and let us merge the arrival time functions of temporal edges between same nodes into a single arrival time function $arr_{(u,v)}$, as explained before. In this setting the FIFO property means that for any pair of nodes u and v, for $\tau' > \tau$ then $arr_{(u,v)}(\tau') \geq arr_{(u,v)}(\tau)$, namely $arr_{(u,v)}$ is non-decreasing.

The definitions of temporal walk and temporal path, and the examples given so far, belong to the *unrestricted waiting* framework. This means that during a temporal walk it is

possible to wait an unbounded quantity of time at any node between a temporal edge and the next one. Indeed, by our definition, given a temporal walk $\langle (e_1, \tau_1), (e_2, \tau_2), \ldots, (e_k, \tau_k) \rangle$ we required that $\tau_{i+1} \geq arr_{e_i}(\tau_i)$ for any two consecutive temporal edges. This means that the time spent waiting in the head of e_i, namely $\tau_{i+1} - arr_{e_i}(\tau_i)$, is not restricted and can be any non-negative value.

Notice that when the FIFO property is not satisfied, a temporal path minimising the arrival time might involve some time spent waiting in the nodes along the path, as it might be convenient to wait until the arrival time function has a lower value. This suggests why it is necessary to assume the FIFO property in algorithms that follow a Dijkstra-like approach for temporal paths, like the one that we are describing in Section 1.2.4.

However, as explained by Orda and Rom [51], in a temporal graph with unrestricted waiting that does not satisfy the FIFO property, it is possible to modify the travel time functions in a way such that in the resulting temporal graph the earliest arrival times are preserved and it is never convenient to wait in the nodes.

In the literature, models deviating from the unrestricted waiting have been investigated, taking into account different kinds of waiting policies.

Orda and Rom [50] studied, among several cases, the *forbidden waiting policy* in which it is not possible to wait at all at a node during a temporal walk. In other words, after reaching a node using a temporal edge, it is necessary to take the following temporal edge in the sequence leaving the node at the very same time it was reached. In particular they proved a hardness result which can be stated as follows.

Theorem 1 [50]. *Given a piecewise constant temporal graph $G = (V, E)$ with forbidden waiting, a departure time τ, a source node s and a destination node d, computing the earliest arrival time among the walks from s to d departing at time τ is NP-hard.*

A simpler and shorter proof is given by Sherali, Ozbay and Subramanian [57]. It works even when waiting is allowed at the source. A reduction from the partition problem is formulated in the following way. Let us first recall a formulation of the partition problem.

> PARTITION PROBLEM. Given n non-negative integers $x_1, x_2, \ldots, x_n$ such that $\sum_{i=1}^{n} x_i = 2S$, does there exists a partition of these numbers into two sets such that the sum of each set equals S?

We present their result with a very slight modification in the domains of the travel time functions, so that their reduction proves that even computing whether there exists a temporal walk from the source to the destination is NP-hard.

Given an instance of the partition problem they build a temporal graph $G = (V, E)$, where the set of vertices is $V = \{v_0, v_1, \ldots, v_{n+1}\}$, the source is $s = v_0$ and the target $t = v_{n+1}$. There exist two temporal edges from the source s to v_1, both available only at time zero, one with travel time x_1 and the other with travel time zero. Then, for each $i \in \{1, \ldots, n\}$ there exist two temporal edges from v_i to v_{i+1} with availability interval $[0, S]$, one with constant travel time x_i and the other with constant travel time zero. Finally, there exists a temporal edge from v_n to t that is available only at time instant S with travel time zero. The temporal graph obtained this way is represented in Figure 1.8. The idea is that it is possible to reach t from s with a temporal walk if and only if there exists a temporal walk departing from s at time zero, and arriving in t at time S. In particular, due to the choice of travel times, such a walk corresponds to choosing a subset $X \subseteq \{x_1, \ldots, x_n\}$ such that $\sum_{x_i \in X} x_i = S$, and thus a partition that solves the problem.

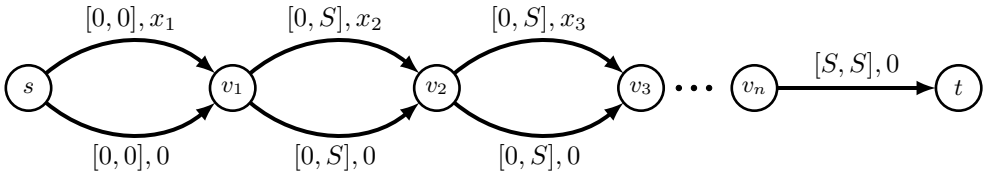

Figure 1.8: Representation of the reduction from the partition problem to the computation of earliest arrival time in piecewise constant temporal graphs with forbidden waiting.

Another interesting result in the literature about temporal paths hardness was provided by Casteigts, Himmel, Molter and Zschoche [18]. The authors worked on a model that corresponds to an *undirected* version of a uniform non-strict temporal graph. In the undirected setting, a temporal edge that connects node u to node v at time τ also connects node v to node u at the same time. They focused on the waiting policy called Δ-*restless*, which means that in a temporal walk, in an intermediate node, it is possible to wait at most Δ units of time before moving to the next one. Let L be the lifetime of the temporal graph. The result is formulated in a slightly weaker version for the sake of simplicity.

Theorem 2 [18]. *Given a uniform non-strict temporal graph $G = (V, E)$, a source node s and a target node t, an integer $\Delta \geq 1$ such that $\Delta + 2 \leq L$, it is NP-complete to determine whether there exists a temporal path from s to t.*

Notice that this result, unlike the previous one from Orda and Rom, concerns exclusively temporal paths and not temporal walks. Indeed, temporal walks under waiting time constraints in this model are tractable, as shown by Bentert, Himmel, Nichterlein and Niedermeir [5]. More details about this problem and related results are given in Chapter 2.

1.2.4 Temporal paths problems

When it comes to temporal graphs there are several paths and connectivity problems that received attention in the literature. Let us summarize some of the most notable ones.

EARLIEST ARRIVAL TIME. Given a source node s and a destination node t, compute the minimum arrival time among the temporal walks from s to t.

SHORTEST DURATION. Given a source node s and a destination node t, compute the minimum duration among the temporal walks from s to t, where the duration of a temporal walk Q is defined as $dur(Q) = arr(Q) - dep(Q)$.

PROFILE. Given a source node s and a destination node t, compute a representation of the function $f_{st} : \mathbb{R} \to \mathbb{R}$ that associates to each time τ the earliest arrival time among st-walks departing at time τ or later.

FEWEST HOPS. Given a source node s and a destination node t, compute the minimum number of edges that compose a temporal walk from s to t.

See Figure 1.9 for an example of a uniform strict temporal graph with two nodes s and t. An example of an earliest arrival temporal path is $\langle (s, a, 1), (a, b, 3), (b, t, 4) \rangle$, which has

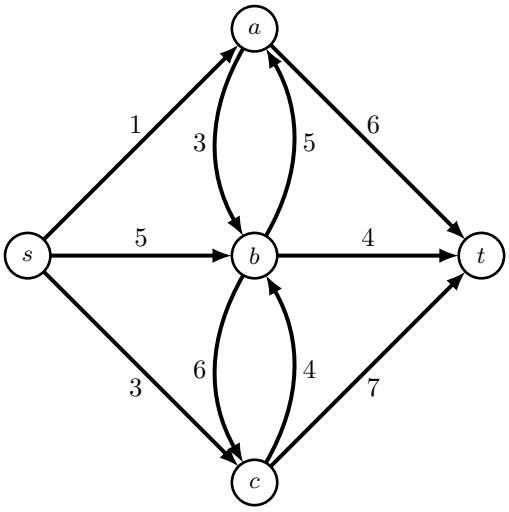

Figure 1.9: An example of a uniform strict temporal graph with different optimal walks for different criteria.

arrival time $4+1=5$. A shortest duration temporal path is $\langle(s,b,5),(b,c,6),(c,t,7)\rangle$, which has duration $7+1-5=3$. A temporal path with fewest hops is $\langle(s,a,1),(a,t,6)\rangle$ which has two temporal edges. Finally, the profile function is given by:

$$f_{st}(\tau)=\begin{cases}5 & \text{for } \tau \leq 1\\ 7 & \text{for } 1<\tau\leq 3\\ 8 & \text{for } 3<\tau\leq 5\\ +\infty & \text{for } \tau>5\end{cases}$$

For example, $f_{st}(2)=7$ because it is possible to wait at the source node until time 3 and then use the temporal path $\langle(s,c,3),(c,b,4),(b,a,5),(a,t,6)\rangle$ which has arrival time 7.

All the problems listed above are given in the respective *single-source single-destination* formulation, which in the literature can also be referred to as *one-to-one*. However, each of them could also be defined in the formulations *single-source all-destinations*, *all-sources single-destination* and *all-source all-destinations*, that can appear in the literature as *one-to-all*, *all-to-one* and *all-to-all* respectively.

We can now summarize several algorithmic results in the state of the art concerning these problems in all the models described before (see Table 1.1). In order to express the complexity of the algorithms we need to define a few parameters of a temporal graph $G=(V,E)$:

- n : the number of nodes, i.e. $|V|$,
- m : the number of arcs in the underlying graph,
- M : the number of temporal edges, i.e. $|E|$,
- Δ : the maximum of temporal in-degree and out-degree among the nodes in the temporal graph, this parameter is bounded by M,

- η : the maximum arc activity over the arcs in the underlying graph, this parameter is bounded by M,
- D : the "diameter" of the temporal graph in terms of hops, informally the maximum number of temporal edges needed to connect a pair of nodes, which is bounded by n in the case of unrestricted waiting,
- P : the maximum number of parameters needed to represent the profile function between any pair of nodes.

All results in the table refer to the single-source all-destinations version of the problems, except for the shortest duration and profile in the piecewise linear model that are single-source single-destination. Moreover, all results refer to unrestricted waiting models. Especially the algorithms in the piecewise constant and piecewise linear models, rely on the FIFO property of the arrival functions of edges.

Problem	Temporal graph model			
	Uniform strict	Point availability	Piecewise constant	Piecewise linear
Earliest arrival time	$O(M)$[46]	$O(M)$[26, 63]	$O(m(\log\eta + \log n))$[13]	$O(m\log\eta + n\log n)$[52, 22]
Shortest duration	$O(M)$	$O(M)$[64]	$O(M(m + n\log n))$[23]	$O(M(n\log n + m))$[35]
Profile	$O(M)$[44]	$O(M\log\Delta)$[26] $\boldsymbol{O(M)}$	$O(M(m + n\log n))$[23] $\boldsymbol{\Omega(Mm)}$	$O((P + M)(m + n\log n))$[23] $\Omega(n^{\log n})$[35]
Fewest hops	$O(M)$	$O(M\log\Delta)$[63] $O(M)$ [65]	$O(MD)$[13]	?

Table 1.1: State of the art for shortest paths in temporal graphs. Our contribution is shown in bold.

We will see a brief description of two results in the table, in order to have a first approach on algorithms designed to solve problems in temporal graphs.

Temporal Dijkstra in piecewise constant temporal graphs A common technique used to solve the earliest arrival time problem in models like piecewise constant and piecewise linear temporal graphs, is a temporal variant of Dijkstra's algorithm. Here we are going to see how this algorithm works in the case of piecewise constant temporal graphs, given an adjacency list representation as input, according to Xuan and Ferreira [13]. They assume the FIFO property on the temporal graph: in this setting, if v is reachable from u there exists an earliest arrival time path from u to v such that each prefix of such a temporal path has itself minimum arrival time among the other temporal paths ending in the same node as the prefix. The algorithm performs a Dijkstra-like procedure, with a different way of computing distances. In this scenario the distance corresponds to the earliest arrival time at each node. Whenever a node u, with associated earliest arrival time τ, is extracted from the priority queue, the algorithm computes the arrival time to the out-neighbours leaving u at time τ. For that, the algorithm performs a binary search among the intervals of availability of temporal edges from u and each out-neighbour v. The FIFO property plays a key role here, because it guarantees that the first time instant $\tau' \geq \tau$ in which it is possible to leave u towards a neighbour is also the one that minimises the arrival time through the

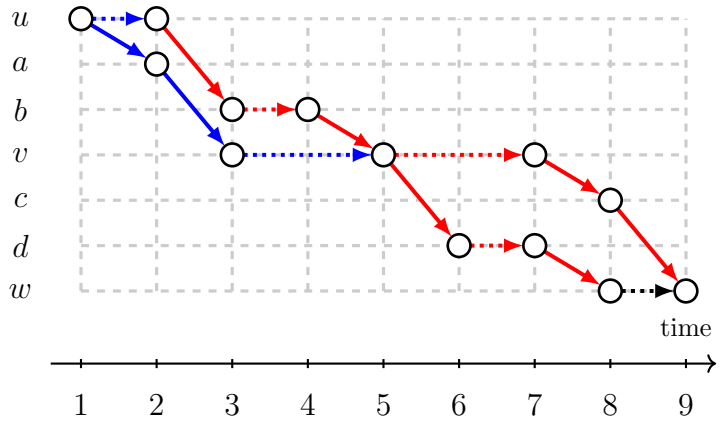

Figure 1.10: The time-expanded representation of the uniform temporal graph in Figure 1.5. The nodes on the top of the digraph correspond to the copies $\{u_1, u_2\}$ of the source u. In red there is the first BFS arborescence computed from u_2 and in blue the second BFS arborescence computed from u_1.

corresponding edge. The correctness of the algorithm follows from similar arguments as for Dijkstra's algorithm and the logarithmic factor $\log \eta$ in the time complexity is due to the binary search needed to compute the correct distance.

Solving shortest duration in point availability temporal graphs Wu, Cheng, Huan, Ke, Lu and Xu [64] designed an algorithm that takes as input a time-expanded representation of a point availability temporal graph and solves the shortest duration problem in linear time. We describe the algorithm and show an example of execution using the temporal graph in Figure 1.5 and its time-expanded representation which is displayed in Figure 1.7.

Let u be the source for the shortest duration problem. Let us consider the copies of the source sorted by decreasing time label. In the example in Figure 1.7 those would be $\{u_2, u_1\}$. The algorithm picks the first source copy and performs a BFS search from there. From the paths in the BFS arborescence it obtains the earliest arrival time starting from the source at the time corresponding to the label of the source copy. In Figure 1.10 the first BFS computed by the algorithm starting from node u_2 is represented in red. From the paths in the BFS arborescence rooted in u_2, it obtains the earliest arrival times of the temporal walks departing from u at time 2.

For example, from u_2, there is a path in the time-expanded representation that belongs to the BFS arborescence that reaches a copy of v, such path is $\langle (u_2, b_3), (b_3, b_4), (b_4, v_5) \rangle$. This path corresponds to the temporal path $P = \langle (u, b, 2), (b, v, 4) \rangle$ from s to u which has earliest arrival time among the temporal paths from u to v with departure time 2. Then, the algorithm computes the duration of the corresponding temporal paths, and this leads to some tentative solutions of shortest duration temporal paths. In the example, the duration of P is $4 + 1 - 2 = 3$, thus we know that the shortest duration from u to v has to be less or equal to 3. Then it picks the next copy of u, u_1 in our example, performs a new BFS search from there, stopping whenever it encounters a node that has already been visited in a previous iteration. The BFS search from u_1 visits node v_5, which has already been visited by the previous BFS, the algorithm thus would stop computing this branch of the

BFS arborescence in v_5. This second BFS search is represented in blue in Figure 1.10. Then, it compares the duration corresponding to each new path with the previous tentative solution, and updates it by keeping the one with lower duration. The path from u_1 to v_3 found in this iteration through the BFS search from u_1 corresponds to the temporal path $P' = \langle (u, a, 1), (a, v, 2) \rangle$ which has duration $2 + 1 - 1 = 2$, thus it improves the tentative solution found so far, and the tentative shortest duration from u to v is updated to 2. After repeating the process for each copy of the source, the tentative solutions correspond indeed to the shortest duration. The algorithm visits each arc in the time expanded graph at most once, and thus takes linear time.

Finally, notice that all temporal paths corresponding to paths in the time expanded representation we would find by not stopping the BFS in the moment it visits the copy of a node that was already found in a previous iteration, would have higher duration than the ones we already found. In the first BFS search from u_2 the algorithm finds a certain path to w_8. If the BFS search from u_1 would not be stopped in v_5, it would also find a path, now from u_1 instead of u_2, to w_8. The corresponding temporal walks have the same arrival time, which is 8, as the time label of w_8. However, the first has a later departure time than the second (and possibly of the next temporal walks identified to w_8 in the following iterations), and thus a shorter duration.

Chapter 2

Temporal walks computation under waiting constraints

This chapter is based on the results obtained in [**9, 12, 11**]. As the algorithmic results in [**12**] improve those in [**9**] the latter will not be presented here. However, the model and the algebraic cost structure, that allows us to solve in a very general way a great variety of temporal walk optimization problems, was first introduced in [**9**].

As we hinted in Chapter 1, while the notion of "shortest" path is fairly standard for static graphs, there exists several natural extensions for defining "shortest" temporal walks. Indeed, the following natural criteria can be optimized: earliest arrival time, shortest duration, or fewest number of edges, to the name most popular ones. Most criteria result in optimal temporal walks which are indeed temporal paths. Minimizing overall waiting time at nodes is a notable exception where walks obviously help compared to paths, as performing a loop instead of waiting in a node reduces the overall waiting time.

Indeed, optimizing different criteria might appear as different problems and the single-source optimal temporal path/walk problem has mostly been addressed with different algorithms for different criteria. A further level of difficulty happens when the waiting time at each node is bounded: the computation of temporal paths becomes NP-hard, as reported in Theorem 2. However, as discovered in [5], by focusing on temporal walks instead, it is still possible to work on connectivity and optimization problems in polynomial time. Indeed, in this chapter we will consider point availability temporal graphs with waiting restrictions as defined in [5]: for each node there is a minimum and a maximum amount of time that is possible to wait in it during a walk. See Figure 2.1 for an example of a point availability temporal graph with waiting time constraints and examples of "shortest" temporal walks for different constraints, including *shortest-fastest*, which is a temporal walk that, among temporal walks with minimum duration, is the one with fewest hops. Such a combination is proposed in [45] for defining a temporal version of betweenness centrality.

Related work. Interestingly, after numerous works inspired by Dijkstra's algorithm (see e.g. [6, 8, 49, 53]), a linear-time algorithm for earliest arrival time in a point availability temporal graph, with unrestricted waiting policy, was first claimed in [63, 64]. The algorithm, operates through a single scan of temporal edges ordered by non-decreasing departure time, as done in [26, 27]. Both works assume positive travel times, ensuring that temporal paths are strict. Single-source shortest duration temporal paths are then obtained by basi-

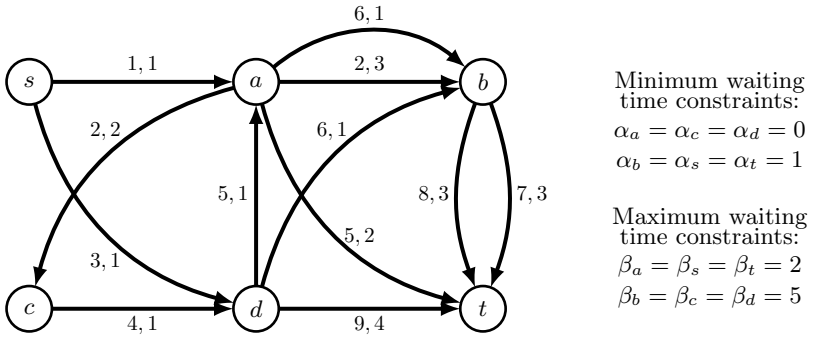

Figure 2.1: A point availability temporal graph with waiting constraints. Different temporal walks from s to t optimize different criteria: $\langle (s, a, 1, 1), (a, b, 2, 3), (b, t, 7, 3)\rangle$ has earliest arrival time, $\langle (s, d, 3, 1), (d, t, 9, 4)\rangle$ has fewest hops, $\langle (s, d, 3, 1), (d, a, 5, 1), (a, b, 6, 1), (b, t, 8, 3)\rangle$ has shortest duration, $\langle (s, a, 1, 1), (a, c, 2, 2), (c, d, 4, 1), (d, a, 5, 1), (a, b, 6, 1), (b, t, 8, 3)\rangle$ has minimum overall waiting time, $\langle (s, d, 3, 1), (d, b, 6, 1), (b, t, 8, 3)\rangle$ is shortest-fastest.

cally solving the profile problem with a more intricate version of the scanning algorithm [27] that takes $O(M \log \Delta)$ time where M is the total number of temporal edges and $\Delta \leq M$ is the maximum number of temporal edges with same head. A linear time algorithm for shortest duration, the one explained in Chapter 1, was later included in [64] by taking as input a time-expanded representation of the temporal graph. This time-expanded approach is also used for temporal paths with fewest number of edges in [65] in linear time. However, this approach uses a modified time-expanded representation where all node events at the source are contracted to a single node, preventing it to support some other criteria such as shortest duration. Overall, it was thus unclear whether linear time was possible for all classical criteria (including overall waiting time) and possible combinations of them, and a unifying linear-time algorithm was still missing. Despite the hardness result for temporal path computation [18], a recent break-through [5] shows that computing single-source optimal walks is still possible under such waiting-time constraints in $O(M \log M)$ time. Similarly to optimizing overall waiting time, such constraints indeed impose to switch from paths to walks for other criteria also. The algorithm is generic in the sense that it optimizes a linear combination of all classical criteria, although lexicographic combinations such as shortest-fastest are not supported. Apart from an initial sorting of the temporal edges, the $\log M$ factor comes from using several calls to Dijkstra's algorithm on graphs that can have up to $\Theta(M)$ nodes and $\Theta(M)$ edges. The high number of nodes comes from a preliminary transformation introducing a dummy node for each temporal edge to obtain an equivalent temporal graph where all travel times are zero. This leaves open the quest for a simpler algorithm working directly on the original temporal graph, and whether linear time is possible for any criterion under waiting-time constraints. Notice that the techniques used in [64, 65] that rely on the time-expanded representation cannot be replicated in the context of waiting restrictions. Indeed, we lose the correspondence between paths in the time-expanded and temporal walks in the temporal graph, that we have in the unrestricted waiting model. See Figure 2.2 for an example: the path $s_1, a_2, a_6, b_7, b_8, t_{11}$ would correspond to the sequence of temporal edges $\langle (s, a, 1, 1), (a, b, 6, 1), (b, t, 8, 3)\rangle$, however the latter is not a temporal walk as it does not satisfy the waiting time constraint at node a.

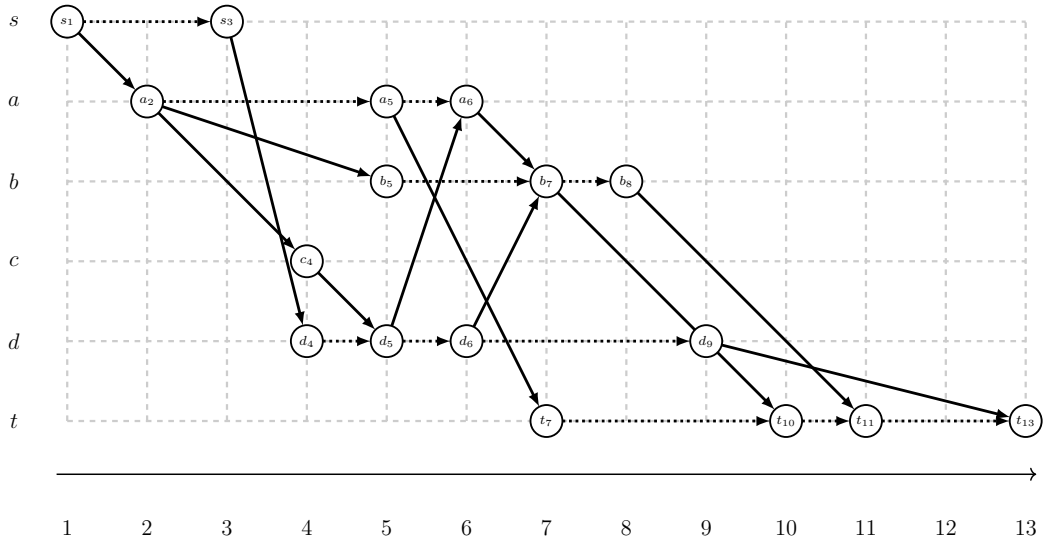

Figure 2.2: The time-expanded representation of the temporal graph in Figure 2.1.

Contribution. In **[9]** we developed a general algorithm to compute optimal temporal walks, supporting all classical criteria in the unrestricted waiting setting, including lexicographic combination. The algorithm follows a similar approach as [27] and [64] with a similar complexity including a logarithmic factor. As we improved this result in our next work, it will not be discussed in this dissertation. Here we first present a warm-up algorithm to compute reachability, disregarding any other optimization, which for simplicity is given under the assumption of strictly positive travel times. It helps familiarize the reader with waiting constraints and the use of the representation of temporal graphs with two lists, and at the same time provide a useful tool, as it solves the reachability problem with waiting constraints elegantly and in linear time.

Afterwards, we propose a temporal-edge scanning algorithm for single-source minimum-cost walks that runs in linear time given an acyclic time-expanded representation. The acyclic assumption means that the temporal graph does not contain a cycle of temporal edges with zero travel-time at any time instant. This acyclicity property is obviously satisfied when travel times are positive which is the case in many practical settings modelling the spread of information or any kind of agent. The algorithm can handle waiting-time constraints as defined in [5]. We use an algebraic definition of cost inspired by [59, 60], enabling a large variety of cost definitions, including the linear combination considered in [5], or lexicographic compositions such as shortest-fastest [45, 58]. This shows that linear time is possible for all criteria given an acyclic time-expanded representation when this was unknown for shortest-fastest and overall waiting time. Moreover, this holds even with waiting-time constraints while a logarithmic factor was previously necessitated for all criteria in that context. Our algorithm also solves the profile problem (again in linear time) with waiting-time constraints. No such algorithm was previously claimed, although we suspect that [5] can be adapted for that, but with a logarithmic slowdown. See Table 2.1 for a comparison with previous work.

Our algorithm does not work directly on the time-expanded representation but on two locally ordered lists of temporal edges, one where edges arriving at any given node are

Criterion	Time complexity	Model	Waiting restr.	Input
Earliest arrival time	$O(M)$ [27, 64]	$\lambda > 0$	✗	pre-sorted
Shortest duration	$O(M)$ [64]	$\lambda > 0$	✗	time-expanded
Fewest hops	$O(M)$ [65]	$\lambda > 0$	✗	time-expanded
Any above Overall waiting time Linear combination	$O(M \log M)$ [5]	–	✓	any
Shortest-fastest	$O(M \log \Delta)$ **[9]**	$\lambda > 0$	✗	pre-sorted
Profile	$O(M \log \Delta)$ [27]	$\lambda > 0$	✗	pre-sorted
	Chapter 2 Contribution			
Any above	$\boldsymbol{O(M)}$ Algorithm 2	acyclic	✓	time-expanded
Overall waiting time	$\boldsymbol{\Omega(M \log M)}$ Theorem 5	$\lambda > 0$	✗	pre-sorted
Any above	$\boldsymbol{O(M \log n)}$Algorithm 5	–	✓	time-expanded
Latest-dep., min.-len.	$\boldsymbol{\Omega(M \log n)}$ Theorem 7	–	✗	time-expanded

Table 2.1: Best time complexities for solving single-source optimal temporal walks for various criteria in a temporal graph with M temporal edges, n nodes, and where the temporal in-degree is at most $\Delta \leq M$. The "Model" column indicates if positive travel times are assumed with "$\lambda > 0$"; "acyclic" stands for the more general setting where no cycle of zero-travel-time edges occurs at any time instant; and a dash stands for the general model where such cycles can occur. A check-mark in column "Waiting restr." indicates that waiting-time constraints are supported while a cross indicates that unrestricted waiting is assumed. The "Input" column indicates if the input is required to be a "pre-sorted" list of temporal edges, or a "time-expanded" representation, or a list of temporal edges in "any" order.

sorted by non-decreasing arrival time and one where edges departing from any given node are sorted by non-decreasing departure time. We call this representation a "doubly-sorted" representation of the temporal graph. It is indeed equivalent to the time-expanded representation in the sense that one can easily be computed from the other in linear time and space.

We also show that the setting in which we obtain a linear-time algorithm is the widest possible. First, a single sorted list of temporal edges is not sufficient for obtaining linear time. A classical sorting argument allows to derive a lower bound of $\Omega(M \log M)$ with such a "singly-sorted" representation for algorithms using comparisons only, which is indeed a desirable algorithmic feature in any approach supporting a wide variety of abstract costs. This shows that our requirement for a time-expanded representation or equivalently a doubly-sorted representation with two orderings is somehow necessary for allowing linear-time computation. It also sheds light on why the time-expanded representation could enable linear time for shortest duration and fewest number of edges. Second, dropping the acyclic assumption with zero travel times leads to a setting encompassing classical shortest path computation. An $\Omega(M \log n)$ conditional lower-bound can easily be obtained assuming that directed single-source shortest paths computation must take $\Omega(n \log n)$ time in the comparison-addition model. It is thus unlikely to reach linear time outside our acyclic setting unless progress is made along the precise complexity of directed single-source shortest paths which is still open as far as we know [66]. Note that such a lower-bound holds for algorithms visiting nodes by non-decreasing distance or more generally those following the component hierarchy approach [54] even though this approach leads to linear time in the undirected setting [61].

Finally, we show how to handle the setting where cycles of edges with zero travel time

can occur. It is then possible to compute for a fixed source an adequate pair of orderings allowing our algorithm to run correctly for that source. This pair can be computed in $O(M \log n)$ time where n is the number of nodes, allowing to reduce the complexity from $O(M \log M)$ to $O(M \log n)$ compared to previous work. Note that M is not bounded with respect to n and can be much larger in practice. Note also that this matches the $\Omega(M \log n)$ conditional lower-bound.

Our main new technique consists in maintaining at each node a list of intervals spanning a sliding window of outgoing temporal edges. It allows to update in constant time the cost of candidate minimum-cost walks departing in a time interval. These intervals may be split as temporal edges are scanned, and a careful use of the two orderings of temporal edges given as input allows to manage them with linear amortized complexity. We think that this technique is a valuable contribution and could appear useful for other temporal graph problems involving temporal connectivity such as computing temporal betweenness [15] or delay-robust temporal walks [37].

A main difficulty behind our algorithm resides in requiring the appropriate properties on the order in which temporal edges are scanned. Good intuition can be obtained by seeing them as sorted by non-decreasing arrival time, but such an ordering cannot be obtained in linear time from a time-expanded representation. Instead, we rely on a locally ordered list computed in linear time through a procedure which is similar to topological sorting of the time-expanded representation. We also have to take care of tricky dependencies with the other ordering of the doubly-sorted representation in the core algorithm. Overall, this nevertheless results in a relatively simple algorithm unifying the temporal edge scanning and time-expanded approaches while integrating waiting constraints in a novel manner.

The generality of the algorithm relies on computing for each temporal edge a temporal walk with minimum abstract cost ending with that edge. This relies on a natural property of these abstract costs called isotonicity [59, 60] and grasping that, when several temporal walks can be extended by the same temporal edge, the minimum cost is obtained by extending the walk with minimum cost. This property is formally defined in Section 2.3. Although most classical criteria do *not* satisfy isotonicity, we can use an auxiliary cost structure which does and that indirectly lets us find the optimum for such criteria after some post-processing. Shortest duration is an example of non-isotonic cost: for example consider in Figure 2.1 the sa temporal walks $Q_1 = \langle (s, a, 1, 1) \rangle$ and $Q_2 = \langle (s, d, 3, 1), (d, a, 5, 1) \rangle$. Temporal walk Q_1 has duration $1 + 1 - 1 = 1$, which is lower than the duration of Q_2, that is $5 + 1 - 3 = 3$. Both temporal walks can be extended with temporal edge $(a, t, 5, 2)$, but the first temporal walk now has duration $5 + 2 - 1 = 6$, while the second has duration $5 + 2 - 3 = 4$. The trick to compute temporal walks with shortest duration is to use the opposite of departure time as an auxiliary cost, since a walk with latest possible departure arriving with a given temporal edge also has minimum duration among temporal walks arriving with that edge. A shortest duration temporal walk reaching a node is then obtained by minimizing over all possible arrival edges at the node.

2.1 Model and representation

In this chapter we will call a *temporal graph* a point availability temporal graph subject to waiting constraints, defined as follows. A temporal graph is a tuple $G = (V, E, \alpha, \beta)$, where V is the set of nodes, E is the set of temporal edges and $\alpha, \beta \in [0, +\infty]^V$ are minimum and maximum waiting-times at each node. In this model of temporal graph, a

temporal walk has to satisfy the following waiting policy. A sequence of temporal edges $Q = \langle e_1 = (u_1, v_1, \tau_1, \lambda_1), \ldots, e_k = (u_k, v_k, \tau_k, \lambda_k)\rangle \subseteq E^k$ is a *temporal walk* from u to v, if in addition to the temporal walk constraints already defined in Section 1.1.5, it satisfies $a_{i-1} + \alpha_{u_i} \leq \tau_i \leq a_{i-1} + \beta_{u_i}$ for all $i \in \{2, \ldots, k\}$, where $a_{i-1} = \tau_{i-1} + \lambda_{i-1}$ is the arrival time of e_{i-1}. Note that the waiting time $\tau_i - a_{i-1}$ at node u_i is constrained to be in the interval $[\alpha_{u_i}, \beta_{u_i}]$. The *unrestricted waiting* policy can be modelled by setting $\alpha_v = 0$ and $\beta_v = +\infty$ for all $v \in V$.

For the sake of brevity, when it is clear from the context, we often write edge instead of temporal edge and walk instead of temporal walk.

We say that a temporal edge $e = (x, y, \tau, \lambda)$ *extends* Q when $x = v_k$ and $arr(Q) + \alpha_x \leq \tau \leq arr(Q) + \beta_x$. When e extends Q, we can indeed define the walk $Q.e = \langle e_1, \ldots, e_k, e\rangle$ from u to y. Moreover, we also say that e extends e_k as it indeed extends any walk Q having e_k as last edge. A *zero-walk* is a walk consisting of temporal edges with same departure time and with zero travel time, and going through nodes with zero minimum waiting constraint. More formally, we define a zero-walk as a walk $\langle e_1 = (u_1, v_1, \tau_1, \lambda_1), \ldots, e_k = (u_k, v_k, \tau_k, \lambda_k)\rangle$ such that $\tau_i = \tau_j$ for $i, j \in [k]$, $\lambda_i = 0$ and $\alpha_{u_i} = 0$ for $i \in [k]$. Such a zero-walk is called a zero-cycle when $u_1 = v_k$. We say that a temporal graph G is *zero-acyclic* when there is no zero-cycle in G.

Without loss of generality, we can restrict our attention to nodes appearing as head or tail of at least one temporal edge and we thus assume $|V| = O(|E|)$. An algorithm is said to be linear in time and space when it runs in $O(|E|)$ time and uses $O(|E|)$ space.

Doubly-sorted representation. In order to introduce a novel representation of a temporal graph, we define the following type of orderings of temporal edges with respect to time. We say that an ordering E^{ord} of all temporal edges is *node-departure sorted* if all edges departing from the same node are ordered by non-decreasing departure time in E^{ord}, that is we have $e <_{E^{ord}} f$ whenever $e, f \in E$ have the same tail and satisfy $dep(e) < dep(f)$. Similarly, we say that an ordering E^{ord} of all temporal edges is *node-arrival sorted* if all edges arriving to the same node are ordered by non-decreasing arrival time in E^{ord}, that is we have $e <_{E^{ord}} f$ whenever $e, f \in E$ have the same head and satisfy $arr(e) < arr(f)$.

We can now define the *doubly-sorted representation* of a temporal graph (V, E, α, β) as a data-structure with two lists (E^{dep}, E^{arr}), containing $|E|$ quadruples each, representing all temporal edges in E, where E^{arr} is a node-arrival sorted list, and E^{dep} is a node-departure sorted list. Moreover, we assume that we have implicit pointers between the two lists, that link each quadruple of one list to the quadruple representing the same temporal edge in the other list. We also assume that each list of temporal edges is stored in an array T such that each element $T[i]$ can be accessed directly through its index $i \in [1, |T|]$ in constant time. Given two indexes $i \leq j$, we also let $T[i : j]$ denote the sub-array of elements of T with index in $[i, j]$.

We will see that it is possible to compute a doubly-sorted representation from a time-expanded representation, and vice versa, in linear time and space. In Proposition 2 we will provide a stronger result with more details.

2.2 Computing reachability under waiting constraints

Reachability, that is connectivity through a walk, is a fundamental notion in classical graphs. There are trivial algorithms for static graphs that determine efficiently which nodes or

edges can be reached from a given source node. On the other hand, in temporal graphs, especially with waiting constraints, this notion has not yet been extensively studied. In this section we will focus on reachability without concern for any optimization criterion, and we aim at designing a simple algorithm under waiting constraints. The main strength of the algorithm we develop is its simplicity, which comes with no downplay in efficiency, since it runs in linear time. It thus matches the time complexity $O(M)$ of the state of the art [62]. Our algorithm performs a linear scan of the list of temporal edges, in a spirit similar to [26]. The algorithm will also serve as a warm-up to the more involved algorithms that will follow in the next sections. For simplicity, it is presented in the more restrictive setting where all temporal edges have a strictly positive travel time and the input doubly-sorted representation satisfies a stronger property. In particular, we assume to be given a *fully doubly-sorted* representation (E^{dep}, E^{arr}) of a temporal graph, which means that E^{dep} (resp. E^{arr}) is sorted by non-decreasing departure time (resp. non-decreasing arrival time). Let us formulate the reachability problem as follows.

Singles-Source Edge Reachability Problem. Given a temporal graph with waiting constraints $G = (V, E, \alpha, \beta)$ and a source node s, compute the set of all temporal edges that are s-reachable.

Notice that this problem generalises the single-source earliest arrival time problem. Indeed, given the set of all s-reachable edges it is sufficient to perform a linear scan of the set to identify for each node v the s-reachable edge with head v that has lowest arrival time, and which corresponds to the earliest arrival time at v.

We design an algorithm which mainly consists of scanning linearly edges in E^{arr} while updating the set A_v of s-reachable edges terminating sv-walks in the temporal graph resulting from the edges read so far. To help identify edges that will appear in such walks in next iterations, we also mark edges that extend these walks.

We now describe Algorithm 1 more precisely. We first build the lists E_v^{dep} of temporal edges with tail v sorted by non-decreasing departure time by bucket sorting E^{dep} at Line 1. We then identify the s-reachable edges as follows. We linearly scan E^{arr}. In the temporal graph resulting from the temporal edges read up to edge $e = (u, v, \tau, \lambda) \in E^{arr}$, the only walks from s that have not been considered yet must contain e, and must have it as the last edge. This comes from the positive travel time assumption, which implies that edges along a walk have increasing arrival time and from the fact that E^{arr} is sorted by non-decreasing arrival time. If its tail u is s, or if e is marked, then we know that there exists a walk from s to its head v. In that case, we add edge e to A_v at Line 9, and we then mark edges that extend e, that is edges in E_v^{dep} with departure time in $[a + \alpha_v, a + \beta_v]$, since the arrival time of e is $a = \tau + \lambda$. These edges appear consecutively in E_v^{dep} which is processed linearly as walks from s to v are identified. This process is done in Lines 10-14 in Algorithm 1, starting from the index p_v of the last processed edge in E_v^{dep}, and such edges f that extend e are marked at Line 13 before updating p_v. Moreover, we use classical parent pointers to be able to compute an sv-walk for each s-reachable edge with head v. Each parent pointer $P[f]$ of an edge f is initially set to a null value $\perp$ at Line 6. Whenever we mark edge f, that extends the currently scanned edge e, we set the parent pointer of f to e. If f is an s-reachable edge at v, we can then get an sv-walk by following the parent pointers $P[f], P[P[f]], \ldots$.

Example of execution We will now present an example of execution of an Algorithm 1 for computing the reachable temporal edges in the temporal graph represented in Figure 2.3 with source node s. The algorithm receives in input the list of temporal edges:

Input: A fully doubly-sorted representation (E^{arr}, E^{dep}) of a temporal graph G with waiting constraints (α, β), and a source node $s \in V$.

Output: The sets $(A_v)_{v \in V}$ of s-reachable edges at each node v sorted by non-decreasing arrival time.

For each node v, generate the list E_v^{dep} by bucket sorting E^{dep}.
For *each node* v **do**
    Set $A_v := \emptyset$. `/* Set of s-reachable edges (as a sorted list). */`
    Set $p_v := 0$. `/* Index of the last processed edge in` E_v^{dep}`. */`
Set all the edges in E^{arr} as unmarked.
Set $P[e] := \perp$ for each edge $e \in E^{arr}$.`/* Parent of` e`, initially null. */`
For *each edge* $e = (u, v, \tau, \lambda)$ *in* E^{arr} **do**
    If $u = s$ *or* e *is marked* **then**
        `/*` e `is s-reachable. */`
        $A_v := A_v \cup \{e\}$
        Let $a = \tau + \lambda$ be the arrival time of e.
        `/* Process further edges from` v `until dep.time` $\geq a + \beta_v$`: */`
        Let $l > p_v$ be the first index of an edge $(v, w, \tau', \lambda') \in E_v^{dep}$ such that $\tau' \geq a + \alpha_v$ (set $l := |E_v^{dep}| + 1$ if no such index exists).
        Let $r \geq l$ be the last index of an edge $(v, w, \tau', \lambda') \in E_v^{dep}$ such that $\tau' \leq a + \beta_v$ (set $r := l - 1$ if no such index exists).
        `/* Mark unmarked edges with dep.time in` $[a + \alpha_v, a + \beta_v]$`: */`
        If $l \leq r$ **then** mark each edge $f \in E_v^{dep}[l : r]$ and set $P[f] := e$.
        Set $p_v := r$.
Return *the sets* $(A_v)_{v \in V}$.

Algorithm 1: Computing, for each node v, the set A_v of all s-reachable edges with head v.

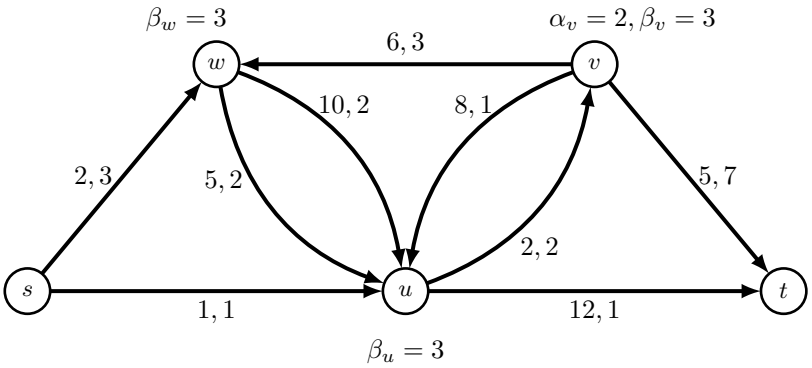

Figure 2.3: A point availability temporal graph with waiting time constraints, where $\alpha_s = \alpha_u = \alpha_w = \alpha_t = 0$ and $\alpha_v = 2$, $\beta_s = \beta_t = +\infty$ and $\beta_u = \beta_v = \beta_w = 3$.

$$E^{arr} = \{(s, u, 1, 1), (u, v, 2, 2), (s, w, 2, 3), (w, u, 5, 2), (v, u, 8, 1), (v, w, 6, 3), (w, u, 10, 2), (v, t, 5, 7), (u, t, 12, 1)\}$$

and

$$E^{dep} = \{(s,u,1,1),(s,w,2,3),(u,v,2,2),(w,u,5,2),(v,t,5,7),(v,w,6,3),\\ (v,u,8,1),(w,u,10,2),(u,t,12,1)\}.$$

In the preprocessing step, the algorithm computes the following lists from E^{dep}:

- $E_s^{dep} = \{(s,u,1,1),(s,w,2,3)\}$,
- $E_u^{dep} = \{(u,v,2,2),(u,t,12,1)\}$,
- $E_v^{dep} = \{(v,t,5,7),(v,w,6,3),(v,u,8,1)\}$,
- $E_c^{dep} = \{(w,u,5,2),(w,u,10,2)\}$.

During the execution the algorithm will work on such lists rather then E^{dep}. Moreover, still in the preprocessing step, all sets A_x are initialised as empty. We will follow the execution of the algorithm iteration by iteration: in each figure the temporal edge currently scanned is coloured in red and the temporal edges that extends it are coloured in blue. Moreover, we will mark with a $*$ edges that have been marked as reachable in the course of the execution up until the end of the current iteration.

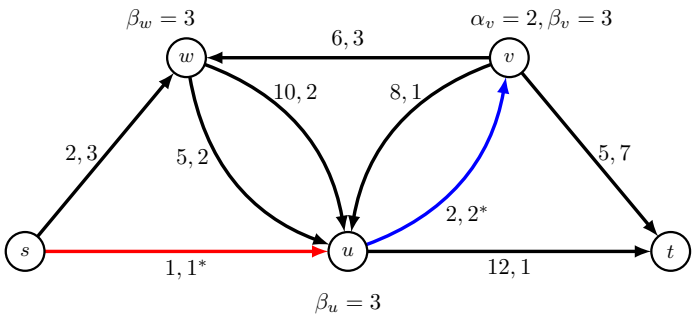

Figure 2.4: Iteration 1.

Iteration 1. In the first iteration we scan the temporal edge $(s,u,1,1)$. As its tail is the source, it is reachable and we append it to A_u. Its arrival time is $a = 1+1 = 2$. The edges that extend $(s,u,1,1)$, and that we want to mark as reachable, are those that have tail u and a departure time that belongs to $[a+\alpha_u, a+\beta_u] = [2,5]$. As p_u is set to zero, we start scanning E_u^{dep} from the beginning. The only edge that extends $(s,u,1,1)$ is $(u,v,2,2)$, which is in the firs position of E_u^{dep}. We mark it as reachable, set the parent pointer $P[(u,v,2,2)] = (s,u,1,1)$ and we set $p_u = 1$.

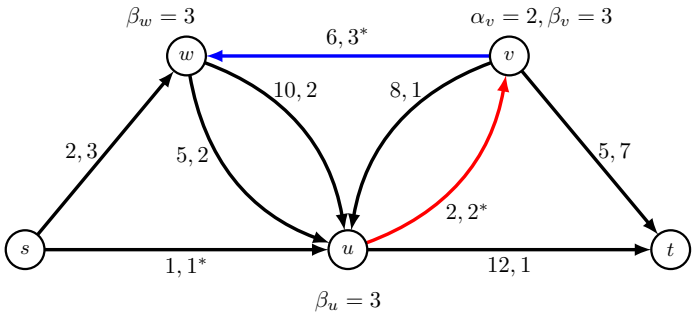

Figure 2.5: Iteration 2.

Iteration 2. In the second iteration we scan edge $(u, v, 2, 2)$. It is marked as reachable, thus we add it to A_v. Its arrival time is $a = 2 + 2 = 4$. The edges that extend $(u, v, 2, 2)$, must have a departure time that belongs to $[a + \alpha_v, a + \beta_v] = [6, 7]$. As p_v is set to zero, we start processing E_v^{dep} from the beginning. The first edge in E_v^{dep} that we read is $(v, t, 5, 7)$, as it departs too early we do not mark it and move to the next one. The second edge is $(v, w, 6, 3)$ which extends $(u, v, 2, 2)$, thus we mark it as reachable ans set the parent pointer $P[(v, w, 6, 3)] = (u, v, 2, 2)$. The next edge in E_v^{dep}, which is $(v, u, 8, 1)$, has a departure time greater than $a + \beta_v = 7$, thus we stop processing E_v^{dep} and we set $p_v = 2$.

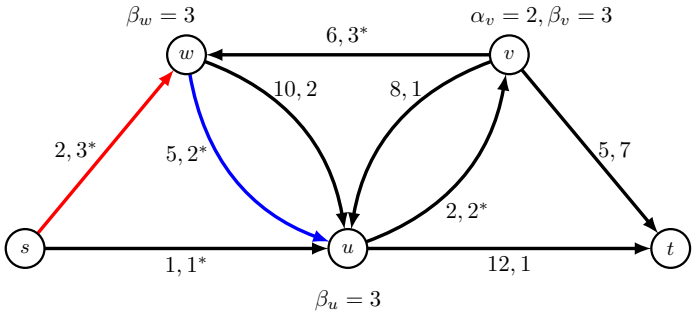

Figure 2.6: Iteration 3.

Iteration 3. We scan edge $(s, w, 2, 3)$, as it departs from the source, it is reachable and we add it to A_w. The edges that extend $(s, w, 2, 3)$ must have a departure time that belongs to $[a + \alpha_w, a + \beta_w] = [5, 8]$. Since $p_w = 0$ we start processing E_w^{dep} from the beginning. The first edge we read is $(w, u, 5, 2)$ which extends $(s, w, 2, 3)$, thus we mark it as reachable and set its parent pointer to $(s, w, 2, 3)$. The next edge in E_w^{dep}, which is $(w, u, 10, 2)$, has a departure time greater than $a + \beta_w = 8$, thus we stop processing E_v^{dep} and we set $p_w = 1$.

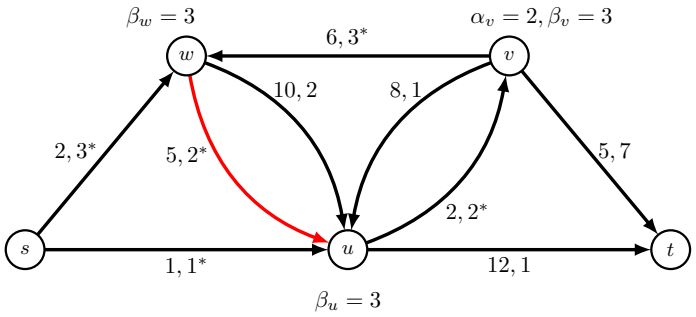

Figure 2.7: Iteration 4.

Iteration 4. We scan edge $(w, u, 5, 2)$. It is marked as reachable, thus we add it to A_u. Its arrival time is $a = 5 + 2 = 7$. The edges that extend $(w, u, 5, 2)$ must have a departure time that belongs to $[a+\alpha_u, a+\beta_u] = [7, 10]$. As $p_u = 1$ the first edge that we consider from E_u^{dep} is $(u, t, 12, 1)$. As its departure time is greater than $a + \beta_u = 10$, we do not process any edge from E_u^{dep} and do not increase the value of p_u.

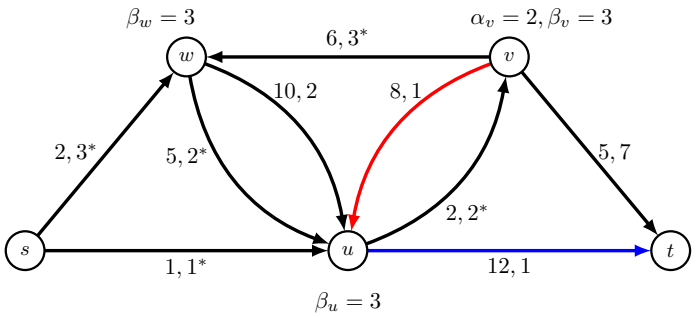

Figure 2.8: Iteration 5.

Iteration 5. We scan edge $(v, u, 8, 1)$. As it does not depart from the source and it is not marked as reachable we do not perform any operation.

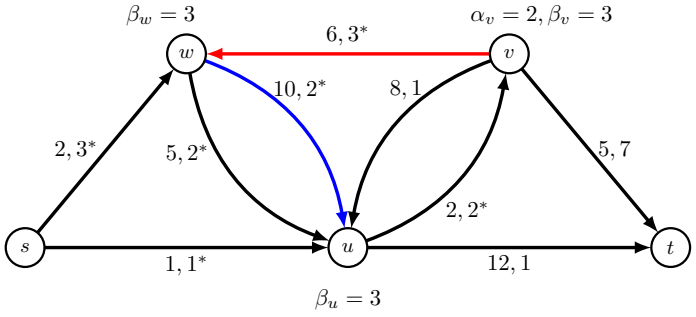

Figure 2.9: Iteration 6.

Iteration 6. We scan edge $(v, w, 6, 3)$. It is marked as reachable, thus we add it to A_w.

Its arrival time is $a = 6 + 3 = 9$. The edges that extend $(v, w, 6, 3)$ must have a departure time that belongs to $[a + \alpha_w, a + \beta_w] = [9, 12]$. As $p_w = 1$ the first edge that we consider from E_w^{dep} is $(w, u, 10, 2)$. As it extends $(v, w, 6, 3)$ we mark it as reachable and set its parent pointer to $(v, w, 6, 3)$. There are no more edges to consider in E_w^{dep}, thus we set $p_w = 2$ and conclude the iteration.

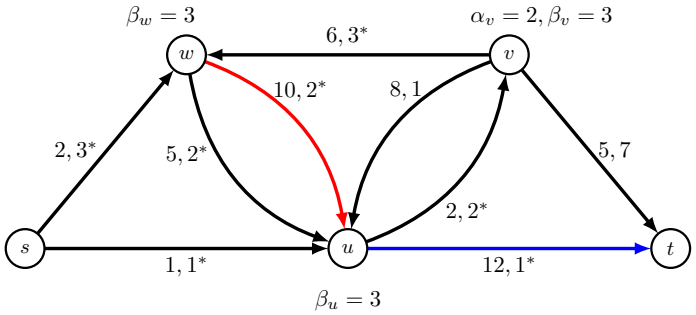

Figure 2.10: Iteration 7.

Iteration 7. We scan edge $(w, u, 10, 2)$. It is marked as reachable, thus we add it to A_u. Its arrival time is $a = 10 + 2 = 12$. The edges that extend $(w, u, 10, 2)$ must have a departure time that belongs to $[a + \alpha_u, a + \beta_u] = [12, 15]$. As $p_u = 1$ the first edge that we consider from E_u^{dep} is $(u, t, 12, 1)$. As it extends $(w, u, 10, 2)$ we mark it as reachable and set its parent pointer to $(w, u, 10, 2)$. There are no more edges to consider in E_u^{dep}, thus we set $p_u = 2$ and conclude the iteration.

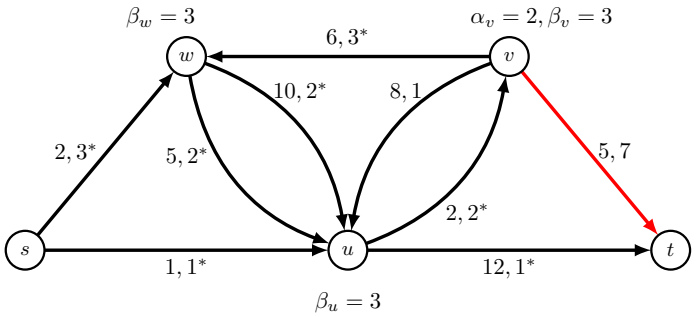

Figure 2.11: Iteration 8.

