Generalized Visual Fog Scheduling

Abstract

Generalized visual fog scheduling is the problem of assigning the visual computing tasks to various devices to optimize network utilization over time. The *fog* paradigm envisions heterogeneous devices distributed across the Internet that inter-dependent *visual computing* tasks can be deployed onto with reduced network bandwidth requirements. The ever-changing nature in device connectivity and task dependency in visual computing necessitates *online* solutions to schedule tasks over devices with short turnaround time. This paper formulates the *generalized visual fog scheduling* problem in dynamic graphs representing task dependencies and device connectivities over time, and proposes novel online solutions based on the theoretic foundation of online linear programming with polynomial time complexity. The ample experimental results show that our approach is feasible and robust against partial update ordering, and ready to meet the dynamicity challenges introduced by visual fog.

1 Introduction

With over 70 exabytes of live video expected to be flowing over the Internet by 2021, video analytics is becoming an increasingly important source for deriving business insights in various use cases like digital surveillance, retail analytics, factory automation and smart city. It is forecasted that the amount of data generated by video surveillance cameras installed globally will be more than 859 petabytes per day by 2017 [IHS, 2015]. Despite its abundance and ubiquity, visual data is still significantly underutilized due to the computation requirements and the outstanding volume of the collected data. While today's solutions have made advances in converting video data into actionable information at scale [Lu, Chowdhery, and Kandula, 2016], they are often limited to the use of cloud infrastructure and with fixed function capability, on a per use case basis.

Given the limited bandwidth of the Internet, it is not feasible to send all video streams from the data sources to the cloud [Botta et al., 2016a; Yang, Tickoo, and Chen, 2017]. While this presents a problem in using cloud infrastructure alone, it is also impractical to rely solely on edge devices, due to the significant computation and resource expectations of visual data. Edge computing has the additional drawback of not being economically scalable, as having on-premise servers translates to recurring expenses to the users. The fog paradigm uses collaborative devices from edge to cloud for improved resource management and network bandwidth utilization [Bonomi et al., 2012; Botta et al., 2016b]. In the context of video analytics, visual fog computing uses the fog paradigm for scaling out visual processing with automatic allocation of workloads, allowing for scaling out deployments in terms of the number of concurrent video streams [Yang, Tickoo, and Chen, 2017; Chu et al., 2018].

However, the prior solutions assumed tasks and their dependency, and devices and their connectivity remain static over time, and hence offline optimal schedulers are derived. In real world settings, task dependency and device connectivity can change over time rather frequently due to the resource sharing nature of deployments and the fact of multitenancy [Li et al., 2011; Liu, Quan, and Ren, 2011; Sahni and Vidyarthi, 2018]. In this paper, we formulate the problem of generalized visual fog scheduling in dynamic graphs [Demetrescu et al., 2010] and propose novel online solutions based on the theoretical foundation of linear programming. The experiment results showed that our formulation is feasible and effective with minimal performance degradation. Our online method is also robust against partial update ordering, which is essential for online methods to deal with unanticipated changes in tasks or their dependency, or devices or their connectivity.

2 Problem Formulation

Visual fog scheduling is the problem of assigning inter-dependent tasks to inter-connected devices with device constraints and network constraints; both of them are represented as dynamic graphs whose topology can change over time. The respective optimization problem is to find a valid solution given a predetermined objective. Hereinafter, we use minimal total bandwidth utilization as an example and it is straightforward to generalize to any arbitrary quantitatively measurable metric. In this section, we start with definition of graphs representing device connectivity and task dependency followed by formalizing the scheduling problem `VFC` and the online scheduling problem `D-VFC`.

Dynamic graphs are graphs which are subject to a sequence of discrete updates [Demetrescu et al., 2010], which is also known as temporal graphs [Michail, 2016], evolving graphs [Ferreira, 2004] or time-varying graphs [Casteigts et al., 2012]. An update can be any arbitrary operation associated with edge or vertices including addition, deletion and change of attributes. In the context of visual fog scheduling, we model addition and removal of devices and tasks as updates on graphs representing device connectivity and task dependency, respectively. The goal of deriving scheduling algorithms on dynamic graphs is to find a solution of a problem based on an update and the prior solution, where possible, instead of having to

recompute it from scratch every time.

Device graph is a dynamic graph $G_{\mathcal{D}}$ which can be expanded into $\{G_{\mathcal{D}}^{[t]}\}_{t\in T}$ where $G_{\mathcal{D}}^{[t]} = (V_{\mathcal{D}}^{[t]}, E_{\mathcal{D}}^{[t]})$ denotes the static expansion of $G_{\mathcal{D}}$ at time $t \in T \subset \mathbb{N}$. Each $v \in V_{\mathcal{D}}^{[t]}$ represents a *device* at time t and each edge $(u, v) \in E_{\mathcal{D}}$ represents the *connectivity* between device u and device v at time t. Specifically, $v \in V_{\mathcal{D}}^{[t]}$ is a source device if $\deg_+^{[t]}(v) = 0$ at time t, and is a sink device if $\deg_-^{[t]}(v) = 0$ at time t, where $\deg_+^{[t]}(\cdot)$ and $\deg_-^{[t]}(\cdot)$ denote in-degree and out-degree of a vertex at time t, respectively. We denote $V_{\mathcal{D}}^{[t]}+$ and $V_{\mathcal{D}}^{[t]}-$ as the collections of source devices and sink devices at time t, respectively, and $E_{\mathcal{D}}^{[t]}+$ and $E_{\mathcal{D}}^{[t]}-$ as the collections of edges start with a source device and end with a sink device at time t, respectively.

