MANUFACTURING & SERVICE OPERATIONS MANAGEMENT
Vol. 00, No. 0, Xxxxx 0000, pp. 000–000
ISSN 1523-4614 | EISSN 1526-5498 | 00 | 0000 | 0001

INFORMS
DOI 10.1287/xxxx.0000.0000
© 0000 INFORMS

Collaboration and Multitasking in Networks: Architectures, Bottlenecks and Capacity

Itai Gurvich
Kellogg School of Management, i-gurvich@kellogg.northwestern.edu,

Jan A. Van Mieghem
Kellogg School of Management, vanmieghem@kellogg.northwestern.edu,

Motivated by the trend towards more collaboration in work flows, we study networks where some activities require the simultaneous processing by multiple types of multitasking human resources. Collaboration imposes constraints on the capacity of the process because multitasking resources have to be simultaneously at the right place. We introduce the notions of collaboration architecture and unavoidable bottleneck idleness (UBI) to study the maximal throughput or capacity of such networks. Collaboration and multitasking introduce synchronization requirements that may inflict unavoidable idleness of the bottleneck resources: even when the network is continuously busy (processing at capacity), bottleneck resources can never be fully utilized. The conventional approach that equates network capacity with bottleneck capacity is then incorrect because the network capacity is below that of the bottlenecks. In fact, the gap between the two can grow linearly with the number of collaborative activities.

Our main result is that networks with nested collaboration architectures have no unavoidable bottleneck idleness. Then, regardless of the processing times of the various activities, the standard bottleneck procedure correctly identifies the network capacity. We also prove necessity in the sense that, for any non-nested architecture, there are values of processing times for which unavoidable idleness persists.

The fundamental tradeoff between collaboration and capacity does not disappear in multi-server networks and has important ramifications to service-system staffing. Yet, even in multi-server networks, a nested collaboration architecture still guarantees that the bottleneck capacity is achievable. Finally, simultaneous collaboration, as a process constraint, may limit the benefits of flexibility. We study the interplay of flexibility and unavoidable idleness and offer remedies derived from collaboration architectures.

Key words: simultaneous collaboration, multitasking, architecture, work flow design, organizational design, capacity, stability, flexibility, control, priorities, bottlenecks
History:

1. Introduction

Motivated by the prevalence of collaborative processing in services, we study how simultaneous collaboration and multitasking impact capacity. By simultaneous collaboration we mean that some activities require the simultaneous processing by multiple types of resources. Discharging a patient, for example, may require the presence of both a doctor and a nurse. Multitasking means that a resource performs multiple activities. Multitasking is equivalent to resource sharing, which means that multiple activities require the same resource. A doctor, for example, may be required for

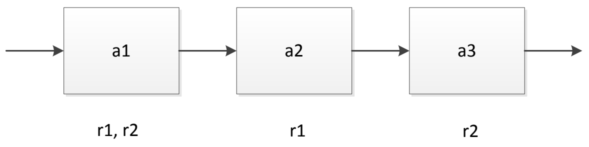

Figure 1 **A basic collaboration (BC) network where two resources collaborate on the first activity.**

both patient diagnosis and for patient discharge. Simultaneous collaboration imposes constraints on the capacity of the process because *multitasking* resources have to be *simultaneously* at the right place. The effects of these resource-synchronization requirements are more pronounced in human operated settings in which resources cannot be "split". An emergency room doctor may split her time between multiple activities—spending x% of her time in one activity and the remaining $(100-x)$% in another—and she may switch between the activities frequently, yet she cannot process both activities at the same time (which may, in this example, require her physical presence in two distinct locations).

The conventional approach for computing the capacity of a processing network follows three steps: (i) compute the capacity of each resource; (ii) identify the bottleneck resources—these are the resources with the smallest capacity. Steps (i) and (ii) have been formalized through a linear program that is called the static planning problem (SPP); see Harrison (2002). Finally, step (iii) of the conventional approach equates the network capacity with the bottleneck capacity.

In the presence of collaboration and resource sharing, however, the network capacity can be strictly smaller than the bottleneck capacity. Indeed, it is intuitively clear that synchronization constraints may lead to capacity losses. Given the simplicity and ubiquity of the traditional bottleneck approach, it is important to know when it is valid: when does bottleneck analysis yield the correct network capacity? This paper proffers some explicit and constructive answers to those questions for flow systems where some activities require multiple resources. Two simple examples serve well to set the ground for the general development in our paper.

Consider the basic collaboration (BC) network in Figure 1 with three activities $a1$, $a2$ and $a3$ and two resources $r1$ and $r2$. The resources co-process $a1$, namely they both have to be present for a unit of flow to be processed, and both resources are shared among multiple activities. The average processing time of activity $i \in \{1,2,3\}$ is m_i.

Conventional bottleneck analysis considers each resource in isolation: for resource i working in isolation, the maximal number of customers it could serve per unit of time would be $1/(m_1+m_{i+1})$. This maximal throughput is called the resource capacity and is denoted by:

$$\text{Capacity of resource } i \text{ is } \overline{\lambda}_i = (m_1 + m_{i+1})^{-1}. \tag{1}$$

Resource i is a bottleneck if it has the lowest resource capacity:

$$\text{Resource } k \text{ is a bottleneck } \Leftrightarrow \overline{\lambda}_k \leq \overline{\lambda}_j \text{ for all } j \neq k. \tag{2}$$

In the example, resource 1 is the bottleneck if $m_2 > m_3$ and resource 2 is the bottleneck if the reverse holds. They are both bottlenecks if $m_2 = m_3$. The network capacity cannot exceed the capacity of the bottleneck resource(s):

$$\lambda^{\mathrm{BN}} = \min_i \overline{\lambda}_i = \min \left\{ (m_1 + m_2)^{-1}, (m_1 + m_3)^{-1} \right\}. \tag{3}$$

Conventionally, one equates network capacity with bottleneck capacity but this can be incorrect:

A first observation is that, in the presence of collaboration, one must be careful about the prioritization policy even in the simplest of networks. To see this, assume first that both resources give priority to their individual tasks: whenever resource i has work available for activity $i+1$, the resource prioritizes that work (in contrast, say, to prioritizing work in activity 1). The central observation here is that under this policy the total number of jobs in the system is identical to that in a single server queue with service time equal to the sum of the three activity times so that the maximal throughput or network capacity λ^{net} is

$$\lambda^{\mathrm{net}} = (m_1 + m_2 + m_3)^{-1} < \lambda^{\mathrm{BN}}. \tag{4}$$

This can be easily argued through a sample path argument. Let $\{t_j\}$ be the consecutive customer arrival times and $\{(\sigma_1^j, \sigma_2^j, \sigma_3^j),\ j \geq 0\}$ be the corresponding service time triplets (customer j spends requires σ_1^j of service in activity 1, σ_2^j in activity 2 etc.). Let us assume that the system starts at time $t_0 = 0$ with 0 jobs in the system and consider the first customer to arrive, say at time t_1. Then, both resources will start working on this customer at time t_1. They will finish working on this customer at time $t_1 + \sigma_1^1$ at which time (due to the priorities) $r1$ will move to process this customer at $a2$. Once $r1$ finishes at time $t_1 + \sigma_1^1 + \sigma_2^1$, $r2$ will start processing this customer at $a3$. Only then will the two resources be available to process the next arrival at $a1$ (note that the resource will not switch to $a1$ even if there are arrivals before $t_1 + \sigma_1^1 + \sigma_2^1 + \sigma_3^1$). Continuing this way, one sees that the j^{th} to arrive (let its arrival time be t_j and its waiting time in the first queue be w_j) departs at time $t_j + w_j + \sigma_1^j + \sigma_2^j + \sigma_3^j$. The process behaves as if there were a single server processing sequentially each customer unit through the steps $a1$, $a2$ and $a3$.

We find then that (i) the queue count grows without bound even with an inflow rate below the bottleneck capacity; and that (ii) under this prioritization scheme, collaboration results in a

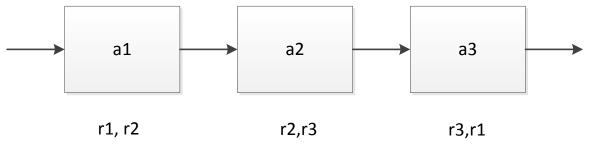

Figure 2 The BC+ network: the BC network augmented with a third collaborative resource.

capacity loss: If $m_i \equiv 1/2$, the maximal throughput drops to $2/3 < 1$: in the long run, both resources are only utilized $66\frac{2}{3}\%$ and are forced to idle $33\frac{1}{3}\%$ even though customers are waiting for service.

In this basic collaboration example the reason for the significant capacity loss is not the required collaboration in itself but rather the prioritization policy that introduces idleness. In the BC network, under the priority policy we discussed, a resource ($r1$ or $r2$) will remain idle frequently when there are jobs waiting in $a1$. Indeed, per our sample path discussion above, for each and every job, $r2$ must be idle while $r1$ is processing a customer in $a2$ and $r1$ must be idle while $r2$ is processing this customer in $a3$. Informally at this stage we can say that there is *avoidable idleness* in the system if resources remain idle even if there are jobs waiting in queues that they serve AND if there exist a policy that could avoid this idleness.

A possible resolution to avoid idleness in the BC network is to prioritize the collaborative work in activity $a1$. Under a preemptive version of that policy, if any customers wait at $a1$, both resources will process them. Thus, in the sample path discussion above, if customer 2 arrives before customer 1 completed processing in all activities, both resources move to $a1$ as soon they are available. Preemptively prioritizing the collaborative work eliminates avoidable idleness thereby maximizing the throughput in the BC network and achieving the bottleneck capacity.

In this example, then, the bottleneck analysis is valid – there is a policy under which bottleneck idleness is avoided so that the network capacity is equal to the bottleneck capacity. There are collaboration architectures that inherently introduce unavoidable bottleneck idleness (UBI) and for which collaboration comes at a capacity loss relative to the bottleneck capacity. Figure 2 augments the BC network with a third collaborative resource $r3$. The bottleneck capacity is

$$\lambda^{\mathrm{BN}} = \min_i \overline{\lambda}_i = \min\left\{(m_1+m_2)^{-1}, (m_2+m_3)^{-1}, (m_3+m_1)^{-1}\right\}. \tag{5}$$

Regardless of the prioritization policy, however, the collaboration architecture inherently forces at least one resource to be idle at any time. Consider, as before, the case where $m_i \equiv 1/2$. All three resources have equal workload and are bottlenecks with $\lambda^{\mathrm{BN}} = 1$. However, the collaboration architecture prevents any parallel processing; i.e., no activities can be simultaneously executed. Under no policy can the total queues be smaller than in a single server queue with service time equal to the sum of the three activity times and, consequently, the associated maximal throughput λ^{net} equals (4). Thus, each bottleneck resource is at most 2/3 utilized and features 1/3 unavoidable idleness due to the specific collaboration architecture. This phenomenon is driven by three requirements that, in combination, create unavoidable bottleneck idleness: simultaneous collaboration by multitasking resources that cannot be split. Collaboration, multitasking, or non-splitting in isolation do not create a capacity loss, but in combination their effect is significant.

Assume next the "multi-server" case where we have **two** units of each of the resource types $1, 2$ and 3. Then, the bottleneck capacity is 2 and it is now achievable. Indeed, we can split each resource type and dedicate one unit of each resource to each of its activities thus breaking the collaboration need (and removing its implications). But this is very particular to these numbers (i.e, to $m_i \equiv 1/2$) and even for this simple network there are processing times for which having multiple-units of each resource cannot remove the unavoidable bottleneck idleness.

We make the following contributions in the paper:

(1) *Formalizing unavoidable bottleneck idleness (UBI):* Collaboration introduces UBI meaning that bottlenecks can never be fully utilized. The conventional bottleneck approach (SPP) then overestimates the network capacity. We formulate an augmented linear program (SPPC - see (13) on p. 12) that takes into account processing conflicts and that correctly calculates network capacity. UBI captures the gap between the bottleneck capacity, as calculated by the SPP, and the network capacity, as derived by the SPPC. We show that this gap can grow linearly with the number of collaborative activities.

(2) *Architectures of collaboration:* We introduce a notion of *architecture of collaboration* that is characterized by the way resources are assigned to activities. We subsequently identify classes of architectures, namely nested architectures, for which we prove that UBI is always 0. For these architectures collaboration comes at no capacity loss (provided an appropriate policy is used) and the network capacity equals the bottleneck capacity. This simple architectural condition characterizes when the conventional bottleneck approach is correct.

Collaboration architectures can be nested, weakly non-nested, or strongly non-nested. Nested and weakly non-nested architectures never feature unavoidable idleness (i.e., UBI $= 0$). Strongly non-nested architectures can have unavoidable idleness (UBI > 0) for certain arrival and service-time parameter values.

The existence of UBI is intimately linked to integrality gaps in integer and linear programming. Indeed, our proofs build on relating the architectures of collaboration to the algebraic structure of the linear program that defines bottleneck capacity.

(3) *Collaboration and scale:* We show that UBI is not a "small-number issue:" it does not disappear in large systems that have multiple units of each resource type. Nested collaboration architectures guarantee that multiserver networks feature no unavoidable bottleneck idleness. This fact has ramifications for the staffing of service systems with collaboration.

(4) *Flexibility and unavoidable idleness:* Flexibility is expected to increase network capacity by shifting some work from bottleneck resources to non-bottlenecks. Simultaneous collaboration can, however, limit the benefits of flexibility: in most cases, unavoidable bottleneck idleness remains unavoidable when flexibility is introduced. In some cases flexibility brings no increase to the network capacity despite improvement promised by the bottleneck analysis. Extending our previous results, we provide conditions on the architecture that guarantees that UBI equal zero and, in particular, that the expected benefit of flexibility can be materialized despite the synchronization requirements. We draw some implications to flexibility investments: simple rules of thumb that can help distinguish between "problematic" and "safe" flexibility investments.

We end this introduction by pointing out important connections to existing literature. The dynamic control of processing networks that feature "simultaneous possession of non-splittable resources" with the objective of maximizing throughput has been studied by, for example, Tassiulas and Ephremides (1992), Dai and Lin (2005) and Jiang and Walrand (2010). The control literature takes the network capacity (as captured by the SPPC) as its departure point and studies how to achieve this maximal throughput through dynamic control. That stream of literature allows for a variety of processing constraints—collaboration being only one such possible constraint—and seeks to optimally control the network to achieve the network capacity or to optimize various refined performance metrics. The literature on (generalized) switches and on max-weight policies is in this spirit; see e.g. Stolyar (2004), Shah and Wischik (2012). While less applicable to human operated processes, simultaneous possession of *splittable* resources appears in the context of bandwidth sharing; see e.g. Massoulie and Roberts (2000).

We do not consider general constraints; we specifically are interested in collaboration and resource sharing. The question we address is when, and why, structural properties of the collaboration architecture result in a gap between the network capacity and the bottleneck capacity.

Simultaneous resource requirements also appear in project management, specifically Resource Constrained Project Scheduling (see e.g. Herroelen et al. (1998), Brucker et al. (1999)), machine scheduling and loss networks. Embedded in processing networks—and indeed in the concept "capacity"—is the repetitive nature of activities and hence "flow." While the BC and BC+ networks are fundamentally different from a capacity point of view, the project management literature would be indifferent between the two. With all service times equal to 1 hour, the BC and BC+ network both take 3 hours to complete a "project" that consists of activities 1, 2 and 3 with the precedence relation $1 \to 2 \to 3$. The same observations apply to the scheduling of machines to optimize flow time for a *finite* list of jobs as studied, e.g., by Dobson and Karmarkar (1989).

Loss networks feature a stream of arrivals that may require simultaneous access to multiple so-called "links" (the equivalent of resources in our setting) for their transmission. If not all the required links are available at the moment of arrival, the arriving unit is lost. Loss networks, by definition, do not have buffers. In our setting, if resources 1 and 2 in the BC network are busy with activities 2 and 3, an arriving job is not lost even though the resources are not available in activity 1. We can place the arriving unit in "inventory" and process it later. This would be not be the case in a loss network: the lack of buffers/storage leads to a very different behavior of throughput. Yet, also here the underlying resource-to-activity mapping (what we call the collaboration architecture) plays a facilitating role; see e.g. Kelly (1991), Zachary and Ziedins (1999) and the references therein.

Recent attention has been devoted to collaboration from the viewpoint of team dynamics and incentives for collaboration; see e.g. Roels et al. (2010) and the references therein. We do not model such issues in this work.

Finally, our paper studies the validity of capacity analysis for collaborative networks based on conventional bottleneck analysis. Other settings where a "naive" view of capacity falls short of capturing reality are studied, for example, in Bassamboo et al. (2010) and Chan et al. (2014) or in the context of closed queueing networks see Haviv (2013, Chapter 10). In a supply chain context, Graves and Tomlin (2003) observe that the total shortfall as computed by considering each stage of the supply chain in isolation may underestimate the total network (i.e, multi-stage supply chain) shortfall. Their observation is similar in spirit to ours: considering each stage or each resource in isolation may underestimate network losses. Their paper focuses on the shortfall (unmet demand) in a single-period problem while ours focuses on the loss in network capacity (maximal sustainable throughput) in a dynamic processing network.