Iteration 8. We scan edge $(v, t, 5, 7)$. As it does not depart from the source and it is not marked as reachable we do not perform any operation.

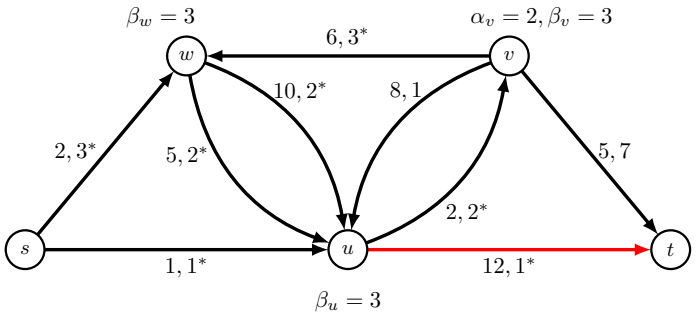

Figure 2.12: Iteration 9.

Iteration 9. We scan edge $(u, t, 12, 1)$. It is marked as reachable, thus we add it to A_t. As $E_t^{dep} = \emptyset$ we conclude the iteration and the execution of Algorithm 1.

We identified all the s-reachable temporal edges in the temporal graph in Figure 2.3. Let us suppose that we are interested in a particular destination node t. As there is at least an s-reachable edge with head t we know that t is reachable, and we also know that the earliest arrival time from s to t is 13. We can compute a temporal walk from s to t arriving at time 13 following the parent pointers starting from $(u, t, 12, 1)$. By doing so, we obtain the sequence $\langle (s, u, 1, 1), (u, v, 2, 2), (v, w, 6, 3), (w, u, 10, 2), (u, t, 12, 1) \rangle$, which is represented in green in Figure 2.13.

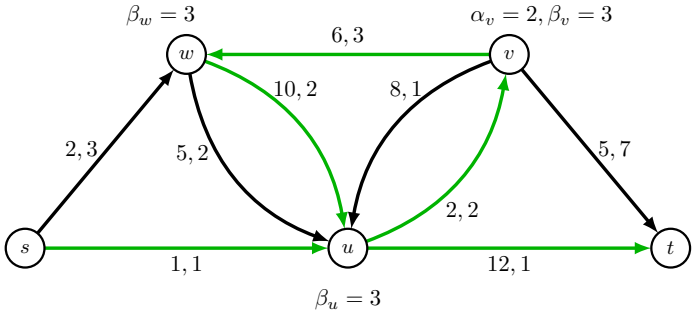

Figure 2.13: A temporal walk from s to t obtained following the parent pointers computed during the execution, starting from the one of $(u, t, 12, 1)$.

Theorem 3. *Given a fully doubly-sorted representation of a temporal graph $G = (V, E, \alpha, \beta)$ having positive travel times, and a source node $s \in V$, Algorithm 1 computes all s-reachable temporal edges in linear time and space.*

Proof. Correctness. Let us denote by $G_k = (V, E^{arr}[1 : k], \alpha, \beta)$ the temporal graph induced by the first k temporal edges in E^{arr}. We will prove, by induction on k, the following two invariants:

(I_k^1) For every node v, A_v contains all s-reachable edges with head v in G_k.

(I_k^2) The marked edges are all the edges in E that extend a walk from s in G_k.

The correctness of the algorithm will follow from the invariant (I_k^1) for $k = |E|$. The invariants are satisfied for $k = 0$ since there are no edges in G_0 while the sets $(A_v)_{v\in V}$ of s-reachable edges are initially empty and no edge is initially marked.

Now suppose that the two invariants hold for $k-1$, with $k \geq 1$, and let us prove that they still hold for k after scanning the kth edge $e_k = (u, v, \tau, \lambda)$ in E^{arr}. To prove (I_k^1) and (I_k^2), we first show that the condition of the if statement at Line 8 is met when e_k is an s-reachable edge in G_k. It is obviously the case when $u = s$ as $\langle e_k\rangle$ is in G_k, or when e_k was previously marked, as Invariant (I_{k-1}^2) then implies that it extends a walk Q from s in G_{k-1} and that $Q.e_k$ is a walk in G_k. The converse also holds: if e_k is an edge of a walk Q from s in G_k, then either it is the first edge and we have $u = s$ or the sequence Q' of edges before e_k in Q is a walk in G_{k-1} and (I_{k-1}^2) implies that it is marked.

Note that when e_k appears in a walk Q of G_k, it must be the last edge of Q as E^{arr} is sorted by non-decreasing arrival time and edges have positive travel time. This allows to prove (I_k^1): as we assume (I_{k-1}^1), we just have to consider walks from s that are in G_k but not in G_{k-1}, that is those containing e_k. Since all these walks have e_k as last edge, and e_k is the only edge added to A_v when such walks exist, we can conclude that (I_k^1) holds.

Similarly, to prove (I_k^2) when (I_{k-1}^2) holds, we just have to consider the edges extending a walk Q from s which is in G_k but not in G_{k-1}. As discussed above, when such a walk Q exists, e_k is its last edge and the condition of the if statement at Line 8 holds. Edges extending such a walk Q are thus those extending e_k, that is all edges $f \in E_v^{dep}$ such that $a + \alpha_v \leq dep(f) \leq a + \beta_v$. Note that the ordering of E_v^{dep} implies that these edges are consecutive in E_v^{dep}. If no such edges exist, let l' and r' designate the first and last indexes respectively where they are placed in E_v^{dep}. To prove (I_k^2), it thus suffices to prove that all edges in $E_v^{dep}[l' : r']$ are marked after scanning e_k and that only edges in $E_v^{dep}[l' : r']$ are marked during the iteration for e_k (if no such edges exist we prove that we mark no edges). Consider the values l and r computed at Lines 11 and 12 respectively. If no edge f extends e_k, then we get $r = l-1$ and no edge is marked. Now, we assume that such edges exist and that l' and r' are well defined. First assume $l \leq r$ and thus that l was not set to $|E_v^{dep}| + 1$. The choice of l, r then imply $a + \alpha_v \leq dep(E_v^{dep}[l])$ and $dep(E_v^{dep}[r]) \leq a + \beta_v$. We thus have $l' \leq l \leq r \leq r'$ and all marked edges at Line 13 are in $E_v^{dep}[l' : r']$. Moreover, the choice of r indeed then implies $r = r'$. We still need to prove that edges in $E_v^{dep}[l' : l-1]$ have already been marked. Otherwise, when $r = l-1$, no edge is marked. This occurs when $p_v \geq r'$ and we then have $l = p_v + 1$. In both cases, it remains to prove that all edges $f \in E_v^{dep}[l' : \min\{l-1, r'\}]$ have already been marked. This interval is non-empty when $l' \leq l-1$ and thus $p_v = l-1$ by the choice of l. We thus have $p_v \geq \min\{l-1, r'\}$. Let i be the index of f in E_v^{dep} and consider the iteration $j < k$ when p_v was updated form a value smaller than i to a value $r'' \geq i$ where l'' and r'' denote the indexes computed for variables l and r respectively during the j-th iteration for edge $e_j \in E^{arr}$.

Since E^{arr} is sorted by non-decreasing arrival time, the arrival time a' of e_j satisfies $a' \leq a$ and we thus have $dep(f) \geq a + \alpha_v \geq a' + \alpha_v$. The choice of index l at Line 11 in that iteration thus guarantees that the index l'' must satisfy $l'' \leq i$. We thus have $l'' \leq i \leq r''$ and f was marked at Line 13 during the jth iteration. This completes the proof of (I_k^2).

We finally prove that the parent pointers allow us to compute for each s-reachable edge $f = (u, v, \tau, \lambda)$ with head v an sv-walk ending with f. If $f \in A_v$ and it is not marked, then $P[f] = \perp$, and we must have $u = s$ as f was added to A_v. In this case, $\langle f\rangle$ is an sv-walk itself. Now consider the case $f \in A_v$ and f is marked. Consider the iteration k where f was marked. By (I_{k-1}^2) and (I_k^2), f extends a walk from s ending with e_k, where e_k is the edge scanned at iteration k, and $P[f]$ was then set to e_k.

This guarantees by a simple induction that, if $P[f] \neq \perp$, by following the parent pointers in classical manner, namely $P[f], P[P[f]], \ldots$, until $\perp$ is found, it is possible to obtain a walk terminating with edge f.

Complexity analysis. The preprocessing of E^{dep} and the initialization from Line 1 to Line 6 clearly takes linear time. The main for loop scans each temporal edge $e = (u, v, \tau, \lambda)$ in E^{arr} exactly once. For each iteration there are three operations that may require non-constant time: the computation of l and r at Lines 11 and 12, and marking edges in $E_v^{dep}[l, r]$ at Line 13. They all take $O(r - p_v)$ time as l and r can be found by scanning edges in E_v^{dep} from $p_v + 1$. Thanks to the update of the index p_v to r, each edge in E_v^{dep} is processed at most once for a total amortized cost of $O(|E_v^{dep}|)$. Overall, this leads to a time complexity of $O(|E| + \sum_{v \in V} |E_v^{dep}|) = O(|E|)$. Algorithm 1 thus runs in linear time. Finally, let us notice that for all nodes v, the set A_v has size bounded by the number of temporal edges with head v. We thus have $\sum_{v \in V} |A_v| \leq |E|$, and the space complexity of Algorithm 1 is linear. □

2.3 Computing single-source all-reachable-edge minimum-cost walks

To solve the problem of computing minimum-cost walks from a single source s, we will consider a more general problem consisting in computing at each destination v, and for each possible s-reachable edge e with head v, an sv-walk with minimum cost among all sv-walks ending with e.

General cost structure for walks. We integrate a temporal graph $G = (V, E, \alpha, \beta)$ with an algebraic *cost structure* $(C, \gamma, \oplus, \preceq)$, where C is the set of possible *cost values*, γ is a *cost function* $\gamma : E \to C$, $\oplus$ is a *cost combination function* $\oplus : C \times C \to C$, and $\preceq$ is a *cost total order* $\preceq \subseteq C \times C$. We also define the relation $\prec$ between the elements of C as $a \prec b$ if and only if $a \preceq b$ and $a \neq b$. For any walk $Q = \langle e_1, \ldots, e_k \rangle$, the *cost function* of Q is recursively defined as follows: $\gamma_Q = \gamma_{\langle e_1, \ldots, e_{k-1} \rangle} \oplus \gamma(e_k)$, with $\gamma_{\langle e_1 \rangle} = \gamma(e_1)$. In other words, the costs combine along the walk according to the cost combination function. The cost structure is supposed to satisfy the following *right-isotonicity property* [59, 60] (*isotonicity* for short):

$$\text{for any } c_1, c_2, c \in C \text{ such that } c_1 \preceq c_2, \text{ we have } c_1 \oplus c \preceq c_2 \oplus c. \qquad \text{(isotonicity)}$$

This property guarantees that if several walks are extended by a given temporal edge e, then the best cost is obtained by extending the walk Q^* with minimum cost: as for any other walk Q we have $\gamma_{Q^*} \preceq \gamma_Q$, we get $\gamma_{Q^*.e} \preceq \gamma_{Q.e}$ by the isotonicity property and the cost function definition. However, a prefix of a minimum-cost walk is not necessarily a minimum-cost walk.

We define the single-source all-reachable-edge minimum-cost problem as follows.

> SINGLE-SOURCE ALL-REACHABLE-EDGE MINIMUM-COST PROBLEM. Given a temporal graph $G = (V, E, \alpha, \beta)$ with cost structure $(C, \gamma, \oplus, \preceq)$, and a source node $s \in V$, compute for each destination $v \in V$ and each possible s-reachable edge e with head v the minimum cost of any sv-walk ending with edge e.

The problem consists in computing for each node v all pairs (e, c) such that $e \in A_v$ and $c = \min\{\gamma_Q : Q \text{ is an } sv\text{-walk ending with edge } e\}$, where A_v denotes the set of all

s-reachable edges with head v. We will denote with A'_v the list of such pairs (e, c) ordered by non-decreasing arrival time of the edges. In this section we consider this problem in the case of zero-acyclic temporal graphs. Algorithm 1 represent a first step towards solving the minimum-cost problem, as it detects whether there exists a temporal walk ending with an edge, however there is no guarantee that the temporal walk computed is the one with minimum cost. It is thus necessary to keep track of several temporal walks as the algorithm progresses to find the optimal one.

We can now state our main result as follows.

Theorem 4. *Given either a time-expanded representation or a doubly-sorted representation of a zero-acyclic temporal graph $G = (V, E, \alpha, \beta)$ with cost structure $(C, \gamma, \oplus, \preceq)$ satisfying isotonicity, and a source node $s \in V$, the single-source all-reachable-edge minimum-cost problem can be solved in linear time and space.*

To prove the above theorem, we design an algorithm that scans linearly edges in E^{arr}. The general idea is to maintain for each unscanned edge f the minimum cost of any walk Q from s in the partial temporal graph induced by edges scanned so far such that f extends Q. We can update these costs each time a new edge $e \in E^{arr}$ is scanned relying on the property that the edges of Q are scanned in order. In particular, the cost that has been associated to e itself allows to infer easily the minimum cost of a walk from s ending with e.

More precisely, our algorithm considers only a subset of all walks from the source. This subset subtly depends on the given doubly-sorted representation (E^{dep}, E^{arr}) and matches the following definition. We say that a walk Q is (E^{dep}, E^{arr})-*respected* when for each pair e and f of consecutive edges in Q, and for each edge $e' \in E$ having same tail as f and satisfying $f \leq_{E^{dep}} e'$, we have $e <_{E^{arr}} e'$. In particular, we get $e <_{E^{arr}} f$ for $e' = f$, and the edges of Q must appear in order in E^{arr}. It also implies that edges e' after f in E^{dep} will be scanned after e by our algorithm, a property we will use for getting efficient updates. In the zero-acyclic setting, we will later see that any walk is (E^{dep}, E^{arr})-respected when E^{arr} satisfies some additional properties related to zero-acyclicity.

We now introduce two notions related to (E^{dep}, E^{arr})-respected walks. An (s, E^{dep}, E^{arr})-reachable edge is defined as an s-reachable edge that ends an (E^{dep}, E^{arr})-respected walk from s. Moreover, given a subset E' of edges, and an edge $f \in E$ with tail v, we define its *best extendable cost* with respect to E' as the minimum cost of an sv-walk Q that f extends in the partial graph induced by $E' \cup \{f\}$ and such that $Q.f$ is (E^{dep}, E^{arr})-respected.

Our algorithm maintains the best extendable costs of all edges with respect to the prefix of edges scanned so far as follows. Initially, all best extendable costs are undefined as expressed by a special value $\perp$. Then each time an edge $e \in E^{arr}$ is scanned, it is sufficient to update the costs of edges f that extend e and such that $\langle e, f \rangle$ is (E^{dep}, E^{arr})-respected as detailed in the proof of Lemma 1. The main difficulty is to perform this update in constant amortized time although a large number of edges f may extend e. For that purpose, by proceeding as in Algorithm 1, E^{dep} is first bucket sorted according to tails, and thus edges from a node v that extend the same walks from s to v are grouped into intervals of the array E^{dep}_v of edges from v. These intervals are stored in a doubly linked list $\mathcal{I}_v$ of quadruples where each interval $(l, r, c, e) \in \mathcal{I}_v$ represents the association of edges in $E^{dep}_v[l : r]$ to best extendable cost c and parent edge e where e is an edge they all extend and such that there exists an (E^{dep}, E^{arr})-respected walk from s having cost c and ending with e. We also maintain the overall interval (l_v, r_v) spanned by $\mathcal{I}_v$. Note that this interval is considered to be empty when $r_v < l_v$. Algorithm 2 describes how to update these intervals each time an edge e with head v is scanned. It relies on the fact that intervals of $\mathcal{I}_v$ are consecutive

in E_v^{dep} and are also ordered by non-decreasing associated costs. When the scan of E^{arr} has progressed sufficiently, the best extendable cost of some edges will not change any more and it is then stored directly in an array B through the procedure FinalizeCosts(v, j) which erases intervals of $\mathcal{I}_v$ up to index j and stores the cost associated to the corresponding edges in B as detailed in Algorithm 3. At the end of the scan, our algorithm returns the lists $(A'_v)_{v\in V}$ which contain the minimum cost associated to all possible (s, E^{dep}, E^{arr})-reachable edges. During the execution, we build parent pointers, that allow to represent, for each such edge $e \in A'_v$, an sv-walk with minimum cost ending with edge e by associating to each edge f the edge $P[f]$ preceding it in such a walk.

2.3.1 Computing shortest duration walks.

The duration of a walk Q is defined as the time $arr(Q) - dep(Q)$ elapsed between its departure and its arrival. It is one of the classical criteria for evaluating the quality of walks, shortest duration being usually preferred. As an example, we now show how to obtain shortest-duration walks with Algorithm 2 and obtain the following result as a corollary of Theorem 4.

Corollary 1. *Given a doubly-sorted representation (E^{arr}, E^{dep}) of a strict temporal graph $G = (V, E, \alpha, \beta)$, and a source node s, the single-source shortest-duration walk problem, that is computing a shortest duration sv-walk for all nodes v, can be solved in linear time and space.*

For a given destination v and a given edge e with head v and arrival time a, the duration of any sv-walk Q terminating with e is $a - dep(Q)$. To find walks that minimise this quantity, we define the following cost structure $(C, \gamma, \oplus, \preceq)$. The cost set is $C = \mathbb{R}$, to each temporal edge $e = (u, v, \tau, \lambda)$ we assign as cost its departure time $\gamma(e) = \tau$. The cost combination function $\oplus$ return the first parameter, thus $\tau \oplus \tau' = \tau$. Finally, the cost total order is defined by $\tau \preceq \tau'$ when $\tau \geq \tau'$, meaning that later departure times are preferred. Note that the $\oplus$ definition implies that the cost of a walk $Q = \langle e_1, \ldots, e_k\rangle$ depends only on its first edge $e_1 = (u_1, v_1, \tau_1, \lambda_1)$ and is given by $\gamma_Q = \gamma(e_1) = \tau_1 = dep(Q)$. The isotonicity property is trivially satisfied: for any $\tau_1, \tau_2, \tau \in \mathbb{R}$, such that $\tau_1 \preceq \tau_2$, as $\tau_1 \oplus \tau = \tau_1$ and $\tau_2 \oplus \tau = \tau_2$, we have $\tau_1 \oplus \tau \preceq \tau_2 \oplus \tau$.

Running Algorithm 2 results in a set A'_v for each node v where every possible edge e terminating an sv-walk is associated to a walk Q minimum cost among those ending with e, that is maximising $dep(Q)$. The duration of Q, that is $arr(Q) - dep(Q) = a - dep(Q)$, where e is the arrival time of e, is thus guaranteed to be minimum among all sv-walks ending with edge e, since they all have arrival time a. We can finally scan all pairs in A'_v to obtain an sv-walk with shortest duration. Corollary 1 is thus a consequence of Theorem 4.

Input: A doubly-sorted representation (E^{dep}, E^{arr}) of a temporal graph G with waiting-time constraints (α, β) and cost structure $(C, \gamma, \oplus, \preceq)$ satisfying isotonicity, and a source node s.

Output: Minimum cost of an (E^{dep}, E^{arr})-respected sv-walk for each node v and for each (s, E^{dep}, E^{arr})-reachable edge e with head v.

For each node v, generate the list E_v^{dep} by bucket sorting E^{dep}.
For *each node* v **do**
    Set $A'_v := \emptyset$. `/* List of pairs of` s`-reachable edge and cost. */`
    Set $\mathcal{I}_v := \emptyset$. `/* Doubly linked list of consecutive intervals of` E_v^{dep}`. */`
    Set $(l_v, r_v) := (1, 0)$. `/* Overall interval of` E_v^{dep} `spanned by` $\mathcal{I}_v$`. */`
Set best extendable cost $B[e] := \perp$ and parent pointer $P[e] := \perp$ for each edge $e \in E^{arr}$.
For *each edge* $e = (u, v, \tau, \lambda)$ *in* E^{arr} **do**
    Let i be the index of e in E_u^{dep}.
    If $i \geq l_u$ **then** FinalizeCosts(u, i) `/* Obtain` $B[e]$ `in particular. */`
    If $u = s$ *or* $B[e] \neq \perp$ **then**
        `/* Get the minimum cost` c `of a walk having` e `as last edge: */`
        If $u = s$ *and* $(B[e] = \perp$ *or* $\gamma(e) \prec B[e] \oplus \gamma(e))$ **then** $c := \gamma(e)$ and $P[e] := e$
        else $c := B[e] \oplus \gamma(e)$.
        Append (e, c) to A'_v.
        `/* Find the interval` (l, r) `of edges in` E_v^{dep} `that extend` e`: */`
        Let $a = \tau + \lambda$ be the arrival time of e.
        Let $l \geq l_v$ be the first index of an edge $(v, w, \tau', \lambda') \in E_v^{dep}$ such that $\tau' \geq a + \alpha_v$ (set $l := |E_v^{dep}| + 1$ if no such index exists).
        Let $r \geq r_v$ be the last index of an edge $(v, w, \tau', \lambda') \in E_v^{dep}$ such that $\tau' \leq a + \beta_v$ (set $r := r_v$ if no such index exists).
        `/* Remove from` $\mathcal{I}_v$ `intervals preceding` l `and set` $l_v := l$`: */`
        FinalizeCosts$(v, l - 1)$
        `/* Remove from` $\mathcal{I}_v$ `intervals with cost greater than` c`: */`
        Set $l_c := \max\{l, r_v + 1\}$. `/* First index in` (l, r) `after` $\mathcal{I}_v$`. */`
        While $\mathcal{I}_v \neq \emptyset$ *has last interval* $I' = (l', r', c', e')$ *satisfying* $c \prec c'$ **do**
            Remove I' from $\mathcal{I}_v$ and update $l_c := l'$.
        `/* Associate cost` c `and parent` e `to edges in` $E_v^{dep}[l_c : r]$`: */`
        If $l_c \leq r$ **then** append interval $I = (l_c, r, c, e)$ to $\mathcal{I}_v$.
        Set $r_v := r$.
Return *the lists* $(A'_v)_{v \in V}$.

Algorithm 2: Computing, for each node v and each (s, E^{dep}, E^{arr})-reachable edge e with head v, the minimum cost of any (E^{dep}, E^{arr})-respected sv-walk ending with e.

```
Procedure FinalizeCosts(v, j)
    While the first interval I = (l, r, c, e) in I_v satisfies l ≤ j do
        Let l' = min{r, j}.
        For each edge f = (v, w, τ, λ) in E_v^dep[l : l'] do B[f] := c and P[f] := e.
        If l' = r then remove I from I_v else update I := (j + 1, r, c, e).
    Set l_v := j + 1.
```

Algorithm 3: Set the best extendable cost and parent of edges in $E_v^{dep}[l_v : j]$ and remove corresponding intervals from $\mathcal{I}_v$.

Example of execution We will now present an example of execution of Algorithm 2 for the specific case of computing shortest duration walks in the temporal graph represented in Figure 2.14 with source node s.

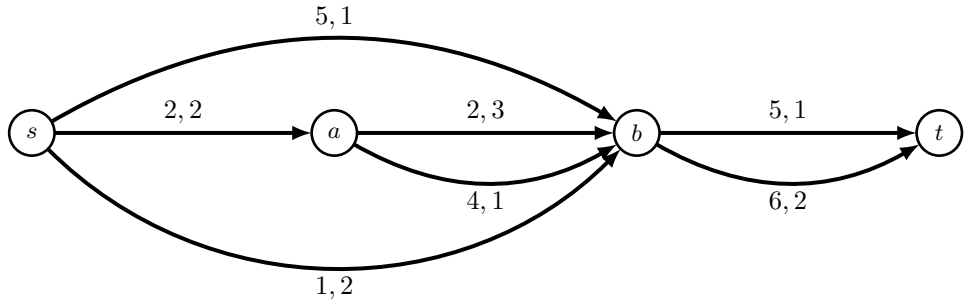

Figure 2.14: A point availability temporal graph with waiting time constraints, where $\alpha_s = \alpha_a = \alpha_b = \alpha_t = 0$, $\beta_s = \beta_a = \beta_t = +\infty$ and $\beta_b = 2$.

The algorithm receives in input the list of temporal edges:

$$E^{arr} = \{(s, b, 1, 2), (s, a, 2, 2), (a, b, 4, 1), (a, b, 2, 3), (s, b, 5, 1), (b, t, 5, 1), (b, t, 6, 2)\}$$

and

$$E^{dep} = \{(s, b, 1, 2), (a, b, 2, 3), (s, a, 2, 2), (a, b, 4, 1), (s, b, 5, 1), (b, t, 5, 1), (b, t, 6, 2)\}.$$

In the preprocessing step, the algorithm computes the following lists from E^{dep}:

- $E_s^{dep} = \{(s, b, 1, 2), (s, a, 2, 2), (s, b, 5, 1)\}$,
- $E_a^{dep} = \{(a, b, 2, 3), (a, b, 4, 1)\}$,
- $E_b^{dep} = \{(b, t, 5, 1), (b, t, 6, 2)\}$.

During the execution the algorithm will work on such lists rather then E^{dep}. Moreover, still in the preprocessing step, all sets A'_v and sets of intervals $\mathcal{I}_v$ are empty, and for each temporal edge its best extendable cost and parent pointer are set to $\perp$. We will follow what the algorithm does iteration by iteration: in each figure the temporal edge currently

scanned is coloured in red and the temporal edges that extends it are coloured in blue. We will focus on the more meaningful operations and for example we do not mention the calls to FinalizeCosts when they do not perform any action.

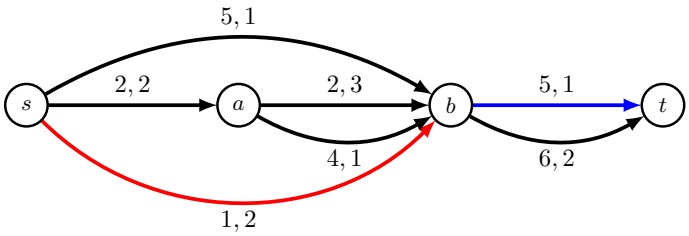

Figure 2.15: Iteration 1.

Iteration 1. In the first iteration we scan edge $(s, b, 1, 2)$, whose tail is the source. Its extendable cost $B[(s, b, 1, 2)]$ is still set to $\perp$ as for every other temporal edge, and thus c is set to $\gamma((s, b, 1, 2)) = 1$. The pair $((s, b, 1, 2), 1)$ is added to A'_b. Because of the waiting time constraint $\beta_b = 2$, the only edge that extends $(s, b, 1, 2)$ is $(b, t, 5, 1)$. Since $\mathcal{I}_b$ is empty, we create a new interval I_b^1 that contains the index of $(b, t, 5, 1)$ in E_b^{dep}, associated to cost 1 and to the edge $(s, b, 1, 2)$.

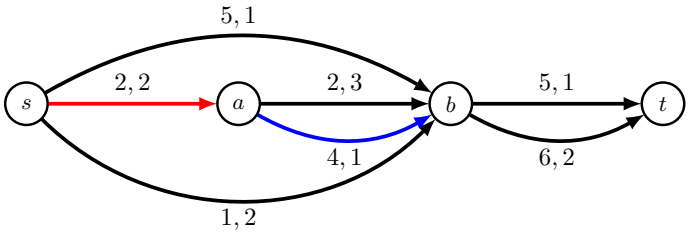

Figure 2.16: Iteration 2.

Iteration 2. We scan edge $(s, a, 2, 2)$. Again, it is an edge from the source with no defined best extendable cost, thus c is set to $\gamma((s, a, 2, 2)) = 2$. The pair $((s, a, 2, 2), 2)$ is added to A'_a. The only edge that extends $(s, a, 2, 2)$ is $(a, b, 4, 1)$. Edge $(a, b, 2, 3)$ is finalized as it precedes $(a, b, 4, 1)$ in E_a^{dep}. Since it does not belong to an interval in the moment it is finalized, its best extendable cost is still $\perp$. Moreover, since $\mathcal{I}_a$ is empty, we create a new interval I_a^1 that contains the index of $(a, b, 4, 1)$ in E_a^{dep}, associated to cost 2 and to edge $(s, a, 2, 2)$.

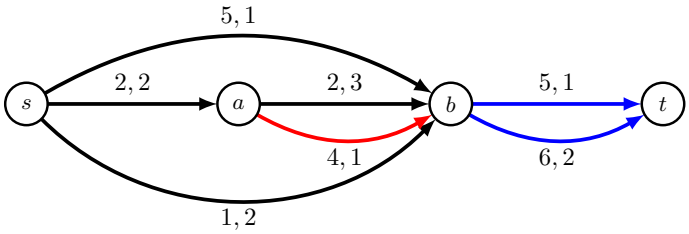

Figure 2.17: Iteration 3.

Iteration 3. We scan edge $(a, b, 4, 1)$. It gets finalized and thus its best extendable cost is set to the cost of the interval I_a^1 that contains it, yielding $B[(a, b, 4, 1)] = 2$ and $P[(a, b, 4, 1)] = (s, a, 2, 2)$. Since its tail is not the source, c is set to $2 \oplus 4 = 2$. The pair $((a, b, 4, 1), 2)$ is added to A'_b. Both edges in the graph with tail b extends the edge that is currently scanned. Since interval I_b^1 is associated to cost 1, while the value of c is currently 2 which satisfies $2 \prec 1$, I_b^1 is removed. We create a new interval, let us call it I_b^2, containing the indexes of both $(b, t, 5, 1)$ and $(b, t, 6, 2)$, it is associated with cost 2 and edge $(a, b, 4, 1)$.

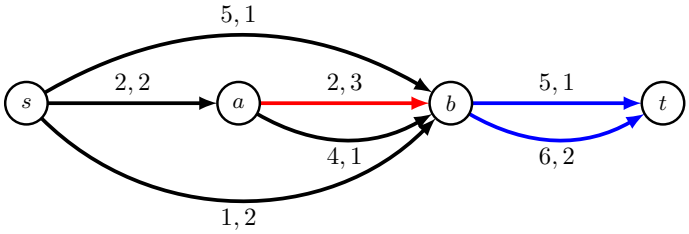

Figure 2.18: Iteration 4.

Iteration 4. We scan edge $(a, b, 2, 3)$. As its tail is not the source and its best extendable cost is still equal to $\perp$, the algorithm does not perform any operation.

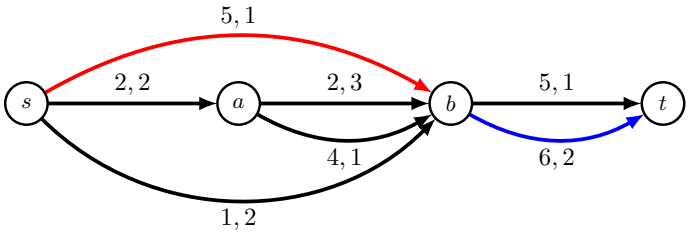

Figure 2.19: Iteration 5.

Iteration 5. We scan edge $(s, b, 5, 1)$. Its tail is the source and its best extendable cost is undefined, thus c is set to $\gamma((s, b, 5, 1)) = 5$. The pair $((s, b, 5, 1), 5)$ is added to A'_b. The only edge that extends it is $(b, t, 6, 2)$. At this point E_b^{dep} is finalized until $(b, t, 6, 2)$, which means that $(b, t, 5, 1)$ is finalized, by setting $B[(b, t, 5, 1)] = 2$ and $P[(b, t, 5, 1)] = (a, b, 4, 1)$, as this values were associated to I_b^2. Moreover, I_b^2 is updated to contain the sole index of $(b, t, 6, 2)$. Finally, as $c = 5 \prec 2$ interval I_b^2 is removed and a new interval is added, let us

call it I_b^3, containing the index of $(b, t, 6, 2)$ in E_b^{dep}, associated to cost 5 and edge $(s, b, 5, 1)$.

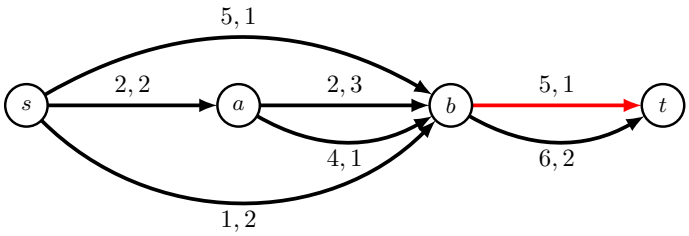

Figure 2.20: Iteration 6.

Iteration 6. We scan edge $(b, t, 5, 1)$. It has been finalized in the previous iteration and $B[(b, t, 5, 1)] = 2$. The pair $((b, t, 5, 1), 2)$ is added to A'_t. Since there are no edges extending it, the algorithm does not perform any further operation.

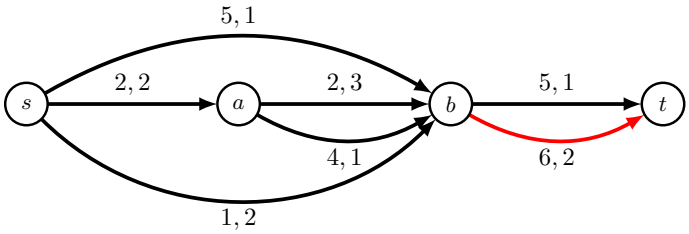

Figure 2.21: Iteration 7.

Iteration 7. In the last iteration, edge $(b, t, 6, 2)$ is scanned. It gets finalized, thus its best extendable cost is set to $B[(b, t, 6, 2)] = 5$ and the parent pointer is set to $P[(b, t, 6, 2)] = (s, b, 5, 1)$, as these values are associated to I_b^3 . The pair $((b, t, 6, 2), 5)$ is added to A'_t. Since there are no edges extending it, the algorithm does not perform any further operation.

Let us suppose we are interested in shortest duration temporal from s to t. We scan A'_t. The first pair is $((b, t, 5, 1), 2)$: this means that there exists a temporal walk ending with $(b, t, 5, 1)$ and that the latest departure time of walks ending with such edge is 2, and this correspond to a duration of $arr((b, t, 5, 1)) - 2 = 4$. The second pair is $((b, t, 6, 2), 5)$: this means that there exists a temporal walk ending with $(b, t, 6, 2)$ and that the latest departure time of walks ending with this edge is 5, corresponding to a duration of $arr((b, t, 6, 2)) - 5 = 3$. To conclude, the minimum duration of a temporal walk from s to t is 3, and if we are interested in the temporal walk itself we can follow that parent pointer starting from $(b, t, 6, 1)$ and obtain the walk $\langle (s, b, 5, 1), (b, t, 6, 2) \rangle$.

The correctness of the algorithm mainly follows from the following lemma.

Lemma 1. *After the kth iteration of Algorithm 2, if an edge f with tail v is associated to cost c, either through an interval $(l, r, c, e) \in \mathcal{I}_v$ containing the index of f in E_v^{dep} or by the value $c = B[f]$ when $B[f] \neq \perp$, then c is the best extendable cost of f with respect to the subset $E^{arr}[1 : k]$ inducing graph G_k. Moreover, the edge e_k scanned at the kth iteration gets associated to cost c in A'_v if and only if it is an (s, E^{dep}, E^{arr})-reachable edge and c is the minimum cost of any (E^{dep}, E^{arr})-respected sv-walk ending with e_k.*

Note that exactly one of the three following cases occurs: f has no associated cost, or f is in an interval of $\mathcal{I}_v$, or we have $B[f] \neq \perp$. This is due to the way we maintain the value l_v which is always the leftmost bound of an interval of $\mathcal{I}_v$: once a value $B[f]$ is set for an edge f with tail v by a call to FinalizeCosts(v, j), l_v is updated to a value greater than j, and f cannot appear in an interval of $\mathcal{I}_v$ anymore.

Proof. We prove the statement by induction on k. As there are no edges and no walks in G_0 and no edge has initially an associated cost, the statement holds for $k = 0$. Assume that the statement holds for $k-1$ and let us prove it for k. Recall that the best extendable cost of an edge f with respect to $E^{arr}[1:k]$ is the minimum cost of an sv-walk Q in G_k that f extends and such that $Q.f$ is (E^{dep}, E^{arr})-respected (where v denotes the tail of f). By the induction hypothesis, we must update the cost associated to f only when there is such a walk Q with cost c which is in G_k but not in G_{k-1} and such that c is lower than the cost associated to f after the previous iteration. This can occur only when Q contains the edge $e_k = E^{arr}[k] = (u, v, \tau, \lambda)$ which is scanned at the kth iteration. Moreover, as $Q.f$ is (E^{dep}, E^{arr})-respected, so is Q. Its edges thus appear in order in E^{arr} and e_k must be its last edge. It is thus sufficient to consider the minimum cost c^* of an (E^{dep}, E^{arr})-respected sv-walk Q ending with e_k and compare it with the cost associated to edges f that extend e_k and such that $Q.f$ is (E^{dep}, E^{arr})-respected.

We first show that the value c computed at Lines 11-12 is indeed c^*. Consider an (E^{dep}, E^{arr})-respected sv-walk Q ending with e_k and having cost c^*. In the case where Q has at least two edges, let Q' denote the prefix of Q excluding e_k. The induction hypothesis and the call to FinalizeCosts(u, i) at Line 9 then ensure that $B[e_k]$ is set to the minimum cost of such a walk Q' that e_k extends and such that $Q'.e_k$ is (E^{dep}, E^{arr})-respected. By isotonicity, this implies that $B[e_k] \oplus \gamma(e_k)$ is the minimum cost of an (E^{dep}, E^{arr})-respected sv-walk ending with e_k and having at least two edges. In the case where Q has one edge, we have $Q = \langle e_k \rangle$, and e_k must be an edge from s and the cost of Q is $\gamma(e_k)$. In both cases, the test at Line 10 passes when e_k is an (s, E^{dep}, E^{arr})-reachable edge and the computation of c at Lines 11-12 sets c to the minimum of $\gamma(e_k)$ and $B[e_k] \oplus \gamma(e_k)$ when both cases occur, ensuring that $c = c^*$ is the minimum cost of any (E^{dep}, E^{arr})-respected sv-walk ending with e_k. Moreover, Line 13 then ensures that e_k gets associated to cost $c = c^*$ in A'_v.

We now show that the set F of edges f that extend e_k and such that $Q.f$ is (E^{dep}, E^{arr})-respected is precisely $E_v^{dep}[l:r]$ where (l, r) are the values computed at Lines 15-16. As these edges extend e_k, their departure time lies within $arr(e_k) + \alpha_v$ and $arr(e_k) + \beta_v$ which correspond to an interval (l', r') of E_v^{dep} as E_v^{dep} is node-departure sorted. We further restrict our attention to those edges f such that $Q.f$ is (E^{dep}, E^{arr})-respected or equivalently $\langle e_k, f \rangle$ is (E^{dep}, E^{arr})-respected when Q is assumed to be (E^{dep}, E^{arr})-respected. That is we should consider only edges f so that no edge e' with tail v satisfies both $f \leq_{E^{dep}} e'$ and $e' \leq_{E^{arr}} e_k$. Let l'' be the highest index in E_v^{dep} of such an edge e'. The call to FinalizeCosts(u', i') at Line 9, where $u' = v$ is the tail of such an edge e' scanned before e_k and $i' \leq l''$ is the index of e' in E_v^{dep}, ensures that l_v is at least $i' + 1$. When $u = v$, e_k is itself such an edge e', and the call to FinalizeCosts(u, i) at Line 9 where $i \leq l''$ is the index of e_k ensures that l_v is at least $i+1$. We thus have $l_v \geq l'' + 1$. Note also that l_v was updated to $l'' + 1$ at most in these calls from Line 9. Moreover, each call to FinalizeCosts(v, j) at Line 17 for an edge $e' <_{E^{arr}} e_k$ with head v was made for an arrival time $arr(e') \leq arr(e_k)$ as E^{arr} is node-arrival sorted. This ensures that the argument j of such a call was at most $\max\{l'' + 1, l'\}$. Similarly, the update of r_v at the end of the corresponding iteration was at most r'. We thus conclude that we have $l'' + 1 \leq l_v \leq \max\{l', l'' + 1\}$ and $r_v \leq r'$ at the beginning of the kth iteration.

The computation of l and r at Lines 8-13 thus implies $l = \max\{l', l'' + 1\}$ and $r = r'$ and the interval (l, r) of E_v^{dep} indeed corresponds to edges of F.

We finally show that each edge $f \in F$ gets associated to its best extendable cost with respect to $E^{arr}[1 : k]$. This mainly relies on the induction hypothesis and the fact that $c = c^*$ is the minimum cost of a walk Q in G_k and not in G_{k-1} that f extends and such that $Q.f$ is (E^{dep}, E^{arr})-respected. Among those edges $f \in F$ which are already associated with a cost c', the removal of intervals at Lines 18-20 ensures that we modify their associated cost only when c' is greater than $c = c^*$. This relies on the property that $\mathcal{I}_v$ is sorted by non-decreasing cost which is an invariant of the algorithm as the eventual interval (l_c, r, c, e) added at the end of $\mathcal{I}_v$ at Lines 21 has cost c which is greater or equal to the cost of remaining intervals. The induction hypothesis and the optimality of c ensure that the best extendable cost of these edges is c. The update of bound l_c at Line 20 ensures that these edges get associated to cost c. All edges in F that were not previously associated to a cost are those in interval $(\max\{l'' + 1, l', r_v + 1\}, r') = (\max\{l, r_v + 1\}, r)$ which is included in (l_c, r) as l_c is initialized to $\max\{l, r_v + 1\}$ at Line 18 and can only decrease by the updates at Line 20. These edges also get associated to c through interval (l_c, r, c, e), and it is their best extendable cost by optimality of c. Finally, all edges in F that were associated to a cost $c' \prec c$ remain associated to the same cost which is their best extendable cost by the induction hypothesis. □

We can now state the following.

Proposition 1. *Given a doubly-sorted representation (E^{dep}, E^{arr}) of a temporal graph $G = (V, E, \alpha, \beta)$ with cost structure $(C, \gamma, \oplus, \preceq)$ satisfying isotonicity, and a source node s, Algorithm 2 computes in linear time and space, for each node v and each (s, E^{dep}, E^{arr})-reachable edge e with head v, the minimum cost of any (E^{dep}, E^{arr})-respected sv-walk ending with e.*

Proof. The correctness of the algorithm follows directly from Lemma 1. The reason is that any (E^{dep}, E^{arr})-respected sv-walk Q ending with an edge e must have edges appearing in order in E^{arr} so that if k is the index of e in E^{arr}, all edges of Q are in $E^{arr}[1 : k]$, and Q is also a walk in G_k.

Let us turn to the complexity analysis. Each edge $e \in E^{arr}$ is scanned only once. For all nodes v, each edge $f \in E_v^{dep}$ with index i is finalized at most once: the first time FinalizeCosts(v, j) is called with a value $j \geq i$. The update of l_v to $j+1$ in FinalizeCosts(v, j) ensures that f is never finalized again. Computing the value of l at Line 15 takes $O(l - l_v)$ time, which thanks to the update of l_v to l in the call to FinalizeCosts$(v, l - 1)$ results in amortized time of $O(|E_v^{dep}|)$. Similarly, computing the value of r takes $O(r - r_v)$ time, which thanks to the update of r_v to r results in amortized time of $O(|E_v^{dep}|)$. In addition, at most one interval is created at each iteration and later removed. The number of times we modify the left bound of an interval is bounded by the number of times we update l_v which is $|E_v^{dep}|$ at most. As $\sum_{v \in V} |E_v^{dep}| = |E|$, Algorithm 2 runs in linear time assuming that operations with $\oplus$ and $\preceq$ can be computed in constant time. Finally, let us notice that for all nodes v, $\mathcal{I}_v$ contains at most $|E_v^{dep}|$ intervals and the set A'_v has size bounded by the number of temporal edges with head v. We thus have $\sum_{v \in V} |\mathcal{I}_v| \leq |E|$ and $\sum_{v \in V} |A'_v| \leq |E|$. The space complexity of Algorithm 2 is thus linear. □

In the remaining part of the section, we show that starting from a time-expanded representation it is possible to compute a doubly-sorted representation (E^{dep}, E^{arr}) satisfying

some additional properties which guarantee that any temporal walk is (E^{dep}, E^{arr})-respected and thus allowing Algorithm 2 to indeed compute minimum cost walks. We start by defining the following looser notion of extending. We say that an edge $f = (x, y, \tau, \lambda)$ *half-extends* an edge $e = (u, v, \tau', \lambda')$ when $x = v$ and $arr(e) + \alpha_x \leq \tau$. Note that f half-extends e whenever f extends e. Let us now introduce some orderings of temporal edges with respect to certain temporal criteria. We say that an ordering E^{ord} of all edges is *half-extend-respecting* when for any pair $e, f \in E$ of edges such that f half-extends e, then e appears before f in E^{ord} which is denoted by $e <_{E^{ord}} f$. We also write $e \leq_{E^{ord}} f$ for $e <_{E^{ord}} f$ or $e = f$. Note that the edges of any walk Q in G must appear in order in such an ordering E^{ord} as each edge of Q half-extends the edge preceding it. We also say that a doubly-sorted representation (E^{dep}, E^{arr}) is *half-extend-respecting* when E^{arr} is additionally half-extend-respecting.

We can now state the following equivalence.

Proposition 2. *Let $G = (V, E, \alpha, \beta)$ be a temporal graph.*

A. If G is zero-acyclic and a time-expanded representation of G is given, it is possible to compute in linear time and space a doubly-sorted representation (E^{dep}, E^{arr}) of G such that E^{dep} and E^{arr} are both half-extend-respecting.

B. Given a doubly-sorted representation (E^{dep}, E^{arr}) of G it is possible to compute in linear time and space a time-expanded representation of G.

Proof. In order to prove Proposition 2.A, we design an algorithm that computes a node-arrival-sorted half-extend-respecting list from a time-expanded representation of a zero-acyclic temporal graph. It is inspired by Kahn's algorithm for computing a topological ordering of a directed acyclic graph [41].

We first define a notion of extending for arcs in D, where D is the time-expanded representation. Given two arcs f_1, f_2 in $F^c \cup F^w$, we say that f_2 *arc-extends* f_1 when the head v_ν of f_1 is also the tail of f_2 and we have $f_1 \in F^w$ or $f_2 \in F^w$ or $\alpha_v = 0$ (v is the node whose copy v_ν is the head of f_1). In particular, when f_1 and f_2 are both connection arcs, we must have $\alpha_v = 0$. Note that for every pair e_1, e_2 of temporal edges corresponding respectively to two connection arcs f_1, f_2 in F^c, and such that e_2 extends e_1, there must exist a path P in D starting with f_1, ending with f_2, and containing possibly intermediate waiting arcs in F^w. When $arr(e_1) = dep(e_2)$, P contains only f_1 and f_2, and the minimum waiting time α_v at the head v of e_1 must be zero since e_2 extends e_1. In all cases, each arc in P arc-extends the preceding one according to our new definition. We say that an ordering F^{ord} of the arcs of D is *arc-extend-respecting* when we have $f_1 <_{F^{ord}} f_2$ whenever f_2 arc-extends f_1 for any pair of arcs $f_1, f_2 \in F^c \cup F^w$. It is thus sufficient to produce an arc-extend-respecting ordering F^{ord} of $F^c \cup F^w$ to obtain a half-extend-respecting ordering of the temporal edges according the respective positions of their corresponding arcs in F^{ord}.

The main idea of the algorithm is to produce such an ordering by iteratively removing an arc from D so that no other remaining arc arc-extends it. Each time an arc is removed, it is prepended to the list F^{ord} which is initially empty. When a node v_ν has out-degree zero, we can safely remove all the arcs entering it. Repeating this would suffice when D is acyclic. However, it may contain a cycle. This can only occur when all nodes in the cycle have same time label τ as each arc (u_τ, v_ν) satisfies $\tau \leq \nu$. This implies that all the arcs of the cycle must be in F^c. In such a case, the zero-acyclicity of G ensures that at least one node v_ν of the cycle is a copy a node v of the temporal graph with waiting-time constraint $\alpha_v > 0$ as otherwise this cycle would correspond to a zero-cycle in G. When no node has out-degree zero, the algorithm thus selects any node v_ν having no out-arc in F^w and satisfying $\alpha_v > 0$,

and then removes all its in-arcs that are in F^c. Note that all out-arcs of v_ν are then in F^c and none of them arc-extends these in-arcs by the choice of v_ν such that $\alpha_v \neq 0$. Such a node must exist when G is zero-acyclic and no remaining node has out-degree zero as there must then exist a cycle among nodes with maximum time label τ while no remaining waiting arc can lead to a copy with time label greater than τ. As the algorithm can always progress, it terminates when all arcs have been removed. See Algorithm 4 for a formal description, where $\delta^c_{out}(v_\tau)$ is the number of out-neighbors of v_τ through an arc in F^c, $\delta^w_{out}(v_\tau)$ is the number of out-neighbors of v_τ through an arc in F^w. Furthermore, we denote with $N^c_{in}(v_\tau)$ the set of in-neighbors w_ν of v_τ such that $(w_\nu, v_\tau) \in F^c$.

Input: A time-expanded representation $D = (W, F^c \cup F^w)$ of a zero-acyclic temporal graph $G = (V, E, \alpha, \beta)$, given as adjacency lists $N^c_{in}(v_\tau), Pred^w(v_\tau)$ for each $v_\tau \in W$.
Output: An arc-extend-respecting ordered list F^{ord} of the arcs in D.

Compute $\delta^c_{out}(v_\tau)$ and $\delta^w_{out}(v_\tau)$ for each node $v_\tau \in W$.
Compute the set S of nodes v_τ such that $\delta^c_{out}(v_\tau) = 0$ and $\delta^w_{out}(v_\tau) = 0$.
Compute the set S' of nodes v_τ such that $\delta^w_{out}(v_\tau) = 0$ and $\alpha_v > 0$.
Set $F^{ord} := \emptyset$. `/* Arc-extend-respecting ordered list. */`
While $S \cup S' \neq \emptyset$ **do**
    If $S \neq \emptyset$ **then**
        Select any $v_\tau \in S$.
        For *each node* u_ν *in* $N^c_{in}(v_\tau) \cup Pred^w(v_\tau)$ **do** RemoveArc(u_ν, v_τ).
        Remove v_τ from D.
    else
        Select any $v_\tau \in S'$.
        For *each node* u_ν *in* $N^c_{in}(v_\tau)$ **do** RemoveArc(u_ν, v_τ).
Return F^{ord}

Procedure RemoveArc(u_ν, v_τ)
    Prepend (u_ν, v_τ) to F^{ord} and remove it from D.
    Update accordingly the degrees of u_ν and the sets S, S'.

Algorithm 4: Computing an arc-extend-respecting arc ordering of a time-expanded representation of a zero-acyclic temporal graph.

This algorithm runs in linear time by maintaining $\delta^c_{out}(v_\tau)$, $\delta^w_{out}(v_\tau)$, and $N^c_{in}(v_\tau)$ for each node v_τ. This allows to maintain the set S of nodes with out-degree zero, and the set S' of nodes v_ν having no out-edges in F^w and satisfying $\alpha_v > 0$. A node in S or S' can then be selected in constant time. Each arc removal is performed with constant-time updates of out-degrees, in-neighbours, sets S and S'. As each node is considered at most twice, the overall execution thus takes linear time.

Each time an arc is prepended to F^{ord} by the algorithm, all arcs that arc-extend it must have already been removed and are thus already in F^{ord}. The resulting ordering F^{ord} is thus arc-extend-respecting. We then obtain a half-extend-respecting ordering E^{arr} of the temporal edges by removing waiting arcs from F^{ord} and replacing each remaining connection arc (u_τ, v_ν) by its corresponding temporal edge $(u, v, \tau, \nu - \tau)$. Note that E^{arr} is also node-arrival sorted. This is due to the fact that for any node v and any time labels τ associated to v, arcs entering the copy v_τ cannot be removed as long as it has an out-edge in F^w, and

this out-edge is removed only when its next copy v_ν has out-degree zero. This implies that all arcs entering a copy v_μ with $\mu > \tau$ are prepended to F^{ord} before all arcs entering v_τ.

A similar algorithm can be symmetrically designed to obtain an ordering E^{dep} which is half-extend-respecting and node-departure sorted. When producing a temporal edge in E^{arr} and in E^{dep}, we can associate it to the index of its corresponding arc in F^c so that we can easily construct pointers linking each edge in E^{arr} to its copy in E^{dep}.

On the other side, to prove Proposition 2.B, we can use the following procedure. We obtain for each $v \in V$, the lists E_v^{dep} and E_v^{arr} through bucket sorting. We merge these two lists into a single list E_v^{event} for each node v. A linear scan of E_v^{event} then produces all the sorted copies of v in W, and associates to each edge having v as head or tail the corresponding copy of v. A linear scan of all temporal edges then allows to construct F^c in linear time. Finally, we construct F^w by scanning E_v^{event} for each node v. □

Interestingly, the above result implies that any zero-acyclic temporal graph admits a half-extend-respecting ordering. Conversely, the existence of a half-extend-respecting ordering obviously prevents the presence of zero-cycles and implies zero-acyclicity. We thus obtain the following statement grounding the notion of half-extend-respecting ordering as a characterization of zero-acyclicity.

Proposition 3. *A temporal graph $G = (V, E, \alpha, \beta)$ is zero-acyclic if and only if there exists a half-extend-respecting ordering of its edges.*

Moreover, notice that a fully doubly-sorted representation (E^{dep}, E^{arr}), is also a doubly-sorted half-extend-respecting representation.

Proposition 2 also implies that a half-extend-respecting doubly-sorted representation (E^{dep}, E^{arr}) can be computed in linear time and space from any doubly-sorted representation of a zero-acyclic temporal graph by constructing its time-expanded representation as an intermediate step.

Now we show that any walk is (E^{dep}, E^{arr})-respected when (E^{dep}, E^{arr}) is half-extend-respecting.

Lemma 2. *Let G be a zero-acyclic temporal graph and let (E^{dep}, E^{arr}) be a half-extend-respecting doubly-sorted representation of G. Then any walk in G is (E^{dep}, E^{arr})-respected and any s-reachable edge is an (s, E^{dep}, E^{arr})-reachable edge.*

Proof. Consider a walk Q in G and consider two consecutive edges e, f of Q. We have to prove that for any edge e' with same tail v as f and satisfying $f \leq_{E^{dep}} e'$, we have $e <_{E^{arr}} e'$. First, we have $arr(e) + \alpha_v \leq dep(f)$ as f extends e. Second, $f \leq_{E^{dep}} e'$ implies $dep(f) \leq dep(e')$ as E^{dep} is node-departure sorted. Combining both inequalities, we get $arr(e) + \alpha_v \leq dep(e')$, that is e' half-extends e. We must thus have $e <_{E^{arr}} e'$ as E^{arr} is half-extend-respecting. □

Theorem 4 is a direct consequence of Lemma 2, Proposition 1 and Proposition 2.