Task graph is a dynamic graph $G_{\mathcal{J}}$ which can be expanded into $\{G_{\mathcal{J}}^{[t]}\}_{t\in T}$ where $G_{\mathcal{J}}^{[t]} = (V_{\mathcal{J}}^{[t]}, E_{\mathcal{J}}^{[t]}) = \cup_{j=1}^{J^{[t]}} G_{\mathcal{J}}^{[t],j}$ where $J^{[t]}$ is the number of *disjoint* subgraphs at time t, each of which represent a job. Each subgraph $G_{\mathcal{J}}^{[t],j} = (V_{\mathcal{J}}^{[t],j}, E_{\mathcal{J}}^{[t],j})$ is a linear (or path) graph having exact one (entry) producer task and one (exit) consumer task at time $t \in T \subset \mathbb{N}$. Any intermediate tasks are neither producer nor consumer tasks; namely, $|\{\deg_+^{[t]}(v) = 0 | v \in V_{\mathcal{J}}^{[t],j}\}| = |\{\deg_-^{[t]}(v) = 0 | v \in V_{\mathcal{J}}^{[t],j}\}| = 1$. Each $v \in V_{\mathcal{J}}^{[t]}$ represents a *task* at time t and $(u, v) \in E_{\mathcal{J}}^{[t]}$ represents the *dependency* between task u and task v at time t. Specifically, $v \in V_{\mathcal{J}}^{[t]}$ is a producer task at time t if $\deg_+^{[t]}(v) = 0$, and is a consumer task at time t if $\deg_-^{[t]}(v) = 0$. We denote $V_{\mathcal{J}}^{[t]}+$ and $V_{\mathcal{J}}^{[t]}-$ as the collections of producer and consumer tasks at time t, respectively, and $E_{\mathcal{J}}^{[t]}+$ and $E_{\mathcal{J}}^{[t]}-$ as the collections of edges start with a producer task and end with a consumer task at time t, respectively. Hereinafter, we may omit the mention of time t when it is implicit but apparent for space consideration.

The mappings of producer tasks and consumer tasks are predetermined, which can be expressed as $\tilde{g}+ = \{\tilde{g}^{[t]}+\}_{t\in T}$ and $\tilde{g}- = \{\tilde{g}^{[t]}-\}_{t\in T}$ where $\tilde{g}^{[t]}+ : V_{\mathcal{J}}^{[t]}+ \to V_{\mathcal{D}}^{[t]}+$ and $\tilde{g}^{[t]}- : V_{\mathcal{J}}^{[t]}- \to V_{\mathcal{D}}^{[t]}-$ for all time t, respectively. Then, the collection of VFC instances defined on $G_{\mathcal{D}}$ and $G_{\mathcal{J}}$ can be expressed as follows:

$$\left\{\left(G_{\mathcal{D}}^{[t]}, G_{\mathcal{J}}^{[t]}, \tilde{g}^{[t]}+, \tilde{g}^{[t]}-, c_V^{[t]}, c_E^{[t]}, r_V^{[t]}, r_E^{[t]}\right)\right\}_{t\in T} \quad (1)$$

where $c_V = \{c_V^{[t]}\}_{t\in T}$ and $c_E = \{c_E^{[t]}\}_{t\in T}$ are device capacity functions over $G_{\mathcal{D}}$, and $r_V = \{r_V^{[t]}\}_{t\in T}$ and $r_E = \{r_E^{[t]}\}_{t\in T}$ are task requirement functions over $G_{\mathcal{J}}$; each of $c_V^{[t]}$, $c_E^{[t]}$, $r_V^{[t]}$ and $r_E^{[t]}$ is a function mapping vertices or edges in $G_{\mathcal{D}}^{[t]}$ or $G_{\mathcal{J}}^{[t]}$ to $\mathbb{R}$.

2.1 The Scheduling Problem

A *schedule* maps an edge in $G_{\mathcal{J}}$ to a simple path, hereinafter, we may refer to simple path as path, in arbitrary lengths in $G_{\mathcal{D}}$, which can be expressed as:

$$f : E_{\mathcal{J}} \times T \to \bigcup_{k=0}^{|V_{\mathcal{D}}|} (v_i)_{i=1}^{k} \quad (2)$$

where $(v_i)_{i=1}^{k}$ represents an arbitrary length-(k-1) path at certain time $t \in T$. A schedule f implies vertex-to-vertex mapping. Specifically, the edge-to-path mapping of a schedule f determines the correspondence between the two endpoints of an edge in the task graph and the two endpoints of a path in the device graph. To facilitate formulation, we define *task schedule* as:

$$g : V_{\mathcal{J}} \times T \to V_{\mathcal{D}} \quad (3)$$

that maps tasks to devices at time $t \in T$. To facilitate the description, we define partial order relation $\preceq_{\mathcal{J}}^{[t]}$ over task graph $G_{\mathcal{J}}$, and $\preceq_{\mathcal{D}}^{[t]}$ over device graph $G_{\mathcal{D}}$, which can be expressed as:

$$u \preceq_{\mathcal{J}}^{[t]} v \iff v \text{ is reachable from } u, \forall u, v \in V_{\mathcal{J}}^{[t]} \quad (4)$$

$$u \preceq_{\mathcal{D}}^{[t]} v \iff v \text{ is reachable from } u, \forall u, v \in V_{\mathcal{D}}^{[t]} \quad (5)$$

Task schedule g implies the *single choice constraint* of mapping from task to devices; in addition, the *ordering constraint* ensures the ordering of tasks along a job in $G_{\mathcal{J}}^{[t]}$ must remain in their mapped devices in $G_{\mathcal{D}}^{[t]}$, which can be expressed as follows:

$$u \preceq_{\mathcal{J}}^{[t]} v \Longrightarrow g(u) \preceq_{\mathcal{D}}^{[t]} g(v) \quad (6)$$

Note that task schedule g is not a dual function of schedule f, and, by definition, $\tilde{g}+, \tilde{g}- \subseteq g$. Specifically, there may exist multiple paths between any arbitrary pairs of devices in $G_{\mathcal{D}}^{[t]}$; a schedule selects not only a collection of devices to run a sequence of tasks but also a collection of links between devices for data transmission among tasks.

The single choice constraint and ordering constraint, in combination, define a *valid* schedule. A *feasible* schedule is, further, subject to *device and network constraints*. An *optimal* schedule is a feasible schedule that minimizes an *objective*. Throughout this paper, we will use overall network utilization as the objective, while extension to other objectives, e.g., overall device resource utilization is straightforward. The network utilization is the total network used given a particular schedule f.

Recall that f defines an edge-to-path mapping from an edge $e = (u_1, u_2) \in G_{\mathcal{J}}$ to an arbitrary length path $(v_1, v_2, \cdots, v_k)$ in $G_{\mathcal{D}}$. This manifests the nature of data transmission in VFC. Specifically, if the length of path $f(e, t)$ is zero, i.e., $k = 1$, $r_E(e)$ doesn't implicate any links in $G_{\mathcal{D}}$ since $f(e, t)$ doesn't contain any edge in $G_{\mathcal{D}}$. In other words, data transmission within a device is essentially omitted since inter-process or inter-thread communication is much faster than inter-device communication. Otherwise, if the length of path $f(e, t)$ is greater than or equal to one, i.e., $k \geq 2$, it implies task u_1 and u_2 run on v_1 and v_k, respectively, i.e., $g(u_1, t) = v_1$ and $g(u_2, t) = v_k$, and the link requirement $r_E(e)$ remains constant on all edges $\{(v_i, v_{i+1})\}_{i=1}^{k-1}$ along path $f(e, t)$. To facilitate description, we define $f^{[t]}(\cdot) = f(\cdot, t)$ and $g^{[t]}(\cdot) = g(\cdot, t)$, and may use the notions interchangeably hereinafter.