In contrast to the work cited above, we limit our attention to deterministic or "fluid" analysis of the stochastic system. Our objective is to study the impact of collaboration and resource sharing on network capacity by relating to bottleneck analysis and network topology (the collaboration architecture). Dynamic control of stochastic systems is postponed as future work; see §7.

Some of the answers we provide in this paper are sufficiently simple to bring to the classroom. Indeed, two well-known cases that are widely used to teach capacity—"Pizza-Pazza" by

Van Mieghem (2008) and "Weight Solutions Clinic - Bariatric Surgery" by Chopra and Savaskan-Ebert (2013)—feature simultaneous collaboration and resource sharing. Smart students invariably question whether bottleneck analysis is correct or whether timing conflicts would create losses. Our theory provides a simple tool to show in both cases that the network capacity does equal the bottleneck capacity (regardless of processing times) because both networks have nested collaboration architectures and thus are guaranteed to exhibit no unavoidable idleness.

2. Network Notation and Graphic Conventions

We introduce some notation to facilitate the discussion of general networks. There is a set $\mathcal{K} = \{1,\dots,K\}$ of resources and a set $\mathcal{I} = \{1,\dots,I\}$ of activities. Following standard terminology we introduce the $K \times I$ *resource-activity incidence matrix* A where $A_{ki} = 1$ if resource k is required for activity i, and $A_{ki} = 0$ otherwise. The distinguishing feature of collaborative networks is that A has at least one column (activity) with multiple 1's (collaborative resources). The distinguishing feature of resource sharing is that at least one row (resource) has multiple 1's.

For $i \in \mathcal{I}$, we let $\mathcal{R}(\{i\})$ be the set of resources required for activity i (i.e, $k \in \mathcal{R}(\{i\})$ if $A_{ki} = 1$). More generally, $\mathcal{R}(\mathcal{A})$ is the set of resources required for some activity in the set $\mathcal{A} \subseteq \mathcal{I}$: $k \in \mathcal{R}(\mathcal{A})$ if $k \in \mathcal{R}(\{i\})$ for some $i \in \mathcal{A}$. In this paper we assume that each activity is associated with a single buffer: there are I buffers. To avoid trivialities we assume that each resource is required for at least one activity so that $\mathcal{R}(\mathcal{I}) = \mathcal{K}$. We let $\mathcal{S}(i,j)$ be the set of resources shared by activities i and j:

$$\mathcal{S}(i,j) = \mathcal{R}(\{i\}) \cap \mathcal{R}(\{j\}).$$

Each of the buffers can have exogenous arrivals. Let $\alpha = (\alpha_1,\dots,\alpha_I)$ be the rates of exogenous arrivals where $\alpha_i = 0$ if there are no exogenous arrivals to buffer i. We assume that there exists at least one i with $\alpha_i > 0$ (the network is open – it has exogenous arrivals). The routing in the network is assumed to be Markovian with a routing matrix P so that P_{kl} is the probability that a customer is routed to buffer l upon its completion of service at activity k. The matrix P' denotes the transpose of P. To ensure that our queueing network is open, the matrix P (and, in turn, P') is assumed to have spectral radius less than 1. The matrix

$$Q = (\mathbf{1} - P')^{-1} = \mathbf{1} + P' + (P')^2 + (P')^3 + \cdots,$$

where $(P')^n$ denotes the n^{th} power of P', is then well defined and the j^{th} element of

$$\lambda = Q\alpha,$$

is interpreted as the rate of arrivals to activity j if all service times were 0. Informally, if the network can sustain the exogenous arrival rate vector α then one expects that, in the long run, λ_i will be the input (and also the output) rate in activity i. Again, to avoid trivialities we assume that λ is a strictly positive vector. Finally, the service time in activity i has a mean m_i.

The probabilistic properties of the arrival, service and routing processes are immaterial for this paper. We refer to (α, P, A, m) as the *network primitives.* It is an established fact that under simple independence assumptions and assuming finite means for the various variables, a deterministic "fluid" model is a powerful tool to characterize capacity and throughput; see e.g. Dai (1999).

Our setup is reminiscent of the traditional multi-class queueing network setting; see e.g. Williams (1998). Yet, it is different in terms of what can be referred to as a *station* here. In the typical multiclass setting where each activity is performed by a single resource, a station typically corresponds to a resource performing a set of activities. That representation can be called a resource view of a network. In collaborative networks with resource sharing, however, an activity view is more appropriate. A *station* here corresponds to an activity. A station may require the collaboration of multiple resources and subsets of these resources may be required also at other stations.

The setup we introduced thus far does not cover parallel server networks or systems where an activity can pull jobs from several buffers simultaneously (such as in a fork-join network). We choose the more restricted setting to focus on key aspects of collaboration.

Graphic conventions: We draw multiple examples throughout this paper. Our convention is to depict each activity by a rectangle (the activity has a single buffer but may have multiple resource involved). Two activities are connected by a line if some of the jobs leaving the first station are routed to the second. Within each rectangle we put the activity number (we add the letter a to emphasize that this is an activity). Below each rectangle we list the resources that are required for this activity (adding the letter r for resource). When it is clear from the context that we refer to an activity (respectively a resource) we will omit the letter 'a' (respectively 'r').

3. Bottlenecks, Feasible Configurations and Unavoidable Idleness

The conventional approach to identify bottlenecks and capacity is easily extended from our basic examples in the introduction to the general network defined in the previous section. Recall that λ_i represents the flow rate or throughput at activity i and $\lambda_i m_i$ represents the "load" of activity i: the expected amount of processing time of activity i per unit of time. Similarly, the corresponding amount of processing time required from resource k, or the "load" of resource k equals $\sum_i A_{ki}\lambda_i m_i$.

Assume for now that *there is only one unit of each resource type*; this will be relaxed in §5. The highest loaded resources are the bottlenecks and we denote the set of bottleneck resources by

$$\mathbf{BN} = \arg\max_{k\in\mathcal{K}} \sum_i A_{ki}\lambda_i m_i. \tag{6}$$

and the bottleneck load by

$$\rho^{\mathrm{BN}} = \max_{k\in\mathcal{K}} \sum_i A_{ki}\lambda_i m_i. \tag{7}$$

If $\rho^{\mathrm{BN}} \leq 1$, then ρ^{BN} is the *bottleneck utilization* or the fraction of time the bottlenecks are busy processing. Given that the utilization depends on the inflow λ, we often will make that dependence explicit and write $\rho^{\mathrm{BN}}(\lambda)$. This is consistent with the static planning problem (SPP) – see Harrison (2002) – which in our setting specializes to

$$\begin{aligned} \rho^{\mathrm{BN}}(\lambda) = \text{minimize} \quad & \rho \\ \text{s.t.} \quad & \textstyle\sum_i A_{ki}(\lambda_i m_i) \leq \rho, \text{for all } k \in \mathcal{K}, \end{aligned} \tag{SPP}$$

and has the solution in (7). When there is flexibility in the allocation of resources to activities, the SPP is augmented with additional decision variables. For now, however, the set of resources required for an activity is given; we relax this in §6.

For example, the incidence matrix of the BC network is

$$A = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}, \tag{8}$$

and the SPP requires here that $\lambda_1 m_1 + \lambda_2 m_2 \leq \rho$ and $\lambda_1 m_1 + \lambda_3 m_3 \leq \rho$. Given that only activity 1 has exogenous arrivals (with rate α_1), $\lambda_1 = \lambda_2 = \alpha_1$ and the solution to the SPP is given by

$$\rho^{\mathrm{BN}} = \alpha_1 \max\{m_1 + m_2, m_1 + m_3\}. \tag{9}$$

As $\alpha_1 \to \min\{(m_1+m_2)^{-1}, (m_1+m_3)^{-1}\}$, $\rho^{\mathrm{BN}} \to 1$ in agreement with the bottleneck capacity (3).

The conventional capacity approach equates the capacity of the network with full utilization of the bottlenecks: the network capacity then is the family of vectors λ for which $\rho^{\mathrm{BN}}(\lambda) = 1$ (the interior of this set is often referred to as the stability region). Notice that this bottleneck analysis considers each resource in isolation to quantify network capacity. Collaborative networks, however, feature activities that require multiple resources and this introduces resource synchronization constraints that are not captured by this procedure. These can be accounted for by explicitly incorporating the *collaboration constraints* using *configuration vectors*; see e.g. (Jiang and Walrand 2010, Chapter 6).

A feasible configuration vector is a binary I-vector such that $a_i = a_j = 1$ if activities i and j can be performed simultaneously, which means that they do not share resources. Clearly, the (column) unit vectors in $\mathbb{R}^I$ are the natural activity vectors for each network; indeed, unit vector e^i, where $e^i_j = 1$ if $j = i$ and 0 otherwise, represents a configuration where only activity i is being performed. Activities i and j can be performed simultaneously if the set $\mathcal{R}(\{i\})$ of resources required by activity i does not overlap with the resource set $\mathcal{R}(\{j\})$ required by activity j, i.e., $\mathcal{S}(i,j) = \emptyset$. (Recall that, for now, each resource type contains exactly one, non-splittable unit.)

To illustrate, let us return to the BC network with incidence matrix A given by (8). The three natural configuration vectors $\{(1,0,0)', (0,1,0)', (0,0,1)'\}$ represent the allocation of resources to a unique activity. Activities 2 and 3 can be performed simultaneously because they require different resources. This is represented by the fourth configuration vector $(0,1,1)'$. The vectors $(1,1,0)'$ and $(1,0,1)'$ are **not** feasible configuration vectors. The family of feasible configuration vectors for this network is then $\{(1,0,0)', (0,1,0)', (0,0,1)', (0,1,1)'\}$. *If there is no resource sharing, all* 2^I *binary vectors are feasible configuration vectors.*

Now incorporate the collaboration constraints into the network capacity calculation as follows. Let $\pi_k \geq 0$ be a proxy for the fraction of time that the k-th configuration vector is active. Utilization level $\rho \leq 1$ is then feasible if there exists a time-allocation vector π such that $\sum_k \pi_k = \rho$ and

$$\pi_1 \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + \pi_2 \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} + \pi_3 \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} + \pi_4 \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} \lambda m_1 \\ \lambda m_2 \\ \lambda m_3 \end{pmatrix}. \tag{10}$$

For this basic example, a feasible time-allocation vector is calculated as a simple function of ρ:

$$\pi_1 = \alpha_1 m_1, \pi_2 = \rho - \alpha_1(m_1 + m_3), \pi_3 = \rho - \alpha_1(m_1 + m_2), \pi_4 = \alpha_1(m_1 + m_2 + m_3) - \rho. \tag{11}$$

As $\alpha_1 \to \min\{(m_1 + m_2)^{-1}, (m_1 + m_3)^{-1}\}$, we get that $\pi_i \to 0$, for $i = 2, 3$, if resource $i - 1$ is a bottleneck and π_1, π_4 have the appropriate values. If $m_2 = m_3$, only two configuration vectors are active at capacity: one corresponding to the collaborative activity and the other to the simultaneous processing of the specialized activities. Thus, there exists a feasible time allocation to feasible configurations for all $\alpha_1 \leq \min\{(m_1 + m_2)^{-1}, (m_1 + m_3)^{-1}\}$. This proves that in the BC network the bottleneck capacity equals the network capacity.

It is convenient to introduce the following set notations. An activity subset $\mathcal{A} \subseteq \mathcal{I}$ is feasible if no two activities in $\mathcal{A}$ share resources. Let $\mathbb{C}$ represent the family of all feasible activity subsets:

$$\mathbb{C} = \{\mathcal{A} \subseteq \mathcal{I} : \mathcal{S}(i,j) = \emptyset \text{ for all } i, j \in \mathcal{A}\}. \tag{12}$$

With each feasible activity subset $\mathcal{A} \subseteq \mathcal{I}$ corresponds a feasible configuration vector $a(\mathcal{A})$, which is the binary I-vector where $a_i(\mathcal{A}) = 1$ if $i \in \mathcal{A}$ and $a_i(\mathcal{A}) = 0$ otherwise, and $a_i(\mathcal{A})a_j(\mathcal{A}) = 0$ if

$i \neq j \in \mathcal{A}$ and $\mathcal{S}(i,j) \neq \emptyset$. Equivalently, the activity subset $\mathcal{A}$ is feasible if $\sum_i A_{ki} a_i(\mathcal{A}) \leq 1$ for each resource k. Obviously, the number of configuration vectors may be large, up to 2^I.

Each element in $\mathbb{C}$ is a set of activity indices. In the BC network for example, $\mathbb{C} = \{\{1\},\{2\},\{3\},\{2,3\}\}$, where the first three sets are singletons (each corresponding to one of the activities 1, 2, 3) and the last contains activity 2 and 3. These have a one to one correspondence with the set of vectors used in (10). For example $a(\{1\}) = (1,0,0)'$.

Let λm denote the vector $(\lambda_1 m_1, \ldots, \lambda_I m_I)$. We define the *static planning problem for collaboration* (SPPC) as: Find time-allocation vector π and minimal network utilization ρ^{net} such that

$$\begin{aligned} \rho^{\text{net}}(\lambda) = \text{minimize} \quad & \rho \\ \text{s.t.} \quad & \textstyle\sum_{\mathcal{A} \in \mathbb{C}} a(\mathcal{A}) \pi(\mathcal{A}) = \lambda m, \\ & \textstyle\sum_{\mathcal{A} \in \mathbb{C}} \pi(\mathcal{A}) \leq \rho, \\ & \pi \geq 0. \end{aligned} \tag{13}$$

The *network capacity* is the set of throughput vectors $\lambda \geq 0$ for which $\rho^{\text{net}}(\lambda) = 1$, which implies that the network is fully utilized: the total time allocated under the optimal solution equals 1.

Any feasible solution $(\pi^*, \rho^{\text{net}})$ to the SPPC corresponds to a feasible solution ρ^{BN} to the SPP but not vice-versa. Since a feasible configuration vector satisfies $\sum_i A_{ki} a_i \leq 1$, we have:

$$\sum_i A_{ki}(\lambda_i m_i) = \sum_i A_{ki} \left(\sum_{\mathcal{A} \in \mathbb{C}} a_i(\mathcal{A}) \pi(\mathcal{A}) \right) \leq \sum_{\mathcal{A} \in \mathbb{C}} \pi(\mathcal{A}) \leq \rho^{\text{net}}, \text{ for all } k \in \mathcal{K}, \tag{14}$$

which shows that ρ^{net} is feasible for the SPP. In words, due to the collaboration synchronization requirements, more time may be needed to process the same throughput. This establishes the following lemma.

Lemma 3.1 *The bottleneck utilization is a lower bound to the network utilization ρ^{net}. That is,*

$$\rho^{BN}(\lambda) \leq \rho^{net}(\lambda),$$

for any $\lambda \geq 0$. Without multitasking, or without collaboration, $\rho^{BN}(\lambda) = \rho^{net}(\lambda)$.

The second part of this lemma is a direct corollary of our Theorem 4.4 in the next section. Thus, the SPP provides a lower bound on the SPPC that is tight without multitasking or without collaboration but may not be tight otherwise.

Example 3.1 Re-consider the BC+ network in Figure 2. If $m_i \equiv 1$ and $\alpha_1 = 1/2$, then $\rho^{\text{BN}} = 1$. Indeed, in this case, $\lambda_1 = \lambda_2 = \lambda_3 = \alpha_1$ and the solution to the SPP is given by

$$\rho^{\text{BN}} = \alpha_1 \max\{m_1 + m_2, m_1 + m_3, m_2 + m_3\} = 2\alpha_1.$$

The bottleneck utilization approaches 1 as α_1 approaches 1/2. There are **three** feasible configuration vectors here – these are the unit vectors in $\mathbb{R}^3_+$ associated with the sets $\mathbb{C} = \{\{1\},\{2\},\{3\}\}$ and the SPPC has the (unique) feasible solution $\pi_i = \lambda_i m_i = \alpha_1 = 1/2$ for $i = 1,2,3$ so that $\rho^{\text{net}} = 3\alpha_1 = 3/2 > \rho^{\text{BN}}$. There is a gap between the SPP and the SPPC: the maximal value of α_1 for which the SPPC has an optimal value $\rho^{\text{net}} \leq 1$ is $\alpha_1 = 1/3$ in which case $\pi_1 = \pi_2 = \pi_3 = 1/3$. ∎

The discussion thus far leads to the following definition.

Definition 3.1 *For any $\lambda \geq 0$, we define unavoidable bottleneck idleness (UBI) as*

$$UBI(\lambda) = \rho^{net}(\lambda) - \rho^{BN}(\lambda).$$

We say that the network features unavoidable idleness if there exists λ for which $UBI(\lambda) > 0$.

