Dynamics of a Finite Buffer Fluid Overflow Jackson Network

Stijn Fleuren

420616

Bachelor Final Project

Fleuren, S. 0630075

Supervisor Prof.dr.ir. Rooda, J.E.
Coaches Dr.ir. Lefeber, A.A.J.
Dr. Nazarathy, Y.

Eindhoven University of Technology
Department of Mechanical Engineering
Systems Engineering Group

Eindhoven, July 4th 2010

Abstract:

This report concerns a finite buffer fluid overflow Jackson network. This network consists of several nodes with buffers in front of them. The fluid that is processed by a node is either routed to another node in the network or leaves the network. If a buffer is full, the fluid that cannot enter this buffer is rerouted as well. Besides the fluid that is routed to a node, the nodes also have an exogenous input.

Very little was known about this kind of networks. Therefore, as a first attempt to gather information for this network, in this report continuous and deterministic fluid flows are concerned. In this report equations that concern the dynamics of the network and the steady state behavior are derived. Further it is shown that for a network of two nodes there is a single steady state solution except for five singular cases where there are either an infinite number of solutions or no solution at all. For general networks, where it is known what nodes are empty and what nodes are full in steady state, it depends on the routing in the network whether there is one or an infinite amount of solutions. It is shown that increasing the exogenous input of a node results in the fact that the fluid that arrives at a node in steady state cannot decrease. Furthermore two algorithms for finding the steady state inputs are given and analyzed. One of those algorithms has an exponential time complexity. The other algorithm has a polynomial time complexity. The efficient algorithm can also be used to solve an equation concerning the dynamics. Therefore, this algorithm can be used to efficiently calculate exact trajectories. Further an interactive graphical representation is made for a network of 3 nodes, a code is given for the efficient algorithm and a program is given that calculates exact trajectories for networks with general network sizes. This trajectory calculating program can be used to gain more insight in the dynamic behavior of the finite buffer fluid overflow Jackson network.

Table of contents

Chapter 1 Introduction

Nowadays a wide variety of factories exist. The main goal of these manufacturing networks is to transform raw fluid into finished products which in turn can be sold. These manufacturing networks can be modeled by using network models. To optimize factories, insight for these network models is needed which leads to the development of new theories and methods. This report is about a finite buffer fluid overflow Jackson network. Because very little was known about this kind of networks, as a first attempt to fully understand these overflow networks stochastic processes are not concerned.

In this chapter, first the notations that are used in this report are shown. Hereafter the parameters that are used in this report are introduced. In the end of this chapter the behavior for a network with network size $N = 1$ is evaluated.

1.1 Notation

Below the notations that are used in this report are shown.

$\delta \wedge \beta = \min(\delta, \beta)$

$\delta^{+} = \max(\delta, 0)$

$\dot{A} = \dfrac{dA(t)}{dt}$

$\bar{\mathbb{A}}(t)$: complement of $\mathbb{A}(t)$ (all i that are not part of subset $\mathbb{A}(t)$ are part of subset $\bar{\mathbb{A}}(t)$)

$\left|\mathbb{A}(t)\right|$: the amount of nodes that are part of subset $\mathbb{A}(t)$

$$1_{\mathbb{A}(t)}: \begin{Bmatrix} 1_{\mathbb{A}(t)}(i) = 1 & \text{for} & i \in \mathbb{A}(t) \\ 1_{\mathbb{A}(t)}(i) = 0 & \text{for} & i \notin \mathbb{A}(t) \end{Bmatrix}$$

$$S_{\mathbb{A}(t)} = Diag(1_{\mathbb{A}(t)}): \begin{Bmatrix} S_{\mathbb{A}(t)}(i,j) = 1_{\mathbb{A}(t)}(i) & \text{for} & i = j \\ S_{\mathbb{A}(t)}(i,j) = 0 & \text{for} & i \neq j \end{Bmatrix}$$

