Valid Inequalities Based on Demand Propagation for Chemical Production Scheduling MIP Models

Sara Velez, Arul Sundaramoorthy, And Christos Maravelias[1]

Department of Chemical and Biological Engineering
University of Wisconsin – Madison
1415 Engineering Dr., Madison, WI, 53706

Abstract. The planning of chemical production often involves the optimization of the *size* of the tasks to be performed subject to *unit* capacity constraints, as well as inventory constraints for intermediate materials. While several mixed-integer programming (MIP) models have been proposed that account for these features, the development of tightening methods for these formulations has received limited attention. In this paper, we develop a constraint propagation algorithm for the calculation of lower bounds on the number and size of tasks necessary to satisfy given demand. These bounds are then used to express three types of tightening constraints which greatly enhance the computational performance of the MIP scheduling model. Importantly, the proposed methods are applicable to a wide range of problem classes and time-indexed MIP models for chemical production scheduling.

1. Introduction

Chemical production involves the conversion of raw materials (feedstocks) into final products, both of which are most often fluids (gases or liquids). Fluids can be mixed or split in variable proportions and are often recycled (e.g., unreacted raw materials are recycled after they are separated from the final product). Production tasks convert a set of *input* materials into a set of *output* materials, and are carried out in *equipment units* that are capable of processing different amounts of (fluid) materials. In other words, the *size* and therefore the number of tasks to be executed can be an optimization decision.

While there are chemical facilities with a wide range of processing characteristics and constraints, there are two major types of chemical processing, *sequential* and *network*. In *sequential* processing, it is assumed that production tasks have a single input and a single output material; the input material of a task is produced by a single upstream task; and the output of a task is consumed by a single downstream task. Thus, materials from different tasks are not mixed, and the output of a task cannot be split to be consumed by multiple tasks. Also, it is implicitly assumed that each material is produced and consumed by at most one task. In *network* processing, tasks may consume or produce multiple materials, and a material can be produced and consumed by different types of tasks. Also, material coming from different tasks can be mixed in a storage vessel, the output of a task can be consumed by multiple downstream tasks, and material can also be recycled.

[1] Corresponding author; email: maravelias@wisc.edu; Tel: +1-608-265-9026

Interestingly, the methods that have been developed to address chemical production scheduling problems follow the aforementioned classification. Early work focused on sequential facilities where a *batch* of raw material has to go through multiple production stages to be converted to the final product. In other words, the processing of fluids is represented as the processing of a *discrete* batch that has to go through different processing stages; amounts of materials are not monitored and no material balances are included. We note that this type of modeling is possible because splitting, mixing and recycling are not allowed. We term these approaches as *batch-based*. In most cases, it is also assumed that the sizes and thus the number of batches (for known demand) were determined prior to optimization [Reklaitis, 2012], while recent approaches consider simultaneous batching and scheduling [Prasad and Maravelias, 2008]. In the process systems engineering (PSE) literature these facilities are termed either as *multi-stage*, if the routing through the stages is the same for all batches/products, or *multi-purpose*, if there are batch/product-specific routings [Mendez et al., 2006; Maravelias, 2012a].

Clearly, batch-based methods cannot be used to address problems in network facilities where mixing, splitting, and recycling are allowed; materials can be produced and consumed by different tasks; and the sizes of the tasks are optimization decisions. To address these problems researchers developed what we term *material-based* models, where the amounts of material processed by a task, as well as the inventories of materials, are explicitly modeled. Another key modeling aspect of these approaches is that a time reference grid is defined, and (i) the start/end times of tasks are mapped onto this grid and (ii) inventory constraints are expressed at the time points of the grid. The pioneering work of Pantelides and co-workers employed a discrete-time grid, where the time horizon is divided into periods of equal and known length and processing times are assumed to be constant [Kondili et al., 1993; Shah et al., 1993]. Since then, a wide range of formulations have been proposed including continuous-time and mixed-time grids, as well as unit-specific grids.

Regardless of the nature of the time representation and the details of the formulation, there are three processing constraints that have to be enforced [Maravelias, 2012b]:

a) ***Unit utilization***: a unit cannot perform more than one task at a time.
b) ***Material balance***: the inventory of a material in a storage vessel has to be nonnegative and less than the capacity of the vessel.
c) ***Batch-size***: the size of a processing task (batch-size) has to be between a nonzero lower and an upper bound (capacity).

The main difference between the various material-based formulations lies in the modeling of the first constraint. In discrete-time models, it is enforced through the widely used one-machine clique constraint, while in continuous-time formulations more complex sets of constraints are used. The second constraint is always a flow balance constraint for a material-time node, similar to those used in production planning formulations [Pochet and Wolsey, 2006]. Finally, the third constraint results in variable lower/upper bound constraints: if a task is executed, then the amount processed should be within a lower and upper bound; otherwise it is zero. The size of a task should be greater than or equal to the lower bound for safety and controllability reasons, as well as due to recipe constraints. Most of the aforementioned types of constraints have been studied extensively. First, valid inequalities and specialized solution methods have been developed for time-indexed single-machine problems [Wolsey, 1990; van den Akker et al., 1999; van den Akker et al., 2000]. Also, many constraint propagation algorithms have been developed for this constraint [Hooker, 2000; Baptiste et al., 2001; Van Hentenryck and Michel, 2005; Hooker, 2007]. Second, lot sizing problems have material balances that are similar to those in network scheduling problems, and adding valid constraints based on these material balances and bill-of-materials information results in a significantly tighter formulation [Pochet and Wolsey, 1993; Wolsey, 1997; Miller and Wolsey, 2003].Finally, valid inequalities have been derived from variable upper bound constraints in the context of fixed-charge network problems [Nemhauser and Wolsey, 1988; Wolsey, 1998]. However, scheduling problems with variable lower bound constraints have not been studied sufficiently. Burkard and Hatzl (2005) bound the number of batches of each task and the number of batches produced of each material by solving a small MIP for every task and an LP for every material, while Janak and Floudas (2008) propose bounding the number of batches and cumulative production for each task by solving an MIP bounding problem for every task.

Accordingly, in this paper, we study how the variable lower bound constraints can be used to develop strong valid inequalities for MIP chemical production scheduling problems. Specifically, we first develop an algorithm that allows us to calculate tight lower bounds on the number of batches of each task and the total amount of material processed by each task that are necessary to satisfy the given demand. These lower bounds, which are calculated from instance data within seconds, are then used to write two classes of valid inequalities which greatly reduce the computational requirements. Our algorithm is applicable to all types of network facilities, including facilities with (i) multiple tasks producing or consuming the same material(s), (ii) multiple units capable of performing the same task(s), and (iii) recycle loops. Also, they are applicable to all types of material-based MIP formulation that employ time-indexed variables.

It is important to note that material-based time-indexed MIP models are used as the backbone in a wide range of chemical production scheduling tools, as well as in in-house tools developed by chemical companies [Wassick, 2009]. Also, material-based models have been extended to address problems in sequential production environments as well as facilities that combine sequential and network environments [Sundaramoorthy and Maravelias, 2011]. Nevertheless, one of the limitations of both commercial products and specialized tools is that they remain computationally inefficient. Thus, our methods, which are applicable to all aforementioned models, have the potential to benefit a wide range of tools and models.

The paper is structured as follows. In Section 2, we introduce a network-inspired representation of chemical facilities, present a formal statement of the problem we consider and a widely used MIP scheduling model, and close with a motivating example. In Section 3, we present the constraint propagation algorithm for the calculation of the lower bounds on the number of batches and the total material processed. In Section 4, we present two types of tightening constraints, as well as extensions for problems with intermediate due times. In Section 5, we perform an extensive computational study including more than 70 problem instances. We use lowercase italic letters for indices, uppercase bold letters for sets, lowercase Greek letters for parameters, and uppercase italics for optimization variables.

2. Background

2.1. Chemical Production Scheduling Notation

A chemical facility transforms a set of raw materials into a set of valuable products through a sequence of processing tasks carried out in equipment units. A processing task converts a set of input materials (raw materials or intermediates) into a set of outputs (intermediates or final products) according to specified conversion coefficients. Note that the term *task* is used to describe a type of operation (e.g., the conversion of A to B via reaction A → B is a task), which implies that a schedule may include multiple *executions* of the same *task*. A task can in general be carried out in multiple units of unequal capacity, which implies that the number of batches of a task necessary to satisfy given demand is not known prior to optimization. Thus, the term scheduling in the PSE literature has been used to describe a problem which includes three levels of decisions: (i) the determination of the number (and size) of batches to be performed, (ii) the assignment of batches to processing units, and (iii) the timing of batches so that at most one batch is executed on a unit.

We assume that a chemical facility consists of: (i) processing units, where tasks are executed; (ii) storage vessels, where materials are stored; and (iii) piping for material transfer. Processing units and storage vessels have capacity limitations, and each processing unit and storage vessel is capable of processing/storing a subset of tasks/materials. We do not consider other shared resources such as utilities (e.g., electricity) and labor. Figure 1a shows the *process flow diagram* (PFD) of a simple chemical facility, where raw material S1 is converted into intermediate S2 in unit U1, and the latter is converted to either product S3 or S4 in unit U2 or U3. Note that the PFD shows the physical equipment of the facility, but not necessarily the tasks that take place on the units. Also, units can perform multiple tasks; e.g., units U2 and U3 can convert S2 to S3 or convert S2 to S4.

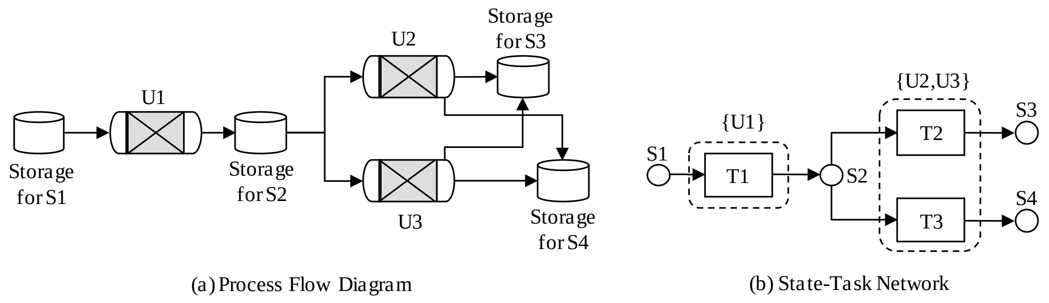


Figure 1. Process flow diagram (physical layout of the facility) and state-task network representation of a facility.

2.2. Problem Statement

To study scheduling problems, we have to use a representation that includes not only the physical resources, but also the different tasks that can be carried out. In this paper, we adopt the state-task network representation, which is based on the following concepts:

a) States (materials): include feeds, intermediates, and final products and are represented by circles.
b) Tasks: are operations that produce and consume states (materials) and are depicted with rectangles.
c) Units: unary resources for the execution of tasks; multiple units may be able to process a single task, and a single unit may be able to process multiple tasks.

States (circles) and tasks (rectangles) are connected by arrows, referred to as *streams*, showing the flow of material. If a task consumes (produces) a state, an arrow points from the state (task) to the task (state). Figure 1b shows the STN representation for a scheduling problem in the facility shown in Figure 1a. Note that units are shown implicitly through the task-unit compatibility information given by the dotted lines. The structure of the network is defined in terms of the following sets (see Figure 2a):

Indices/Sets

$i,i'\in\mathbf{I}$	tasks
$j\in\mathbf{J}$	processing units
$s,s'\in\mathbf{S}$	states

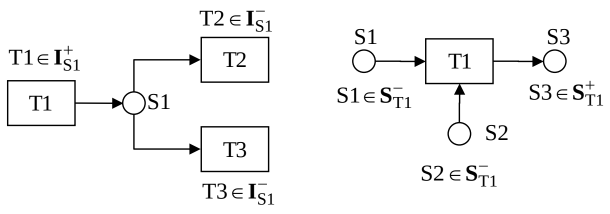
Subsets

$\mathbf{I}_s^+$ / $\mathbf{I}_s^-$	tasks producing/consuming state *s*
$\mathbf{J}_i^P$	processing units that can process task *i*
$\mathbf{S}_i^+/\mathbf{S}_i^-$	states produced/consumed by task *i*
$\mathbf{S}^F$	final products

We are also given the following parameters:

ϕ_s	total customer demand for state *s*
$\beta_j^{\min}/\beta_j^{\max}$	minimum/maximum capacity of unit *j*
ρ_{is}^+/ρ_{is}^-	fraction of state *s* produced/consumed by task *i*
τ_{ij}	fixed processing time for task *i* in unit *j*
ζ_s^0	initial inventory of state *s*

In Figure 2a, S1 is consumed by T2 and T3 and produced by T1; therefore, $\mathbf{I}_{\text{S1}}^-$={T2, T3} and $\mathbf{I}_{\text{S1}}^+$={T1}. In Figure 2b, T1 produces S3 and consumes S2 and S1, so $\mathbf{S}_{\text{T1}}^+$={S3} and $\mathbf{S}_{\text{T1}}^-$={S1,S2}. In general, a plus (+) superscript indicates production, and a minus (–) indicates consumption. If T2 and T3 can both be processed in either U2 or U3, then $\mathbf{J}_{\text{T2}}^P=\mathbf{J}_{\text{T3}}^P$={U1, U3}.


$\text{T1}\in\mathbf{I}_{\text{S1}}^+$
T1
S1
$\text{T2}\in\mathbf{I}_{\text{S1}}^-$
T2
T3
$\text{T3}\in\mathbf{I}_{\text{S1}}^-$
S1
T1
S3
$\text{S1}\in\mathbf{S}_{\text{T1}}^-$
$\text{S3}\in\mathbf{S}_{\text{T1}}^+$
S2
$\text{S2}\in\mathbf{S}_{\text{T1}}^-$

(a) Sets $\mathbf{I}_s^+/\mathbf{I}_s^-$ (b) Sets $\mathbf{S}_i^+/\mathbf{S}_i^-$

Figure 2. Illustration of general sets.