The device constraint and the network constraint can be

expressed, respectively, as follows:

$$\left(\sum_{v\in V_{\mathcal{J}}} [\![u = g^{[t]}(v)]\!] r_V^{[t]}(v)\right) \leq c_V^{[t]}(u),\ \forall u \in V_{\mathcal{D}}^{[t]}, \forall t \in T \tag{7}$$

$$\left(\sum_{e\in E_{\mathcal{J}}} [\![d \in f^{[t]}(e)]\!] r_E^{[t]}(e)\right) \leq c_E^{[t]}(e),\ \forall d \in E_{\mathcal{D}}^{[t]}, \forall t \in T \tag{8}$$

where Eq. 8 implies that when multiple consecutive tasks mapped to one device, the network overhead do not add to any links between devices. For simplicity, we assume homogeneous resource type upon device and link throughout this paper. It is straightforward to generalize the VFC formulation to compound resource types; each device resource type defines a separate device capacity function c_V over $V_{\mathcal{D}}$ over time t and task requirement function r_V over $V_{\mathcal{J}}$ over time t.

2.2 The Online Scheduling Problem

Let $\lambda : T \to (\Upsilon+, \Upsilon-, \Xi+, \Xi-)$ denote update over a graph over time where $\Upsilon+$ and $\Xi+$ indicate addition of vertices and edges, respectively, and $\Upsilon-$ and $\Xi-$ represent vertices and edges to be removed, respectively. We define operator $\otimes$: $G \times \lambda \to G$ where $G_t \otimes \lambda(t) \to G_{t+1}$ for all time t; essentially, $G^{[t+1]} = ((V^{[t]} \cup \Upsilon^{[t]}+) \setminus \Upsilon^{[t]}-, (E^{[t]} \cup \Xi^{[t]}+) \setminus \Xi^{[t]}-)$ where $\lambda(t) = (\Upsilon^{[t]}+, \Upsilon^{[t]}-, \Xi^{[t]}+, \Xi^{[t]}-)$. We define $\lambda_{\mathcal{D}}$ and $\lambda_{\mathcal{J}}$ as updates on device graph $G_{\mathcal{D}}$ and task graph $G_{\mathcal{J}}$, respectively.

The online scheduling problem D-VFC aims to derive a strategy of solving $f^{[t+1]}$ based upon $f^{[t]}$ and the updates $\lambda_{\mathcal{D}}(t)$ and $\lambda_{\mathcal{J}}(t)$ between $G_{\mathcal{D}}^{[t]}$ and $G_{\mathcal{D}}^{[t+1]}$, and $G_{\mathcal{J}}^{[t]}$ and $G_{\mathcal{J}}^{[t+1]}$, respectively. Hence, a D-VFC instance can be represented as follows:

$$\Big(f^{[t]}, G_{\mathcal{D}}^{[t]}, G_{\mathcal{J}}^{[t]}, \lambda_{\mathcal{D}}(t), \lambda_{\mathcal{J}}(t), \tilde{g}^{[t+1]}+, \tilde{g}^{[t+1]}-, c_V^{[t+1]}, c_E^{[t+1]}, r_V^{[t+1]}, r_E^{[t+1]}\Big) \tag{9}$$

In other words, to schedule for time t+1, addition information of the prior schedule $f^{[t]}$ and the updates $\lambda_{\mathcal{D}}(t)$ and $\lambda_{\mathcal{J}}(t)$ over prior graphs $G_{\mathcal{D}}^{[t]}$ and $G_{\mathcal{J}}^{[t]}$ can be explicitly considered, as opposed to scheduling from scratch, where a feasible schedule should meet the ordering constraint, the device constraint and the network constraint as defined in Eqs. 6–8.

2.3 The Optimization Problem

The optimization problem of VFC or D-VFC is to find a feasible $f^\star$ that minimizes an objective. In this paper, we defined the objective as minimizing total network utilization, which can be expressed as follows.

$$\sum_{d\in E_{\mathcal{D}}^{[t]}} \sum_{e\in E_{\mathcal{J}}^{[t]}} [\![d \in f^{[t]}(e)]\!] r_E^{[t]}(e) \tag{10}$$

However, it is straightforward to extend to others, like minimizing sink device resource utilization, maximizing source device resource utilization or minimizing overall device resource utilization.

3 Proposed Algorithms

In this section, we start with formulating the scheduling problem (VFC) in integer linear programming (ILP), and, then, introduce our solution for the online scheduling problem (D-VFC) with novel formulations in linear programming (LP).

Integer linear programming is a non-convex optimization problem that had been well-studied in the literature [Dantzig, Orden, and Wolfe, 1955; Schrijver, 1998]. We describe the well-known 0-1 (or binary) integer linear programming, which will be used for formulating VFC. The ILP instance corresponding to determining the schedule f can be expressed as:

$$\begin{aligned}
&\text{minimize} && \mathbf{c}^\top \mathbf{x} && (11)\\
&\text{subject to} && \mathbf{A}\mathbf{x} \leq \mathbf{b} && (12)\\
& && \mathbf{C}\mathbf{x} = \mathbf{d} && (13)\\
&\text{and} && \mathbf{x}_\ell \in \{0,1\}^L && (14)
\end{aligned}$$

where L is the length of $\mathbf{x}$, Eq. 11 is the objective term, Eq. 12 is the inequality constraint, Eq. 13 is the equality constraint and Eq. 14 is the binary constraint. We reduce VFC to ILP in the following form:

- $\mathbf{x}$ to present a schedule f
- Objective (Eq. 11) to present total network utilization (Eq. 10)
- Constraints (Eq. 12–13) to present the ordering constraint (Eq. 6), device and network constraints (Eqs. 7–8), and the implicit single choice constraint