2.4 Solving classical optimal temporal walks problems

2.4.1 Single-source fewest-edges walks.

As a very basic example, optimizing the number of edges in Algorithm 2 is straightforward: it suffices to consider the cost structure $(\mathbb{N}, \gamma, +, \leq)$ associated to integers ordered as usual and where each edge e has cost $\gamma(e) = 1$, the combination function being addition. It obviously satisfies isotonicity and the cost of a temporal walk coincides with its number of temporal edges. Using Theorem 4 thus implies that the single-source fewest-edges walk problem, can be solved in linear time and space.

2.4.2 Minimum-overall-waiting-time walks.

The overall waiting time of a walk $Q = \langle e_1 = (v_0, v_1, \tau_1, \lambda_1), \ldots, e_k = (v_{k-1}, v_k, \tau_k, \lambda_k)\rangle$ is defined as $arr(Q) - dep(Q) - \sum_{i=1}^{k} \lambda_i$. In order to find for each temporal edge e a walk that minimise such quantity among the walks from the source ending with e, we define the following cost structure $(C, \gamma, \oplus, \preceq)$. The cost set is $C = \mathbb{R} \times \mathbb{R}_{\geq 0}$. Given an edge $e = (u, v, \tau, \lambda)$, we define its cost as $\gamma(e) = (\tau, \lambda) \in \mathbb{R} \times \mathbb{R}_{\geq 0}$. We define the cost combination function $\oplus$ by $(\tau, \lambda) \oplus (\tau', \lambda') = (\tau, \lambda + \lambda')$. Finally, we define the cost total order by $(\tau, \lambda) \preceq (\tau', \lambda')$ when $\tau + \lambda \geq \tau' + \lambda'$. It is trivial to verify the isotonicity property for this cost structure and that the cost associated to a walk $Q = \langle e_1 = (v_0, v_1, \tau_1, \lambda_1), \ldots, e_k = (v_{k-1}, v_k, \tau_k, \lambda_k)\rangle$ is $\gamma_Q = (dep(Q), \sum_{i=1}^{k} \lambda_i)$. If we consider two walks Q and Q' with same arrival time a, if $\gamma_Q \prec \gamma_{Q'}$ then the overall waiting time of Q, that is $a - dep(Q) - \sum_{i=1}^{k} \lambda_i$, is lower than the overall waiting time of Q'. This guarantees that the cost we associate to an edge e corresponds to a walk that, once extended with e, has minimum overall waiting time among the walks from the source that end with e. We can thus obtain the minimum overall waiting time of an sv-walk as $W^* = \min_{(e,(\tau,\lambda)) \in A'_v} arr(e) - \tau - \lambda$. Notice that the cost associated to a walk is not the overall waiting time of the walk, however, it is a quantity that when minimised corresponds to optimising the overall waiting time for walks with same arrival time.

2.4.3 Shortest-fastest walks

A walk that among the ones with shortest duration has the minimum number of edges is called a shortest-fastest walk. To find such walks, we define a cost structure $(C, \gamma, \oplus, \preceq)$ where $C = \mathbb{R} \times \mathbb{N}$. In the first component we proceed as we did for shortest duration in Section 2.3.1 and in the second component as we did for fewest hops in Section 2.4.1. Therefore, given an edge $e = (u, v, \tau, \lambda)$, we define its cost $\gamma(e) \in \mathbb{R} \times \mathbb{N}$ as $\gamma(e) = (\tau, 1)$. We define the cost combination function $\oplus$ by $(\tau, k) \oplus (\tau', k') = (\tau, k + k')$. A walk departing at time τ and having k edges thus has cost (τ, k). We define the cost total order by $(\tau, k) \preceq (\tau', k')$ when $\tau > \tau'$ or $\tau = \tau'$ and $k \leq k'$. Among two walks, the one with latest departure is thus always preferred, and among several walks with same departure time, one with fewest edges is always preferred. Given a source s, Algorithm 2 now outputs for each destination v the set A'_v of all pairs (e, c) such that e is an s-reachable edge with head v and $c = (\tau, k)$ is the minimum cost of an sv-walk ending with e. Note that our cost definition implies that τ is the latest departure time of an sv walk ending with e and k is the minimum number of edges among walks with departure time τ and last edge e. Similarly to the previous paragraph, we thus obtain the shortest duration of an sv-walk as

$D^* = \min_{(e,(\tau,k))\in A'_v} arr(e) - \tau$. And then, we also obtain the minimum number of edges in a shortest duration sv-walk as $k^* = \min_{(e,(\tau,k))\in A'_v : arr(e)-\tau = D^*} k$. The edge e^* for which we get the minimum value allows to obtain, through parent pointers, a walk having duration D^* and k^* edges, that is a shortest-fastest walk. Theorem 4 thus implies that the single-source shortest-fastest walk problem can be solved in linear time and space.

2.4.4 Linear combination of classical criteria

To exemplify the generality of the algebraic approach, we now give an example of cost structure allowing Algorithm 2 to compute optimal temporal walks for the linear combination of criteria used in [5]. Our formalism enables more modularity as all complex updates required by such an exhaustive combination are then encapsulated in operations $\oplus$ and $\prec$. Given a walk $Q = \langle e_1 = (v_0, v_1, \tau_1, \lambda_1), \ldots, e_k = (v_{k-1}, v_k, \tau_k, \lambda_k)\rangle$, we consider the following criteria that we usually seek to minimize:

(1)	$\tau_k + \lambda_k$	arrival time (or foremost)
(2)	$-\tau_1$	departure time (or reverse-foremost)
(3)	$\tau_k + \lambda_k - \tau_1$	duration
(4)	$\sum_{i=1}^{k} \lambda_i$	total travel time
(5)	$\sum_{i=1}^{k} c(e_i)$	total cost (each edge $e \in E$ is associated to a cost $c(e) \in \mathbb{R}$)
(6)	k	number of edges (or fewest-hops)
(7)	$\sum_{i=1}^{k-1} \tau_{i+1} - (\tau_i + \lambda_i)$	overall waiting time

Given $\delta_1, \ldots, \delta_7 \in \mathbb{R}$, the *linear combined cost* of Q is defined in [5] as:

$$lin(Q) = \delta_1(\tau_k + \lambda_k) + \delta_2(-\tau_1) + \delta_3(\tau_k + \lambda_k - \tau_1) + \delta_4(\sum_{i=1}^{k} \lambda_i) + \delta_5(\sum_{i=1}^{k} c(e_i)) + \delta_6\ k + \delta_7(\sum_{i=1}^{k-1} \tau_{i+1} - (\tau_i + \lambda_i)).$$

It is simply a linear combination of all classical criteria. Note that we do not need to assume non-negativity of costs or scalars $\delta_1, \ldots, \delta_7$, enabling a more general framework than [5]. To optimize such a combined cost, we define the cost structure $(C, \gamma, \oplus, \preceq)$ where $C = \mathbb{R} \times \mathbb{R}$. Given an edge $e = (u, v, \tau, \lambda)$, we define its combined cost $\delta(e) \in \mathbb{R}$ and its cost $\gamma(e) \in \mathbb{R} \times \mathbb{R}$ as:

$$\delta(e) = (\delta_4 - \delta_7)\lambda_i + \delta_5 c(e_i) + \delta_6 \text{ and } \gamma(e) = (\tau, \delta(e)).$$

Observe that they are linked to the linear combined cost of Q by:

$$lin(Q) = (\delta_1 + \delta_3 + \delta_7)\, arr(Q) - (\delta_2 + \delta_3 + \delta_7)\, dep(Q) + \sum_{i=1}^{k} \delta(e_i).$$

Recall that $arr(Q) = \tau_k + \lambda_k$ and $dep(Q) = \tau_1$ are the arrival time and the departure time of Q respectively. We define the cost combination function $\oplus$ by

$$(\tau, \Delta) \oplus (\tau', \Delta') = (\tau, \Delta + \Delta').$$

This definition implies that the cost of Q is then $\gamma_Q = (\tau_1, \sum_{i=1}^{k} \delta(e_i)) = (\tau, \Delta)$ with $\tau = dep(Q)$ and $\Delta = \sum_{i=1}^{k} \delta(e_i)$. We finally define the cost total order $\preceq$ by

$$(\tau, \Delta) \preceq (\tau', \Delta') \text{ when } -(\delta_2 + \delta_3 + \delta_7)\tau + \Delta \leq -(\delta_2 + \delta_3 + \delta_7)\tau' + \Delta'.$$

This order is related to the minimization of $lin(Q)$ for a fixed arrival time a: for all sv-walks Q such that $arr(Q) = a$, minimizing $lin(Q)$ is equivalent to minimizing $-(\delta_2 + \delta_3 + \delta_7)\, dep(Q) + \sum_{i=1}^{k} \delta(e_i) = -(\delta_2 + \delta_3 + \delta_7)\tau + \Delta$ where $(\tau, \Delta) = \gamma_Q$ is the cost of Q. A walk Q with minimum cost according to $\preceq$ thus has minimum value for $lin(Q)$ among all walks with same arrival time.

Note that the cost structure satisfies the isotonicity property: for any costs $(\tau_1, \Delta_1), (\tau_2, \Delta_2)$, $(\tau, \Delta) \in \mathbb{R} \times \mathbb{R}$, we have $(\tau_1, \Delta_1) \oplus (\tau, \Delta) = (\tau_1, \Delta_1 + \Delta)$ and $(\tau_2, \Delta_2) \oplus (\tau, \Delta) = (\tau_2, \Delta_2 + \Delta)$. If $(\tau_1, \Delta_1) \preceq (\tau_2, \Delta_2)$, then we have $-(\delta_2 + \delta_3 + \delta_7)\tau_1 + \Delta_1 \leq -(\delta_2 + \delta_3 + \delta_7)\tau_2 + \Delta_2$. By adding Δ on both sides of the inequality, we obtain $(\tau_1, \Delta_1) \oplus (\tau, \Delta) \preceq (\tau_2, \Delta_2) \oplus (\tau, \Delta)$.

Now running Algorithm 2 with this cost structure from a source node s allows to compute for each destination v the set A'_v of all pairs (e, c) such that e is an s-reachable edge with head v and $c = (\tau, \Delta)$ is the minimum cost of any sv-walk end with e according to our cost structure. The minimum linear combination cost of an sv-walk can then be obtained through a linear scan of A'_v as:

$$\min\{lin(Q) : Q \text{ is an } sv \text{ walk}\} = \min_{(e,(\tau,\Delta)) \in A'_v} (\delta_1 + \delta_3 + \delta_7)\, arr(e) - (\delta_2 + \delta_3 + \delta_7)\, \tau + \Delta.$$

This is due to the fact that for a given arrival time $arr(e)$, minimizing $lin(Q)$ is equivalent to minimizing γ_Q according to $\preceq$, as discussed above, and that A'_v contains a pair for all s-reachable edges with head v. Using the above cost structure, we thus obtain the following corollary.

Corollary 2. *Given either a time-expanded representation or a doubly-sorted representation of a zero-acyclic temporal graph $G = (V, E, \alpha, \beta)$, a source node s, and $\delta_1, \ldots, \delta_7 \in \mathbb{R}$, the single-source minimum-combined-cost walk problem, that is computing for all nodes v an sv-walk with minimum linear combined cost for $(\delta_1, \ldots, \delta_7)$, can be solved in linear time and space.*

2.4.5 Pareto optimal walks.

We now show how our algorithm generalizes the Pareto set computation defined as follows. We first give the definition of the Pareto problem. We say that a pair $(a_1, c_1) \in \mathbb{R} \times C$ *dominates* a pair $(a_2, c_2) \in \mathbb{R} \times C$ if $a_1 < a_2$ and $c_1 \preceq c_2$, or $a_1 \leq a_2$ and $c_1 \prec c_2$. Consider a strict temporal graph $G = (V, E, \alpha, \beta)$ with cost structure $(C, \gamma, \oplus, \preceq)$. An sv-walk Q is *Pareto optimal* if there is no sv-walk Q' such that $(arr(Q'), \gamma_{Q'})$ dominates $(arr(Q'), \gamma_{Q'})$. The *single-source Pareto* problem is then defined as follows. Given a source node $s \in V$, compute, for each destination $v \in V$, the set P_v containing all pairs $(a, c) \in \mathbb{R} \times C$ for which there exists a Pareto optimal walk Q such that $a = arr(Q)$ and $c = \gamma_Q$. The Pareto problem, in terms of generality, can be placed between all-reachable-edge minimum-cost walks and the profile problems. Thus, it is a useful intermediate step to formalize profile computations in the next Section 2.4.6. Moreover, it builds a connection with [27], where the authors considered a Pareto problem based on the criteria of arrival time and number of trips in public transit networks.

The single-source Pareto problem can easily be solved by Algorithm 2 as it computes for each destination v the set A'_v of all pairs (e, c) such that some sv-walks ends with edge e and such that c is the minimum cost among them. In particular, A'_v is ordered by increasing arrival time of the temporal edges. We first replace each pair (e, c) in A'_v with $(arr(e), c)$, and then we keep only the pair with minimum cost among the ones with same arrival time.

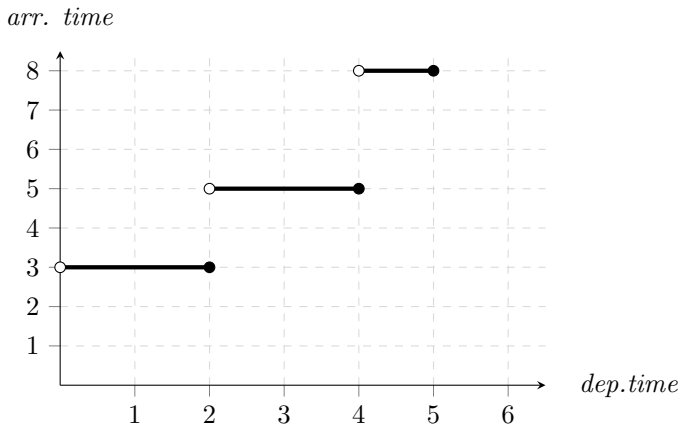

Figure 2.22: The profile function from a source node s to a destination node t. It can be computed starting from the set of Pareto optimal pairs $\{(2,3),(4,5),(5,8)\}$. Each pair (d,a) correspond to a walk departing at time d with earliest arrival time a among the walks departing at time d.

Then, it suffices to remove from A'_v dominated pairs to obtain P_v. This operations can easily be done in linear time due to the order of A'_v.

2.4.6 Profiles.

The profile function from a source node s to a destination node v associates to any starting time τ the earliest arrival time of a walk from s to v departing at time τ or later. The single-source profile problem consists in computing, for a given source s and for every node v, a representation of the profile function from s to v.

A classical representation consists in listing all Pareto optimal pairs (d,a) where d is the departure time of a sv-temporal walk and a is its arrival time. The Pareto optimality of a pair (d,a) means that d is the latest departure time of a walk arriving at time a or before, such that a is also the earliest arrival time at v when departing from s at time d or later. The profile function is indeed the only piecewise-constant non-decreasing function passing through these points. Note that this classical representation was introduced in the setting where waiting is unrestricted. See Figure 2.22 for an example.

This extends naturally to waiting constraints when waiting at the source is unrestricted at starting time, i.e. we impose the waiting constraint only between two consecutive edges of a walk, and consider any walk from s departing at time $\tau' \geq \tau$ as a valid walk when starting at time τ from s. By using the same cost structure introduced to solve shortest duration in Section 2.3.1, and by solving the single-source Pareto problem with that cost structure, we obtain for each node v a list of pairs (a,d), where a is an arrival time of an sv-walk, and d is the latest departure time of a temporal walk with earliest arrival time a. Such a list is ordered both by increasing arrival time and by increasing departure time. One can easily see that it is a representation of the profile function. Note that filtering out dominated pairs relies on the assumption that waiting is unrestricted at the source s when starting from it.

We now consider the setting where waiting at s should be bounded by β_s also when

starting from it, that is a walk from s departing at time τ' is considered as a valid walk when starting at time τ from s, only if we have $\tau + \alpha_s \leq \tau' \leq \tau + \beta_s$. Note that the profile function might not be non-decreasing in this setting. However, we can still solve the all-sources single-destination profile problem as follows. We run our algorithm on the reverse temporal graph where time is reversed, and which is obtained by turning each edge (u, v, τ, λ) into $(u, v, -(\tau + \lambda), \lambda)$. This is equivalent to running a symmetric version of our algorithm for solving the all-sources single-destination problem by scanning edges backwards and computing for each departing edge at a node the earliest arrival time at the destination (which corresponds to the latest departure time in the reverse temporal graph). For a given destination t, this allows to obtain for each node v, and for each edge e departing from v, the earliest arrival time a of walks from v to t starting with e. As this list is sorted by departure time, we can obtain in linear time the pairs (d, a) where a is the minimum arrival time among walks starting with edges departing at time d. Notice that this is different from what we get by a normal execution of our algorithm, as this would normally lead to pairs (d, a) where d is the latest departure time among walks terminating with edges arriving at time a. In particular, now we have a pair for each departure time rather then a pair for each arrival time. Now each pair (d, a) provides the earliest arrival time when starting in interval $(d-\beta_v, d-\alpha_v)$ and using an edge departing at time d. A representation of the profile function from v to t can be obtained by keeping for each window of time covered by multiple overlapping intervals, the lowest earliest arrival time corresponding to such intervals. It can be computed by merging the list of the left bounds of these intervals with the list of their right bounds: scanning the resulting list, while maintaining a queue of currently open left bounds with their associated arrival time, allows to compute for each consecutive interval of starting times, the earliest arrival time. We omit the details of how to get efficiently the minimum arrival time associated to open intervals in the queue.

2.5 Lower bound for the single-source optimal walk problem

We now show that computing optimal temporal walks in linear time somehow requires both orderings needed by our algorithm. More precisely, we define an arrival-sorted representation (resp. a departure-sorted representation) of a temporal graph as a list of its temporal edges sorted by non-decreasing arrival times (resp. non-decreasing departure times). We say that an algorithm is *comparison-only* when it uses only comparisons for deciding whether an edge extends another one, or for deciding which walk has minimum cost among several walks. We show that any comparison-only algorithm optimizing general costs that can encompass overall waiting time, and taking as input either a departure-sorted representation or an arrival-sorted representation, must be slower than linear time by a logarithmic factor at least for some inputs.

Theorem 5. *For each integral n there exists a family of instances $\mathcal{I}_n$ (resp. $\mathcal{I}'_n$) of temporal graphs with unrestricted waiting and strictly positive travel times, given as departure-sorted representations (resp. arrival-sorted representations) with $O(n)$ nodes and $O(n)$ temporal edges, such that any comparison-only deterministic algorithm computing single-source minimum-overall-waiting-time walks from instances in $\mathcal{I}_n$ (resp. $\mathcal{I}'_n$) has time complexity $\Omega(n \log n)$. Moreover, for any comparison-only randomized algorithm computing single-source minimum-overall-waiting-time walks from instances in $\mathcal{I}_n$ (resp. $\mathcal{I}'_n$), there exists an instance in $\mathcal{I}_n$ (resp. $\mathcal{I}'_n$) for which the expected running time is $\Omega(n \log n)$.*

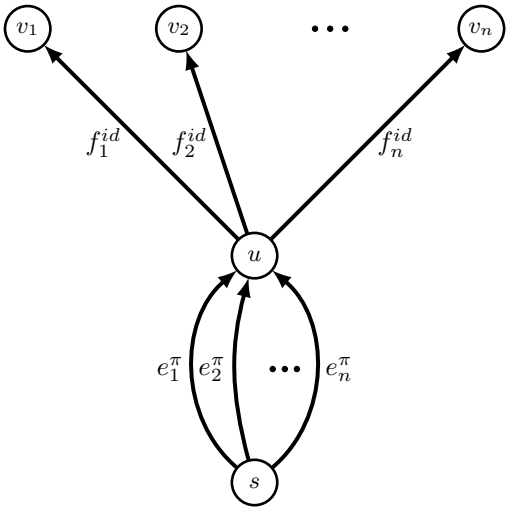

Figure 2.23: A generic representation of a temporal graph in $\mathcal{I}_n$.

Recall that unrestricted waiting is equivalent to $\alpha_u = 0$ and $\beta_u = +\infty$ for all u. This result also holds when restricting the temporal graph model to integer times and $O(n)$ lifetime.

Proof. The first step of the proof is to build $\mathcal{I}_n$ and $\mathcal{I}'_n$. Let us fix $2n$ integer times $\tau_1, \ldots, \tau_n$ and $t_1, \ldots, t_n$ such that $0 < \tau_1 < t_1 < \tau_2 < t_2 < \cdots < \tau_n < t_n < 3n$. Consider the set of vertices $V = \{s, u, v_1, \ldots, v_n\}$. For any two permutation π and π' of $[n] = \{1, \ldots, n\}$, we define the temporal edges $E_\pi = \{e_i^\pi = (s, u, -i, \tau_{\pi(i)} + i) : i = 1, \ldots, n\}$, and $F_{\pi'} = \{f_j^{\pi'} = (u, v_j, t_{\pi'(j)}, t_n + j - t_{\pi'(j)}) : j = 1, \ldots, n\}$. We can now define the temporal graphs $G_{\pi,\pi'} = (V, E_\pi \cup F_{\pi'}, \alpha, \beta)$, where $\alpha_x = 0$ and $\beta_x = +\infty$ for all $x \in V$.

The family of instances $\mathcal{I}_n$ is given by the temporal graphs $\bigcup_\pi G_{\pi,id}$ given as departure-sorted representations, where s is marked as the source node and id denotes the identity permutation. Notice that the ordering of its temporal edges $E_\pi \cup F_{id}$ by non-decreasing departure time, is $(e_1^\pi, \ldots, e_n^\pi, f_1^{id}, \ldots, f_n^{id})$ for any π. See Figure 2.23 for a representation of a temporal graph in $\mathcal{I}_n$. Similarly, the family of instances $\mathcal{I}'_n$ is given by the temporal graphs $\bigcup_{\pi'} G_{id,\pi'}$ given as arrival-sorted representations. Note also that the ordering of its temporal edges $E_{id} \cup F_{\pi'}$ by non-decreasing arrival time, is $(e_1^{id}, \ldots, e_n^{id}, f_1^{\pi'}, \ldots, f_n^{\pi'})$ for any π'.

Now suppose that we are given a deterministic algorithm A for computing minimum overall-waiting-time walks from s to all nodes. In the temporal graph $G_{\pi,id}$ the possible temporal walks from s to v_j are given by $\langle e_i^\pi, f_j^{id} \rangle$ such that $\tau_{\pi(i)} \leq t_j$, and the overall waiting time of such walk is $t_j - \tau_{\pi(i)}$. The minimum overall waiting time is thus obtained for the largest $\tau_{\pi(i)} \leq t_j$, that is for $\pi(i) = j$. This means that, there is a one to one correspondence between the outputs of A and the permutations of $[n]$. In particular, if algorithm A is correct then there are at least $n!$ possible different outputs. Since A is a deterministic comparison only algorithm and the input instance order $(e_1^\pi, \ldots, e_n^\pi, f_1^{id}, \ldots, f_n^{id})$ does not depend on π, two executions of A with same comparisons lead to the same output. This means that, if we denote with c the maximum number of comparisons made by A, there are at most 2^c different outputs. The correctness of A thus implies $2^c \geq n!$. We then get $c \geq n \ln n - n$

and conclude that the time complexity of A is $\Omega(n \log n)$.

More precisely, consider the decision tree corresponding to each time comparison. We have just argued that this tree has depth $n(\ln n - 1)$ at least. Consider the permutations where the execution terminates after $\frac{n}{2} \ln n$ comparisons only. As the subtree corresponding to such executions has at most $n^{n/2}$ leaves, there are at most $n^{n/2}$ such permutations. On instances built according to other permutations, algorithm A requires at least $\frac{n}{2} \ln n$ comparisons. With uniform distribution over the inputs in $\mathcal{I}_n$, the average complexity of A is thus at least $\frac{n! - n^{n/2}}{n!} \frac{n}{2} \ln n \geq (1 - \exp(n - \frac{n}{2} \ln n)) \frac{n}{2} \ln n = \Omega(n \ln n)$. Yao's principle then implies that for any randomized algorithm solving the single-source minimum-overall-waiting-time walk problem, there exists an instance in $\mathcal{I}_n$ on which its average running time is $\Omega(n \log n)$.

Similarly, in a temporal graph $G_{id,\pi'}$ the possible temporal walks from s to v_j are given by $\langle e_i^{id}, f_j^{\pi'} \rangle$ such that $\tau_i \leq t_{\pi'(j)}$, and the overall waiting time of such walk is $t_{\pi'(j)} - \tau_i$. The minimum overall waiting time is thus obtained for the largest $\tau_i \leq t_{\pi'(j)}$, that is for $i = \pi'(j)$. This, again, means that, there is a one to one correspondence between the outputs of A and permutations of $[n]$ and we can conclude similarly to the previous case. □

2.6 Handling zero travel-times

We now show how our approach can be adapted to handle instances containing zero-cycles. Note that a temporal graph containing a zero-cycle fails to admit any half-extend-respecting ordering of its edges. We will give an algorithm based on Algorithm 2, that solves the single-source all-reachable-edge minimum-cost problem in $O(|E| \log |V|)$ in this setting, where E is the set of temporal edges and V is the set of nodes. It still requires a doubly-sorted representation as input. The key point of this algorithm is to compute an ordering of the edges that can be handled correctly by Algorithm 2 for a given source as long as the temporal graph satisfies the following property generalizing non-negative weights.

Consider a temporal graph $G = (V, E, \alpha, \beta)$ with a cost structure $\mathcal{C} = (C, \gamma, \oplus, \preceq)$. A cost $d \in C$ is said to be *$\mathcal{C}$-non-negative* when it satisfies $c \preceq c \oplus d$ for all $c \in C$. This can be seen as a generalization of classical non-negativity. We consider instances satisfying the following *right-absorption* property (*absorption* for short) which is a restriction to zero travel time edges of a property similarly considered in [60]:

$$\text{for any } e = (u, v, \tau, \lambda) \in E \text{ such that } \lambda = 0 \text{ and } \alpha_u = \alpha_v = 0, \quad \gamma(e) \text{ is } \mathcal{C}\text{-non-negative, that is } c \preceq c \oplus \gamma(e) \text{ for all } c \in C. \tag{absorption}$$

Notice that under absorption and isotonicity, there cannot exist any zero-walk Q in G such that $c \oplus \gamma_Q \prec c$, for any cost $c \in C$. Indeed, this absorption property captures a property similar to non-negativity of weights in classical shortest path computation.

Let us define the following property on a doubly sorted representation of a temporal graph. Let $G = (V, E, \alpha, \beta)$ be a temporal graph, (E^{dep}, E^{arr}) be a doubly-sorted representation of G and $s \in V$ be a source node. We say that (E^{dep}, E^{arr}) is *s-optimal-respecting* if for each s-reachable edge e there exists a (E^{dep}, E^{arr})-respected walk Q from s having last edge e and with cost c, where c is the minimum cost of any walk from s ending with e. We can now state the following result.

Theorem 6. *Given a doubly-sorted representation (E^{arr}, E^{dep}) of a temporal graph $G = (V, E, \alpha, \beta)$ with cost structure $(C, \gamma, \oplus, \preceq)$ satisfying isotonicity and absorption, and a source node s, the single-source all-reachable-edge minimum-cost problem can be solved in $O(|E| \log |V|)$ time and space.*

To prove this, we design an algorithm that reorders edges with zero travel time and same arrival time in both E^{arr} and E^{dep}, producing a different doubly-sorted representation $(\bar{E}^{arr}, \bar{E}^{dep})$ while performing a linear scan like in Algorithm 2. The doubly-sorted representation obtained through the reordering is s-optimal-respecting. The procedure is formalised in Algorithm 5 and 6.

To identify efficiently such sequences of zero travel time edges, we first sort E^{arr} according to non-decreasing arrival times as follows. We obtain through bucket sorting the lists E^{arr}_v of edges with head v ordered by non-decreasing arrival time for all nodes v, and then merge them back using a priority queue in $O(|E| \log |V|)$ time. We obtain the ordered list A^{arr} of all the arrival times of the edges by scanning E^{arr}. We can now refine the ordering of the edges with same arrival time in the following way. By bucket sorting, we separate edges with same arrival time into four blocks and then merge them back together one after the other. The first block are those edges having positive travel time, the second the edges with zero travel time and positive minimum waiting-time at the tail, the third the edges with zero travel time and zero minimum waiting-time both at the source and the head, and finally the fourth are the edges with zero travel time and positive minimum waiting-time at the head. The reason for this separation is that an edge arriving at time a can be extended by a zero-walk at time a only if its head has zero minimum waiting-time, and similarly, a zero-walk at time a can be extend by an edge departing at time a only if its tail has zero minimum waiting-time. More precisely, the notation we use to indicate the four blocks is the following. We denote by $E^{arr}_{a,\lambda>0}$ the sub-array of edges of E^{arr} with arrival time a and with positive travel time. Among the edges with arrival time a and with zero travel time we distinguish the following three blocks. We denote by $E^{arr}_{a,\lambda=0,\alpha_{tail}>0}$ the sub-array of edges $e = (u, v, \tau, 0)$ such that $\alpha_u > 0$, by $E^{arr}_{a,\lambda=0,\alpha=0}$ the sub-array of edges $e = (u, v, \tau, 0)$ that $\alpha_u = \alpha_v = 0$, and finally by $E^{arr}_{a,\lambda=0,\alpha_{head}>0}$ the sub-array of edges $e = (u, v, \tau, 0)$ such that $\alpha_v > 0$. In particular we refer to $E^{arr}_{a,\lambda=0,\alpha=0}$ as a *zero-block*. Before scanning each zero-block, its edges are reordered as described next.

Our goal is to identify certain minimum-cost walks that contain edges in the zero-block, and preserve the ordering of their edges. We represent the edges of the zero-block through a weighted static graph; indeed the time labels and the waiting constraints in this case play a marginal role, since all the edges in the zero-block have same departure time, zero travel time and no minimum waiting constraints on their head and tail. In particular, walks in the digraph correspond to walks in the temporal graph. We first identify edges that terminate minimum-cost walks containing exactly one edge in the zero-block. The heads of such edges will serve as sources in the static graph, and are associated to the cost of the aforementioned walks. From these sources, with their initial associated starting costs, we run Dijkstra algorithm [28] and build a shortest path forest from them. We then reorder the edges of the zero-block in the following way: first the edges terminating minimum cost walks to the sources, then edges corresponding to arcs in the shortest path forest so as to preserve path order for all paths in the forest, and finally the remaining edges of the zero-block. Moreover, we will also partially reorder the edges in E^{dep}: among the edges with same departure time we will put first those edges that we identified preceding the sources.

The algorithm then scans the reordered zero-block and the following edges again as in Algorithm 2 up to the next zero-block. We will prove that the two reordered lists obtained

Input: A doubly-sorted representation (E^{dep}, E^{arr}) of a temporal graph G with waiting-time constraints (α, β) and cost structure $(C, \gamma, \oplus, \preceq)$ satisfying isotonicity and absorption, and a source node s.

Output: Minimum cost of an sv-walk for each node v and for each s-reachable edge e with head v.

1 Sort E^{arr} by non-decreasing arrival time.
2 Scan E^{arr} to compute A^{arr} and blocks $E^{arr}_{a,\lambda>0}$, $E^{arr}_{a,\lambda=0,\alpha_{tail}>0}$, $E^{arr}_{a,\lambda=0,\alpha=0}$, $E^{arr}_{a,\lambda=0,\alpha_{head}>0}$.
3 Rebuild E^{arr} by concatenating these blocks for increasing $a \in A^{arr}$.
4 Initialize variables as in Algorithm 2 (Lines 1 - 6).
5 **For** *each arrival time* a *in* A^{arr} **do**
6 **For** $e \in E^{arr}_{a,\lambda>0}$ **do** scan e as in Algorithm 2 (line 7 - 22).
7 **For** $e \in E^{arr}_{a,\lambda=0,\alpha_{tail}>0}$ **do** scan e as in Algorithm 2 (line 7 - 22).
8 $E^{ord} := Reorder(E^{arr}_{a,\lambda=0,\alpha=0}, a)$.
9 Replace $E^{arr}_{a,\lambda=0,\alpha=0}$ by E^{ord} in E^{arr}.
10 **For** $e \in E^{ord}$ **do** scan e as in Algorithm 2 (line 7 - 22).
11 **For** $e \in E^{arr}_{a,\lambda=0,\alpha_{head}>0}$ **do** scan e as in Algorithm 2 (line 7 - 22).
12 **Return** *the sets* $(A'_v)_{v \in V}$

Algorithm 5: Adaption of Algorithm 2 to handle edges with zero travel time.

are indeed a doubly-sorted representation which is s-optimal-respecting.

We now describe more precisely the call to Reorder$(E^{arr}_{\tau,\lambda=0,\alpha=0}, \tau)$ which is formally described in Algorithm 6. We first identify the set $V' = V'_{tail} \cup V'_{head}$ of nodes appearing as tail or head of edges in the zero-block $E^{arr}_{\tau,\lambda=0,\alpha=0}$ at Line 2. We notice that edges in the zero-block sharing the same tail have the same associated cost according to Algorithm 2: as they have same departure time τ they extend the same set of walks and they belong to the same interval. This allows us to assign to each node u in V'_{tail} the cost $B_{tail}[u]$ associated to edges of the zero-block with tail u at the moment of the call to Reorder(), namely just before scanning the edges of the zero-block (see Lines 3-6). This is the minimum cost of any su-walk composed of edges scanned so far, excluding in particular walks containing edges in $E^{arr}_{\tau,\lambda=0,\alpha=0}$, that can be extended with edges departing at time τ. We also associate to each u in V'_{tail} the index of the first edge in E^{dep}_u that has departure time τ at Lines 5-6. Notice that this is not necessarily an edge in the zero-block.

Next, we compute for each edge e in the block the minimum cost to reach its head considering both the case when e extends a walk from the source, and the case when starts itself a walk from the source at Lines 7-16. In particular, by keeping the minimum among the edges with same head, we compute for each node v in V'_{head} the minimum cost $B'[v]$ of sv-walks that contain one, and only one, edge of the zero-block and terminate with it. We store in $P'[v]$ the last edge of such a walk with minimum cost at Line 16. All nodes $v \in V'_{head}$ that can be reached by such an sv-walk are stored in a set of sources S at Line 11.

We then build a weighted directed graph D with node set $V'_{tail} \cup V'_{head}$ and arc set A' which is defined by associating an arc $(u, v, \gamma(e))$ to each edge $e = (u, v, \tau, 0) \in E^{arr}_{\tau,\lambda=0,\alpha=0}$ at Lines 17-18.

Finally, we reorder the edges in $E^{arr}_{\tau,\lambda=0,\alpha=0}$ and E^{dep} at Lines 19-26 based on a minimum-cost forest F from S in D which is computed at Line 19 through an algebraic version of

Function Reorder($E^{arr}_{\tau,\lambda=0,\alpha=0}, \tau$)

`/* Identify nodes appearing in edges of` $E^{arr}_{\tau,\lambda=0,\alpha=0}$`: */`

Let $V'_{tail} := \{u : \exists(u, v, \tau, 0) \in E^{arr}_{\tau,\lambda=0,\alpha=0}\}$ and $V'_{head} := \{v : \exists(u, v, \tau, 0) \in E^{arr}_{\tau,\lambda=0,\alpha=0}\}$.

`/* Set` $B_{tail}[u]$ `to the best extendable cost of edges from` u `in the zero-block. */`

For *each node* $u \in V'_{tail}$ **do**

  Let $I = (l, r, c, e)$ be the first interval in $\mathcal{I}_u$ containing an edge e with $dep(e) \geq \tau$.

  Let i be the index of the first edge e in I such that $dep(e) = \tau$.

  Set $p[u] := i$ and $B_{tail}[u] := c$.

`/* Set` $B'[v]$ `to the minimum cost of an` sv`-walk ending with exactly one edge in the zero-block. */`

For *each node* $v \in V'_{head}$ **do** Initialize $B'[v] = \perp$.

Initialize a set $S := \emptyset$ of reachable nodes.

For *each edge* $e = (u, v, \tau, 0) \in E^{arr}_{\tau,\lambda=0,\alpha=0}$ **do**

  If $u = s$ *or* $B_{tail}[u] \neq \perp$ **then**

    $S := S \cup \{v\}$ `/* v can be reached from s. */`

    If $u = s$ *and* $(B_{tail}[u] = \perp$ *or* $\gamma(e) \prec B_{tail}[u] \oplus \gamma(e))$ **then** $c := \gamma(e)$

    else $c := B_{tail}[u] \oplus \gamma(e)$

    If $B'[v] = \perp$ *or* $c \prec B'[v]$ **then**

      $B'[v] := c$

      $P'[v] := e$ `/* Last edge of a walk defining` $B'[v]$`. */`

`/* Define a weighted static digraph` $D = (V'_{tail} \cup V'_{head}, A')$`. */`

$A' := \{(u, v, \gamma(e)) : e = (u, v, \tau, 0) \in E^{arr}_{\tau,\lambda=0,\alpha=0}\}$

Construct a weighted digraph D with vertex set $V'_{tail} \cup V'_{head}$ and arc set A'.

`/* Reorder the sets of edges` $E^{arr}_{\tau,\lambda=0,\alpha=0}$ `and` E^{dep}`. */`

Compute a minimum-cost forest $F :=$ DijkstraFromSet(D, S, B').

$E^{ord} := \emptyset$

For $v \in S$ *such that* $F[v] = \perp$ **do**

  Let $e = (u, v, \tau, 0) := P'[v]$ and append e to E^{ord}.

  Swap in E^{dep}_u edge e with the edge that has index $p[u]$ and set $p[u] := p[u] + 1$.

  Compute a BFS ordering F_v of the arcs of the tree rooted at v in F.

  For each arc $(u, v, c) \in F_v$, append the associated edge $(u, v, \tau, 0)$ to E^{ord}.

Append to E^{ord} the remaining edges in $E^{arr}_{\tau,\lambda=0,\alpha=0}$.

Return E^{ord}.

Algorithm 6: Reorder the temporal edges in a zero-block so that a sufficiently big set of minimum-cost walk are (E^{dep}, E^{arr})-respected.

```
1  Function DijkstraFromSet((V', A'), S, B')
2      For v ∈ V' do initialize a key K[v] :=⊥ and a parent pointer F[v] :=⊥.
3      Initialize a Fibonacci heap H := ∅.
4      For v ∈ S do add v to H with key K[v] := B'[v].
5      While H ≠ ∅ do
6          u := PopMin(H)
7          For (u, v, c) ∈ A' do
8              If K[v] =⊥ or K[u] ⊕ c ≺ K[v] then
9                  K[v] := K[u] ⊕ c
10                 F[v] := (u, v, c)
11                 If v ∉ H then add v to H with key K[v]
12                 else decrease key of v in H to K[v].
13     return F.
```

Algorithm 7: Algebraic version of Dijkstra from a set S of sources with initial costs B' in a digraph (V', A').

Dijkstra algorithm which is described in detail in the next paragraph. Note that this Dijkstra computation uses the fact that all arcs of D have $\mathcal{C}$-non-negative costs. The forest F is given by parent pointers where $F[v]$ provides for each node v an arc allowing to reach v from S through a path with minimum cost in D. The pointer $F[v]$ has value $\perp$ if v is the root of a tree or if it is not reachable from S in D. For each tree T rooted at a node v in F, we use a BFS ordering of T to make sure at Lines 24-25 that any path in T corresponds to a walk whose edges appear in order in E^{ord}, and after $P'[v]$. Note that such a walk extends edge $P'[v]$ which is added first. Note also that the edge $P'[v]$ is added only once to E^{ord}: since it has head v which is a root, it cannot be associated to an arc of F (we have $F[v] = \perp$). Additionally, we move edges $P'[v]$ with tail u in E^{dep}_u at Line 23 so that they appear in E^{dep}_u before other edges of the zero-block. This guarantees that any walk resulting from concatenating an edge $P'[v]$ and a walk corresponding to a path in the tree rooted at v will be $(\bar{E}^{arr}, \bar{E}^{dep})$-respected.

For the sake of completeness, we also include an algebraic version of Dijkstra algorithm as detailed in Algorithm 7. It is similar to the version of [60] with slightly different hypothesis and the mild generalization of computing minimum-cost paths in a weighted digraph (V', A') from a set S of sources where each source $v \in S$ is associated to an initial cost $B'[v]$. More precisely, any walk $W = \langle (u_1, v_1, c_1), \ldots, (u_k, v_k, c_k) \rangle$ of $k \geq 1$ arcs from a source $u_1 \in S$ in the digraph is associated to a cost $\gamma^D(B', W) = (\cdots (B'[u_1] \oplus c_1) \cdots \oplus c_{k-1}) \oplus c_k$. A node v is said to be *reachable* from S in (V', A') if there exists a walk from a node $u \in S$ to v. We consider that all nodes $v \in S$ are reachable through an empty path with cost $B'[v]$. A *minimum-cost walk* from S to $v \in V'$ is defined as a walk W from any node $u \in S$ to v such that $\gamma^D(B', W) \preceq \gamma^D(B', W')$ for any walk W' from $u' \in S$ to v. The following elementary lemma allows us to focus on paths rather than walks.

Lemma 3. *Let W be a walk in D from $u \in S$ to $v \in V'$, then there exists a path P in D from u to v such that $\gamma^D(B', P) \preceq \gamma^D(B', W)$.*

Proof. We show that we can iteratively remove any cycle from W without increasing the cost of the walk. Let us decompose W into $W_1.W_c.W_2$, where W_c is a cycle. Then we

have $\gamma^D(B', W_1) \preceq \gamma^D(B', W_1.W_c)$ by absorption. We then obtain $\gamma^D(B', W_1.W_2) \preceq \gamma^D(B', W_1.W_c.W_2) = \gamma^D(B', W)$ by isotonicity. □

We define a *minimum-cost forest* F from S as a union of minimum-cost paths from S to all reachable nodes from S, that forms a forest, that is where each node v has at most one entering arc $F[v]$. Such a forest can be computed through Dijkstra algorithm. Recall that it consists in visiting nodes according to a non-decreasing cost order. More precisely, each node v is associated to a key $K[v]$ storing the minimum-cost of a path reaching v from S that has been identified so far. Initially, only nodes in S have a defined key which is initialized according to B', see Line 4 in Algorithm 7. The next node u to visit, that is a node with minimum key, can be found efficiently through a Fibonacci heap H at Line 6. We can then update the keys of each out-neighbour v of u according to Lines 8-12. The correctness of Algorithm 7 follows from the following lemma.

Lemma 4. *Given a weighted digraph (V', A') and a cost structure $(C, \gamma, \oplus, \preceq)$ satisfying isotonicity, suppose that for every arc $(u, v, c) \in A'$ the cost c is $\mathcal{C}$-non-negative. Given a set $S \subseteq V'$ associated with initial costs $B'[v] \in C$ for $v \in S$, Algorithm 7 returns a minimum-cost forest F from S.*

Before proving this lemma, note that C-non-negativity is not required for initial costs $B'[v]$ of nodes $v \in S$. Despite its algebraic abstraction, the proof is nevertheless similar to the one found in algorithm textbooks [20] for the classical version.

Proof. As usual with Dijkstra algorithm, we can prove by induction that the nodes are popped from the heap H at Line 6 by non-decreasing order of keys. The reason is that all nodes v remaining in H when we pop u have a key $K[v]$ satisfying $K[u] \preceq K[v]$ by the correctness of the heap operations which rely on the fact that $\preceq$ is a total order. Second, each node v added to the heap at Line 11, or whose key is decreased at Line 12, has key $K[v] = K[u] \oplus c$ and we have $K[u] \preceq K[u] \oplus c = K[v]$ as c is $\mathcal{C}$-non-negative. This non-decreasing order of popped keys together with the C-non-negativity of arc costs also imply that once a node has been popped, it is never re-inserted in the heap later. As the parent pointer $F[v]$ of a node always correspond to an arc (u, v, c) such that u has been popped before v, F cannot induce any cycle and is indeed a forest. Moreover, following recursively the pointer $F[u]$ as long as $F[u] \neq \perp$, we obtain a path P_F^v with nodes ordered according to popping order. Note that the first node of P_F^v must have been inserted in H initially and is thus in S. The cost $\gamma^D(B', P_F^v)$ of P_F^v is thus defined and it equals the value of $K[v]$ when v is popped (this directly results from the mutual updates of $K[v]$ and $F[v]$).

Suppose for the sake of contradiction that there exist nodes $v \in V'$ which are reachable from S and such that F does not contain a minimum-cost path from S to v. Without loss of generality, we can choose such a node v so that no other such node is popped from H before v. Consider a minimum cost path P from S to v. Either P is non-empty and we let $u \in S$ denote the tail of its first arc, or we have $v \in S$ and no path from S to v has cost less than $B'[v]$, in which case we set $u := v$. As $u \in S$ implies that u is initially added to H, it must be popped at some point.

First assume that P is empty. We then have $v = u \in S$, and v has been added to H, implying that v is popped at some point. The path P_F^v from S to v in F has cost $\gamma^D(B', P_F^v) = K[v]$. As the key of $u = v$ can only decrease, we get $K[v] \preceq B'[v]$ and thus $\gamma^D(B', P_F^v) = B'[v]$ as no path from S to v has cost less than $B'[v]$ in that case. This is in contradiction with our hypothesis on v.

From now on, we assume that P is non-empty. Suppose additionally that v is popped before u. This implies that the path P_F^v from S to v in F has cost $\gamma^D(B', P_F^v) = K[v] \preceq K[u]$. As the key of u can only decrease, we have $K[u] \preceq B'[u]$. Since the arcs of P have $\mathcal{C}$-non-negative costs, we have $B'[u] \preceq \gamma^D(B', P)$ and we get $\gamma^D(B', P_F^v) \preceq \gamma^D(B', P)$, contradicting again the choice of v.

Otherwise, we can consider the last node u' in P such that all nodes from u to u' in P have been popped before v. Let (u', v', c) be the arc following u' in P. The update of $K[v']$ according to arc (u', v', c) at Lines 8-12 implies $K[v'] \preceq K[u'] \oplus c$ and $v' \in H$. In particular, v' will be popped at some point. Our choice of v implies $\gamma^D(B', P_F^{u'}) \preceq \gamma^D(B', P[u : u'])$ where $P[u : u']$ denotes the subpath of P from u to u'. As discussed previously, we have $K[u'] = \gamma^D(B', P_F^{u'})$ when u' is popped, and we thus get $K[v'] \preceq \gamma^D(B', P[u : u']) \oplus c = \gamma^D(B', P[u : v'])$ by isotonicity. In the case $v' = v$, we thus get $\gamma^D(B', P_F^v) \preceq \gamma^D(B', P)$. In the case $v' \neq v$, v is popped before v' according to the choice of u', implying $\gamma^D(B', P_F^v) = K[v] \preceq K[v'] \preceq \gamma^D(B', P[u : v']) \preceq \gamma^D(B', P)$ as the arcs of $P[v' : v]$ have $\mathcal{C}$-non-negative costs. In all cases, we get $\gamma^D(B', P_F^v) \preceq \gamma^D(B', P)$, in contradiction with our hypothesis on v. □

We now prove Theorem 6.

Proof of Theorem 6. Our proof mainly relies on the correctness and the complexity of Algorithm 5 thanks to the following observation.

Claim 1. *The execution of Algorithm 5 corresponds to an execution of Algorithm 2 with input $(\bar{E}^{dep}, \bar{E}^{arr})$ where $\bar{E}^{dep}$ and $\bar{E}^{arr}$ are the orderings resulting from the calls to Algorithm 6.*

Recall that, after line 3, the list E^{arr} is a sequence of blocks $E^{arr}_{a,\lambda>0}$, $E^{arr}_{a,\lambda=0,\alpha_{tail}>0}$, $E^{arr}_{a,\lambda=0,\alpha=0}$, $E^{arr}_{a,\lambda=0,\alpha_{head}>0}$ by increasing value of a, and the ordering $\bar{E}^{arr}$ is obtained from it by replacing each zero-block $E^{arr}_{a,\lambda=0,\alpha=0}$ with the local order E^{ord} computed through the call to Reorder($E^{arr}_{a,\lambda=0,\alpha=0}$). The ordering $\bar{E}^{dep}$ is obtained as a concatenation of the lists $\bar{E}^{dep}_u$, where $\bar{E}^{dep}_u$ is the list obtained from E^{dep}_u through the swaps made at Line 23 by Algorithm 6.

We prove Claim 1 through the following observations. The zero-blocks of edges $E^{arr}_{\tau,\lambda=0,\alpha=0}$ are reordered before any of its edges has been scanned, and edges are indeed scanned according to the ordering of $\bar{E}^{arr}$. Concerning $\bar{E}^{dep}$, we first note that none of the edges in a zero-block $E^{arr}_{\tau,\lambda=0,\alpha=0}$ has been finalized when Reorder($E^{arr}_{\tau,\lambda=0,\alpha=0}$) is called for the following two reasons. First, all edges with head u scanned so far have arrival time at most τ, and since $\alpha_u = 0$ edges with departure time greater or equal to τ in E^{dep}_u were not considered by any call to FinalizeCosts() at Line 17 of Algorithm 2. Second, $E^{arr}_{a,\lambda=0,\alpha_{tail}>0}$ does not contain any edge with tail u since $\alpha_u = 0$, and all edges from u that have been scanned so far have departure time less than τ. They thus appear before edges from u in $E^{arr}_{\tau,\lambda=0,\alpha=0}$ since E^{dep}_u is sorted by non-decreasing departure time, and these edges have not been finalized by any call at Line 9 of Algorithm 2 either. Finally, the swaps in E^{dep}_u concern edges with same departure time τ and same tail u. This means that they can extend exactly the same set of walks and thus belong to the same interval in $\mathcal{I}_u$. We thus have exactly the same intervals in $\mathcal{I}_u$ for each node u as if the algorithm had been run with input $(\bar{E}^{dep}, \bar{E}^{arr})$ from the beginning. As the subsequent processing occurs with edges ordered according to $(\bar{E}^{dep}, \bar{E}^{arr})$ until the next-zero block, this concludes the proof of Claim 1

Correctness.

The core of the proof of correctness consists in showing that the reordered lists $(\bar{E}^{dep}, \bar{E}^{arr})$ are an s-optimal-respecting doubly-sorted representation as we can then conclude by Proposition 1.

First note, that the lists $\bar{E}^{dep}$ and $\bar{E}^{arr}$ are still node departure and node arrival sorted respectively. The list $\bar{E}^{dep}$ is node departure sorted, as it is a reordering of E^{dep} obtained by swapping edges with same tail and departure time. On the other side, the reordering of E^{arr} concerns only sets of edges within the same zero-block which all have same arrival time, thus $\bar{E}^{arr}$ is still ordered by non-decreasing arrival time of the edges.

The rest of the correctness part is dedicated to proving that $(\bar{E}^{dep}, \bar{E}^{arr})$ is s-optimal-respecting. Suppose for the sake of contradiction that there exists an s-reachable edge e such that there exists no minimum cost walk from s ending with e that is $(\bar{E}^{dep}, \bar{E}^{arr})$-respected. Without loss of generality, let e be the first edge in $\bar{E}^{arr}$ such that this happens, and let $Q = \langle e_1, \ldots, e_k \rangle$ be a walk with minimum cost among the walks from s ending with $e = e_k = (u_k, v_k, \tau_k, \lambda_k)$. As our assumption implies that Q itself is not $(\bar{E}^{dep}, \bar{E}^{arr})$-respected, it must have at least two edges, we thus assume $k \geq 2$.

Note. In the following, whenever we have to verify if $e <_{\bar{E}^{arr}} f$, for some edges e and f, it is sufficient to show that the block containing e precedes the block that contains f according to the ordering of E^{arr} computed at Line 3. This comes from the fact that $\bar{E}^{arr}$ follows the same sequence of blocks and differs only by swaps within the zero-blocks.

Case A: edge e_{k-1} does not belong to a zero-block or $arr(e_k - 1) < dep(e_k)$. Let us first consider the case in which e_{k-1} does not belong to a zero-block. This implies $e_{k-1} <_{\bar{E}^{arr}} e_k$. The reason is that we have $arr(e_{k-1}) \leq dep(e_k)$ since e_k extends e_{k-1} and $\bar{E}^{arr}$ is ordered by non-decreasing arrival time. Hence, $e_k <_{\bar{E}^{arr}} e_{k-1}$ would imply $arr(e_k) \leq arr(e_{k-1})$ and thus $arr(e_{k-1}) = dep(e_k) = arr(e_k)$. As e_k extends e_{k-1}, we then must have $\alpha_{u_k} = 0$. If e_{k-1} does not belong to a zero-block we then have either $\lambda_{k-1} > 0$ or $\alpha_{u_{k-1}} > 0$, and thus e_{k-1} belongs to a block before the block of e_k according to the ordering of E^{arr} computed at Line 3. As $e_{k-1} <_{\bar{E}^{arr}} e_k$, our choice of $e = e_k$ implies that there exists a walk Q' from s ending with e_{k-1} that is $(\bar{E}^{dep}, \bar{E}^{arr})$-respected and has minimum cost among the walks from s ending with e_{k-1}. Because of isotonicity we obtain that $\gamma(Q'.e_k) \preceq \gamma(Q)$, and proving that $Q'.e_k$ is $(\bar{E}^{dep}, \bar{E}^{arr})$-respected would raise a contradiction. Since Q' is $(\bar{E}^{dep}, \bar{E}^{arr})$-respected, we just need to check that for each edge e' with tail u_k such that $e_k \leq_{\bar{E}^{dep}} e'$, then $e_{k-1} <_{\bar{E}^{arr}} e'$. Due to the fact that e_k extends e_{k-1} and $e_k \leq_{\bar{E}^{dep}} e'$, we have $arr(e_{k-1}) \leq dep(e_k) \leq dep(e') \leq arr(e')$. If $arr(e_{k-1}) < arr(e')$ we can conclude that $e_{k-1} <_{\bar{E}^{arr}} e'$, since $\bar{E}^{arr}$ is non-decreasing arrival time sorted. Otherwise, $arr(e_{k-1}) = arr(e')$ implies that the travel time of e' is zero and that $\alpha_{v_{k-1}} = 0$ as wet get $arr(e_{k-1}) = dep(e_k)$ and e_k extends e_{k-1}. If e_{k-1} has positive travel time, we can again conclude, as e' has zero travel time and the two edges have same arrival time, the block of e_{k-1} precedes the block of e'. Finally, suppose that e_{k-1} has also zero travel time. As $\alpha_{v_{k-1}} = 0$ and e_{k-1} does not belong to a zero-block, we must have $\alpha_{u_{k-1}} > 0$. On the other hand, v_{k-1} is the tail of e' and we have $\alpha_{v_{k-1}} = 0$. Also in this case the block of e_{k-1} precedes the block of e'.