2.3. Discrete-time Model

To illustrate the underlying concepts and perform computational studies, we introduce the widely-used MIP formulation of Shah et al. (1993). We use this model because a recent computational study showed that discrete-time models are more effective for large-scale problems and, most importantly, can be easily extended to account for a range of processing restrictions and characteristics (e.g., intermediate due dates, holding and utility costs, variable resource requirements during task execution) at almost no computational cost [Sundaramoorthy and Maravelias, 2011b]. However, we highlight that our methods are applicable to all time-indexed MIP models for chemical production scheduling that account for variable batch-sizes.

Time points are indexed by *t*. The formulation includes one family of binary variables:

X_{ijt}^{P}	is one if unit *j* processes task *i* starting at time *t*

and the following non-negative continuous variables:

B_{ijt}	batch-size of task *i* processed in unit *j* starting at time *t*
S_{st}	inventory of state *s* at time *t*
P_{st}	amount of feed state *s* purchased at time *t*
D_{st}	amount of final product *s* delivered to customers at time *t*

The model consists of unit utilization (eqn. 1), unit capacity (eqn. 2), inventory capacity (eqn. 3), and material balance (eqn. 4) constraints.

$$\sum_{i \in \mathbf{I}_j} \sum_{t' \geq t-\tau_{ij}+1}^{t} X_{ijt'}^{P} \leq 1 \quad \forall j,t \tag{1}$$

$$\beta_j^{\min} X_{ijt}^{P} \leq B_{ijt} \leq \beta_j^{\max} X_{ijt}^{P} \quad \forall i, j \in \mathbf{J}_i^{P}, t \tag{2}$$

$$S_{st} \leq \gamma_s^{\max} \quad \forall s,t \tag{3}$$

$$S_{st} = S_{s(t-1)} + \sum_{i \in \mathbf{I}_s^{+}} \rho_{is}^{+} \sum_{j \in \mathbf{J}_i^{P}} B_{ij(t-\tau_{ij})} - \sum_{i \in \mathbf{I}_s^{-}} \rho_{is}^{-} \sum_{j \in \mathbf{J}_i^{P}} B_{ijt} + P_{st} - D_{st} \tag{4}$$

where D_{st} is fixed to the total amount of state *s* due at time *t*; and P_{st} is only non-zero for feed states. Variables X_{ijt}^{P} are fixed to zero for the τ_{ij} -1 time points before the end of the time horizon.

2.4. Motivating Example

We consider an instance of the facility introduced in Figure 1, with a demand for 90kg of S3 and 25kg of S4. For each unit, the capacity (min-max) and processing cost of a task in that unit are: 25-60kg and \$10 for U1; 40-50kg and \$25 for U2; and 35-45kg and \$30 for U3. Our objective is to minimize cost. We generated a model for this instance based on formulation presented in §2.3. The minimum cost for the LP-relaxation is \$76.7, while the number of batches of tasks T1, T2, and T3 are 1.9, 1.8, and 0.5, respectively (see Table 1).

Table 1. Effect of tightening on the number of batches and cost for the network in Figure 1.

	# of Batches			Min.
	T1	T2	T3	Cost (\$)
Optimal solution	3	2	1	105
Minimum # of batches				
Burkard's method	2	2	1	
Proposed method	3	2	1	
LP-Relaxation				
No tightening	1.9	1.8	0.5	76.7
Burkard's method	2.2	2	1	96.7
Proposed method	3	2	1	105

However, if we take into account the capacities of the units, we can infer that more batches will be required in any feasible solution. T2 must produce 90kg of S3 to satisfy demand; since the maximum batch-size is 50kg, this requires at least two batches. Similarly, T3 must produce at least 25kg of S4, but, since the minimum batch-size is 35kg, T3 must produce at least 35kg in at least one batch. Together, T2 and T3 require a minimum of 125kg (=90+35) of S2, which T1 produces in at least three batches. Using this *constraint propagation* procedure, we calculate lower bounds on the number of batches and cumulative production for each task.

Table 1 shows the effect of tightening on the example in Figure 1. Without tightening, the minimum cost for the LP-relaxation is \$76.7. With the proposed method, the cost of the LP relaxation is \$105. Interestingly, the minimum cost and number of batches of the LP-relaxation in this simple problem, match the optimal MIP solution. Our goal is to develop a method that allows us to systematically calculate similar bounds in general production networks, including facilities with recycle streams. Note that using the tightening methods proposed by Burkard and Hatzl (2005) improves the cost of the LP-relaxation to \$96.7. Burkard and Hatzl (2005) do not consider minimum unit capacities, so their method requires T1 to produce only 115kg (=90+25) of S2 in two batches. The tightening methods of Janak and Floudas (2008) find a lower bound on the number of batches which is as tight as ours, but require solving a MIP for every task which is often as time consuming as solving the original MIP model. Also, their approach relies on a specific continuous-time models. Our proposed methods do not require solving any auxiliary MIPs, are valid for any network, and can be used with any model employing a time grid.

3. Constraint Propagation Algorithm

Our constraint propagation algorithm calculates parameters which are then used to formulate the tightening constraints. The algorithm depends on the structure of the process network. We classify networks into four categories according to Figure 3:

(1) networks with no loops (Figure 4a);

(2) networks with loops but no recycle states (Figure 4b);

(3) networks with recycle states in a single loop (Figure 4c); and

(4) networks with a recycle state and multiple nested loops (Figure 4d).

A recycle loop is any closed path (moving only in the direction of the stream arrows) within the network, and a recycle state is a state in a loop that can be produced by multiple tasks. Each colored box in Figure 3 has its own specific tightening procedures that are described in the indicated section. All networks use the tightening procedures for general networks (§3.1). Networks are first divided into networks with and without recycle loops. If there are recycle loops, §3.2 describes the additional tightening methods. Recycle loops are further divided into loops with and without recycle states, and §3.3 explains more tightening methods for networks with recycle states. When a loop has a recycle state, it is classified as having either a single loop or a nested loop, which is described in §3.4. In §3.5, we present a general algorithm for applying the tightening methods to any network. In §3.6, we describe an alternative approach for networks with recycle states that also works well when multiple tasks can produce a single state.

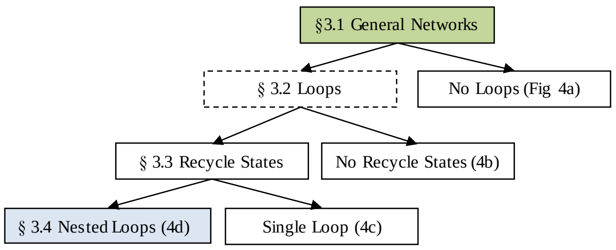
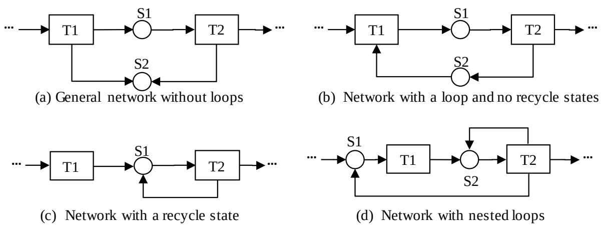


Figure 3. Classification of networks.



(a) General network without loops
(b) Network with a loop and no recycle states
(c) Network with a recycle state
(d) Network with nested loops

Figure 4. Examples of categories of networks.

3.1. General Networks

3.1.1. Backward Propagation

For backward propagation, we introduce the following parameters:

ω_s	lower bound on the amount of state *s* required to meet final demand
$\mu_i/\tilde{\mu}_i$	lower bound on the production of task *i* required to meet final demand, $\mu_i \le \tilde{\mu}_i$
v_{is}	lower bound on the amount of state *s* that must be produced by task *i*

First, we need to find the minimum amount required for all states, ω_s, and the minimum production for all tasks, $\tilde{\mu}_i$; we calculate these parameters sequentially by backward propagating demand for final products. We estimate ω_s once $\tilde{\mu}_i$ is known for all tasks consuming state *s* (all $i \in \mathbf{I}_s^-$). Similarly, we need to know ω_s for all states produced by task *i* (all $s \in \mathbf{S}_i^+$) before we calculate $\tilde{\mu}_i$ (see Figure 5).

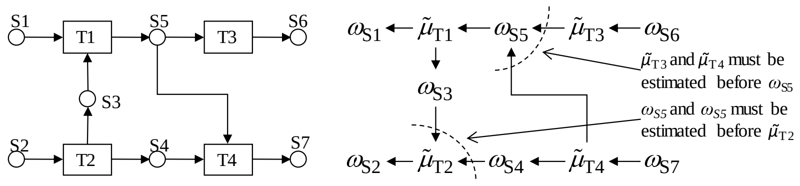
We calculate ω_s starting with the final products:

$$\omega_s = \begin{cases} \phi_s - \zeta_s^0 & s \in \mathbf{S}^F \\ \sum_{i \in \mathbf{I}_s^-} \rho_{is}^- \tilde{\mu}_i - \zeta_s^0 & s \notin \mathbf{S}^F \end{cases} \tag{5}$$

In the RHS of eqn 5, the top expression is the customer demand for final products minus any initial inventory while the bottom expression is the amount of intermediate *s* required by all tasks consuming it minus its initial inventory. If the initial inventory exceeds the amount required, ω_s will be negative, and the process does not need to produce state *s*.

The minimum amount of state *s* that must be produced by task *i*, v_{is}, is introduced as an intermediate parameter. Although ω_s can be negative, v_{is} must be at least zero.

$$v_{is} = \begin{cases} \max\{\omega_s, 0\} & s \in \mathbf{S}^{ST}, i \in \mathbf{I}_s^+ \\ 0 & s \in \mathbf{S}^{MT} \setminus \mathbf{S}^R, i \in \mathbf{I}_s^+ \end{cases} \tag{6}$$



(a) STN representation of example
(b) Series of calculations in back-propagation
$\tilde{\mu}_{T3}$ and $\tilde{\mu}_{T4}$ must be estimated before ω_{S5}
ω_{S5} and ω_{S5} must be estimated before $\tilde{\mu}_{T2}$

Figure 5. Example of backward propagation of demand. From demand for S6 and S7, we calculate ω_{S6} and ω_{S7}; next, we calculate $\tilde{\mu}_{T3}$ and $\tilde{\mu}_{T4}$, followed by ω_{S5} and ω_{S4}. We cannot find $\tilde{\mu}_{T2}$ yet because ω_{S3} is not known, so we calculate $\tilde{\mu}_{T1}$, followed by ω_{S3} and ω_{S1}. Finally, we calculate $\tilde{\mu}_{T2}$ and ω_{S2}.

where $\mathbf{S}^{ST}$ is the set of states produced by a single task, $\mathbf{S}^{MT}$ is the set of states that can be produced by multiple tasks, and $\mathbf{S}^{R}$ is the set of recycle states. Eqn. 6 does not consider recycle states, which we discuss in §3.3. If multiple tasks can produce a state, we assume that any single task can meet the full demand; therefore, the minimum production of that state by each task is zero, and, for these states, an alternative approach that gives better results is described in §3.6.

After calculating v_{is} for all states produced by a task, we calculate μ_i,

$$\mu_i = \max_{s \in \mathbf{S}_i^+} \left\{ \frac{v_{is}}{\rho_{is}^+} \right\} \tag{7}$$

The term inside the brackets is the total amount task *i* must process to meet the demand for *s*. We take the maximum over all states produced by task *i* to ensure the task satisfies demand for all states. At this point, we must find the minimum attainable production amount, $\tilde{\mu}_i \geq \mu_i$

$$\tilde{\mu}_i = \mu_i + \Delta\mu_i \tag{8}$$

where $\Delta\mu_i$ is the minimum amount that must be added to μ_i to reach an attainable production amount and is discussed in §3.1.2. Parameter $\tilde{\mu}_i$ provides a tighter lower bound on the required production of task *i*. We backward propagate demand until ω_s and $\tilde{\mu}_i$ are known for all states and tasks.

3.1.2. Attainable Production Amounts

After finding μ_i, we need $\Delta\mu_i$ to calculate $\tilde{\mu}_i$. When only one unit can process a task, it is straightforward to find the range of attainable production amounts for any number of batches and to check if the required production is in one of those attainable ranges (see example in Figure 6). If μ_i falls in an attainable region, demand can be met exactly, and $\Delta\mu_i$ is zero; otherwise, $\Delta\mu_i$ is the distance between μ_i and the start of the next attainable range.

However, when multiple units can process a task, we must find and check attainable ranges for every possible combination of batches in units. For example, if two units can process a task, we need to check 1 batch in U1, 0 in U2; 0 in U1, 1 in U2; 1 in U1, 1 in U2; etc. To reduce the number of combinations, we find an upper bound on the number of batches in a particular unit, $\alpha_j^{\max}$, by dividing μ_i by the largest possible batch unit *j* can process and rounding up,

$$\alpha_j^{\max} = \left\lceil \frac{\mu_i}{\beta_j^{\max}} \right\rceil \quad \forall j \in \mathbf{J}_i^P \tag{9}$$

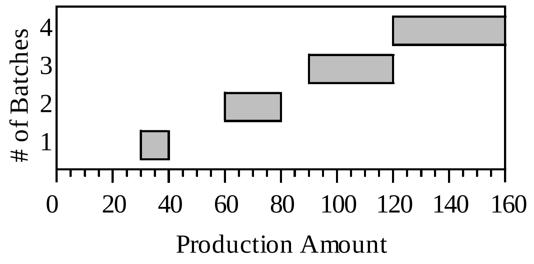


Figure 6. Attainable ranges for a unit with a capacity of 30-40kg are shaded.