3.1 Solving VFC using ILP

We first establish the baseline of solving VFC using ILP [Chu et al., 2018] by representing the *reverse* schedule $h^{[t]}$ of $f^{[t]}$ in $\mathbf{x}$. A reverse schedule $h^{[t]}$ defines the reverse mapping between links in device graph $E_{\mathcal{D}}^{[t]}$ and dependencies in task graph $E_{\mathcal{J}}^{[t]}$ at time t. In particular, $h^{[t]}$ maps $E_{\mathcal{D}}^{[t]}$ to $2^{E_{\mathcal{J}}^{[t]}}$, the power set of $E_{\mathcal{J}}^{[t]}$, which can be expressed as:

$$h^{[t]} = E_{\mathcal{D}}^{[t]} \to 2^{E_{\mathcal{J}}^{[t]}} \tag{15}$$

The above implies that $d \in f^{[t]}(e) \Leftrightarrow e \in h^{[t]}(d)$. Additionally, $h^{[t]}$ can be represented as a collection of reverse sub-schedule $h^{[t],j}$, who map a device back to a task in a per job manner:

$$h^{[t],j} : E_{\mathcal{D}}^{[t]} \to \{E_{\mathcal{J}}^{[t],j} \cup \phi\} \qquad 1 \leq j \leq J^{[t]} \tag{16}$$

$$h^{[t]}(d) = \cup_{j=1}^{J} h^{[t],j}(d) \qquad \forall d \in E_{\mathcal{D}}^{[t]} \tag{17}$$

Here $J^{[t]}$ is the number of jobs (disjoint subgraphs) in $G_{\mathcal{J}}^{[t]}$. Grounded on the above definition, $\mathbf{x}$ can be defined as follows

$$\mathbf{x} = \{x_{d,e}^{[t],j} | d \in E_{\mathcal{D}}^{[t]}, e \in E_{\mathcal{J}}^{[t],j}\}_{j=1}^{J}, \tag{18}$$

$$x_{d,e}^{[t],j} = 1 \Leftrightarrow h^{[t],j}(d) = e$$

That is, the binary variable $x_{d,e}^{[t],j} = 1$ indicates that device connection d is used by task dependency e in job j at time t. Consequently, the ILP objective (Eq. 11) can be derived as

$$\mathbf{c}^\top \mathbf{x} = \sum_{j=1}^{J^{[t]}} \sum_{d\in E_{\mathcal{D}}^{[t]}} \sum_{e\in E_{\mathcal{J}}^{[t],j}} r_E(e) x_{d,e}^{[t],j} \tag{19}$$

The Single Choice Constraint The single choice constraint represents flow conservation between a pair of a source device and a sink device so that each job is processed and transmitted along *exactly one* path between the designated source and sink device. A device has at most one out-edge mapped by $f^{[t]}$ for

each job, which can be expressed as:

$$\sum_{\substack{d+=u \\ d\in E_{\mathcal{D}}^{[t]}}} \sum_{e\in E_{\mathcal{J}}^{[t],j}} x_{d,e}^{[t],j} \leq 1,\ \forall u \in V_{\mathcal{D}}^{[t]} \tag{20}$$

where $\cdot+$ and $\cdot-$ denote the tail and the head of an arbitrary edge; namely, $e+ = u$ and $e- = v$ if $e = (u, v)$. In addition, a source device in $V_{\mathcal{D}}^{[t]}+$ and a sink device in $V_{\mathcal{D}}^{[t]}-$ must have exact one out-edge and one in-edge, respectively. Such condition can be formulated as follows:

$$\sum_{\substack{d+\in V_{\mathcal{D}}^{[t]}+ \\ d\in E_{\mathcal{D}}^{[t]}}} \sum_{e\in E_{\mathcal{J}}^{[t],j}} x_{d,e}^{[t],j} = \sum_{\substack{d-\in V_{\mathcal{D}}^{[t]}- \\ d\in E_{\mathcal{D}}^{[t]}}} \sum_{e\in E_{\mathcal{J}}^{[t],j}} x_{d,e}^{[t],j} = 1 \tag{21}$$

On the other hand, every intermediate device $u \in V_{\mathcal{D}} \setminus (V_{\mathcal{D}}^{[t]}+ \cup V_{\mathcal{D}}^{[t]}-)$, which is neither a source device nor a sink device, must either have one in-edge and one out-edge or have none of them, which can be formalized as:

$$\sum_{e\in E_{\mathcal{J}}^{[t],j}} \left(\sum_{\substack{d+=u \\ d\in E_{\mathcal{D}}^{[t]}}} x_{d,e}^{[t],j} - \sum_{\substack{d-=u \\ d\in E_{\mathcal{D}}^{[t]}}} x_{d,e}^{[t],j} \right) = 0 \tag{22}$$

The Ordering Constraint The ordering constraint ensures a job must be mapped in order along a path in the device graph as formalized in Eq. 6. The ordering constraint, under the context of the reverse scheduling $h^{[t]}$, can be expressed as:

$$\sum_{\substack{d-=u \\ d\in E_{\mathcal{D}}^{[t]}}} \sum_{e\in E_{\mathcal{J}}^{[t],j}} \mathbf{I}(e) x_{d,e}^{[t],j} \leq \sum_{\substack{d+=u \\ d\in E_{\mathcal{D}}^{[t]}}} \sum_{e\in E_{\mathcal{J}}^{[t],j}} \mathbf{I}(e) x_{d,e}^{[t],j}, \forall u \in V_{\mathcal{D}}^{[t]} \tag{23}$$

where $\mathbf{I} : E_{\mathcal{J}}^{[t]} \to \mathbb{R}$ is an arbitrary function that satisfies the property $e_1^+ \preceq_{\mathcal{J}}^{[t]} e_2^+ \Leftrightarrow e_1^- \preceq_{\mathcal{J}}^{[t]} e_2^- \Rightarrow \mathbf{I}(e_1) \leq \mathbf{I}(e_2)$ for all $e_1, e_2 \in E_{\mathcal{J}}^{[t]}$. That is, usage of $\mathbf{I}$ with such property guarantees Eq. 6 to hold by preserving the total order of $G_{\mathcal{J}}^{[t],j}$.

The Device & Network Constraints The network constraint, resembling Eq. 8, is straightforward and can be formulated as follows:

$$\left(\sum_{j=1}^{J^{[t]}} \sum_{e\in E_{\mathcal{J}}^{[t],j}} r_E^{[t]}(e) x_{d,e}^{[t],j} \right) \leq c_E^{[t]}(d), \forall d \in E_{\mathcal{D}}^{[t]} \tag{24}$$

To formalize the device constraint, the aggregated requirement function $\mathbf{R}^{[t]} : E_{\mathcal{J}}^{[t]} \to \mathbb{R}$ is firstly defined that sum up the task requirements from the start to the tail of e. Formally,

$$\mathbf{R}^{[t]}(e) = \sum_{\substack{u \preceq_{\mathcal{J}}^{[t]} e+ \\ u\in V_{\mathcal{J}}^{[t]}}} r_V^{[t]}(u), \forall e \in E_{\mathcal{J}}^{[t]} \tag{25}$$

The actual tasks as well as the total resource requirements on a device can be uniquely determined by the difference between aggregated requirements of the in-edge and out-edge of the particular device. Following the observation, the device constraint for all $u \in V_{\mathcal{D}}^{[t]}$ can be formalized as:

$$\sum_{j=1}^{J^{[t]}} \sum_{e\in E_{\mathcal{J}}^{[t],j}} \mathbf{R}^{[t]}(e) \left(\sum_{d+=u} x_{d,e}^{[t],j} - \sum_{d-=u} x_{d,e}^{[t],j} \right) \leq c_V^{[t]}(u) \tag{26}$$

where each e in the inner summation is either all zero or has exact one positive term and one negative term whose difference defines the actual task or tasks running on device u.