1.2 Network description

A finite buffer fluid overflow Jackson network consists of N nodes, numbered $i = 1, 2, ..., N$. In this report it is used that $\mathbb{N} = \{1, 2, .., N\}$. In this network each node receives an exogenous input at rate $\alpha_i \geq 0$. Furthermore each node has a buffer in front of it. Node i has a capacity K_i and fluid can maximally be processed at rate $\mu_i > 0$. After the fluid is processed by node i a proportion $p_{ij} \geq 0$ of the processed fluid goes to node j. The remaining portion $p_i^D = 1 - \sum_j p_{ij}$, the proportion that is not routed to another node, leaves the network. If a buffer is full, and thus K_i fluid is inside the buffer, it can overflow. In this case a proportion $q_{ij} \geq 0$ of the fluid that cannot enter buffer i (because it is full) is routed to node j. The proportion $q_i^D = 1 - \sum_j q_{ij}$ that is not routed to another node, leaves the network. Below it is shown how these parameters are constructed into matrices and vectors. Also the requirements for these parameters are shown.

$$\alpha = \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_N \end{pmatrix}, \quad \mu = \begin{pmatrix} \mu_1 \\ \vdots \\ \mu_N \end{pmatrix}, \quad P = \begin{pmatrix} p_{11} & \cdots & p_{1N} \\ \vdots & \ddots & \vdots \\ p_{N1} & \cdots & p_{NN} \end{pmatrix}, \quad Q = \begin{pmatrix} q_{11} & \cdots & q_{1N} \\ \vdots & \ddots & \vdots \\ q_{N1} & \cdots & q_{NN} \end{pmatrix}, \quad K = \begin{pmatrix} K_1 \\ \vdots \\ K_N \end{pmatrix},$$

$$\alpha_i \geq 0, \qquad \mu_i > 0, \qquad p_{ii} = 0, \qquad q_{ii} = 0, \qquad K_i > 0,$$

$$\sum_j p_{ij} \leq 1, \qquad \sum_j q_{ij} \leq 1,$$

$$p_{ij} \geq 0, \qquad q_{ij} \geq 0,$$

$$p_i^D = 1 - \sum_j p_{ij}, \qquad q_i^D = 1 - \sum_j q_{ij}.$$

Furthermore $X_i(t)$ denotes the amount of fluid inside buffer i at time t and X_i^o is used for the amount of fluid inside buffer i at time $t = 0$. The variable $\lambda_i(t)$ is used for the total amount of fluid per time unit that arrives at node i. The vector $X(t)$ and $\lambda(t)$ is constructed as follows.

$$X(t) = \begin{pmatrix} X_1(t) \\ \vdots \\ X_N(t) \end{pmatrix}, \quad \lambda(t) = \begin{pmatrix} \lambda_1(t) \\ \vdots \\ \lambda_N(t) \end{pmatrix},$$

$$0 \leq X_i(t) \leq K_i.$$

1.3 Behavior for *N=1*

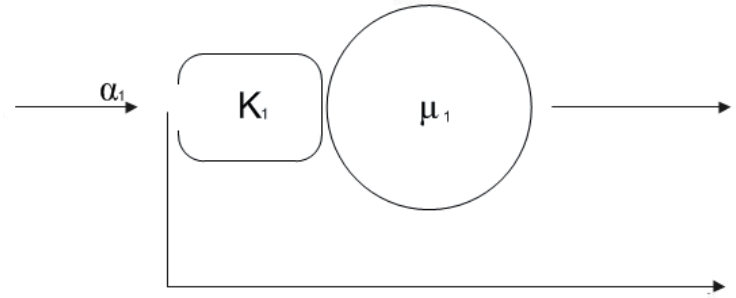

Figure 1.1: visualization of a finite buffer fluid overflow Jackson network consisting of one node

For a finite buffer fluid overflow Jackson network consisting of one node 3 cases exist. The first case is where $\alpha_1 > \mu_1$. In this case more fluid per time unit arrives at the node then what the node can maximally process. This causes the buffer to fill up until it reaches its full buffer capacity. The second case is where $\alpha_1 < \mu_1$. In this case the node can process more fluid per second than the amount of fluid that arrives at the node. This causes the buffer to become empty. The third case is where $\alpha_1 = \mu_1$. In this case the amount of fluid that arrives at the node is equal to the fluid processed by the nodes with as a result $X(t) = X(0) = X_0$. This third case is not analyzed.