When the number of batches for a particular unit is at its upper bound, we are guaranteed the attainable range meets or exceeds demand with just one unit, and the number of batches in all other units is set to zero to eliminate more combinations. In total there are K_i ranges to check,

$$K_i = \prod_{j \in \mathbf{J}_i^P} \left(\alpha_j^{\max} \right) + \left| \mathbf{J}_i^P \right| \tag{10}$$

For each range, $k \in \{1,2,\ldots,K_i\}$, there is a unique combination of α_j^k, where α_j^k gives the number of batches in unit j for range k. The first term in eqn. 10 is the number of ranges with α_j^k = 0,1,2,…($\alpha_j^{\max}$ - 1) for all $j \in \mathbf{J}_i^P$, and the second term is the number of ranges with $\alpha_j^k = \alpha_j^{\max}$ for one $j \in \mathbf{J}_i^P$ and α_j^k =0 for all other $j \in \mathbf{J}_i^P$. We loop over all $k \in \{1,2,\ldots,K_i\}$ and check if μ_i falls into an attainable range. If, for task *i*,

$$\sum_{j \in \mathbf{J}_i^P} \alpha_j^k \beta_j^{\min} \le \mu_i \le \sum_{j \in \mathbf{J}_i^P} \alpha_j^k \beta_j^{\max} \tag{11}$$

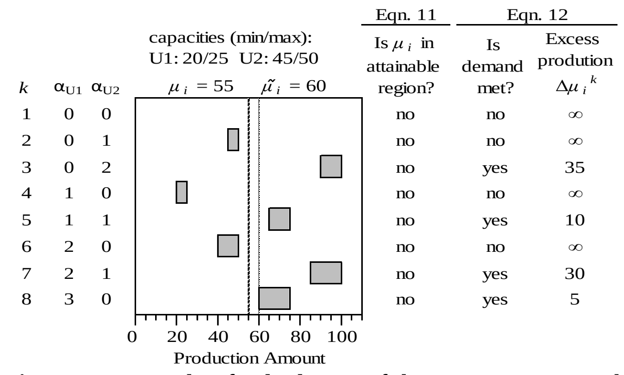
for some k, the minimum required production can be met exactly, and $\Delta\mu_i$ is zero. The LHS of eqn. 11 gives the lower bound of the attainable range, and the RHS gives the upper bound. If μ_i is not attainable for any range, we perform another loop over all k to find the range that is able to meet production requirements and whose lower bound is closest to μ_i.

$$\Delta\mu_i^k = \begin{cases} \sum_{j \in \mathbf{J}_i^P} \alpha_j^k \beta_j^{\min} - \mu_i & \text{if } \sum_{j \in \mathbf{J}_i^P} \alpha_j^k \beta_j^{\min} \ge \mu_i \\ \infty & \text{if } \sum_{j \in \mathbf{J}_i^P} \alpha_j^k \beta_j^{\min} < \mu_i \end{cases} \tag{12}$$

The top line sets $\Delta\mu_i^k$ equal to the excess amount produced (the lower bound on the range minus μ_i) if the combination of units is at least able to meet demand. The second line sets $\Delta\mu_i^k$ to infinity when the combination's capacity is too small. Once all ranges have been checked, $\Delta\mu_i$ is calculated:

$$\Delta\mu_i = \min_k \left\{ \Delta\mu_i^k \right\} \tag{13}$$

As an example, we consider a task that can be processed in U1 or U2 and must process 55 kg (see Figure 7). We find $\alpha_{\text{U1}}^{\max} = 3$ and $\alpha_{\text{U2}}^{\max} = 2$ from eqn. 9; and K_i=8 from eqn. 10. All α_j^k for $k \in \{1,2,\ldots,8\}$ are shown on the left side of Figure 7. The attainable ranges are shaded (eqn. 11). Since μ_i= 55 does not fall in a gray range, eqn. 11 is never satisfied. The excess produced by each range is the distance between μ_i and the start of the next shaded range (eqn. 12). For this example, the smallest excess production is 5,



			Eqn. 11	Eqn. 12	
k	α_{U1}	α_{U2}	Is μ_i in attainable region?	Is demand met?	Excess prodution $\Delta\mu_i^k$
1	0	0	no	no	∞
2	0	1	no	no	∞
3	0	2	no	yes	35
4	1	0	no	no	∞
5	1	1	no	yes	10
6	2	0	no	no	∞
7	2	1	no	yes	30
8	3	0	no	yes	5

Figure 7. Example of calculation of the minimum attainable production amount.

so $\Delta\mu_i$ is 5, and $\tilde{\mu}_i$ is 60.

The algorithm for finding $\Delta\mu_i$ is as follows:

```
1    Calculate $\alpha_j^{\max}$ and $K_i$ and set $\alpha_j^k$.
2    For $k \in \{1,2,\ldots, K_i\}$
3      If eqn. 11 is true
4          Set $\Delta\mu_i = 0$ and stop
5      end
6    end
7    For $k \in \{1,2,\ldots, K_i\}$
8      Calculate $\Delta\mu_i^k$ (eqn. 12)
9    end
10   Calculate $\Delta\mu_i$ (eqn. 13)
```

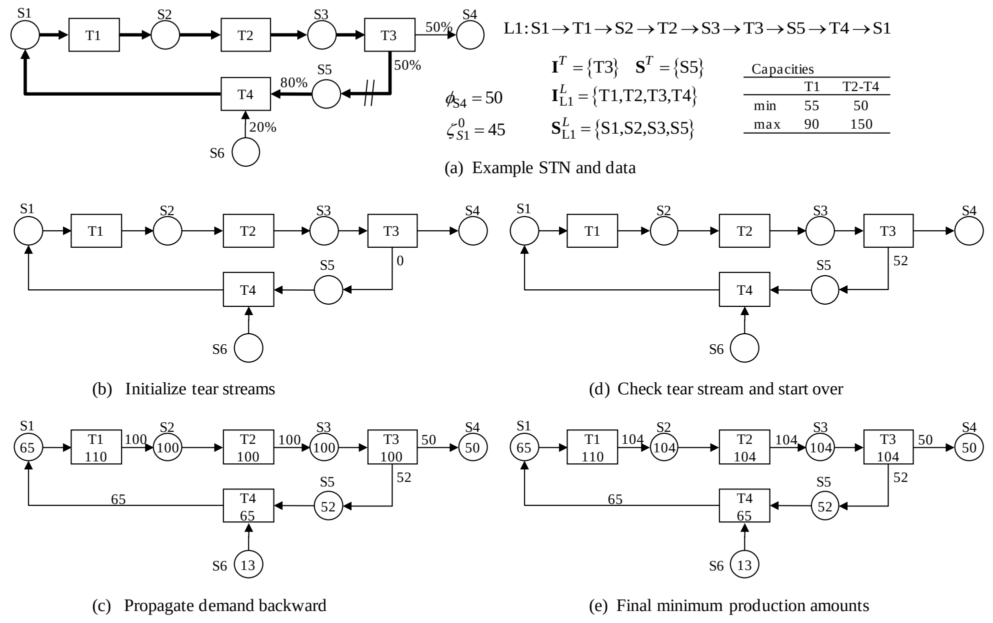
The attainable production amount, $\tilde{\mu}_i$, is now calculated using eqn. 8.

3.2. Networks with Loops

When there are loops in a network, simple backward propagation will not work. We use *tear streams* to break the loop with one tear stream in every loop. A tear stream consists of a task, referred to as the *tear task*, and a state produced by that task, referred to as the *tear state*. Although any stream can be chosen as the tear stream, some choices are more convenient. If a loop has a task that produces a state outside the recycle loop, this task should be used as the tear task; otherwise, any task can be chosen. The sets $\mathbf{I}^T$ and $\mathbf{S}^T$ contain all tear tasks and states, and $\mathbf{I}_l^L$ and $\mathbf{S}_l^L$ give all tasks and states in loop *l*.

Figure 8 provides an example for propagating demand for a network with a loop. We write the value of μ_i inside the box representing task *i*, ω_s inside the circle representing state *s*, and v_{is} next to the stream connecting task *i* to state *s*. For simplicity, all examples have one unit per task, and capacities are given for each task. The recycle loop is shown in bold. We chose T3 as the tear task because it produces S4 outside the recycle loop, and S5 is the tear state. We initialize $v_{\mathrm{T3,S5}}$ for the tear stream to zero (Figure 8b). Now, we can estimate $\tilde{\mu}_{\mathrm{T3}}$ as soon as ω_{S4} is known. We backward propagate demand as follows: $\omega_{\mathrm{S4}}=50 \Rightarrow \tilde{\mu}_{\mathrm{T3}}=100 \Rightarrow \omega_{\mathrm{S3}}=100 \Rightarrow \tilde{\mu}_{\mathrm{T2}}=100 \Rightarrow \omega_{\mathrm{S2}}=100 \Rightarrow \tilde{\mu}_{\mathrm{T1}}=110$ ($\mu_{\mathrm{T1}}=100$ and $\Delta\mu_{\mathrm{T1}}=10$) $\Rightarrow \omega_{\mathrm{S1}}=110-45=65 \Rightarrow \tilde{\mu}_{\mathrm{T4}}=65 \Rightarrow \omega_{\mathrm{S5}}=65*0.8=52$. We stop after calculating $\omega_{\mathrm{S5}}=52$ for the tear state (Figure 8c). We need a minimum of 52 kg of S5, but T3 only produces 50kg (=100*0.5). Therefore, we set $v_{\mathrm{T3,S5}}=52$, and delete ω_s and μ_i for all other states and tasks (Figure 8d). We repeat backward propagation using the new value $v_{\mathrm{T3,S5}}=52$ (Figure 8e). Now, T3 produces the required 52kg of S5.

In general, we begin by initializing v_{is} for all tear streams to zero. We backward propagate demand until ω_s is estimated for a tear state. We update v_{is} for the tear stream using eqn. 6 and compare it to the production of the tear task. If production can meet demand ($v_{is} \le \rho_{is}^{+}\tilde{\mu}_i$), there is no problem, and we continue with the backward propagation. If demand is greater than production ($v_{is} > \rho_{is}^{+}\tilde{\mu}_i$), we reset all other ω_s and $\tilde{\mu}_i$, and restart the backward propagation using the new initial value for the tear stream. If, on the second pass, demand for a tear state still exceeds production, the problem is infeasible with the given initial inventories.



S1
T1
S2
T2
S3
T3
50%
S4
50%
S5
80%
T4
20%
S6
L1: S1→T1→S2→T2→S3→T3→S5→T4→S1
$\mathbf{I}^T=\{T3\}$ $\mathbf{S}^T=\{S5\}$
$\phi_{S4}=50$
$\mathbf{I}^L_{L1}=\{T1,T2,T3,T4\}$
$\zeta^0_{S1}=45$
$\mathbf{S}^L_{L1}=\{S1,S2,S3,S5\}$
Capacities
T1 T2-T4
min 55 50
max 90 150
(a) Example STN and data
(b) Initialize tear streams
(c) Propagate demand backward
(d) Check tear stream and start over
(e) Final minimum production amounts

Figure 8. Network with a loop and no recycle states.

3.3. Networks with Recycle States

Previously, when multiple tasks could produce a single state, we assumed that each task was capable of producing the full amount, and, therefore, each task had a minimum production of zero (eqn. 6). However, when a state $s\in\mathbf{S}^{MT}$ is part of a loop, we can find a better estimate for ν_{is}. We refer to these states as recycle states, $\mathbf{S}^R\subseteq\mathbf{S}^{MT}$. Note that the fraction of a state that is recycled is less than one; in other words, if we start out with some amount of the recycle state and run only tasks in the recycle loop, we will never end up with more of the recycle state than what we had initially.

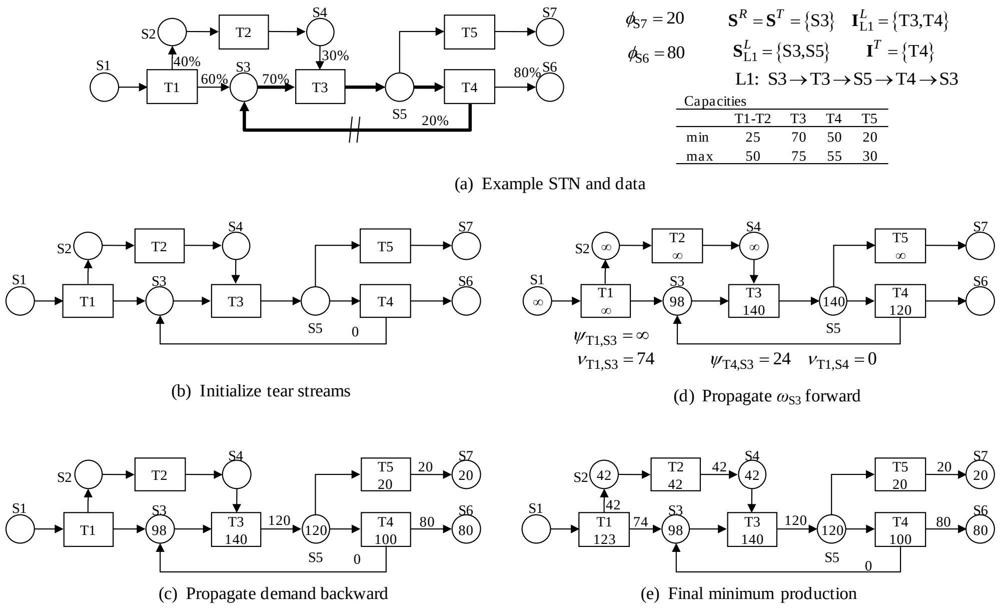
We introduce the following new parameters for networks with recycle states:

ω_s^R	upper bound on the amount of state *s* that can be produced when only the minimum amount of a recycle state, $\omega_{s'}$, is available
μ_i^R	upper bound on the production of task *i* when only the minimum amount of a recycle state is available
ψ_{is}	upper bound on the amount of recycle state *s* produced by task *i* when only the minimum amount of state *s* is available

3.3.1. *Example*

For the network in Figure 9, we start by selecting the tear stream in exactly the same way as in §3.2. We choose T4→S3 as the tear stream and initialize $\nu_{T4,S3}$ to zero. The backward propagation continues until we calculate ω_{S3}=98 in Figure 9c. There is a maximum amount of S3 that T4 (T1) can produce without increasing the required production of T1 (T4) (see Proposition 1); this amount is given by the parameter ψ_{is}, which we estimate by propagating ω_{S3} forward (Figure 9d). In the forward propagation, we start out with ω_{S3}=98kg of S3, which means T3 can produce at most μ^R_{T3}=140kg (=98/0.7) of S5 (assuming there is enough S4). T5 needs $\tilde{\mu}_{T5}$=20kg of S5 (from Figure 9c) and the remaining 120kg (=140-20) of S5 are available for T4. Therefore, T4 produces up to $\psi_{T4,S3}$=24kg

(=120*0.2) of S3. Since ω_{S3}=98kg of S3 are needed in all (from Figure 9c), we conclude that T1 must produce the remaining 74kg (=98-24), and $\nu_{T1,S3}$ =74. T1 is capable of producing all 98kg of S3 ($\psi_{T1,S3}=\infty$ with unlimited initial inventory of feeds), so T4 is not required to produce any S3, and $\nu_{T4,S3}$=0. Since S3 is also a tear state, we must ensure that the production of S3 by T4 exceeds $\nu_{T4,S3}$; since it does (0≤0.2*120), we continue with the backward propagation until we have calculated ω_s and $\tilde{\mu}_i$ for all states and tasks (Figure 9e).