The meaning of unavoidable bottleneck idleness: Our notion of unavoidable idleness is grounded in the physical meaning of idleness. Consider the BC+ network with $m_i \equiv 1$ and recall that its network capacity is $\lambda^* = (1/3, 1/3, 1/3)$ for which the network is fully utilized: $\rho^{\text{net}}(\lambda^*) = 1$. The bottleneck capacity, however, exceeds the network capacity: viewed in complete isolation, each bottleneck resource can process up to $1/(1+1) = 1/2$ jobs per unit of time. When all resources must work in concert, however, that bottleneck capacity is not achievable by the network due to collaboration and resource sharing requirements. At the network capacity, the bottleneck utilization is $\rho^{\text{BN}}(\lambda^*) = 2/3$ so that the bottleneck resources are idle for a third of their time. This idleness is **unavoidable** – one cannot increase the throughput of the process beyond $\lambda^* = (1/3, 1/3, 1/3)$ to make the resources more busy. This is visualized by the utilization profile on the right in Figure 3.

In fact, for any throughput $\lambda \leq \lambda^*$, $\rho^{\text{net}}(\lambda) - \rho^{\text{BN}}(\lambda)$ captures the idleness forced on the bottleneck resources by collaboration. Consider a throughput $\lambda = (1/4, 1/4, 1/4)$ below network capacity $\lambda^* = (1/3, 1/3, 1/3)$. Then, $\rho^{\text{net}} = 3/4$ while $\rho^{\text{BN}} = 1/2$. As shown by the utilization profile on the left in Figure 3, the bottleneck resources idle half the time. Part of their total idleness is unavoidable idleness $= \rho^{\text{net}} - \rho^{\text{BN}} = 1/4$, while the other part $1/2 - 1/4 = 1/4$ is avoidable (e.g., by increasing throughput to network capacity).

A production interpretation provides another illustration of unavoidable bottleneck idleness. Say we wish to produce at a rate of a 1/4 per hour. Then the bottleneck resources will idle 1/2 of the time—or 30 minutes out of every hour—since $\rho^{\text{BN}} = 1/2$. Yet, this does not mean that we could reduce the production shift length by 1/2. The amount of idleness per hour that can be removed is only a 1/4 (15 minutes). There are 15 minutes per hour of bottleneck idleness that we must absorb

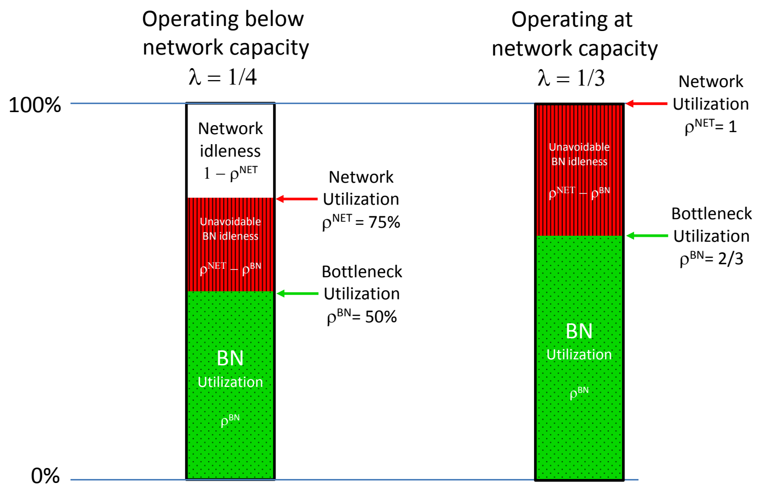

Figure 3 **In the BC+ network, the bottleneck resources exhibit unavoidable idleness for any throughput λ.**

to be able to work at the throughput of 1/4 due to resources waiting for other resources. If the team of resources works for 45 minutes per hour, they can continue processing at a rate of 1/4 per hour, but not if the they work only 30 minutes per hour.

Thus, while in the absence of collaboration, a production process' hours can, in first order, be cut by a fraction $1 - \rho^{\text{BN}}$, this is not true here. The maximum we can cut here is $1 - \rho^{\text{BN}} -$ (Unavoidable idleness) $= 1 - \rho^{\text{net}}$. In other words, in the presence of collaboration and resource sharing, bottleneck utilization and network utilization measure different things: the first measures the fraction of time that bottlenecks are busy while the latter measures the fraction of time the network is busy. Due to synchronization constraints, the network must work longer hours to process the same amount of input as the bottlenecks process.

4. Collaboration Graphs and Architectures of Collaboration

Quantifying unavoidable bottleneck idleness by solving and comparing two linear programs, as we did thus far, requires the values of all network parameters (arrival rates, services times and routing probabilities). We next introduce architecture terminology that allows us to state (and prove) conditions for unavoidable idleness that avoid such direct computation and that do not depend on these parameters.

The resource-activity matrix A captures the processing conflicts – that is activities that cannot be performed simultaneously because they share resources. A graph representation of the matrix A provides a formal tool to characterize properties of what we call the *collaboration architecture.* An intuitive graph representation of the matrix A represents each activity by a node and connects two

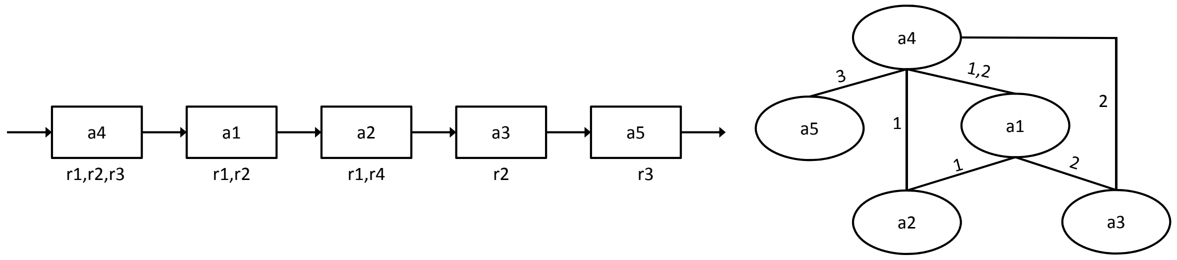

Figure 4 **A network with nested collaboration architecture (left) and its associated collaboration graph (right).**

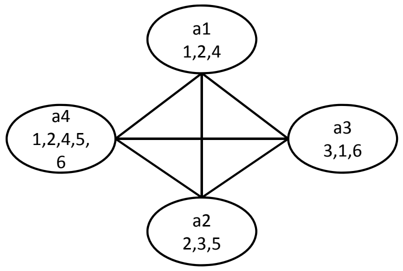

Figure 5 **A (complete) collaboration graph of a 4-activity network**

nodes with an (undirected) edge if those two activities share resources. For example, the network with five activities on the left in Figure 4 has the *collaboration graph* shown on the right of that figure. There are multiple conflicts in this network. Activities 1 and 4, for example, cannot be processed simultaneously as both require resources 1 and 2 which gives rise to 2 cycles. Removing activity 4 would yield a network with acyclic collaboration graph.

The collaboration graph is an undirected graph. Notice that, without multitasking, this collaboration graph is just a collection of isolated nodes (activities). Without collaboration, each activity requires a single resource so that the graph is a collection of isolated paths where each path links all activities that utilize the same resource.

Our first architectural result shows that UBI can grow linearly with the number of activities. Collaboration and multitasking can thus lead to large capacity losses. The result is shown by constructing a network whose collaboration graph is a complete graph (meaning that there is an edge between any pair of nodes). Figure 5 gives an example.

Theorem 4.1 *Given I, one can construct a network with I activities and $K = 3 + \sum_{i=3}^{I-1}$ resources such that the collaboration graph is a complete graph and $\rho^{net} - \rho^{BN} = I/2 - 1$.*

(All proofs are relegated to the online supplement.) In the special case of the BC+ network ($I = 3$) with mean service time equal to 1 and arrival rate equal to 1/2, we have $\rho^{\mathrm{BN}} = 1$ and $\rho^{\mathrm{net}} = I/2 = 3/2$.

Recall that ρ^{net} captures the time (not fraction of time) it takes the network to process an input of λ that arrives in one time unit. $\rho^{\text{net}} > 1$ means that the network has to work multiple time units to process a single time unit worth of arrivals. UBI $= I/2 - 1$ captures the fact that the constructed network will take $I/2$ to complete processing the input but, out of this time, the bottlenecks will be working only for one time unit.

Our first positive result is the following– recall that a polyhedron is integral if all its extreme points are integer valued.

Theorem 4.2 *$\rho^{BN} = \rho^{net}$ and UBI $= 0$* **for any primitives** (α, P, μ)*, if and only if the Polyhedron $\mathcal{P} = \{x \in \mathbb{R}^I_+ : Ax \le 1\}$ is integral.*

In general, having an integral polyhedron is not a necessary condition for $\rho^{BN} = \rho^{net}$. There can be, as we show in §4.1, networks with UBI > 0 under some choice of parameters but with UBI $= 0$ under other choices. The integrality of the relevant polyhedron is necessary and sufficient if one seeks to get a result that is independent of the specific service and arrival parameter values.

We focus below on sufficient conditions on A that guarantee that the polyhedron $\mathcal{P}$ is indeed integral. The first condition comes directly from the theory of integer programming–see, e.g., (Schrijver 1998, Corollary 19.2a)–and, to a large extent, reduces our pursuit to identifying conditions on the collaboration architecture that guarantee that A is totally unimodular.

Theorem 4.3 *If the matrix A is totally unimodular (TUM) then the network features no unavoidable idleness. For any throughput $\lambda > 0$, UBI$(\lambda) = 0$ and $\rho^{BN}(\lambda) = \rho^{net}(\lambda)$.*

For purely computational purposes, Theorem 4.3 might be sufficient as there exist algorithms that verify whether a matrix is totally unimodular. We proceed to analyze which kinds of collaboration architectures result in a totally unimodular matrix A. The following corollary follows almost directly from existing results about TUM matrices:

Corollary 4.1 *If each resource has at most two activities and the collaboration graph is bi-partite, then the network features no unavoidable idleness. For any throughput $\lambda > 0$, UBI$(\lambda) = 0$ and $\rho^{BN}(\lambda) = \rho^{net}(\lambda)$.*

For the network of Figure 4, the collaboration graph is not bi-partite– it includes a cycle that has the three activities $a1, a2$ and $a4$. This, however, is what we will call a nested-sharing cycle: The resources shared by activities $a1$ and $a2$ are a subset of the resources shared by activities $a1$ and $a4$. We will show that nested-sharing cycles do not generate unavoidable idleness.

A set of l activities $i_1,\dots,i_l$ form a cycle if activity i_j shares resources with activity i_{j+1} ($j = 1,\dots,l-1$) and activity i_l shares resources with activity i_1. A set of such activities are said to form a *nested-sharing cycle* if whenever activities i_j and i_l with $\ell > j$ share resources, these resources are also shared by activities i_{l-1} and i_l; formally, if the activities can be ordered so that for all $\ell > k > j$:

$$\mathcal{S}(i_j, i_\ell) \subseteq \mathcal{S}(i_k, i_\ell). \tag{15}$$

In Figure 4, $\mathcal{S}(4,2) \subseteq \mathcal{S}(1,2)$ so that activities 1, 2 and 4 form a nested-sharing cycle. Naturally when a cycle is not nested we say that it is non-nested. Note that if the collaboration graph is acyclic the architecture is, in particular, nested. It is also useful to observe that in a nested-sharing cycle there must be a resource that is shared by all activities in the cycle.

The intuition that nested-sharing cycles do not insert unavoidable idleness rests on how a finite amount of work is allocated to resources (ignoring the precedence processing constraints). In Figure 4 resources $1,2$ and 3 can be allocated to activity $a4$ until it is "exhausted," meaning its amount of work $\lambda_4 m_4$ is processed. Afterwards, these resources are free to work on activities $a1$ and $a5$ *in parallel* since there is no sharing between these activities. Note that once activity $a1$'s work is done, resources $1,4$ and 2 can work in parallel on activities $a2$ and $a3$ even though these two activities are part of a cycle that contain also $a1$ and $a4$. With nested architectures, work can be organized so that once two resources complete all the tasks on which they collaborate they impose no further constraints on each other.

Non-nested cycles are in general problematic. We cannot go through the exercise we performed for the network in Figure 4. Resources cannot be gradually "freed" in a non-nested cycle: when certain resources are working, other resources are forced to idle. In the BC+ network when resources 1 and 2 work, resource 3 *must idle.*

Definition 4.2 (**Nested Collaboration Architecture**) We say that the network has a *nested collaboration architecture* if *any* cycle in the collaboration graph is a nested-sharing cycle.

Definition 4.3 (**Non-nested Collaboration Architectures**) We say that the network has a *non-nested collaboration architecture* if the network has a non-nested cycle. We say that it is *weakly non-nested* if every such cycle has a resource that is shared by all activities in the cycle. We say that it is *strongly non-nested* otherwise.

We are able to prove that if a network has a nested collaboration architecture then its matrix A is TUM. If the network is weakly non-nested it can be transformed, without changing ρ^{BN} and ρ^{net} into a network with a matrix A that is TUM. Invoking Theorem 4.3 then yields:

Theorem 4.4 *A network with nested or weakly non-nested collaboration architecture features no unavoidable idleness. For any throughput* $\lambda > 0$, $UBI(\lambda) = 0$ *and* $\rho^{BN}(\lambda) = \rho^{net}(\lambda)$.

This sufficient condition is strong in that it depends only on the network structure (its collaboration architecture) and is independent of service-time, routing probability and arrival-rate parameters. The value of the nested structure is that it is "robust" in that it hold for *any* service and arrival parameters.

We will show that our sufficient conditions are necessary in the following sense: If a collaboration graph is not bi-partite or if it has a strongly non-nested cycle, then there exist parameter values (α, P, μ) for which the network will feature unavoidable idleness.

We say that a cycle $i_1, \ldots, i_l$ is simple non-nested if each two activities connected by an edge share a resource that is not used in any other activity in the cycle: for each j, $\mathcal{S}(i_j, i_{j+1}) \neq \mathcal{S}(i_\ell, i_{\ell+1})$ for all $\ell \neq j$. In a simple cycle, there is a one-to-one mapping from the edges of the cycle to a subset of the resources. *The BC+ network is the prototypical simple non-nested cycle of odd length.*[1] It has three edges each of which corresponds to a distinct shared resource. We show that every non-nested collaboration architecture has a simple non-nested cycle (not necessarily of three edges) and, subsequently, deduce a sufficient condition for the persistence of unavoidable idleness.

Lemma 4.2 *A network with a strongly non-nested collaboration architecture has a simple non-nested cycle.*

Theorem 4.5 *Consider a network with a strongly non-nested collaboration architecture. If one of its simple non-nested cycles has an odd number of activities, then there exist parameter values* (α, P, μ) *(and, in turn,* λ*) such that* $\rho^{net}(\lambda) \neq \rho^{BN}(\lambda)$ *and* $UBI(\lambda) > 0$.

The following is a rough summary of the results derived in this section:

(i) **Bounds:** One can construct networks with arbitrarily large values of UBI; consistent with Theorem 4.5, the examples underlying Theorem 4.1, as in Fig. 5, have a strongly non-nested architecture.

(ii) **Sufficiency for UBI $= 0$:** Networks with either a bi-partite, nested, or a weakly non-nested architectures have $\rho^{\text{net}} = \rho^{\text{BN}}$ and UBI $= 0$.

(iii) **Necessity for UBI $= 0$:** If the architecture is non-nested and one of its simple cycles is of odd length, there exists a choice of parameters (α, P, μ) for which $\rho^{\text{net}} < \rho^{\text{BN}}$ and $UI > 0$.

[1] The restriction to odd length is necessary. If each activity has at most two resources and the collaboration graph is a simple non-nested cycle of even length, then the graph is bipartite in which case Theorem 4.1 shows that $\rho^{\text{BN}} = \rho^{\text{net}}$.

The fact that item (iii) allows to "choose" the parameters is important. For non-nested networks UBI might be equal to 0 for certain parameter values but be strictly positive for others. We explore this further in the next subsection.

4.1. Non-nested networks

We now focus specifically on non-nested networks and to derive some parameter-specific necessary conditions that relate configurations to bottlenecks.

Example 4.2 Consider the BC++ network in Figure 6. Assume that $m_i \equiv 1$ and that the input rate to activity 1 is $\alpha_1 = 1/3$. The incidence matrix A for this network is given by

$$A = \begin{pmatrix} 1\,0\,1\,1\,0 \\ 1\,1\,0\,0\,1 \\ 0\,1\,1\,0\,0 \end{pmatrix},$$

and does not satisfy any of our sufficient conditions. The load is $\sum_i A_{ki}(\lambda_i m_i) = (1, 1, 2/3)$ such that resources 1 and 2 are the bottlenecks and the SPP has the solution $\rho^{\mathrm{BN}} = 1$. Also, note that $\lambda_i m_i = 1/3$ for all $i = 1, \ldots, 5$. By directly solving the SPPC linear program we find that the network capacity equals the bottleneck capacity.