Proposition:
For a finite buffer fluid overflow Jackson network with one node that satisfies the requirements shown on the previous page.

$$X_1(t) = \begin{Bmatrix} X_1^0 + (\alpha_1 - \mu_1)\cdot t & \text{for} & t \le (K_1 - X_1^0)/(\alpha_1 - \mu_1) & ,\alpha_1 > \mu_1 \\ K_1 & \text{for} & t > (K_1 - X_1^0)/(\alpha_1 - \mu_1) & " \\ X_1^0 - (\mu_1 - \alpha_1)\cdot t & \text{for} & t \le X_1^0/(\mu_1 - \alpha_1) & ,\alpha_1 < \mu_1 \\ 0 & \text{for} & t > X_1^0/(\mu_1 - \alpha_1) & " \end{Bmatrix}$$

When for all buffers the amount of fluid inside the buffer does not change over time, then the network is said to be in steady state. Therefore, for $\alpha_1 > \mu_1$ and $\alpha_1 < \mu_1$ a one node finite buffer fluid overflow Jackson network is in steady state for respectively $t > (K_1 - X_1^0)/(\alpha_1 - \mu_1)$ and $t > X_1^0/(\mu_1 - \alpha_1)$.

Proof:

$\alpha_1 > \mu_1$:

In this case the node always processes μ_1 fluid per time unit. If the buffer has not reached its full capacity K_1 then α_1 fluid per time unit enters the buffer in the same time that μ_1 fluid per time unit is processed. Therefore, net $\alpha_1 - \mu_1$ fluid enters the buffer per time unit. This causes $X_1(t)$ to have a slope of $\alpha_1 - \mu_1$. In this stage no fluid overflows.

If the buffer has reached its full capacity K_1, still α_1 fluid per time unit arrives and μ_1 fluid per time unit is processed by the node. However the remaining fluid cannot fill up the buffer anymore. Instead this fluid overflows and leaves the system.

The breaking point between those 2 stages can easily be determined. In the first stage $X_1(t)$ has a slope of $\alpha_1 - \mu_1$ and thus $X_1(t) = X_1^0 + (\alpha_1 - \mu_1) \cdot t$. In the second stage $X_1(t) = K_1$ and therefore the transition happens when $X_1^0 + (\alpha_1 - \mu_1) \cdot t = K_1$ or $t = (K_1 - X_1^0) / (\alpha_1 - \mu_1)$.

Numerical example:

Arrival rate	Processing rate	Buffer capacity	Starting value
$\alpha_1 = 1$	$\mu_1 = 0.5$	$K_1 = 10$	$X_1^0 = 5$