Capacities

	T1-T2	T3	T4	T5
min	25	70	50	20
max	50	75	55	30

(a) Example STN and data

(b) Initialize tear streams

(d) Propagate ω_{S3} forward

(c) Propagate demand backward

(e) Final minimum production

Figure 9. Backward propagation in a network with a single recycle state in a single loop.

3.3.2. General Methods for Recycle States

Backward propagation for networks with recycle states starts by initializing ν_{is} for tear streams to zero. As with standard loops, when ν_{is} is calculated for a tear stream, we check if production exceeds demand. After calculating ω_s for a recycle state, we label this state s* and find the maximum amount that can be recycled. The minimum amount of the labeled recycle state that is required to meet demand, ω_{s*}, is propagated forward around the recycle loop to find ω_s^R and μ_i^R for all states and tasks, respectively. For all states and tasks outside the recycle loop, we set ω_s^R and μ_i^R to infinity; this ensures no feasible solutions are cutoff and assumes that these states are produced starting from feeds with an unlimited supply. Once μ_i^R is known for all tasks producing a state ($i \in \mathbf{I}_s^+$), we calculate ω_s^R. We can calculate ω_s^R for the labeled recycle state immediately.

$$\omega_s^R = \begin{cases} \omega_s + \zeta_s^0 & \text{for } s = s^* \\ \sum_{i \in \mathbf{I}_s^+} \rho_{is}^+ \mu_i^R + \zeta_s^0 & \text{if } s \in \bigcup_{l \in \mathbf{L}_{s^*}^S} \mathbf{S}_l^L \text{ and } \max_{l \in \mathbf{L}_{s^*}^S} \left\{ \left| \mathbf{I}_s^+ \cap \mathbf{I}_l^L \right| \right\} = 1 \end{cases} \tag{14}$$

The first expression gives ω_s^R for the labeled recycle state s*. The second expression is for all other states in the recycle loop except states produced by multiple tasks (other recycle states) within any loop containing s*. The first term in the condition in the second line gives all states in any loop containing s*, and the second term is the number of tasks within the loop that produce state *s*. If there is another recycle state inside the loop, there are nested loops in the network, and §3.4.1 discusses how to find ω_s^R. The total amount of state *s* available, ω_s^R, is the amount produced by all tasks plus the initial inventory. Once ω_s^R is known for all states consumed by task *i* ($s \in \mathbf{S}_i^-$), we find μ_i^R.

$$\mu_i^R = \min_{s \in \mathbf{S}_i^-} \left\{ \frac{\omega_s^R - \sum_{i' \in \mathbf{I}_s^-, i' \neq i} \rho_{i's}^- \tilde{\mu}_{i'}}{\rho_{is}^-} \right\} \quad \text{if } i \in \bigcup_{l \in \mathbf{L}_{s*}} \mathbf{I}_l^L \text{ and } \max_{l \in \mathbf{L}_{s*}} \left\{ \left| \mathbf{S}_i^- \cap \mathbf{S}_l^L \right| \right\} = 1 \tag{15}$$

Eqn. 15 gives μ_i^R for any task inside the recycle loop except for tasks that consume multiple states within any loop containing *s**, which are discussed in §3.4.2. The first term in the condition gives all tasks in any loop containing state s* and the second term gives the number of states within the loop that are consumed by task *i*. In the numerator, the minimum amount of state *s* required for other tasks is subtracted to give the maximum amount of state *s* available for task *i*. Dividing by ρ_{is}^- gives the total amount task *i* needs to process to consume all of the available *s*. Taking the minimum over all states consumed by task *i* ensures there is enough of every state. We do not round μ_i^R to an attainable production amount; rounding up will result in weaker bound, and rounding down may cutoff feasible points. Once μ_i^R is known for all tasks producing the labeled recycle state, we find the maximum amount produced by each task.

$$\psi_{is*} = \rho_{is*}^+ \mu_i^R \quad \forall i \in \mathbf{I}_{s*}^+ \tag{16}$$

Finally, we calculate the minimum amount of state s* produced by each task.

$$\nu_{is*} = \max\left\{ 0, \omega_{s*} - \sum_{i' \in \mathbf{I}_{s*}^+ \setminus \{i\}} \psi_{i's*} \right\} \quad \forall i \in \mathbf{I}_{s*}^+ \tag{17}$$

The most a task can produce of a recycle state is ψ_{is} and the least is zero. If $\nu_{is} > \psi_{is}$, there is not enough initial inventory, and the problem is infeasible. If the demand cannot be satisfied without a particular task, eqn. 17 gives the minimum that task must produce. If a single task or combination of tasks can satisfy the demand, all other tasks do not need to supply any material, and the final term in eqn. 17 is less than zero.

Proposition. There is a maximum amount of each recycle state that can be produced by each task, ψ_{is}, without increasing the minimum required production of other tasks producing that state.

Proof: If ω_s is the minimum amount of recycle state *s* required to meet final demand, and ψ_{is} is the maximum amount of state *s* that can be produced by task *i* when starting with ω_s, then ν_{is} is the amount of *s* produced by task *i* and

$$\nu_{is} = \max\left\{ 0, \omega_s - \sum_{i' \in \mathbf{I}_s^+, i' \neq i} \psi_{i's} \right\} \quad \forall i \in \mathbf{I}_s^+ .$$

Let ξ_{is} be the maximum fraction of state *s* that is recycled by task *i*. If an additional amount of state *s*, $\delta_i \geq 0$, is produced by task *i*, the total amount of *s* now required is at least $\omega_s + \sum_{i \in \mathbf{I}_s^+} \frac{\delta_i}{\xi_{is}}$, and the maximum amount produced by all tasks is $\sum_{i \in \mathbf{I}_s^+} \psi_{is} + \delta_i$. The total amount of *s* that needs to be produced by task *i* is

$$\nu'_{is} = \max\left\{0, \omega_s + \sum_{i' \in \mathbf{I}_s^+} \frac{\delta_{i'}}{\xi_{i's}} - \sum_{i' \in \mathbf{I}_s^+, i' \neq i} \left(\psi_{i's} + \delta_{i'}\right)\right\} \quad \forall i \in \mathbf{I}_s^+ .$$

since $\xi_{is} \le 1$, $\sum_{i \in \mathbf{I}_s^+} \frac{\delta_i}{\xi_{is}} \ge \sum_{i \in \mathbf{I}_s^+, i \neq i'} \delta_i$, and $\nu'_{is} \ge \nu_{is}$. Therefore, all tasks producing state *s* are at their minimum production when all other tasks recycle at most ψ_{is}. ■

3.4. Networks with Nested Loops

Not all networks with multiple loops need the additional procedures described in this section. There are two cases with multiple loops where the forward propagation will not work.

3.4.1. Case 1

When multiple tasks in a single loop produce another recycle state, the forward propagation described previously fails when calculating ω_s^R for that state. For the network in Figure 10a, S3 is produced by T3 and T2, which both belong to loop L3 containing recycle state S2. We propagate the demand as described previously until it is time to propagate ω_{S2} forward and calculate ω_{S3}^R in Figure 10b. We need μ_{T2}^R and μ_{T3}^R to calculate ω_{S3}^R, but ω_{S3}^R is needed to calculate μ_{T3}^R (Figure 10c). During the forward propagation, T2 produces at most μ_{T2}^R =120kg of S3. Previously (Figure 10b), we determined that T2 must produce a minimum of $\tilde{\mu}_{\text{T2}}$=112.5kg of S3, meaning T2 can now produce an extra 7.5kg (=120-112.5). If T3 processes all additional 7.5kg of S3, it will produce another 0.75kg (=7.5*0.1) of S3. Processing the additional 0.75kg gives 0.075kg (=0.75*0.1) more of S3; this is a geometric series that eventually converges to 8.33kg (=7.5/(1-0.1)) more of S3. We add the extra 8.33kg to the original 125kg of S3 needed to meet the demand to give $\omega_{\text{S3}}^R = 133.3$ (Figure 10c). Demand propagation continues until ω_s and $\tilde{\mu}_i$ are known for all states and tasks (Figure 10d).

The example is generalized to any process network. We will refer to the first labeled recycle state as s* (the one for which we are trying to find ν_{is}) and the second recycle state as s** (the one for which we are trying to find ω_s^R). The total amount of state s** available is

$$\omega_s^R = \left(\sum_{i \in \mathbf{I}_s^+ \setminus \mathbf{I}^{RNC}} \left(\rho_{is}^+ \mu_i^R - \nu_{is}\right)\right)\left(\frac{1}{1-\xi_s}\right) + \omega_s + \zeta_s^0 \quad \text{if } s \in \bigcup_{l \in \mathbf{L}_{s^*}^S} \mathbf{S}_l^L \text{ and} \max_{l \in \mathbf{L}_{s^*}^S}\left\{\left|\mathbf{I}_s^+ \cap \mathbf{I}_s^L\right|\right\} \ge 2 \qquad (18)$$

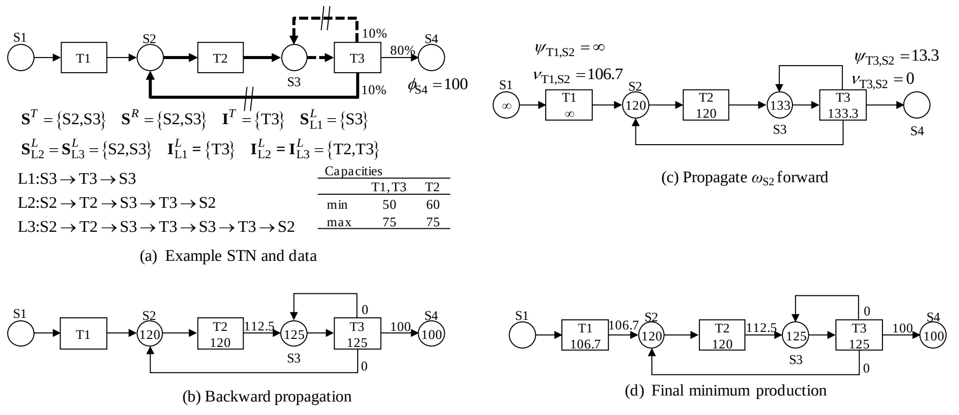

S1
T1
S2
T2
S3
T3
10%
80%
10%
S4
$\phi_{\text{S4}} = 100$
$\mathbf{S}^T = \{\text{S2,S3}\}$ $\mathbf{S}^R = \{\text{S2,S3}\}$ $\mathbf{I}^T = \{\text{T3}\}$ $\mathbf{S}_{\text{L1}}^L = \{\text{S3}\}$
$\mathbf{S}_{\text{L2}}^L = \mathbf{S}_{\text{L3}}^L = \{\text{S2,S3}\}$ $\mathbf{I}_{\text{L1}}^L = \{\text{T3}\}$ $\mathbf{I}_{\text{L2}}^L = \mathbf{I}_{\text{L3}}^L = \{\text{T2,T3}\}$
L1:S3 → T3 → S3
L2:S2 → T2 → S3 → T3 → S2
L3:S2 → T2 → S3 → T3 → S3 → T3 → S2
Capacities
T1,T3 T2
min 50 60
max 75 75
(a) Example STN and data
$\psi_{\text{T1,S2}} = \infty$
$\nu_{\text{T1,S2}} = 106.7$
$\psi_{\text{T3,S2}} = 13.3$
$\nu_{\text{T3,S2}} = 0$
∞
T1 ∞
120
T2 120
133
T3 133.3
(c) Propagate ω_{S2} forward
120
T2 120
112.5
125
T3 125
0
100
(b) Backward propagation
T1 106.7
106.7
(d) Final minimum production

Figure 10. Demand propagation with multiple recycle states in the same loop.

where $\mathbf{I}^{RNC}$ is the set of tasks for which μ_i^R has not been calculated, and ξ_s is the maximum fraction of state *s* that can be recycled by the loop containing only tasks in $\mathbf{I}^{RNC}$. The algorithm provides a method for keeping track of $\mathbf{I}^{RNC}$. Eqn. 18 gives ω_s^R for any states excluded from eqn. 14. The first term is the additional amount available during the forward propagation from all tasks for which μ_i^R has already been calculated. Dividing by 1-ξ_s gives the total additional amount of s** produced during recycling. Finally the original ω_s and the initial inventory are added to give ω_s^R.