We consider now the case in which $arr(e_{k-1}) < dep(e_k)$. This implies $arr(e_{k-1}) < arr(e_k)$, and thus $e_{k-1} <_{\bar{E}^{arr}} e_k$. We can choose Q' as above: a walk from s ending with e_{k-1} that is $(\bar{E}^{dep}, \bar{E}^{arr})$-respected and has minimum cost among the walks from s ending

with e_{k-1}. If we prove that $Q'.e_k$ is $(\bar{E}^{dep}, \bar{E}^{arr})$-respected we can conclude by isotonicity. In particular, we just need to check that for each edge e' with tail u_k such that $e_k \leq_{\bar{E}^{dep}} e'$, then $e_{k-1} <_{\bar{E}^{arr}} e'$. Due to the fact that $arr(e_{k-1}) < dep(e_k)$ and $e_k \leq_{\bar{E}^{dep}} e'$, we have $arr(e_{k-1}) < dep(e_k) \leq dep(e') \leq arr(e')$. As $\bar{E}^{arr}$ is ordered by non-decreasing arrival time we obtain $e_{k-1} <_{\bar{E}^{arr}} e'$.

Case B: edge e_{k-1} belongs to a zero-block. We can now focus on the case where e_{k-1} belongs to a zero-block, namely $e_{k-1} = (u_{k-1}, v_{k-1}, \tau, 0) \in E^{arr}_{\tau,\lambda=0,\alpha=0}$ and $\alpha_{u_{k-1}} = \alpha_{v_{k-1}} = 0$. Consider the call to Reorder($E^{arr}_{\tau,\lambda=0,\alpha=0}$) from Algorithm 5. Let $D = (V'_{tail} \cup V'_{head}, A', A')$ be the digraph constructed at Lines 17-18 in Algorithm 6, and let $S \subseteq V'_{head}, A'$ be the set of source nodes computed according to Line 11. The proof of correctness follows from the last of the three following claims.

Claim 2. *Any path P in D from a node in S corresponds to a walk Q_P in G with cost $\gamma^D(B', P)$ arriving at time τ.*

The reason is twofold. First, each node $v \in S$ is associated to the cost $B'[v]$ of an sv-walk Q_v which ends with edge $P'[v]$ as set in Lines 10-16 in Algorithm 6. More precisely, suppose $P'[v] = e = (u, v, \tau, 0) \in E^{arr}_{\tau,\lambda=0,\alpha=0}$. In the case where $u = s$ and the cost $c = B'[v]$ is indeed $\gamma(e)$, we define $Q_v = \langle e \rangle$. Otherwise, e must have an associated cost $c' = B_{tail}[u]$ computed using Algorithm 2 and we have $c = B_{tail}[u] \oplus \gamma(e)$. The correctness of Algorithm 2 implies that c' is the cost of an su-walk Q_e that e can extend. We then define $Q_v = Q_e.e$ whose cost is precisely $\gamma_{Q_e.e} = B_{tail}[u] \oplus \gamma(e) = c$.

Second, as edge $P'[v]$ has arrival time τ, Q_v also has arrival time τ and any edge $(v, w, \tau, 0)$ can extend it as long as $\alpha_v = 0$. More generally each arc (x, y, c) in a path P from v in D corresponds to an edge $f = (x, y, \tau, 0)$ with cost $\gamma(f) = c$ and such that $\alpha_x = \alpha_y = 0$ according to the construction of A' at Line 17. The condition $\alpha_x = 0$ implies that f extends the edge associated to the arc preceding (x, y, c) in P or extends $P'[v] = e$ if it is the first arc. The path P is thus associated to a walk Q_P of edges in $E^{arr}_{\tau,\lambda=0,\alpha=0}$ such that $Q_v.Q_P$ is a walk. Moreover, the cost of P in D is obtained as $\gamma^D(B', P) = (\cdots (B'[v] \oplus c_1) \cdots \oplus c_{k-1}) \oplus c_k$ where $c_1, \ldots, c_k$ denote the respective costs of arcs in P. As $B'[v] = \gamma_{Q_e}$, we get $\gamma^D(B', P) = \gamma_{Q_v.Q_P}$.

Claim 3. *There exists a path P_Q from S to $v_{k-1} = u_k$ in D that has cost $\gamma^D(B', P_Q) \preceq \gamma_{\langle e_1,\ldots,e_{k-1}\rangle}$.*

To prove this, we decompose $\langle e_1, \ldots, e_{k-1} \rangle = Q^1.Q^2$ where $Q^2 = \langle e_i, \ldots, e_{k-1} \rangle$ is its longest suffix of edges in the zero-block $E^{arr}_{\tau,\lambda=0,\alpha=0}$ containing e_{k-1}. Note that all edges of $\langle e_1, \ldots, e_{k-1} \rangle$ belonging to the zero-block must be consecutive as edges are sorted according to non-decreasing arrival time in a walk and no edge of the zero-block can be extend by and edge having positive travel time or positive minimum waiting-time.

First, consider the moment when $e_i = (u_i, v_i, \tau, 0)$ is considered at Line 10 in Algorithm 6 for possibly updating $B'[v_i]$. If its associated cost is not $\perp$, we have $B_{tail}[u_i] \preceq \gamma_{Q^1}$. The reason is that $B_{tail}[u_i]$ is the cost of a walk R that e_i extends, which is composed of edges scanned so far, and such that $R.e_i$ is $(\bar{E}^{dep}, \bar{E}^{arr})$-respected. Moreover, it has minimum cost among such walks according to Lemma 1. We thus have $B_{tail}[u_i] \preceq \gamma_{Q^1}$ since Q^1 is $(\bar{E}^{dep}, \bar{E}^{arr})$-respected by our choice of e_k. Otherwise, Q^1 is empty and we must have $u_i = s$ and $i = 1$. In this latter case, we let $Q^1.e_i$ denote the walk $\langle e_i \rangle$. In both cases, c is set at Lines 12-13 to a value such that $c \preceq \gamma_{Q^1.e_i}$. The update of $B'[v_i]$ according to

Lines 14-15 then ensures $B'[v_i] \preceq \gamma_{Q^1.e_i}$. Second, each edge $e_j = (u_j, v_j, \tau, 0)$ for $j > i$, is associated to an arc $e'_j = (u_j, v_j, \gamma(e_j))$ in D according to the construction of A' at Line 17. Let W_Q denote the walk $\langle e'_{i+1}, \ldots, e'_{k-1}\rangle$ in D which has cost $\gamma^D(B', W_Q) = (\cdots(B'[v_i] \oplus \gamma(e_{i+1}))\cdots)\oplus\gamma(e_{k-1})$. As $B'[v_i] \preceq \gamma_{Q^1.e_i}$, we get $\gamma^D(B', W_Q) \preceq \gamma_Q$ according to isotonicity. According to Lemma 3, there exists a path P_Q in D from v_i to v_{k-1} satisfying $\gamma^D(B', P_Q) \preceq \gamma^D(B', W_Q) \preceq \gamma_Q$. P_Q is thus a path from $v_i \in S$ to v_{k-1} with cost $\gamma^D(B', P_Q) \preceq \gamma_{\langle e_1,\ldots,e_{k-1}\rangle}$ as claimed.

Claim 4. *There exists a walk $\tilde{Q}$ such that $\tilde{Q}.e_k$ is a $(\bar{E}^{dep}, \bar{E}^{arr})$-respected minimum-cost walk among the walks from s that end with e_k.*

This claim will clearly conclude the proof of correctness. We first note that u_k is reachable from S according to Claim 3 through a path P_Q with cost $\gamma^D(B', P_Q) \preceq \gamma_{\langle e_1,\ldots,e_{k-1}\rangle}$. Lemma 4 then ensures that F contains a minimum-cost path P in D. Its cost must thus satisfy $\gamma^D(B', P) \preceq \gamma^D(B', P_Q)$. Isotonicity then implies $\gamma^D(B', P)\oplus\gamma(e_k) \preceq \gamma^D(B', P_Q)\oplus \gamma(e_k) \preceq \gamma_{\langle e_1,\ldots,e_{k-1}\rangle} \oplus \gamma(e_k) = \gamma_Q$.

Let us denote with $\tilde{Q} = \langle \tilde{e}_1, \ldots, \tilde{e}_l\rangle$ the walk corresponding to P according to Claim 2, and $\tilde{e}_j = (\tilde{u}_j, \tilde{v}_j, \tilde{\tau}_j, \tilde{\lambda}_j)$ for $j = 1, \ldots, l$. According to the construction of $\tilde{Q}$ in Claim 2, let h be the index of the edge $P'[v]$, that is h is the (only) index satisfying $\tilde{e}_h = P'[\tilde{v}_h]$. Note that the subsequent edges $\tilde{e}_{h+1}, \ldots, \tilde{e}_l$ correspond to the arcs of P. On the other hand, edges $\tilde{e}_1, \ldots, \tilde{e}_{h-1}$ precede the zero-block $E^{arr}_{\tau,\lambda=0,\alpha=0}$ in E^{arr}. Note that they thus also precede e_k in $\bar{E}^{arr}$ as e_k extends an edge of the zero-block and thus satisfy $arr(e_k) \geq \tau$ and $\alpha_{u_k} = 0$.

We can assume e_k is not a $P'[v]$ edge for some v. If this was the case, as v is then the head of e_k, this is equivalent to $e_k = P'[v_k]$. Then the walk $Q'' = \langle \tilde{e}_1, \ldots, \tilde{e}_h\rangle$ ends with edge $e_k = \tilde{e}_h$ and has cost $\gamma_{Q''} \preceq \gamma_{\tilde{Q}}$ by C-non-negativity of the edges $\tilde{e}_{h+1}, \ldots, \tilde{e}_l$. As Claim 2 guarantees $\gamma_{\tilde{Q}} = \gamma^D(B', P)$, we then have $\gamma_{Q''} \preceq \gamma^D(B', P) \oplus \gamma(e_k)$ by C-non-negativity of e_k, implying $\gamma_{Q''} \preceq \gamma_Q$. As $\tilde{e}_{h-1}$ is not in the zero-block at time τ, then either $\tilde{e}_{h-1}$ is not in a zero-block or $arr(\tilde{e}_{h-1}) < dep(e_k)$, and we can conclude as case A.

We now prove that $\tilde{Q}.e_k$ is a walk: first e_k is not an edge of $\tilde{Q}$ and second, e_k extends $\tilde{Q}$. The only case where e_k could be in $\tilde{Q}$ is when the edge e_k itself belongs to the zero-block $E^{arr}_{\tau,\lambda=0,\alpha=0}$. Let us rule out this eventuality. As P is a path and not a walk, $\tilde{v}_l = u_k$ cannot be the tail of $\tilde{e}_j$ for any $j \in [h+1, l]$. We also just proved we can assume $e_k \neq \tilde{e}_h$. Second, e_k extends $\tilde{Q}$. The reason is that it extends e_{k-1} which belongs to the zero-block $E^{arr}_{\tau,\lambda=0,\alpha=0}$. We thus have $\tau + \alpha_{u_k} = \tau \leq dep(e_k) \leq \tau + \beta_{u_k}$. As $\tilde{e}_l$ also belongs to the zero-block, it has also arrival time τ and e_k also extends $\tilde{e}_l$ since $\tilde{v}_l = u_k$.

It just remains to prove that $\tilde{Q}.e_k$ is $(\bar{E}^{dep}, \bar{E}^{arr})$-respected. Since $\tilde{e}_{h-1} <_{\bar{E}^{arr}} e_k$, our choice of e_k implies that we can assume without of loss of generality that $\langle \tilde{e}_1, \ldots, \tilde{e}_{h-1}\rangle$ is $(\bar{E}^{dep}, \bar{E}^{arr})$-respected. Thus, we have to consider the $(\bar{E}^{dep}, \bar{E}^{arr})$-respected property with respect to the following three types of pairs of consecutive edges in $\tilde{Q}.e_k$:

1) $\tilde{e}_{h-1}, \tilde{e}_h$,

2) $\tilde{e}_j, \tilde{e}_{j+1}$ for $j = h, \ldots, l-1$,

3) $\tilde{e}_l, e_k$.

In Case 1, we have to prove that for each edge e' such that $tail(e') = tail(\tilde{e}_h) = \tilde{u}_h$ and $\tilde{e}_h \leq_{\bar{E}^{dep}} e'$ we have $\tilde{e}_{h-1} <_{\bar{E}^{arr}} e'$. As $\tilde{e}_h$ extends $\tilde{e}_{h-1}$ and $\tilde{e}_h \leq_{\bar{E}^{dep}} e'$, we have

$arr(\tilde{e}_{h-1}) \leq dep(\tilde{e}_h) \leq dep(e') \leq arr(e')$. If $arr(\tilde{e}_{h-1}) < arr(e')$ we can conclude since $\bar{E}^{arr}$ is sorted by non-decreasing arrival time. Thus let us suppose that $arr(\tilde{e}_{h-1}) = arr(e')$, which implies that the travel time of e' is zero and $arr(\tilde{e}_{h-1}) = dep(\tilde{e}_h) = \tau$. Moreover, we have $\alpha_{tail(\tilde{e}_h)} = \alpha_{tail(e')} = 0$ since $e_h \in E^{arr}_{\tau,\lambda=0,\alpha=0}$. Since $\tilde{e}_{h-1}$ is not in the zero-block $E^{arr}_{\tau,\lambda=0,\alpha=0}$ and $arr(\tilde{e}_{h-1}) = \tau$, we must have $\alpha_{tail(\tilde{e}_{h-1})} > 0$. This implies $\tilde{e}_{h-1} <_{\bar{E}^{arr}} e'$ since in $\bar{E}^{arr}$ the block of edges in $E^{arr}_{a,\lambda=0,\alpha_{tail}>0}$ precedes block $E^{arr}_{a,\lambda=0,\alpha=0}$ and $E^{arr}_{a,\lambda=0,\alpha_{head}>0}$.

In Case 2, we have to prove that for each edge e' such that $tail(e') = tail(\tilde{e}_{j+1}) = \tilde{u}_{j+1}$ and $\tilde{e}_{j+1} \leq_{\bar{E}^{dep}} e'$ we have $\tilde{e}_j <_{\bar{E}^{arr}} e'$. Again, we have $arr(\tilde{e}_j) \leq dep(\tilde{e}_{j+1}) \leq dep(e') \leq arr(e')$ and if $arr(\tilde{e}_j) < arr(e')$ we can conclude. So let us suppose $arr(\tilde{e}_j) = arr(e')$, implying that e' has zero travel time and departure τ. If $\alpha_{head(e')>0}$, we then have $\tilde{e}_j <_{\bar{E}^{arr}} e'$ since $\tilde{e}_j$ belongs to the zero-block and e' is scanned after the zero-block. Otherwise, we have $\alpha_{head(e')=0}$ and e' is in the zero-block $E^{arr}_{\tau,\lambda=0,\alpha=0}$. Having $e' = P'[head(e')]$ would contradict $\tilde{e}_{j+1} \leq_{\bar{E}^{dep}} e'$. The reason is that $\tilde{e}_{j+1}$ is in F while the swaps performed at Line 23 ensures that all edges $P'[v]$ with tail $tail(\tilde{e}_{j+1})$ for some $v \in V'_{head}$ precede other edges with same tail and same departure time in $\bar{E}^{dep}$. We can thus assume $e' \neq P'[head(e')]$. If $e' \in F$, notice that either $\tilde{e}_j = P'[\tilde{v}_j]$ or $\tilde{e}_j \in F$ and precede e' in the BFS ordering, since $tail(e') = head(\tilde{e}_j)$, and thus in both cases we have $\tilde{e}_j <_{\bar{E}^{arr}} e'$. Finally, if e' is not in F, it is added at the end of the reordered zero-block at Line 26 and we again have $\tilde{e}_j <_{\bar{E}^{arr}} e'$.

In Case 3, we have to prove that for each edge e' such that $tail(e') = tail(e_k) = u_k$ and $e_k \leq_{\bar{E}^{dep}} e'$ we have $\tilde{e}_l <_{\bar{E}^{arr}} e'$. Since $e_k \leq_{\bar{E}^{dep}} e'$ and $e_k \neq P'[v_k]$, we deduce that e', as e_k is ordered after edges with tail u_k of the form $P'[v]$ for some $v \in V'_{head}$, which implies $e' \neq P'[head(e')]$. The proof is now similar to the previous case: if $arr(\tilde{e}_l) < arr(e')$, we can directly conclude. Otherwise, e' has zero travel time and arrival time τ. In the three cases $\alpha_{head(e')} > 0$, $e' \in F$ and $e' \notin F$, we conclude similarly. This achieves the proof of correctness.

Complexity analysis.

We finally analyse the complexity of Algorithm 5. As discussed previously, sorting E^{arr} by non-decreasing arrival time can be done in $O(|E| \log |V|)$ time by computing the lists $(E^{arr}_v)_{v \in V}$ and then merging them using a priority queue. Once E^{arr} is sorted by non decreasing arrival time we can partition the list of edges with same arrival time into $E^{arr}_{a,\lambda>0}$, $E^{arr}_{a,\lambda=0,\alpha_{tail}>0}$, $E^{arr}_{a,\lambda=0,\alpha=0}$ and $E^{arr}_{a,\lambda=0,\alpha_{head}>0}$ by bucket sorting into the four lists.

The calls to Reorder($E^{arr}_{\tau,\lambda=0,\alpha=0}, \tau$) incur the only other additional costs compared to Algorithm 2. The nodes in V'_{tail} and V'_{head} are identified in $O(|E^{arr}_{\tau,\lambda=0,\alpha=0}|)$, and both sets cardinality is bounded by $|E^{arr}_{\tau,\lambda=0,\alpha=0}|$. Thus the construction of digraph D is linear in $|E^{arr}_{\tau,\lambda=0,\alpha=0}|$. In order to identify interval I at Line 4 we might scan up to $|\mathcal{I}_u|$ intervals. However, all interval scanned contain edges with departure time less or equal to τ. This means that, at the moment of the next call to Reorder() these intervals will have already been removed by the calls to FinalizeCosts() at Line 9 of Algorithm 2, and thus will not be scanned again. Overall, we can bound the time complexity of this operation by the total number of intervals created during the execution of the algorithm, which is bounded by $|E|$. Similarly, the computation of index i at Line 5 requires to scan all the edges from $E^{dep}_u[l_u]$ until an edge with departure time τ is found. We know that such an edge exists in interval I as some edges in the zero-block have tail u, and again, at the moment of the next call to Reorder() these edges will have already been processed, and thus will not be scanned again. Overall, we can bound the time complexity of this operation by the total number of edges $|E|$. Computing set S and computing for each node v in such set the cost $B'[v]$ and the edge $P'[v]$ is linear in $|E^{arr}_{\tau,\lambda=0,\alpha=0}|$, since it requires a single scan of the edges in the zero-block

and few constant time operations per edge.

The time complexity of Algorithm 6 is thus dominated by the Dijkstra call which costs time $O(|A'| + n' \log n')$ where $n' = |V'_{tail} \cup V'_{head}| \leq |V|$. As $|A'| = |E^{arr}_{\tau,\lambda=0,\alpha=0}|$ and $n' = O(|E^{arr}_{\tau,\lambda=0,\alpha=0}|)$, the overall time complexity of the calls to Algorithm 6 is $O(\sum_{a \in A^{arr}} |E^{arr}_{\tau,\lambda=0,\alpha=0}|(1 + \log |V|)) = O(|E| \log |V|)$. The space complexity is clearly linear.

□

Optimizing a linear combination of classical criteria. A temporal graph with the cost structure defined in Section 2.4 for optimizing a linear combination of classical criteria also satisfies the absorption property under the following assumption: for any edge $e = (u, v, \tau, \lambda)$ such that $\lambda = 0$, $\alpha_u = 0$ and $\alpha_v = 0$, we require $\delta(e) = \delta_5 c(e) + \delta_6 \geq 0$. This implies that for any cost $(\tau, \Delta) \in \mathbb{R} \times \mathbb{R}$, we indeed have $(\tau, \Delta) \preceq (\tau, \Delta) \oplus \gamma(e)$ as $(\tau, \Delta) \oplus \gamma(e) = (\tau, \Delta + \delta(e))$ and $\Delta + \delta(e) \geq \Delta$. If we assume $\delta_5, \delta_6 \geq 0$, non-negativity of costs is required only for edges with zero travel time, and negative values are allowed for $\delta_1, \ldots, \delta_4$ and δ_7. Theorem 6 thus extends the result of [5] to a wider range of linear combinations, and has a slightly better complexity.

2.6.1 Matching conditional lower-bound.

We now state that our algorithm seems optimal according to current knowledge about the complexity of directed single-source shortest paths in the comparison-addition model where lengths of paths are obtained through addition and comparison only [66, 54].

Theorem 7. *Solving the single-source all-reachable-edge minimum-cost problem on a temporal graph with M edges and n nodes requires $\Omega(M \log n)$ time assuming a lower-bound of $\Omega(n \log n)$ for solving single-source shortest paths in directed graphs with n nodes and $\Theta(n)$ arcs.*

The idea is simply to consider k instances of single-source shortest paths with n nodes and $\Theta(n)$ edges. Without loss of generality, all instances have same set of nodes and same source. Each instance is then considered as a temporal sub-graph with zero travel time edges at a certain time slot. This union over time results in a temporal graph with $M = \Theta(kn)$ edges. Preferring walks with latest departure time, and among those with latest departure time, one with shortest length, indeed allows to solve all instances of shortest paths within a single instance of the single-source all-reachable-edge minimum cost problem on this temporal graph.

Proof. Given k instances $(D_1, s_1), \ldots, (D_k, s_2)$ of single-source shortest paths in directed graphs $D_1 = (V_1, A_1), \ldots, D_k = (V_k, A_k)$ with n nodes and $\Theta(n)$ edges each, from sources $s_1 \in V_1, \ldots, s_k \in V_k$ respectively, we construct a temporal graph $G = (V, E, \alpha, \beta)$ with n nodes and $M = \sum_{i=1}^{k} |A_i|$ edges such that a solution to the single-source all-reachable-edge minimum-cost problem provides a solution for each instance as follows. We number the nodes in each directed graph D_i from 1 to n so that s_i is 1. We can then assume $V = [n] = V_1 = \cdots = V_k$ and $s_1 = \cdots = s_k = 1$. For each time $i \in [n]$ and for each arc $(u, v) \in A_i$, we create a temporal edge $e = (u, v, i, 0) \in E$ with associated length $\ell(e)$ the length of (u, v) in D_i. We assume unrestricted waiting: we set $\alpha_u = 0$ and $\beta_u = \infty$ for all $u \in V$.

We now define the cost structure $(C, \gamma, \oplus, \preceq)$ where $C = \mathbb{R} \times \mathbb{R}_{\geq 0}$. Given an edge $e = (u, v, \tau, 0)$, we define its cost as $\gamma(e) = (\tau, \ell(e))$. We define the cost combination

function $\oplus$ by $(\tau, \ell) \oplus (\tau', \ell') = (\tau, \ell + \ell')$ so that the cost of a walk Q is $(dep(Q), \ell(Q))$ where $\ell(Q)$ denotes the sum of the lengths of edges in Q. We finally define the cost total order by $(\tau, \ell) \preceq (\tau', \ell')$ when $\tau > \tau'$ or $\tau = \tau'$ and $\ell \leq \ell'$ so that among several walks, we always prefer one with latest departure, and among several walks with same departure time, we prefer one with shortest length.

Now consider a solution to the single-source all-reachable-edge minimum-cost problem on G with source $s = 1$ and cost structure $(C, \gamma, \oplus, \preceq)$. Let $v \in V_i$ be a node of instance D_i. If no arc enters v in D_i, we know that it is not reachable from s_i and that its distance is ∞. Otherwise, consider an arc $(u, v) \in A_i$. If there exists a path P from s_i to v ending with arc (u, v), it corresponds to a walk Q from $s = s_i$ that ends with $e = (u, v, i, 0)$ and has cost $(i, \ell(P))$. Conversely, any walk with departure time i, cost (i, ℓ), and ending with e must have all its edges departing at time i and corresponds to a path in D_i with length ℓ. To obtain the distance $d_{D_i}(s_i, v)$, we thus consider edges e with departure time i and entering v. If no such edge is associated to a cost in A'_v, no walk departing at i or before can reach v and we have $d_{D_i}(s_i, v) = \infty$. Otherwise, let $c = (\tau, \ell)$ be the minimum cost associated to such an edge in A'_v. If $\tau = i$, there exists paths from s_i to v in D_i and ℓ is the minimum length of such a path by the definition of $\preceq$, implying $d_{D_i}(s_i, v) = \ell$. On the other side, when $\tau < i$, we can conclude that there is no path from s_i to v in D_i and that we have $d_{D_i}(s_i, v) = \infty$. Note that a shortest path tree from s_i in D_i can obtained from parent pointers allowing to recover minimum-cost walks such as constructed by our single-source all-reachable-edge minimum-cost algorithm. □

2.7 Conclusions

In this chapter we worked on the problem of computing optimal temporal walks in temporal graphs subjects to waiting constraints. We designed algorithms that compute minimum cost walks, from a single source node towards all possible destinations. The term "cost" is very general as it can model all classical criteria of optimisation, their linear combination and even lexicographical composition.

The algorithm proposed in the first part works under the assumption that the temporal graph at handle is zero-acyclic and solves the problem in linear time. The graph can be given in input either as a doubly-sorted representation of the temporal graph or as a, more classical, time-expanded representation. Indeed, we proved that the two representations are "equivalent", in the sense that it is possible to compute in linear time and space one from the other. Moreover, we prove some lower bounds showing that either by weakening the input assumption or dropping the zero-acyclicity property, an additional logarithmic factor is need. In particular if the input consists in a single sorted list of temporal edges, $\Omega(M \log M)$ time is needed for any comparison-only algorithm to compute minimum overall waiting time walks. On the other hand, if the zero-acyclicity assumption is dropped, even identifying the temporal edges that are reachable from a given source takes $\Omega(M \log n)$ time, assuming a lower bound of $\Omega(n \log n)$ for solving single-source shortest paths in static digraphs with n nodes and $\Theta(n)$ arcs.

The algorithm that we propose in the second part works without relying on the zero-acyclicity assumption, and solves the problem with time complexity $O(M \log n)$, that matches the lower bound. It relies on the first algorithm as a subroutine, but it needs to carefully reorder the lists of temporal edges during the execution.

Chapter 3

Temporal walks in public transit networks

Profile computation in public transit networks. In this chapter we are going to present a particular case of temporal graph model that is especially suitable for application in public transit networks. In a public transit network there are scheduled vehicles, like buses, trains or metros, that move from a stop to another following predefined routes. A scheduled vehicle leaves each stop in its route at specific points in time and takes a certain amount of time to reach the next stop. These movements can be modelled by a point availability temporal graph. However, it is also necessary to consider possible walking transfers from certain stops to some others. This enables, like in the real life application, to navigate the network in more efficient ways. An example of these concepts is represented in Figure 3.1. There are public available data containing the information to represent this type of networks. They are usually stored in what is called *General Transit Feed Specification (GTFS)* format, containing the public transportation schedules of vehicles together with a representation of the pedestrian network. The GTFS files consist in a collection of CSV files, that encodes routes, trips and stops with IDs, geographic locations and walking transfers. This type of representation can also be referred to as GTFS static, to highlight the fact that they are not updated with real time information.

In this chapter we will focus on the profile problem. We are given a source stop s, that represent the starting point of a user in the network, and a target stop t, that the user desires to reach. We are interested in a function that tells us for each possible departure time from s what is the earliest arrival time to the target, and what is the journey that leads to this arrival time.

Related work. The problem of studying profile computation in public transit networks has already received attention in the past decade. The two most notable works are from Dibbelt, Pajor, Strasser and Wagner [26] and Delling, Pajor and Werneck [25]. The first proposed an algorithm named *Connection Scan Algorithm (CSA)*, while the second introduced an algorithm called *Round bAsed Public Transit Optimized Router (RAPTOR)*. Both algorithms brought some novel approaches to solve the problem as they are not based on a Dijsktra-like procedure. However, even if experiments show how successful they are in practice, both algorithms lack a precise theoretical complexity analysis. The model of public transit network we present here is very closely related to the *timetable* model used in [26] and in [25].

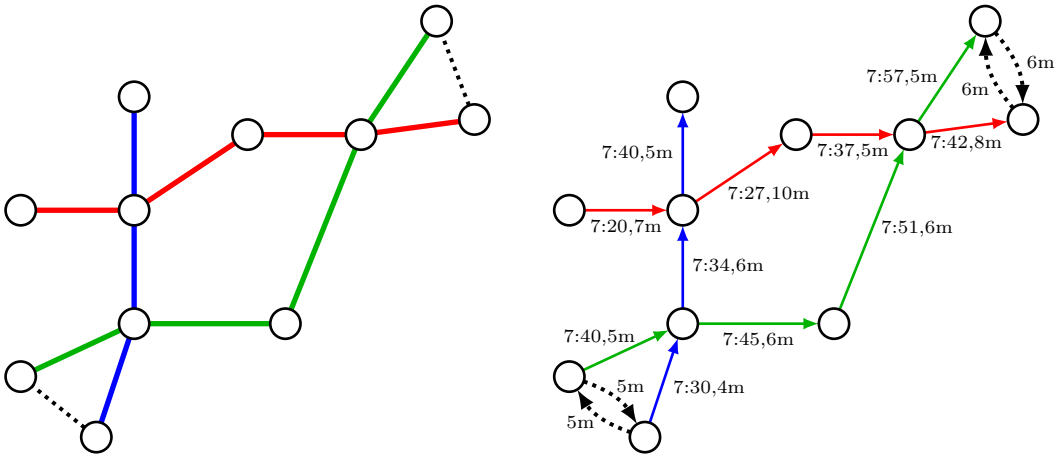

Figure 3.1: On the left, a partial representation of a public transit network through the routes that compose it and dotted arcs between stops that are reachable by foot from each other. On the right, a scheduled vehicle for each route represented on the right generates connections (temporal edges) between stops. There are dotted arcs representing the footpaths labelled with the travel time distance.

Contribution. We first briefly present the Connection Scan Algorithm [26] for profile computation, we then proceed to provide a time complexity analysis of this algorithm that was not present in the literature. Finally, we propose an algorithm for profile computation in public transit networks that relies on Algorithm 2 presented in Chapter 2. The algorithm we designed brings a slight improvement compared to the theoretical complexity of CSA. Moreover, due to its generality, it can be used to solve a great variety of journey computation problems in public transit networks. The notation that we use in this chapter is different from the one used in the rest of the dissertation. The aim is to propose an algorithm that could be useful to the community working in public transit networks. Thus, we tried to get closer to the application by adapting our notation, even though we give definitions that are consistent with what has been introduced so far.

3.1 Model

Transit network The aim of the model we are about to introduce is to depict the scenario where vehicles like buses, trains and metros travel through stops in a network following scheduled trips during the course of a day (or multiple days). A user can navigate such network by getting on and off scheduled vehicles at stops and also walking from a stop to another. In addition to that, we are able to represent a user preferences about how much time they are willing to wait at each stop.

We define a model that is a variation of the timetable model and that brings it closer to our temporal graph definition. After giving the definitions, we comment about similarities and differences with the temporal graph models introduced so far. We define a *transit network* as a tuple $N = (V, E, F, \beta)$, where V is a set of stops, E is a set of connections, F is a set of footpaths and $\beta \in [0, +\infty]^V$ is the maximum waiting-time at each stop.

A connection e is a quintuple $(u, v, \tau, \lambda, id)$, where:

- $u \in V$ is the *departure stop*, or *tail*, of e, denoted by $tail(e)$,
- $v \in V$ is the *arrival stop*, or *head* of e, denoted by $head(e)$,
- $\tau \in \mathbb{R}$ is the *departure time* of e, denoted by $dep(e)$,
- $\lambda \in \mathbb{R}_{>0}$ is the *travel time* of e,
- $id \in \mathbb{N}$ is the *trip ID* of e, denoted by $trip(e)$.

Moreover, we define the *arrival time* of e as $\tau + \lambda$, and we denote it by $arr(e)$. Connections with the same trip ID form a sequence $\langle e_1, \dots, e_k \rangle$ such that $head(e_i) = tail(e_{i+1})$ and $arr(e_i) \leq dep(e_{i+1})$ for $1 \leq i < k$. We denote by $next(e_i)$ the connection e_{i+1} after e_i in such a sequence. Similarly, we denote by $prev(e_i)$ the connection e_{i-1} preceding e_i in such a sequence.

On the other side, a footpath is a weighted arc $(u, v, \mu) \in F$, where $u, v \in V$, and $\mu \in \mathbb{R}_{\geq 0}$ is its weight, which represent the travel time that it takes to go walking from stop u to stop v. We assume the following properties on the set of footpaths. Let us denote by $G^F = (V, F)$ the footpath weighted digraph. We assume that G^F is transitively closed and that its edges satisfy the triangle inequality. This means that if there exist a footpath (u, v, μ_1) and a footpath (v, w, μ_2), then there exists also a footpath (u, w, μ_3), such that $\mu_3 \leq \mu_1 + \mu_2$. We also assume that for each stop $u \in V$ there exists a selfloop footpath (u, u, μ) which represents the time needed to change trip at stop u. Such selfloops are the only footpaths allowed to have zero duration. We define a *footpath event* as a pair $f = ((u, v, \mu), \tau)$ where $(u, v, \mu) \in F$ is a footpath and $\tau \in \mathbb{R}$ is a time label. A footpath event represent a passenger moving from a stop to another through a footpath by departing at time τ. Notice that a footpath event $((u, v, \mu), \tau)$ can correspond to a quintuple $(u, v, \tau, \mu, \perp)$, similarly to a connection with an undefined value for the *trip ID* field, denoting the fact that it indeed corresponds to a footpath rather than a connection. This way, similarly to connections, given a footpath event $f = ((u, v, \mu), \tau)$ we can refer to u as $tail(f)$, to v as $head(f)$, to τ as $dep(f)$ and to $\tau + \mu$ as $arr(f)$.

Let us introduce some notation that we will use for the remaining of the chapter. Given a transit network $N = (V, E, F, \beta)$ we denote with $M = |E|$ the number of connections. We denote with $N^F_{in}(u)$ and $N^F_{out}(u)$ respectively the in-neighbours and the out-neighbours of u in the footpaths graph G^F, with δ^{in}_u and δ^{out}_u the in-degree and out-degree of u in G^F, and let $\delta = \max_{u \in V} \max\{\delta^{in}_u, \delta^{out}_u\}$. Finally, we denote with Δ_u the number of connections $e \in E$ with $tail(e) = u$, and with $\Delta = \max_{u \in V} \Delta_u$. To represent lists of connections we adopt the same notation used for list of temporal edges in Chapter 2.

Journeys A journey represent a passenger's way to move from a stop to another one, possibly alternating connections and footpath events. A journey alternates sequences of connections within the same trip with footpaths that connect each sequence. In the following we will only consider journeys that contain at least a connection and that start with a footpath event. More formally, given a transit network $N = (V, E, F, \beta)$ a *journey* J from a stop u to a stop v, or a *uv-journey* for short, is a sequence of connections and footpath events $\langle x_1, \dots, x_k \rangle$ such that, x_1 is a footpath event, $tail(x_1) = u$, $arr(x_1) = dep(x_2)$, $head(x_k) = v$ and for $1 \leq i < k$:

- if x_{i+1} is a footpath event f, then x_i is a connection, $tail(f) = head(x_i)$ and $dep(f) = arr(x_i)$. Moreover, if x_{i+2} exists, it is also a connection and $arr(f) \leq dep(x_{i+2}) \leq arr(f) + \beta_v$, where $v = tail(x_{i+2})$,
- if both x_i and x_{i+1} are connections then $next(x_i) = x_{i+1}$.

We say that a connection e (resp. a footpath event f) *extends* a journey $J = \langle x_1, \ldots, x_k \rangle$ if the sequence $\langle x_1, \ldots, x_k, e \rangle$ (resp $\langle x_1, \ldots, x_k, f \rangle$) is a journey, and we denote it by $J.e$ (resp $J.f$). In this case, we also say that e (resp. f) extends x_k, since it does indeed extend any journey ending with x_k. Moreover, let s be a stop, we say that a connection e with arrival stop v is *s-reachable* if there exists a sv-journey ending with e. In the following we will assume that if a journey starts with a footpath event, the arrival of this footpath event coincides with the departure time of the first connection. Moreover, whenever a footpath event follows a connection, the departure time of the footpath event coincides with the arrival time of the connection.

This definitions of transit network and journey are closely related to the definitions of temporal graphs and temporal walk used so far, but with some key differences. The connections in this model correspond to temporal edges that are available at specific points in time, while footpaths correspond to temporal edges that have constant travel time and are available during the whole lifespan of the temporal graph. Thus, speaking about availability intervals and travel times only, transit networks can be seen as a particular case of piecewise constant temporal graphs. We can associate to a transit network a temporal graph in a way that each journey corresponds to a temporal walk in it. However, the fact that connections belong to trips constitutes an extension compared to the temporal graph models. Notice that the trip IDs are needed when there are two adjacent connections to know if a footpath loop is needed for the transfer or if it possible to stay in the same vehicle. Moreover, connections and footpath events have specific interactions when navigating the transit network through journeys, hence they have to be treated as different objects rather then considering them grouped into single set of temporal edges.

3.2 Complexity study of the Connection Scan Algorithm

In this section we first recall how the classical algorithm for public transit networks called *Connection Scan Algorithm (CSA)* operates. Then we proceed to study its complexity, which was not provided in the paper that introduced this algorithm. The algorithm, which was first proposed in [26], solves the profile problem in the formulation all-source single-destination. For simplicity, as it does not change the complexity of the algorithm, we will present the simpler version of the algorithm without parent pointers to extract journeys corresponding to points in the profile function. The algorithm is not designed to handle waiting time restrictions, thus, in the remaining of the section, β is intended to be ∞. Moreover, the algorithm only considers journeys starting and ending with a footpath event. The goal is to compute for each stop a representation of the profile function, stored as a list of pairs (departure time, arrival time) as explained in Section 2.4.6. The overall idea of the algorithm is to scan the list of connections by non-increasing departure time and compute journeys from later to earlier departure times, exploiting the fact that a journey can only have a journey departing later as suffix. A key observation in the following is that

a passenger who is currently using a connection e has three options to continue their journey towards the destination:

- exit the vehicle at the head of the connection and walk to the destination, if there is a footpath that enables it;
- keep sitting in the same vehicle, thus keep using the same trip, and take the next connection in the trip, if it exists;
- exit the vehicle at the head of the connection and get on another vehicle, meaning to take a connection that belongs to a different trip, either by changing vehicle at the same stop through the selfloop footpath or by walking to a different stop.

Now we describe more precisely how the profile functions are computed as formalized in Algorithm 8. First of all the transit network (V, E, F, β) is given in input as sorted list $\overleftarrow{E^{dep}}$ of the connections ordered by non-increasing departure time, and the footpath graph G^F is given as an adjacency array. Notice that $\overleftarrow{E^{dep}}$ in this section is thus sorted in the opposite way compared to E^{dep} in Chapter 2, we use a different notation to highlight the difference.

The algorithm performs a linear scan of the connections by non-increasing departure time. It makes use of three arrays T, S and D. Let us describe what is stored in each of these arrays at the k-th iteration. Let us denote with $N_k = (V, E^{dep}[1:k], F, \beta)$ the transit network induced by the first k connection in $\overleftarrow{E^{dep}}$, namely, the first k connections scanned. The array T stores for each trip ID id the earliest arrival time of a journey reaching the target stop t in N_k, i. e. starting with the last scanned connection in $\overleftarrow{E^{dep}}[1:k]$ with trip ID id if there is one. The array S stores for each stop s a representation of the profile function from s to t in N_k. The elements of S are Pareto optimal pairs (departure time, arrival time). The array D, stores for each stop s the walking distance to the target or ∞ if there is no footpath from s to the target. Notice that this array does not change during the execution as these distances are fixed. Let us see how these arrays get updated. At each iteration, when we scan the connection $e = (u, v, \tau, \lambda, id)$, the algorithm computes the earliest arrival time a_e of a journey to the destination t that uses e, and then integrates this information in S and T. The arrival time a_e is computed by analysing the three scenarios listed before. The first option corresponds to a_1, which consists in the arrival time of the connection, that is $\tau + \lambda$, plus the time to walk to the target $D[v]$. The second option corresponds to a_2, which is the earliest arrival time using the next connection of the trip, namely $next(e)$, and it is stored in $T[id]$, as $next(e)$ is the last connection scanned with trip ID equal to id. Finally the third option corresponds to a_3, which is computed as follows. The arrival time of the connection e in v is $\tau + \lambda$. We want the earliest arrival time to reach the destination, leaving v at time $\tau + \lambda$, either by changing trip in v or by walking to a different stop to catch another connection. This information is the result of a call to the profile function from v to t at time $\tau + \lambda$. The minimum among these three possible arrival times is the earliest arrival time $a_e = \{a_1, a_2, a_3\}$ of a journey that uses connection e. Now we include this information in the profile function of u and of each stop from which it is possible to walk to u. Let be u' a stop from which it is possible to reach u with a footpath of length μ. This means that the journey starting from u' at time $\tau - \mu$ and using connection e has arrival time a_e. To incorporate this information in $S[u']$ we add $(\tau - \mu, a_e)$ to the array if it is not dominated by other pairs, and then, if this is the case, we remove the pairs it dominates. Finally, to update $T[id]$ is sufficient to set it to a_e, as this is the earliest arrival time of a journey using e, which is now the last connection scanned with trip ID id.

Input: A list $\overleftarrow{E^{dep}}$ of the connections sorted by non-increasing departure time and adjacency list of the footpath graph representing a transit network $N = (V, E, F, \beta)$, and a target stop $t \in V$.

Output: A representation of the profile function from v to t is sorted in $S[v]$, for each stop $v \in V$.

For *each stop* $v \in V$ **do** Set $D[v] = \infty$ and $S[v] = \{(\infty, \infty)\}$.
For *each trip ID* id **do** Set $T[id] = \infty$.
For *each footpath* $(v, t, \mu) : v \in N_{in}^F(t)$ **do** set $D[v] = \mu$.
For *each connection* $e = (u, v, \tau, \lambda, id)$ *in* $\overleftarrow{E^{dep}}$ **do**
    Set $a_1 = \tau + \lambda + D[v]$.
    Set $a_2 = T[id]$.
    Let a_3 be the evaluation of $S[v]$ at time $\tau + \lambda$.
    Set $a_e = \min\{a_1, a_2, a_3\}$.
    If (τ, a_e) *is non-dominated in the profile* $S[u]$ **then**
        For *each footpath* $(u', u, \mu) : u' \in N_{in}^F(u)$ **do**
            Incorporate $(\tau - \mu, a_e)$ into the profile $S[u']$.
    Set $T[id] = a_e$.

Algorithm 8: Connection Scan Algorithm [26, 27]. It computes, for each stop v, a representation $S[v]$ of the profile function from v to t.

3.2.1 CSA complexity analysis

Let us analyse CSA 8 algorithm to study its time complexity. First we are going to consider the case where the algorithm is implemented as explained in [26, 27]. This means that the profile function representation from u to t is an array $S[u]$ of pairs (departure time, arrival time) sorted by non-increasing departure time. Every search in $S[u]$, is performed as a binary search, as the array is sorted. Elements can be appended in constant time. However, if a pair (d, a) has to be inserted in the array in position i, the pairs from position i to $|S[u]|$, that are the ones with departure time earlier than d, are sequentially removed, then (d, a) is appended if it is not dominated, and finally the removed pairs are appended back if they are not dominated by (d, a). This choice is likely to be the most efficient when the algorithm is run on real world instances, where we expect changes in $S[u]$ to occur mainly close to the end of the array. However, when it comes to the theoretical time complexity of the algorithm, different choices can lead to better results. This is the case when $S[u]$ is stored as a balanced binary search tree. We will cover this case after the proof of Theorem 8.

Theorem 8. *The worst case time complexity of Connection Scan Algorithm, with a basic array implementation of the profile function representation S, is $\Theta(M\delta^2\Delta)$.*

Proof. Upper bound on time complexity. First we are going to bound the number of elements added to the profile functions.

Observation: The number of pairs (departure time, arrival time) added to the representation $S[u]$ of a profile function from u to t is bounded by $\delta\Delta$. This is the case because $\delta\Delta$ bounds the number of all the possible ways to start a journey from u to t. Indeed, any journey from u starts with a footpath to a neighbour $v \in N_{out}^F(u)$ and a connection e with $tail(e) = v$. In particular, the number of neighbours of u in the footpath graph is bounded

by δ and the number of connections departing from a stop v is bounded Δ. This also means that the size of $S[u]$ is bounded by $\delta\Delta$ at any point during the execution.

The number of iteration performed by the algorithm is M, as in each iteration we scan a connection $e \in \overleftarrow{E^{dep}}$. The computation of a_1 and a_2 at Line 5 and Line 6 respectively, requires constant time. To compute a_3 we need to locate in $S[v]$ the two consecutive pairs whose departure times define the interval containing $\tau + \lambda$. The search requires $O(\log |S[v]|)$ when performed as a binary search. Similarly, the time needed to check if (τ, a_e) is dominated in $S[u]$ at Line 9 is $O(\log |S[u]|)$. Finally, the number of footpaths considered at Line 11 is bounded by δ. For each footpath (u', u, μ), the departure time $\tau - \mu$ has to be located in $S[u']$, which requires $O(\log |S[u']|)$ time. If $(\tau - \mu, a_e)$ is not dominated, it is added to $S[u']$ in $O(|S[u']|)$ time. With the same time complexity we also remove the pairs that are eventually dominated by $(\tau - \mu, a_e)$. Thanks to the above observation, we obtain a bound on the overall running time as $O(M\delta^2\Delta)$.

Reaching maximal complexity. Due to the assumption of transitive closure and triangle inequality of the footpaths, it is not trivial to understand whether there exist instances in which this time complexity is actually matched by the algorithm execution. Let us now prove that this is indeed the case. We define the following family of transit networks $N = (V, E, F, \beta)$ parameterised by δ, Δ, L, l and K. Let us assume that δ is even.

Stops. We define the set of nodes as $V = A \cup B \cup \{t\}$, where $A = \{a_1, a_2, \ldots, a_{\delta/2}\}$ and $B = \{b_1, b_2, \ldots, b_{\delta/2}\}$. The stop t is the destination for the all-source single-destination instance of the profile problem.

Footpaths. We define the set F of footpaths as follows:

- $(a_i, a_j, l) \in F$, for each $i, j \in [\delta/2]$,
- $(b_i, b_j, l) \in F$, for each $i, j \in [\delta/2]$,
- $(a_i, b_j, L) \in F$ and $(b_i, a_j, L) \in F$, for each $i, j \in [\delta/2]$,

where $L > l$. We will need further assumptions on the relation between L and l that we will discuss later. The idea is that each stop is close to the other stops in the same set, and far from the other ones. Finally, $(t, t, 0) \in F$, namely there is a self loop of length zero in t. Note that the set F of footpaths defined this way respects the properties of transitively closed and triangle inequality as required. Moreover, each stop except t has δ incoming footpaths and δ outgoing footpaths.

Connections. In the following we assume that each connection has a different trip ID. For simplicity we will omit the corresponding field and represent each connection as a quadruple. The set of connections is partitioned into two sets $E = E_A \cup E_B$, where:

$$E_A = \bigcup_{\substack{1 \le i \le \delta/2 \\ 0 \le k \le \Delta}} \left(a_i, t, K - \Delta\frac{\delta}{2} - k\frac{\delta}{2} - (i-1), K + 2h - k\frac{\delta}{2} - (i-1) \right),$$

$$E_B = \bigcup_{\substack{1 \le i \le \delta/2 \\ 0 \le k \le \Delta}} \left(b_i, t, K - k\frac{\delta}{2} - (i-1), K + h - k\frac{\delta}{2} - (i-1) \right),$$

and K and h are parameters that will be discussed later. Notice that for each stop $v \in A \cup B$ there are exactly Δ connections with tail v.

Execution. The connections are scanned by non-increasing departure time. This means that the connections in E_B are scanned before the connection in E_A, in particular $e = (b_1, t, K, K + h)$ being the first one. In the first iteration a_e is set to $K + h$, and the pair $(K - L, K + h)$ is added to the profile representation $S[a_i]$ for each stop $a_i \in A$. The next connection scanned is $(b_2, v, K - 1, K + t - 1)$, during this iteration, the pair $(K-1-L, K+t-1)$ is appended to the profile of each stop a_i. Note that this pair, has an earlier departure time than the previous one and it is not dominated, thus it is appended to $S[a_i]$ and no pair is removed from the profile. Once all the connections in E_B have been scanned, for each stop $a_i \in A$ its profile representation $S[a_i]$ is given by the elements:

$$\bigcup_{\substack{1 \leq i \leq \delta/2 \\ 0 \leq k \leq \Delta}} \left(K - k\frac{\delta}{2} - (i-1) - L, K + t - k\frac{\delta}{2} - (i-1) \right),$$

placed in an array sorted by non-increasing departure time. Each connection in E_B caused a new pair to be added to $S[a_i]$ with earlier departure time compared to the paris already in the array and without removing any other pair. At this point of the execution $|S[a_i]| = \frac{\delta}{2}\Delta$, for $i = 1, \ldots, \delta/2$. However, as each pair that is added has departure time earlier than all pairs already stored in $S[a_i]$, the computational cost is constant for each insertion.

In the next iterations we scan the connections in E_A. The network is designed with the idea that in these iterations, pairs (departure time, arrival time) are added to all the profiles $S[a_i]$ and have departure time greater than all pairs already stored. The first connection scanned from E_A is $(a_1, v, K - \Delta\frac{\delta}{2}, K + 2t)$. This leads to the insertion of the pair $(K - \Delta\frac{\delta}{2} - l, K + 2t)$ to $S[a_i]$, for $i \in [\delta/2]$. In order to accomplish our goal, we need $K - \Delta\frac{\delta}{2} - l$ to be greater than each departure time in $S[a_i]$, namely we are asking that:

$$K - k\frac{\delta}{2} - (j-1) - L < K - \Delta\frac{\delta}{2} - l \ , \ 0 \leq k \leq \Delta \text{ and } 1 \leq j \leq \frac{\delta}{2},$$

which is true if $L > \Delta\frac{\delta}{2} + l$. Note that this way, the pair we are adding to $S[a_i]$ is not dominated and does not dominate any other pair. To insert this pair we have to search in the profile representation $S[a_i]$ whose cardinality is $\Delta\frac{\delta}{2}$. Then we need to insert it in the first position of the array, which requires linear time in its size.

The next iterations generalise this process. Each one of the connections in E_A leads to the insertion in $S[a_i]$ of a pair (departure time, arrival time) with departure time grater than the departure times of the $\Delta\frac{\delta}{2}$ elements already in $S[a_i]$. In order for this to happen we need to require that the earliest departure time induced by a connection in E_A is greater than the latest departure time in $S[a_i]$, namely:

$$K - L < K - \Delta\frac{\delta}{2} - \frac{\delta}{2}(\Delta - 1) - \left(\frac{\delta}{2} - 1\right) - l,$$

which is true if $L > \Delta\delta - 1 + l$. Finally, we have to avoid the scenario where the new pairs dominate and delete all the pairs in the profile representations. To do this we have to make an assumption on h. In particular, we need that the new candidate pairs have an arrival time which is greater than the arrival time of the pairs that are already in the profile.

To do that we have to find a value of h such that the earliest arrival time of the new pairs is greater than the latest arrival time of the pairs previously stored:

$$K + h < K + 2h - (\Delta - 1)\frac{\delta}{2} - \left(\frac{\delta}{2} - 1\right),$$

which is true for $h > \Delta\frac{\delta}{2} - 1$.

To conclude, we produced an instance such that, when the profile function representations are stored as arrays, an execution of CSA has to perform at least $M/2$ iterations that require $O(\delta^2\Delta)$ operations each.

□

Let us now consider the case in which the profile function representations $S[u]$ are implemented as balanced binary search trees. We can bound the time needed to perform searches, insertions and deletions in $S[u]$ with $O(\log |S[u]|)$. Each connection $(u, v, \tau, \lambda, id)$ can lead to the creation of at most δ pairs, one for each incoming footpath in u, thus $M\delta$ in total. Each pair in a profile representation can be deleted only once. Thus the overall time complexity is bounded by $O(M\delta \log(\delta\Delta))$.

The family of transit networks defined in the proof of Theorem 8 leads also in this case to maximal complexity executions. Indeed, each search in the profile representation $S[u]$ requires $O(\log \delta\Delta)$. Notice that as we are adding specifically elements with lowest or greatest departure times this searches could be performed in linear time with a slight modification in the data structure. However, it is possible to define a relation between L and l such that the departure time of the the pairs added from the scan of E_A is in arbitrary positions of the profiles. In this case, the pair would not be added, as it is dominated by the pairs with greater departure time, since they have earlier arrival time as well. Nevertheless, the algorithm needs $O(\log(\delta\Delta))$ to locate the correct interval in the profile.

We can thus formulate the following result.

Theorem 9. *The worst case time complexity of Connection Scan Algorithm, when profile function representations are implemented as balancaed binary search trees, is* $\Theta(M\delta \log(\delta\Delta))$.

3.3 Double Scan Algorithm

Let us now present an algorithm that extends Algorithm 2 for computing minimum-cost journeys in transit networks, solving, in particular, the profile problem as well. The idea is to emulate an execution of Algorithm 2 in a temporal graph obtained from the transit network. In this temporal graph each connection is turned into a temporal edge, and each footpath is turned into 2Δ temporal edges. Indeed, even if a footpath (u, v, μ) is available during the whole lifespan of the temporal graph, we are only interested in footpath events $((u, v, \mu), \tau)$ such that τ coincides with the arrival time of a connection with head u or such that $\tau+\mu$ coincides with the departure time of a connection with tail v. However, generating an instance of this temporal graph and reordering its temporal edges would not be efficient in terms of time and space.

General cost structure for journeys. We integrate a transit network $N = (V, E, F, \beta)$ with an algebraic *cost structure* $(C, \gamma, \oplus, \preceq)$, similarly to what we did in Chapter 2 for temporal walks. The set C is the set of possible *cost values*, γ is a *cost function* $\gamma : E \cup (F \times \mathbb{R}) \to C$, $\oplus$ is a *cost combination function* $\oplus : C \times C \to C$, and $\preceq$ is a *cost total*

order $\preceq \subseteq C \times C$. We also define the relation $\prec$ between the elements of C as $a \prec b$ if and only if $a \preceq b$ and $a \neq b$. For any journey $J = \langle x_1, \ldots, x_k \rangle$, the *cost function* of J is recursively defined as follows: $\gamma_J = \gamma_{\langle x_1,\ldots,x_{k-1}\rangle} \oplus \gamma(x_k)$, with $\gamma_{\langle x_1 \rangle} = \gamma(x_1)$. In other words, the costs combine along the journey according to the cost combination function. We suppose also in this case that right-isotonicity property is satisfied, as defined in Section 2.3. This guarantees that if several journeys are extended by a given connection or footpath event x, then the best cost is obtained by extending the journey J^* with minimum cost: as for any other journey J we have $\gamma_{J^*} \preceq \gamma_J$, we get $\gamma_{J^*.x} \preceq \gamma_{J.x}$ by the isotonicity property and the cost function definition. However, a prefix of a minimum-cost journey is not necessarily a minimum-cost journey. Finally, we denote an undefined cost as $\perp$ and consider that $c \prec \perp$ for any cost $c \in C$.