The feasible configurations that utilize the bottlenecks correspond to the activity sets $\mathcal{A}_1 = \{3, 5\}$, $\mathcal{A}_2 = \{4, 2\}$, $\mathcal{A}_3 = \{4, 5\}$ and $\mathcal{A}_4 = \{1\}$. Note that, for each of these sets $\pi(\mathcal{A}_i) \le \min_{i \in \mathcal{A}_l} \lambda_i m_i$ (otherwise there would be an activity that is allocated more time than it can use). A necessary condition for the bottlenecks to not suffer unavoidable idleness is that they are utilized 100% or that $\sum_{l=1}^4 \pi(\mathcal{A}_l) \ge 1$. This requires that $\sum_{l=1}^4 \min_{i \in \mathcal{A}_i} \lambda_i m_i \ge 1$. In this case, indeed,

$$\sum_{l=1}^{4} \min_{i \in \mathcal{A}_l} \lambda_i m_i = \lambda_4 m_4 + \lambda_5 m_5 + \lambda_5 m_5 + \lambda_1 m_1 = 4/3 \ge \rho^{\mathrm{BN}} = 1.$$

The fact that $\rho^{\mathrm{BN}} = \rho^{\mathrm{net}}$ for this network is, however, dependent on the service-time means $m_i, i = 1, \ldots, 5$. To see this, consider the service-time vector $m = (1/2, 1, 2 - \epsilon, 1/2, 1/4)$ with $0 < \epsilon < 5/4$. Then, $\sum_i A_{ki} \lambda_i m_i = \alpha_1 (3 - \epsilon, 7/4, 3 - \epsilon)$ so that resources 1 and 3 are the bottlenecks with capacity $1/(3 - \epsilon)$. The activity sets that use the bottlenecks are $\mathcal{A}_1 = \{3\}, \mathcal{A}_2 = \{4, 2\}$ and $\mathcal{A}_3 = \{3, 5\}$. Then,

$$\sum_{l=1}^{3} \min_{i \in \mathcal{A}_l} \lambda_i m_i = \lambda_3 m_3 + \lambda_4 m_4 + \lambda_5 m_5 = \frac{3 - \epsilon - 1/4}{3 - \epsilon} < 1 = \rho^{\mathrm{BN}}.$$

This means that the feasible configuration vectors cannot utilize the bottlenecks 100% of the time so that they suffer unavoidable idleness. (Equivalently, the SPP is assigning positive probabilities to infeasible configuration vectors.) ■

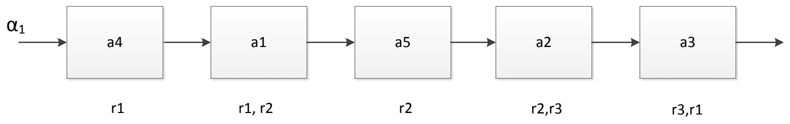

Figure 6 The BC++ network whose unavoidable idleness depends on the service-time means.

We generalize the arguments in Example 4.2 to the following lemma:

Lemma 4.3 (**A condition for non-nested networks**) *Suppose that* $\rho^{BN} = 1$ *(in particular,* $\rho^{net} \geq 1$*). If*

$$\sum_{\mathcal{A}\in\mathbb{C}:\mathbf{BN}\subseteq\mathcal{R}(\mathcal{A})} \min_{i\in\mathcal{A}} \lambda_i m_i < 1, \tag{16}$$

then the network features unavoidable idleness. Also, $\rho^{net} \geq \min_{k\in\mathbf{BN}} \left(\sum_{\mathcal{A}:k\in\mathcal{R}(\mathcal{A})} \min_{i\in\mathcal{A}} \lambda_i m_i\right)^{-1}$.

This lemma bridges two "worldviews": the bottleneck view (the SPP) and the network view (the SPPC). Intuitively speaking, (16) implies that the SPP is assigning strictly positive probabilities to "infeasible" configurations (that is, it requires that the bottleneck be assigned simultaneously to two activities that require its participation – i.e, that the resource is split). In Example 3.1 the SPP splits each of the three resources so that half a unit of each resource is assigned to an activity.

5. Collaboration and Scale: Multi-server Networks

In this section we consider multi-server networks with $n_k \in \mathbb{N}$ units of resource type $k \in \mathcal{K}$. The vector $\mathbf{n} = (n_1, \ldots, n_K)$ is the resource-units, or "staffing", vector. The multi-server scenario here should be interpreted as follows: a job processed in activity i requires *one* unit from each of the resource-types $k \in \mathcal{K}$ for which $A_{ki} = 1$. With multiple units per resource type, one can process multiple jobs (in one or multiple activities) in parallel. In the BC network, if there are three resource units of type 1 and three of type 2, the controller can choose to assign two units of each resource type to the collaborative activity 1 and one unit of resource of each of types 1 and 2, to activities 2 and 3 respectively.

For multi-server networks, the SPP considers the load of the entire type-k resource group

$$\begin{aligned} \rho^{\mathrm{BN}}(\lambda) = \text{minimize} \quad & \rho \\ \text{s.t.} \quad & \textstyle\sum_i A_{ki}\lambda_i m_i \leq \rho n_k, \text{for all } k \in \mathcal{K}. \end{aligned} \tag{17}$$

If $n_k \equiv 1$ this reduces to the original SPP. As before, the input to this SPP captures the load of resource k. Specifically, $\lambda_i m_i$ can be interpreted as the average number of resource units required by activity j from *each of the resource types participating in the processing of* j. Fixing the service times m, we say that a staffing vector $\mathbf{n}$ is feasible for λ if the SPP has the optimal value $\rho^{\rm BN}(\lambda) \le 1$.

The "world" of configuration vectors is richer in the multi-server case. A configuration not only defines which activities are performed simultaneously but also how many units of each resource are allocated to each of the activities in the configuration. Here, activities i and j can participate in a given configuration even if $\mathcal{S}(i,j) \neq \emptyset$ since one can assign only some of the units of a resource $k \in \mathcal{S}(i,j)$ to activity i and the remainder to activity j without creating a conflict. A configuration a is, here, an integer-valued vector of dimension I such that the i^{th} entry specifies the number of units of resource (of each resource type $k \in \mathcal{R}(\{i\})$) assigned to activity i under the configuration a. (This suffices in order to capture a configuration since we assume that each activity requires the same number of units from each resource.) In the BC network, the configuration vector $(n,0,0)'$ means that n units of *each of the resources involved in this activity* are assigned to this activity.

Formally, given incidence matrix A and staffing vector $\mathbf{n} = (n_1, \ldots, n_K)$, a configuration vector $a = (a_1, \ldots, a_I) \in \mathbb{N}^I$ is feasible if

$$\sum_{i \in \mathcal{I}} A_{ki} a_i \le n_k, \text{ for all } k \in \mathcal{K}. \tag{18}$$

The family of feasible configuration vectors depends on the vector $\mathbf{n} = (n_1, \ldots, n_K)$ and is denoted by $\mathcal{N}(\mathbf{n})$. Given $\mathbf{n} \in \mathbb{N}^K$, the SPPC is identical to the single server case. That is,

$$\begin{aligned} \rho^{\rm net}(\lambda) = \text{minimize} \quad & \rho \\ \text{s.t.} \quad & \textstyle\sum_{a \in \mathcal{N}(\mathbf{n})} a\pi(a) = \lambda m, \\ & \textstyle\sum_{a \in \mathcal{N}(\mathbf{n})} \pi(a) \le \rho, \\ & \pi \ge 0. \end{aligned} \tag{19}$$

One expects that unavoidable bottleneck idleness will shrink as the number of resource units grows: The simple intuition is that for a number of resources $n = (n_1, \ldots, n_K)$ that has an SPP solution $\rho^{\rm BN}(\lambda) \le 1$, we could permanently assign $\lceil \lambda_i m_i \rceil$ units of each resource $k \in \mathcal{R}(\{j\})$ to activity j. In the end there will be a deficit of at most I units of each resource. Yet, as n and λ grow, this deficit will become negligible relative to the number of resource units.

This intuition deserves a formalization. Our base network has parameters α (and $\lambda = Q\alpha$) and $n = (n_1, \ldots, n_K)$. We introduce an integral scalar θ and consider a sequence of networks, indexed by θ, so that the input rate in the θth network is $\alpha^\theta = \alpha\theta$ (and $\lambda^\theta = \theta\lambda$ and $\mathbf{n}^\theta = \theta\mathbf{n}$). The SPP and SPPC are re-written to reflect the dependence on θ (note, however, that with this scaling, the SPP's objective value does not depend on θ, i.e, $SPP(\theta) \equiv SPP$ as the bottleneck utilization remains unchanged under this scaling). The set of configuration vectors $\mathcal{N}(\mathbf{n}^\theta)$ does, however, depend on θ and we let $\rho^{\rm net}(\theta)$ be the corresponding solution to SPPC.

Lemma 5.4 *Suppose that SPP has a solution $\rho^{BN} \le 1$. Then, $\rho^{net}(\theta) \downarrow \rho^{BN}$, as $\theta \to \infty$,*

While correct in an asymptotic sense, the splitting intuition that underlies this result is too conservative – it is equivalent to restricting attention to a limited family of configuration vectors. It must be (and is indeed) the case that in some networks a small finite number of resource units will do the trick.

Example 5.3 Let us re-visit the BC+ network. If $m_i \equiv 1$, all resources are bottlenecks with capacity 1/2. With $\alpha = 1/2$ and $\theta = 2$ (so that $\alpha^{\theta} = \alpha\theta = 1$) we can permanently assign one unit of each resource to each activity where it is required, thus breaking all collaboration requirements.

This splitting is facilitated by, and is specific to, these particular service rates. With different parameters, we may still be able to get away with two units of each resource without splitting by properly allocating time to configurations. For example, assume that $m_1 = m_3 = \epsilon < 1/2$ and $m_2 = 2 - \epsilon$. With $\alpha = 1$ the number of units of resource 2 that are needed is $\lceil 2\alpha \rceil$ (since the load on resource 2 is 2 per job). Taking the splitting approach one assigns $\lceil \alpha\epsilon \rceil$ of the units of resource 1 to activity 1 and the remainder to activity 2. With "unsplittable" resources this works only if $\alpha\epsilon$ is an integer. In principal, ϵ may be an irrational number in which case, $\alpha\epsilon$ cannot be an integer.

Instead, we show that we can construct a solution to the SPPC (still assuming $m_1 = m_3 = \epsilon < 1/2$ and $m_2 = 2 - \epsilon$) that has $\rho^{\text{net}} = \rho^{\text{BN}}$ (and $UBI = 0$) by using *time splitting* between suitably chosen configuration vectors. Assume that $\mathbf{n} = (2, 2, 2)$. A solution to the SPPC is obtained by assigning positive weights to the configuration vectors

$$a^1 = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \text{ and } a^2 = \begin{pmatrix} 0 \\ 2 \\ 0 \end{pmatrix},$$

and setting $\pi(a^1) = \epsilon$ and $\pi(a^2) = 1 - \epsilon$ so that $(a^1\pi(a^1) + a^2\pi(a^2))_2 = 2 - \epsilon = \lambda_2 m_2$. ■

The second part of this last example may lead the reader to conjecture that $\rho^{\text{BN}} = \rho^{\text{net}}$ should mostly hold also for moderately sized systems and facilitated by carefully allocating time to corresponding configurations. This is not generally true.

Example 5.4 Re-visit yet again the BC+ network. Let $m_1 = m_3 = 1/4$ and $m_2 = 2 - 1/4$ (i.e. $\epsilon = 1/4$). Resources 2 and 3 are bottleneck resources with capacity of 1/2 each, while the capacity of resource 1 equals 2. Thus, to process input rate $\alpha_1 = 401$ (to activity 1), the minimal number of resource units of each type is given by $\mathbf{n} = (\lceil 401/2 \rceil, \lceil 2 * 401 \rceil, \lceil 2 * 401 \rceil) = (201, 802, 802)$.

With this "minimal" number of units, to be able to process all the input, we must allocate time only to configurations in which all of the 802 units of each of resources 2 and 3 are used. Also, while resource 1 is not a bottleneck with this capacity, using only 200 units of this resource does not suffice to process the load of $401/2 = 200.5$ on this resource. We must allocate a positive time to a configuration in which all resource units are used. Yet, there exists no such configuration–there exists no $n \in \mathbb{N}^3$ such that $n_1 + n_2 = 802$, $n_2 + n_3 = 802$ and $n_1 + n_3 = 201$–and we conclude that this network features unavoidable bottleneck idleness. ∎

The argument in this last example leads to a simple necessary condition for the network capacity to equal the bottleneck capacity.

Lemma 5.5 *If the process capacity equals the bottleneck capacity, then there exists a configuration vector $a \in \mathcal{N}(\mathbf{n})$ that fully utilizes each bottleneck: for each $k \in \mathbf{BN}$, $\sum_i A_{ki} a_i = n_k$.*

Fortunately, the benefits of the nested collaboration structure extend to multi-server networks:

Theorem 5.6 *A multiserver network that satisfies the bipartite condition of Corollary 4.1 or the conditions of Theorem 4.4 features no unavoidable bottleneck idleness. For any throughput $\lambda > 0$, UBI$(\lambda) = 0$ and $\rho^{BN}(\lambda) = \rho^{net}(\lambda)$.*

In principle, the number of servers of each type is decided by solving a staffing problem. If a server of type k costs c_k, a lower bound on the cost of capacity required to process all input is obtained by solving

$$\begin{array}{ll} \text{minimize} & \sum_{k\in\mathcal{K}} c_k n_k \\ \text{s.t} & \sum_i A_{ki}(\lambda_i m_i) \le n_k, \;\; k \in \mathcal{K}, \\ & \mathbf{n} \in \mathbb{N}^K, \end{array} \tag{20}$$

which has the trivial solution $n_k = \lceil \sum_i A_{ki} \lambda_i m_i \rceil$. Due to unavoidable bottleneck idleness, however, this may underestimate the true staffing requirements. One must, in general, make explicit the dependence of the configurations and solve

$$\begin{array}{ll} \text{minimize} & \sum_{k\in\mathcal{K}} c_k n_k \\ \text{s.t} & \sum_{a\in\mathcal{N}(\mathbf{n})} a\pi(a) = \lambda m, \\ & \sum_{a\in\mathcal{N}(\mathbf{n})} \pi(a) = 1, \\ & \mathbf{n} \in \mathbb{N}^K. \end{array} \tag{21}$$

As the set of feasible configuration vectors depends explicitly on the staffing vector $\mathbf{n}$, this adds significant complexity relative to (20). With tight capacities, as in Example 5.4, the collaboration architecture may induce unavoidable idleness even when the number of servers is large. Thus, maximizing efficiency (having capacity highly utilized) in networks with simultaneous collaboration poses a managerial challenge. With nested architectures, however, Theorem 5.6 removes any worry:

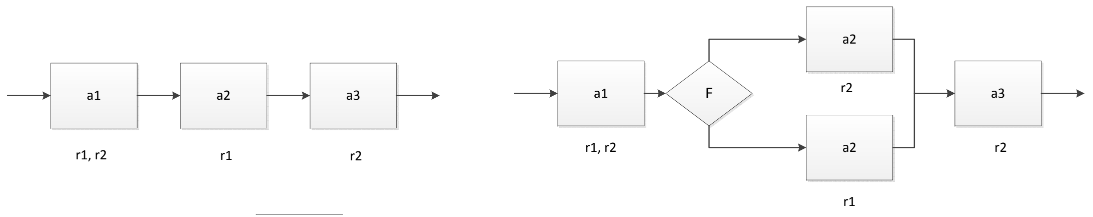

Figure 7 **(Left) No-flexibility in the BC network. (Right) A flexibility extended activity view.**

at least as far as capacity is concerned, staffing problems may be solved by considering each server pool in isolation.

Making progress beyond first order staffing problems by imposing constraints on the stochastic performance of the network (e.g., constraining average waiting times) is a desirable follow-up for this work. Questions of staffing are intimately entangled with questions of optimal control. The control of networks with collaboration is mostly uncharted territory and offers a significant challenge that must be resolved before turning to staffing with refined performance constraints; see §7.

6. Unavoidable Bottleneck Idleness and Flexibility

In modeling flexibility it is useful to consider *activity resource-groups* and *discretionary routing*. In the original BC network (left panel of Figure 7) the resource group $\{1,2\}$ performs activity 1 and the groups $\{1\}$ and $\{2\}$ perform activities 2 and 3, respectively. Flexibility expands the family of activity-resource groups. Training resource 2 to perform activity 2 *introduces routing discretion*: there are now two optional resource groups (each, in this case, containing an individual resource) $\{1\}$ and $\{2\}$ that can process activity 1 and we can choose how to distribute the activity's load between them. This added discretion is graphically captured by the diamond-shaped decision node in the right panel of Figure 7.