Parameter ξ_s can be estimated when there is only one loop containing *s*** but not *s**; i.e., $l\in\{l{:}s^{**}\in\mathbf{S}_l^L, s^*\notin\mathbf{S}_l^L\}$.

$$\xi_s = \frac{\prod_{i\in\mathbf{I}_l^L, s'\in\mathbf{S}_i^+\cap\mathbf{S}_l^L} \rho_{is'}^+}{\prod_{i\in\mathbf{I}_l^L, s'\in\mathbf{S}_i^-\cap\mathbf{S}_l^L} \rho_{is'}^-} \quad \text{for } s\in\mathbf{S}^R \tag{19}$$

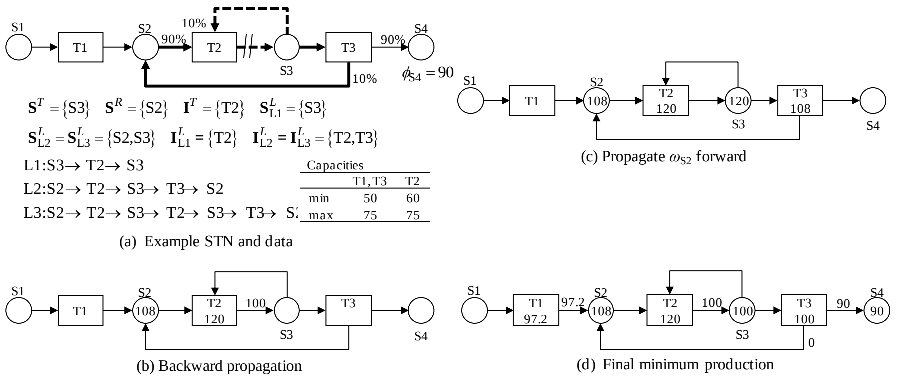
The numerator gives the fraction of material remaining in the loop after task *i*. The denominator is the fraction of the material consumed by task *i* that is already in the recycle loop. Eqn. 19 is only valid when there is only one loop containing s** but not s*. Eqn. 18 is valid for any network, but ξ_s needs to be estimated.

3.4.2. Case 2

When a task inside a loop consumes multiple states in a loop, the forward propagation fails. In Figure 11, T2 consumes S2 and S3 which both belong to recycle loop L2. In Figure 11b, demand has been backward propagated to calculate the demand for recycle state S2. The demand needs to be propagated forward, but we cannot calculate μ_{T2}^R until ω_{S2}^R and ω_{S3}^R are known, and ω_{S3}^R cannot be calculated until μ_{T2}^R is known (Figure 11c). During the forward propagation we have ω_{S2}^R=108kg of S2 available. If there is enough S3, T2 can process at most 120kg. We ignore S3 in eqn. 15, and set $\mu_{\mathrm{T2}}^R = 120$. We continue propagating demand (Figure 11d).

The procedure is easily generalized to any process network,

$$\mu_i^R = \min_{s\in\mathbf{S}_i^-\backslash\mathbf{S}^{RNC}} \left\{ \frac{\omega_s^R - \sum_{i'\in\mathbf{I}_s^-, i'\neq i} \rho_{i's}^- \tilde{\mu}_{i'}}{\rho_{is}^-} \right\} \quad \text{if } i\in\bigcup_{l\in\mathbf{L}_{s^*}^S} \mathbf{I}_l^L \text{ and } \max_{l\in\mathbf{L}_{s^*}^S} \left\{ \left| \mathbf{S}_i^- \cap \mathbf{S}_l^L \right| \right\} \geq 2 \tag{20}$$


S1
T1
S2
90%
10%
T2
S3
T3
90%
S4
10%
$\phi_{S4} = 90$
$\mathbf{S}^T=\{S3\}$ $\mathbf{S}^R=\{S2\}$ $\mathbf{I}^T=\{T2\}$ $\mathbf{S}_{L1}^L=\{S3\}$
$\mathbf{S}_{L2}^L=\mathbf{S}_{L3}^L=\{S2,S3\}$ $\mathbf{I}_{L1}^L=\{T2\}$ $\mathbf{I}_{L2}^L=\mathbf{I}_{L3}^L=\{T2,T3\}$
L1:S3→ T2→ S3
L2:S2→ T2→ S3→ T3→ S2
L3:S2→ T2→ S3→ T2→ S3→ T3→ S2
(a) Example STN and data
108
120
100
(b) Backward propagation
120
108
(c) Propagate ω_{S2} forward
97.2
90
0
(d) Final minimum production

Capacities

	T1, T3	T2
min	50	60
max	75	75

Figure 11. Demand propagation in a network with a recycle state and multiple loops.

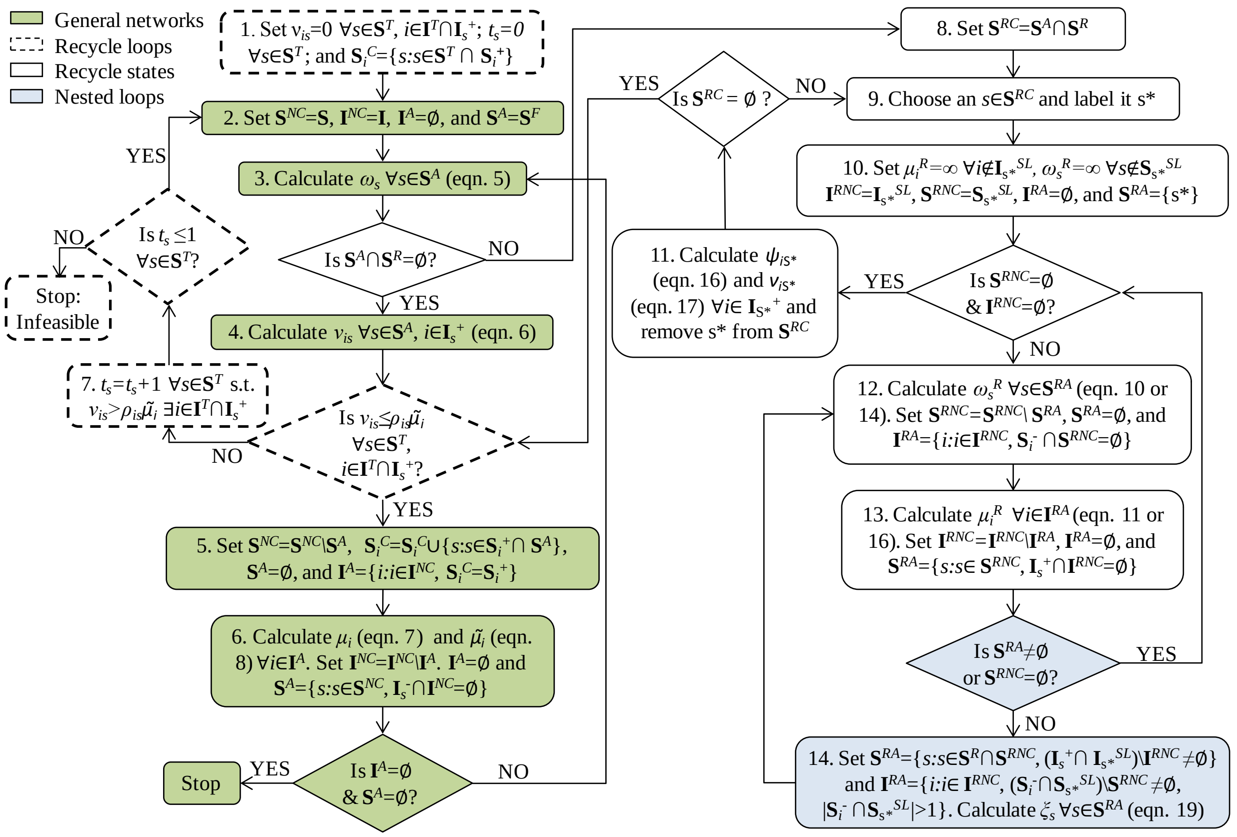
where $\mathbf{S}^{RNC}$ is the set of states for which ω_s^R has not been calculated; these states are ignored when calculating μ_i^R. The algorithm provides a way to keep track of $\mathbf{S}^{RNC}$. Eqn. 20 gives μ_i^R for any tasks in the loop excluded from eqn. 15.

3.5. Complete Algorithm

The complete algorithm combines the tightening procedures for all network types. For each of the four categories, the relevant portions of the algorithm in Figure 12 are colored as in Figure 3. For convenience, we define the new sets:

$\mathbf{S}_s^{SL}$	states in any recycle loop containing state s, $\mathbf{S}_s^{SL} = \bigcup_{l \in \mathbf{L}_s^S} \mathbf{S}_l^L$
$\mathbf{I}_s^{SL}$	tasks in any recycle loop containing state s, $\mathbf{I}_s^{SL} = \bigcup_{l \in \mathbf{L}_s^S} \mathbf{I}_l^L$
$\mathbf{I}^{NC}/\mathbf{I}^{RNC}$	tasks for which μ_i /μ_i^R is not known
$\mathbf{I}^{A}/\mathbf{I}^{RA}$	tasks available (ω_s /ω_s^R has been calculated for all $s \in \mathbf{S}_i^+ / \mathbf{S}_i^-$) for calculating μ_i /μ_i^R
$\mathbf{S}^{NC}/\mathbf{S}^{RNC}$	states for which ω_s /ω_s^R is not known
$\mathbf{S}^{A}/\mathbf{S}^{RA}$	states available (μ_i /μ_i^R has been calculated for all $i \in \mathbf{I}_s^- / \mathbf{I}_s^+$) for calculating ω_s /ω_s^R
$\mathbf{S}_i^C$	states for which v_{is} has been calculated for task i
$\mathbf{S}^{RC}$	recycle states for which we need to perform a forward propagation

In the algorithm, we use the parameter t_s to keep track of how many times tear state $s \in \mathbf{S}^T$ has been updated. As mentioned in §3.2, if the tear state is updated more than once, the instance is infeasible.



🟩 General networks
⬚ Recycle loops
☐ Recycle states
🟦 Nested loops
1. Set v_{is}=0 ∀s∈$\mathbf{S}^T$, i∈$\mathbf{I}^T$∩$\mathbf{I}_s^+$; t_s=0 ∀s∈$\mathbf{S}^T$; and $\mathbf{S}_i^C$={s:s∈$\mathbf{S}^T$ ∩ $\mathbf{S}_i^+$}
2. Set $\mathbf{S}^{NC}$=$\mathbf{S}$, $\mathbf{I}^{NC}$=$\mathbf{I}$, $\mathbf{I}^A$=∅, and $\mathbf{S}^A$=$\mathbf{S}^F$
3. Calculate ω_s ∀s∈$\mathbf{S}^A$ (eqn. 5)
Is $\mathbf{S}^A$∩$\mathbf{S}^R$=∅?
4. Calculate v_{is} ∀s∈$\mathbf{S}^A$, i∈$\mathbf{I}_s^+$ (eqn. 6)
Is v_{is}≤$\rho_{is}\tilde{\mu}_i$ ∀s∈$\mathbf{S}^T$, i∈$\mathbf{I}^T$∩$\mathbf{I}_s^+$?
5. Set $\mathbf{S}^{NC}$=$\mathbf{S}^{NC}$\$\mathbf{S}^A$, $\mathbf{S}_i^C$=$\mathbf{S}_i^C$∪{s:s∈$\mathbf{S}_i^+$∩ $\mathbf{S}^A$}, $\mathbf{S}^A$=∅, and $\mathbf{I}^A$={i:i∈$\mathbf{I}^{NC}$, $\mathbf{S}_i^C$=$\mathbf{S}_i^+$}
6. Calculate μ_i (eqn. 7) and $\tilde{\mu}_i$ (eqn. 8) ∀i∈$\mathbf{I}^A$. Set $\mathbf{I}^{NC}$=$\mathbf{I}^{NC}$\$\mathbf{I}^A$. $\mathbf{I}^A$=∅ and $\mathbf{S}^A$={s:s∈$\mathbf{S}^{NC}$, $\mathbf{I}_s^-$∩$\mathbf{I}^{NC}$=∅}
Is $\mathbf{I}^A$=∅ & $\mathbf{S}^A$=∅?
Stop
7. t_s=t_s+1 ∀s∈$\mathbf{S}^T$ s.t. v_{is}>$\rho_{is}\tilde{\mu}_i$ ∃i∈$\mathbf{I}^T$∩$\mathbf{I}_s^+$
Is t_s≤1 ∀s∈$\mathbf{S}^T$?
Stop: Infeasible
8. Set $\mathbf{S}^{RC}$=$\mathbf{S}^A$∩$\mathbf{S}^R$
Is $\mathbf{S}^{RC}$ = ∅ ?
9. Choose an s∈$\mathbf{S}^{RC}$ and label it s*
10. Set μ_i^R=∞ ∀i∉$\mathbf{I}_{s^*}^{SL}$, ω_s^R=∞ ∀s∉$\mathbf{S}_{s^*}^{SL}$ $\mathbf{I}^{RNC}$=$\mathbf{I}_{s^*}^{SL}$, $\mathbf{S}^{RNC}$=$\mathbf{S}_{s^*}^{SL}$, $\mathbf{I}^{RA}$=∅, and $\mathbf{S}^{RA}$={s*}
Is $\mathbf{S}^{RNC}$=∅ & $\mathbf{I}^{RNC}$=∅?
11. Calculate ψ_{is^*} (eqn. 16) and v_{is^*} (eqn. 17) ∀i∈ $\mathbf{I}_{s^*}^+$ and remove s* from $\mathbf{S}^{RC}$
12. Calculate ω_s^R ∀s∈$\mathbf{S}^{RA}$ (eqn. 10 or 14). Set $\mathbf{S}^{RNC}$=$\mathbf{S}^{RNC}$\ $\mathbf{S}^{RA}$, $\mathbf{S}^{RA}$=∅, and $\mathbf{I}^{RA}$={i:i∈$\mathbf{I}^{RNC}$, $\mathbf{S}_i^-$ ∩$\mathbf{S}^{RNC}$=∅}
13. Calculate μ_i^R ∀i∈$\mathbf{I}^{RA}$ (eqn. 11 or 16). Set $\mathbf{I}^{RNC}$=$\mathbf{I}^{RNC}$\$\mathbf{I}^{RA}$, $\mathbf{I}^{RA}$=∅, and $\mathbf{S}^{RA}$={s:s∈ $\mathbf{S}^{RNC}$, $\mathbf{I}_s^+$∩$\mathbf{I}^{RNC}$=∅}
Is $\mathbf{S}^{RA}$≠∅ or $\mathbf{S}^{RNC}$=∅?
14. Set $\mathbf{S}^{RA}$={s:s∈$\mathbf{S}^R$∩$\mathbf{S}^{RNC}$, ($\mathbf{I}_s^+$∩ $\mathbf{I}_{s^*}^{SL}$)\$\mathbf{I}^{RNC}$≠∅} and $\mathbf{I}^{RA}$={i:i∈ $\mathbf{I}^{RNC}$, ($\mathbf{S}_i^-$∩$\mathbf{S}_{s^*}^{SL}$)\$\mathbf{S}^{RNC}$≠∅, |$\mathbf{S}_i^-$ ∩$\mathbf{S}_{s^*}^{SL}$|>1}. Calculate ξ_s ∀s∈$\mathbf{S}^{RA}$ (eqn. 19)
YES
NO

Figure 12. General algorithm for propagating demand through a network.