To solve the problem of computing minimum-cost journeys from a single source s, we will consider a more general problem consisting in computing at each destination v, and for each possible s-reachable connection e with arrival stop v, an sv-journey with minimum cost among all sv-journeys ending with e.

We define the single-source all-reachable-connection minimum-cost problem as follows.

> SINGLE-SOURCE ALL-REACHABLE-CONNECTION MINIMUM-COST PROBLEM. Given a transit network $N = (V, E, F, \beta)$ with cost structure $(C, \gamma, \oplus, \preceq)$, and a source stop $s \in V$, compute for each destination $v \in V$ and each possible s-reachable connection e with arrival stop v the minimum cost of any sv-journey ending with connection e.

Notice that we are using a single-source all-destination kind of approach, rather then the all-source single-destination used in CSA. In this formulation of the problem, and in the algorithm we designed, we are considering journeys that end with a connection. However, it is possible to obtain also the journeys that end with a footpath event by applying a patch as explained after the proof of Theorem 10, without changing the time complexity.

Letting A_v denote the set of all s-reachable connections with head v, the problem consists in computing for each node v all pairs (e, c) such that $e \in A_v$ and $c = \min\{\gamma_J : J \text{ is an } sv\text{-journey ending with connection } e\}$. We will denote with A'_v the list of such pairs (e, c) ordered by non-decreasing arrival time of the connections. In the next algorithmic result, we represent a transit network (V, E, F, β) as two lists (E^{dep}, E^{arr}) and an adjacency list representation of G^F. The lists contain $|E|$ quintuples each, representing all connections in E, where E^{arr} is sorted by non-decreasing arrival time and E^{dep} by non-decreasing departure time. We assume that we have implicit pointers between the two lists, that link each quintuple of one list to the quintuple representing the same connection in the other list.

We adapt the ideas from Algorithm 2 to handle footpaths and solve the single-source all-reachable-connection minimum-cost problem. The algorithm consists in a main scan of the connections sorted by non-decreasing arrival time. When a connection e is scanned we compute the minimum cost of a journey that e can extend, which consists in the most involved operation of the algorithm. Then, with this information, we compute the minimum cost of a journey ending with e, taking into account that e might be the first connection of a journey that was not detected before.

Let us describe more precisely the operations performed by Algorithm 9, focusing on the differences compared to Algorithm 2. The first task when we scan a connection $e = (v, w, \tau, \lambda, id)$ consists in computing its best extendable cost, namely the minimum cost

among the journeys from the source that e can extend. We can distinguish two types of such journeys to v:

- journeys that end with a footpath event and that have an arrival time consistent such that e can extend it, taking into account the waiting time restriction,
- journeys that end with the connection that precedes e in the trip.

The first case is the hardest to handle. The idea for finding this type of journeys is to consider journeys ending with a connection in a stop u where it is possible to walk to v. This information, together with the minimum cost for such journeys is stored in A'_u for $u \in N^F_{in}(u)$. Each connection in these sets, corresponds to a journey that, when extended with a footpath event, leads to a journey to v that e might extend. To handle together all these journeys we apply the same interval strategy as we did in Algorithm 2. The main difference is that in the temporal graph case, a temporal edge from v could only extend temporal walks ending in v, thus it was sufficient to analyse the sorted set A'_v to produce the correct intervals. In this case, on the other hand, we do not have the whole set of connections in $\bigcup_{u \in N^F_{in}(u)} A'_u$ sorted by arrival time. For this reason we need to introduce a priority queue. In the priority queue we add the connections from $\bigcup_{u \in N^F_{in}(u)} A'_u$ based on their arrival time plus the travel time of the footpaths from u to v. We can thus extract them in order and build correctly the intervals. Due to the order in which we are scanning the connections, each element of $\bigcup_{u \in N^F_{in}(u)} A'_u$ has to be considered just once. However, notice that a connection in A'_u needs to be considered again when in the main cycle we scan a different connection e' with tail v' if there is a footpath from u to v'. The indexes p_{uv} associated to each arc in G^F serve exactly this purpose.

The second case is handled in the moment e gets finalized. It is sufficient to follow a pointer to the connection $prev(e)$ preceding e in the trip. This connection has already been scanned and finalized, due to the earlier arrival time, which means the minimum cost of a journey ending with $prev(e)$ has already been computed.

Input: Two sorted lists (E^{dep}, E^{arr}) of connections and an adjacency list of the footpath graph representing a transit network (V, E, F, β), a cost structure $(C, \gamma, \oplus, \preceq)$ satisfying isotonicity, and a source stop s.

Output: Minimum cost of an sv-journey for each node v and for each s-reachable connection e with arrival stop v.

For each stop v, generate the list E_v^{dep} by bucket sorting E^{dep}.
For *each stop* v **do**
  Set $A'_v := \emptyset$. `/* List of pairs (s-reachable connection, cost). */`
  Set $p_{uv} := 1$ for each $u \in N_{in}^F(v)$. `/* Next connection to consider in` A'_u `that leads to an arrival at` v`. */`
  Set $\mathcal{I}_v := \emptyset$. `/* List of consecutive intervals of` E_v^{dep}`. */`
  Set $(l_v, r_v) := (1, 0)$. `/* Overall interval of` E_v^{dep} `spanned by` $\mathcal{I}_v$`. */`
For *each connection* $e \in E^{arr}$ **do**
  Set best extendable cost $B[e] := \perp$ and parent pointer $P[e] := (\perp, \perp)$.
For *each connection* $e = (v, w, \tau, \lambda, id)$ *in* E^{arr} **do**
  Let i be the index of e in E_v^{dep}.
  `/* Compute` $B[e]$ `for each connection in` E_v^{dep} `up to` e `included. */`
  UpdateCost(v, i)
  If $v = s$ *or* $v \in N_{out}^F(s)$ *or* $B[e] \neq \perp$ **then**
    `/* Get the minimum cost` c `of a journey ending with` e`: */`
    If $v = s$ **then** set $c_1 := \gamma(e)$.
    If $v \in N_{out}^F(s)$ **then**
      Let $(s, v, \mu_{sv}) \in F$ be the footpath from s to v.
      Set $f := ((s, v, \mu_{sv}), dep(e) - \mu_{sv})$ and $c_2 := \gamma(f) \oplus \gamma(e)$.
    If $B[e] \neq \perp$ **then** set $c_3 := B[e] \oplus \gamma(e)$.
    Set $c := \min\{c_1, c_2, c_3\}$.`/* Minimum among defined costs. */`
    If $c = c_1$ **then** $P[e] := (\perp, \perp)$.
    else if $c = c_2$ **then** $P[e] := (\perp, f)$.
    Append (e, c) to A'_w.
Return *the lists* $(A'_v)_{v \in V}$.

Algorithm 9: Computing, for each stop v and each s-reachable connection e with head v, the minimum cost of any sv-journey ending with e.

Procedure UpdateCost(v, j)
  Set $\tau := dep(E_v^{dep}[j])$.
  Initialize a priority queue $Q := \emptyset$.
  For *each* $u \in N_{in}^F(v)$ **do**
    Let (u, v, μ_{uv}) be the footpath from u to v and Enqueue$(Q, (u, v, \mu_{uv}), \tau)$.
  While $Q \neq \emptyset$ **do**
    $(e, c, (u, v, \mu_{uv})) :=$ PopMin(Q).
    Set $a := arr(e) + \mu_{uv}$ and $f := ((u, v, \mu_{uv}), arr(e))$.
    `/* Find interval` (l, r) `of connections in` E_v^{dep} `extending` f`: */`
    Let $l \geq l_v$ be the first index of a connection $(v, w, \tau', \lambda', id') \in E_v^{dep}$ such that $\tau' \geq a$ (set $l := |E_v^{dep}| + 1$ if no such index exists).
    Let $r \geq r_v$ be the last index of a connection $(v, w, \tau', \lambda', id') \in E_v^{dep}$ such that $\tau' \leq a + \beta_v$ (set $r := r_v$ if no such index exists).
    `/* Remove from` $\mathcal{I}_v$ `intervals with cost greater than` c`: */`
    Set $l_c := \max\{l, r_v + 1\}$. `/* First index in` (l, r) `after` $\mathcal{I}_v$`. */`
    While $\mathcal{I}_v \neq \emptyset$ *has last interval* $I' = (l', r', c', (e', f'))$ *satisfying* $c \prec c'$ **do**
      Remove I' from $\mathcal{I}_v$ and update $l_c := l'$.
    `/* Assign cost` c `and parent` (e, f) `to connections` $E_v^{dep}[l_c : r]$`: */`
    If $l_c \leq r$ **then** append interval $I = (l_c, r, c, (e, f))$ to $\mathcal{I}_v$.
    FinalizeCosts$(v, l_v, l - 1)$
    Set $l_v := l$ and $r_v := r$.
    `/* Insert next element of` A'_u `in the priority queue */`
    Enqueue$(Q, (u, v, \mu_{uv}), \tau)$
  FinalizeCosts(v, l_v, j)
  Set $l_v := j + 1$.
Procedure Enqueue$(Q, (u, v, \mu_{uv}), \tau)$
  If $p_{uv} \leq |A'_u|$ **then**
    Set $(e, c) := A'_u[p_{uv}]$ and $\bar{c} := c \oplus \gamma((u, v, \mu_{uv}), arr(e))$.
    Associate $(e, \bar{c}, (u, v, \mu_{uv}))$ with key $Key(e, \bar{c}, (u, v, \mu_{uv})) = arr(e) + \mu_{uv}$.
    If $Key(e, \bar{c}, (u, v, \mu_{uv})) \leq \tau$ **then**
      Add $(e, \bar{c}, (u, v, \mu_{uv}))$ to Q and set $p_{uv} := p_{uv} + 1$.
Procedure FinalizeCosts(v, i, j)
  `/* Find connections in` $E_v^{dep}[i : j]$ `reachable using the same trip */`
  For *each connection* $e' = (v, w, \tau, \lambda, id)$ *in* $E_v^{dep}[i : j]$ **do**
    If $B[prev(e')] \neq \perp$ **then**
      $B[e'] := B[prev(e')] \oplus \gamma(prev(e'))$ and $P[e'] := (prev(e'), \perp)$.
  `/* Get the best extendable cost and parent of connections in` $E_v^{dep}[i : j]$ `and remove corresponding intervals. */`
  While *the first interval* $I = (l, r, c, (e, f))$ *in* $\mathcal{I}_v$ *satisfies* $l \leq j$ **do**
    Let $l' = \min\{r, j\}$.
    For *each connection* $e' = (v, w, \tau, \lambda, id)$ *in* $E_v^{dep}[l : l']$ **do**
      If $B[e'] = \perp$ *or* $c \prec B[e']$ **then** $B[e'] := c$ and $P[e'] := (e, f)$.
    If $l' = r$ **then** remove I from $\mathcal{I}_v$ **else** update $I := (j + 1, r, c, (e, f))$.

Algorithm 10: Associate its best extendable cost to each connection e in E_v^{dep} up until the j-th one.

3.3.1 Double Scan complexity analysis

Theorem 10. *Given a transit network $N = (V, E, F, \beta)$ a source node s, Algorithm 9 solves the single-source all-reachable-connection minimum-cost problem in $O(M\delta \log \delta)$ time and linear space.*

Proof. Let us start by analysing the complexity of the preprocessing steps. First we compute the lists E_v^{dep} by bucket sorting E^{dep}, which can be done in $O(M)$ time. Initialising A'_v, $\mathcal{I}_v$ and (l_v, r_v) requires $O(n)$ time, while each index p_{uv} corresponds to footpath in F, thus they can be initialised in $O(|F|)$ time. To conclude the preprocessing, the array of best extendable costs B and parent pointers P can be defined in $O(M)$ time. In the following we will assume that operations with $\oplus$ and $\preceq$ can be computed in constant time. Let us consider the main cycle in the execution. At each iteration we scan a connection $e = (v, w, \tau, \lambda, id)$ from E^{arr}, thus the algorithm performs M iterations. For the moment we neglect the call to UpdateCost, and we will come back to it with an amortized analysis. During each iteration we might compute three different costs c_1, c_2 and c_3. Cost c_1 and c_3 can be computed in constant time, while c_2 requires δ time to scan the adjacency list. Computing the minimum among these three costs, setting the parent pointer and appending the connection to A'_w requires constant time. Let us conclude with an amortized analysis of the calls to UpdateCost. First we analyse the cost needed to handle the priority queues. Each footpath $(u, v, \mu_{uv}) \in F$ is added to a priority queue, together with a time instant τ, at most $|A'_v|$ times, thanks to the update of index p_{uv}. We can thus bound the overall number of elements enqueued and removed from the priority queues by $(\sum_{v \in V} |A'_v|)\delta \leq M\delta$. Moreover, at any time during the execution of a call UpdateCost(v, j), in the priority queue Q there is at most an element per footpath entering v. Moreover, the key associated to each element can be computed in constant time. This means that the size of the priority queue is bounded by $|N_{in}^F(v)| \leq \delta$. Thus we can enqueue and pop each element with a time complexity of $O(\log \delta)$. The total cost to handle the priority queues can be expressed as $M\delta \log \delta$. The intervals $\mathcal{I}_v$, similarly to Algorithm 2 are handled in linear time with respect to the number of arrival times to v. In the case of temporal graphs this was bounded by the number of temporal edges entering v. In the case of transit networks this is bounded by the number of connections entering a neighbour of v in the footpath graph. Overall we can thus bound the total cost to handle intervals with $O(M\delta)$. Finally, the calls to FinalizeCosts require linear time in the number of connections in E, as in Algorithm 2. The only difference is that for each connection, for computing its correct best extendable cost B before finalizing it, we also need to check the cost obtained by coming from the previous connection of the same trip. This can be done in constant time for each connection. To conclude, the time complexity needed for the calls to UpdateCost and of the whole execution of Algorithm 9 is $O(M\delta \log \delta)$.

Let us turn to the space complexity. Each set of interval $\mathcal{I}_v$ contains at most $|E_v^{dep}|$ intervals, each set A'_v has size bounded by the number of connections with head v. Thus we have $\sum_{v \in V} |\mathcal{I}_v| \leq |E|$ and $\sum_{v \in V} |A'_v| \leq |E|$. As we already mentioned the size of each priority queue Q is bounded by δ. Finally, the number the number of indexes $p_u v$ is $|F|$. We can conclude that Algorithm 9 runs in linear space $O(|E| + |F|)$. □

As we mentioned before, Algorithm 9 solves the problem of computing the optimum cost of journeys ending with a connection. It is possible to patch the algorithm to compute also journeys that end with a footpath event. To do this we introduce for each node v a set A''_v that contains the minimum cost for both journeys ending with connection and

footpath event with $head(v)$. The connections are added exactly in the same way as we add them to A'_v. For the footpath events, whenever we pop an element $(e, c, (u, v, \mu_{uv}))$ from Q we add to A''_v the footpath event $((u, v, \mu_{uv}), arr(e))$ together with cost c. Notice that these values are already computed by the algorithm but they are not stored for efficiency reason. Indeed, computing also these journeys ending with a footpath event would lead to an increase of space complexity to $O(M\delta)$, since each connection ending a journey corresponds to δ journeys ending with a footpath event.

If we are interested in a single destination t, rather than patching the algorithm it is possible to do a post processing step, that does not change the time and space complexity of the algorithm. It is sufficient to consider the sets A'_v for each $v \in N^F_{in}(t)$, extract their elements with the help of a priority queue. Let (v, t, μ_{vt}) denote the footpath from v to t. Each element (e, c) from A'_v corresponds to a journey from s to t ending with a footpath event $f = ((v, t, \mu_{vt}), arr(e))$ and cost $c \oplus \gamma(f)$.

3.3.2 Computing optimal journeys

The algebraic cost structure we introduced for journeys allows us to model and solve many different problems, as we did for the case of point availability temporal graphs in Section 2.4. We can easily define cost structures to solve profile, shortest duration and minimum waiting time problems, considering connections and footpath events as temporal edges.

Profile The first step is to define a cost structure that allows to compute shortest duration journeys. As we did before, to achieve this results, we actually compute for a fixed connection e journeys with latest departure time ending with e. To find these journeys we define the following cost structure $(C, \gamma, \oplus, \preceq)$. The cost set is $C = \mathbb{R}$, to each connection $e = (u, v, \tau, \lambda)$ we assign as cost its departure time $\gamma(e) = \tau$. Similarly, to each footpath event $f = ((uv, \mu_{uv}), \tau)$ we assign $\gamma(f) = \tau$. The cost combination function $\oplus$ return the first parameter, thus $\tau \oplus \tau' = \tau$. Finally, the cost total order is defined by $\tau \preceq \tau'$ when $\tau \geq \tau'$, meaning that later departure times are preferred. To obtain a representation of the profile function from s to v it is sufficient to remove from A'_v the paris dominated in a Pareto sense with a linear scan.

There are some other interesting problems we can easily model and solve in this setting.

Minimum number of trips The problem consists in computing journeys that minimise the number of different trips that appear in the journey. As every time we change trip in a journey, we need by definition to use a footpath between the connections belonging to two different trips, it is sufficient to count the number of footpaths. Thus, we define the following cost structure $(C, \gamma, \oplus, \preceq)$, where $C = \mathbb{N}$, $\oplus = +$ and $\preceq = \leq$. Then, we define $\gamma(e) = 0$ for each $e \in E$, $\gamma((u, v, \mu_{uv}), \tau) = 1$ for each footpath event. Given a journey J, its cost γ_J is number of trips used in J, thus $\gamma_J - 1$ corresponds to the number of trip changes in J. With a linear scan of the set A'_v we can thus compute the minimum number of trips of a journey from s to v.

Minimum walking time The problem consists in computing journeys that minimise the time spent walking during the journey. In this case we define the following cost structure $(C, \gamma, \oplus, \preceq)$, where $C = \mathbb{R}$, $\oplus = +$ and $\preceq = \leq$. Then, we define $\gamma(e) = 0$ for each $e \in E$, $\gamma((u, v, \mu_{uv}), \tau) = \mu_{uv}$ for each footpath in F. This way the cost γ_J of a journey J correspond

to the sum of the duration of the footpaths along the journey. To conclude, with a linear scan of the set A'_v we can thus compute the minimum time spent walking needed from a journey from s to v.

Pareto optimal journeys By proceeding in a similar way as we did in Section 2.4.5 we can compute Pareto optimal sets combining arrival time and any other cost we can define with the algebraic cost structure. An interesting case is to optimize arrival time and number of trips in a Pareto sense. This is done also by Dibbelt et al. [27], motivated by the fact that an optimal journey for a certain single criterion could actually be a bad choice in practice.In order to solve this problem they bound the maximum number of trips used in a journey by a value t_{max}. Then, all the variables for each trip ID $T[id]$ and for each stop $S[v]$ appearing in the algorithm have to be represented as vectors of length t_{max}. For each entry i of these vectors they store the value that correspond to journeys with at most i trips. On the other side, we do not need to modify our algorithm in any way and we do not need extra space usage to solve this problem.

3.3.3 Conclusions and future work

We summarised and analysed the complexity of Connection Scan Algorithm, an algorithm that computes single-destination profiles in public transit networks. Then we propose a flexible algorithm that in its generality can solve the single-source profile problem and several other journey computation problems. The time complexity of our algorithm slightly improves over the CSA one and can take waiting restrictions into account.

Future work resides in implementing Algorithm 9, to test it on GTFS datasets. and compare it with CSA. The slight advantage that it brings in terms of theoretical complexity does not guarantee a better performance in real life applications. However, we expect better results in the context of Pareto optimal journeys, as the technique presented for CSA does not seem to be the most efficient. Finally, it would be interesting to study the theoretical complexity of RAPTOR as well, and compare with it our implementation of Algorithm 9.

Chapter 4

Walk temporalisation

This chapter is based on the results obtained in **[10, 3]**.

Inspired by the problem of scheduling buses/metros/tramways in a public transport network, we consider the problem of assigning departure times to the arcs of a directed graph so that the resulting temporal graph maximises temporal reachability. Given a weighted directed multigraph D with strictly positive weights, an *arc temporalisation* assigns to each arc (u, v, λ) of D a departure time τ, making it a temporal edge (u, v, τ, λ). For example, the weighted directed multigraph D could represent the map of a public transit network where the weight of an arc represents the time needed by a vehicle to travel along that arc. Multiple types of vehicle traversing the same arc can be captured by multiple arcs with appropriate weights.

However, in this public transit context, the arcs are not "independent", in the sense that a departure time cannot be assigned to an arc independently of the departure time assigned to other arcs. Indeed, we assume that the trajectory of each bus/metro/tramway is fixed and is given by a walk in the digraph so that the arcs of that walk must be scheduled one right after another. We are thus given a collection $\mathbb{T}$ of walks in D that we call *trips*, in reference to the application to transit networks, to distinguish them from other arbitrary walks in D. When several vehicles travel along the same walk, we assume that a distinct trip is associated to each one of them. We also suppose that the time spent by a vehicle between arriving at a stop and departing from it is negligible and that scheduling a vehicle amounts to assigning a departure time to the first arc of the corresponding trip, and that all the other departure times are a consequence of it: indeed, each temporal edge resulting from a scheduled trip departs right after the arrival of the previous one. The operation of assigning a starting time to each walk in $\mathbb{T}$ is called *trip temporalisation*. The temporal graph *induced* by a trip temporalisation of $\mathbb{T}$ is the temporal graph whose node set is the same as the node set of D, and whose set of temporal edges is the disjoint union of all the temporal edges resulting from the assignment of the starting times to the walks in $\mathbb{T}$. As a specific point in time is assigned to each arc in a trip as departure time, rather than a whole availability interval, the induced temporal graph is a point availability temporal graph. Moreover, for the rest of the chapter we will consider temporal graphs that are not subject to waiting constraints.

The goal is to maximise the *reachability* of the induced temporal graph, that is the number of pairs of nodes that are temporally connected.

To summarise, the main network optimisation problem that we will analyse in this chapter is called Maximum Reachability Trip Temporalisation (in short, mrtt): *given a weighted directed multigraph D and a collection $\mathbb{T}$ of walks on D, find a trip temporalisation of $\mathbb{T}$ which maximises the reachability of the induced temporal graph.*

Related work. Optimisation of timetables in a transit network has been studied as an operation research problem (for a survey see the work of Cacchiani and Toth [16]) at a fine grained level of modelling, taking into account sharing of route segments or tracks, and mixing various objectives such as operation costs or overall user waiting time when the traffic demand is known. We have a higher level approach that aims at grasping the connectivity of the network.

Some problems similar to the one considered in this chapter have already been analysed in the literature. For instance, Kempe, Kleinberg and Kumar [42] study a problem that consists in reconstructing a partially specified time-labelling of a network in a consistent way with an observed history of information flow. The input is an undirected graph such that a time interval is assigned to each arc, together with a source node, and two disjoint sets of nodes P and N. The goal is to assign to each arc a departure time that has to lie inside the corresponding time interval, in a way that P is temporally reachable from r and N is not. They provide a polynomial time algorithm that detects whether or not such an assignment exists, and returns one whenever it does. Another related problem is considered by Enright, Meeks, Mertzios and Zamaraev [30], where the authors analyse the problem of deleting edges from a given temporal graph in order to reduce its reachability, motivated by the context of epidemiology. Later on, the problem of assigning departure times to the arcs of a graph in order to minimise the reachability of the resulting temporal graph is studied by Enright, Meeks and Skerman [31]. Another example is from Mertzios, Michail and Spirakis [47], where the authors propose two cost minimisation parameters for temporal network design (that is, the maximum number of departure times of an edge and the total number of departure times of all edges), and study the problem of optimizing these parameters subject to some connectivity constraint. Another closely related work from Deligkas and Potapov [24] studies the problem of minimising the average reachability (as well as other similar objectives which could be interesting goals in the context of transport networks too) in a temporal graph by delaying some edges. Various NP-hardness results as well as a polynomial-time algorithm are given, depending on the type of delay operations that are permitted. The authors leave as open the complexity of maximising reachability which is similar to our goal. The same problem of delaying has then been further investigated by Molter, Renken and Zschoche [48].

As far as we know, the trip temporalisation problem has never been studied before in such generality. The topic of temporalising arcs to increase reachability is related to gossip and broadcasting protocols (for a survey see the work of S. M. Hedetniemi, S.T. Hedetniemi and Liestman [40]). The problem studied by Göbel, Cerdeira and Veldman [39] is closely related to our setting when the trips are one arc long (case that we also analyse) but in an undirected model. They show that the problem of deciding whether the resulting temporal graph is temporally connected (that is, for any two nodes u and v, v is temporally reachable from u) is NP-complete (clearly, this implies that the mrat problem restricted to undirected graphs is NP-hard). It is also easy to see that the mrat problem restricted to undirected connected graphs can be approximated within a constant approximation ratio, since this simply requires to look for a "centroid" in a spanning tree as a temporalisation where half of the nodes can reach the other half can then easily be computed. Note, however, that temporalising a symmetric digraph is not equivalent to temporalising an undirected

graph as different times can be assigned to an arc (u, v) and the symmetric arc (v, u). Nonetheless, apart from [39] where the undirected setting and the independence of edges makes the problem different from here, the objective in gossiping is usually different, that is, minimising the time for a message to reach all nodes.

Contribution. Our results are summarised in Table 4.1, where three other combinatorial problems are also considered. The first decision problem, denoted by O2O-RTT, is the one-to-one version of the MRTT problem, in which the question is whether a trip temporalisation exists making a given target node t temporally reachable from another given source node s. The second maximisation problem, denoted by SS-MRTT, is the single-source version of the MRTT problem, in which the question is to find a trip temporalisation maximising the number of nodes temporally reachable from a given source node s. Finally, the third maximisation problem, denoted by MRAT which stands for MAXIMUM REACHABILITY ARC TEMPORALISATION, is the particular case of MRTT where all trips are one arc long.

Note that, although we assume that the arcs of a trip must be scheduled one right after the other, our results can be generalized in a setting where, for each pair of consecutive arcs of a trip, a fixed amount of time is imposed between the arrival and departure time. The reason is similar to the fact that we can restrict ourselves to a setting where all travel times are 1 as explained in the Section 4.1.

Quite surprisingly, our first result (see Theorem 11) shows that *the* O2O-RTT *problem is NP-complete.* Using a classical gap technique, we then obtain that if P $\neq$ NP, then *the* MRTT *and the* SS-MRTT *problems cannot be approximated within a factor* $n^{1-\varepsilon}$ *for any* $\varepsilon > 0$*, where* n *is the number of nodes* (see Theorems 12 and 13). We also show that the parameterised version of the O2O-RTT problem with respect to the number k of trips used in the resulting temporal graph, in order to go from s to t, can be solved in time $2^{O(k)} M \log |\mathbb{T}|$ where $M = \sum_{T \in \mathbb{T}} |T|$ is the sum of the number of arcs that compose each trip (see Theorem 14).

The above non-approximability results are the main reason for focusing our attention to an interesting restriction of the MRTT problem, that is, the one in which the collections of trips $\mathbb{T}$ satisfies the very natural property of being "temporally" strongly connected in the following sense. A collection of trips $\mathbb{T}$ is *strongly temporalisable* if, for each pair of nodes u and v, there exists a trip temporalisation of $\mathbb{T}$ that allows u to (temporally) reach v. Note that this requirement is a rather weak one, since we are not asking for a unique trip temporalisation, but for a trip temporalisation for each pair of nodes (indeed, this is a requirement which is satisfied in many applications of temporal graphs). We first show that the strong temporalisability property is not sufficient to get high reachability. To this aim, we prove that *there exists an infinite family of trip collections, all strongly temporalisable, such that any trip temporalisation connects at most an* $O(1/\sqrt{n})$ *fraction of all pairs* (see Theorem 15). By using this construction, we then show that if P $\neq$ NP, then *the* MRTT *and the* SS-MRTT *problems cannot be approximated within a factor less than* $\sqrt{n}/12$*, when restricted to strongly temporalisable trip networks* (see Theorems 16 and 17).

However, the situation changes if we add another quite natural property of a trip collection, that is, symmetry. A trip collection $\mathbb{T}$ is *symmetric* if, for each trip $T \in \mathbb{T}$, $\mathbb{T}$ includes also a reverse trip, that is, a trip starting from the last node of T, arriving in the first node of T, and passing through all the nodes in T in reverse order. Referring to public transport systems, symmetry is almost always respected, since for any bus/metro/tramway trip, there is usually also the same bus/metro/tramway trip in the opposite direction. It is quite easy to show that, if the collection of trips $\mathbb{T}$ is *symmetric*, then $\mathbb{T}$ is strongly temporalisable if and only if the weighted directed multigraph D is strongly connected (see Corollary 3).

Maximisation problems	
Problem	Complexity
MRTT	Not approximable within a factor $n^{1-\varepsilon}$ for any $\varepsilon > 0$ (Theorem 12)
SS-MRTT	Not approximable within a factor $n^{1-\varepsilon}$ for any $\varepsilon > 0$ (Theorem 13))
MRAT	NP-hard (Theorem 20)

Decision problems	
Problem	Complexity
O2O-RTT	NP-complete (Theorem 11)
k-O2O-RTT	Solvable in time $2^{O(k)} l \log \lvert\mathbb{T}\rvert$ (Theorem 14)

Problem	Property	
	Strongly temporalisable	Strongly temporalisable and symmetric
O2O-RTT	Linear-time solvable (trivial)	Linear-time solvable (trivial)
MRTT	Not approximable within a factor less than $\sqrt{n}/12$ (Theorem 16)	NP-hard (Theorem 18) and constant factor approximable (Theorem 19)
SS-MRTT	Not approximable within a factor less than $\sqrt{n}/12$ (Theorem 17)	Linear-time solvable (Fact 2)

Table 4.1: Our results assuming P $\neq$ NP (n denotes the number of nodes, $M = \sum_{T \in \mathbb{T}} |T|$ denotes the sum of number of arcs in each trip, and k denotes the number of trips that can be used in a temporal path). The approximability result is obtained by proving that we can get a high temporal reachability (that is, a temporal reachability proportional to the total number of pairs of nodes). A further result is Theorem 15, which intuitively states that the strong temporalisability property is not sufficient to get high reachability. The table leaves as the main open problem the question whether the MRTT and the SS-MRTT problems are approximable within a sub-linear factor, when restricted to strongly temporalisable collections of trips. Moreover, the table also leaves open whether or not it is possible to approximate MRAT, which was later proved possible within a constant factor approximation in [7].

We show that *the* MRTT *problem is NP-hard even if restricted to symmetric and strongly temporalisable collection of trips* (see Theorem 18). However, we also show that, *given a symmetric and strongly temporalisable collection of trips, it is possible to find in polynomial time a trip temporalisation achieving a reachability proportional to the total number of pairs* (see Theorem 19). This implies the existence of a constant-factor approximation algorithm in the symmetric and strongly temporalisable setting. It also gives ground to the classical design of public transit networks using symmetric lines.

We finally analyse the MRAT problem, namely the case of single arc trips, which correspond to assigning starting times to arcs of a digraph without the trip constraint. We prove that the MRAT problem is NP-hard, even when restricted to strongly connected digraphs. We will conclude by speculating whether the problem is approximable within a constant approximation ratio, and suggesting a graph theory conjecture which could be used to prove such approximation result for the MRAT problem. This conjecture has later been proved by Bessy, Thomassé and Viennot [7].

All our hardness results are proved starting from the 3-SAT problem, which is NP-complete [38]. Moreover, it is easy to show that the decision and maximisation problems we consider are in NP or in NPO (that is, the class of NP optimisations problems [2]), respectively. Indeed, given an instance of the problem and a trip temporalisation, checking that a node t is reachable from a node s in the induced temporal graph can be done in polynomial time.

4.1 Preliminary definitions and results

A *trip network* is a weighted multidigraph $D = (V, F)$ with positive weights (also called the *underlying multidigraph* of the trip network) along with a collection $\mathbb{T} = \{T_1, \ldots, T_{|\mathbb{T}|}\}$ of, not necessarily distinct, walks in D. The walks in $\mathbb{T}$ are also called *trips* to distinguish them from other arbitrary walks in D. In the following, without loss of generality, we will assume that any node in V and any arc in F appears in at least one trip in $\mathbb{T}$. Note that the disjoint union of the trips in $\mathbb{T}$ defines a weighted multidigraph $\mathcal{M}$changed notation that we call the *induced multidigraph* of $(D, \mathbb{T})$ (we assume that the arcs of $\mathcal{M}$ have an additional label specifying which trip they belong to, and which is represented in the figures by means of different line colors and styles). The *length* of a trip $l(T)$ is defined as the sum of the weights of all arcs of T.

Given a trip network $(D, \mathbb{T})$, a *temporalisation* σ of the trip network assigns a real number $\sigma(T)$ to each trip T in $\mathbb{T}$, indicating the starting time of T. Such a temporalisation induces a point availability temporal graph $G[D, \mathbb{T}, \sigma] = (V, E)$ defined as follows. For each $T = \langle e_1, \ldots, e_k \rangle$ in $\mathbb{T}$, with $e_i = (u_i, v_i, \lambda_i)$, E contains the temporal edges $(u_i, v_i, \sigma(T) + \sum_{j=1}^{i-1} \lambda_j, \lambda_i)$[1]: we say that these temporal edges are *induced* by T (with respect to the temporalisation σ). Notice that the assumption of positive weights of arcs in the underlying digraph translates into an assumption of positive travel times in the temporal graph. The set of nodes temporally reachable from a node u in a temporal graph G is denoted as $\mathcal{R}_G(u)$ (we assume that a node is reachable from itself, thus $u \in \mathcal{R}_G(u)$ for each node u). The *temporal reachability* of a node u in G is thus $|\mathcal{R}_G(u)|$. We define the *temporal reachability* of a temporal graph as the number of pairs of nodes that are temporally connected, namely the *temporal reachability* of G is defined as $\sum_{u \in V} |\mathcal{R}_G(u)|$. In this setting, the nodes reachable from a given node can be computed in $O(M)$, where M is the number of temporal edges in the temporal graph, for example using the earliest arrival time algorithm from [63]. This means the temporal reachability of a temporal graph can be computed in $O(nM)$. See Figure 4.1 for an example about the notion of temporal reachability. A node v is said to be σ-reachable from a node u if $v \in \mathcal{R}_{G[D,\mathbb{T},\sigma]}(u)$. The *$\sigma$-reachability* of a node u is the temporal reachability of u in $G[D, \mathbb{T}, \sigma]$, and the *σ-reachability* of the trip network is the temporal reachability of $G[D, \mathbb{T}, \sigma]$.

For example, let us consider the weighted directed multigraph D shown in the left part of Figure 4.2, and the following collection $\mathbb{T}$ of walks on D (depicted in the right part of the figure): $T_1 = (v_1, v_2, 1), (v_2, v_3, 2), (v_3, v_4, 2)$ (blue solid trip), $T_2 = (v_2, v_3, 1), (v_3, v_6, 1), (v_6, v_7, 1), (v_7, v_2, 1)$ (green dashed trip), and $T_3 = (v_5, v_6, 1), (v_6, v_7, 1), (v_7, v_8, 2)$ (red dotted trip). Note how the arc $(v_6, v_7, 1)$ is "used" by two different trips (that is, T_2 and T_3): this might correspond to two different vehicles travelling through this arc. Note also that there is no trip temporalisation such that both pairs of nodes v_1, v_8 and v_5, v_4 are temporally

[1] As it is standard, we assume that the summation with no summands evaluates to zero.

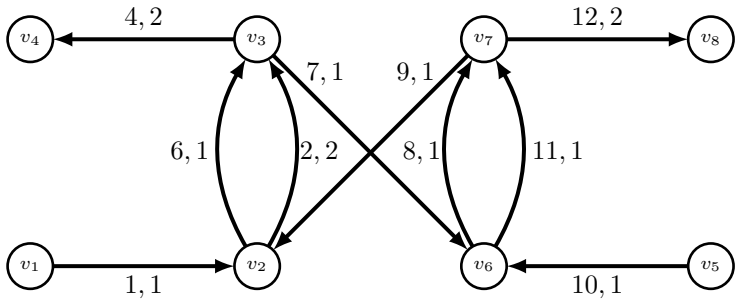

Figure 4.1: A point availability temporal graph. Node v_1 can reach v_8 at time 14 by following the temporal path $\langle(v_1, v_2, 1, 1), (v_2, v_3, 6, 1), (v_3, v_6, 7, 1), (v_6, v_7, 8, 1), (v_7, v_8, 12, 2)\rangle$. However, the pair v_5, v_4 is not temporally connected as v_5 cannot reach v_7 before time 12 while the only temporal edge crossing the cut $\{v_5, v_6, v_7, v_8\}$, $\{v_1, v_2, v_3, v_4\}$ is $(v_7, v_2, 9, 1)$ whose departure time is $9 < 12$. Indeed, node v_1 can reach all nodes but node v_5, while node v_7 can reach only nodes v_2, v_7, and v_8. The reachability of this temporal graph is 30.

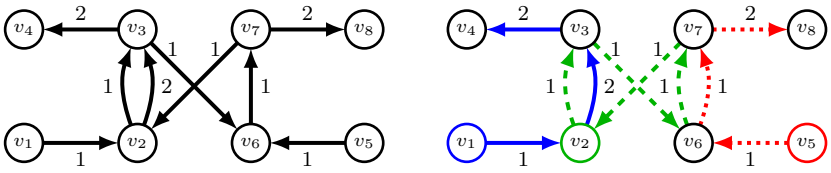

Figure 4.2: An example of a weighted directed multigraph D (left) where weights represent travel times of arcs, and a collection $\mathbb{T}$ of walks on D (right) called "trips". Each starting node of a trip has a colored border. The arc $(v_6, v_7, 1)$ of D is "used" by both the green dashed trip T_2 and the red dotted trip T_3. The length of the blue solid trip T_1 is the sum of the travel times of its arcs which amounts to 5, while the length of the other two trips is 4. No trip temporalisation exists such that both v_8 is reachable from v_1 and v_4 is reachable from v_5 in the induced temporal graph.

connected in the induced temporal graph. Indeed, if v_8 is reachable from v_1, then the starting time assigned to T_1 has to be smaller than the starting time assigned to T_3 as reaching v_7 from v_2 requires at least 3 units of time (using arcs of T_2), while if v_4 is reachable from v_5, then the starting time assigned to T_3 has to be smaller than the starting time assigned to T_1 as reaching v_3 from v_6 also requires at least 3 units of time: these two inequalities cannot be satisfied at the same time. Let us consider the trip temporalisation which assigns to T_1 the starting time 1, to T_2 the starting time 6, and to T_3 the starting time 10 (this trip temporalisation intuitively corresponds to scheduling the three trips one after the other). The temporal graph induced by this trip temporalisation is indeed shown in Figure 4.1 and its reachability is equal to 30 as mentioned in the caption. On the other hand, it is possible to verify that the trip temporalisation which assigns to T_1 the starting time 9, to T_2 the starting time 5, and to T_3 the starting time 1 induces a temporal graph whose reachability is 32 (see also Table 4.2 at page 104).

	T_1, T_2, T_3	T_1, T_3, T_2	T_2, T_1, T_3	T_2, T_3, T_1	T_3, T_1, T_2	T_3, T_2, T_1
v_1 :	$V \setminus \{v_5\}$	$V \setminus \{v_5, v_8\}$	$\{v_1, v_2, v_3, v_4\}$	$\{v_1, v_2, v_3, v_4\}$	$V \setminus \{v_5, v_8\}$	$\{v_1, v_2, v_3, v_4\}$
v_2 :	$V \setminus \{v_1, v_5\}$	$V \setminus \{v_1, v_5, v_8\}$	$V \setminus \{v_1, v_5\}$	$V \setminus \{v_1, v_5\}$	$V \setminus \{v_1, v_5, v_8\}$	$V \setminus \{v_1, v_5, v_8\}$
v_3 :	$V \setminus \{v_1, v_5\}$	$V \setminus \{v_1, v_5, v_8\}$	$V \setminus \{v_1, v_5\}$	$V \setminus \{v_1, v_5\}$	$V \setminus \{v_1, v_5, v_8\}$	$V \setminus \{v_1, v_5, v_8\}$
v_5 :	$\{v_5, v_6, v_7, v_8\}$	$V \setminus \{v_1, v_3, v_4\}$	$\{v_5, v_6, v_7, v_8\}$	$\{v_5, v_6, v_7, v_8\}$	$V \setminus \{v_1, v_3, v_4\}$	$V \setminus \{v_1\}$
v_6 :	$\{v_2, v_6, v_7, v_8\}$	$\{v_2, v_6, v_7, v_8\}$	$\{v_2, v_6, v_7, v_8\}$	$V \setminus \{v_1, v_5\}$	$\{v_2, v_6, v_7, v_8\}$	$V \setminus \{v_1, v_5\}$
v_7 :	$\{v_2, v_7, v_8\}$	$\{v_2, v_7, v_8\}$	$V \setminus \{v_1, v_5, v_6\}$	$V \setminus \{v_1, v_5, v_6\}$	$\{v_2, v_7, v_8\}$	$V \setminus \{v_1, v_6, v_5\}$
	30	28	29	31	28	32

Table 4.2: Possible schedules of the trip network of Figure 4.2. For each schedule S of trips one after another, and for each source node v, the corresponding cell shows the set of nodes S-reachable from v (the last row shows the value of the S-reachability). Note that in the underlying multidigraph of the trip network the number of pairs of nodes u and v such that v is reachable from u is equal to 38.

Our main optimisation problem is the following one.

> Maximum Reachability Trip Temporalisation (mrtt). Given a trip network $(D, \mathbb{T})$, find a temporalisation σ of the trip network which maximises its σ-reachability.

We will also study the restriction of the mrtt problem to the case in which, for each pair of nodes, there exists a temporalisation allowing to reach one from the other. More precisely, given a trip network $(D, \mathbb{T})$ and two nodes s and t, $(D, \mathbb{T})$ is said to be (s, t)-*temporalisable* if there exists a temporalisation σ of $(D, \mathbb{T})$ such that t is σ-reachable from s, and it is said to be *strongly temporalisable* if, for any two nodes s and t, $(D, \mathbb{T})$ is (s, t)-temporalisable. Moreover, we will also consider symmetric trip networks in the following sense. We say that a trip network $(D, \mathbb{T})$ is *symmetric* if all trips in $\mathbb{T}$ can be grouped into disjoint pairs (T, Υ) such that Υ is the reverse of T (T and Υ are two distinct trips in $\mathbb{T}$): Υ visits the same nodes as T, but in reverse order.

We will often refer to a particular kind of temporalisations. Given a trip network $(D, \mathbb{T})$, a *schedule* of the trip network is an ordering of the trips in $\mathbb{T}$. Note that a schedule S immediately induces a temporalisation σ_S of the trip network defined as follows. If $S = T_1, \ldots, T_{|\mathbb{T}|}$, then $\sigma_S(T_1) = 0$ and $\sigma_S(T_{i+1}) = \sum_{j=1}^{i} l(T_j)$, for $i \in [|\mathbb{T}| - 1]$. A node v is said to be S-reachable from a node u if it is σ_S-reachable. The *S-reachability* of the trip network is defined as its σ_S-reachability (see Table 4.2 where, for any possible schedule S, we indicate the S-reachability of the trip network shown in Figure 4.2).

Fact 1. *Let $(D, \mathbb{T})$ be a trip network and S be a schedule of $(D, \mathbb{T})$. Let C be a weighted multidigraph obtained starting from D by arbitrarily modifying only the weights of the arcs of D. The S-reachability of $(D, \mathbb{T})$ is equal to the S-reachability of $(C, \mathbb{T})$.*

Proof. The fact simply follows from the fact that a temporal path in $G[D, \mathbb{T}, \sigma_S]$ is also a temporal path in $G[C, \mathbb{T}, \sigma_S]$, since temporal edges from different trips cannot be interleaved inside a temporal path obtained through a schedule, where all arcs of a trip T are assigned smaller starting times than all arcs of the trips scheduled after T. □

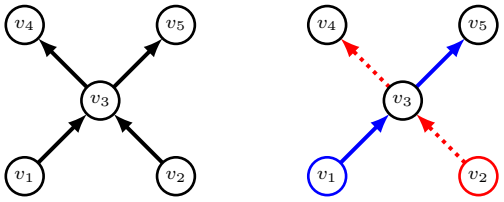

Figure 4.3: An example of a trip network $(D, \mathbb{T})$, where the underlying digraph D is depicted on the left (all arcs have weight 1, so that D is a simple digraph) and $\mathbb{T}$ (depicted in the induced multidigraph on the right) contains the trips $T_1 = \langle v_1, v_3, v_5 \rangle$ (blue solid trip) and $T_2 = \langle v_2, v_3, v_4 \rangle$ (red dotted trip), such that the maximum σ-reachability obtainable through a temporalisation is higher than the maximum S-reachability obtainable through a schedule. Indeed, the two possible schedules both achieve a reachability equal to 12, while a temporalisation that assigns the same starting time to T_1 and T_2 achieves a reachability equal to 13 (which is also the number of pairs of nodes u and v such that v is reachable from u in the underlying digraph).

For the sake of simplicity and without loss of generality, in the following we will present our results by referring to trip networks in which the weight of all arcs are equal to 1: indeed, as a consequence of Fact 1, *all* our results will apply to general trip networks as well (either because they are hardness results or because the lower bounds on the temporal reachability are obtained by referring to schedules). Under this assumption, a trip $T_i = \langle (u_0, u_1, 1), \ldots, (u_{k-1}, u_k, 1) \rangle$ will also be indicated as $T_i = \langle u_0, \ldots, u_k \rangle$. This means that the induced temporal graph will be a specific case of a point availability temporal graph, namely a uniform strict one. Note that, however, in general, the maximum σ-reachability obtainable through a temporalisation can be higher than the maximum S-reachability obtainable through a schedule (see, for example, Figure 4.3), and that the presence of weights can, in general, increase (or decrease) the maximum σ-reachability of a trip network (see, for example, Figure 4.4).

Finally, we could consider an extension of the model that takes into account footpaths, in the same spirit as in Chapter 3. The results of this chapter would still hold in this more general setting. On one hand, this would not affect the hardness results. On the other hand, the lower bounds on the temporal reachability are obtained using schedules. In this case, it would be sufficient to define temporalisations starting from these schedules. We need to shift the starting time of each trip after the arrival time of the previous one, by enough time to enable any transfer through footpaths.

4.2 The maximum reachability walk temporalisation problem

We first consider the following one-to-one version of the MRTT problem, called One-To-One Reachability Walk Temporalisation (in short, O2O-RTT): given a trip network $(D, \mathbb{T})$ and two nodes s and t, is $(D, \mathbb{T})$ (s, t)-temporalisable? Quite surprisingly, even this restricted version of the MRTT problem seems to be difficult to be solved in polynomial time.

Theorem 11. *The* O2O-RTT *problem is NP-complete.*

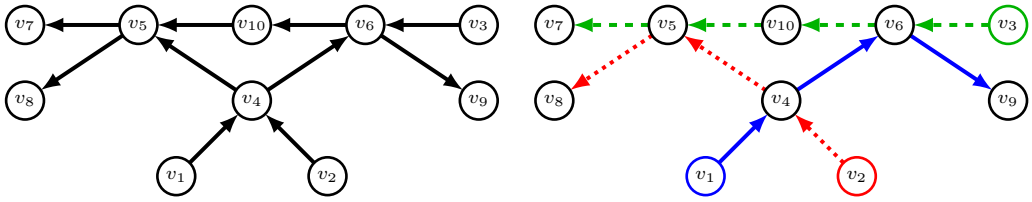

Figure 4.4: An example of a trip network $(D, \mathbb{T})$, where the underlying digraph D is depicted on the left and $\mathbb{T}$ contains the three trips (depicted on the right) T_1 (blue solid trip), T_2 (green dashed trip), and T_3 (red dotted trip), such that the presence of weights can increase the maximum σ-reachability obtainable through a temporalisation. Indeed, if all weights are equal to 1, no temporalisation σ can make the four nodes v_7, v_8, v_9, and v_{10} all σ-reachable from the three nodes v_1, v_2, and v_3: hence, for any temporalisation σ, the σ-reachability is less than the number R of pairs of nodes u and v such that v is reachable from u in D. On the contrary, if the arc from v_4 to v_5 has weight 3, then there exists a temporalisation whose reachability is equal to R (such a temporalisation assigns 1 to the trips T_1 and T_2, and 2 to T_3).

Proof. We reduce in polynomial time 3-SAT to O2O-RTT. We remind to Section 1.1.3 for a definition of the problem. Let us consider a 3-SAT formula Φ, with n variables $x_1, \dots, x_n$ and m clauses $c_1, \dots, c_m$. We first define the directed graph $D = (V, F)$ as the union of the following gadgets.

Intermediate and final nodes. V contains two nodes v_{n+1} and w_{m+1}.

Variable gadgets (see Figure 4.5(a)). For each variable x_i with $i \in [n]$, let p_i be the number of clauses that contains the literal x_i, and n_i the number of clauses that contains the literal $\neg x_i$ (without loss of generality, we may assume that both p_i and n_i are positive numbers). Then, V contains the following $p_i + n_i + 1$ nodes: $v_i, f_i^1, \dots, f_i^{p_i}, t_i^1, \dots, t_i^{n_i}$. Moreover, F contains the following p_i+n_i+2 directed arcs: (v_i, f_i^1), (f_i^h, f_i^{h+1}) for $h \in [p_i-1]$, $(f_i^{p_i}, v_{i+1})$, (v_i, t_i^1), (t_i^h, t_i^{h+1}) for $h \in [n_i - 1]$, and $(t_i^{n_i}, v_{i+1})$.

Clause gadgets (see Figure 4.5(b)). For each clause c_j with $j \in [m]$, V contains the following four nodes: w_j, l_j^1, l_j^2, l_j^3. Moreover, F contains the following six arcs: (w_j, l_j^h) and (l_j^h, w_{j+1}), for $h \in [3]$.

Variable-clause arc. F contains the arc (v_{n+1}, w_1).

Clause-variable arcs (see Figure 4.5(c)). For each clause c_j with $j \in [m]$, for each variable x_i with $i \in [n]$, for $h \in [3]$, and for $k \in [n_i]$, F contains the arc (l_j^h, t_i^k) if the h-th literal of c_j is $\neg x_i$ and c_j is the k-th clause in which the literal $\neg x_i$ occurs. Analogously, for each clause c_j with $j \in [m]$, for each variable x_i with $i \in [n]$, for $h \in [3]$, and for $k \in [p_i]$, F contains the arc (l_j^h, f_i^k) if the h-th literal of c_j is x_i and c_j is the k-th clause in which the literal x_i occurs.

We now define the trip collection $\mathbb{T}$ on D. For each clause c_j with $j \in [m]$ and for $h \in [3]$, $\mathbb{T}$ contains the trip $\langle w_j, l_j^h, t_i^k, o_i^k \rangle$, if $(l_j^h, t_i^k) \in F$ and o_i^k is defined as the unique out-neighbour of t_i^k (that is, $o_i^k = t_i^{k+1}$ if $k < n_i$, and $o_i^k = v_{i+1}$ if $k = n_i$), and the trip $\langle w_j, l_j^h, f_i^k, o_i^k \rangle$, if $(l_j^h, f_i^k) \in F$ and o_i^k is defined as the unique out-neighbour of f_i^k (that is, $o_i^k = f_i^{k+1}$ if $k < p_i$, and $o_i^k = v_{i+1}$ if $k = p_i$). Each of the other $2n+3m+1$ arcs, that are not

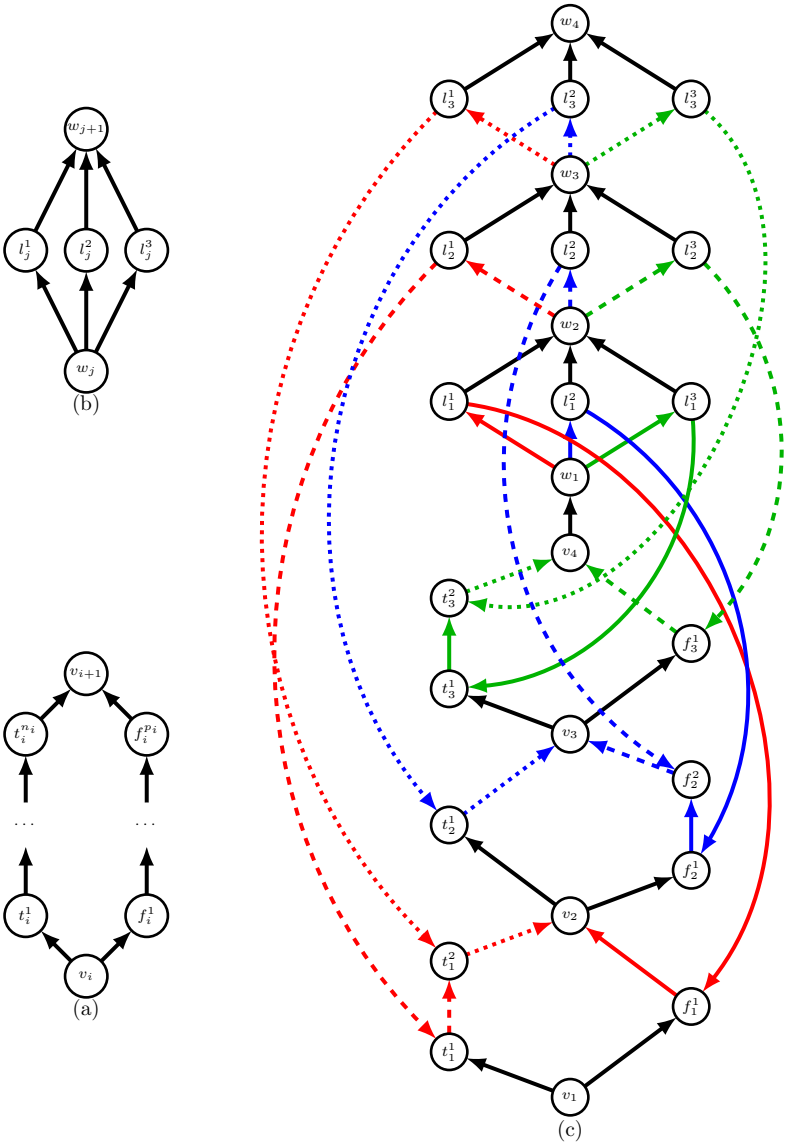

Figure 4.5: The reduction from 3-SAT to O2O-RTT. The variable gadget (a) corresponding to the variable x_i (p_i is the number of clauses that contains the literal x_i, while n_i is the number of clauses that contains the literal $\neg x_i$), the clause gadget (b) corresponding to the clause c_j, and the trip network (c) corresponding to the 3-SAT formula $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$ (the trips in red, blue, and green are the three trips corresponding to the first clause (solid arcs), to the second clause (dashed arcs), and to the third clause (dotted arcs), and each black arc forms a trip of length one).

yet included in a trip, forms a one-arc trip. Figure 4.5(c) shows an example of the reduction in the case of the Boolean formula $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$.