The incidence matrix A is extended here to a resource-to-extended-activity matrix A_F. Having trained resource 2 to perform activity 2, the matrix for the flexible BC network is

$$A_F = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix},$$

where the 4th column corresponds to activity 2 being performed by resource 2. This is consistent with the framework of Harrison (2002)—flexibility enlarges the family of activities. The SPP is expanded correspondingly to include the discretionary activities:

$$\begin{aligned} \rho^{\mathrm{BN}}(\lambda) = \text{minimize} \quad & \rho \\ \text{s.t.} \quad & x_{2,1} + x_{2,2} = \lambda_2 m_2, \\ & \lambda_1 m_1 + x_{2,1} \le \rho, \\ & \lambda_1 m_1 + \lambda_3 m_3 + x_{2,2} \le \rho, \\ & x \ge 0, \end{aligned} \tag{22}$$

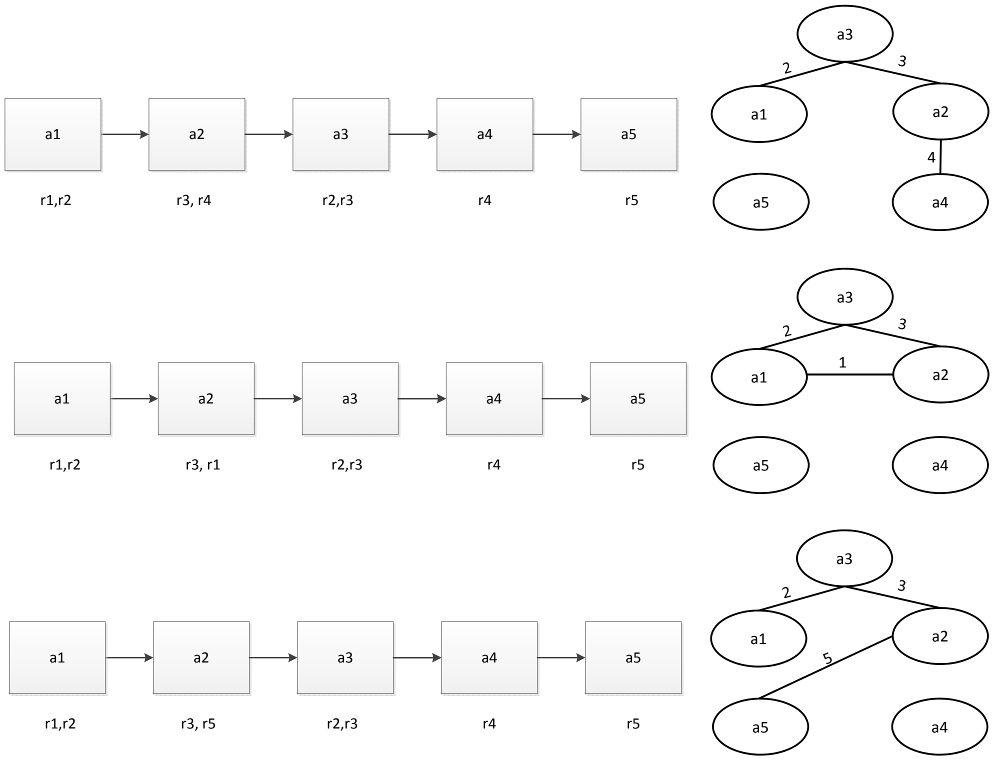

Figure 8 **(TOP) Without flexibility, the network has a nested collaboration architecture. Depending on which resource adds flexibility to activity 2 the collaboration architecture can become non-nested (MIDDLE) or remain nested (BOTTOM).**

where $x_{2,k}$ is the amount of activity 2 load assigned to resource k. (We will shortly provide a general formulation of the SPP.)

Adding flexibility also enlarges the family of feasible configuration vectors. In Figure 6, for example, training resource 3 to perform also activity 1 adds the configuration vector $(1,0,0,1,0)'$ corresponding to activities 1 and 4 being processed in parallel. As flexibility grows and the family of configuration vectors becomes "denser" one expects unavoidable idleness to shrink. Evidently, an upper bound on the network capacity is given by the *full flexibility case* in which all resources are trained to perform all tasks. Due to non-splitting, this upper bound will typically still be strictly smaller than the bottleneck capacity.[2]

Example 6.5 Consider the network at the top of Figure 8 with service time vector $m = (1,1,1,2.5,1)$. This network has a nested collaboration architecture and, by Theorem 4.4, features

[2] It is assumed here that the number of resource units is the same across all groups that can process an activity. Activity 1 in the BC network requires, for example, two units of resource. The two units cannot be replaced by a single unit with one "arch-skill." Such pooling of skills may be viewed as process re-design. We assume here that the process is given but that some skill requirements can be satisfied by several resource types.

no unavoidable idleness. Resource 4 is the bottleneck with a load of 3.5 so that the bottleneck capacity of 1/3.5 equals the network capacity. Assume now that resource 1 is trained also to replace resource 4 in activity 2. The SPP then has a maximal throughput of 0.4. In the (unique) optimal solution activity 2 is assigned entirely to the resource group $\{3,1\}$ (the group $\{3,4\}$ does not do any work on this activity). In this example, to achieve the SPP optimal solution, the SPPC must allocate no weight to those configurations where resource 4 works at activity 2. The remaining configuration vectors are those that correspond to the collaboration graph in the middle, which contains a strongly non-nested cycle. Resource 4 is still the bottleneck but now with a load of 2.5. With $\alpha = 0.4$ (the input to the first station) the SPPC has the optimal solution $\rho^{\rm net} = 1.1 > 1 = \rho^{\rm BN}$. The maximal throughput for which the SPPC has $\rho^{\rm net} \le 1$ is $1/2.75 < 0.4$. The added flexibility of resource 1 increased the network the capacity from 1/3.5 to 1/2.75 but by less than the amount predicted by bottleneck analysis.

Consider the alternative flexibility investment that cross-trains resource 5 to replace resource 4 in activity 2. As before resource 4 is still the bottleneck and the theoretical capacity is 0.4. Now, however, the $\rho^{\rm net} = 1 = \rho^{\rm BN}$ as the graph induced by the optimal solution to the SPP is acyclic.

Thus, in choosing between flexibility investments that are equally costly and yield the same value for theoretical capacity it is "safer" to take the one that does not introduce non-nested cycles. ■

To generalize the observations made in this example and provide some prescriptions we must introduce some notation. We let $\mathcal{G}(i)$ be the family of resource groups that can process activity i. In the no-flexibility setting, the family $\mathcal{G}(i)$ contains a single group of resources – this is the group $\mathcal{R}(\{i\})$. Our restriction that the number of resource units required is fixed means that $|G|$ is constant for all $G \in \mathcal{G}(i)$. For $G \in \mathcal{G}(i)$, denote $A_{k,(iG)} = 1$ if resource k participates in the processing of activity i as part of resource group G. Let

$$\mathcal{I}^E = \{(i,G) : i \in \mathcal{I}, G \in \mathcal{G}(i)\}$$

be the family of extended activities. The SPP, augmented to include flexibility, is then given by

$$\begin{aligned} \rho^{\rm BN}(\lambda) = \min\ & \rho \\ \text{s.t. }\ & \textstyle\sum_{(i,G)\in\mathcal{I}^E} x_{iG} = \lambda_i m_i,\ i \in \mathcal{I}, \\ & \textstyle\sum_{(i,G)\in\mathcal{I}^E : k\in G} A_{k,(iG)} x_{iG} \le \rho,\ k \in \mathcal{K}, \\ & \rho, x \ge 0. \end{aligned} \tag{23}$$

For $G \in \mathcal{G}(i)$, the variable x_{iG} should be interpreted as the time that resource group G works on activity i. We use the notation $(x^{\rm BN}, \rho^{\rm BN})$ to denote an optimal solution. The bottleneck resources are

$$\mathbf{BN} = \Big\{ k \in \mathcal{K} : \sum_{(i,G)\in\mathcal{I}^E : k\in G} x^{\rm BN}_{iG} = \rho^{\rm BN} \Big\},$$

which is a strict generalization of (6). A feasible configuration set is a collection of extended activities such that the associated resource groups do not overlap. Consequently, the family of feasible configuration sets is

$$\mathbb{C} := \Big\{ \mathcal{A} \subseteq \mathcal{I}^E : \bigcap_{(i,G)\in\mathcal{A}} G = \emptyset \Big\}. \tag{24}$$

A feasible configuration *vector a* is constructed from a feasible configuration set, as before, by letting $a_{iG}(\mathcal{A}) = 1$ for all $(i,G) \in \mathcal{A}$. The SPPC becomes:

$$\begin{aligned} \rho^{\text{net}}(\lambda) = \text{minimize} \quad & \rho \\ \text{s.t.} \quad & \textstyle\sum_G \left(\sum_{\mathcal{A}\in\mathbb{C}} a(\mathcal{A})\pi(\mathcal{A})\right)_{iG} = \lambda_i m_i, \ i \in \mathcal{I}, \\ & \textstyle\sum_{\mathcal{A}\in\mathbb{C}} \pi(\mathcal{A}) \leq \rho, \\ & \rho, \pi \geq 0. \end{aligned} \tag{SPPC}$$

For the remainder of this section the terms SPP and SPPC refer to these extended versions.

The extended collaboration graph: The collaboration graph includes an edge between two extended activities (i_1, G_1) and (i_2, G_2) if $G_1 \bigcap G_2 \neq \emptyset$. It is useful to look at resource-group combinations that are "active" under the solution x^{BN} to the SPP. Fix a solution $(x^{\text{BN}}, \rho^{\text{BN}})$ to the SPP. An activity-resource group pair (i,G) is an extended activity under x^{BN} if $G \in \mathcal{G}(i)$ and $x^{\text{BN}}_{iG} > 0$. The *extended collaboration graph for x^{BN}* is then constructed as before from these extended activities (and does not include extended activities for which $x^{\text{BN}}_{iG} = 0$).

One can now define nested-sharing cycles limiting attention to extended activities that have positive weights under x^{BN}. We say that *the extended collaboration architecture* (for x^{BN}) is nested if all cycles are nested. Obviously, if the extended collaboration graph is itself acyclic then so is the graph for x^{BN} for each solution x^{BN} to the SPP. The following is a corollary of Theorem 4.4.

Theorem 6.7 *Fix $\lambda \geq 0$. If the extended collaboration architecture for $x^{BN}(\lambda)$ is nested then the network features no unavoidable bottleneck idleness.*

In the BC network with added flexibility of Figure 7, the extended graph is itself acyclic (for each solution x^{BN} to the SPP) so that the extended collaboration architecture is nested. This is not the case in the network of Example 6.5. There, if x^{BN} uses the extended activity corresponding to resources 3 and 1 performing activity 2, the extended architecture is non-nested. A weaker sufficient condition is stated in the online supplement.

We say that two resources k and l are part of the same collaborative pool if there exists a path between them in the extended collaboration graph. The following corollary states that cross-training across collaborative pools is a "safe" flexibility investment. In graph terms such investments correspond to adding an edge to previously disconnected components of the collaboration graph.

Corollary 6.2 *Training a resource k to perform a task of a resource in a different collaborative pool does not introduce unavoidable idleness into a network with a nested collaboration architecture.*

On flexibility, bottlenecks and coverings: In Lemma 4.3 we learned that it is useful to map the unavoidable idleness to "coverings" of the bottlenecks. A feasible configuration set $\mathcal{A}$ (see (24)) is said to cover the bottlenecks if each bottleneck k is used for one of the activities in the configuration $\mathcal{A}$: for each $k \in \mathbf{BN}$ there exists $(i,G) \in \mathcal{A}$ such that $k \in G$. We then write $\mathbf{BN} \subseteq \mathcal{R}(\mathcal{A})$.

Example 6.6 Consider the BC++ network with service times $m = (1/2, 1, 7/4, 2, 1/4)$. Suppose first that the resource-activity mapping is as in Figure 6. Without flexibility, resource 1 is the only bottleneck with capacity 4/17, which equals the network capacity (direct computation shows that there is no unavoidable idleness, i.e, $\rho^{\text{BN}} = \rho^{\text{net}}$).

Introduce flexibility by training resource 2 so it can replace resource 1 in either activities 3 or 4. Then resources 1 and 2 are bottlenecks with bottleneck capacity 1/3 – the flexibility allows us to shift some work from resource 1 to resource 2. By adding this flexibility we also make the configuration vector $(0,0,1,1,0)'$ feasible. The maximal α_1 for which SPPC has a solution $\rho^{\text{net}} \leq 1$ is 1/3.25 – which is the full-flexibility bound and, thus, cannot be further improved.

The flexibility we added here has a special feature: assigning resource 2 to activity 3 requires it to collaborate with resource 3 which **is not** required for any work with the bottleneck resource 1 so we are "free" to exploit this flexibility: we added a bottleneck but also a configuration that covers both bottlenecks – the previous one (resource 1) and the new one (resource 2). ∎

Flexibility is likely then to be less beneficial when it adds an extended activity that conflicts with the current bottlenecks. Cross-training resource k to perform activity i as part of resource group G is formally captured by adding an extended activity (i,G) with $k \in G$. The following lemma shows that given a set of bottlenecks and the corresponding bottleneck coverings one can predict that certain investments fail to remove the unavoidable idleness and hence require a careful analysis.

Lemma 6.6 *(i) Let* $\mathbf{BN}$ *be the set of bottlenecks and* $\mathbb{C}$ *be the family of feasible configuration sets. Let* λ *be such that* $\rho^{BN}(\lambda) = 1$*. The network features unavoidable idleness if*

$$\sum_i \min_{\mathcal{A} \in \mathbb{C}: \mathbf{BN} \subseteq \mathcal{R}(\mathcal{A})} \lambda_i m_i < 1. \tag{25}$$

(ii) Regardless of whether (25) *holds, suppose that resource* $k \notin \mathbf{BN}$ *is cross-trained by adding the extended activity* (i, G_i) *(with* $k \in G_i$ *and* $\mathbf{BN} \bigcap G_i = \emptyset$*) and that* λ^F *is such that* $\mathbf{BN}^F = \mathbf{BN} \bigcup \{k\}$

are post cross-training bottlenecks with an SPP value $\rho^{BN^F}(\lambda^F) = 1$. *Let* $\mathbb{C}^F$ *be the updated family of feasible configuration sets. The network features unavoidable idleness if* $\mathcal{A}\bigcup(i,G_i) \notin \mathbb{C}^F$ *for each covering* $\mathcal{A}$ *of* $\mathbf{BN}$ *(adding the extended activity* (i,G_i) *to any of the original* $\mathbf{BN}$ *coverings* $\mathcal{A} \in \mathbb{C}$ *results in an infeasible configuration set) and if*

$$\sum_i \min_{\mathcal{A}\in\mathbb{C}:\mathbf{BN}\subseteq\mathcal{R}(\mathcal{A})} \lambda_i^F m_i < 1.$$

For the incremental analysis in Lemma 6.6, one does not need to identify the new (post-investment) configuration sets: the evaluation in part (ii) uses the pre cross-training family $\mathbb{C}$ of feasible configuration sets.

The guidelines that emerge from our flexibility discussions, examples and results are as follows:

1. *Architectures:* If the extended collaboration architecture under the optimal solution x^{BN} to the SPP is nested then $\rho^{\mathrm{BN}} = \rho^{\mathrm{net}}$ and the benefit of flexibility as projected by the conventional bottleneck analysis will materialize. From a design perspective, this means that when facing multiple options for flexibility (for which the SPP predicts the **same** improvement) one should choose, if possible, an option for which the extended graph has a nested structure; see Corollary 6.2.

2. *Bottlenecks and configurations:* Among several options of flexibility investments for which the SPP reflects the **same** increment, those options that add configurations that cover the (new set of) bottlenecks are likely to be more valuable. One must be careful with incremental flexibility investments that create conflicts with the bottleneck; see Lemma 6.6.

The discussion above only distinguishes flexibility investments that have the same bottleneck-based effect. Ideally, a framework should be developed that facilitates choosing between two distinct flexibility investments that generate different bottleneck capacity and different unavoidable idleness. Such development is beyond the scope of this paper as it requires going beyond the characterization of the existence of unavoidable idleness to developing tight bounds on its magnitude.

7. Concluding Remarks

In this paper we offered an initial exploration of the effect of simultaneous collaboration and resource sharing on network capacity and of its implications for conventional bottleneck analysis. We use this section to outline some directions that deserve further explorations.

Dynamic control: This paper has focused on deterministic ("fluid") analysis. The study of the dynamic control of networks with simultaneous collaboration is a promising direction for future research. Human operated services impose certain constraints on the dynamic control—resources cannot be split and preemption if often undesirable or plainly infeasible.