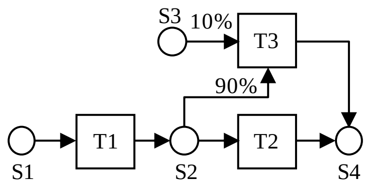
3.6. States Produced by Multiple Tasks

When multiple tasks can produce a state, eqn. 6 gives $v_{is} = 0$ for each task, which may mean μ_i is zero for upstream tasks that must produce some material. In Figure 13, S4 is produced from T2 or T3, so each task has a minimum production of zero. However, T2 and T3 both require S2, which is produced by T1. If we used eqn. 6 to find v_{is} in step 4 of the algorithm, we find that $\mu_{\mathrm{T1}}=0$, which is not a tight lower bound. Instead, we can solve a simple linear program (LP$_i$) to find v_{is}.

$$\begin{aligned}
&\min Q_i \\
\text{s.t.}\quad & \zeta_s^0 + \sum_{i' \in \mathbf{I}_s^+} \rho_{i's}^+ Q_{i'} \geq \sum_{i' \in \mathbf{I}_s^-} \rho_{i's}^- Q_{i'} \;\; \forall s \qquad \forall i \qquad\qquad (\mathrm{LP}_i) \\
& \sum_{i' \in \mathbf{I}_s^+} \rho_{i's}^+ Q_{i'} \geq \omega_s \;\; \forall s \notin \mathbf{S}^{NC}
\end{aligned}$$

where Q_i is a positive variable for the total amount of material task i produces in a particular solution of (LP$_i$). The first constraint requires that, for each state, the amount produced plus any initial inventory is greater than the amount consumed. The second constraint enforces that the amount produced of a state must exceed ω_s and is only written for states for which ω_s is known. The objective is to minimize the amount produced by task i. When it is time to find v_{is} in the algorithm, we solve (LP$_i$) for task i and then multiply the optimal objective value by ρ_{is}^+ to get v_{is}. When a task (other than a tear task) produces multiple states, we only need to solve (LP$_i$) once and multiply by the optimal value by ρ_{is}^+ for each state $s \in \mathbf{S}_i^+$. For tear tasks that produce multiple states, we solve (LP$_i$) twice, first to calculate v_{is} for any non-tear states produced by the task and second to update v_{is} for the tear state.

When we use this method, we solve (LP$_i$) to find v_{is} for all tasks in the network. This method also provides an alternative to the methods for recycle states described in §3.3 and §3.4. For more complicated networks with nested loops, this method may give tighter bounds and may be much simpler to use. Demand is still backward propagated according to the algorithm in Figure 12 with two changes: (1) we calculate v_{is} in step 4 of the algorithm with (LP$_i$) instead of with eqn. 6, and (2) we skip (or answer yes to) all steps involving recycle states (steps 8-14).



Method for v_{is}	$\tilde{\mu}_i$ T1	T2	T3	ω_s S1	S2	S3	S4
Eqn. 6	0	0	0	0	0	0	50
(LP)	15	0	0	45	0	0	50

Figure 13. Compares the two methods for finding v_{is}: (1) using eqn. 6 and (2) solving (LP). Customers demand 50kg of S4 and all tasks are processed in a unit with a capacity of 0-50kg.

4. Valid Inequalities

We write tightening constraints after calculating $\tilde{\mu}_i$ and ω_s for all tasks and states. We find the minimum number of batches processed by a task, λ_i, by dividing the required production by the largest possible size of a single batch of task i and rounding up.

$$\lambda_i = \left\lceil \frac{\tilde{\mu}_i}{\max\limits_{j \in \mathbf{J}_i^P} \left\{ \beta_j^{\max} \right\}} \right\rceil \tag{21}$$

The minimum number of batches provides a lower bound for the sum of the assignment variables.

$$\sum_{j \in \mathbf{J}_i^P, t} X_{ijt}^P \geq \lambda_i \quad \forall i \tag{22}$$

When multiple tasks produce a state, eqn. 22 may not provide a tight bound. Instead, we find bounds for the minimum number of batches from all tasks producing a state, κ_s. Again, we divide the required amount of each state by the largest possible amount of that state that can be produced in a single batch and round up,

$$\kappa_s = \left\lceil \frac{\omega_s}{\max\limits_{i \in \mathbf{I}_s^+, j \in \mathbf{J}_i^P} \left\{ \rho_{is}^+ \beta_j^{\max} \right\}} \right\rceil \tag{23}$$

Now, κ_s provides a bound for the assignment variables for all tasks producing state s.

$$\sum_{i \in \mathbf{I}_s^+, j \in \mathbf{J}_i^P, t} X_{ijt}^P \geq \kappa_s \quad \forall s \in \mathbf{S}^{MT} \tag{24}$$

Eqn. 24 does not provide new information for states produced by a single task. Tightening constraints 22 and 24 have the same form as those proposed by Burkard and Hatzl (2005) and Janak and Floudas (2008), but the bounds from the different methods may be different.

When a task can be processed in units with very different capacities, eqn. 22 and 24 may not provide tight bounds. Instead, we use the maximum batch-size and minimum production requirements to find stronger general inequalities.

$$\sum_{j \in \mathbf{J}_i^P, t} \beta_j^{\max} X_{ijt}^P \geq \hat{\mu}_i \quad \forall i \tag{25}$$

where $\hat{\mu}_i$ is the tightest bound for eqn. 25. In §3.1.2, we calculate $\tilde{\mu}_i$ based on $\beta_j^{\min}$. Since eqn. 25 is based on $\beta_j^{\max}$, $\tilde{\mu}_i$ does not provide a tight bound. We find $\hat{\mu}_i$ by performing another loop over all $k \in \{1,2,\ldots,K_i\}$ with the same α_j^k used to calculate the attainable production amount.

$$\hat{\mu}_i^k = \begin{cases} \sum\limits_{j \in \mathbf{J}_i^P} \alpha_j^k \beta_j^{\max} & \text{if } \mu_i \leq \sum\limits_{j \in \mathbf{J}_i^P} \alpha_j^k \beta_j^{\max} \\ \infty & \text{otherwise} \end{cases} \tag{26}$$

The top expression sets $\hat{\mu}_i^k$ to the production amount at the end of attainable range k if the range is large enough to meet demand. The second expression sets $\hat{\mu}_i^k$ to infinity when the attainable range is too small to meet demand. After checking all ranges, we set $\hat{\mu}_i$ to the smallest of all $\hat{\mu}_i^k$.

$$\hat{\mu}_i = \min_k \left\{ \hat{\mu}_i^k \right\} \tag{27}$$

The LHS of eqn. 25 can only take on a discrete set of values when X_{ijt}^P is binary; these values are given by $\Sigma\alpha_j \beta_j^{\max}$ where α_j is an integer and are the same as the production amounts at the end of the attainable ranges shown in Figure 7. Therefore all possible values for the LHS occur at the end of an attainable range. To find the value of $\hat{\mu}_i$ that gives the tightest bound, we use the production amount at the end the attainable region that is closest to but greater than $\tilde{\mu}_i$ (see Figure 14). In general, $\hat{\mu}_i$ is the smallest production amount at the end of an attainable range ($\Sigma\alpha_j^k \beta_j^{\max}$) that is at least able to meet demand ($\tilde{\mu}_i$).

Eqn. 25 can also be written for states produced by multiple tasks:

$$\sum_{i \in \mathbf{I}_s^+ \, j \in \mathbf{J}_i^P, t} \rho_{is}^+ \beta_j^{\max} X_{ijt}^P \geq \omega_s \quad \forall s \in \mathbf{S}^{MT} \tag{28}$$

Figure 15 illustrates the effect of the tightening constraints on the feasible region of the LP-relaxation of the model for the example in Figures 7 and 14. The total demand is 55, μ_i=55, $\tilde{\mu}_i$ =60, and $\hat{\mu}_i$ =75. The lines show eqn. 22 and eqn. 25 with three RHS values: μ_i, $\tilde{\mu}_i$, and $\hat{\mu}_i$. Eqn. 22 requires at least two batches. Improving (increasing) the RHS of constraint 25 improves the LP-relaxation of the model.

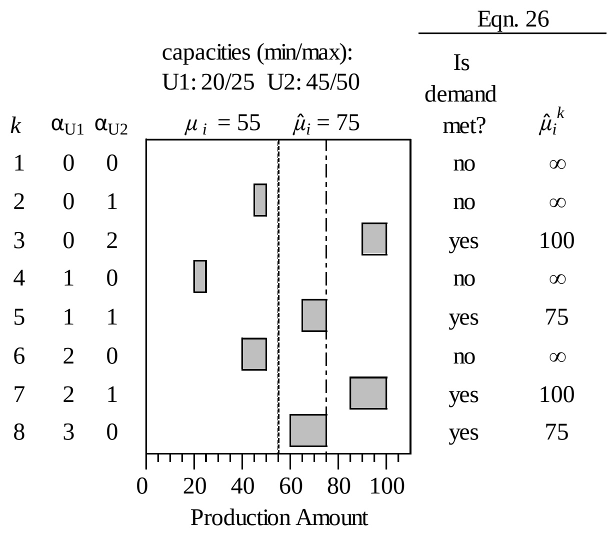
When backlogging is not allowed, we use due times to tighten the formulation further. Now, all parameters include a time index (ϕ_{st}, ω_{st}, etc.) and are zero for times when no orders are due. For every due time, we add all earlier orders to get ϕ_{st} and calculate the other parameters as before. Instead of summing the assignment variable over the entire horizon, we sum it only over the time available to start the task.

$$\sum_{j \in \mathbf{J}_i^P} \sum_{t'=0}^{t-\tau_{ij}} X_{ijt'}^P \geq \lambda_{it} \quad \forall i, t \tag{29}$$

For times when no orders are due, all parameters are zero, and eqn. 29 is not written. We can also write constraints 24, 25, and 28 to consider due times by changing the times over which the assignment variables are summed. For the example in Figure 1, suppose customer C1 demands 30kg of S3 at *t=6* and 25kg of S4 at *t=9*, and customer C2 demands 60kg of S3 at *t=9*. The total final demands up to *t=6* and up to *t=9* are calculated. Table 2 lists order sizes (ϕ_{st}), required productions for states (ω_{st}) and tasks ($\tilde{\mu}_{it}$), and assignment bounds (λ_{it}).

Table 2. Parameter values when due times are used to tighten the formulation.

t	Order (ϕ_{st})		ω_{st}	$\tilde{\mu}_{it}$			Assignment Bounds (λ_{it})		
	S3	S4	S2	T1	T2	T3	T1	T2	T3
6	30	0	35	35	35	0	1	1	0
9	90	25	125	125	90	35	3	2	1



			Eqn. 26	
k	α_{U1}	α_{U2}	Is demand met?	$\hat{\mu}_i^k$
1	0	0	no	∞
2	0	1	no	∞
3	0	2	yes	100
4	1	0	no	∞
5	1	1	yes	75
6	2	0	no	∞
7	2	1	yes	100
8	3	0	yes	75

Figure 14. Example of $\hat{\mu}_i$ calculation. The LHS of eqn. 25 must belong to the set {0, 25, 50, 75…}. Since the minimum production is $\tilde{\mu}_i = 60$, the LHS of eqn. 25 must be at least 75 in any feasible solution.

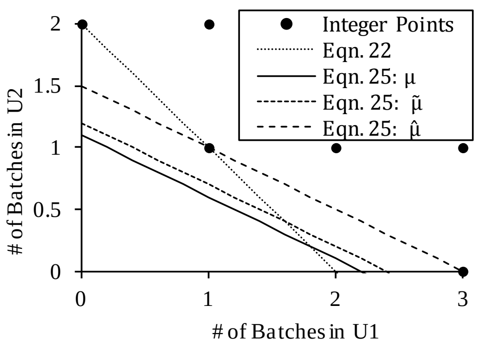


Figure 15. Effect of tightening constraints on feasible region.