3.2 Solving D-VFC using LP

In this section, we base our LP-based approximate algorithm on the ILP formulation with a novel *partial* update strategy. The algorithm, theoretically, reduces the computation time from exponential using ILP to polynomial using LP, which are verified empirically through our ample experiments.

Problem Formulation In D-VFC, it is straightforward that the updates of $\lambda_{\mathcal{D}}(t)$ and $\lambda_{\mathcal{J}}(t)$ at time t determines variables $\tilde{\mathbf{x}}$ to be removed from $\mathbf{x}$, variables $\bar{\mathbf{x}}$ to be retained from $\mathbf{x}$, or new variables $\hat{\mathbf{x}}$ to be added. This essentially augments Eqs. 11–13 into the following form:

$$\begin{bmatrix} \mathbf{c} \\ \hat{\mathbf{c}} \end{bmatrix}^\top \begin{bmatrix} \mathbf{x} \\ \hat{\mathbf{x}} \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{c}} \\ \tilde{\mathbf{c}} \\ \hat{\mathbf{c}} \end{bmatrix}^\top \begin{bmatrix} \bar{\mathbf{x}} \\ \tilde{\mathbf{x}} \\ \hat{\mathbf{x}} \end{bmatrix} \tag{27}$$

$$\begin{bmatrix} \mathbf{A} & \mathbf{O} \\ & \hat{\mathbf{A}} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \hat{\mathbf{x}} \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{A}} & \mathbf{O} \\ \tilde{\mathbf{A}} & \\ & \hat{\mathbf{A}} \end{bmatrix} \begin{bmatrix} \bar{\mathbf{x}} \\ \tilde{\mathbf{x}} \\ \hat{\mathbf{x}} \end{bmatrix} \leq \begin{bmatrix} \bar{\mathbf{b}} \\ \tilde{\mathbf{b}} \\ \hat{\mathbf{b}} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ \hat{\mathbf{b}} \end{bmatrix} \tag{28}$$

$$\begin{bmatrix} \mathbf{C} & \mathbf{O} \\ & \hat{\mathbf{C}} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \hat{\mathbf{x}} \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{C}} & \mathbf{O} \\ \tilde{\mathbf{C}} & \\ & \hat{\mathbf{C}} \end{bmatrix} \begin{bmatrix} \bar{\mathbf{x}} \\ \tilde{\mathbf{x}} \\ \hat{\mathbf{x}} \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{d}} \\ \tilde{\mathbf{d}} \\ \hat{\mathbf{d}} \end{bmatrix} = \begin{bmatrix} \mathbf{d} \\ \hat{\mathbf{d}} \end{bmatrix} \tag{29}$$

where $\bar{\mathbf{c}}$, $\tilde{\mathbf{c}}$ and $\hat{\mathbf{c}}$, $\bar{\mathbf{A}}$, $\tilde{\mathbf{A}}$ and $\hat{\mathbf{A}}$, $\bar{\mathbf{b}}$, $\tilde{\mathbf{b}}$ and $\hat{\mathbf{b}}$, $\bar{\mathbf{C}}$, $\tilde{\mathbf{C}}$, and $\hat{\mathbf{C}}$, $\bar{\mathbf{d}}$, $\tilde{\mathbf{d}}$ and $\hat{\mathbf{d}}$ correspond to $\bar{\mathbf{x}}$, $\tilde{\mathbf{x}}$ and $\hat{\mathbf{x}}$.

The Partial Update Strategy The partial update strategy is based on online linear programming, and, is, specifically, relevant to the problem of multi-dimensional linear program [Agrawal, Wang, and Ye, 2014]. This is based on a key characteristic of multi-dimension decision making; each multi-dimension decision involves *mutually exclusive* alternatives. In the ILP formulation of VFC, the sub-problem of mapping between a job and the collection of corresponding devices resembles the property of mutual exclusion. Following Eq. 18, we define the collection of *per-job per-device* binary decisions as:

$$x_u^{[t],j} = \{x_{d,e}^{[t],j} | e \in E_{\mathcal{J}}^{[t],j}, d+ = u \in V_{\mathcal{D}}^{[t]}\} \tag{30}$$

each of which selects an edge d starting from u and determines the mapping between the edge $d \in E_{\mathcal{D}}^{[t]}$ and an edge $e \in E_{\mathcal{J}}^{[t],j}$. Note that devices can have multiple outward edges; the aforementioned aggregating all outward edges guarantees the single choice constraint, as in Eq. 20.

Based on the fact that, as defined in Eq. 16, $h^{[t],j}(\cdot)$ involves alternating among a collection of exclusive mapping decisions, $x_d^{[t],j}$ can, thereby, be considered a multi-dimension decision.

The LP Relaxation We first consolidate the ILP formulation of D-VFC based upon the fact that $\tilde{\mathbf{x}}$ is to be removed, Eqs. 27–

29 can be consolidated into the following:

$$\begin{bmatrix}\bar{\mathbf{c}}\\ \hat{\mathbf{c}}\end{bmatrix}^\top \begin{bmatrix}\bar{\mathbf{x}}\\ \hat{\mathbf{x}}\end{bmatrix} \tag{31}$$

$$\begin{bmatrix}\bar{\mathbf{A}} & \mathbf{O}\\ \hat{\bar{\mathbf{A}}} & \hat{\hat{\mathbf{A}}}\end{bmatrix} \begin{bmatrix}\bar{\mathbf{x}}\\ \hat{\mathbf{x}}\end{bmatrix} \leq \begin{bmatrix}\bar{\mathbf{b}}\\ \hat{\mathbf{b}}\end{bmatrix} \tag{32}$$

$$\begin{bmatrix}\bar{\mathbf{C}} & \mathbf{O}\\ \hat{\bar{\mathbf{C}}} & \hat{\hat{\mathbf{C}}}\end{bmatrix} \begin{bmatrix}\bar{\mathbf{x}}\\ \hat{\mathbf{x}}\end{bmatrix} = \begin{bmatrix}\bar{\mathbf{d}}\\ \hat{\mathbf{d}}\end{bmatrix} \tag{33}$$

where $\tilde{\mathbf{c}}$, $\tilde{\mathbf{A}}$, $\tilde{\mathbf{b}}$, $\tilde{\mathbf{C}}$ and $\tilde{\mathbf{d}}$ are removed; $\hat{\bar{\mathbf{A}}}$ and $\hat{\bar{\mathbf{C}}}$ can be interpreted as constraints for scheduling existing tasks on newly added devices, whereas $\hat{\hat{\mathbf{A}}}$ and $\hat{\hat{\mathbf{C}}}$ can be interpreted as constraints for scheduling newly added tasks on newly added devices.