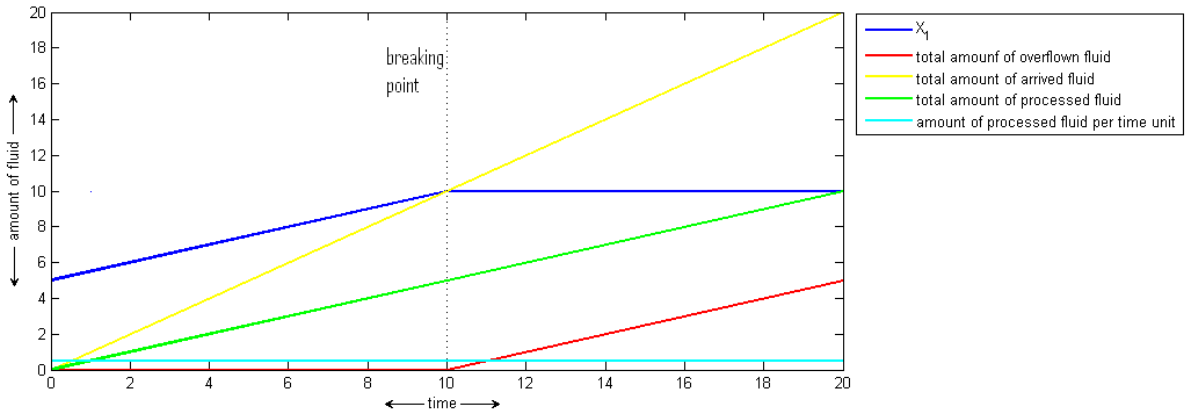

Figure 1.2: results for a one node finite buffer fluid overflow Jackson network for $\alpha_1 > \mu_1$

$\alpha_1 < \mu_1$:

In this case the buffer is never full. Hence, no fluid overflows. Before the buffer is empty the node processes μ_1 fluid per time unit. at all times, α_1 fluid arrives and therefore net $\mu_1 - \alpha_1$ fluid leaves the buffer. As a result $X_1(t)$ has a slope of $\alpha_1 - \mu_1$ until the buffer is empty. Hence, $X_1 = X_1^0 - (\mu_1 - \alpha_1) \cdot t$. The buffer is empty when X_1^0 fluid has left the buffer. Therefore, the breaking point occurs at $t = X_1^0 / (\mu_1 - \alpha_1)$. If the buffer is empty all the fluid that arrives is immediately processed.

Numerical example

Arrival rate	Processing rate	Buffer capacity	Starting value
$\alpha_1 = 0.5$	$\mu_1 = 1$	$K_1 = 10$	$X_1^0 = 5$

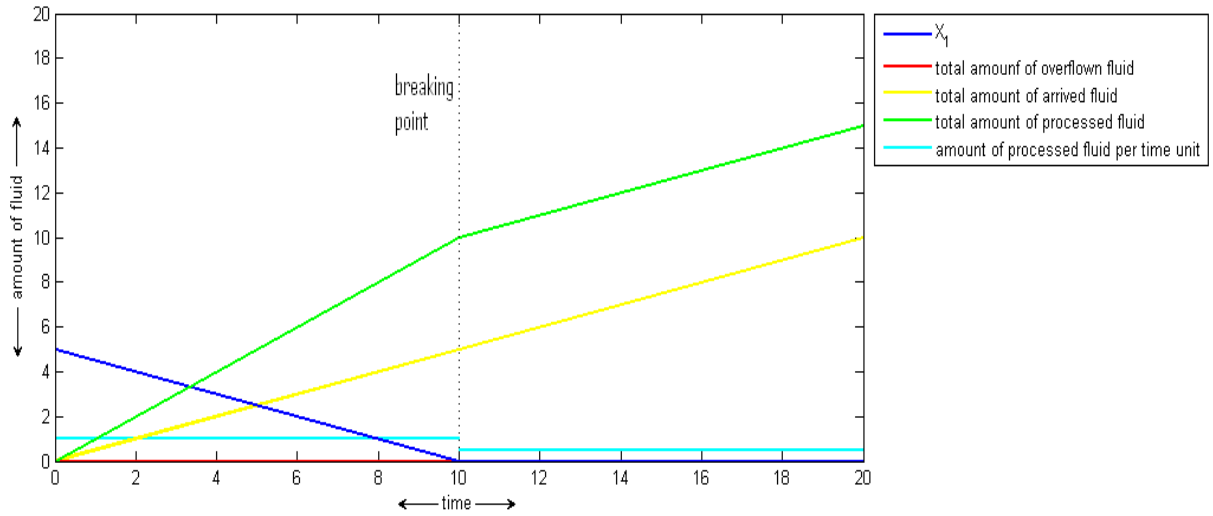

Figure 1.3: results for a one node finite buffer fluid overflow Jackson network for $\alpha_1 < \mu_1$

1.4 Overview of the results

In this section all the results obtained throughout this report are presented per chapter.