The maximization of throughput in non-splitting, non-preemptive settings is studied by Dai and Lin (2005), Jiang and Walrand (2010, Chapter 6) and Tassiulas and Ephremides (1992). In our terminology those articles take unavoidable idleness as given and are concerned with designing dynamic policies that achieve the throughput predicted by the SPPC. More refined objectives for collaborative networks – such as minimizing queue-holding costs – are mostly unexplored. With such studies in mind, an important observation to be made is that non-splitting, non-preemptive collaborative networks introduce a fundamental tradeoff between achieving high throughput (resource utilization or "efficiency") versus controlling queue lengths.

This tradeoff is best understood by first taking a step back and away from collaborative networks to consider a basic two-class single server $M/G/1$ queue. It is well known that the so-called $c\mu$ rule is optimal here towards minimizing long-run average linear holding costs. Under this rule, whenever the server becomes available it serves a customer from the non-empty queue with the highest value of $c_i\mu_i$. Suppose, however, that switching between classes requires a setup time. That is, after completing a service of a class-1 customer, if the server wishes to serve a class-2 customer it cannot do so immediately but only after some setup time has elapsed (the server, can, however, immediately serve another class-1 customer). Frequent changeovers will allow the controller to keep the expensive queue short but lead to a capacity loss. Infrequent changeovers eliminate some of the capacity waste but at the expense of longer queues of the expensive class; this model has been studied in Reiman and Wein (1998).

Returning to collaborative networks, we observe that the changeovers (setups) arise here endogenously. In our BC network, consider a time at which both resources are working on their individual tasks (resource 1 in activity $a2$ and resource 2 in activity $a3$) and the controller wishes to move them to work on the collaborative activity $a1$. If, for example, resource 1 completes his current processing first, it will have to wait until resource 2 completes its current job. This introduces a *synchronization idleness* that is unavoidable with random service times and that is the analogue of the exogenous setup time in the single-server example above.

This tradeoff between utilization and queue length presents an interesting further object of study.

Decentralized control – setting the incentives right: The dynamic control of queueing networks typically assumes a central controller that can allocate resources to activities in real time. The controller can guarantee that resources are "at the right place at the right time" to perform collaborative activities. In various human-operated services (health-care being a primary example here as well) resources have "discretion" or degrees of freedom in choosing jobs for processing. Each of the "players" has its own objectives and the aggregation of what may be individually-optimal action may lead to overall sub-optimal performance. The BC network again offers a case in point – in the introduction we have shown that letting the resources prioritize their individual task leads to significant capacity loss. Yet, it seems clear that one can find individual objective (utility) functions that would render such a prioritization scheme the individually rational thing to do.

Collaboration, throughput and incentives offer promising candidates for empirical investigations of questions such as (1) does collaboration indeed work better in nested architectures in the presence of individually rational agents, or (2) is there a relationship between our notion of nested collaboration and some organizational notions of teamwork and hierarchy?

Practical applications – the case of health-care: There are tasks in health-care delivery were simultaneous collaboration of multiple resources is required. This is evident in the case of surgeries but is also true for more mundane tasks as patient discharge from an internal ward. The structure of collaboration in patient flow is not always evident and a thorough understanding of these processes can suggest meaningful ways to change and expand on the initial study we conducted here.

Hospital beds are one type of resource that presents an interesting challenge to the study of collaborative processing. On one hand, beds (like other hospital equipment such as MRI machines) could be viewed as just another processing resource that collaborates with other resources (doctors, nurses etc.) in "serving" patients. Yet, beds are different in a subtle way. One cannot embed these directly into the framework in this paper. In principle one can model, for example, a visit of the doctor to a patient as an activity in which the doctor and the bed collaborate in processing the patient followed by a period in which the patient waits for the next round which can be modeled as another (distinct) activity. This requires a departure from the standard framework to capture the fact that, in between activities, the patient remains on *the same bed* for the remainder of her flow through the network.

One may generate an upper bound on the network's capacity by pretending that resources can be re-allocated at the end of an activity. Our framework would apply to this relaxation.

Collaboration seems to be prevalent in service networks, and the complexities of these networks are important to understand and model so as to understand the tradeoffs imposed by collaboration. It is our hope that our work in this paper can serve as a stepping stone.

References

Bassamboo, A., R.S. Randhawa, A. Zeevi. 2010. Capacity sizing under parameter uncertainty: Safety staffing principles revisited. *Mngt. Sci.* **56**(10) 1668–1686.

Brucker, P., A. Drexl, R. Möhring, K. Neumann, E. Pesch. 1999. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research* **112**(1) 3–41.

Chan, C. W., G. Escobar, G. Yom-Tov. 2014. When to use speedup: An examination of service systems with returns. *Oper. Res.* **62**(2) 462–482.

Chopra, S., R. C. Savaskan-Ebert. 2013. Weight solutions clinic bariatric surgery center Kellogg Case Publishing, Case Number: 5-104-006.

Dai, J. 1999. Stability of fluid and stochastic processing networks. MaPhySto Miscellanea Publication.

Dai, J. G., W. Lin. 2005. Maximum pressure policies in stochastic processing networks. *Oper. Res.* **53**(2) 197–218.

Dobson, G., U. S. Karmarkar. 1989. Simultaneous resource scheduling to minimize weighted flow times. *Oper. Res.* **37**(4) 592–600.

Graves, S. C., B. T. Tomlin. 2003. Process flexibility in supply chains. *Mngt. Sci.* **49**(7) 907–919.

Harrison, J. M. 2002. Stochastic networks and activity analysis. Y. Suhov, ed., *Analytic methods in Applied Probability, In memory of Fridrih Karpelevich*. AMS, Providence, RI.

Haviv, M. 2013. *Queues: A Course in Queueing Theory*. Springer New York.

Herroelen, W., B. De Reyck, E. Demeulemeester. 1998. Resource-constrained project scheduling: a survey of recent developments. *Computers & Operations Research* **25**(4) 279–302.

Jiang, L., J. Walrand. 2010. Scheduling and congestion control for wireless and processing networks. *Synthesis Lectures on Communication Networks* **3**(1) 1–156.

Kelly, F. P. 1991. Loss networks. *Ann. Appl. Prob.* **1** 318–378.

Massoulie, L., J. W. Roberts. 2000. Bandwidth sharing and admission control for elastic traffic. *Telecommunication Systems* **15**(1-2) 185–201.

Reiman, M., L. Wein. 1998. Dynamic scheduling of a two-class queue with setups. *Oper. Res.* **46**(4) 532–547.

Roels, G., U. S. Karmarkar, S. Carr. 2010. Contracting for collaborative services. *Mngt. Sci.* **56**(5) 849–863.

Schrijver, A. 1998. *Theory of linear and integer programming*. Wiley, Chichester New York.

Shah, D., D. Wischik. 2012. Switched networks with maximum weight policies: Fluid approximation and multiplicative state space collapse. *Ann. Appl. Prob.* **22**(1) 70–127.

Stolyar, A. L. 2004. Maxweight scheduling in a generalized switch: State space collapse and workload minimization in heavy traffic. *Ann. Appl. Prob.* **14**(1) 1–53.

Tassiulas, L., A. Ephremides. 1992. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *Automatic Control, IEEE Transactions on* **37**(12) 1936–1948.

Van Mieghem, J. A. 2008. Pizza pazza Available from permissions@vanmieghem.us.

Williams, R. J. 1998. Diffusion approximations for open multiclass queueing networks: Sufficient conditions involving state space collapse. *Queueing Systems* **30**(1-2) 27–88.

Zachary, S., I. Ziedins. 1999. Loss networks and markov random fields. *Journal of Appl. Prob.* **36**(2) 403–414.

Online Supplement

Proof of Theorem 4.1: We first argue that given I, we can construct a network with I activities and $K = 3 + \sum_{i=3}^{I-1} i$ resources that has a complete collaboration graph and such that each resource has only two activities, i.e, $\sum_j A_{kj} = 2$ for all $k \in \mathcal{K}$. We take the collaboration graph of the BC+ network as the starting point for this construction. This is a complete graph with three nodes: 1,2 and 3. With $\mathcal{R}(\{1\}) = \{1,2\}$, $\mathcal{R}(\{2\}) = \{2,3\}$ and $\mathcal{R}(\{3\}) = \{3,1\}$ and has $K = 3$ resources as needed. Now assume we construct the required network for all network sizes less than or equal to $I-1$. We next add an activity (numbered I) as follows. For each node i in the current graph (a total of $I-1$ nodes) create a (new) resource k_i, assign it to activity i and add it to $\mathcal{K}$ (before this step $K = 3 + \sum_{i=3}^{I-2} i$). Assign all these new resources to the new activity I. That is, $\mathcal{R}(\{I\})$ is the set of these newly added resources. Each of the added resources k_i has two activities (activities i and I) and activity I is connected with an edge to each of the nodes of step $I-1$. After this step, then, we have a network with a complete collaboration graph, where each resource has two activities and $K = 3 + \sum_{i=3}^{I-2} i + (I-1) = K = 3 + \sum_{i=3}^{I-1} i$. Finally, set the service time and arrival rate to each activity i to $m_i = 1$ and $\alpha_i = 1/2$. Set $P = 0$. Then, $\rho^{BN} = \max_{k\in\mathcal{K}} \sum_j A_{kj} = 1$. However, the only feasible configuration vectors are the identity vectors so that to satisfy $(\sum_l \pi(e^l)e^l)_j = \lambda_j m_j = 1/2$ we must $\pi(e^j) = 1/2$ and, in turn, that $\rho^{\text{net}} = \sum_j \pi(e^j) = I/2$. ■

Proof of Theorem 4.2: We prove this result for multiserver networks and the single-server case follows as a special case. We start with the sufficiency. Consider the polyhedron

$$\Xi = \Big\{ x \geq 0 : \sum_i A_{ki} x_i \leq n_k,\ k \in \mathcal{K} \Big\}.$$

By definition, if the SPP (17) has a solution $\rho^{\text{BN}} \leq 1$, we have that $\sum_i A_{ki}\lambda_i m_i \leq \rho^{\text{BN}} n_k$ or, re-writing,

$$\sum_i A_{ki} \frac{\lambda_i m_i}{\rho^{\text{BN}}} \leq n_k,\ k \in \mathcal{K}.$$

In particular, $x^{\text{BN}} = (\lambda_1 m_1/\rho^{\text{BN}}, \ldots, \lambda_I m_I/\rho^{\text{BN}})$ is in the convex polyhedron Ξ and can be expressed as a convex combination of its extreme points. From the assumption that the extreme points are integer it follows that the extreme points of Ξ are integer valued for any integer right hand side. In particular, the extreme points of Ξ are feasible configuration vectors; see (18)). Thus, x^{BN} can be written as a convex combination of configurations: there exists $\pi \geq 0$ where $e'\pi = 1$ and

$$\sum_j \pi^j a^j = x^{\text{BN}},$$

where a^j are feasible configuration vectors. Setting $\hat{\pi} = \rho^{\text{BN}}\pi$ we have, as required, that

$$\sum_j \hat{\pi}^j a^j = \lambda m, \text{ and } \rho^{\text{net}} = \sum_l \hat{\pi}_l = \rho^{\text{BN}}.$$

For the necessity, assume that the polyhedron Ξ has a non-integral extreme point x^*. Choose (α, P, μ) such that $\lambda m = x^*$. Then, x^* cannot be expressed as a (non-trivial) convex combination of integer vectors in Ξ. In particular, there exists no π such that $\pi \geq 0$, $e'\pi \leq 1$ and with $\sum_i \pi_i a^i = 1$ for configuration vectors a^i. In particular, it must be the case that $\rho^{net} > 1 \geq \rho^{BN}$.

Finally, note that if the adjacency matrix A is Totally Unimodular (TUM) it follows that the extreme points of Ξ are integer valued (for any integer right hand side); see (Schrijver 1998, Corollary 19.2a). ■

Proof of Corollary 4.1: Notice first that we may ignore resources that have a single activity. Those resources must be assigned in full to their single activity and it suffices to consider the residual network. Here, each resource has exactly two activities with an edge connecting the two activities in the graph. In particular, each edge is associated with a single resource. There can be two resources collaborating on the two activities but in that case they can be treated as a single resource. Thus, we may assume without loss of generality that there is a one to one mapping between edges and resources. With each resource corresponding to an edge and each activity to a node, the matrix A^T (the transpose of A) is then (by the assumption of the corollary) the incidence matrix of a bi-partite graph and is hence totally unimodular; see e.g. (Schrijver 1998, page 273). The transpose of a totally unimodular matrix is itself totally unimodular and we conclude that A is a TUM matrix. In turn, by Theorem 4.3 the network features no unavoidable idleness. ■

The following will be used in subsequent proofs.

An auxiliary synchronization graph for networks with nested architectures: Recall that for a network to have a nested collaboration architecture it is not necessary that there be no cycles in the collaboration graph. It is only required that all cycles are nested-sharing cycles. For nested architectures we can construct an auxiliary acyclic graph that has the useful property that activities at the same "level" of the graph do not share resources – we will refer to this acyclic graph as the synchronization graph. This graph is a tool rather than a conceptual entity.

Recall that a nested-sharing cycle is a set of l connected nodes (activities) $i_0, \ldots, i_l$ such that

$$\mathcal{S}(i_k, i_j) \subseteq \mathcal{S}(i_m, i_j) \text{ for all } k, m, j \in \{1, \ldots, l\} : j > m > k.$$

We refer to i_0 as the highest rank activity in the cycle, i_1 the second rank etc. Following standard terminology we say that a simple path in a graph between nodes i and j is a set of *distinct* nodes $i, i_1, \ldots, i_l, j$ such that each two consecutive nodes are connected by an edge. Given a network, we construct the synchronization graph as follows:

0. **Initialization:** Set $\ell = 0$ and $\mathcal{C}^0 = \emptyset$. Add a fictitious node r and set $d_r = 0$. (think of the root as an activity with $\lambda_r = 0$ that uses all K resources.)

1. If $\mathcal{C}^\ell = \mathcal{I}$ stop. Otherwise, consider the maximal sharing between an activity outside the graph and one in the graph, i.e,

$$\mathcal{O} = \max\{i \notin \mathcal{C}^{\ell-1}, j \in \mathcal{C}^{\ell-1} : |\mathcal{S}(i,j)|\}.$$

Set $\mathcal{O} = 0$ if there are no such i and j.

2. $\mathcal{O} > 0$: Pick an activity $i_\ell \notin \mathcal{C}^{\ell-1}$ with $|\mathcal{S}(i_\ell, j)| = \mathcal{O}$ for some $j \in \mathcal{C}^{\ell-1}$. If taking an activity from a nested-sharing cycle take the highest-rank activity in that cycle not yet in the graph (if there are multiple nested-sharing cycles with activities not yet in the graph, take an activity from a cycle with the largest number of nodes).

Pick a node $j_\ell \in \arg\max_{j \in \mathcal{C}^{\ell-1} : |\mathcal{S}(i_\ell, j)| = \mathcal{O}} d_j$, add the edge (i_ℓ, j_ℓ) and set $d_{i_\ell} = d_{j_\ell} + 1$. (we are connecting i_ℓ to the activity with the greatest distance from the root among those that share resources with i_ℓ). Stop if there exist $j, k \in \mathcal{C}^{\ell-1}$ with $|\mathcal{S}(i_\ell, j)| = |\mathcal{S}(i_\ell, k)| = \mathcal{O}$ but $\mathcal{S}(i_\ell, j) \neq \mathcal{S}(i_\ell, k)$.

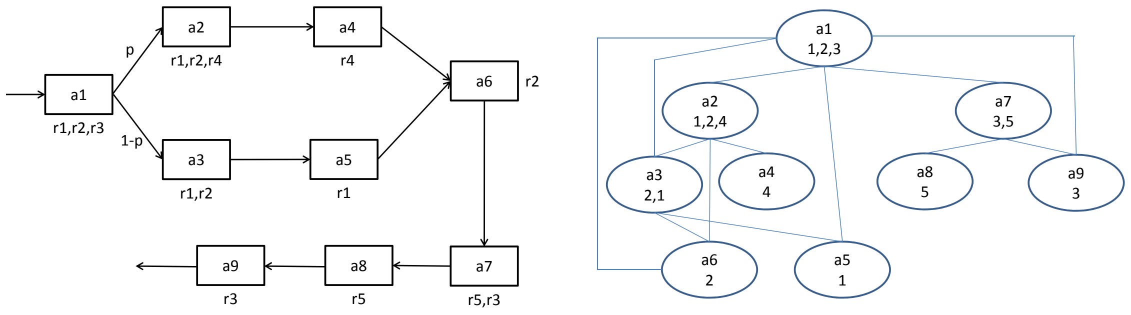

Figure 9 **A network with nested hierarchical architecture**

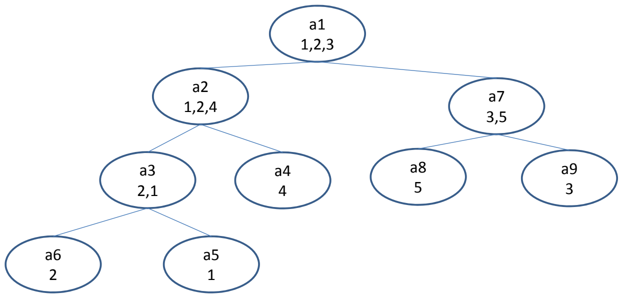

Figure 10 **A synchronization graph for the network in Figure 9**

3. $\mathcal{O}=0$ (there are no activities i,j one in the graph and the other out of it that share resources): pick an arbitrary $i_\ell \notin \mathcal{C}^{\ell-1}$ and add to the graph connecting it via an edge to the root. If adding an activity from a nested-sharing cycle take the one with the highest rank (if there are multiple nested-sharing cycles with activities not yet in the graph, take an activity from a cycle with the largest number of nodes). Set $d_{i_\ell}=1$.