The ILP formulation of D-VFC, as formulated in Eqs. 31–33, is first relaxed to an LP problem with continuous relaxation on $\mathbf{x}$, i.e., $0 \leq \mathbf{x}_\ell \leq 1, 1 \leq \ell \leq L$, as opposed to Eq. 14. Then, the dual problem of the LP relaxation [Padberg, 2013] is solved, which provides an upper bound to the optimal value of the primal problem. Let $\mathbf{p}$ denote the solution to the relaxed dual problem which will be used for partial update.

Per-job Per-device Formulation Following the partial update strategy, we partition $\hat{\mathbf{x}}$ into a collection of per-job per-device multi-dimensional decisions, which can be expressed as:

$$\hat{\mathbf{x}} = \{\hat{\mathbf{x}}_m\}_{m=1}^{M} \tag{34}$$

where M is the number of exclusive decisions for the collections of $x_d^{[t],j}$ defined by $\lambda_{\mathcal{D}}(t)$ and $\lambda_{\mathcal{J}}(t)$. While the proposed algorithm can solve $\hat{\mathbf{x}}_m$ in an arbitrary order, hereinafter, we, without loss of generality, assume the ordering of $\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_2, \cdots, \hat{\mathbf{x}}_M$.

In per-job per-device update, we can significantly reduce the complexity of the constraints by separating *global* constraints from *local* constraints and removing local constraints irrelevant to the current update. Specifically, global constraints are defined by all jobs whereas local constraints are defined in a per-job per-device manner. Recall that Eq. 33 presents the single choice constraint, and Eq. 32 presents the ordering constraint, and the device and network constraints. It is straightforward to rewrite Eq. 33 into the following form:

$$\mathbf{1}^\top \hat{\mathbf{x}}_m = 1 \tag{35}$$

where there is no global constraint and local constraints on $\{\hat{\mathbf{x}}_i | i \neq m\}$ are all removed.

In Eq. 32, the global constraints include the device and network constraints that are established by all jobs, whereas the local constraints includes the ordering constraint. We first rewrite Eq. 32 into the following form:

$$\begin{bmatrix}\bar{\mathbf{A}} & \mathbf{O}\\ \hat{\bar{\mathbf{A}}} & \hat{\hat{\mathbf{A}}}\end{bmatrix} \begin{bmatrix}\bar{\mathbf{x}}\\ \hat{\mathbf{x}}\end{bmatrix} = \begin{bmatrix}\mathbf{A_r}\\ \mathbf{A_o}\end{bmatrix} \begin{bmatrix}\bar{\mathbf{x}}\\ \hat{\mathbf{x}}\end{bmatrix} \leq \begin{bmatrix}\mathbf{b_r}\\ \mathbf{b_o}\end{bmatrix} = \begin{bmatrix}\bar{\mathbf{b}}\\ \hat{\mathbf{b}}\end{bmatrix} \tag{36}$$

where $\mathbf{A_r}$ and $\mathbf{b_r}$ resemble the global constraints and $\mathbf{A_o}$ and $\mathbf{b_o}$ represent the local constraints. It can be expanded into the following:

$$\begin{bmatrix}\mathbf{A_r}\\ \mathbf{A}_{\mathbf{o},1}\\ \vdots\\ \mathbf{A}_{\mathbf{o},m}\\ \vdots\\ \mathbf{A}_{\mathbf{o},M}\end{bmatrix} \begin{bmatrix}\bar{\mathbf{x}}\\ \hat{\mathbf{x}}_1\\ \vdots\\ \hat{\mathbf{x}}_m\\ \vdots\\ \hat{\mathbf{x}}_M\end{bmatrix} \leq \begin{bmatrix}\mathbf{b_r}\\ \mathbf{b}_{\mathbf{o},1}\\ \vdots\\ \mathbf{b}_{\mathbf{o},m}\\ \vdots\\ \mathbf{b}_{\mathbf{o},M}\end{bmatrix} \tag{37}$$

where $\mathbf{A}_{\mathbf{o},m}$ and $\mathbf{b}_{\mathbf{o},m}$ denote rows in $\mathbf{A_o}$ and elements in $\mathbf{b_o}$, respectively, corresponding to the job defined in $\mathbf{x}_m$. Observe that $\bar{\mathbf{x}}$ is known a priori, $\{\hat{\mathbf{x}}_i\}_{i=1}^{m-1}$ is solved in preceding updates, and $\{\hat{\mathbf{x}}_i\}_{i=m+1}^{M}$ is to be solved in later updates, the inequality constraints can be further trimmed into the following:

$$\begin{bmatrix}\mathbf{A_r}\\ \mathbf{A}_{\mathbf{o},1}\\ \vdots\\ \mathbf{A}_{\mathbf{o},m}\end{bmatrix} \begin{bmatrix}\bar{\mathbf{x}}\\ \hat{\mathbf{x}}_1\\ \vdots\\ \hat{\mathbf{x}}_m\\ \mathbf{0}\end{bmatrix} \leq \begin{bmatrix}\mathbf{b_r}\\ \mathbf{b}_{\mathbf{o},1}\\ \vdots\\ \mathbf{b}_{\mathbf{o},m}\end{bmatrix} \tag{38}$$

Putting together the aforementioned per-job per-device binary decisions, there are two consideration to solve $\hat{\mathbf{x}}_m$: partial update strategy and partial update ordering. We will discuss in detail the former here and show that our novel partial update strategy is robust against the latter.