Chapter 2:

In chapter two it is explained and graphically shown that for a network of 2 nodes there is always a single solution for steady state inputs λ, except for 5 singular cases.

Chapter 3:

In this chapter it is explained that the dynamics of a finite buffer fluid overflow Jackson network are defined by the following equations:

$$\lambda(t) = \alpha + (S_{\bar{\mathbb{E}}(t)} P)' \mu + (S_{\mathbb{E}(t)} S_{\mathbb{G}(t)} P)' \lambda(t) + (S_{\mathbb{E}(t)} S_{\bar{\mathbb{G}}(t)} P)' \mu + (S_{\mathbb{F}(t)} S_{\bar{\mathbb{G}}(t)} Q)' (\lambda(t) - \mu)$$

$$\dot{X}(t) = (I - S_{\mathbb{G}(t)} S_{\mathbb{E}(t)} - S_{\bar{\mathbb{G}}(t)} S_{\mathbb{F}(t)})(\lambda - \mu)$$

The definitions for $\mathbb{E}(t)$, $\mathbb{F}(t)$ and $\mathbb{G}(t)$ can be found in section 3.1. Besides it is shown that, if it is known what nodes are empty and what nodes are full in steady state, the steady state inputs λ can be calculated with the equation shown below. To obtain this equation the assumption that all

buffers are either empty or full in steady state is used.

$$\lambda = \alpha + (S_{\mathbb{E}}P)'\lambda + (S_{\bar{\mathbb{E}}}P)'\mu + (S_{\bar{\mathbb{E}}}Q)'(\lambda - \mu)$$

Furthermore it is shown that there is a single solution for steady state inputs λ, for known steady state empty and steady state full nodes, if $\det(I - S_{\mathbb{E}}P - S_{\bar{\mathbb{E}}}Q) \neq 0$. Further it is shown the steady state inputs λ are monotonic in α_i. This means that increasing the exogenous input α_i of node i cannot result into the fact that the steady state inputs λ decrease.

Chapter 4:

In this chapter two different algorithms that can find the nodes that are empty and full in steady state and the steady state inputs λ are shown. The brute force algorithm has an average-case and a worst-case time complexity of $O(N^3 2^N)$. Which means that the average time needed and the maximal time that can be needed to find the solution increases exponentially with increasing network size N. The other algorithm is more efficient for networks bigger than 2 nodes and has a polynomial time complexity. The worst-case complexity of the algorithm is $O(N^5)$. However the average-case time complexity of the algorithm is probably $O(N^3 \log(N))$. This average-case time complexity is obtained for networks that are generated with the Mathematica code in Figure 4.8.

Chapter 5:

In chapter five it is shown that the efficient algorithm can also be used to calculate exact trajectories. Two programs are given that can calculate the exact trajectories. One of these programs is an interactive graphical demonstration of a finite buffer fluid overflow Jackson network consisting of 3 nodes. The other program can calculate and graphically show exact trajectories for all network sizes. With this last program it is observed that:

- There is no cyclic behavior: There is no buffer that switches between being empty and full until infinity.
- Buffers can switch between being empty and full before steady state is reached.
- If the buffer capacities and the initial amount of fluid inside the buffer increase with a factor f and the other conditions stay the same, the time at which a certain buffer becomes full or empty increases with a factor f as well.
- In steady state a buffer is either empty or full. Buffer i can only be neither full nor empty in steady state if the input λ_i is exactly the same as the maximal processing rate μ_i. This is very unlikely to happen for the random matrices and random vectors the program is tested for.