Example A.1 Consider the network in Figure 9 together with its collaboration graph. The collaboration graph contains only nested-sharing cycles. The outcome of the algorithm applied to this network is as in Figure 10 (we removed here the fictitious root). ■

Since the choice of the edge to add in step 2 of the algorithm is arbitrary there can be multiple synchronization graphs but, importantly, the following holds.

Lemma A.1 *(1) If the network has a nested collaboration architecture the algorithm generates a graph with I nodes and no cycles and, (2) in this case, activities in the same level of the graph (i.e, with the same parameter d) do not share resources ($\mathcal{S}(i,j)=\emptyset$ if $d_i=d_j$).*

Proof: If the algorithm stops after I steps then all nodes were added with a single edge and no cycles were formed. Suppose, towards contradiction, that the algorithm stops after $\ell<I$ steps. In this step a node i_ℓ is added and there are j_ℓ and k_ℓ with $\mathcal{O}=|\mathcal{S}(i_\ell,j_\ell)|=|\mathcal{S}(i_\ell,k_\ell)|$ and $\mathcal{S}(i_\ell,j_\ell)\neq\mathcal{S}(i_\ell,k_\ell)$. We claim that the cycle containing i_ℓ, k_ℓ and j_ℓ must be a non-nested cycle. Suppose that these activities are, in fact, part of a

nested-sharing cycle. Let us further assume that k_ℓ was added to the graph after j_ℓ (the other case is argued identically). By assumption, $\mathcal{O} = |\mathcal{S}(i_\ell, j_\ell)| = |\mathcal{S}(i_\ell, k_\ell)|$. Since this is a nested-sharing cycle and k_ℓ has a lower rank than j_ℓ we have that $\mathcal{S}(j_\ell, i_\ell) \subseteq \mathcal{S}(k_\ell, i_\ell)$ and, in turn, $\mathcal{S}(i_\ell, j_\ell) = \mathcal{S}(i_\ell, k_\ell)$ which is a contradiction to the stopping rule. We may thus conclude that, if the network has a nested architecture, the algorithm ends with a tree that includes I nodes.

We argue next that if the architecture is nested then i and j with $d_i = d_j$ must have $\mathcal{S}(i,j) = \emptyset$. The case that i and j are not part of a cycle is trivial as, if $d_i = d_j$ and $\mathcal{S}(i,j) \neq \emptyset$, we would have in fact found a cycle containing i and j in the collaboration graph. To argue the case that i and j are part of a nested-sharing cycle, let ℓ be the first step in which a node i_ℓ is added with the property that $d_{i_\ell} \leq d_j$ for some $j \in \mathcal{C}^{\ell-1}$ with $\mathcal{S}(i_\ell, j) \neq \emptyset$. Let $k \neq j$ be such that (i_ℓ, k) is the edge that is added to the graph with node i_ℓ (if $k = j$ we would have $d_{i_\ell} = d_j + 1 > d_j$). Then, $d_j \geq d_{i_\ell} = d_k + 1$. In particular $d_k < d_j$. Note that $\mathcal{S}(j,k) \neq \emptyset$ as they are both part of a nested-sharing cycle containing j. Since i_ℓ is the first node added with the required property, the fact that $d_k < d_j$ implies that k was added to the graph before j (and has higher rank in the nested-sharing cycle containing both). By the definition of nested-sharing cycles we must then have that $\mathcal{S}(k, i_\ell) \subseteq \mathcal{S}(j, i_\ell)$ and, in particular, that $|\mathcal{S}(k, i_\ell)| \leq |\mathcal{S}(j, i_\ell)|$. Recall that also $d_j > d_k$ so that, when adding i_ℓ we would have added the edge (i_ℓ, j) instead of (i_ℓ, k). ■

Proof of Theorem 4.4 and Theorem 5.6: First, note that Theorem 4.4 is a special case of Theorem 5.6 with the staffing vector set to be the vector of ones. We divide the proof into two parts. In the first we treat nested architectures and in the second we treat weakly non-nested architectures.

Nested architectures: A known sufficient condition for the total unimodularity of the matrix A is that it (or a permutation of its rows) has the consecutive ones property; see (Schrijver 1998, Example 7, Chapter 19). We next prove that we can re-label the resources and permute the rows so that the 1s in each column (corresponding to an activity) appear consecutively.

Our starting point is the synchronization tree constructed above. We first re-organize the tree. We make sure that at every level of the tree the nodes with the least number of sons are far from the root. Formally, i is a parent node of j (and j the son of i) if there is an edge between them and $d_j = d_i + 1$. Returning to the example we used before, the graph in Figure 11(LHS) would be re-organized into the one on the RHS.

Proceeding with this example, we can now re-label resources following depth-first-search to traverse the tree. We first visit activity $a8$. This activity has the single resource 5 – we re-label this resource as 1 (i.e, $5 \to 1$). We then proceed to activity $a9$ and re-label 3 as 2. At this point labels 1 and 2 are already taken and the next available label is 3. In activity $a7$, $3,5$ is replaced with $2,1$ (or 1,2 for convenience of display) following the re-labeling already done in the son nodes of $a7$. We then visit node $a6$ and replace $2 \to 3$ and in activity $a5$ $1 \to 4$. In activity $a3$ we re-label based on the son nodes $a5$ and $a6$. For activity $a4$, the next available resource number is 5 so we re-label $4 \to 5$ and follow accordingly in activities $a2$ and finally $a1$. By the end of this procedure we have re-labeled the resources $(5,3,2,1,4) \to (1,2,3,4,5)$. This re-labeling guarantees the consecutive 1's property: $a1$, for example, uses resources 2,3 and 4 (previously 1,2,3), $a2$ uses 3,4,5 (previously 1,2,4), etc.

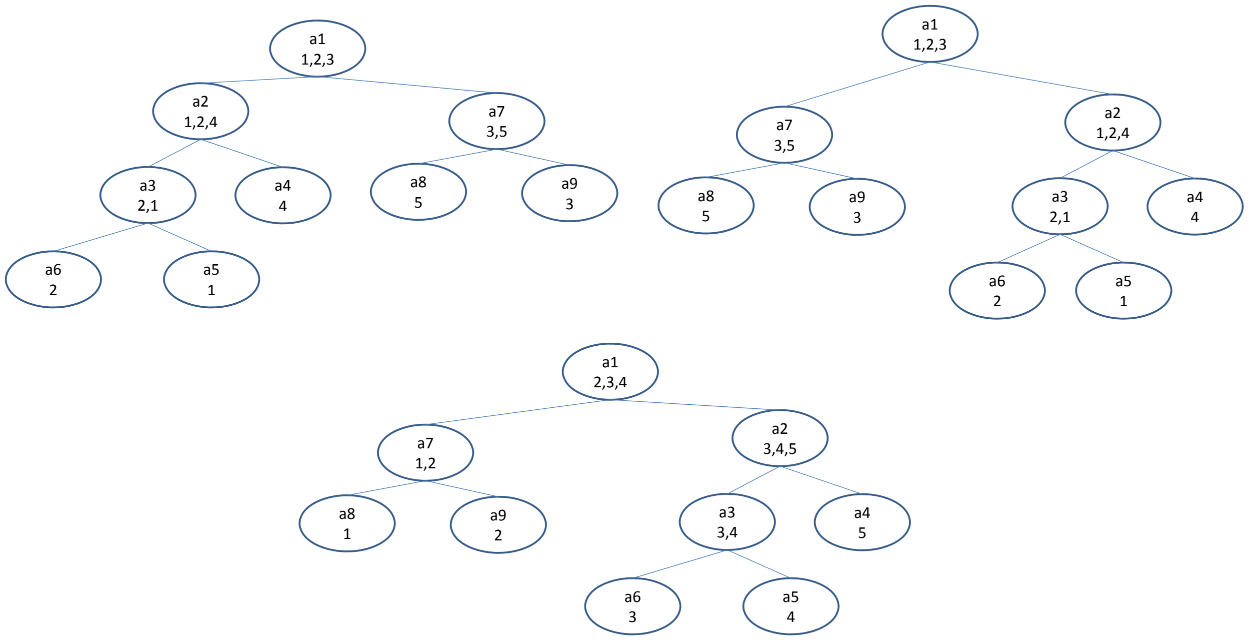

Figure 11 (LHS) A synchronization graph for the network in Figure 9 and (RHS) re-organized version and (BOTTOM) re-labeled resources

The following is the formalization of the re-labeling algorithm:

Initialize $num = 0$ and $z_0 = 0$. Each resource has a tuple containing its original number k, its current label $\ell(k)$ (which is initialized to k), and a binary variable $v(k)$ which is 0 initially and set to 1 once k is labeled. We take the following actions in step l of the depth first search:

1. If the node is a leaf (corresponding to activity i say), we label all unlabeled resources in this node in an arbitrary order starting with the numbers $num+1, \dots, num+|\mathcal{R}(\{i\})|$. We advance $num \leftarrow num + |\mathcal{R}(\{i\})|$. For each labeled resource k, we write $\ell(k)$ for its new label and set $v(k) = 1$.

2. If node i is not a leaf:

2a. If node i has a resource k that has not yet been marked (i.e. $v(k) = 0$): if i is on the left of the root assign it the number $z_l - 1$ (and change $z_l \leftarrow z_l - 1$). If i is on the right of the root, label $\ell(k) = num + 1$, set $v(k) = 1$ and advance $num \leftarrow num + 1$.

2b. Order the resources in each activity in increasing order of their labels. If after completing step $2a$ there is a gap in the labels of resources in activity i (there are resources $k, l \in \mathcal{R}(\{i\})$ such that $\ell(k) > \ell(l) + 1$ but no $\kappa \in \mathcal{R}\{i\})$ with $\ell(\kappa) = \ell(l) + 1$)) we take the following actions: Let k (with label $\ell(k)$) be the first resource after the gap. Let j be a son of i such that $k \in \mathcal{S}(i,j)$ (by Lemma A.1 there can be at most one such son node). Re-label all resources in the sub-tree rooted in j by shifting them by $-\ell(k)$. Repeat as long as there are gaps.

To illustrate step 2b consider Figure 12. The top graph on the left is the original one and the bottom graph on the left is the one obtained after applying all steps except for step 2b on the root node $a1$. Note that in the root node there is now a gap (between 2 and 4). In this last step we take the sub-tree rooted at

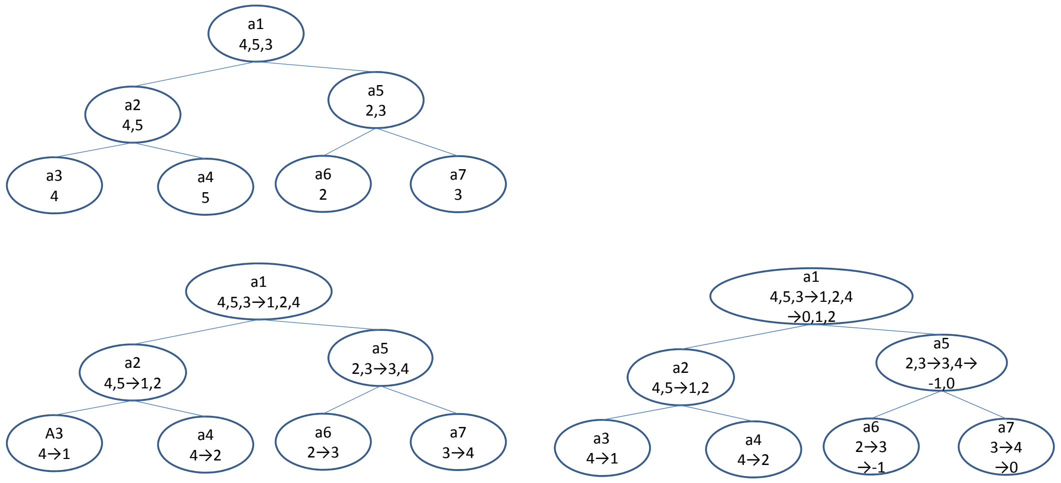

Figure 12 Relabeling example

$a5$ and shift all labeling by -4, thus creating the two new labels -1 and 0. All labels in the graph are now consecutive.

To argue that the resulting labeling has consecutive labels we perform induction on the step number. This is obviously true for the first visited leaf. Since no two leafs in the synchronization tree have shared resources, when a leaf is visited no resource is already labeled. Assuming that for all activities visited in step $l \leq k-1$ resources are consecutively numbered, the algorithm preserves this property. Let i be the activity visited in the k^{th} step: if step 2b is not applied to node i, it means there is no gap and the consecutive labeling is inherited from the son nodes because new resources are added to the left (if the node is to the left of the root) or right (if the node is on the right of the root). If step 2b is applied in this node then the resource numbers are merely shifted and hence, by the induction assumption, all son nodes preserve the consecutive-labels property.

Notice that the fact (recall Lemma A.1) that the synchronization tree does not have nodes with shared resources in the same level, is used in step 2b.

Finally, permuting the rows of A according to the labeling we created, each column in the graph (corresponding to each activity) will have consecutive ones. Recall that this guarantees that the matrix A is totally unimodular which concludes the proof for nested architectures.

Weakly non-nested architectures: We will show that if an architecture is weakly non-nested, we can alter the network in a way that preserves the value of ρ^{BN} and can only increase ρ^{net} but has a nested collaboration architecture. This will imply, by the first part of this theorem, that there is no unavoidable idleness.

We start with an example. Consider a network of 3 activities and 4 resources as in the collaboration graph in Figure 13(LHS)–each circle corresponds to an activity and the required resources are listed below the activity's label.

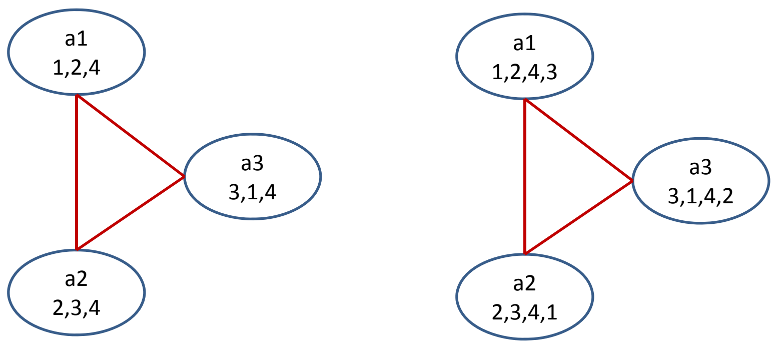

Figure 13 Transforming a weakly non-nested network into a nested one without changing ρ^{BN}.

This network contains a non-nested cycle. It is weakly non-nested because resource 4 is shared by all activities in the cycle. The bottleneck in this network is trivially resource 4 with $\rho_4 = \rho^{\mathrm{BN}} = \sum_{j=1}^{3} \lambda_j m_j$ (and $\rho_2, \rho_1, \rho_3 \leq \rho_4$). If we add each of the resources 1,2 and 3 (each, notice, is assigned initially to 2 activities) to activities to which they are not assigned, we obtain the network with the collaboration graph on Figure 13(RHS). The resulting network is trivially nested. Importantly, this action does not affect the theoretical utilization which remains $\rho^{\mathrm{BN}} = \rho_i = \sum_{j=1}^{3} \lambda_j m_j,\ i = 1, \ldots, 4$ and it can only increase ρ^{net} as, by assigning more resources to activities we can only shrink the family of feasible configuration vectors.

The new network we constructed is nested so that $\rho^{\mathrm{BN}} = \rho^{\mathrm{net}}$. In particular, we can construct an allocation π that achieves $\rho^{\mathrm{net}} = \rho^{\mathrm{BN}}$. In this special example, positive weights are given only to the identity vectors $e^i,\ i = 1, 2, 3$.

To generalize this argument, fix a network with a weakly non-nested collaboration architecture. As in the above example, we first transform all weakly non-nested cycles into nested ones.