Greedy Partial Update We first establish the baseline. Given the single choice constraint, as in Eq. 34, a naive strategy is to greedily make the binary decision that minimizes the objective value. The strategy can be implemented by solving the following optimization problem:

$$\begin{aligned}\text{minimize} \quad & \mathbf{c}_m^\top \hat{\mathbf{x}}_m\\ \text{subject to} \quad & \mathbf{1}^\top \hat{\mathbf{x}}_m = 1\\ & \begin{bmatrix}\mathbf{A}'\\ \mathbf{A}_{\mathbf{o},m}\end{bmatrix} \begin{bmatrix}\mathbf{x}'\\ \hat{\mathbf{x}}_m\\ \mathbf{0}\end{bmatrix} \leq \begin{bmatrix}\mathbf{b}'\\ \mathbf{b}_{\mathbf{o},m}\end{bmatrix}\end{aligned}$$

where $\mathbf{c}_m$ is the corresponding objective of $\hat{\mathbf{x}}_m$, $\mathbf{A}' = [\mathbf{A_r}\ \mathbf{A}_{\mathbf{o},1}\ \cdots \mathbf{A}_{\mathbf{o},m-1}]^\top$, $\mathbf{b}' = [\mathbf{b_r}\ \mathbf{b}_{\mathbf{o},1}\ \cdots \mathbf{b}_{\mathbf{o},m-1}]^\top$ and $\mathbf{x}' = [\bar{\mathbf{x}}\ \hat{\mathbf{x}}_1\ \cdots \hat{\mathbf{x}}_{m-1}]^\top$.

Dual LP Partial Update The greedy partial update strategy, however, fails to take into account the problem as a whole, especially task dependency and device connectivity, which may lead to unsatisfactory performance. Motivated by [Agrawal, Wang, and Ye, 2014], we propose to incorporate the global optimality by instead assigning the values of $\hat{\mathbf{x}}_m$ using the dual solution of the LP-relaxation of D-VFC. Particularly, $\hat{\mathbf{x}}_m$ can be determined by solving the following optimization problem:

$$\text{maximize} \quad \mathbf{p}^\top \mathbf{A}' \hat{\mathbf{x}}_m \tag{39}$$

$$\text{subject to} \quad \mathbf{1}^\top \hat{\mathbf{x}}_m = 1 \tag{40}$$

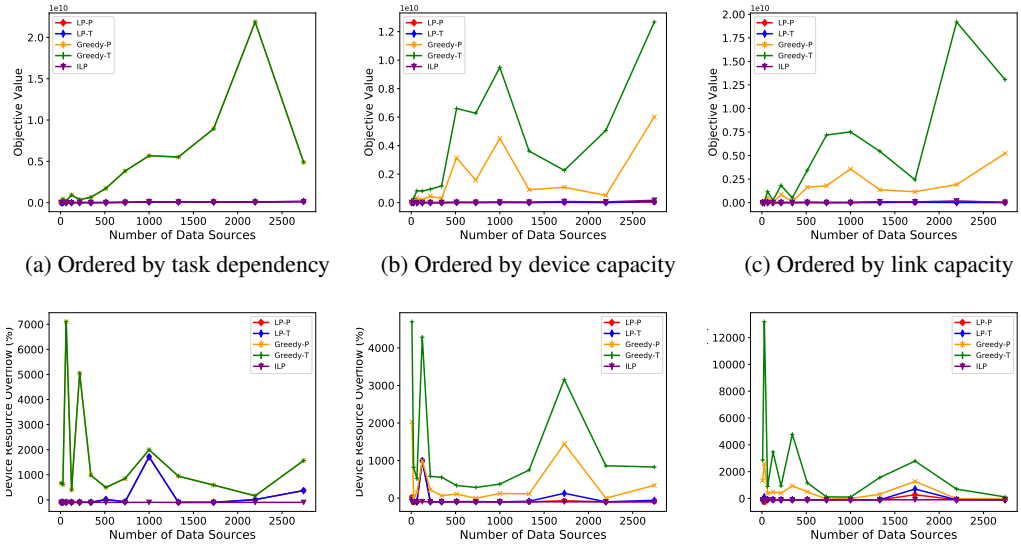
$$\begin{bmatrix}\mathbf{A}'\\ \mathbf{A}_{\mathbf{o},m}\end{bmatrix} \begin{bmatrix}\mathbf{x}'\\ \hat{\mathbf{x}}_m\\ \mathbf{0}\end{bmatrix} \leq \begin{bmatrix}\mathbf{b}'\\ \mathbf{b}_{\mathbf{o},m}\end{bmatrix} \tag{41}$$

where $\mathbf{p}$ is the solution to the LP-relaxed dual problem. The intuition behind the strategy is to ensure the resulting $\hat{\mathbf{x}}_m$ sufficiently close to the optimal solution of the LP-relaxation of D-VFC whilst having binary values in $\hat{\mathbf{x}}_m$, instead of real ones.

Solving D-VFC It is straightforward to solve $\hat{\mathbf{x}}_m$ given the singleton constraint presented in Eq. 40. Essentially, $\hat{\mathbf{x}}_m$ can be enumerated in linear time, each of which are evaluated against Eq. 41 in polynomial time. Then, the collection of feasible values of $\hat{\mathbf{x}}_m$ are evaluated against Eq. 39 in polynomial time, and the values of $\hat{\mathbf{x}}_m$ maximizes Eq. 39 are used to update for the corresponding job and the corresponding device defined by $\hat{\mathbf{x}}_m$.

Note that if a feasible solution to $\hat{\mathbf{x}}_m$ does not exist using either greedy partial update or dual LP partial update, the device and network constraints are relaxed allowing excess resource utilization of devices or their connectivities.

While solving $\mathbf{p}$ is also in polynomial time, our per-job per-device formulation of D-VFC leads to an efficient polynomial time solution. In both partial update strategies, the ordering constraint can be used to further improve the computational efficiency of Eq. 40 in the dual LP partial update strategy or



(a) Ordered by task dependency (b) Ordered by device capacity (c) Ordered by link capacity

(d) Ordered by task dependency (e) Ordered by device capacity (f) Ordered by link capacity

Figure 1: Quantitative Evaluations

its counterpart in greedy partial update strategy. Instead of enumerating and evaluating all possible binary decisions in $\hat{\mathbf{x}}_m$, it is straightforward to eliminate binary decisions that violate the ordering constraint.

4 Experiments

In this section, we start with describing the experimental setup followed by quantitative evaluations of our approach against ILP based offline optimal scheduler and the baseline online approximate method using greedy partial update.

4.1 Experimental Setup

The dataset is generated syntheticaly as follows. For simplicity of data generation and further analysis, a device graph $G_{\mathcal{D}}$ is a full b-nary tree where the root vertex, representing the cloud, is the only sink device, and the leaf vertices, representing data sources, are all source devices. A task dependency graph $G_{\mathcal{J}}$ consists of a collection of subgraphs $\{G^j_{\mathcal{J}}\}$, each of which corresponds to a unique source-sink device pair in $G_{\mathcal{D}}$.