Let σ be a temporalisation of the trip network $(D, \mathbb{T})$ and let $G = G[D, \mathbb{T}, \sigma]$ be the temporal graph induced by σ. Note that, each arc in D belongs to exactly one trip in $\mathbb{T}$, which means that, for each arc $e \in F$, there is exactly one temporal arc in G with the same head and tail of e. Note also that, due to the topology of D, if $w_{m+1} \in \mathcal{R}_G(v_1)$, then the first part of the temporal path P from v_1 to w_{m+1} consists in moving from v_1 to v_{n+1} by passing, for each $i \in [n]$, through the node v_i and either through the nodes $t_i^1, \ldots, t_i^{n_i}$ or through the nodes $f_i^1, \ldots, f_i^{p_i}$. The second part of the temporal path P consists in moving from v_{n+1} to w_1 and, then, from w_1 to w_{m+1} by passing, for each $j \in [m]$, through the node w_j and exactly one l_j-node. Indeed, we can assume that P does not go back from a l_j-node to a variable node to which it is connected, since otherwise P should have to pass again through the arc (v_{n+1}, w_1), contradicting the fact that, as observed above, there is only one temporal edge in G corresponding to this arc. Moreover, if P uses a temporal edge with tail w_j and head l_j^h, for some $h \in [3]$, and if $(l_j^h, t_i^k) \in F$ (respectively, $(l_j^h, f_i^k) \in F$), for some $i \in [n]$ and $k \in [n_i]$ (respectively, $k \in [p_i]$), then P must have passed, in its first part, through the f_i-nodes (respectively, t_i-nodes) corresponding to the variable x_i. Otherwise, as P is a temporal path, the edge outgoing t_i^k (respectively, f_i^k) with head o_i^k would have a departure time smaller than the one assigned to (w_j, l_j^h), contradicting the fact that σ is a temporalisation of the trip $\langle w_j, l_j^h, t_i^k, o_i^k \rangle$ (respectively, $\langle w_j, l_j^h, f_i^k, o_i^k \rangle$).

Let us now prove that $(D, \mathbb{T})$ is (v_1, w_{m+1})-temporalisable if and only if there exists an assignment α to the variables that satisfies the Boolean formula Φ. Let us first suppose that $(D, \mathbb{T})$ is (v_1, w_{m+1})-temporalisable, that is, there exists a temporalisation σ of $(D, \mathbb{T})$ such that $w_{m+1} \in \mathcal{R}_G(v_1)$, where $G = G[D, \mathbb{T}, \sigma]$ is the temporal graph induced by σ. Let P be a temporal path from v_1 to w_{m+1} in G. For each variable x_i with $i \in [n]$, we set $\alpha(x_i) = \textsc{True}$ if and only if P passes through the t_i-nodes corresponding to x_i. We now prove that any clause c_j, with $j \in [m]$, is satisfied by α. Let l_j^h, for some $h \in [3]$, be the node which P goes to, when moving from w_j. If $(l_j^h, f_i^k) \in F$ (respectively, $(l_j^h, t_i^k) \in F$), for some $i \in [n]$ and $k \in [p_i]$ (respectively, $k \in [n_i]$), then the h-th literal of c_j is x_i (respectively, $\neg x_i$), and, because of the previous observations, P passes through the t_i-nodes (respectively, f_i-nodes) corresponding to x_i: this implies that $\alpha(x_i) = \textsc{True}$ (respectively, $\alpha(x_i) = \textsc{False}$) and, hence, that the clause c_j is satisfied.

Let us now suppose that there exists an assignment α to the variables that satisfies the formula Φ, and let us consider the following walk P in D, which starts from v_1 and arrives in w_{m+1}. The first part of P arrives at v_{n+1} and consists in moving from v_i to v_{i+1}, for $i \in [n]$, by passing through the t_i-nodes (respectively, f_i-nodes) corresponding to x_i, if $\alpha(x_i) = \textsc{True}$ (respectively, $\alpha(x_i) = \textsc{False}$). We know that, for each clause c_j with $j \in [m]$, at least one literal of c_j is satisfied: suppose that the first such literal is the h_j-th one, for some $h_j \in [3]$. Note that the choice of h_j implies that the first part of P does not use any arc of the trip T containing the arc $(c_j, l_j^{h_j})$: indeed, if the corresponding literal is x_i (respectively, $\neg x_i$), T goes through the f_i-nodes (respectively, t_i-nodes), while P goes through the t_i-nodes (respectively, f_i-nodes) as $\alpha(x_i) = \textsc{True}$ (respectively, $\alpha(x_i) = \textsc{False}$). The second part of P starts from v_{n+1}, moves to w_1, arrives at w_{m+1}, and consists in moving from w_j to w_{j+1}, for $j \in [m]$, by passing through the node $l_j^{h_j}$. Because of the definition of $\mathbb{T}$ and the choice of $h_1, \ldots, h_m$, each arc of P belongs to a different trip in $\mathbb{T}$. We can then consider a schedule S of $(D, \mathbb{T})$ in which the trips corresponding to the arcs in P are scheduled in the same order as they appear in P itself. The walk P thus induces a temporal path in $G[D, \mathbb{T}, \sigma_S]$ from v_1

to w_{m+1}. Thus, $(D, \mathbb{T})$ is (s,t)-temporalisable and the theorem has been proved. □

Note that, as a consequence of its proof, the above theorem holds even with the following restrictions: the in-degree and the out-degree of the nodes in D are bounded by 3, and $\mathbb{T}$ contains only *simple* trips (that is, trips that do not pass through the same node more than once), which are pairwise arc-disjoint.

We are now ready to prove the inapproximability result of the MRTT problem. Note that, given an instance $(D, \mathbb{T})$ with n nodes, any solution of the problem has a value greater than or equal to n (since we have assumed that any node is temporally reachable from itself). Since the number of pairs of nodes is n^2, the MRTT problem is trivially n-approximable. The following theorem states that a better approximation is not achievable, unless P = NP.

Theorem 12. *Unless* P = NP*, the* MRTT *problem is not $r(\cdot)$-approximable, for any $\epsilon \in (0,1)$ and for any non-decreasing function r in $O(n^{1-\epsilon})$, where n is the number of nodes.*

Proof. The proof makes use of the well-known gap technique (see Section 3.1.4 of [2]). Suppose by contradiction that there exists a $r(\cdot)$-approximation algorithm $\mathcal{A}$ for the MRTT problem, for some $\epsilon \in (0,1)$ and for some function $r(n)$ of the number n of nodes that satisfies $r(n) \leq cn^{1-\epsilon}$ for some constant c. We will now show that it is possible to exploit such an algorithm in order to solve in polynomial time the O2O-RTT problem, which would imply that P = NP (because of Theorem 11). Let us consider an instance $\langle (D = (V,F), \mathbb{T}), s, t \rangle$ of the O2O-RTT problem, where $V = \{s = v_1, \ldots, v_n = t\}$. Without loss of generality, we assume that $n > c + 1$. We define an instance $(D' = (V', F'), \mathbb{T}')$ of MRTT as follows.

- $V' = V \cup \{v_{n+i} : i \in [2K]\}$ with $K = \left\lceil (cn)^{1/\epsilon}(n+2)^{\frac{2-\epsilon}{\epsilon}} \right\rceil$.
- $F' = F \cup \{(v_{n+i}, s), (t, v_{n+K+i}) : i \in [K]\}\}$.
- $\mathbb{T}'$ is the union of $\mathbb{T}$ with all the one-arc trips corresponding to the arcs in $F' \setminus F$.

Consider an optimal temporalisation σ^* of $(D', \mathbb{T}')$: the maximum reachability is thus $\texttt{opt} = \sum_{u \in V} |\mathcal{R}_{G[D', \mathbb{T}', \sigma^*]}(u)|$. Moreover, let x be the value of the reachability achieved by the temporalisation computed by the approximation algorithm $\mathcal{A}$ with input $(D', \mathbb{T}')$: hence, $\frac{\texttt{opt}}{r(n')} \leq x \leq \texttt{opt}$ where $n' = n + 2K$.

Let us upper bound opt in the case in which $(D, \mathbb{T})$ is not (s,t)-temporalisable. To this aim, we upper bound the number of nodes σ'-reachable from each node in V', for any temporalisation σ' of $(D', \mathbb{T}')$ (in the following, $G' = G[D', \mathbb{T}', \sigma']$ is the temporal graph induced by σ').

- $|\mathcal{R}_{G'}(v_1)| \leq n - 1$ (since t is not σ'-reachable from $s = v_1$).
- For each $i \in [n-1]$, $|\mathcal{R}_{G'}(v_{i+1})| \leq n + K$ (since all nodes v_{n+i} with $i \in [K]$ have in-degree equal to zero in D').
- For each $i \in [K]$, $|\mathcal{R}_{G'}(v_{n+i})| \leq n$ (since t is not σ'-reachable from s).
- For each $i \in [K]$, $|\mathcal{R}_{G'}(v_{n+K+i})| = 1$ (since all these nodes have out-degree equal to zero in D').

Thus, if $(D, \mathbb{T})$ is not (s,t)-temporalisable, we have that $x \leq \texttt{opt} \leq (n-1)+(n-1)(n+K)+Kn+K = n^2+2nK-1$. On the other hand, if $(D, \mathbb{T})$ is (s,t)-temporalisable, then it is easy to produce a temporalisation σ' of $(D', \mathbb{T}')$ such that $v_{n+K+i} \in \mathcal{R}_{G[D',\mathbb{T}',\sigma']}(v_{n+j})$, for any $i, j \in [K]$. Hence, in this case we have that $x \geq \frac{\texttt{opt}}{r(n')} \geq \frac{K^2}{r(n')}$.

If we prove that $\frac{K^2}{r(n')} > n^2+2nK-1$, then we have that $x > n^2+2nK-1$ if and only if $(D, \mathbb{T})$ is (s,t)-temporalisable. This would imply that the O2O-RTT problem is solvable in polynomial time, and the theorem is proved. Let us then show that $\frac{K^2}{r(n')} > n^2+2nK-1$. Since

$$\frac{K^2}{r(n')} \geq \frac{K^2}{c(n+2K)^{1-\epsilon}} > \frac{K^2}{c(n+2)^{1-\epsilon}K^{1-\epsilon}} = \frac{K^{1+\epsilon}}{c(n+2)^{1-\epsilon}},$$

it is sufficient to prove that $K^{1+\epsilon} \geq c(n+2)^{1-\epsilon}n(n+2K)$. Since, by definition, $K \geq (cn)^{1/\epsilon}(n+2)^{\frac{2-\epsilon}{\epsilon}}$, that is, $K^{\epsilon} \geq cn(n+2)^{2-\epsilon}$, we have that $K^{1+\epsilon} \geq cnK(n+2)^{2-\epsilon} > cn(n+2K)(n+2)^{1-\epsilon}$, and the proof is completed. □

We can state a third hardness result (whose proof is given in the appendix), concerning an optimisation problem which is, somehow, in between the O2O-RTT problem and the MRTT problem, that is, the following SINGLE SOURCE MAXIMUM REACHABILITY WALK TEMPORALISATION (in short, SS-MRTT) problem: given a trip network $(D, \mathbb{T})$ and a node s, find a temporalisation σ of $(D, \mathbb{T})$ which maximises the σ-reachability of s. Note that, given an instance of this problem, any solution has a value greater than or equal to 1. Since the maximum number of nodes temporally reachable from s is n, we have that the SS-MRTT problem is trivially n-approximable. The following theorem states that a better approximation is not achievable, unless P = NP.

Theorem 13. *Unless* P = NP*, the* SS-MRTT *problem is not $r(\cdot)$-approximable, for any $\epsilon \in (0,1)$ and for any non-decreasing function r in $O(n^{1-\epsilon})$, where n is the number of nodes.*

Proof. Similarly to the proof of Theorem 12, we will make use of the gap technique. Suppose by contradiction that there exists a $r(\cdot)$-approximation algorithm $\mathcal{A}$ for the SS-MRTT problem, for some $\epsilon \in (0,1)$ and for some function $r(n)$ of the number n of nodes that satisfies $r(n) \leq cn^{1-\epsilon}$ for some constant c. We will now show that it is possible to exploit such an algorithm in order to solve in polynomial time the O2O-RTT problem, which would imply that P = NP (because of Theorem 11). Let us consider an instance $\langle (D = (V,F), \mathbb{T}), s, t \rangle$ of the O2O-RTT problem, where $V = \{s = v_1, \ldots, v_n = t\}$. Without loss of generality, we assume that $n > c+1$. We define an instance $(D' = (V', F'), \mathbb{T}')$ of SS-MRTT as follows.

- $V' = V \cup \{v_{n+i} : i \in [K]\}$ with $K = \lceil cn^{2/\epsilon} \rceil$.
- $F' = F \cup \{(t, v_{n+i}) : i \in [K]\}$.

The trip collection $\mathbb{T}'$ is then defined as the union of $\mathbb{T}$ with all the one-arc trips corresponding to the arcs in $F' \setminus F$.

Consider an optimal temporalisation σ^* of $(D', \mathbb{T}')$: the maximum reachability of s is thus $\texttt{opt} = |\mathcal{R}_{G[D',\mathbb{T}',\sigma^*]}(s)|$. Moreover, let x be the value of the temporalisation computed by the approximation algorithm $\mathcal{A}$: hence, $\frac{\texttt{opt}}{r(n')} \leq x \leq \texttt{opt}$ where $n' = n+K$.

If $(D, \mathbb{T})$ is (s,t)-temporalisable, then $\texttt{opt} \geq K+2$. Indeed, the very same temporalisation can be extended to $(D', \mathbb{T}')$, by assigning to all the new one-arc trips a starting time greater than the arrival time in t, so that all the K new out-neighbours of t are reachable. Note that $\texttt{opt} \geq K+2$ implies that $x \geq \frac{\texttt{opt}}{r(n')} \geq \frac{K+2}{r(n')} \geq \frac{cn^{2/\epsilon}}{r(n')}$. Since $n' = n+K$, we have that $n' < n + cn^{2/\epsilon} + 1 < n^{1+2/\epsilon}$ (since $n > c+1$ and $\epsilon \in (0,1)$). Since r is non-decreasing, we get $x \geq \frac{cn^{2/\epsilon}}{r(n')} > \frac{cn^{2/\epsilon}}{r(n^{1+2/\epsilon})} \geq \frac{cn^{2/\epsilon}}{c(n^{1+2/\epsilon})^{1-\epsilon}} = n^{1+\epsilon} > n$. Hence, if $(D, \mathbb{T})$ is (s,t)-temporalisable, then $x > n$. On the other hand, if $(D, \mathbb{T})$ is not (s,t)-temporalisable, then $\texttt{opt} < n$, since none of the new out-neighbors of t are σ'-reachable from s (without passing through t), for any temporalisation σ' of $(D', \mathbb{T}')$. Hence, $x \leq \texttt{opt} < n$.

We can conclude that $(D, \mathbb{T})$ is (s,t)-temporalisable if and only if the value x of the solution computed by the approximation algorithm $\mathcal{A}$ is greater than n. This implies that the O2O-RTT problem is solvable in polynomial time, and the theorem is proved. □

4.2.1 Bounding the number of used trips

In this section, we study the O2O-RTT problem parameterised by the number of trips needed to go from the source to the destination.

Given a trip network $(D, \mathbb{T})$, a temporalisation σ and $k \in \mathbb{N}$, a node v is said to be (k,σ)-*reachable* from u if there exists a temporal path P in $G = G[D, \mathbb{T}, \sigma]$ from u to v which is composed of edges induced by at most k different trips in $\mathbb{T}$: let $\mathbb{P}$ be the set of such trips. Note that, without loss of generality, we can suppose that the edges induced by the same trip are contiguous in P. Indeed, if $P = e_1, \ldots, e_p$, let $T_1 \in \mathbb{P}$ be one of the trips that induces e_1 and let e_h be the last edge in P which is induced by T_1. The temporalisation of T_1 induces a temporal path $e'_1 = e_1, e'_2, \ldots, e'_{l-1}, e'_l = e_h$ in G. We can then consider the temporal path $P' = e'_1, \ldots, e'_l, e_{h+1}, \ldots, e_p$, which has the property that all the edges which are induced by T_1 are contiguous. We can now apply the same argument by considering the trip $T_2 \in \mathbb{P}$ as one of the trips that induces e_{h+1}, and go on like this until all the edges induced by each trip considered are contiguous in the final temporal path from u to v.

We now consider the O2O-RTT problem parameterised by the number k of trips used in the resulting temporal graph, in order to go from s to t. More precisely, given a trip network $(D, \mathbb{T})$, a source node s, and a target node t, the parameterised problem k-O2O-RTT consists in deciding whether there exists a temporalisation σ such that v is (k,σ)-reachable from u. By using the color coding technique developed in [1], we can obtain the following result.

Theorem 14. *The k-O2O-RTT problem can be solved in $2^{O(k)} M \log |\mathbb{T}|$ time where $M = \sum_{T \in \mathbb{T}} |T|$ is the number of arcs in the induced multidigraph $\mathcal{M}$.*

Proof. Let us consider a trip network $(D = (V, F), \mathbb{T})$, a source node s and a target node t, and let $\mathcal{M}$ be the induced multidigraph of $(D, \mathbb{T})$. Note that $M = \sum_{T \in \mathbb{T}} |T|$ is also the number of arcs in $\mathcal{M}$. We suppose that for each trip T, we are given the list of its arcs in their respective order in T. We also let $V(T)$ denote the set of nodes appearing in T and write $u \prec_T v$ when $u, v \in V(T)$ and u precedes v in T. Let $\chi : \mathbb{T} \to [k]$ be any color assignment to the trips in $\mathbb{T}$. For $i \in [k]$, a (i,χ)-*path* P in $\mathcal{M}$ is a path which is the concatenation of exactly i subtrips of distinct trips in $\mathbb{T}$ with pairwise distinct colors (in the following, $\chi(P)$ will denote the set of colors "used" by such a path P). Note that the existence of a (i,χ)-path in $\mathcal{M}$ from s to t with $i \in [k]$ implies the existence of a schedule S such that t is (k, σ_S)-reachable from s: indeed, we can first schedule the i trips of the path in the order they appear in it, and then the remaining trips of $\mathbb{T}$ in any order. In order

to test the existence of a (i,χ)-*path* in $\mathcal{M}$ from s to t for $i \in [k]$, we can use the dynamic programming technique [4]. For any node $v \in V$, let $A_\chi[v,i]$ denote the collection of sets C of colors for which there exists a (i,χ)-path P in $\mathcal{M}$ from s to v such that $\chi(P) = C$. Clearly, there exists a (i,χ)-*path* in $\mathcal{M}$ from s to t if and only if $A_\chi[t,i] \neq \emptyset$. We have that, for any node $v \in V$,

$$A_\chi[v,1] = \{\{\chi(T)\} : T \in \mathbb{T} \wedge s \prec_T v\}.$$

Moreover, for any $i \in [k-1]$,

$$\begin{aligned} A_\chi[v,i+1] &= \bigcup_{T \in \mathbb{T}: v \in V(\mathbb{T})} \{C \cup \{\chi(T)\} : u \prec_T v \wedge C \in A_\chi[u,i] \wedge \chi(T) \notin C\} \\ &= \bigcup_{T \in \mathbb{T}: v \in V(\mathbb{T})} \{C \cup \{\chi(T)\} : C \in \mathcal{C}(T,v) \wedge \chi(T) \notin C\} \end{aligned}$$

where $\mathcal{C}(T,v) = \bigcup_{u \prec_T v} A_\chi[u,i]$. We can compute $A_\chi[v,1]$ for all $v \in V$ by scanning all trips that contain s. These trips can be obtained in $O(M)$ time by scanning all trips in $\mathbb{T}$. Moreover, we can execute the above update rule for all $v \in V$ by scanning once each trip $T \in \mathbb{T}$ as follows. We first set $A_\chi[v,i+1] := \emptyset$ for all $v \in V$. Then, for each trip T, we iterate over the arcs of T in their respective order in T. For each arc (u,v) of T, we compute $\mathcal{C}(T,v) = \mathcal{C}(T,u) \cup A_\chi[u,i]$ and update $A_\chi[v,i+1] := A_\chi[v,i+1] \cup \{C \cup \{\chi(T)\} : C \in \mathcal{C}(T,v) \wedge \chi(T) \notin C\}$ (if (u,v) is the first arc of T we simply use $\mathcal{C}(T,v) = A_\chi[u,i]$ as u is then the only node preceding v in T). Note that both the computation of $\mathcal{C}(T,v)$ and the update of $A_\chi[v,i+1]$ take $O(2^k)$ time since, for any node $u \in V$ and for $j \in [k]$, $|A_\chi[u,j]| \leq 2^k$. Each update step is thus performed in $O(2^k M)$ time and the whole computation requires $2^{O(k)}M$ time.

Observe now that if there exists a temporalisation of $(D,\mathbb{T})$ such that t is (k,σ)-reachable from s, then there must exist a color assignment χ such that $\mathcal{M}$ includes a (i,χ)-path from s to t for some $i \in [k]$. In order to find such a path, we can use an appropriate set of perfect hash functions from $[|\mathbb{T}|]$ to $[k]$. Indeed, it is possible to design $2^{O(k)} \log |\mathbb{T}|$ hash functions such that any subset of k trips has image $\{1,\ldots,k\}$ for at least one function [55]. This implies that any subset of i trips has i pairwise distinct colors as image for at least one function as such a set can be completed in a set of k trips. Each hash function can be coded with $O(k + \log\log|\mathbb{T}|)$ bits, and can be generated in $O(k^3 \log|\mathbb{T}|)$ time [36]. The number of such functions is $O(2^k \log|\mathbb{T}|)$ and their computation takes $2^{O(k)} \log^2|\mathbb{T}|$ time. As they can be accessed in $O(1)$ time, testing the coloring obtained through each function yields a $2^{O(k)} M \log|\mathbb{T}|$-time algorithm for solving the k-O2O-RTT problem (we use $\log|\mathbb{T}| = O(|\mathbb{T}|)$ and $|\mathbb{T}| \leq M$). The theorem is thus proved.

□

4.3 Strongly temporalisable trip networks

We now switch to strongly temporalisable trip networks where one-to-one reachability is assumed for all pairs of nodes. This clearly implies that the O2O-RTT problem is trivially solvable when restricted to strongly temporalisable trip networks, since the answer is always yes (actually, one-to-one reachability is always satisfied under strong temporalisability). On the other hand, we will prove that both the MRTT and the SS-MRTT problem cannot be approximated within a factor less than $\frac{\sqrt{n}}{12}$ (unless P = NP). To this aim, we first

show that the strong temporalisability by itself is not enough to ensure the existence of a temporalisation σ with a σ-reachability which is a constant fraction of all pairs of nodes.

Theorem 15. *For any $r > 3$, there exists a strongly temporalisable trip network $(D_r, \mathbb{T}_r)$ with $n = r^2 + 2r$ nodes, such that any temporalisation σ of $(D_r, \mathbb{T}_r)$ has σ-reachability $O(n^{\frac{3}{2}})$.*

Proof. We first define the trip network $(D_r, \mathbb{T}_r)$ through the gadgets that compose it (see Figure 4.6). We then prove that the trip network is strongly temporalisable and, finally, we prove that, for any temporalisation σ, the σ-reachability is $O(n^{\frac{3}{2}})$.

Upper gadget $V^U, F^U, \mathbb{T}^U$. The set V^U contains the nodes $c_1, \ldots, c_r$, and the nodes $u_1, \ldots, u_r$. These nodes are connected through the following set of directed arcs:

$$F^U = \{(c_{i+1}, c_i) : i \in [r-1]\} \cup \{(c_i, c_{i+2}) : i \in [r-2]\} \cup \{(c_1, u_i), (u_i, c_2) : i \in [r]\}.$$

On this gadget, we have the following collection of trips: $\mathbb{T}^U = \{T_i^U : i \in [r]\}$, where

$$T_i^U = \langle c_1, u_i, c_2, c_1, c_3, c_2, \ldots, c_{r-1}, c_{r-2}, c_r, c_{r-1} \rangle$$

(each arc (c_{i+1}, c_i) is followed by (c_i, c_{i+2}), see also Figure 4.6, where the upper red solid trip is T_3^U).

Lower gadget $V^L, F^L, \mathbb{T}^L$. The set V^L contains the nodes $c_{r+1}, \ldots, c_{2r}$, and the nodes $l_1, \ldots, l_r$. These nodes are connected through the following set of directed arcs:

$$\begin{aligned} F^L \quad = \quad & \{(c_{r+i+1}, c_{r+i}) : i \in [r-1]\} \cup \{(c_{r+i}, c_{r+i+2}) : i \in [r-2]\} \\ & \cup \{(c_{2r-1}, l_i), (l_i, c_{2r}) : i \in [r]\}. \end{aligned}$$

On this gadget, we have the following collection of trips: $\mathbb{T}^L = \{T_i^L : i \in [r]\}$, where

$$T_i^L = \langle c_{r+2}, c_{r+1}, c_{r+3}, c_{r+2}, \ldots, c_{2r-2}, c_{2r}, c_{2r-1}, l_i, c_{2r} \rangle$$

(each arc (c_{r+i+1}, c_{r+i}) is followed by (c_{r+i}, c_{r+i+2}), see also Figure 4.6, where the lower red solid trip is T_3^L).

Descending gadgets $V_i^{\downarrow}, F_i^{\downarrow}, \mathbb{T}^{\downarrow}$. For any i with $i \in [r]$, we refer to node u_i and l_i as d_i^1 and d_i^r, respectively. The set $V_i^{\downarrow}$ contains the nodes $d_i^2, \ldots, d_i^{r-1}$. These nodes are connected among them and to the previous gadgets through the following set of directed arcs:

$$F_i^{\downarrow} = \{(d_i^j, d_i^{j+1}) : j \in [r-1]\}.$$

On this gadget, we have the following collection of trips: $\mathbb{T}^{\downarrow} = \{T_i^{\downarrow \mathrm{l}}, T_i^{\downarrow \mathrm{r}} : i \in [r]\}$, where

$$T_i^{\downarrow \mathrm{l}} = T_i^{\downarrow \mathrm{r}} = \langle u_i = d_i^1, d_i^2, \ldots, d_i^{r-1}, d_i^r = l_i \rangle$$

(see Figure 4.6, where the blue dashed trip is $T_3^{\downarrow \mathrm{l}}$ and the violet dotted trip is $T_3^{\downarrow \mathrm{r}}$).

Ascending gadget $e^{\uparrow}$, $T^{\uparrow}$. This gadget contains the arc $e^{\uparrow} = (c_{r+1}, c_r)$, which connects the lower gadget to the upper gadget, and is also a one-arc trip $T^{\uparrow}$. Note that this gadget does not introduce any new nodes.

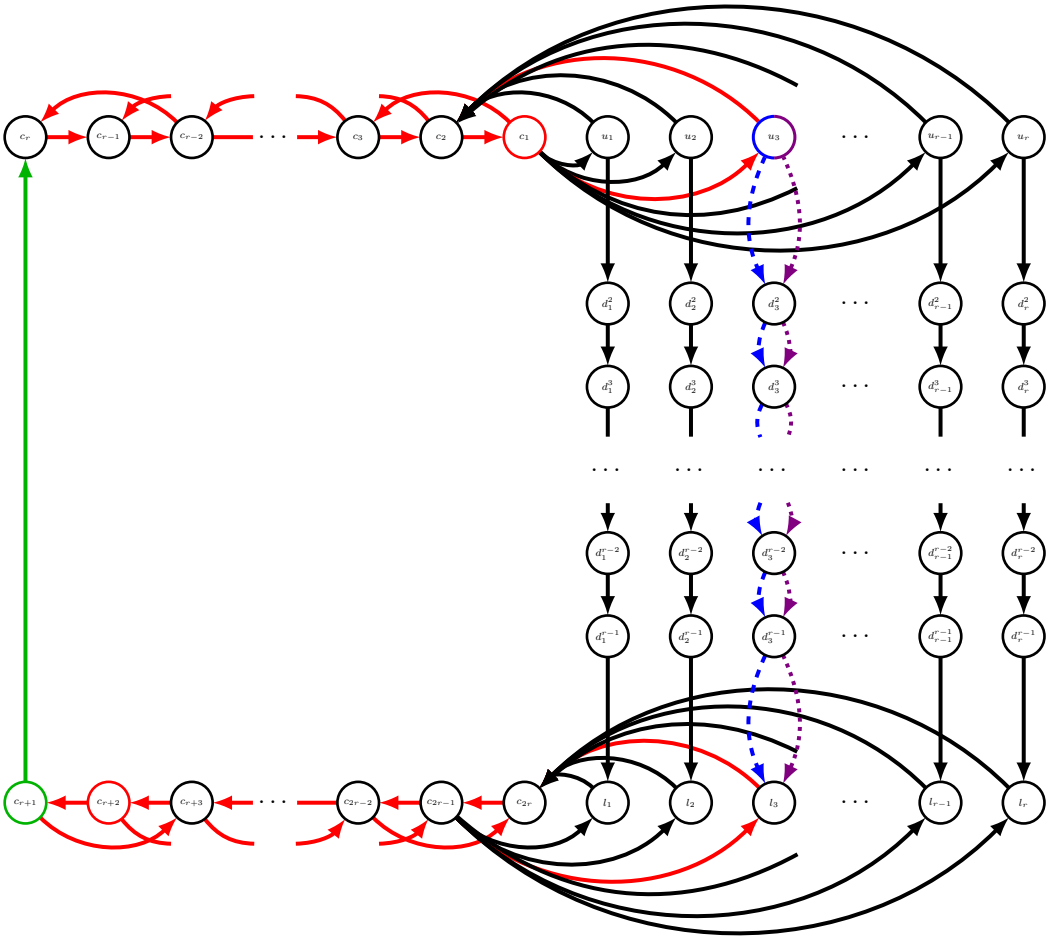

Figure 4.6: An example of a strongly temporalisable trip network, such that any temporalisation cannot connect a constant fraction of the total pairs of nodes, obtained via the construction described in the proof of Theorem 15. The two red solid trips, starting from the two red nodes, correspond to the trip T_3^U and T_3^L, respectively, in the construction (the first time a trip passes through a node with more than one red outgoing arc, it continues towards the node with the smaller index). The blue dashed (respectively, violet dotted) trip, starting from the half blue (respectively, violet) node, corresponds to the trip $T_3^{\downarrow \mathrm{l}}$ (respectively, $T_3^{\downarrow \mathrm{r}}$) in the construction. Finally, the green solid trip, starting from the green node, corresponds to the trip $T^{\uparrow}$ in the construction.

To conclude the definition of the network, we set $D_r = (V, F)$ where

$$V = V^U \cup V^L \cup \bigcup_{i=1}^{r} V_i^{\downarrow}$$

(note that $|V| = 2r + 2r + r(r-2) = r^2 + 2r$) and

$$F = F^U \cup F^L \cup \bigcup_{i=1}^{r} F_i^{\downarrow} \cup \{e^{\uparrow}\},$$

and we set

$$\mathbb{T}_r = \mathbb{T}^U \cup \mathbb{T}^L \cup \mathbb{T}^{\downarrow} \cup \{T^{\uparrow}\}.$$

Note that the descending gadgets contain the majority of the nodes in the trip network, and that it is possible to visit them by entering from the upper nodes and by travelling all the way down (the second descending trip $T_i^{\downarrow \mathrm{r}}$ is necessary in order to ensure that the trip network is strongly temporalisable).

$(D_r, \mathbb{T}_r)$ **is strongly temporalisable.** In order to ease the reading of the proof let us first define the following r (partial) schedules of the trip networks $(D_r, \mathbb{T}^U)$ and $(D_r, \mathbb{T}^L)$, respectively.

- Schedule S_i^U for $i \in [r]$. This schedule has the purpose of making u_i S_i^U-reachable from c_r. The schedule S_i^U consists in having the trips $\mathbb{T}^U$ scheduled in any order that has T_i^U as the last one. Starting from c_r, it is possible to go to c_{r-1} through the arc (c_r, c_{r-1}) of the first trip scheduled, to c_{r-2} through the arc (c_{r-1}, c_{r-2}) of the second trip scheduled, and so on. In this way, it is possible to reach c_1 using the first $r-1$ trips scheduled. Finally, from c_1 it is possible to go to u_i through the arc (c_1, u_i) of the trip T_i^U. Note that this also shows that, for any $h, k \in [r]$ with $h > k$, c_k is S_i^U-reachable from c_h.
- Schedule S_i^L for $i \in [r]$. This schedule is similar to the previous one, but applied to the lower gadget. It allows node c_{r+1} to be S_i^L-reachable from l_i. The schedule S_i^L consists in having the trips $\mathbb{T}^L$ scheduled in any order that has T_i^L as the first one. Starting from l_i, it is possible to go to c_{2r} through the arc (l_i, c_{2r}) of the trip T_i^L. At this point, it is possible to reach c_{r+1} from c_{2r} by using one arc of each of the remaining $r-1$ trips. Note that this also shows that, for any $h, k \in [r]$ with $h > k$, c_{k+r} is S_i^L-reachable from c_{h+r}.

By using the (partial) schedules above, we can now easily show that, for any two nodes u and v in V, there exists a schedule S such that v is S-reachable from u. These schedules are specified in Table 4.3 for each possible pair of nodes. For example, in order to reach $d_h^{l_2} \in V^{\downarrow}$ from $d_h^{l_1} \in V^{\downarrow}$ with $l_2 < l_1$, we first schedule the trip $T_h^{\downarrow \mathrm{l}}$ in order to reach $l_h \in V^L$, we then use the (partial) schedule S_h^L in order to reach c_{r+1}, we then schedule the trip $T^{\uparrow}$ in order to reach c_r, we then use the (partial) schedule S_h^U in order to reach u_h, and we finally schedule the trip $T_h^{\downarrow \mathrm{r}}$ to reach $d_h^{l_2}$ (note how, in this case, we need the second descending trip).

Any temporalisation σ has $O(n\sqrt{n})$ σ-reachability. Given any temporalisation σ of the trip network $(D_r, \mathbb{T}_r)$, let $T_{i_{\min}}^L$ be one of the trips with minimum starting time according to

Source	Destination				
	$c_k \in V^U$	$u_k \in V^U$	$c_k \in V^L$	$l_k \in V^L$	$d_k^{l_2} \in V_k^{\downarrow\downarrow}$
$c_h \in V^U$	S_1^U if $k < h$ T_1^U if $k > h$	S_k^U	$S_1^U, T_1^{\downarrow\downarrow}, S_1^L$	$S_k^U, T_k^{\downarrow\downarrow}$	$S_k^U, T_k^{\downarrow\downarrow}$
$u_h \in V^U$	T_h^U	T_h^U, T_k^U	$T_h^{\downarrow\downarrow}, S_h^L$	$T_k^{\downarrow\downarrow}$ if $k = h$ $T_h^U, T_k^U, T_k^{\downarrow\downarrow}$ if $k \neq h$	$T_k^{\downarrow\downarrow}$ if $k = h$ $T_h^U, T_k^U, T_k^{\downarrow\downarrow}$ if $k \neq h$
$c_h \in V^L$	$S_1^L, T^{\uparrow}, S_1^U$	$S_1^L, T^{\uparrow}, S_k^U$	S_1^U if $k < h$ T_1^L if $k > h$	T_k^L	$S_h^L, T^{\uparrow}, S_k^U, T_k^{\downarrow\downarrow}$
$l_h \in V^L$	$S_h^U, T^{\uparrow}, S_k^U$	$S_h^U, T^{\uparrow}, S_k^U$	S_h^L	T_h^L, T_k^L	$S_h^L, T^{\uparrow}, S_k^U, T_k^{\downarrow\downarrow}$
$d_h^{l_1} \in V_h^{\downarrow\downarrow}$	$T_h^{\downarrow\downarrow}, S_h^L, T^{\uparrow}, S_k^U$	$T_h^{\downarrow\downarrow}, S_h^L, T^{\uparrow}, S_k^U$	$T_h^{\downarrow\downarrow}, S_h^L$	$T_h^{\downarrow\downarrow}$ if $k = h$ $T_h^{\downarrow\downarrow}, T_h^L, T_k^L$ if $k \neq h$	$T_h^{\downarrow\downarrow}, S_h^L, T^{\uparrow}, S_k^U, T_k^{\downarrow\downarrow}$ if $k \neq h$ $T_h^{\downarrow\downarrow}$ if $k = h \wedge l_2 > l_1$ $T_h^{\downarrow\downarrow}, S_h^L, T^{\uparrow}, S_k^U, T_k^{\downarrow r}$ if $k = h \wedge l_2 < l_1$

Table 4.3: The different cases in the proof that the trip network $(D_r, \mathbb{T}_r)$, defined in the proof of Theorem 15 and illustrated in Figure 4.6, is strongly temporalisable. For each node u labeling the row and for each node v labeling the column, the corresponding cell specifies which (partial) schedule S has to be used in order to guarantee that v is S-reachable from u (the trips that do not appear can be scheduled in any order).

σ among all the trips in the lower gadget, and let $T^U_{i_{\max}}$ be one of the trips with maximum starting time according to σ among all the trips in the upper gadget. We will prove the following claim.

Claim 5. *For any pair of nodes $(d^{l_1}_{h_1}, d^{l_2}_{h_2})$ with $1 < l_1, l_2 < r$, $h_1, h_2 \in [r]$, $h_1 \neq h_2$, and $h_1 \neq i_{\min} \vee h_2 \neq i_{\max}$, $d^{l_2}_{h_2}$ is not σ-reachable from $d^{l_1}_{h_1}$.*

Note that the number of pairs of nodes satisfying the conditions in the claim is at least $(r-1)(r-2)^3 > (r-1)(r^3 - 6r^2) = r^4 - 7r^3 + 6r^2 > r^4 - 7r^3$, thus implying that, since $n = r^2 + 2r$, the σ-reachability is at most $(r^2+2r)^2 - (r^4 - 7r^3) = r^4 + 4r^3 + 4r^2 - r^4 + 7r^3 = 11r^3 + 4r^2 < 15r^3$. Since $n = r^2 + 2r$, we have that $r < \sqrt{n}$, and that the σ-reachability is at most $15n\sqrt{n}$. The theorem thus follows.

It thus remains to prove the claim. To this aim, let $(d^{l_1}_{h_1}, d^{l_2}_{h_2})$ be such that $1 < l_1, l_2 < r$, $h_1, h_2 \in [r]$, $h_1 \neq h_2$, and $h_1 \neq i_{\min} \vee h_2 \neq i_{\max}$. Note that, since $h_1 \neq h_2$, each walk in D_r from $d^{l_1}_{h_1}$ to $d^{l_2}_{h_2}$ has to pass through the nodes l_{h_1}, c_{r+1}, c_r, and u_{h_2} in this order. Note also that any walk from l_{h_1} to c_{r+1} contains at least r arcs. Since travelling on more than one arc of the same trip in the lower gadget results in going back towards l_{h_1}, all the r trips in $\mathbb{T}^L$ have to be used in order to go from l_{h_1} to c_{r+1}. As in any temporal path from l_{h_1} to c_{r+1} in $G[D_r, \mathbb{T}_r, \sigma]$, the starting times of the temporal edges must increase, this implies that all starting times of trips in $\mathbb{T}^L$ must be pairwise distinct and that the trip with the earliest starting time is $T^L_{h_1}$ as it is the only one containing the arc (l_{h_1}, c_{2r}). If $h_1 \neq i_{min}$, then σ fails to make c_{r+1} σ-reachable from l_{h_1}. If $h_2 \neq i_{max}$, a similar reasoning allows us to show that σ fails to make u_{h_2} reachable from c_r by considering a temporal path from c_r to u_{h_2} in $G[D_r, \mathbb{T}_r, \sigma]$, and the trips in $\mathbb{T}^U$. Hence, $h_1 \neq i_{\min} \vee h_2 \neq i_{\max}$ and $h_1 \neq h_2$ implies that σ fails to make $d^{l_2}_{h_2}$ σ-reachable from $d^{l_1}_{h_1}$, and the claim is proved. □

The construction of the trip network $(D_r, \mathbb{T}_r)$ in the proof of the above theorem can be adapted in order to prove the following inapproximability results for both the MRTT and the SS-MRTT problem, in the case of strongly temporalisable trip networks.

Theorem 16. *Unless $P = NP$, the* MRTT *problem cannot be approximated within a factor less than $\frac{\sqrt{n}}{12}$ even if the input trip network is strongly temporalisable.*

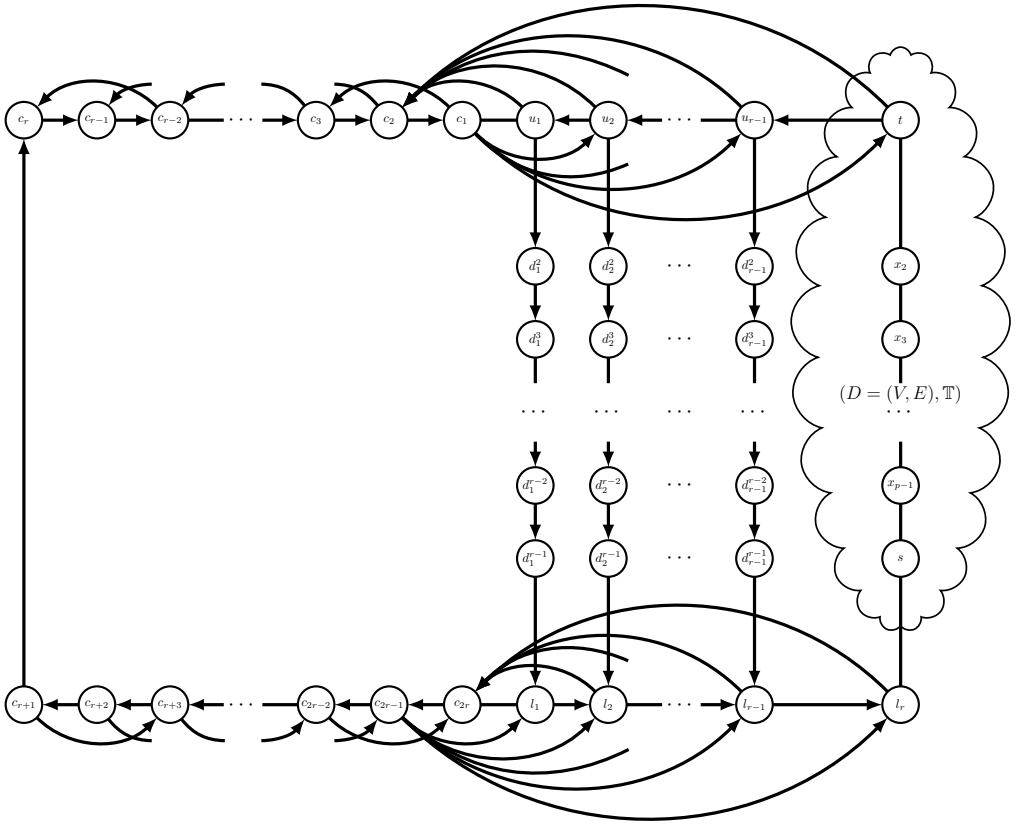

Figure 4.7: The reduction from O2O-RTT to MRTT used in the proof of Theorem 16 (the arcs in D are not shown, unless they coincide with one of the shown arcs). All the arcs without arrows are present in both directions (for instance, both (t, x_2) and (x_2, t) are included in the set of arcs, while only (t, u_{r-1}) is included in the set of arcs).

Proof. As in the proof of Theorem 12, we use the gap technique by reducing in polynomial time the O2O-RTT decision problem to the MRTT problem. Consider an instance $\langle (D = (V, F), \mathbb{T}), s, t \rangle$ of the O2O-RTT problem, where $V = \{t = x_1, \ldots, x_p = s\}$ (without loss of generality, we assume that $p > 22$). We then define a trip network $(D' = (V', F'), \mathbb{T}')$ as follows (see Figure 4.7). Let $(D_r = (V_r, F_r), \mathbb{T}_r)$, with $r = p + 1$, be the trip network constructed in the proof of Theorem 15 (note that $r > 23$): in the following, we identify each node $x_i \in V$ with the node d_r^i of the last descending gadget of D_r (that is, we consider $V = \{d_r^1 = u_r = t, d_r^2 = x_2, \ldots, d_r^{r-2} = x_{r-2} = x_{p-1}, d_r^{r-1} = s\}$). Note that $l_r = d_r^r$ is not a node in V.. We then set $V' = V_r$ and

$$\begin{aligned} F' \quad = \quad & F_r \cup \{(d_r^i, d_r^j) : (x_i, x_j) \in F \cup \{(d_r^{i+1}, d_r^i) : i \in [r-1]\} \\ & \cup \{(u_{i+1}, u_i) : i \in [r-1]\} \cup \{(l_i, l_{i+1}) : i \in [r-1]\} \cup \{(u_1, c_1), (c_{2r}, l_1)\}. \end{aligned}$$

Note that, according to the definition of V' and F', each trip in $\mathbb{T}$ can be considered as a

walk in D'. We then set

$$\mathbb{T}' = \mathbb{T} \cup \mathbb{T}_r \setminus \{T_r^U, T_r^L, T_r^{\downarrow\mathrm{l}}, T_r^{\downarrow\mathrm{r}}\} \cup \{T_U, T_L, T_{\uparrow\downarrow}\},$$

where $T_U, T_L, T_{\uparrow\downarrow}$ are the following three trips.

- $T_U = \langle t, u_{r-1}, \ldots, u_1, c_1, t, c_2, c_1, c_3, c_2, \ldots, c_{r-1}, c_{r-2}, c_r,\ c_{r-1}\rangle$ (intuitively, T_U replaces T_r^U: it first visits $t = u_r, \ldots, u_1$, it then goes to c_1, and it finally continues exactly as T_r^U).
- $T_L = \langle c_{r+2}, c_{r+1}, c_{r+3}, c_{r+2}, \ldots, c_{2r-2}, c_{2r},\ c_{2r-1}, l_r, c_{2r}, l_1, l_2, \ldots, l_r\rangle$ (intuitively, T_L replaces T_r^L: it first starts exactly as T_r^L, and it then visits $l_1, \ldots, l_r$).
- $T_{\uparrow\downarrow} = \langle s, d_r^{r-2}, \ldots, t, d_r^2, \ldots, d_r^{r-2}, s, l_r, s\rangle$ (intuitively, $T_{\uparrow\downarrow}$ replaces both $T_r^{\downarrow\mathrm{l}}$ and $T_r^{\downarrow\mathrm{r}}$).

$(D', \mathbb{T}')$ **is strongly temporalisable** (even if $(D, \mathbb{T})$ is not). The proof is similar to the one proving that $(D_r, \mathbb{T}_r)$ is strongly temporalisable (see the proof of Theorem 15). Indeed, whenever $h = r$ or $k = r$ in Table 4.3, we can replace T_h^U or T_k^U by T_U (respectively, T_h^L or T_k^L by T_L and $T_h^{\downarrow\mathrm{l}}$ or $T_k^{\downarrow\mathrm{l}}$ by $T_{\uparrow\downarrow}$) in the schedules included in the table. Since T_r^U (respectively, T_r^L and $T_r^{\downarrow\mathrm{l}}$) is included in T_U (respectively, T_L and $T_{\uparrow\downarrow}$), by doing so we have that all the reachability properties of the table are still satisfied apart from the last case of the last cell of the table itself, with $h = k = r$. However, in this case, for any temporalisation σ, we have that $d_r^{l_2} \in V^{\downarrow}$ is σ-reachable from $d_r^{l_1} \in V^{\downarrow}$ with $l_2 < l_1$, since we can just use the trip $T_{\uparrow\downarrow}$ (which allows to go up from $d_r^{l_1}$ to $d_r^{l_2}$).

If $(D, \mathbb{T})$ is (s,t)-temporalisable, then there exists a temporalisation σ' of $(D', \mathbb{T}')$ with σ'-reachability at least $((r-1)r)^2$. To this aim, first note that, for any temporalisation σ' of $(D', \mathbb{T}')$, if $\sigma'(T_{\uparrow\downarrow}) = \tau'$, then the trip $T_{\uparrow\downarrow}$ arrives at l_r at time $\tau' + 2r - 3$, and terminates in s at time $\tau' + 2r - 2$. If $(D, \mathbb{T})$ is (s,t)-temporalisable, then there exists a temporalisation σ of $(D, \mathbb{T})$ such that t is σ-reachable from s. Let P be any temporal path in $G[D, \mathbb{T}, \sigma]$ from s to t, and let τ_s (respectively, τ_a) be the departure (respectively, arrival) time of P from s (respectively, in t). We then define a temporalisation σ' of $(D', \mathbb{T}')$ as follows. For any $T \in \mathbb{T}$, $\sigma'(T) = \sigma(T)$. Moreover, for any $h \in [r-1]$, $\sigma'(T_h^{\downarrow\mathrm{l}}) = \tau_s - 2r + 1$, and $\sigma'(T_L) = \tau_s - 1 - |T_L| = \tau_s - 3r$. This allows the trips $T_h^{\downarrow\mathrm{l}}$ to "meet" the trip T_L in l_h at time $\tau_s - (r-h) - 1$: note that T_L arrives in l_r at time $\tau_s - 1$. Hence, we set $\sigma'(T_{\uparrow\downarrow}) = \tau_s - 2r + 2$ so that $T_{\uparrow\downarrow}$ arrives in l_r also at time $\tau_s - 1$. By using the (last edge of the) trip $T_{\uparrow\downarrow}$, and then the path P, we can arrive in t at time τ_a. By setting $\sigma'(T_U) = \tau_a$ and, for any $h \in [r-1]$, $\sigma'(T_h^{\downarrow\mathrm{r}}) = \tau_a + r - h$, we can arrive at any node d_h^k at time $\tau_a + r - h + k - 1$. In other words, we have shown that all nodes of the first $r-1$ descending gadgets are σ'-reachable one from the other. That is, the σ'-reachability is at least $((r-1)r)^2$.

If $(D, \mathbb{T})$ is not (s,t)-temporalisable, then the σ'-reachability of any temporalisation σ' of $(D', \mathbb{T}')$ is at most $3rn + 7r^2(r-1)$. Let σ' be a temporalisation of $(D', \mathbb{T}')$. First note that σ' induces a temporalisation σ of $(D, \mathbb{T})$. Since $(D, \mathbb{T})$ is not (s,t)-temporalisable, we have that t is not σ-reachable from s. Moreover, since the arc (l_r, s) is the last arc in the trip $T_{\uparrow\downarrow}$, it cannot be used before the other arcs in this trip. As a consequence, t is not σ''-reachable from l_r, where σ'' is the temporalisation induced by σ' on $(D', \mathbb{T} \cup \{T_{\uparrow\downarrow}\})$. The topology of D' thus implies that, for all $i, j \in [r]$, all temporal paths from l_i to u_j in $G[D', \mathbb{T}', \sigma']$ must pass through the nodes $c_{2r}, c_{2r-1}, \ldots, c_{r+1}$, the arc (c_{r+1}, c_r), and the nodes $c_r, c_{r-1}, \ldots, c_1$.

Let $T^L_{i_{\min}}$ be one of the trips with minimum starting time according to σ' among all the trips in the lower gadget ($i_{\min} = r$ if T_L is the only trip with minimum starting time), and let $T^U_{i_{\max}}$ be one of the trips with maximum starting time according to σ' among all the trips in the upper gadget ($i_{\max} = r$ if T_U is the only trip with maximum starting time). Similarly to the proof of Theorem 15, we can then show that there is no temporal path in $G[D', \mathbb{T}', \sigma']$ from l_h to c_{r+1} for $h \in [r] \setminus \{i_{\min}\}$ nor from c_r to u_k for $k \in [r] \setminus \{i_{\max}\}$. Note that the additional part $\langle c_{2r}, l_1, \ldots, l_r\rangle$ of T_L (respectively, $\langle t, u_{r-1}, \ldots, u_1, c_1\rangle$ of T_U), compared to T^L_r (respectively, T^U_r), does not change the reasoning as it comes at the end (respectively, the beginning) of the trip and, in particular, after arc (l_r, c_{2r}) (respectively, before the arc (c_1, l_1)).

We can thus similarly conclude that $d^{l_2}_{h_2}$ is not σ'-reachable from $d^{l_1}_{h_1}$ for all $h_1, h_2 \in [r-1]$ with $h_1 \neq i_{\min}$ or $h_2 \neq i_{\max}$ and all l_1, l_2 with $1 < l_1, l_2 < r$. Note that the situation is different from the previous construction for $l_1 = l_2 = r$ or $l_1 = l_2 = 1$ as T_L makes $d^r_{h_2} = l_{h_2}$ σ'-reachable from $d^r_{h_1} = l_{h_1}$ for $h_1 < h_2$ and that T_U makes $d^1_{h_1} = u_{h_1}$ σ'-reachable from $d^1_{h_2} = u_{h_2}$ for $h_1 < h_2$. Overall, this means that only nodes in

$$\{d^1_{h_1}, \ldots d^r_{h_1}, l_1, \ldots, l_r, c_1, \ldots, c_{2r}, u_1, \ldots, u_r, d^1_{i_{\max}}, \ldots, d^r_{i_{\max}}, d^1_r, \ldots, d^r_r\}$$

can be σ'-reachable from $d^{l_1}_{h_1}$ for $h_1 \in [r-1]$ and $l_1 \in [r]$. Thus, the nodes in the first $r-1$ descending gadgets have σ'-reachability at most $7r$. The other $3r$ nodes have σ'-reachability at most n.