5. Computational Study

5.1. Problem Instances

To determine how well the tightening constraints work, we test 72 instances with different objectives, networks, problem features, and time horizons. Each instance is run with different tightening formulations. Specifically, we compare the two types of constraints based on the number of batches (eqn. 22 and 24) and on the minimum production (eqn. 25 and 28). We also look at the effectiveness of the extensions for processes with due times.

We consider two objectives: minimization of processing cost and makespan. The processing cost depends on the task and unit, but not on the batch-size.

$$\text{cost} = \sum_{t,i,j\in\mathbf{J}_i^P} \pi_{ij} X_{ijt}^P \tag{30}$$

$$MS \geq \sum_{i\in\mathbf{I}_j} X_{ijt}^P \left(t+\tau_{ij}\right) \quad \forall j \in \mathbf{J}^P, t \tag{31}$$

Table 3. The seven constraint sets.

Set	Minimum # of batches (eqn. 22, 24)	Minimum production (eqn. 25, 28)	Due Times (eqn. 29)
F1			
F2	X		
F3		X	
F4	X	X	
F5	X		X
F6		X	X
F7	X	X	X

We use four networks, N1-N4, to determine the effect of recycle streams and states produced by multiple tasks on the effectiveness of the tightening constraints (Figure A1 in Electronic Companion). Process and order data are also given in the Electronic Companion (Tables A1-A4). Networks N1 and N4 have no recycle streams, but network N4 has a state produced by multiple tasks. Networks N2 and N3 have a recycle stream. Network N2 is taken from Kondili et al (1993).

We consider three problem classes with different features for each network. Problem (a) has all orders due at the end of the time horizon. To determine the impact of including due times in the tightening constraints, problem (b) has intermediate due times. To compare the two types of tightening constraints, unit capacities are changed for problem (c) so at least one task can be processed by units with different capacities. We solve each instance with three time horizons: 40, 80, and 120 hours. The order size and due time scale with the time horizon to ensure all time horizons lead to schedules that are similarly loaded. Starting with a 40-hour horizon, the order size and due time are doubled for an 80-hour horizon and tripled for a 120-hour horizon.

We consider seven sets of tightening constraints (Table 3). All formulations contain eqns. 1-4. Formulation F1 has no tightening constraints. F2 includes the constraints based on the minimum number of batches. Set F3 contains the constraints based on the required production. F4 combines F2 and F3. Finally, sets F5-F7 are the same as sets F2-F4 but with due times. Only problem (b) is run with formulations F5-F7, and the results focus on formulations F1-F4.

All problems are solved using GAMS 23.7/CPLEX 12.3 on a computer with 6 GB of RAM and a 2.67 GHz Intel Core (i7-920) processor running on Windows 7. We use a resource limit of 1800s. Model and solution statistics for all problems are given in the Electronic Companion (Tables A5-A10). For the 36 instances, implementing the demand propagation algorithm takes an average of 0.26s with a maximum time of 4.3s when using eqn. 6 to calculate v_{is}, and an average of 2.4s with a maximum time of 11.9s when using (LP$_i$) to calculate v_{is}.

5.2. Results

Table 4 shows aggregate results for formulations F1 and F4. Problems are grouped into four categories: (1) those that are solved to optimality by both formulations, (2) those that are solved to optimality only by F4, (3) those that are never solved to optimality, and (4) those that are solved to optimality only by F1. The average computational time or optimality gap is given for the problems in each of the four categories. For cost minimization, 14 problems are in the first category, and tightening reduces the computational time from 189s to 1.4s. Category 2 contains 20 problems, and the tightened formulation solves these problems in an average of 2.4s while, without tightening, there is an average optimality gap of 1.3% after 1800s. For the two problems in category 3, tightening reduces the average optimality gap from 2.1% to 1.5%. There are no problems in the final category. For makespan minimization, 34 problems fall in the first category, and the average computational time is 61s without tightening and 32s with tightening. There is one problem each in categories 2 and 4.

Figure 16 is a performance profile for formulations F1-F4. For each instance, the computational time for each formulation is divided by the fastest time over all formulations for that instance to give r. The vertical axis is the probability a formulation will solve a problem within s (on the horizontal axis) times the fastest formulation. For cost minimization, the untightened formulation solves only 17% of the problems within 15 times the fastest formulation, but F4 solves 94% of the problems within the same time. Using the tightening constraints based on the minimum production (in either F3 or F4) give the best results. For makespan minimization, F3 performs the best, but tightening is generally less effective and does not always improve solution time.

Figure 17 compares the four formulations for the two objectives. Each point is the average computational time or optimality gap over all networks, problem classes, and time horizons (36 runs per point). The tightening constraint based on the number of batches (F2) is the least effective, and F3 and F4 are about equivalent. For cost minimization, tightening increases the fraction of problems solved to optimality and decreases the computational time, while for makespan minimization, it does not have a large impact on the computational time. Since makespan minimization appears to be an easier problem and tightening is not as effective, we will focus on cost minimization for the remainder of the section.

Table 4. Summary of tightening results with average computational time or optimality gap.

	Cost Minimization				Makespan Minimization			
Category	1	2	3	4	1	2	3	4
# of problems	14	20	2	0	34	1	0	1
F1	189s	1.3%	2.1%	--	61s	2.0%	--	302s
F4	1.4s	2.4s	1.5%	--	32s	18s	--	1.1%

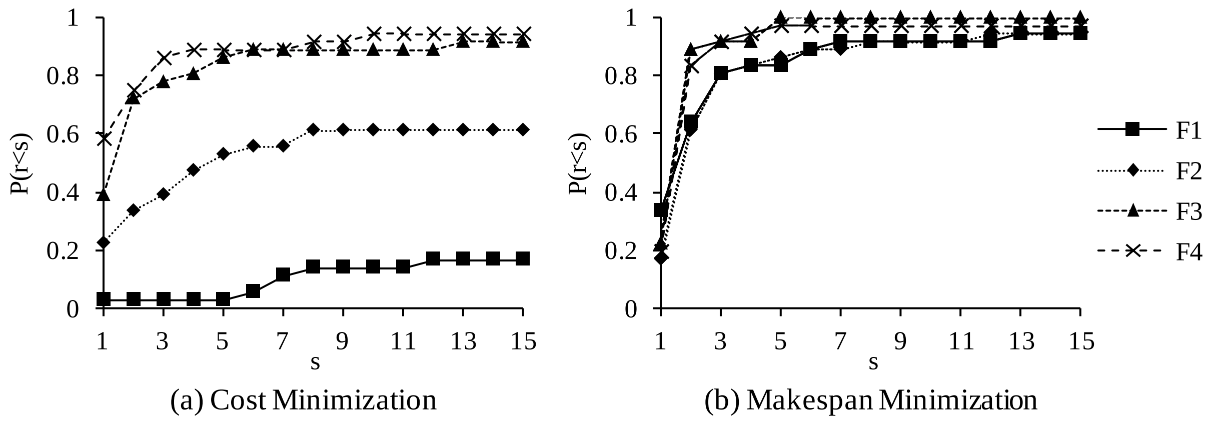


Figure 16. Performance profiles comparing tightening formulations F1-F4.

Figure 18 compares the four formulations for the four networks averaged over the three problem types and three time horizons (9 runs per point). All tightened formulations perform better than F1, and F3 and F4 are the most effective. When there are recycle streams, as in N2 and N3, F2 does not perform as well. Figure 19 shows results for the three problem classes averaged over the four networks and three time horizons (12 runs per point). For all problem classes, tightening based on the minimum number of batches (F2 and F5) is less effective than using the minimum production requirements (F3 and F6), but using both types of inequalities in F4 and F7 is sometimes better. When the problem includes due times in type (b), adding the due times to the tightening constraints (F5-F7) has little impact on the solution time. Finally, in Figure 20 we show results for the different time horizons, where each point is an average over all problems classes and networks (12 runs per point). As expected, increasing the time horizon decreases the fraction of problems solved to optimality, but using tightening increases the fraction solved so only 8% of the problems solved with a 40-hour horizon are not solved with 80- or 120-hour horizons. We also note that the average computational requirement and optimality gap decrease notably.

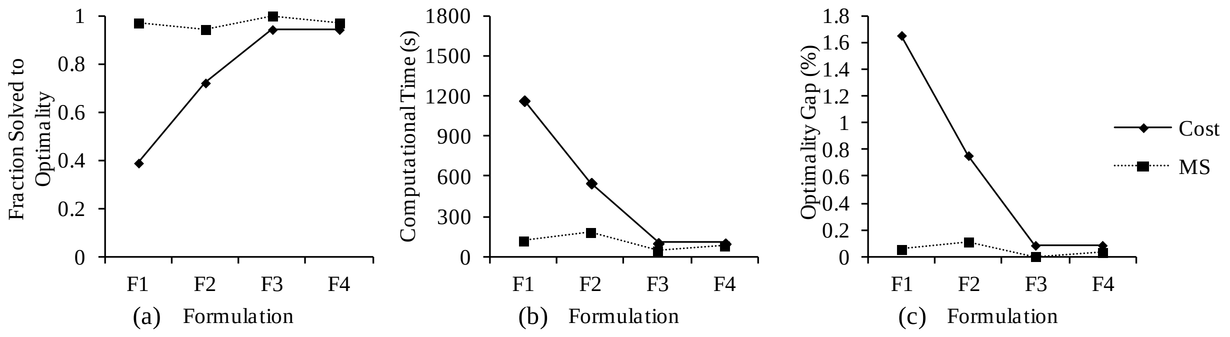


Figure 17. Results for the two objectives.

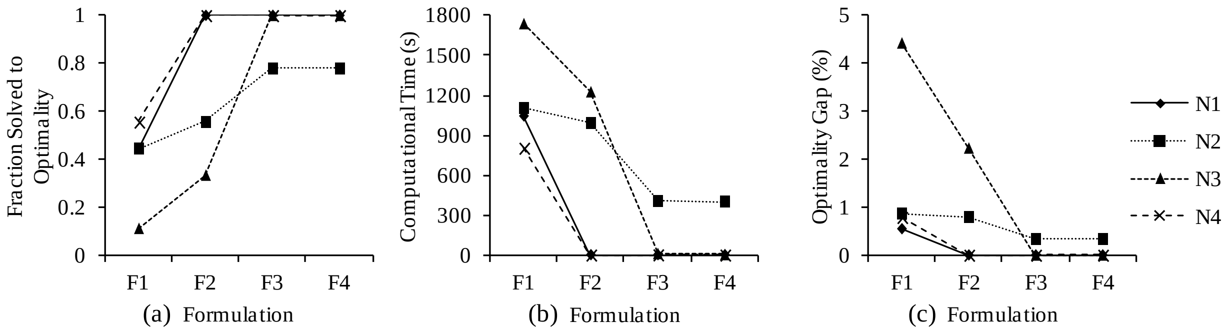


Figure 18. Results for four process networks (cost minimization).

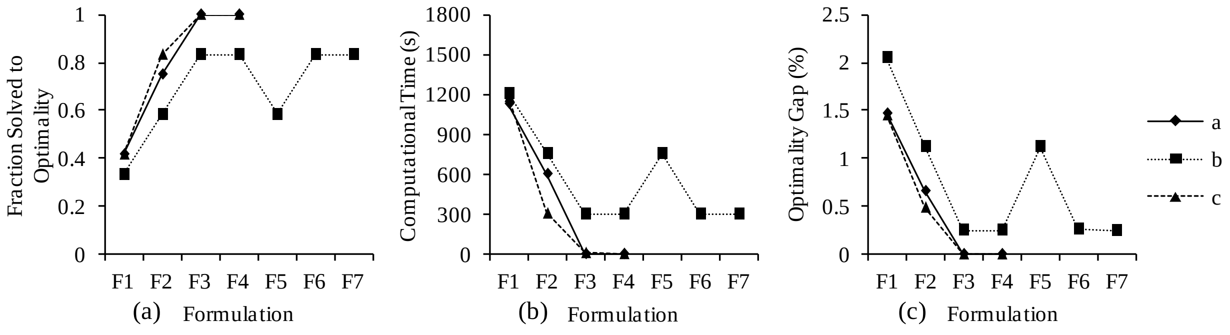


Figure 19. Results for the three problem classes (cost minimization).

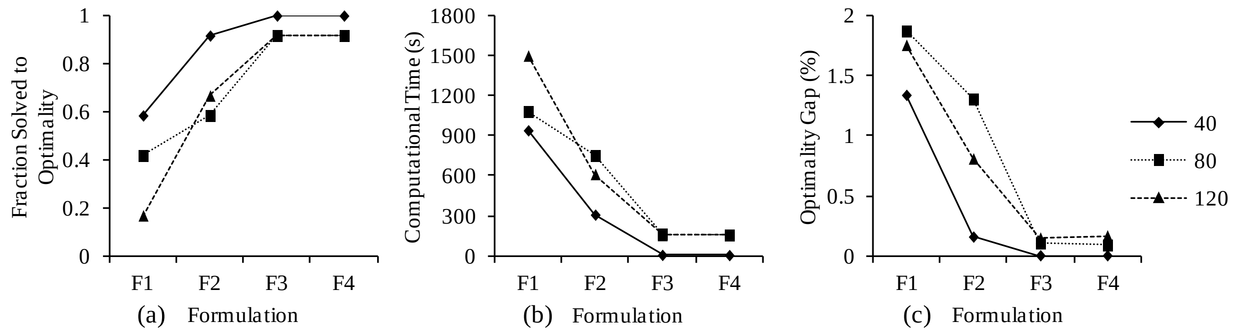


Figure 20. Results for three time horizons (cost minimization).

5.3. Model and Solution Statistics

Table 5 lists model and solution statistics for four representative instances for cost minimization and a 40-hour time horizon. F2-F4 only add a moderate number of equations, about one or two per task. F5-F7 add many more constraints, about one or two per task per due time. The objective for the LP-relaxation improves as tightening constraints are added and is always best when both types of tightening constraints are used.