We compare the proposed online approach with dual LP partial update against two baselines: the online approach with greedy partial update and the offline approach using ILP solver. In addition, to evaluate the implications of ordering of partial update, we further permute $\hat{\mathbf{x}}_m$ by taking into account combinations of groupings and orderings:

- *Groupings* include grouping by source-sink device pairs, grouping by device graph topological ordering
- *Orderings* include task dependency ordering, descending device capacity ordering and descending link capacity ordering

Each setting is repeated 5 times where the tree depth and the size of each job j is assumed, and the branching factor b is incremented iteratively until there are over 2,000 source devices. We, without loss of generality, picked the combination of tree depth of 4 and per-job job size of 3 for quantitative analysis of our approach.

4.2 Experimental Results

The results of our experiment are summarized in Fig. 1, where Fig. 1a and Fig. 1d, Fig. 1b and Fig. 1e, and Fig. 1c and Fig. 1f correspond to task dependency ordering, device capacity ordering and link capacity ordering, respectively. The prefix and postfix of a legend correspond to partial update strategy and grouping, respectively. The prefixes of LP and Greedy represent partial update strategies of dual LP and greedy, respectively, and the postfixes of P and T show grouping by source-sink device pairs and grouping by device graph topological order, respectively. Finally, ILP shows the performance upperbound of an offline ILP based solver.

Regarding optimality, as can be seen in Fig. 1a, Fig. 1b and Fig. 1c, our proposed dual LP partial update leads to objective values closest to the optimum given by offline ILP. In contrast, greedy partial update is sensitive to groupings and orderings, and often leads to rather poor objective values. This presents the feasibility of our dual LP partial update which can effectively take into account the global structure of a D-VFC instance while LP can be solved efficiently in polynomial time.

Regarding effectiveness, as can be seen in Fig. 1d, Fig. 1e and Fig. 1f, while our proposed dual LP partial update can almost surely provides solutions with objective values close to the optimum, the amount of excess resource utilization remains minimal, which can happen when a global optimum cannot be reached.

Note that the comparison of computational complexity is not included as that of the various online methods proposed in this paper are almost identical, and ILP is an offline method. Finally, dynamicities of task dependency and device connectivity, which are harder to simulate, are also resembled implicitly in the various experiments.

5 Conclusion

In this paper, we formulated generalized visual fog as an *online* scheduling problem based on online linear programming. Comparing with prior work using ILP, the novel formulation of online LP is feasible and effective with near optimal performance. Our dual LP partial update, which is efficient with theoretic time complexity of polynomial time, is robust against partial update ordering, which is essential for online methods to deal with unanticipated changes. The ample experimental results show that our solution is ready to meet the dynamicity challenges introduced by visual fog.

References

[Agrawal, Wang, and Ye, 2014] Agrawal, S.; Wang, Z.; and Ye, Y. 2014. A dynamic near-optimal algorithm for online linear programming. *Operations Research* 62(4):876–890.

[Bonomi et al., 2012] Bonomi, F.; Milito, R.; Zhu, J.; and Addepalli, S. 2012. Fog computing and its role in the internet of things. In *the first edition of the MCC workshop on Mobile cloud computing*.

[Botta et al., 2016a] Botta, A.; De Donato, W.; Persico, V.; and Pescapé, A. 2016a. Integration of cloud computing and internet of things: a survey. *Future Generation Computer Systems* 56:684–700.

[Botta et al., 2016b] Botta, A.; de Donato, W.; Persico, V.; and Pescape, A. 2016b. Integration of cloud computing and internet of things: a survey. 56:684–700.

[Casteigts et al., 2012] Casteigts, A.; Flocchini, P.; Quattrociocchi, W.; and Santoro, N. 2012. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems* 27(5):387–408.

[Chu et al., 2018] Chu, H.-M.; Yang, S.-W.; Pillai, P.; and Chen, Y.-K. 2018. Scheduling in visual fog computing: Np-completeness and practical efficient solutions. In *The 32nd AAAI Conference on Artificial Intelligence*.

[Dantzig, Orden, and Wolfe, 1955] Dantzig, G. B.; Orden, A.; and Wolfe, P. 1955. Generalized simplex method for minimizing a linear form under linear inequality restraints. 5:183–195.

[Demetrescu et al., 2010] Demetrescu, C.; Eppstein, D.; Galil, Z.; and Italiano, G. F. 2010. Dynamic graph algorithms. In *Algorithms and theory of computation handbook*, 9–9. Chapman & Hall/CRC.

[Ferreira, 2004] Ferreira, A. 2004. Building a reference combinatorial model for manets. *IEEE network* 18(5):24–29.

[IHS, 2015] 2015. Top video surveillance trends for 2015. Technical report.

[Li et al., 2011] Li, K.; Xu, G.; Zhao, G.; Dong, Y.; and Wang, D. 2011. Cloud task scheduling based on load balancing ant colony optimization. In *Sixth Chinagrid Annual Conference, ChinaGrid 2011*, 3–9.

[Liu, Quan, and Ren, 2011] Liu, S.; Quan, G.; and Ren, S. 2011. On-line real-time service allocation and scheduling for distributed data centers. In *IEEE International Conference on Services Computing, SCC 2011*, 528–535.

[Lu, Chowdhery, and Kandula, 2016] Lu, Y.; Chowdhery, A.; and Kandula, S. 2016. Visflow: A relational platform for efficient large-scale video analytics. Technical report.

[Michail, 2016] Michail, O. 2016. An introduction to temporal graphs: An algorithmic perspective. *Internet Mathematics* 12(4):239–280.

[Padberg, 2013] Padberg, M. 2013. *Linear optimization and extensions*, volume 12. Springer Science & Business Media.

[Sahni and Vidyarthi, 2018] Sahni, J., and Vidyarthi, D. P. 2018. A cost-effective deadline-constrained dynamic scheduling algorithm for scientific workflows in a cloud environment. *IEEE Trans. Cloud Computing* 6(1):2–18.

[Schrijver, 1998] Schrijver, A. 1998. *Theory of linear and integer programming*. John Wiley and Sons.

[Yang, Tickoo, and Chen, 2017] Yang, S.-W.; Tickoo, O.; and Chen, Y.-K. 2017. A framework for visual fog computing. In *The 2017 IEEE International Symposium on Circuits and Systems (ISCAS)*.