Fix a weakly non-nested cycle of activities $\mathcal{C} \subseteq \mathcal{I}$. Let k^* be the focal resource of this cycle: the resource that is shared by all activities in the cycle. If there are multiple cycles in which k^* is the focal resources, $\mathcal{C}$ is taken to be the one with the most activities. We can also assume there is a single such resource for $\mathcal{C}$. If there are two we can treat them, without loss of generality as the same resource. Let $\mathcal{K}(\mathcal{C}) = \{k \in \mathcal{K} : A_{kj} = 1, \text{ for some } j \in \mathcal{C}\}$ be the set of resources that participate in at least one activity in the cycle.

By definition $j \in \mathcal{C}$ if and only if $A_{k^*j} = 1$. We distinguish between two types of resources associated with this cycle:

(i) Resources that participate in two activities or more in the cycle. The set of these is given by $\mathcal{K}_{\geq 2}(\mathcal{C}) := \{k \in \mathcal{K}(\mathcal{C}) : \sum_{j \in \mathcal{C}} A_{kj} \geq 2\}$.

We claim that a resource $k \in \mathcal{K}_{\geq 2}(\mathcal{C})$ cannot have activities $j \notin \mathcal{C}$. Indeed, suppose that there exist $k \in \mathcal{K}_{\geq 2}(\mathcal{C})$ and $j_0 \in \mathcal{I}$ such that $A_{kj_0} = 1$ but $A_{k^*j_0} = 0$. Since k participates in two activities in the cycle $\mathcal{C}$, there must exist two activities $j_1, j_2 \in \mathcal{C}$ such that j_0, j_1 and j_0, j_2 are in the graph. Note that because all activities in $\mathcal{C}$ share a resource we can assume without loss of generality that the activities j_1, j_2 are consecutive activities in the cycle (otherwise we can re-label the activities). Thus, we have identified a non-nested cycle $\bar{\mathcal{C}}$ (with more activities than $\mathcal{C}$). Note that $\bar{\mathcal{C}}$ must be a non-nested cycle. If it were nested than the smaller cycle $\mathcal{C}$ would also be nested. Moreover, it is strongly non-nested because $A_{k^*j_0}=0$. This would be a contradiction to the assumption that all non-nested cycles are weakly non-nested.

(ii) Resources that participate in one activity in the cycle $\mathcal{K}_1(\mathcal{C})$.

We do nothing for resources $k\in\mathcal{K}(\mathcal{C})\setminus\mathcal{K}_{>2}(\mathcal{C})$. Since we have argued that for $k\in\mathcal{K}_{>2}(\mathcal{C})$, $\sum_{j\notin\mathcal{C}}A_{kj}=0$, we can alter the network by assigning k to each of the activities $j\in\mathcal{C}$ with (initially) $A_{kj}=0$ and still have $\sum_j A_{kj}\lambda_j m_j\le\sum_j A_{k_{\mathcal{C}}j}\lambda_j m_j$ so that the value of $\rho^{\text{BN}}=\max_k\rho_k$ does not change. Note that the resulting cycle is nested. Any resource that appears twice appears now in all activities of $\mathcal{C}$ so that, in any order, the condition (15) holds. Repeating the same for each such weakly non-nested cycle, the network is transformed into a nested network. For this network $\rho^{\text{net}}=\rho^{\text{BN}}$. Since, by assigning resources to more activities we only shrink the family of configuration vectors this, in particular, implies for the original network that $\rho^{\text{BN}}=\rho^{\text{net}}$ which concludes the proof. ■

Proof of Lemma 4.2: Let $\mathcal{C}=i_0,\dots i_l$ be the shortest amongst the strongly non-nested sharing cycles. A segment of the cycle is a subset of consecutive activities in the cycle. Since $\mathcal{C}$ is strongly non-nested it can be divided into non overlapping segments (the end point of one segment can serve as a starting point for the next) such that for each segment there is a resource k that is shared by all activities in this segment. There must be at least two such segments since the cycle is, by assumption, strongly non-nested.

Note that there cannot be another strongly non-nested cycle in the graph that has nodes in two distinct segments of the cycle $\mathcal{C}$. Otherwise $\mathcal{C}$ would not be the shortest strongly non-nested cycle. Also, there can not be a nested-sharing cycle (or a weakly non-nested cycle) with nodes in two distinct segment because by definition both nested and weakly non-nested cycles require the existence of a resource that is shared by all activities in the cycle.

We conclude that there are no edges in the collaboration graph with end points in distinct segment of this cycle. We can then assume, without loss of generality, that each segment has one edge (and two activities). Indeed, if there are three activities there will be an edge between each two of them because they share a resource and we can drop one activity. Thus, we have found a simple cycle. ■

Proof of Theorem 4.5: By Lemma 4.2 the network contains at least one simple non-nested cycle. Let M be the (odd) number of nodes in the cycle (it is also the number of edges).

Choose λ and m such that $\lambda_j m_j=1/2$ for each activity on the cycle. Set $\lambda_j m_j=0$ for all other activities in the network. Recall that a cycle $i_1,\dots,i_l$ is simple non-nested if each two activities connected by an edge share a resource that is not used in any other activity in the cycle. With the above parameters we can assume that there is one such resource per edge (if there are multiple we can treat them as the same resources) and a total of M resources assigned to activities in the cycle.

Each resource that defines an edge on the cycle has two activities with a total load of 1 and is thus a bottleneck. Since at most $\lfloor(M-1)/2\rfloor<M/2$ of the M activities on this cycle can be processed in parallel

and each activity uses 2 of these resources there is no feasible configuration set $\mathcal{A}$ with $\mathbf{BN} \subseteq \mathcal{R}(\mathcal{A})$. The condition of Lemma 4.3 is trivially satisfied and we can conclude that the network features unavoidable idleness. ■

Proof of Lemma 4.3: Suppose that $\rho^{\text{net}} = \rho^{\text{BN}} = 1$. Let (π, ρ^{net}) be a solution to the SPPC (i.e, $\sum_{\mathcal{A}\in\mathbb{C}} a(\mathcal{A})\pi(\mathcal{A}) = \lambda m.$ and $\sum_{\mathcal{A}\in\mathbb{C}} \pi(\mathcal{A}) = \rho^{\text{net}}$).

Since $\sum_i A_{ki}a_i(\mathcal{A}) \in \{1, 0\}$ for any feasible configuration set $\mathcal{A}$, we have that

$$\begin{aligned}\rho_k &= \sum_i A_{ki}(\lambda_i m_i) = \sum_i A_{ki}(\sum_{\mathcal{A}\in\mathbb{C}} a(\mathcal{A})\pi(\mathcal{A}))_i \\ &= \sum_{\mathcal{A}\in\mathbb{C}} \pi(\mathcal{A}) \sum_i A_{ki}a_i(\mathcal{A}) = \sum_{\mathcal{A}\in\mathbb{C}, k\in\mathcal{R}(\mathcal{A})} \pi(\mathcal{A}).\end{aligned} \tag{26}$$

Moreover, if $\mathcal{A}$ is such that $\pi(\mathcal{A}) > 0$ then it must be the case that $\mathbf{BN} \subseteq \mathcal{R}(\mathcal{A})$. Indeed, for all $k \in \mathbf{BN}$, the right hand side of (26) is $\rho^{\text{net}} = \rho^{\text{BN}} = 1$. Thus, if there exist $k, l \in \mathbf{BN}$ and $\mathcal{A}$ with $\pi(\mathcal{A}) > 0$ such that $k \in \mathcal{R}(\mathcal{A})$ but $l \notin \mathcal{R}(\mathcal{A})$ then we would have $\sum_{\mathcal{A}\in\mathbb{C}: l\in\mathcal{R}(\mathcal{A})} \pi(\mathcal{A}) < 1 = \rho^{\text{BN}}$.

In turn, if $\rho^{\text{net}} = \rho^{\text{BN}} = 1$ there exists a family $\mathbb{C}(\mathbf{BN}) \subseteq \mathbb{C}$ such that $\mathbf{BN} \subseteq \mathcal{R}(\mathcal{A})$ for each $\mathcal{A} \in \mathbb{C}(\mathbf{BN})$ and $\sum_{\mathcal{A}\in\mathbb{C}(\mathbf{BN})} \pi(\mathcal{A}) = 1$.

Finally, for each i, $(\sum_{\mathcal{A}\in\mathbb{C}} a(\mathcal{A})\pi(\mathcal{A}))_i = \lambda_i m_i$ (recalling that $a(\mathcal{A})$ is a binary vector) so that $\pi(\mathcal{A}) \leq \min_{i\in\mathcal{A}} \lambda_i m_i$. We conclude that, if $\rho^{\text{BN}} = \rho^{\text{net}}$ there must exist a family of subsets $\mathbb{C}(\mathbf{BN})$ of $\mathbb{C}$ such that

$$1 = \sum_{\mathcal{A}\in\mathbb{C}(\mathbf{BN})} \pi(\mathcal{A}) \leq \sum_{\mathcal{A}\in\mathbb{C}(\mathbf{BN})} \min_{i\in\mathcal{A}} \lambda_i m_i.$$

In particular, if

$$\sum_{\mathcal{A}\in\mathbb{C}:\mathbf{BN}\subseteq\mathcal{R}(\mathcal{A})} \min_{i\in\mathcal{A}} \lambda_i m_i < 1,$$

it must be the case that $\rho^{\text{net}} > \rho^{\text{BN}} = 1$.

For the second part of the lemma, arguing as before we obtain that

$$\rho_k = \sum_{\mathcal{A}:k\in\mathcal{R}(\mathcal{A})} \pi(\mathcal{A}) \leq \rho^{\text{net}} \left(\sum_{\mathcal{A}:k\in\mathcal{R}(\mathcal{A})} \min_{i\in\mathcal{A}} \lambda_i m_i \right).$$

(Recall that $\rho^{\text{net}} \geq 1$ in the assumptions of the lemma.) Then, for each $k \in \mathbf{BN}$

$$\rho^{\text{BN}} = \rho_k \leq \rho^{\text{net}} \max_{l\in\mathbf{BN}} \left(\sum_{\mathcal{A}:l\in\mathcal{R}(\mathcal{A})} \min_{i\in\mathcal{A}} \lambda_i m_i \right),$$

which completes the argument. ■

Proof of Lemma 5.4: The proof is by construction. Let ρ^{BN} be the solution of the SPP. Let $c^{0,\theta}$ be the configuration vector that has as its i^{th} entry

$$c_i^{0,\theta} = \frac{1}{\rho^{\text{BN}}} \left\lfloor \lambda_i^\theta m_i \right\rfloor.$$

(note that since we assume throughout the paper that $\lambda_i m_i > 0$ for at least one i, we have that $\rho^{\text{BN}} > 0$.) Let $\pi(c_i^{0,\theta}) = \rho^{\text{BN}}$. Also, let $\mathfrak{n}^{i,\theta}$ be the vector that has $\min_{k:k\in\mathcal{R}(\{i\})} n_k^\theta$ in its i^{th} entry and 0 otherwise. Set

$$\pi(\mathfrak{n}^{i,\theta}) = \frac{1}{\min_{k:k\in\mathcal{R}(\{i\})} n_k^\theta} \left(\lambda_i^\theta m_i - \left\lfloor \lambda_i^\theta m_i \right\rfloor \right).$$

Note that all vectors c^0 and $(\mathfrak{n}^i,\ i\in\mathcal{I})$ are feasible configuration vectors since they satisfy $\sum_i A_{ki}c_i^{0,\theta}\le n_k^\theta$ for all k and $\sum_l A_{kl}\mathfrak{n}_l^{j,\theta}=\min_{l:l\in\mathcal{R}(\{j\})} n_l\le n_k^\theta$ for all k. Finally, note that

$$\pi(c^{0,\theta})c^{0,\theta}+\pi(\mathfrak{n}^{i,\theta})\mathfrak{n}^{i,\theta}=\lambda_i^\theta m_i$$

and

$$\pi(c^{0,\theta})+\sum_i\pi(\mathfrak{n}^{i,\theta})=\rho^{\mathrm{BN}}+\sum_i\frac{1}{\min_{k:k\in\mathcal{R}(\{i\})}n_k^\theta}\left(\lambda_i^\theta m_i-\lfloor\lambda_i^\theta m_i\rfloor\right)=:\rho^\theta.$$

Thus, the θ-s SPPC has a feasible solution (π^θ,ρ^θ) with

$$|\rho^{\mathrm{BN}}-\rho^\theta|\le\sum_i\frac{1}{\min_{k:k\in\mathcal{R}(\{i\})}n_k^\theta}$$

Finally, since λ is strictly positive by assumption and each resource is assigned to at least one activity we must have that $n_k>0$ to have a feasible solution for the SPP. Since $n_k^\theta=\theta n_k\to\infty$ as $\theta\to\infty$, we conclude that $|\rho^{\mathrm{BN}}-\rho^\theta|\to 0,$ as $\theta\to\infty$. ■

Proof of Theorem 6.7: The proof is straightforward given the definitions and Theorem 4.4 for the no-flexibility case. Specifically, given $(x^{\mathrm{BN}},\rho^{\mathrm{BN}})$ as in the statement of theorem (i.e, that solve (23)), consider the following problem

$$\begin{array}{rl}\text{minimize} & \rho\\ \text{s.t.} & \sum_i A_{k,(iG)}x_{iG}^{BN}\le\rho,\text{for all } k\in\mathcal{K},\end{array}$$

This can be interpreted as the SPP corresponding to an artificial network with activities $\{(iG)\}$ and with arrival rate x_{iG}^{BN} to activity (iG). Trivially, this problem has ρ^{BN} as its optimal solution. The collaboration architecture of this artificial network is nested by assumption. By Theorem 4.4, there exists $\pi\ge 0$ such that $\sum_{\mathcal{A}\in\mathbb{C}}\pi(\mathcal{A})=\rho^{\mathrm{BN}}$ and $\sum_{\mathcal{A}\in\mathbb{C}}a(\mathcal{A})\pi(\mathcal{A})=x^{\mathrm{BN}}$. Thus, $\sum_G\sum_{\mathcal{A}\in\mathbb{C}}(a(\mathcal{A})\pi(\mathcal{A}))_{iG}=\sum_G x_{iG}^{\mathrm{BN}}=\lambda_i m_i$ where the last equality follows directly from the SPP. ■

The following provides a weaker sufficient condition than the one in Theorem 6.7. We let $\rho(x)=\max_k\sum_k A_{k,(iG)}x_{iG}$.

Lemma A.2 *Fix λ and let (x^{BN},ρ^{BN}) be an optimal solution to the SPP with $\rho^{BN}(\lambda)\le 1$. Suppose that x^{BN} can be written as a sum of non-negative vectors $x^1,\dots x^\ell$ each of which induces a nested extended collaboration architecture and such that $\sum_{l=1}^{\ell}\rho(x^l)\le 1$. Then, $\rho^{net}(\lambda)\le 1$.*

Proof: For each m we can construct as in the proof of Theorem 6.7 a probability vector such that $\sum_a\pi(a)=\rho^m$ where ρ^m is the value of the static planning problem for x^m. A probability vector $\breve{\pi}$ is then constructed by setting $\breve{\pi}(a)=\sum_m\pi^m(a)$. by assumption $\sum_a\breve{\pi}(a)\le 1$ and $\sum_a a\breve{\pi}(a)=x^{\mathrm{BN}}$. ■

Proof of Lemma 6.6: Identically to Lemma 4.3 it is proved that if the SPP has an optimal solution $\rho^{\mathrm{BN}} = 1$ and

$$\sum_{\mathcal{A}\in\mathbb{C}:\mathbf{BN}\subseteq\mathcal{R}(\mathcal{A})} \min_{(i,G)\in\mathcal{A}} \lambda_i m_i < 1,$$

then the network features unavoidable bottleneck idleness. Let $\mathbb{C}^F$ be the family of feasible configuration sets after the addition of the extended activity (i,G) as in the statement of the lemma. Under the conditions of the Lemma 6.6 the extended activity (i,G) cannot participate in any covering of $\mathbf{BN}^F = \mathbf{BN}\bigcup\{k\}$. In particular, $\{\mathcal{A}\in\mathbb{C}^F : \mathbf{BN}^F \subseteq \mathcal{R}(\mathcal{A})\} \subseteq \{\mathcal{A}\in\mathbb{C} : \mathbf{BN} \subseteq \mathcal{R}(\mathcal{A})\}$ so that under the condition of the lemma

$$\sum_{\mathcal{A}\in\mathbb{C}^F:\mathbf{BN}^F\subseteq\mathcal{R}(\mathcal{A})} \min_{(i,G)\in\mathcal{A}} \lambda_i^F m_i \leq \sum_{\mathcal{A}\in\mathbb{C}:\mathbf{BN}\subseteq\mathcal{R}(\mathcal{A})} \min_{(i,G)\in\mathcal{A}} \lambda_i^F m_i < 1$$

and by the first part of the lemma the network features unavoidable idleness. ■