Table 5. Model and solution statistics for representative problems for cost minimization and a 40-hour horizon.

Formulation	F1	F2	F3	F4	F1	F2	F3	F4	F5	F6	F7
	Problem 1a				Problem 2b						
Discrete vars	230	230	230	230	312	312	312	312	312	312	312
Continuous vars	878	878	878	878	1206	1206	1206	1206	1206	1206	1206
Constraints	1149	1154	1154	1159	1354	1360	1360	1366	1430	1429	1505
LP-relaxation	311.14	320	320	320	424.66	430.16	431.66	431.66	441.65	443.56	443.56
Objective	320	320	320	320	490	490	490	490	490	490	490
Nodes	6615	0	0	0	3199677	3441882	20019	1668	2734482	747	615
CPU time (s)	2.75	0.09	0.09	0.09	1800.02	1800	18.97	3.45	1800.02	2.32	2.39
Gap (%)	0	0	0	0	1.87	1.87	0	0	1.87	0	0
	Problem 3c				Problem 4a						
Discrete vars	427	427	427	427	467	467	467	467			
Continuous vars	1501	1501	1501	1501	1338	1338	1338	1338			
Constraints	1846	1854	1854	1862	1723	1731	1731	1739			
LP-relaxation	253.46	294.43	305.67	309	328.58	355.92	352.13	357.58			
Objective	310	310	310	310	380	380	380	380			
Nodes	2830545	56924	0	0	2668183	502	0	0			
CPU time (s)	1251.22	68.89	0.09	0.09	1800	0.64	0.09	0.09			
Gap (%)	0	0	0	0	2.54	0	0	0			

5.4. Long Solution Times

Figure 21 presents a summary of the computational results – it compares the average optimality gap after 1800s and the solution time on a problem-by-problem basis for formulations F1 and F4 for cost minimization. For the fourteen problems are solved to optimality with both formulations, the proposed methods reduce the average computational time from 189s to 1.4s (speedup of more than two orders of magnitude). Twenty problems that cannot be solved to optimality by F1 within 1800 sec (average optimality gap = 1.3%) are solved to optimality by F4 with an average computational time of only 2.4s, which represents a speedup of more than three orders of magnitude.

Importantly, even this type of speedup is an underestimation of the actual enhancement our methods lead to when applied to hard instances. To illustrate the point, we solved an instance of Network 3 with a 120-hour time horizon using formulation F1 with a resource limit of 16 hours (57,600s). After 16 hours, the optimality gap was 2.22%. The same instance is solved to optimality in 5.8s using F4, a four orders of magnitude improvement. We observed that many of the instances that did not solve using F1 within the 1800s resource limit require a much longer time to solve.

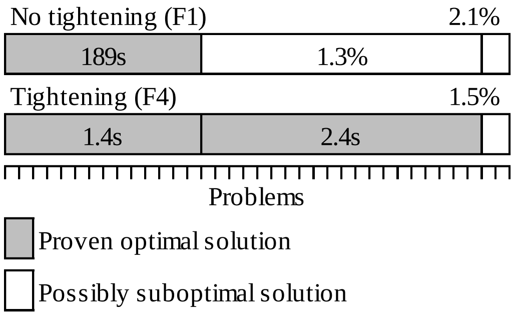


Figure 21. Comparison of average optimality gap and average solution time for formulations F1 and F4.

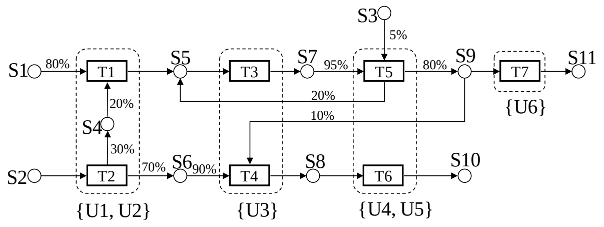


Figure 22. Network for the instance that is solved with a long solution time.

6. Conclusions

We presented methods for the effective solution of chemical production scheduling problems using time-indexed MIP models. We developed a demand propagation algorithm for the calculation of parameters that are used to generate tightening constraints. Specifically, we calculate lower bounds on (i) the number of batches of each task in any feasible solution, (ii) the number of batches of tasks producing a state, (iii) the total amount processed by one task, and (iv) the amount of each state that has to be produced. The algorithm exploits minimum batch-size restrictions, which are common in chemical manufacturing, but have received limited attention in the literature. Furthermore, it is applicable to facilities with recycling of material, a class of problems that cannot be addressed using more standard methods based on bill-of-material information. Most importantly, the demand propagation algorithm has minimal computational requirements (less than 1 minute of CPU time), even for complex large-scale manufacturing facilities. Using the aforementioned bounds, we generated four types of tightening constraints, which lead to substantial computational improvements. The tightening constraints based on the minimum production are more effective than those based on the number of batches, while using both types of constraints leads to the most effective solution. We also observed that the tightening constraints are especially effective for longer time horizons. The proposed methods, which are applicable to all time-indexed MIP scheduling models for chemical manufacturing, can potentially lead to significant computational improvements in a wide range of commercial optimization-based tools.

Acknowledgements

The authors would like to acknowledge financial support from the National Science Foundation under grant CTS-0547443.

Nomenclature

Sets

Indices/Sets

$i,i' \in \mathbf{I}$	tasks
$j \in \mathbf{J}$	processing units
$s,s' \in \mathbf{S}$	states
$t,t' \in \mathbf{T}$	time points
k	range
$l \in \mathbf{L}$	recycle loops

Subsets

Sets for any network:

$\mathbf{I}_s^+ / \mathbf{I}_s^-$	tasks producing/consuming state *s*
$\mathbf{I}_j$	tasks that can be performed by processing unit *j*
$\mathbf{J}_i^P$	processing units that can process task *i*
$\mathbf{S}^{ST}$ / $\mathbf{S}^{MT}$	states produced by a single/multiple tasks
$\mathbf{S}_i^+ / \mathbf{S}_i^-$	states produced/consumed by task *i*
$\mathbf{S}^F$	final products

Sets for networks with recycle loops:

$\mathbf{I}_l^L / \mathbf{S}_l^L$	tasks/states in recycle loop *l*
$\mathbf{I}^T/\mathbf{S}^T$	tear tasks/states
$\mathbf{S}^R$	recycle states (states in a recycle loop that are produced by multiple tasks)
$\mathbf{L}_s^S$	loops containing state *s*

Sets used in the algorithm:

$\mathbf{S}_s^{SL}$	set of states in any recycle loop containing state *s*
$\mathbf{I}_s^{SL}$	set of tasks in any recycle loop containing state *s*
$\mathbf{I}^{NC}/\mathbf{I}^{RNC}$	tasks for which μ_i / μ_i^R is not known
$\mathbf{I}^A/\mathbf{I}^{RA}$	tasks available for calculating μ_i / μ_i^R
$\mathbf{S}^{NC}/\mathbf{S}^{RNC}$	states for which ω_s / ω_s^R is not known
$\mathbf{S}^A/\mathbf{S}^{RA}$	states that are available for calculating ω_s / ω_s^R
$\mathbf{S}_i^C$	states for which v_{is} has been calculated for task *i*
$\mathbf{S}^{RC}$	recycle states for which we need to perform a forward propagation

Parameters

Problem data:

ϕ_s	total demand for state *s*
$\beta_j^{\min} / \beta_j^{\max}$	minimum/maximum capacity of unit *j*

$\gamma_s^{\max}$	maximum inventory of state *s*
$\rho_{is}^{+} / \rho_{is}^{-}$	fraction of state *s* produced/consumed by task *i*
τ_{ij}	fixed processing time for task *i* in unit *j*
ζ_s^{0}	initial inventory of state *s*
π_{ij}	cost of carrying out task *i* in unit *j*

Parameters calculated by tightening methods:

ω_s	lower bound on the amount of state *s* required to meet final demand
$\mu_i/\tilde{\mu}_i$	lower bound on the production of task *i* required to meet final demand
$\Delta\mu_i$	minimum amount added to μ_i to reach an attainable production amount
ν_{is}	lower bound on the amount of state *s* that must be produced by task *i*
λ_i	lower bound on the number of batches task *i* processes
κ_s	lower bound on the number of batches producing state *s*
α_j^k	number of batches in unit *j* during iteration *k*
$\alpha_j^{\max}$	maximum number of batches in unit *j* needed to satisfy final demand
ω_s^R	upper bound on the amount of state *s* that can be produced when only the minimum amount of a recycle state is available
μ_i^R	upper bound on the production of task *i* when only the minimum amount of a recycle state is available
ψ_{is}	upper bound on the amount of recycle state *s* produced by task *i* when only the minimum amount of state *s* is available
ξ_s	maximum fraction of state *s* that can be recycled

Variables

Binary variables:

X_{ijt}^P	is one if unit *j* processes task *i* starting at time *t* at time *t*

Continuous non-negative variables:

B_{ijt}	batch-size of task *i* processed in unit *j* starting at time *t*
S_{st}	inventory of state *s* at time *t*
P_{st}	amount of feed state *s* purchased at time *t*
D_{st}	amount of final product *s* delivered to customers at time *t*
Q_i	total amount of material task *i* produces in a particular solution of (LP).

References

Baptiste, P.; Le Pape, C.; Nuijten, W. *Constraint-Based Scheduling : Applying Constraint Programming to Scheduling Problems*, Kluwer Academic, Boston, **2001**.

Burkard, R.E.; Hatzl, J. Review, extensions and computational comparison of MILP formulations for scheduling of batch processes. *Comput. Chem. Eng*, **2005**, 29, 1752-1769.

Hooker, J. *Logic-Based Methods for Optimization: Combining Optimization and Constraint Satisfaction*, John Wiley & Sons, New York, **2000**.

Hooker, J. Planning and Scheduling by Logic-Based Benders Decomposition, *Op. Res.*, **2007**, 55, 588-602.

Janak, S.L.; Floudas, C.A. Improving unit-specific event based continuous-time approaches for batch processes: Integrality gap and task splitting. *Comput. Chem. Eng*, **2008**, 32, 913-955.

Kondili, E.; Pantelides, C.C.; Sargent, R.W.H. A General Algorithm for Short-Term Scheduling of Batch Operations – I. MILP Formulation. *Comput. Chem. Eng.*, **1993**, 17, 211-227.

Maravelias, C.T. General Framework and Modeling Approach Classification for Production Scheduling. *AIChE Journal*, **2012a**, 58, 1812-1828.

Maravelias, C. T. On the Combinatorial Structure of Discrete-time MIP Formulations for Chemical Production Scheduling. *Computers and Chemical Engineering*, **2012b**, 38, 204-212.

Méndez, C.A.; Cerda, J.; Grossmann, I.E.; Harjunkoski, I.; Fahl, M. State-of-the-art Review of Optimization Methods for Short-term Scheduling of Batch Processes. *Comput. Chem. Eng.*, **2006**, 30, 913-946.

Miller, A.J.; Wolsey, L.A. Tight MIP Formulations for Multi-Item Discrete Lot-Sizing Problems. *Op. Res.*, **2003**, 51, 557-565.

Nemhauser, G.L.; Wolsey, L.A., *Integer and Combinatorial Optimization*, Wiley, New York, **1988**.

Pindo, M. *Planning and Scheduling in Manufacturing Services*, 2nd edition, Springer, New York, **2009**.

Pochet, Y.; Wolsey, L.A. Lot-Sizing with Constant Batches: Formulation and Valid Inequalities. *Math. Oper. Res.*, **1993**, 18, 767-785.

Pochet, Y; Wolsey, L.A.; *Production Planning by Mixed Integer Programming*. Springer, New York, **2006**.

Prasad, P.; Maravelias, C. T. Batch Selection, Assignment and Sequencing in Multistage Processes. *Computers and Chemical Engineering*, **2008**, 32 (6), 1114-1127.

Reklaitis, G. V. *Review of Scheduling of Process Operations*. AIChE Symp. Ser., **1982**, 78, 119-133.

Shah, N.; Pantelides, C.C.; Sargent, R.W.H. A General Algorithm for Short-Term Scheduling of Batch Operations – II. Computational Issues. *Comput. Chem. Eng.*, **1993**, 17, 229-244.

Sundaramoorthy, A.; Maravelias, C. T. A General Framework for Process Scheduling. *AIChE J.*, **2011a**, 57(3), 695-710.

Sundaramoorthy, A.; Maravelias, C. T. Computational Study of Scheduling Approaches for Batch Process Networks. *Industrial and Engineering Chemistry Research*, **2011b**, 50(9), 5023-5040.

van den Akker, J.M.; van Hoesel, C.P.M.; Savelsbergh, M.W.P. A Polyhedral Approach to Single-Machine Scheduling Problems. *Math. Programming*, **1999**, 85, 541-572.

van den Akker, J.M.; Hurkens, C.A.J.; Savelsbergh, M.W.P. Time-Indexed Formulations for Machine Scheduling Problems: Column Generation. *INFORMS Journal on Computing*, **2000**, 12, 111-124.

Van Hentenryck, P.; Michel, L., *Constraint-Based Local Search*, MIT Press, Cambridge, Mass., **2005**.

Wassick, J.M. Enterprise-wide optimization in an integrated chemical complex. *Comput. Chem. Eng.*, **2009**, 33, 1950-1963.

Wolsey, L.A. Valid Inequalities for 0-1 Knapsacks and MIPs with Generalised Upper Bound Constraints. *Discrete Appl. Math.*, **1990**, 29, 251-261.

Wolsey, L.A. MIP Modelling of Changeovers in production Planning and Scheduling Problems. *European Journal of Operational Research*, **1997**, 99, 154-165.

Wolsey, L.A., *Integer Programming*, Wiley, New York, **1998**.