The MRTT problem cannot be approximated within a factor less than $\frac{\sqrt{n}}{12}$. We now prove that any polynomial-time algorithm $\mathcal{A}$ solving MRTT with approximation ratio $\rho < \frac{\sqrt{n}}{12}$ would allow us to decide in polynomial-time whether $(D, \mathbb{T})$ is (s,t)-temporalisable. As this latter problem is NP-complete, as stated by Theorem 11, this will conclude the proof of the theorem. If $(D, \mathbb{T})$ is (s,t)-temporalisable, then then there exists a temporalisation of $(D', \mathbb{T}')$ whose reachability is at least $((r-1)r)^2$. This implies that $\mathcal{A}$, with input the trip network $(D', \mathbb{T}')$, has to provide a temporalisation σ' with σ'-reachability at least $\frac{((r-1)r)^2}{\rho} > \frac{12r^2(r-1)^2}{r+1} > 11r^2(r-1)$ (note that $\sqrt{n} = \sqrt{r^2+2r} < r+1$ and $\frac{r-1}{r+1} = 1 - \frac{2}{r+1} > \frac{11}{12}$ for $r > 23$). On the other hand, if $(D, \mathbb{T})$ is not (s,t)-temporalisable, then the σ'-reachability of any temporalisation σ' of $(D', \mathbb{T}')$ is at most $3rn + 7r^2(r-1) = 10r^3 - r^2 < 11r^2(r-1)$ (note that $r^3 > 10r^2$ for $r > 10$). In summary, $(D, \mathbb{T})$ is (s,t)-temporalisable if and only if $\mathcal{A}$, with input the trip network $(D', \mathbb{T}')$, returns a temporalisation whose reachability is greater than $11r^2(r-1)$. □

Theorem 17. *Unless $P = NP$, the* SS-MRTT *problem cannot be approximated within a factor less than $\frac{\sqrt{n}}{12}$ even if the input trip network is strongly temporalisable.*

Proof. The reduction is exactly the same as the one used in the proof of the previous theorem. According to that proof, if $(D, \mathbb{T})$ is (s,t)-temporalisable, then there exists a temporalisation σ' of $(D', \mathbb{T}')$ such that any source in one of the first $r-1$ descending gadgets has σ'-reachability at least $r(r-1)$. On the other hand, if $(D, \mathbb{T})$ is not (s,t)-temporalisable, then, for any temporalisation σ' of $(D', \mathbb{T}')$, any source in one of the first $r-1$ descending gadgets has σ'-reachability at most $7r$. Any polynomial-time algorithm $\mathcal{A}$ solving SS-MRTT with approximation ratio $\rho < \frac{\sqrt{n}}{12}$ would then allow us to decide, in polynomial-time, whether $(D, \mathbb{T})$ is (s,t)-temporalisable, since $\frac{r(r-1)}{\rho} > \frac{12r(r-1)}{r+1} > 11r > 7r$ for $r > 23$. This concludes the proof of the theorem. □

4.3.1 Symmetric and strongly temporalisable trip networks

Because of the last theorem, we now focus on symmetric *and* strongly temporalisable trip networks.

Fact 2. *Let $(D, \mathbb{T})$ be a symmetric trip network. For any node u, there exists a schedule S of $(D, \mathbb{T})$ such that, for any node v reachable from u in D, v is S-reachable from u.*

Proof. We first note that, due to the symmetricity, any schedule S of $(D, \mathbb{T})$ is such that each node of a trip $T \in \mathbb{T}$ is S-reachable from any other node of T. Let $\mathcal{T}_u$ be a breadth-first search tree in the induced multidigraph $\mathcal{M}$ rooted in u, whose height is $h_{\mathcal{T}_u}$. By using $\mathcal{T}_u$, we will now define a schedule S of $(D, \mathbb{T})$ such that, for any node v in $\mathcal{T}_u$, v is S-reachable from u. This will prove the fact.

We will construct nested partial schedules $S_0, \ldots, S_{h_{\mathcal{T}_u}} = S$ where a larger and larger subset of $\mathbb{T}$ is scheduled. Given a partial schedule S_ℓ, we say that an arc is "covered" by S_ℓ if it belongs to one of the trips scheduled in S_ℓ. Similarly, a node is "covered" by S_ℓ if it is the head or the tail of an arc covered by S_ℓ. At the beginning, we consider an empty schedule S_0. For each level ℓ of $\mathcal{T}_u$ with $\ell \in [h_{\mathcal{T}_u}]$, let $e_1, \ldots, e_{k_\ell}$ be the arcs connecting a node at level $\ell - 1$ to a node at level ℓ which are not yet covered by $S_{\ell-1}$. Let $T_{\ell,1}, \ldots, T_{\ell,k_\ell}$ be k_ℓ (not necessarily distinct) trips in $\mathbb{T}$, which contain the arcs $e_1, \ldots, e_{k_\ell}$, respectively. Recall that $\Upsilon_{\ell,1}, \ldots, \Upsilon_{\ell,k_\ell}$ denote their respective reverse trips. Let $\mathbb{T}_\ell$ denote the set of trips in $\{T_{\ell,1}, \Upsilon_{\ell,1}, \ldots, T_{\ell,k_\ell}, \Upsilon_{\ell,k_\ell}\}$. As the arcs $e_1, \ldots, e_{k_\ell}$ were not covered by $S_{\ell-1}$, trips in $\mathbb{T}_\ell$ are not included in $S_{\ell-1}$. We can thus define S_ℓ as $S_{\ell-1}$ followed by the trips in $\mathbb{T}_\ell$ in an arbitrary order. Note that all arcs from level $\ell - 1$ to level ℓ are now covered by S_ℓ. We continue similarly for the next levels and define $S = S_{h_{\mathcal{T}_u}}$ as the schedule obtained for the last layer.

To prove that any node v in $\mathcal{T}_u$ is S-reachable, we show that the following invariant is preserved: after processing each level ℓ, any node covered by S_ℓ is S_ℓ-reachable. Consider an arc e which is covered by S_ℓ but not by $S_{\ell-1}$ and let v' be its head or tail. Let $T_{\ell,i}, \Upsilon_{\ell,i}$ denote the trip pair in $\mathbb{T}_\ell$ that contains e. Consider the arc e_i from level $\ell - 1$ to ℓ that belongs to $T_{\ell,i}$. The tail u' of e_i is either u or the head of an arc from level $\ell - 2$ to $\ell - 1$. As such an arc is covered by $S_{\ell-1}$, u' is thus $S_{\ell-1}$-reachable according to the invariant. As $T_{\ell,i}$ or $\Upsilon_{\ell,i}$ contains a walk from u' to v', and both $T_{\ell,i}$ and $\Upsilon_{\ell,i}$ are scheduled after $S_{\ell-1}$ in S_ℓ, v' is S_ℓ-reachable. The conclusion follows from the fact that all nodes in $\mathcal{T}_u$ are covered by $S = S_{h_{\mathcal{T}_u}}$. □

Fact 3. *Let $(D, \mathbb{T})$ be a symmetric trip network. For any node u, there exists a schedule S of $(D, \mathbb{T})$ such that, for any node v such that u is reachable from v in D, u is S-reachable from v.*

Proof. The proof is similar to the proof of Fact 2. □

Corollary 3. *Let $(D, \mathbb{T})$ be a symmetric trip network. Then, $(D, \mathbb{T})$ is strongly temporalisable if and only if D is strongly connected.*

Proof. If $(D, \mathbb{T})$ is strongly temporalisable, then, for any two nodes u and v in D, there exists a temporalisation $\sigma_{u,v}$ of $(D, \mathbb{T})$, such that v is $\sigma_{u,v}$-reachable from u. In other words, there exists a temporal path $P_{u \to v}$ from u to v in $G[D, \mathbb{T}, \sigma_{u,v}]$. Hence, v is reachable from u in D. The converse implication is a direct consequence of Fact 2. □

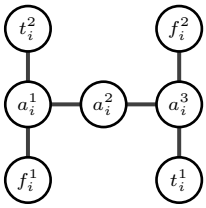

Figure 4.8: The variable gadget in the reduction of 3-SAT to symmetric MRTT (see the proof of Theorem 18).

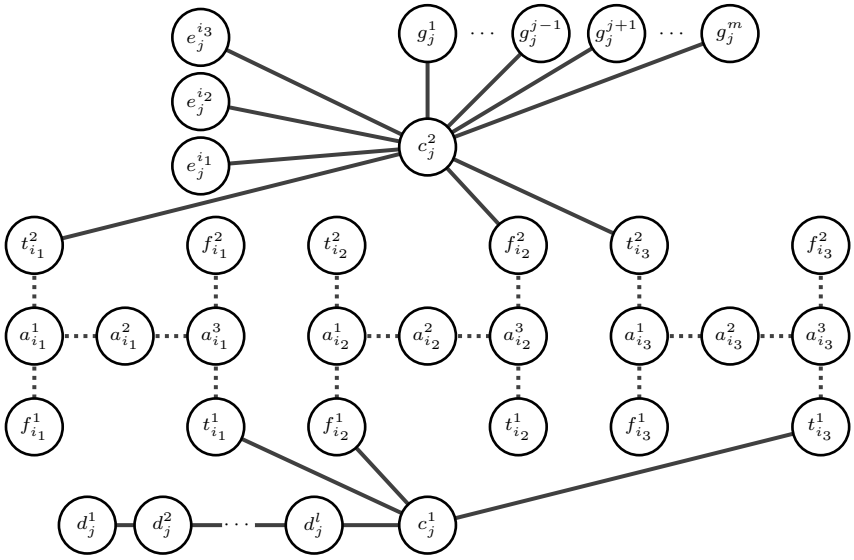

Figure 4.9: The clause gadget in the reduction of 3-SAT to symmetric MRTT (see the proof of Theorem 18), corresponding to the clause $c_j = x_{i_1} \vee \neg x_{i_2} \vee x_{i_3}$ (the dotted arcs are included in the variable gadgets corresponding to the variables x_{i_1}, x_{i_2}, and x_{i_3}). Note that, for each $h \in [m]$ with $h \neq j$, F includes also the arc (c_j^1, c_h^2) (and its reverse arc).

We now prove that the MRTT problem remains NP-hard, even when we assume that the underlying multidigraph is strongly connected and that the trip network is symmetric (from the previous corollary, this implies that the trip networks is strongly temporalisable).

Theorem 18. *The* MRTT *problem problem is NP-hard, even if* $(D, \mathbb{T})$ *is a symmetric trip network and* D *is strongly connected.*

Proof. We reduce 3-SAT to MRTT as follows. Let us consider a 3-SAT formula Φ, with n variables $x_1, \ldots, x_n$ and m clauses $c_1, \ldots, c_m$. Without loss of generality, we will assume that each variable appears positive in at least one clause and negative in at least one clause, that no literal appears in all clauses, and that there are at least two clauses. We set $l = \lceil (7n + m(m+3))^2/(m+2) \rceil + 1$ and $L = (7n + m(m+3))^2 + 1$, and we define the digraph $D = (V, F)$ as the union of the following gadgets (in the following, for each arc included in F, its reverse arc is implicitly also included in F).

Variable gadgets (see Figure 4.8). For each variable x_i of Φ with $i \in [n]$, V contains the

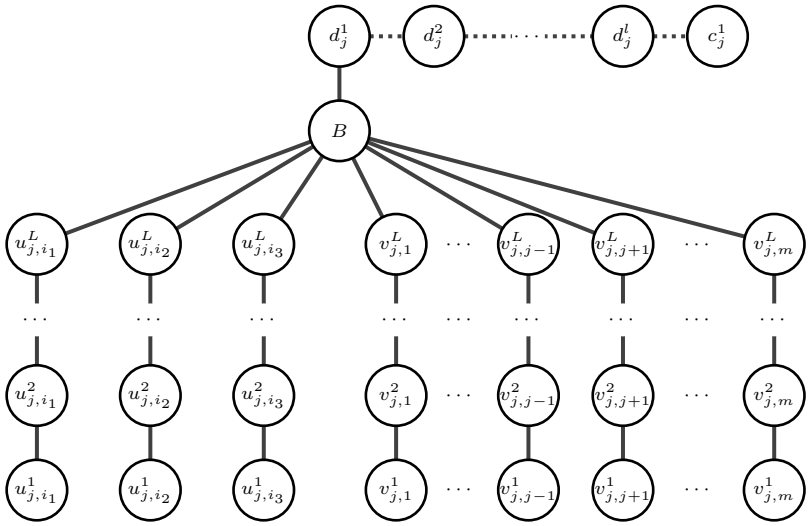

Figure 4.10: The part of the bottom hub gadget in the reduction of 3-SAT to symmetric MRTT (see the proof of Theorem 18), corresponding to the clause c_j which contains the variables x_{i_1}, x_{i_2}, and x_{i_3} (the dotted arcs are included in the clause gadget corresponding to the clause c_j).

seven *variable nodes* t_i^1, t_i^2, f_i^1, f_i^2, a_i^1, a_i^2, and a_i^3, and F contains the six arcs (t_i^1, a_i^3), (a_i^3, f_i^2), (a_i^1, a_i^2), (a_i^2, a_i^3), (f_i^1, a_i^1), and (a_i^1, t_i^2).

Clause gadgets (see Figure 4.9). For each clause c_j of Φ with $j \in [m]$, V contains the two *clause nodes* c_j^1 and c_j^2 (we will call $c_1^1, \ldots, c_m^1$ the *bottom* clause nodes and $c_1^2, \ldots, c_m^2$ the *top* clause nodes), and the *middle nodes* d_j^k for $k \in [l]$. For each variable x_i which appears (positive or negative) in c_j, V contains the *head* node e_j^i. Finally, for each $h \in [m]$ with $h \neq j$, V contains the *head* node g_j^h. Concerning the arcs, for each variable x_i which appears positive in c_j, F contains the arcs (c_j^1, t_i^1) and (t_i^2, c_j^2), while, for each variable x_i which appears negative in c_j, F contains the arcs (c_j^1, f_i^1) and (f_i^2, c_j^2). In both cases, F contains the arc (c_j^2, e_j^i). For each $h \in [m]$ with $h \neq j$, F contains the arcs (c_j^1, c_h^2) and (c_j^2, g_j^h). Finally, F contains the arcs (d_j^k, d_j^{k+1}), for $k \in [l-1]$, and the arc (d_j^l, c_j^1).

Bottom hub gadget (see Figure 4.10). V contains the *bottom hub node* B. For each clause c_j of Φ with $j \in [m]$, and for each variable x_i that appears (positive or negative) in c_j, V contains the set $A_{j,i}^{\mathrm{u}} = \{u_{j,i}^k : k \in [L]\}$. Moreover, for each $h \in [m]$, such that $h \neq j$, V contains the set $A_{j,h}^{\mathrm{v}} = \{v_{j,h}^k : k \in [L]\}$. We will refer to the nodes in $A_{j,i}^{\mathrm{u}}$ and in $A_{j,h}^{\mathrm{v}}$ as the *bottom tail nodes*. Concerning the arcs, F contains the arc (B, d_j^1), and, for each variable x_i that appears (positive or negative) in c_j, the set $\{(u_{j,i}^k, u_{j,i}^{k+1}) : k \in [L-1]\}$ and the arc $(u_{j,i}^L, B)$. Finally, for each $h \in [m]$, such that $h \neq j$, F contains the set $\{(v_{j,h}^k, v_{j,h}^{k+1}) : k \in [L-1]\}$ and the arc $(v_{j,i}^L, B)$.

Top hub gadget (see Figure 4.11) V contains the *top hub node* U. For each clause c_j of Φ with $j \in [m]$, and for each variable x_i that appears (positive or negative) in c_j, V

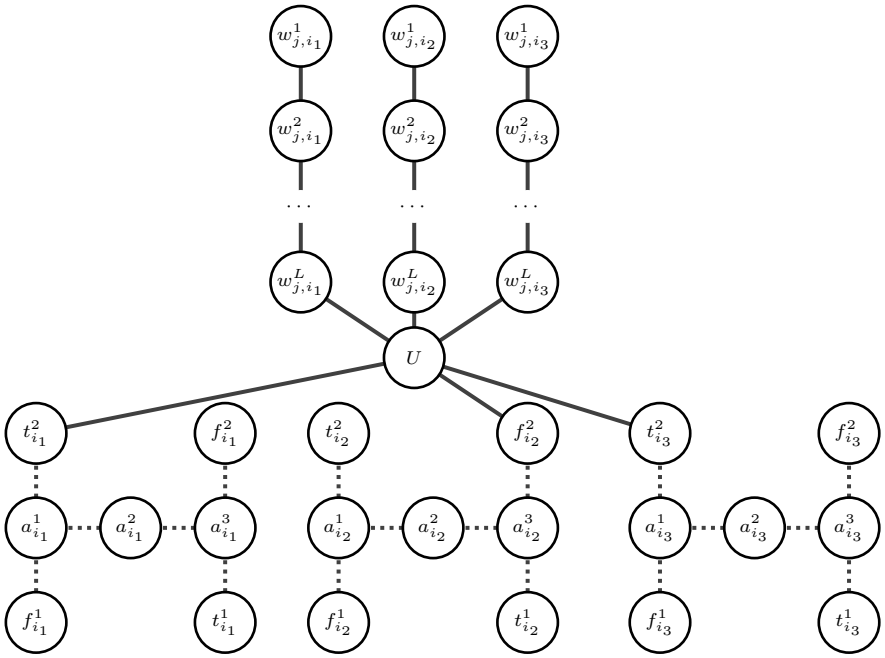

Figure 4.11: The part of the top hub gadget in the reduction of 3-SAT to symmetric MRTT (see the proof of Theorem 18), corresponding to the clause $c_j = x_{i_1} \vee \neg x_{i_2} \vee x_{i_3}$ (the dotted arcs are included in the variable gadgets corresponding to the variables x_{i_1}, x_{i_2}, and x_{i_3}).

contains the set $A^{\mathrm{w}}_{j,i} = \{w^k_{j,i} : k \in [L]\}$. We will refer to the nodes in $A^{\mathrm{w}}_{j,i}$ as the *top tail nodes*. Concerning the arcs, for each variable x_i that appears (positive or negative) in c_j, F contains the set $\{(w^k_{j,i}, w^{k+1}_{j,i}) : k \in [L-1]\}$ and the arc $(w^L_{j,i}, U)$. Finally, if x_i appears positive in c_j, F contains the arc (U, t^2_i), while if x_i appears negative in c_j, F contains the arc (U, f^2_i).

Number of nodes. Let us first compute the cardinality of V. There are $7n$ variable nodes, $2m$ clause nodes, $3m+m(m-1)$ head nodes, ml middle nodes, 2 hub nodes, $3mL+m(m-1)L$ bottom tail nodes, and $3mL$ top tail nodes. Thus, $|V| = 2+7n+m(L+1)(m+4)+ml+mL$.

Trips. We now define the trip collection $\mathbb{T}$ in D (see Figure 4.12). In the following, for each trip T included in $\mathbb{T}$, we implicitly assume that the symmetric trip $\overline{T}$ is also included in $\mathbb{T}$.

Variable trips. For each $i \in [n]$, $\mathbb{T}$ contains the trips $T^{\mathrm{t}}_i = \langle t^1_i, a^3_i, f^2_i\rangle$, $T^{\mathrm{f}}_i = \langle f^1_i, a^1_i, t^2_i\rangle$, and $T^{\mathrm{a}}_i = \langle a^1_i, a^2_i, a^3_i\rangle$ (see Figure 4.12(a)).

Bottom-variable trips. For each clause c_j of Φ with $j \in [m]$, if c_j contains the literal x_i, $\mathbb{T}$ contains the trip $T^{\mathrm{u}}_{j,i} = \langle u^1_{j,i}, \ldots, u^L_{j,i}, B, d^1_j, \ldots, d^l_j, c^1_j, t^1_i\rangle$ (see Figure 4.12(b)), while if c_j contains the literal $\neg x_i$, $\mathbb{T}$ contains the trip $T^{\mathrm{u}}_{j,i} = \langle u^1_{j,i}, \ldots, u^L_{j,i}, B, d^1_j, \ldots, d^l_j, c^1_j, f^1_i\rangle$ (see Figure 4.12(c)).

Bottom-clause trips. For each $(j,h) \in [m]^2$ such that $j \neq h$, $\mathbb{T}$ contains the trip $T^{\mathrm{v}}_{j,h} = \langle v^1_{j,h}, \ldots, v^L_{j,h}, B, d^1_j, \ldots, d^l_j, c^1_j, c^2_h, g^h_j\rangle$ (see Figure 4.12(d)).

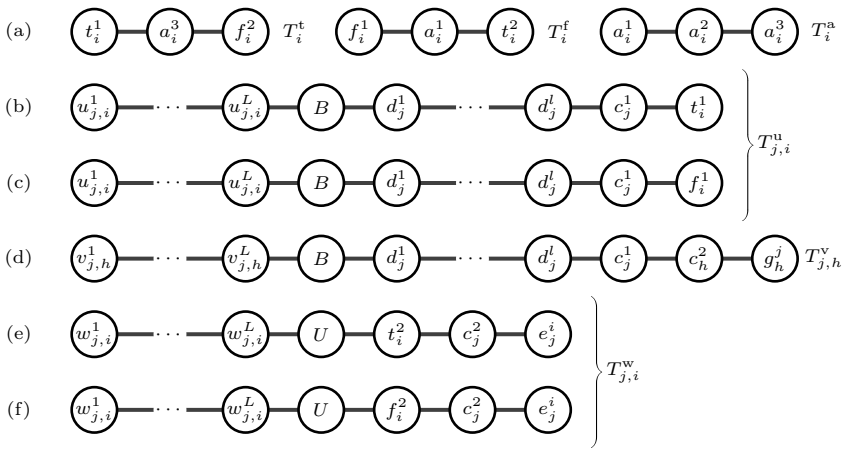

Figure 4.12: The trips in the reduction of 3-SAT to symmetric MRTT (see the proof of Theorem 18): (a) variable trips, (b) and (c) bottom-variable trips, (d) bottom-clause trips, and (e) and (f) top trips. Each kind of trip is included in both directions (that is, the trip collection is symmetric).

Top trips. For each clause $c_j$of Φ with $j \in [m]$, if c_j contains the literal x_i, $\mathbb{T}$ contains the trip $T_{j,i}^{\mathrm{w}} = \langle w_{j,i}^1, \ldots, w_{j,i}^L, U, t_i^2, c_j^2, e_j^i \rangle$ (see Figure 4.12(e)), while if c_j contains the literal $\neg x_i$, $\mathbb{T}$ contains the trip $T_{j,i}^{\mathrm{w}} = \langle w_{j,i}^1, \ldots, w_{j,i}^L, U, f_i^2, c_j^2, e_j^i \rangle$ (see Figure 4.12(f)).

See Figure 4.13 for a global view of the trip network $(D, \mathbb{T})$.

Basic idea of the reduction. The temporal connections that are made possible by a temporalisation of the variable trips T_i^{t}, T_i^{f}, and T_i^{a} correspond to choosing whether the variable x_i is set to true or false. Enabling temporal connections from the middle nodes, which are between the bottom hub B and a bottom clause node c_j^1, to the head nodes connected to the top clause node c_j^2, corresponds to a "reward" in terms of reachability for satisfying the clause. The large size of the tails forces some constraints on the temporalisations with high reachability. This ensures that the "reward" is obtained only if the clause is satisfied. In particular, we will show that if there exists a satisfying assignment for Φ, then it is possible to produce a temporalisation σ such that the σ-reachability is at least Q, where $Q = |V|^2 - (7n + m(m+3))^2$. Otherwise, if Φ is not satisfiable, then any temporalisation has reachability less than Q. More precisely, since $Q > |V|^2 - L$ (recall that $L = (7n + m(m+3))^2 + 1$), we will show that any temporalisation that misses a connection from a bottom/top tail node to any node or from a node to any bottom/top tail node has reachability less than Q. Similarly, since $Q > |V^2| - (m+2)l$ (recall that $l = \lceil (7n + m(m+3))^2/(m+2) \rceil + 1$), we will show that missing the connections from the middle nodes to the head nodes of the corresponding top clause node, also leads to a reachability lower than Q.

Activation of pairs of variable nodes. Consider a variable x_i and the associated variable gadget (see Figure 4.8). For a given temporalisation σ, let $\tau_1 = \sigma(T_i^{\mathrm{f}})$, $\tau_2 = \sigma(T_i^{\mathrm{a}})$, and $\tau_3 = \sigma(T_i^{\mathrm{t}})$. Note that f_i^2 is σ-reachable from f_i^1 by using only the variable trips corresponding to the variable x_i if and only if $\tau_1 + 1 \leq \tau_2$ and $\tau_2 + 2 \leq \tau_3 + 1$, as these conditions enable transfers at a_i^1 and a_i^3. When this is the case, we say that σ *activates* (f_i^1, f_i^2).

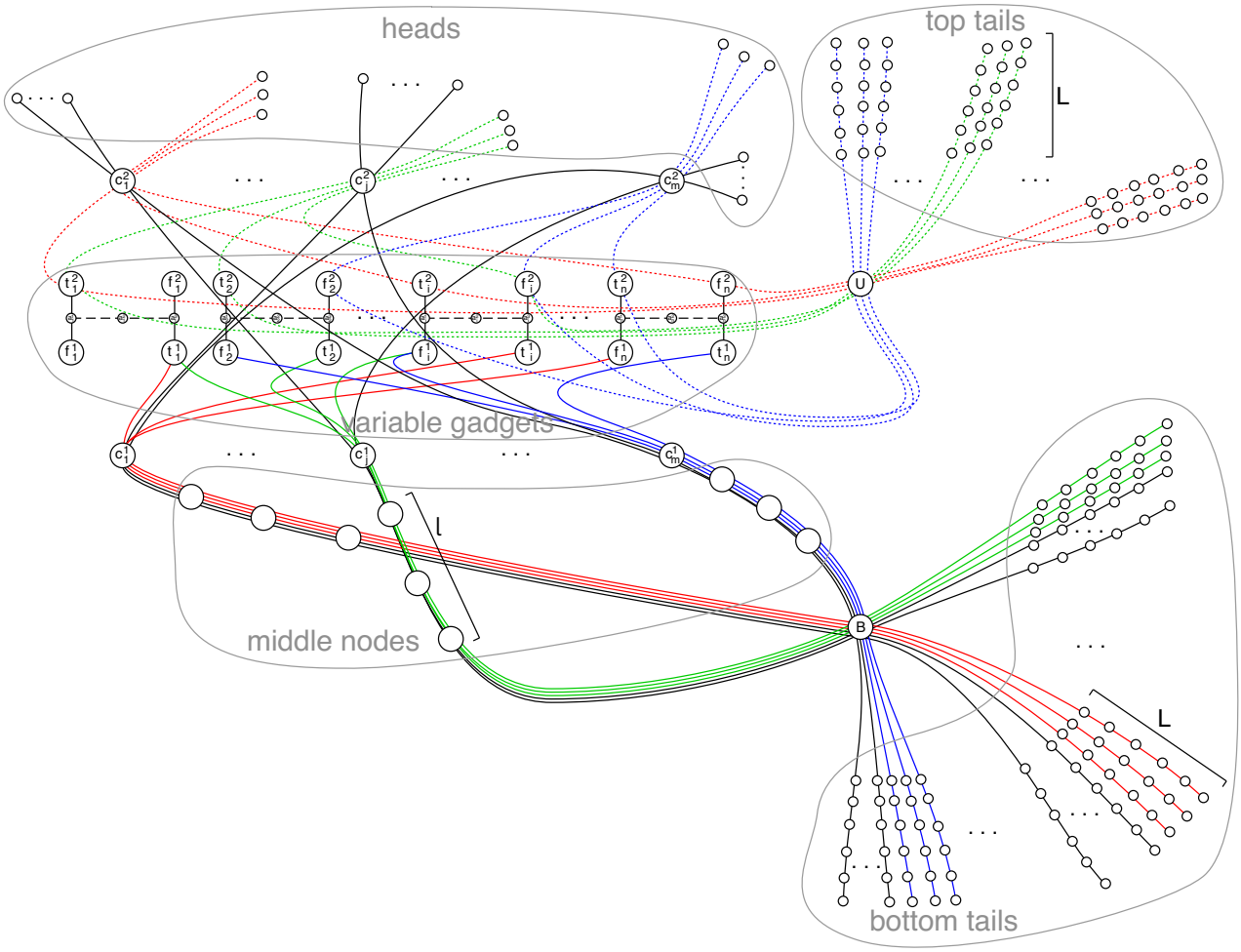

Figure 4.13: A global view of the reduction of 3-SAT to symmetric MRTT. Here the clause $c_j = (x_1 \wedge x_2 \wedge \overline{x_i})$ is associated to two nodes c_j^1 and c_j^2 that are connected to variable gadgets for x_1, x_2 and x_i through green lines (plain lines for bottom-variable trips and dotted lines for top trips). For a more detailed view of each gadget, see Figures 4.8, 4.9, 4.10, 4.11, and 4.12.

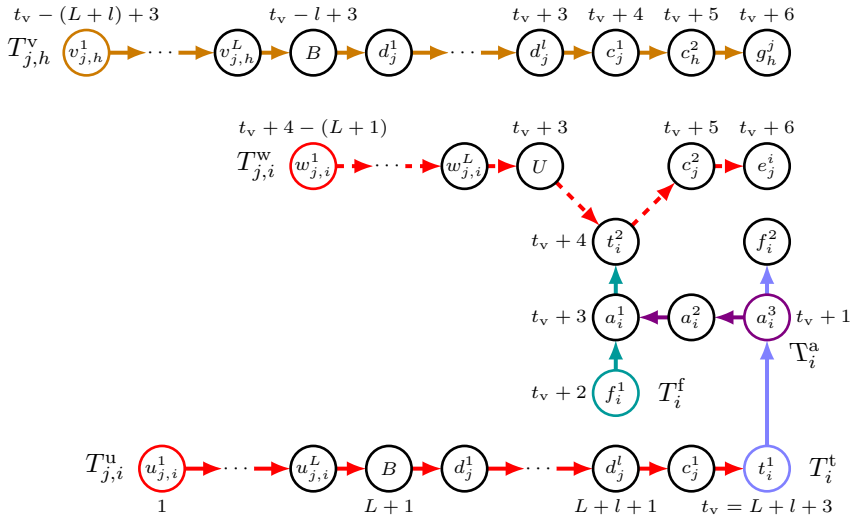

Figure 4.14: The temporalisation of the "forward" trips obtained from a truth assignment α satisfying the Boolean formula Φ (here, we assume that the variable x_i appears positive in the clause c_j, and that $\alpha(x_i) = \text{TRUE}$). The colored nodes are the starting nodes of the trips. Note that $T^u_{j,i}$ arrives in B at time $L+1 = t_v - l - 2$.

Note that if σ *activates* (f^1_i, f^2_i), then $\tau_1 \leq \tau_3 - 2$. Similarly, t^2_i is σ-reachable from t^1_i by using only the variable trips corresponding to the variable x_i if and only if $\tau_3 + 1 \leq \tau_4$ and $\tau_4 + 2 \leq \tau_1 + 1$, where $\tau_4 = \sigma(\Upsilon^a_i)$. When this is the case, we say that σ activates pair (t^1_i, t^2_i). Note that if σ *activates* (t^1_i, t^2_i), then $\tau_3 \leq \tau_1 - 2$. The key observation for the sequel is that *no temporalisation can activate both* (f^1_i, f^2_i) *and* (t^1_i, t^2_i). We similarly say that σ activates pair (f^2_i, f^1_i) (respectively, (t^2_i, t^1_i)) when f^1_i (respectively, t^1_i) is σ-reachable from f^2_i (respectively, t^2_i), by using only the variable trips corresponding to the variable x_i. Once again, no temporalisation can activate both (f^2_i, f^1_i) and (t^2_i, t^1_i). However, one can easily see that it is possible to activate either both (f^1_i, f^2_i) and (f^2_i, f^1_i) or both (t^1_i, t^2_i) and (t^2_i, t^1_i).

Constructing a temporalisation from a satisfying assignment. We first show how to construct a temporalisation σ, when Φ is satisfiable, with reachability at least Q. Let α be a truth assignment to $x_1, \ldots, x_n$ that satisfies Φ (see Figures 4.14 and 4.15, where we assume that x_i appears positive in c_j and that $\alpha(x_i) = \text{TRUE}$).

For any clause c_j and for any variable x_i appearing in c_j, we set $\sigma(T^u_{j,i}) = 1$. Note that $T^u_{j,i}$ arrives in B at time $L+1$, in c^1_j at time $L+l+2$, and in the variable node connected to c^1_j and included in the variable gadget corresponding to x_i at time $\tau_v = L+l+3$. For each $i \in [n]$, if $\alpha(x_i) = \text{TRUE}$, then we set $\sigma(T^t_i) = \tau_v$, $\sigma(\Upsilon^a_i) = \tau_v + 1$, and $\sigma(T^f_i) = \tau_v + 2$, so that σ activates (t^1_i, t^2_i) (t^2_i being reachable at time $\tau_v + 4$). Otherwise (that is, $\alpha(x_i) = \text{FALSE}$), we set $\sigma(T^f_i) = \tau_v$, $\sigma(T^a_i) = \tau_v + 1$, and $\sigma(T^t_i) = \tau_v + 2$ so that σ activates (f^1_i, f^2_i) (f^2_i being reachable at time $\tau_v + 4$). For any clause c_j and for any variable x_i appearing in c_j, we set $\sigma(T^w_{j,i}) = \tau_v + 4 - (L+1)$, and, for any two clause c_j and c_h with $j \neq h$, we set $\sigma(T^v_{j,h}) = \tau_v - (L+l) + 3$: this implies that all these trips reach their top clause node at the same time, that is, $\tau_v + 5$.

For any clause c_j and for any variable x_i appearing in c_j, we set $\sigma(\Upsilon^w_{j,i}) = \tau_v + 6$, and,

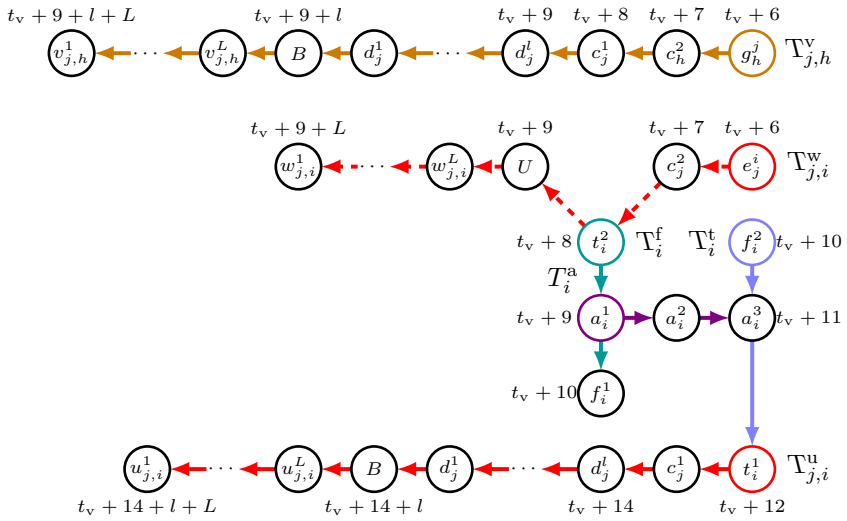

Figure 4.15: The temporalisation of the "backward" trips obtained from a truth assignment α satisfying the Boolean formula Φ (here, we assume that the variable x_i appears positive in the clause c_j, and that $\alpha(x_i) = \text{TRUE}$). The colored nodes are the starting nodes of the trips and $\tau_v = L + l + 3$.

for any two clause c_j and c_h with $j \neq h$, we set $\sigma(\Upsilon^v_{j,h}) = \tau_v + 6$: this implies that all these trips reach their top clause node at the same time, that is, $\tau_v + 7$. For each $i \in [n]$, if $\alpha(x_i) = \text{TRUE}$, then we set $\sigma(\Upsilon^f_i) = \tau_v + 8$, $\sigma(T^a_i) = \tau_v + 9$, and $\sigma(\Upsilon^t_i) = \tau_v + 10$, so that σ activates (t^2_i, t^1_i) (t^1_i being reachable at time $\tau_v + 12$). Otherwise (that is, $\alpha(x_i) = \text{FALSE}$), we set $\sigma(\Upsilon^t_i) = \tau_v + 8$, $\sigma(\Upsilon^a_i) = \tau_v + 9$, and $\sigma(\Upsilon^f_i) = \tau_v + 10$ so that σ activates (f^2_i, f^1_i) (f^1_i being reachable at time $\tau_v + 12$). For any clause c_j and for any variable x_i appearing in c_j, we set $\sigma(\Upsilon^u_{j,i}) = \tau_v + 12$. Note that $\Upsilon^u_{j,i}$ arrives in B at time $\tau_v + 14 + l$.

We now show that the σ-reachability is at least Q. To this aim, let $G = G[D, \mathbb{T}, \sigma]$ be the temporal graph induced by σ, and let X be set of the following nodes: the top and bottom tail nodes, the middle nodes, the two hub nodes, and the nodes c^1_j for $j \in [m]$ (note that $V \setminus X$ contains all the head nodes, all the variable nodes, and the nodes c^2_j for $j \in [m]$).

Claim 6. *For any node $x \in X$ and for any node $v \in V$, $v \in \mathcal{R}_G(x)$ and $x \in \mathcal{R}_G(v)$.*

Proof. Let us first show that, for any node $x \in X$ and for any node $v \in V \setminus X$, $x \in \mathcal{R}_G(v)$. First note that each top clause node c^2_j can be reached at time at most $\tau_v + 7$ by the following set R_j of nodes: c^2_j itself, each head e^i_j through the trip $\Upsilon^w_{j,i}$ and each head g^h_j with $h \neq j$ through the trip $\Upsilon^v_{j,h}$ (see Figure 4.15), and all variable nodes of any gadget associated to a variable x_i appearing in c_j through the trips T^t_i, Υ^a_i, and T^f_i or through the trips T^f_i, T^a_i, and T^t_i (see Figure 4.14). Notice that $\bigcup_{j \in [m]} R_j = V \setminus X$. Now we show that, by departing from c^2_j at time $\tau_v + 7$, it is possible to reach each node in X, thus implying that, for any node $x \in X$ and for any node $v \in V \setminus X$, $x \in \mathcal{R}_G(v)$.

- The top hub U can be reached at time $\tau_v + 9$ through any trip $\Upsilon^w_{j,i}$ such that variable x_i appears in c_j (see Figure 4.15).

- All top tail nodes $w^r_{p,q}$ are reachable through the trips $\Upsilon^{\mathrm{w}}_{p,q}$, which all "meet" in U at time $\tau_{\mathrm{v}} + 9$ (see Figure 4.15).

- For any $h \in [m]$ with $h \neq j$, the bottom clause nodes c^1_h, the middle nodes d^r_h for $r \in [l]$, the bottom hub B, and the bottom tail nodes $v^s_{h,j}$ for $s \in [L]$ are reachable through the trips $\Upsilon^{\mathrm{v}}_{h,j}$ (see Figure 4.15). Note that all these trips arrive in B at time $\tau_{\mathrm{v}} + 9 + l$.

- Consider a variable x_i appearing in c_j and such that the associated literal has value TRUE according to the assignment α satisfying Φ. The bottom clause node c^1_j, the middle nodes d^r_j for $r \in [l]$, and all bottom tail nodes $u^r_{j,i}$ for $r \in [L]$ are reachable through the trips Υ^{f}_i, T^{a}_i, Υ^{t}_i, and $\Upsilon^{\mathrm{u}}_{j,i}$ or through the trips Υ^{t}_i, Υ^{a}_i, Υ^{f}_i, and $\Upsilon^{\mathrm{u}}_{j,i}$ (see Figure 4.15).

- All bottom tail nodes $u^r_{h,i}$ with $h \neq j$ and $r \in [L]$ are reachable through the trips $\Upsilon^{\mathrm{u}}_{h,i}$, which all "meet" in the bottom hub B at time $\tau_{\mathrm{v}} + 14 + l$, that is, later than the bottom-clause trips $\Upsilon^{\mathrm{v}}_{h,j}$ (see Figure 4.15).

Let us now prove that, for any node $x \in X$ and for any node $v \in V$, $v \in \mathcal{R}_G(x)$. First we prove that, for each $j \in [m]$, the top clause node c^2_j is reachable at time $t_{\mathrm{v}} + 5$ from each node in X.

- The bottom hub B and all the bottom tail nodes $u^r_{p,q}$ and $v^r_{p,q}$ can reach B at time at most $\tau_{\mathrm{v}} - l + 3$ through the trips $T^{\mathrm{u}}_{p,q}$ and $T^{\mathrm{v}}_{p,q}$, and, hence, can reach the top clause node c^2_j at time $\tau_{\mathrm{v}} + 5$ through the trips $T^{\mathrm{v}}_{p,q}$ (see Figure 4.14).

- The top hub U and all the top tail nodes $w^r_{p,q}$ can reach c^2_j at time $\tau_{\mathrm{v}} + 5$ through the trips $T^{\mathrm{w}}_{p,q}$, which all "meet" in U at time $\tau_{\mathrm{v}} + 3$ (see Figure 4.14).

- The bottom clause nodes c^1_h, with $h \neq j$, and the middle nodes d^r_h for $r \in [l]$ can reach the top clause node c^2_j at time $\tau_{\mathrm{v}} + 5$ through the trips $T^{\mathrm{v}}_{h,j}$ (see Figure 4.14).

- The bottom clause node c^1_j and the middle nodes d^r_j for $r \in [l]$ can reach c^2_j at time $\tau_{\mathrm{v}} + 5$ through the trips T^{t}_i, Υ^{a}_i, and T^{f}_i or through the trips T^{f}_i, T^{a}_i, and T^{t}_i (see Figure 4.14), where x_i is a variable whose truth assignment satisfies the clause c_j (note that the satisfiability of the formula Φ is also required here).

Now we show that, by departing from c^2_j at time $\tau_{\mathrm{v}} + 5$, it is possible to reach the following set of nodes S_j: all nodes in X (since we already proved that these nodes are reachable from c^2_j, departing at time $\tau_{\mathrm{v}} + 7 > \tau_{\mathrm{v}} + 5$), the head nodes e^r_j and g^r_j through the trips $T^{\mathrm{w}}_{j,r}$ and $T^{\mathrm{v}}_{j,r}$, and all variable nodes of the gadget associated to a variable x_i appearing in c_j through the trips Υ^{f}_i, T^{a}_i, and Υ^{t}_i or through the trips Υ^{t}_i, Υ^{a}_i, and Υ^{f}_i (see Figure 4.15). Notice that $\bigcup_{j \in [m]} S_j = V$, and this concludes the proof of the claim. □

We now determine a lower bound on the σ-reachability by counting the number of nodes temporally reachable from different sources.

- From the nodes in X, that is the $3mL + m(m+2)L$ top and bottom tail nodes, the ml middle nodes, the 2 hub nodes, and the m bottom clause nodes, it is possible to reach each node in V. This adds $|X| \cdot |V| = |V|^2 - (|V| - |X|) \cdot |V|$ to the σ-reachability. Note that $|X| = Lm(m+5) + ml + 2 + m$.

- From the nodes in $V \setminus X$, that is the $7n$ variable nodes, the $m(m+2)$ head nodes, and the m top clause nodes it is possible to reach the nodes in X. This adds $(7n + m(m+3)) \cdot |X| = (|V| - |X|) \cdot |X|$ to the σ-reachability.

Hence, the σ-reachability is at least equal to $|V|^2-(|V|-|X|)^2 = |V|^2-(7n+m(m+3))^2 = Q$ (note that $|V| = 2+7n+m(L+1)(m+4)+ml+mL = (Lm(m+5)+ml+2+m)+(7n+m(m+3)) = |X| + (|V| - |X|)$).

Bounding reachability when Φ is not satisfiable. Let σ be any trip temporalisation of the trip network $(D, \mathbb{T})$ and let $G = G[D, \mathbb{T}, \sigma]$ be the temporal graph induced by σ.

Claim 7. *If the σ-reachability is at least equal to Q, then, for any $v \in V$ and for any bottom/top tail node x, we have $v \in \mathcal{R}_G(x)$ and $x \in \mathcal{R}_G(v)$.*

Proof. Without loss of generality, we prove the claim in the case in which $x \in A^{\mathrm{u}}_{j,i}$, for some clause c_j of Φ with $j \in [m]$ and for some variable x_i that appears (positive or negative) in c_j (the proofs of the other cases are similar). First of all observe that all nodes in $A^{\mathrm{u}}_{j,i}$ have the same reachability set and belong to the same reachability sets. Formally, for any two nodes $u^r_{j,i}$ and $u^s_{j,i}$ in $A^{\mathrm{u}}_{j,i}$ with $r, s \in [L]$, we have that $\mathcal{R}_G(u^r_{j,i}) = \mathcal{R}_G(u^s_{j,i})$, and that, for any node $v \in V$, $u^r_{j,i} \in \mathcal{R}_G(v)$ if and only if $u^s_{j,i} \in \mathcal{R}_G(v)$. This is due to the fact the bottom-variable trips $T^{\mathrm{u}}_{j,i}$ and $\Upsilon^{\mathrm{u}}_{j,i}$ are the only trips passing through the nodes in $A^{\mathrm{u}}_{j,i}$. This observation implies that if there exists $v \in V$ such that either $v \notin \mathcal{R}_G(u^r_{j,i})$ or $u^r_{j,i} \notin \mathcal{R}_G(v)$ for some bottom tail node $u^r_{j,i}$, then the σ-reachability is at most $|V|^2 - L < Q$. Hence, the claim follows. □

Let us now consider the following time constraints that, as a consequence of the above claim, need to be satisfied by σ, if the σ-reachability is at least equal to Q. For the sake of brevity, we will give a detailed proof of the first constraint only, since the proofs of the other ones are similar: intuitively, these proofs are based on the fact that the connections provided by some trips between two nodes use the minimum number of arcs.

C1 For all clauses c_j of Φ with $j \in [m]$ and for all variables x_i that appear (positive or negative) in c_j, all trips $T^{\mathrm{w}}_{j,i}$ are assigned the same starting time τ^{w}, that is, $\sigma(T^{\mathrm{w}}_{j,i}) = \tau^{\mathrm{w}}$ (this implies that all these trips reach node U at time $t^{\mathrm{U}} = \tau^{\mathrm{w}} + L$ and the node c^2_j at time $\tau^{\mathrm{U}} + 2$). This constraint is needed in order to have any top tail node able to reach any head node at the end of a top trip. Indeed, if there exists two trips $T^{\mathrm{w}}_{j_1,i_1}$ and $T^{\mathrm{w}}_{j_2,i_2}$, with $j_1, j_2 \in [m]$, $i_1, i_2 \in [n]$, and the variable x_{i_1} (respectively, x_{i_2}) appearing (positive or negative) in the clause c_{j_1} (respectively, c_{j_2}), such that $\sigma(T^{\mathrm{w}}_{j_1,i_1}) > \sigma(T^{\mathrm{w}}_{j_2,i_2})$, since there is no trip connecting U to $c^2_{j_2}$ by using less than two arcs, any top tail node in $A^{\mathrm{w}}_{j_1,i_1}$ reaches $c^2_{j_2}$ at time $\sigma(T^{\mathrm{w}}_{j_1,i_1})+L+2 > \sigma(T^{\mathrm{w}}_{j_2,i_2})+L+2$, thus implying that it cannot reach the head node $e^{i_2}_{j_2}$ (since the arc $(c^2_{j_2}, e^{i_2}_{j_2})$ is assigned departure time $\sigma(T^{\mathrm{w}}_{j_2,i_2}) + L + 2$). Because of the previous claim, this contradicts the assumption that the σ-reachability is at least equal to Q.

C2 For all clauses c_j of Φ with $j \in [m]$ and for all variables x_i that appear (positive or negative) in c_j, all trips $\Upsilon^{\mathrm{w}}_{j,i}$ are assigned the same starting time $\tau^{\mathrm{w,s}}$, that is, $\sigma(\Upsilon^{\mathrm{w}}_{j,i}) = \tau^{\mathrm{w,s}}$ (this implies that all these trips reach node U at time $\tau^{\mathrm{U,s}} = \tau^{\mathrm{w,s}} + 3$). This constraint is needed in order to have any head node at the end of a top trip able to reach any top tail node.

C3 $\tau^{\mathrm{U}} \leq \tau^{\mathrm{U,s}}$. This constraint is needed in order to have any top tail node in $A^{\mathrm{w}}_{j_1,i_1}$ able to reach any top tail node in $A^{\mathrm{w}}_{j_2,i_2}$, by first using the trip $T^{\mathrm{w}}_{j_1,i_1}$ (in order to reach U at time t^{U}, as stated in C1) and then using the trip $\Upsilon^{\mathrm{w}}_{j_2,i_2}$ (which passes through U at time $\tau^{\mathrm{U,s}}$, as stated in C2).

C4 For each clause c_j of Φ with $j \in [m]$ and for any $h \in [m]$ with $h \neq j$, $\sigma(T^{\mathrm{v}}_{j,h}) = \tau^{\mathrm{U}} - L - l$. This constraint is needed in order to have any top tail node able to reach any head node at the end of a bottom-clause trip and any bottom tail node able to reach any head node at the end of a top trip.

C5 For each clause c_j of Φ with $j \in [m]$ and for any $h \in [m]$ with $h \neq j$, $\sigma(\Upsilon^{\mathrm{v}}_{j,h}) = \tau^{\mathrm{U,s}} - 3$. This constraint is needed in order to have any head node able to reach both any top tail node and any bottom tail node.

C6 For each clause c_j of Φ with $j \in [m]$ and for each variable x_i that appears (positive or negative) in c_j, $\sigma(T^{\mathrm{u}}_{j,i}) \leq \tau^{\mathrm{U}} - L - l$. This constraint is needed in order to have any bottom tail node able to reach any head node at the end of a bottom-clause trip.

C7 For each clause c_j of Φ with $j \in [m]$ and for each variable x_i that appears (positive or negative) in c_j, $\sigma(\Upsilon^{\mathrm{u}}_{j,i}) \geq \tau^{\mathrm{U,s}} - 2$. This constraint is needed in order to have any head node at the end of a bottom-clause trip able to reach any bottom tail node.

C8 For each clause c_j of Φ with $j \in [m]$, for each variable x_i that appears (positive or negative) in c_j, and for any $h \in [m]$ with $h \neq j$, $\sigma(T^{\mathrm{v}}_{j,h}), \sigma(T^{\mathrm{u}}_{j,i}) \leq \tau^{\mathrm{U,s}} - L - l - 4$. These constraints are needed in order to have any bottom tail node able to reach any top tail node.

The above constraints have been derived by using the fact that $Q > |V|^2 - L$. We now take advantage of the fact that $Q > |V|^2 - (m+2)l$. Note that, since Φ is not satisfiable, for any truth-assignment to the variables of Φ, there must exist a clause which is not satisfied by the assignment. Let us then consider the following truth-assignment α, which is derived from σ. For each variable gadget corresponding to a variable x_i, $\alpha(x_i) = \textsc{True}$ if $\sigma(T^{\mathrm{t}}_i) \leq \sigma(T^{\mathrm{f}}_i)$, otherwise $\alpha(x_i) = \textsc{False}$. Note that from the paragraph about the activation of pairs of variable nodes, it follows that if $\alpha(x_i) = \textsc{True}$ (respectively, $\alpha(x_i) = \textsc{False}$), then σ does not activate (f^1_i, f^2_i) (respectively, (t^1_i, t^2_i)). Let c_{j_α} be a clause which is not satisfied by α. We now show that the middle nodes $d^k_{j_\alpha}$, for $k \in [l]$, cannot reach the head nodes connected to $c^2_{j_\alpha}$. Intuitively, the main reason is that $c^2_{j_\alpha}$ cannot be reached from the middle nodes through any of the variable gadgets associated to the variables appearing in c_{j_α}, as σ does not activate, in these gadgets, the pair connected to $c^1_{j_\alpha}$ and $c^2_{j_\alpha}$, and no other temporal path is possible. More formally, let us analyse all possible temporal paths from a middle node x (with $x = d^k_{j_\alpha}$, for some $k \in [l]$) to a head node h connected to $c^2_{j_\alpha}$.

- Going through a node c^2_k with $k \neq j_\alpha$ is not allowed by the above time constraints (in particular, by the synchronization of forward bottom-clause trips and forward top trips at $c^2_{j_\alpha}$ and c^2_k). Indeed, each temporal path from x to c^2_k reaches c^2_k using either a top forward trip or a bottom-clause forward trip, which both arrive in c^2_k at time $\tau^{\mathrm{U}} + 2$ according to time constraints C1 and C4. Although $c^2_{j_\alpha}$ might be σ-reachable from c^2_k, the arrival time will be greater than $\tau^{\mathrm{U}} + 2$ while the trip to h depart from $c^2_{j_\alpha}$ at time $t^{\mathrm{U}} + 2$.

- Using any trip to go to B and then reach h through a node c_k^1 with $k \neq j_\alpha$ is also not possible. Let us suppose we use a backward bottom-variable or bottom-clause trip T_1 to go from x to B, arrive in B at time τ, and then use a forward bottom-variable or bottom-clause trip T_2 to reach c_k^1, and let τ' be the time T_2 goes through B. Clearly, $\tau' \geq \tau$. Because of the temporal constraints C5 and C7, we have that $\tau \geq \tau^{\mathrm{U,s}} + l$, which implies $\tau \geq \tau^{\mathrm{U}} + l$ because of constraint C3. However, because of temporal constraints C4 and C6 we have that $\tau' \leq \tau^{\mathrm{U}} - l$ in contradiction with $\tau \leq \tau'$.

- Going through a variable node t_i^1, if we assume that x_i appears positive in c_{j_α}, is not possible either. As mentioned before the choice of c_{j_α} implies that σ does not activate pair (t_i^1, t_i^2) and the path of length four through the variable gadget for x_i from t_i^1 to t_i^2 is not σ-compatible. We could consider reaching f_i^2 and then a clause node c_k^2 but the situation would be similar as in the first case. We could finally consider to go from t_i^1 to another clause node c_k^1 such that x_i also appears positive in c_k. Let Υ_k denote the backward bottom-variable trip allowing to go from t_i^1 to c_k^1 and let t denote the time when it arrives in c_k^1. The time constraint C7 then imply that $\tau = \sigma(\Upsilon_k)+1 \geq \tau^{\mathrm{U,s}}-1$. Let T be a bottom-variable or bottom-clause trip that we use to leave c_k^1 and later reach h. T has to arrive in c_k^1 at $\tau' \geq \tau$. Since $\tau' = \sigma(T) + L + l + 1$, from the time constraint C8 it follows that $\tau' \leq \tau^{\mathrm{U,s}} - 3$ in contradiction with $\tau' \geq \tau$.

- Going through a variable node f_i^1, if we assume that x_i appears negative in c_{j_α}, is not possible either for similar reasons.

We have thus proved that no head connected to $c_{j_\alpha}^2$ is σ-reachable from any middle node between B and $c_{j_\alpha}^1$: this implies that the reachability of σ is at most $|V|^2 - l(m+2) < Q$. This concludes the proof of the theorem. □

Our last result shows that the maximum temporal reachability obtainable in symmetric strongly temporalisable trip networks is quadratic with respect to the number of nodes. The general idea to prove this result is to find a somewhat central trip, and then schedule trips so that a constant fraction of nodes can reach the central trip and a constant fraction of nodes are reached from the central trip relying on Facts 2 and 3. We will find this central trip as a centroid in a weighted tree.

Let us, then, first recall the definition of centroid. Given a node-weighted tree R, the weight of R is defined as the sum of the weights of its nodes. We then define a *weighted centroid* of a tree R of weight K as a node c such that the removal of c disconnects R into subtrees of weight $2K/3$ at most. Such a centroid can be found efficiently as stated below.

Lemma 5 (Folklore). *Given a node-weighted tree R, a centroid node c can be found in linear time. Moreover, if the weight of R is K and the centroid c has weight $2K/3$ at most, then there exists a partition P_1, P_2 of its pending subtrees such that both $P_1 \cup \{c\}$ and P_2 have total weight $2K/3$ at most. Such a partition can be computed at the cost of sorting the subtrees by non-decreasing weight.*

Proof. Note that the classical algorithm for finding a centroid in an unweighted tree can easily be adapted to the weighted case. Recall that it consists in starting from any node v. If it is not a centroid, then move to a neighbor whose subtree has weight greater than $K/2$ and repeat the test until finding a centroid. The partition P_1, P_2 is obtained by trying to add subtrees in P_1 one after another by non-decreasing weight and stopping as soon as $P_1 \cup \{c\}$ has weight $K/3$ or more. □

Theorem 19. *Let $(D, \mathbb{T})$ be a symmetric and strongly temporalisable trip network. Then there exists a schedule S such that the S-reachability of $(D, \mathbb{T})$ is at least a fraction $2/9$ of all node pairs. Such a schedule can be computed in polynomial time.*

Proof. Without loss of generality, we can assume that all trips in $\mathbb{T}$ are distinct. If this is not the case, we can keep only one of multiple copies of the same trip: this can only reduces the reachability of the modified trip network. We can then consider trips in pairs (T, Υ), where Υ is the reverse trip of T, and we denote by $\mathbb{TP}$ the set of such pairs. For any trip T, we also denote by $V(T) \subseteq V$ the set of nodes which T (and Υ) passes through. Finally, we assign arbitrarily each node $v \in V$ to a single trip pair (T, Υ) such that $v \in V(T)$, and let n_T denote the number of nodes assigned to the pair (T, Υ).

We now define the *transfer* undirected graph $\mathbb{P} = (\mathbb{TP}, \mathbb{FP})$, where two trip pairs are connected when they share a node, that is, $\{(T_1, \Upsilon_1), (T_2, \Upsilon_2)\} \in \mathbb{FP}$ if and only if $V(T_1) \cap V(T_2) \neq \emptyset$. Let $\mathcal{M}$ denote the multidigraph induced by $(D, \mathbb{T})$. According to Corollary 3, $\mathcal{M}$ is strongly connected and, hence, $\mathbb{P}$ is connected. We then compute in linear time a weighted spanning tree R of $\mathbb{P}$, where each trip pair (T, Υ) is weighted by the number n_T. Note that the weight of R is the number $n = |V|$ of nodes in D. We then find a centroid $(C, \circlearrowleft)$ of R according to Lemma 5.

First suppose that $(C, \circlearrowleft)$ has weight greater than $2n/3$. Let S be any schedule of $(D, \mathbb{T})$ that starts with C followed by $\circlearrowleft$. Then, we have that, for any u and v in $V(C)$, v is S-reachable from u, that is, the S-reachability is greater than $4n^2/9$. The theorem follows.

Conversely, let us suppose that $(C, \circlearrowleft)$ has weight $2n/3$ at most. According to Lemma 5, we then consider a partition P_1, P_2 of $R \setminus \{(C, \circlearrowleft)\}$ such that both $P_1 \cup \{(C, \circlearrowleft)\}$ and P_2 have weight $2n/3$ at most. For $i = 1, 2$, let $V(P_i)$ denote the set of nodes assigned to T, for some trip T such that $(T, \Upsilon) \in P_i$. Let $B_1, \ldots, B_{|P_1|}$ be the subtrees in P_1, sorted in an arbitrary way. For each i with $i \in [|P_1|]$, the subtree B_i corresponds to a strongly connected set V_i of D. Moreover, V_i must contain a node $u_i \in V(C)$ as some trip pair of B_i is connected to $(C, \circlearrowleft)$ in $\mathbb{P}$. We can then define a schedule S_i of the trip pairs in B_i according to Fact 3 so that, for any node v in B_i, u_i is S_i-reachable from v. Hence, the schedule $S = S_1, \ldots, S_{|P_1|}, C, \circlearrowleft$ is such that, for any $u \in V(P_1) \cup V(C)$ and $c \in V(C)$, c is S-reachable from u. Similarly, by reasoning on the subtrees in P_2, we can extend S, so that, for any $u \in V(P_2)$ and $c \in V(C)$, u is S-reachable from c. In other words, the final schedule S is such that, for any $u \in V(P_1) \cup V(C)$ and $v \in V(P_2)$, v is S-reachable from u.

Let n_2 denote the weight of P_2, that is, $n_2 = |V(P_2)|$. As both $P_1 \cup \{(C, \circlearrowleft)\}$ and P_2 have weight $2n/3$ at most, we have that $n/3 \leq n_2 \leq 2n/3$. Hence, the number of pairs of nodes u and v such that $u \in V(P_1) \cup V(C)$ and $v \in V(P_2)$ is at least $(n - n_2)n_2 \geq \frac{2}{9}n^2$ for $n/3 \leq n_2 \leq 2n/3$. The theorem thus follows. □

4.4 Single arc trip networks

An interesting special case of trip network is met when all trips consist of a single arc. To lighten the notation, we can thus consider the problem of directly assigning departure times to the arcs in the digraph (or multidigraph) in input without going through the trip set, which becomes redundant. As we did for the previous problems, also in this case for simplicity we will keep working under the assumption that the weights of the arcs are one, leading to uniform strict temporal graphs after the temporalisation. However, all the results and considerations can be applied to arbitrary positive weights.

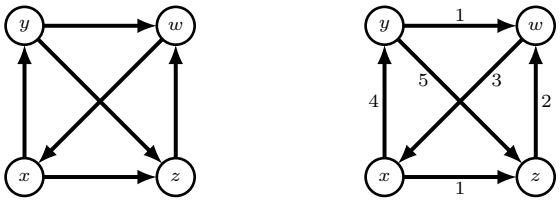

Figure 4.16: A digraph D (left), and an arc temporalisation of D (right). The temporal reachability of the resulting temporal graph is 16, since each node is temporally reachable from any other node.

The problem is formulated as follows.

> Maximum Reachability Arc Temporalisation (mrat). Given a directed graph $D = (V, F)$, find an *arc temporalisation* $\sigma : F \to \mathbb{N}$ such that the temporal reachability of the resulting temporal graph is maximized.

For example, let us consider the digraph shown in the left part of Figure 4.16. In the right part of the figure, we show an arc temporalisation of a digraph D with four nodes, such that the temporal reachability of the resulting temporal graph is equal to 16, which is clearly the maximum possible temporal reachability.

4.4.1 Hardness result

The next result shows that there is no polynomial-time algorithm solving the mrat problem, unless P is equal to NP. Following the approach used until here, we will work with a particular case of arc temporalisation, that is schedules, which consist in orderings of the arcs of the digraph. Indeed, one can easily transform an arc temporalisation σ into a schedule S_σ with greater or equal reachability. Given a digraph $D = (V, F)$ and an arc temporalisation σ, it is sufficient to consider the arcs $(u, v) \in F$ sorted by non-decreasing value of $\sigma((u, v))$ breaking ties arbitrarily to obtain such a schedule S_σ. For any two nodes u and v such that v is temporally reachable from u in the temporal graph $G[D, \sigma]$ induced by σ, v is also temporally reachable from u in the temporal graph $G[D, S_\sigma]$ induced by S_σ. This because the arcs corresponding to a temporal walk in $G[D, \sigma]$ are scheduled in order by S_σ.

Theorem 20. *The* mrat *problem is NP-hard, even if the digraph D is strongly connected.*

Proof. We reduce 3-sat to mrat as follows. Let us consider a 3-sat formula Φ, with n variables $x_1, \ldots, x_n$ and m clauses $c_1, \ldots, c_m$. Without loss of generality we will assume that each variable appears positive in at least one clause and negative in at least one clause. We first define the *unweighted* digraph $D = (V, F)$ as the union of the following gadgets (see Figure 4.17).

Variable gadgets For each variable x_i of Φ, V contains the nodes t_i^1, t_i^2, f_i^1,and f_i^2 and F contains the arcs (t_i^1, f_i^2), (f_i^2, f_i^1), (f_i^1, t_i^2), and (t_i^2, t_i^1).

Clause gadgets For each clause c_j, V contains the nodes c_j^1 and c_j^2. If the literal x_i appears in c_j, F contains the arcs (c_j^1, t_i^1) and (t_i^2, c_j^2), while if the literal $\neg x_i$ appears in c_j, F

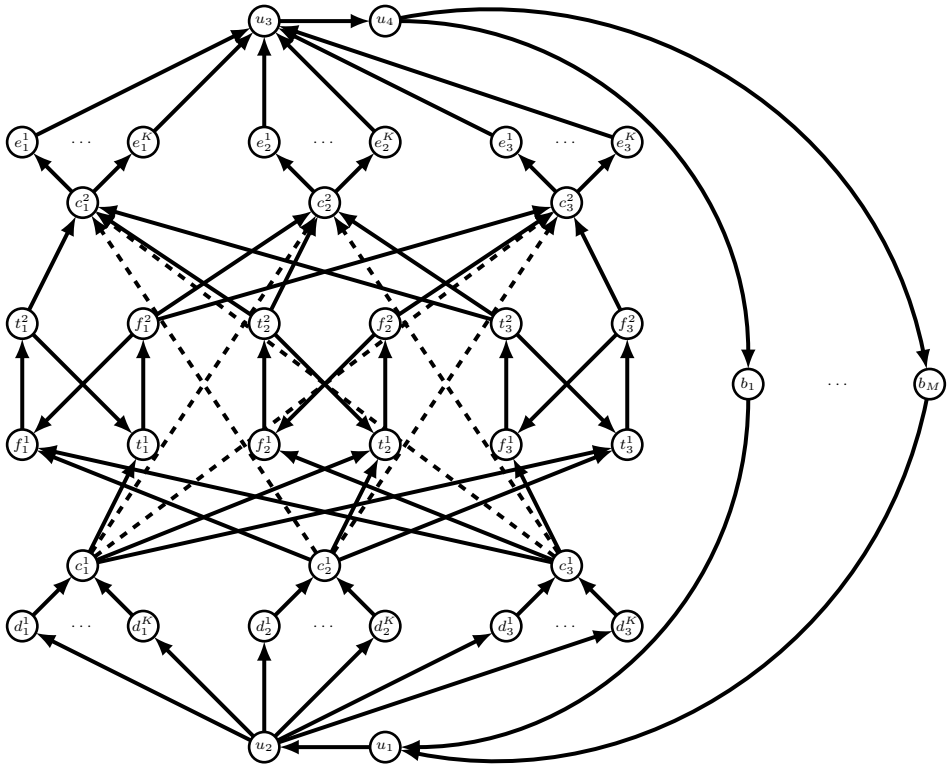

Figure 4.17: An example of the reduction of 3-SAT to MRAT. The 3-SAT formula is $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$.

contains the arcs (c_j^1, f_i^1) and (f_i^2, c_j^2). Moreover, for each two clauses c_h and c_j with $h \neq j$, F contains the arc (c_j^1, c_h^2) (see the dashed arcs in the figure). Finally, for each clause c_j, V also contains the nodes d_j^i and e_j^i, for $i \in [K]$ (the value of K will be specified later in the proof), and F contains the arcs (d_j^i, c_j^1) and (c_j^2, e_j^i), for $i \in [K]$.

Block gadget V contains the nodes u_1, u_2, u_3, and u_4, and the nodes b_i, for $i \in [X]$ (the value of X will be specified later in the proof). F contains the arcs $\{(b_i, u_1) : i \in [X]\}$, (u_1, u_2), $\{(u_2, d_j^l) : j \in [m], l \in [K]\}$, $\{(e_j^l, u_3) : j \in [m], l \in [K]\}$, (u_3, u_4), and $\{(u_4, b_i) : i \in [X]\}$.

Note that D is strongly connected. Indeed, let us consider the cycles

$$C_{i,j,l,p} = \langle u_1, u_2, d_j^l, c_j^1, t_i^1, f_i^2, f_i^1, t_i^2, c_j^2, e_j^l, u_3, u_4, b_p, u_1 \rangle,$$

where $j \in [m]$, i is such that x_i is a literal of the clause c_j, $l \in [K]$, and $p \in [X]$, and the cycles

$$C_{i,j,l,p} = \langle u_1, u_2, d_j^l, c_j^1, f_i^1, t_i^2, t_i^1, f_i^2, c_j^2, e_j^l, u_3, u_4, b_p, u_1 \rangle,$$

where $j \in [m]$, i is such that $\neg x_i$ is a literal of the clause c_j, $l \in [K]$, and $p \in [X]$. The union of these cycles contains each node in V, and each of these cycles contains node u_1. This proves the strong connectivity of D.

In the following, B denotes the set $\{b_i : i \in [X]\}$ and H denotes the set of nodes which do not belong to the block gadget, that is, $H = V \setminus (\{u_1, u_2, u_3, u_4\} \cup B)$ (note that $|V| = X + |H| + 4$ and that $|H| = 2(K+1)m + 4n$).

Activation of pairs of nodes in a variable gadget Consider a variable x_i and the associated variable gadget and a schedule S of the 4 arcs associated to the variable gadget. We say that S *activates* the pair (t_i^1, t_i^2) (respectively, (f_i^1, f_i^2)) if t_i^2 (respectively, f_i^2) is S-reachable from t_i^1 (respectively, f_i^1) within the gadget, that is (t_i^1, f_i^2), (f_i^2, f_i^1), and (f_i^1, t_i^2) (respectively, (f_i^1, t_i^2), (t_i^2, t_i^1), (t_i^1, f_i^2)) are scheduled in that order. Note that no schedule can activate both (t_i^1, t_i^2) and (f_i^1, f_i^2) as the arc (t_i^1, f_i^2) is scheduled either before or after the arc (f_i^1, t_i^2).

Constructing a schedule from a satisfying assignment. Suppose that there is an assignment α that satisfies Φ, and let us consider the following schedule S. First we schedule the arcs in $\{(b_i, u_1) : i \in [X]\}$ (in any arbitrary order), then the arc (u_1, u_2), and then the arcs in $\{(u_2, d_j^i) : j \in [m], i \in [K]\}$ (in any arbitrary order). Then we schedule the arcs $\{(d_j^i, c_j^1) : j \in [m], i \in [K]\}$ (in any arbitrary order), and, then, the arcs going out from the nodes c_j^1, for $j \in [m]$ (in any arbitrary order). Then, for each $i \in [n]$, if $\alpha(x_i) = \textsc{True}$, we schedule the arcs (t_i^1, f_i^2), (f_i^2, f_i^1), (f_i^1, t_i^2), and (t_i^2, t_i^1) in this order (thus activating (t_i^1, t_i^2)). Otherwise (that is, $\alpha(x_i) = \textsc{False}$), we schedule the arcs (f_i^1, t_i^2), (t_i^2, t_i^1), (t_i^1, f_i^2), and (f_i^2, f_i^1) in this order (thus activating (f_i^1, f_i^2)). Then we schedule all the arcs entering the nodes c_j^2, for $j \in [m]$ (in any arbitrary order), and then all the arcs going out from the nodes c_j^2, for $j \in [m]$ (in any arbitrary order). Finally, we schedule the arcs in $\{(e_j^l, u_3) : j \in [m], l \in [K]\}$ (in any arbitrary order), then the arc (u_3, u_4), and all the arcs in $\{(u_4, b_i) : i \in [X]\}$ (in any arbitrary order). Let G be the temporal graph induced by D and the schedule S.

First observe that, for any clause c_j with $j \in [m]$, there exists a literal that satisfies c_j according to the assignment α. Let x_i (respectively, $\neg x_i$) be a literal satisfying c_j. Since (t_i^1, t_i^2) (respectively, (f_i^1, f_i^2)) is activated, there exists a temporal path from c_j^1 to c_j^2 that goes through variable gadget corresponding to x_i. This means that $c_j^2 \in \mathcal{R}_G(c_j^1)$ and that, for $l, l' \in [K]$, $e_j^{l'} \in \mathcal{R}_G(d_j^l)$. We now prove a lower bound $\mathbb{L}$ on the S-reachability by showing a lower bound on the number of nodes temporally reachable from each possible source.

- For any $v \in V$ and for $i \in [X]$, $v \in \mathcal{R}_G(b_i)$. This adds $X(X + |H| + 4)$ to $\mathbb{L}$.
- For $i \in [4]$ and for $j \in [X]$, $b_j \in \mathcal{R}_G(u_i)$. Moreover, $\{u_1, u_2, u_3, u_4\} \cup H \subseteq \mathcal{R}_G(u_1)$, $\{u_2, u_3, u_4\} \cup H \subseteq \mathcal{R}_G(u_2)$, $u_3, u_4 \in \mathcal{R}_G(u_3)$, and $u_4 \in \mathcal{R}_G(u_4)$. This adds $4X + 2|H| + 10$ to $\mathbb{L}$.
- For $j, h \in [m]$, $i, l \in [K]$, and $p \in [X]$, $c_h^2, e_h^l \in \mathcal{R}_G(d_j^i)$ (because of the above observation) and $b_p \in \mathcal{R}_G(d_j^i)$. This adds $Km(X + Km + m)$ to $\mathbb{L}$.
- For $j, h \in [m]$, $l \in [K]$, and $i \in [X]$, $c_h^2, e_h^l, b_i \in \mathcal{R}_G(c_j^1)$. This adds $m(X + Km + m)$ to $\mathbb{L}$.
- For $i \in [n]$, there exists $j \in [m]$ such that c_j is satisfied by $\alpha(x_i)$. Hence, for $p \in [2]$, $l \in [K]$, and $h \in [X]$, $e_j^l, b_h \in \mathcal{R}_G(t_i^p)$ and $e_j^l, b_h \in \mathcal{R}_G(f_i^p)$. This adds $4n(X + K)$ to $\mathbb{L}$.

- For $j \in [m]$, $l \in [K]$, and $h \in [X]$, $e^l_j, b_h \in \mathcal{R}_G(c^2_j)$. This adds $m(X+K)$ to $\mathbb{L}$.
- For $j \in [m]$, $l \in [K]$, and $h \in [X]$, $b_h \in \mathcal{R}_G(e^l_j)$. This adds XKm to $\mathbb{L}$.

Thus, the S-reachability is at least

$$\begin{aligned}\mathbb{L} &= X(X+|H|+4)+(4X+2|H|+10)+Km(X+Km+m)\\ &\quad +m(X+Km+m)+4n(X+K)+m(X+K)+XKm.\end{aligned}$$

Bounding reachability when Φ is not satisfiable. Let us set X equal to any value greater than $(|H|+5)^2$. We now prove that, if there exists no truth-assignment satisfying the formula Φ, then no schedule S can have S-reachability greater than or equal to $\mathbb{L}$. First notice that if S assigns to the edge (u_3,u_4) a departure time smaller than the departure time assigned to (u_1,u_2), then the S-reachability is less than $\mathbb{L}$. This is because, in this case, for $i,j \in [X]$ with $i \neq j$, b_j is not S-reachable from b_i. Hence, the S-reachability is bounded by $\mathbb{U}_1 = X(|H|+4+1)+(|H|+4)(X+|H|+4)$: this would happen if, for each node $v \notin B$, $\mathcal{R}_G(v)=V$. Since $\mathbb{L} > X^2$, $\mathbb{U}_1 = X(|H|+4+1)+(|H|+4)(X+|H|+4) = 2X(|H|+4)+(|H|+4)^2+X < X(|H|+5)^2$, and $X > (|H|+5)^2$, it holds that $\mathbb{L} > \mathbb{U}_1$. We can then focus on schedules that assign to the arc (u_1,u_2) a departure time smaller than the departure time assigned to the arc (u_3,u_4). Let S be such a schedule and let G be the temporal graph induced by D and S. We now prove an upper bound $\mathbb{U}_2$ on the S-reachability by giving an upper bound on the nodes reachable from each possible source. Observe that, for any two nodes u and v, v might belong to $\mathcal{R}_G(u)$ only if in D there exists a path from u to v that does not include the arc (u_3,u_4) before the arc (u_1,u_2).

- For $i \in [X]$, $|\mathcal{R}_G(b_i)| \leq |V|$. This adds $X(X+|H|+4)$ to $\mathbb{U}_2$.
- For $i \in [2]$, $|\mathcal{R}_G(u_i)| \leq |V|$, while $|\mathcal{R}_G(u_3)| \leq X+3$ and $|\mathcal{R}_G(u_4)| \leq |V| = X+|H|+4$.
- For $j \in [m]$ and $i \in [K]$, in the best case $\mathcal{R}_G(d^i_j)$ contains d^i_j, c^1_j, the 12 nodes corresponding to the three variables appearing in c_j, and the nodes in $\{c^2_h : h \in [m]\} \cup \{e^l_h : h \in [m], l \in [K]\} \cup \{u_1,u_3,u_4\} \cup B$, yielding $|\mathcal{R}_G(d^i_j)| \leq X+Km+m+17$. However, we can show that there exists an index j^* such that, for $l,l' \in [K]$, $e^{l'}_{j^*} \notin \mathcal{R}_G(d^l_{j^*})$, implying that the d-nodes add at most $Km(X+Km+m+17)-K^2$ to $\mathbb{U}_2$. For defining j^*, we consider the following truth-assignment α: for any variable x_i with $i \in [n]$, $\alpha(x_i) = \textsc{True}$ if (t^1_i,f^2_i) is scheduled before (f^1_i,t^2_i), otherwise $\alpha(x_i) = \textsc{False}$. Note that if $\alpha(x_i) = \textsc{True}$ (respectively, $\alpha(x_i) = \textsc{False}$) we know that S does not activate (f^1_i,f^2_i) (respectively, (t^1_i,t^2_i)). Since the formula Φ is not satisfiable there exists $j^* \in [m]$ such that c_{j^*} is not satisfied by α. Let x_i (respectively, $\neg x_i$) be a literal in c_{j^*}. Since c_{j^*} is not satisfied by α, we that $\alpha(x_i) = \textsc{False}$ (respectively, $\alpha(x_i) = \textsc{True}$) and that (t^1_i,t^2_i) (respectively, (f^1_i,f^2_i)) is not activated. It is thus impossible to reach $c^2_{j^*}$ from $c^1_{j^*}$ through the variable gadget of x_i. On the other hand, in all the other walks in D that connect $c^1_{j^*}$ to $c^2_{j^*}$ the arc (u_3,u_4) appears before the arc (u_1,u_2). Hence, $c^2_{j^*} \notin \mathcal{R}_G(c^1_{j^*})$ and $e^{l'}_{j^*} \notin \mathcal{R}_G(d^l_{j^*})$ for $l,l' \in [K]$.
- For $j \in [m]$, in the best case $\mathcal{R}_G(c^1_j)$ contains c^1_j, the 12 nodes corresponding to the three variables appearing in clause c_j, and the nodes in $\{c^2_h : h \in [m]\} \cup \{e^l_h : h \in [m], l \in [K]\} \cup \{u_1,u_3,u_4\} \cup B$. This adds $m(X+Km+m+16)$ to $\mathbb{U}_2$.

- For $i \in [n]$ and for $j \in [2]$, in the best case $\mathcal{R}_G(t_i^j)$ and $\mathcal{R}_G(f_i^j)$ contain the corresponding four variable nodes and the nodes in $\{c_h^2 : h \in [m]\} \cup \{e_h^l : h \in [m], l \in [K]\} \cup \{u_1, u_3, u_4\} \cup B$. This adds $4n(X + Km + m + 7)$ to $\mathbb{U}_2$.
- For $j \in [m]$, in the best case $\mathcal{R}_G(c_j^2)$ contains c_j^2 and the nodes in $\{e_j^l : l \in [K]\} \cup \{u_1, u_3, u_4\} \cup B$. This adds $m(X + K + 4)$ to $\mathbb{U}_2$.
- For $j \in [m]$ and $i \in [K]$, in the best case $\mathcal{R}_G(e_j^i)$ contains e_j^i and the nodes in $\{u_1, u_3, u_4\} \cup B$. This adds with $Km(X + 4)$ to $\mathbb{U}_2$.

In summary,

$$\begin{aligned}\mathbb{U}_2 &= X(X + |H| + 4) + (4X + 3|H| + 15) + (Km(X + Km + m + 17) - K^2) \\ &\quad + m(X + Km + m + 16) + 4n(X + Km + m + 7) + m(X + K + 4) \\ &\quad + Km(X + 4).\end{aligned}$$

We have that $\mathbb{L} - \mathbb{U}_2 = -|H| - 5K^2 - 21Km - 4n(K(m-1) + m + 7) - 20m = K^2 - 23Km - 4n(K(m-1) + m + 8) - 22m - 5 > K^2 - Knm(23 + 4(1 + 1 + 8) + 22 + 5) = K^2 - 90Knm$ using $K, n, m \geq 1$. Let us set K equal to any value greater than or equal to $91nm$. We then have $K^2 > 90Knm$ and, thus, $\mathbb{L} > \mathbb{U}_2$. That is, the S-reachability has to be smaller than $\mathbb{L}$.

Conclusion. We have thus proved that the formula Φ is satisfiable if and only if there exists a schedule S such that the S-reachability of D is at least $\mathbb{L}$. This completes the proof of the theorem. □

4.4.2 Approximation

We have considered MRAT problem, that is, the problem of assigning departure times to the arcs of a digraph in order to maximize the total reachability of the resulting temporal graph. We have proved that this problem is NP-hard, even when the digraph is strongly connected. We conjecture that the MRAT problem can be approximated within a constant approximation ratio. In particular, we conjecture that any strongly connected digraph admits an arc temporalisation with temporal reachability at least equal to $c \cdot n^2$ for some constant $c > 0$. One way to prove such a statement would be to prove the following interesting graph theory conjecture.

Almost Spanning Two Rooted-Arborescences conjecture (ASTRA). *Any strongly connected digraph admits an out-arborescence and an in-arborescence that are arc-disjoint, have the same root, and each spans $\Omega(n)$ nodes.*

Note that it is not difficult to prove that the root of the two arborescences mentioned in the ASTRA conjecture cannot be any node in the graph. For example, let us consider the graph shown in Figure 4.18. In this case, the node x_1 cannot be the common root of the two arborescences, since the only in-arborescence and the only out-arborescence with root x_1 share the arc (x_2, y_2), so that one of the two arborescences cannot include more than one node (of course, this example can be generalised to any even number of nodes).

Note also that the ASTRA conjecture is false if we require that the two arborescences span at least $\frac{n}{3-\epsilon}$ nodes, for any positive constant ϵ. For example, consider the digraph $G = (V, F)$ shown in Figure 4.19, where, for some integer parameter $k > 0$, the set of nodes is $V = \{x, y, x_1, y_1, x_2, y_2, x_3, y_3\} \cup \{z_i^j \mid 1 \leq 1 \leq 3, 1 \leq j \leq k\}$, and the set of arcs is

$F = \{(x, y)\} \cup \{(y, x_i), (x_i, y_i), (y_i, x), (y_i, z_i^1), (z_i^k, x_i) \mid 1 \leq i \leq 3\} \cup \{(z_i^j, z_i^{j+1}) \mid 1 \leq i \leq 3, 1 \leq j < k\}$. Observe that the total number of nodes is $n = 3k + 8$. Let us first give an upper bound on the minimum between the amount of nodes in the in-arborescence and in the out-arborescence in the case where the root is not x nor y. For any such node, either the in-arborescence or the out-arborescence can contain at most $k + 3$ nodes, since the arc (x, y) can be in one arborescence only. Consider now the case in which either x or y is the root. Let us suppose that the root is x (the other case can be analysed in a similar way). Since, for each $i = 1, 2, 3$, the arc (x_i, y_i) can be in one arborescence only, then either the in-arborescence or the out-arborescence rooted at x can contain at most $n - 2k = k + 8$ nodes. We thus obtained that, in all cases, either the in-arborescence or the out-arborescence is upper bounded by $k + 8 = n/3 + O(1)$.

Bessy, Thomassé and Viennot [7] solved the ASTRA conjecture, proving it is possible to compute in $O(n^2)$ time an in-arborescence and an out-arborescence with the desired properties and both having size at least $n/6$. The idea to prove this result is to partition the set of vertices into three components I, C, O, such that C is a circuit (i.e. a closed walk with no arc repetition), there are no arcs from nodes in I to node in O, and $I \cup O$ and $O \cup C$ are balanced, meaning that both sets contain at least $n/3$ nodes. In order to get this kind of partition the authors exploit a particular kind of depth-first-search (DFS) tree which is left-maximal, which is a DFS tree such that the children of any node are ordered from left to right by non-increasing sub-tree size.

This result implies that the MRAT problem can be approximated within a factor 18.

4.5 Conclusions and open problems

In this chapter we studied the problem of assigning starting times to a set of walks in a digraph, turning the static graph into a temporal one. We focus on the reachability properties of the resulting temporal graph, with the main goal of maximising it. The digraph together with the set of walks, called trips, is named a trip network. We proved that even deciding whether exists a trip temporalisation that connects two given node in a trip network is NP-complete, even though it is FPT in the number of trips to reach the target from the source. This result leads to the fact that maximising the total number of pairs reachable, or the pairs reachable from a given source, is NP-hard and not even approximable. This pushed us to make further assumptions on the trip network, possibly reasonable from a public transit point of view, to find a setting in which the problem is tractable. The

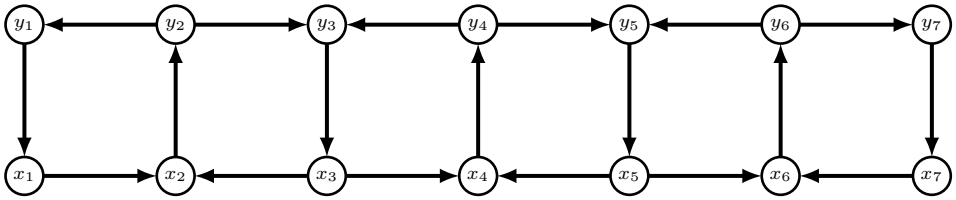

Figure 4.18: An example of a digraph for which only some nodes can be roots of two arborescences each spanning $\Omega(n)$ nodes.

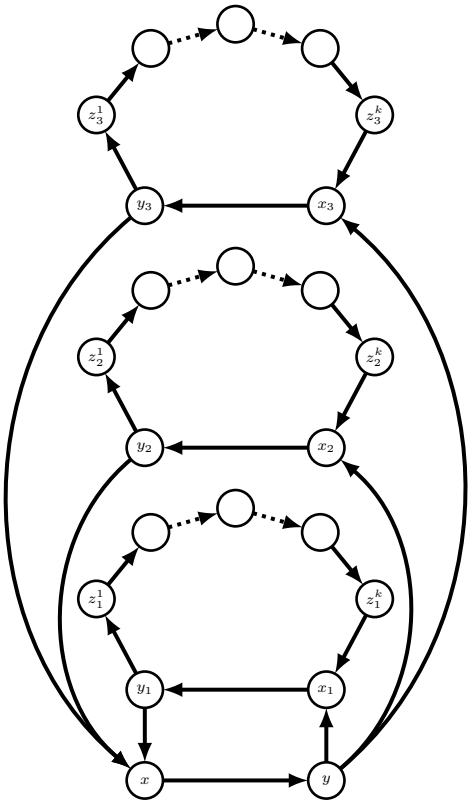

Figure 4.19: An example of a digraph for which there are no two arc-disjoint arborescences with a common root and each spanning more than $n/3+c$ nodes, for some positive constant c.

first assumption we made is that the trip network is strongly temporalisable, meaning that for each pair of node there exists a temporalisation that connects them. However, in this case the problem is still hard and not approximable within a reasonable factor. Thus, we introduced a second assumption, and we required that the trip network is symmetric as well. This means that for each trip there exists a symmetric one that visits the same nodes in opposite order. The problem of maximising the reachability is still NP-hard even with these restrictions. However, it is finally possible to approximate it within a constant factor in polynomial time.

Finally, deviating from the public transport application, we studied the case without the trip constraint. This means that a starting time can be assigned to each arc in the digraph independently from the other arcs. We proved a hardness result also in this case and conjectured a graph theory result that would imply approximability within a constant factor. This conjecture was later proved by Bessy et al. [7].

Open problems. From a theoretical point of view, it would be interesting to close the gap between Theorems 16 and 12 with regard to the inapproximability of MRTT in a strongly temporalisable network.

A challenging variant of the MRTT problem is obtained when the reachability of the induced temporal graph is measured within a given time window. In particular, when the time window is shorter than the sum of the weights of the arcs, temporalisations that are not schedules would be preferred. The FPT result for the O2O-RTT problem, Theorem 14, seems likely to still hold, even though some non-trivial adjustments are required. For example, the existence of a schedule that connects a source s to a destination t with a temporal walk using k trips, does not imply that there exists a temporalisation where (s, t) is connected in a given time window. This because the duration of the temporal walks is not taken into account in the O2O-RTT problem. However, given a temporal walk from s to t in a temporal graph induced by a schedule, it possible to produce a temporalisation that minimises the duration of the corresponding temporal walk. On the other hand, the approximation result, Theorem 19, cannot be easily translated into this setting. This leaves open whether it is possible to approximate reachability in the model.

Moreover, we proved an FPT results for the O2O-RTT problem, however it is still open to design an FPT algorithm for the general problem MRTT. In particular, we could define a different measure of reachability $\mathcal{R}^k_G(u)$ which consists in the nodes reachable from u using at most k trips. Then, the reachability of the temporal graph would be the sum of the cardinalities of these sets. Regarding MRAT, it would be interesting to understand if it is possible to close the gap on the arborescences size. We showed an example proving that upper bounds the size with $n/3$, and the result from Bessy et al. builds arborescences with size $n/6$. From a more applicative point of view, it would be worth exploring other restrictions of the problem where constant approximation is possible.

Finally, an interesting generalisation consists in allowing variable waiting times in-between two consecutive arcs of a trip. In other words, a temporalisation would not assign a departure time just to the first arc, but to each arc of a trip so that each induced temporal edge departs after the arrival time of the previous edge. In particular it would be interesting whenever the difference between the arrival time of an edge and the departure time of the next one is bounded.

Conclusions and perspectives

In this dissertation we explored part of two wide research topics in temporal graphs, temporal walk computation and temporal network design.

Our results on temporal walk computation are concentrated in the point availability model. In this context, we managed to complete the algorithmic picture for the computation of optimal temporal walks for a great variety of optimisation criteria. We achieved this result by designing efficient procedures that improve over sections of the state the art for both restricted and unrestricted waiting settings. We first designed a simple algorithm that computes reachability through temporal walks from a single source to all destinations in linear time and space with respect to the number of temporal edges. A second procedure, which exploits a core idea from the first one, finds the optimal cost of these temporal walks that connect the source to the destinations. The operations executed, more involved than the operations in the first one, can still be performed in linear time and space. As the algebraic definition of cost we give is very general, it is possible to model all classical criteria of optimisation. Finally, we designed a third algorithm to solve the same optimisation problem when all the hypothesis on zero travel times are dropped. In this case the complexity increases by a logarithmic factor.

We proved complexity lower bounds, shuttering any possibility of future theoretical improvements for this specific set of problems, as displayed in the summary of the state of the art about this topic in Section 1.2.4. This moves the focus of future lines of research about temporal path and walk computation towards either variations of these problems, for instance, all-sources all-destinations or multiple (disjoint) temporal walk, or towards models where temporal edges have proper intervals of availability rather then points. Computing fewest hops temporal paths in piecewise constant and piecewise linear temporal graphs could be a problem where there is still room for improvement. Moreover, in the linkstream model temporal edges have intervals of availability and it can be seen as a particular case of piecewise constant temporal graph as we defined it. In this model there is still room for improvement for the shortest duration and fewest hops temporal paths, either by designing algorithms that perform better than the ones working for the piecewise constant case, or by proving matching lower bounds. Furthermore, the restricted waiting setting is also open in this case. Indeed, the NP-hardness result for temporal walks by Orda and Rom [50], leverages on a large range of travel times for the temporal edges, while in linkstreams it is always zero. Notice that the setting where no waiting time at all is allowed is not particularly interesting, precisely due to the zero travel time. However, the case where the waiting time is lower and upper bounded in each node should be explored.

An interesting extension of temporal graphs that we did not consider in the thesis is the case in which, not only the edges, but also vertices are subject to temporal behaviour. For example, nodes could be available during certain intervals of time and unavailable during

some others. This would greatly impact all the problems we approached. Indeed, this implicitly leads to waiting restrictions, as it is not possible to wait in a node that is currently unavailable. A first step would be to understand if the problems that become hard with this new restrictions are the same ones that become hard introducing the waiting restrictions considered in this thesis. A second one would be to develop new efficient algorithms to solve problems subject to this restrictions. Moreover, this leads to interesting settings and variations also in the network temporsalisation problems. For example each node is given a certain amount of time in which it could be available, and the availability intervals of the nodes need to be assigned. Using this approach, temporsalisations that are not schedules would be preferred, raising new challenges.

As we pointed out in the dissertation, the representation chosen to give temporal graphs as input affects the time and space complexity of the algorithms. For the algorithms presented in Chapter 2 we adopted what we call a doubly-sorted representations, which consists in two lists of the temporal edges sorted according to certain temporal criteria. Working with this representation we obtained algorithms that perform in linear time and space, while we also proved that for certain problems a single list would not be sufficient. Moreover, we proved that this newly introduced representation is equivalent to the classical time-expanded representation. This raises a number of questions. For instance, which are the problems that could be solved in linear time with a single sorted list of temporal edges? The reachability problem could be a candidate. In temporal graph models higher in the hierarchy with respect to the point availability model, the representations become more involved. A future line of research could include designing new temporal graphs representations in these models to speed up pre-existing algorithms or develop procedures to exploit the new representations.

A setting that we did not explore in this dissertation is the probabilistic one. Temporal edges could be available at specific points in time or during intervals of time with a certain probability. Also, the travel time of the temporal edges could be a probabilistic function. Different assumptions on the probability space could lead to different problems that can be approached. This would be particularly interesting if studied in the public transit network model. In this case trips could have a certain probability of departing later, or certain connections can have different travel times based on the probability to have traffic jams or other scenarios. This can give a closer model to reality, as in public transport networks there are trips that are more likely to have delays compared to others and road segments that are more likely to be subjected to traffic jams or accidents. In this type of setting, we can try to compute temporal walks or journeys that are "robust", meaning that with a certain amount of probability they enable a user to go from a source to a destination earlier than a fixed time.

Among the many interesting possibilities in temporal network design we focused in particular on temporalisation problems. A static network is given as input and the task is to transform it into a temporal graph from scratch and, in our case, optimise some reachability measure. In this context we designed a novel problem called trip temporalisation, motivated by the application of scheduling in public transit networks. The problem consists in assigning starting time to trips in a digraph with the goal to maximise the number of temporally connected pairs in the resulting temporal graph. A restriction of the problem which consists in deciding whether there exists a temporalisation that temporally connects a given pair of nodes, turned out to be NP-complete. This pushed us tome make further assumptions on the trip network, and after several hardness results we managed to find a setting with reasonable assumptions that allows polynomial time approximation algorithms.

There are still challenging open ways to explore variations of this same problem. Introducing waiting times for the trips, bounding the number of trips used or studying the reachability inside a time window are natural first step enhancements, but they could greatly affect the problem. Moreover, it could be possible to find other assumptions that are meaningful from an application point of view that could lead to further and more promising approximation algorithms. Finally, adopting different measures of reachability could bring interesting variations.

On a more general note, several temporal network design problems have received far less attention than their static counterpart. Thus, there is still a broad collection of problems that can be studied in many different temporal graph models. Without going too far from the setting analysed in this dissertation, a different kind of operation compared to temporalisation could be addition or removal of temporal edges (or trips), and, instead of reachability, other notions that could be explored are temporal diameter or survivability.

Finally, it would be interesting to implement and test the procedures for temporal walk computation described in the thesis, and compare them to state of the art results. Moreover, from an application and implementation point of view, it would be appealing to study heuristics for the trip temporalisation problem, as the hardness results and low reachability example are obtained through ad hoc trip networks that are far from being realistic public transit networks.

Bibliography

[1] Noga Alon, Raphael Yuster, and Uri Zwick. Color coding. In *Encyclopedia of Algorithms*, pages 335–338. Springer, 2016. URL: https://doi.org/10.1007/978-1-4939-2864-4_76.

[2] Giorgio Ausiello, Alberto Marchetti-Spaccamela, Pierluigi Crescenzi, Giorgio Gambosi, Marco Protasi, and Viggo Kann. *Complexity and approximation: Combinatorial optimization problems and their approximability properties*. Springer, 1999. URL: https://link.springer.com/book/10.1007/978-3-642-58412-1.

[3] Alkida Balliu, Filippo Brunelli, Pierluigi Crescenzi, Dennis Olivetti, and Laurent Viennot. A note on the complexity of maximizing temporal reachability via edge temporalisation of directed graphs. *CoRR*, abs/2304.00817, 2023. arXiv:2304.00817, doi:10.48550/arXiv.2304.00817.

[4] Richard Bellman. The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60(6):503–516, 1954.

[5] Matthias Bentert, Anne-Sophie Himmel, André Nichterlein, and Rolf Niedermeier. Efficient computation of optimal temporal walks under waiting-time constraints. *Appl. Netw. Sci.*, 5(1):73, 2020. URL: https://doi.org/10.1007/s41109-020-00311-0.

[6] Kenneth A. Berman. Vulnerability of scheduled networks and a generalization of menger's theorem. *Networks*, 28(3):125–134, 1996.

[7] Stéphane Bessy, Stéphan Thomassé, and Laurent Viennot. Temporalizing digraphs via linear-size balanced bi-trees. *CoRR*, abs/2304.03567, 2023. arXiv:2304.03567, doi:10.48550/arXiv.2304.03567.

[8] Gerth Stølting Brodal and Riko Jacob. Time-dependent networks as models to achieve fast exact time-table queries. *Electron. Notes Theor. Comput. Sci.*, 92:3–15, 2004. URL: https://doi.org/10.1016/j.entcs.2003.12.019.

[9] Filippo Brunelli, Pierluigi Crescenzi, and Laurent Viennot. On computing pareto optimal paths in weighted time-dependent networks. *Inf. Process. Lett.*, 168:106086, 2021. URL: https://doi.org/10.1016/j.ipl.2020.106086.

[10] Filippo Brunelli, Pierluigi Crescenzi, and Laurent Viennot. Maximizing reachability in a temporal graph obtained by assigning starting times to a collection of walks. *Networks*, 81(2):177–203, 2023. doi:10.1002/net.22123.

[11] Filippo Brunelli and Laurent Viennot. To appear: Computing temporal reachability under waiting-time constraints in linear time. In David Doty and Paul Spirakis, editors, *2nd Symposium on Algorithmic Foundations of Dynamic Networks, SAND 2023, June 19-20, 2023, Pisa, Italy*.

[12] Filippo Brunelli and Laurent Viennot. Minimum-cost temporal walks under waiting-time constraints in linear time. *CoRR*, abs/2211.12136, 2022. URL: https://doi.org/10.48550/arXiv.2211.12136.

[13] Binh-Minh Bui-Xuan, Afonso Ferreira, and Aubin Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *International Journal of Foundations of Computer Science*, 14(02):267–285, 2003. URL: https://doi.org/10.1142/S0129054103001728.

[14] Richard T. Bumby. A problem with telephones. *SIAM. J. on Algebraic and Discrete Methods*, 2(1):13–18, 1981. URL: https://doi.org/10.1137/0602002.

[15] Sebastian Buß, Hendrik Molter, Rolf Niedermeier, and Maciej Rymar. Algorithmic aspects of temporal betweenness. In Rajesh Gupta, Yan Liu, Jiliang Tang, and B. Aditya Prakash, editors, *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, pages 2084–2092. ACM, 2020. URL: https://doi.org/10.1145/3394486.3403259.

[16] Valentina Cacchiani and Paolo Toth. Nominal and robust train timetabling problems. *Eur. J. Oper. Res.*, 219(3):727–737, 2012. URL: https://doi.org/10.1016/j.ejor.2011.11.003.

[17] Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *IJPEDS*, 27(5):387–408, 2012. URL: https://doi.org/10.1080/17445760.2012.668546.

[18] Arnaud Casteigts, Anne-Sophie Himmel, Hendrik Molter, and Philipp Zschoche. Finding temporal paths under waiting time constraints. *Algorithmica*, 83(9):2754–2802, 2021. URL: https://doi.org/10.1007/s00453-021-00831-w.

[19] Kenneth L. Cooke and Eric Halsey. The shortest route through a network with time-dependent internodal transit times. *Journal of Mathematical Analysis and Applications*, 14(3):493–498, 1966. URL: https://doi.org/10.1016/0022-247X(66)90009-6.

[20] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, chapter Single-Source Shortest Paths and All-Pairs Shortest Paths, page 580–642. MIT Press and McGraw-Hill, 2001. URL: http://mitpress.mit.edu/books/introduction-algorithms.

[21] Brian Dean. *Continuous-time dynamics shortest path algorithms*. PhD thesis, May 1999.

[22] Brian C Dean. Shortest Paths in FIFO Time-Dependent Networks: Theory and Algorithms. Technical report, MIT Department of Computer Science, 2004.

[23] Frank Dehne, Masoud T. Omran, and Jörg-Rüdiger Sack. Shortest Paths in Time-Dependent FIFO Networks. *Algorithmica*, 62(1-2):416–435, 2012. URL: https://doi.org/10.1007/s00453-010-9461-6.

[24] Argyrios Deligkas and Igor Potapov. Optimizing reachability sets in temporal graphs by delaying. In *AAAI*, pages 9810–9817, 2020. URL: https://doi.org/10.1016/j.ic.2022.104890.

[25] Daniel Delling, Thomas Pajor, and Renato F. Werneck. Round-Based Public Transit Routing. *Transportation Science*, 49(3):591–604, 2015. doi:10.1287/trsc.2014.0534.

[26] Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Intriguingly Simple and Fast Transit Routing. In *Experimental Algorithms*, Lecture Notes in Computer Science, pages 43–54. Springer, 2013. URL: https://doi.org/10.1007/978-3-642-38527-8_6.

[27] Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Connection scan algorithm. *ACM Journal of Experimental Algorithmics*, 23:1.7:1–1.7:56, 2018. URL: https://doi.org/10.1145/3274661.

[28] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959. URL: https://doi.org/10.1007/BF01386390.

[29] Stuart E. Dreyfus. An appraisal of some shortest-path algorithms. *Operations Research*, 17(3):395–412, 1969. URL: https://doi.org/10.1287/opre.17.3.395.

[30] Jessica A. Enright, Kitty Meeks, George B. Mertzios, and Viktor Zamaraev. Deleting edges to restrict the size of an epidemic in temporal networks. In *MFCS*, pages 57:1–57:15, 2019. URL: https://doi.org/10.4230/LIPIcs.MFCS.2019.57.

[31] Jessica A. Enright, Kitty Meeks, and Fiona Skerman. Assigning times to minimise reachability in temporal graphs. *Journal of Computer and System Sciences*, 115:169–186, 2021. URL: https://doi.org/10.1016/j.jcss.2020.08.001.

[32] Étienne Jules Marey. *La méthode graphique*. G. Masson éditeur, 1885.

[33] Afonso Ferreira and Laurent Viennot. A Note on Models, Algorithms, and Data Structures for Dynamic Communication Networks. Research Report RR-4403, INRIA, 2002. URL: https://inria.hal.science/inria-00072185.

[34] Lester R. Ford and Delbert R. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, New Jersey, 1962. URL: https://doi.org/10.1515/9781400875184.

[35] Luca Foschini, John Hershberger, and Subhash Suri. On the Complexity of Time-Dependent Shortest Paths. *Algorithmica*, 68(4):1075–1097, 2014. URL: https://doi.org/10.1007/s00453-012-9714-7.

[36] Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with 0(1) worst case access time. *J. ACM*, 31(3):538–544, 1984. URL: https://doi.org/10.1145/828.1884.

[37] Eugen Füchsle, Hendrik Molter, Rolf Niedermeier, and Malte Renken. Delay-robust routes in temporal graphs. In Petra Berenbrink and Benjamin Monmege, editors, *39th International Symposium on Theoretical Aspects of Computer Science, STACS 2022, March 15-18, 2022, Marseille, France (Virtual Conference)*, volume 219 of *LIPIcs*, pages 30:1–30:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. URL: https://doi.org/10.4230/LIPIcs.STACS.2022.30.

[38] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[39] F. Göbel, J. Orestes Cerdeira, and Henk Jan Veldman. Label-connected graphs and the gossip problem. *Discrete Mathematics*, 87(1):29–40, 1991. URL: https://doi.org/10.1016/0012-365X(91)90068-D.

[40] S. Mitchell Hedetniemi, S.T. Hedetniemi, and A.L. Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, 18(4):319–349, 1988. URL: https://doi.org/10.1002/net.3230180406.

[41] Arthur B. Kahn. Topological sorting of large networks. *Commun. ACM*, 5(11):558–562, 1962. URL: https://doi.org/10.1145/368996.369025.

[42] David Kempe, Jon Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. *Journal of Computer and System Sciences*, 64(4):820 – 842, 2002. URL: https://doi.org/10.1006/jcss.2002.1829.

[43] Emil Klafszky. Determination of shortest path in a network with time-dependent edge-lengths. *Mathematische Operationsforschung und Statistik*, 3(4):255–257, 1972. doi:10.1080/02331887208801081.

[44] Gueorgi Kossinets, Jon M. Kleinberg, and Duncan J. Watts. The structure of information pathways in a social communication network. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27, 2008*, pages 435–443, 2008. URL: https://doi.org/10.1145/1401890.1401945.

[45] Matthieu Latapy, Tiphaine Viard, and Clémence Magnien. Stream graphs and link streams for the modeling of interactions over time. *Social Netw. Analys. Mining*, 8(1):61:1–61:29, 2018. URL: https://doi.org/10.1007/s13278-018-0537-7.

[46] George B. Mertzios, Othon Michail, Ioannis Chatzigiannakis, and Paul G. Spirakis. Temporal network optimization subject to connectivity constraints. In *ICALP*, pages 657–668, 2013. URL: https://doi.org/10.1007/978-3-642-39212-2_57.

[47] George B. Mertzios, Othon Michail, and Paul G. Spirakis. Temporal network optimization subject to connectivity constraints. *Algorithmica*, 81(4):1416–1449, 2019. URL: https://doi.org/10.1007/s00453-018-0478-6.

[48] Hendrik Molter, Malte Renken, and Philipp Zschoche. Temporal reachability minimization: Delaying vs. deleting. In *MFCS*, volume 202 of *LIPIcs*, pages 76:1–76:15, 2021. URL: https://doi.org/10.4230/LIPIcs.MFCS.2021.76.

[49] Karl Nachtigall. Time depending shortest-path problems with applications to railway networks. *European Journal of Operational Research*, 83(1):154–166, 1995. URL: https://doi.org/10.1016/0377-2217(94)E0349-G.

[50] Ariel Orda and Raphael Rom. Traveling without waiting in time-dependent networks is np-hard. Technical report, Institute of Technology, Haifa, 1989.

[51] Ariel Orda and Raphael Rom. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. *J. ACM*, 37(3):607–625, 1990. doi:10.1145/79147.214078.

[52] Ariel Orda and Raphael Rom. Minimum weight paths in time-dependent networks. *Networks*, 21(3):295–319, 1991. URL: https://doi.org/10.1002/net.3230210304.

[53] Stefano Pallottino and Maria Grazia Scutellà. *Equilibrium and Advanced Transportation Modelling*, chapter Shortest path algorithms in transportation models: classical and innovative aspects, pages 245–281. Kluwer Academic Publishers, 1998. URL: https://doi.org/10.1007/978-1-4615-5757-9_11.

[54] Seth Pettie. A new approach to all-pairs shortest paths on real-weighted graphs. *Theor. Comput. Sci.*, 312(1):47–74, 2004. URL: https://doi.org/10.1016/S0304-3975(03)00402-X.

[55] Jeanette P. Schmidt and Alan Siegel. The spatial complexity of oblivious k-probe hash functions. *SIAM J. Comput.*, 19(5):775–786, 1990. URL: https://doi.org/10.1137/0219054.

[56] Frank Schulz, Dorothea Wagner, and Karsten Weihe. Dijkstra's algorithm on-line: An empirical case study from public railroad transport. *ACM J. Exp. Algorithmics*, 5:12, 2000. URL: https://doi.org/10.1145/351827.384254.

[57] Hanif D. Sherali, Kaan Özbay, and Shivaram Subramanian. The time-dependent shortest pair of disjoint paths problem: Complexity, models, and algorithms. *Networks*, 31(4):259–272, 1998.

[58] Frédéric Simard. Evaluating metrics in link streams. *Soc. Netw. Anal. Min.*, 11(1):51, 2021. URL: https://doi.org/10.1007/s13278-021-00759-7.

[59] João L. Sobrinho. An algebraic theory of dynamic network routing. *IEEE/ACM Trans. Netw.*, 13(5):1160–1173, 2005. URL: https://doi.org/10.1109/TNET.2005.857111.

[60] João L. Sobrinho and Timothy G. Griffin. Routing in equilibrium. *19th International Symposium on Mathematical Theory of Networks and System*, pages 941–947, 2010.

[61] Mikkel Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *J. ACM*, 46(3):362–394, 1999. URL: https://doi.org/10.1145/316542.316548.

[62] Juan Villacis-Llobet, Binh-Minh Bui-Xuan, and Maria Potop-Butucaru. Foremost non-stop journey arrival in linear time. In Merav Parter, editor, *Structural Information and Communication Complexity - 29th International Colloquium, SIROCCO 2022, Paderborn, Germany, June 27-29, 2022, Proceedings*, volume 13298 of *Lecture Notes in Computer Science*, pages 283–301. Springer, 2022. doi:10.1007/978-3-031-09993-9_16.

[63] Huanhuan Wu, James Cheng, Silu Huang, Yiping Ke, Yi Lu, and Yanyan Xu. Path Problems in Temporal Graphs. *VLDB Endowment*, 7(9):721–732, 2014. URL: https://doi.org/10.14778/2732939.2732945.

[64] Huanhuan Wu, James Cheng, Yiping Ke, Silu Huang, Yuzhen Huang, and Hejun Wu. Efficient Algorithms for Temporal Path Computation. *IEEE Transactions on Knowledge and Data Engineering*, 28:2927–2942, 2016. URL: https://doi.org/10.1109/TKDE.2016.2594065.

[65] Philipp Zschoche, Till Fluschnik, Hendrik Molter, and Rolf Niedermeier. The complexity of finding small separators in temporal graphs. *J. Comput. Syst. Sci.*, 107:72–92, 2020. URL: https://doi.org/10.1016/j.jcss.2019.07.006.

[66] Uri Zwick. Exact and approximate distances in graphs - A survey. In Friedhelm Meyer auf der Heide, editor, *Algorithms - ESA 2001, 9th Annual European Symposium, Aarhus, Denmark, August 28-31, 2001, Proceedings*, volume 2161 of *Lecture Notes in Computer Science*, pages 33–48. Springer, 2001. URL: https://doi.org/10.1007/3-540-44676-1_3.