Chapter 2 network of 2 nodes

In steady state if node i has an input λ_i smaller than its maximal processing rate μ_i, this node routes $p_{ij}\lambda_i$ fluid per time unit to node j. However if in steady state node i has an input λ_i greater than its maximal processing rate μ_i, this node processes fluid at rate μ_i and fluid overflows at rate $(\lambda_i - \mu_i)$. Therefore, in that case $p_{ij}\mu_i + (\lambda_i - \mu_i)q_{ij}$ fluid per time unit is send from node i to node j. Thus, the following equations can be found for the input rates of the two nodes:

$\lambda_1 = \alpha_1 + \lambda_2 p_{21}$ for $\lambda_2 \le \mu_2$ (2.1)

$\lambda_2 = \alpha_2 + \lambda_1 p_{12}$ for $\lambda_1 \le \mu_1$ (2.2)

$\lambda_1 = \alpha_1 + \mu_2 p_{21} + (\lambda_2 - \mu_2)q_{21}$ for $\lambda_2 \ge \mu_2$ (2.3)

$\lambda_2 = \alpha_2 + \mu_1 p_{12} + (\lambda_1 - \mu_1)q_{12}$ for $\lambda_1 \ge \mu_1$ (2.4)

Lemma: There is one solution for the steady state inputs λ for a 2 node finite buffer fluid overflow Jackson network, except for 5 singular cases.

Proof:

For a finite buffer fluid overflow Jackson network of 2 nodes, λ_1 is plotted as function of λ_2 according to (2.1) and (2.3) and λ_2 is plotted as function of λ_1 according to (2.2) and (2.4). The result is shown in Figure 2.1 (see page 12). According to (2.1) the slope $d\lambda_1/d\lambda_2$ is equal to p_{12} for $\lambda_2 \le \mu_2$. As a result, in this figure θ_1 is equal to $\tan^{-1}(1/p_{21})$ for $p_{21} \neq 0$ and equal to 90^o for $p_{21} = 0$.
In the same way it is obtained that:

$$\theta_1 = \begin{Bmatrix} \tan^{-1}(1/p_{21}) & \text{for} & p_{21} \neq 0 \\ 90^o & \text{for} & p_{21} = 0 \end{Bmatrix}, \theta_2 = \begin{Bmatrix} \tan^{-1}(1/p_{12}) & \text{for} & p_{12} \neq 0 \\ 90^o & \text{for} & p_{12} = 0 \end{Bmatrix},$$

$$\phi_1 = \begin{Bmatrix} \tan^{-1}(1/q_{21}) & \text{for} & q_{21} \neq 0 \\ 90^o & \text{for} & q_{21} = 0 \end{Bmatrix}, \phi_2 = \begin{Bmatrix} \tan^{-1}(1/q_{21}) & \text{for} & q_{21} \neq 0 \\ 90^o & \text{for} & q_{21} = 0 \end{Bmatrix}.$$

Because $0 \le p_{ij} \le 1$ and $0 \le q_{ij} \le 1$ the angles $\theta_1, \theta_2, \phi_1$ and ϕ_2 are between 45° and 90°. These boundaries are visualized in the figure as gray dotted lines. It can be seen that because of these boundaries the lope $d\lambda_1/d\lambda_2$ of the line $\lambda_1(\lambda_2)$ is always bigger or equal to the slope $d\lambda_1/d\lambda_2$ of the line $\lambda_2(\lambda_1)$. This is the reason why the two lines cannot have 2 intersections. There can solely be more than one solution if the two lines coincide in at least one of the four regions. In general steady state can only be achieved when the total output of the network is equal to the total input of the network. The total output of the network is equal to $\sum_{i=1}^{N} p_i^D(\lambda_i \wedge \mu_i) + \sum_{i=1}^{N} q_i^D(\lambda_i - \mu_i)^+$ and the total input is equal to $\sum_{i=1}^{N} \alpha_i$. Therefore, to have an infinite number of solutions in steady state, the total input into the network must be equal to the total output of the network and the two lines must be parallel. For each region there is a single case when this occures.

There is also a singular case when there is no solution at all. In that case there is no solution in regions I, II_a and II_b and in region III the two lines are parallel but do not coincide. In this case both buffers fill up and the inputs λ increase until infinity. The two lines are parallel in region III if $q_{12} = q_{21} = 1$ and the two lines do not coincide if the total input into the network is not equal to the total output of the network. An example of such a network, where no solution for steady state inputs λ excists, is a network that has an input while no fluid can leave the network ($p_{12} = p_{21} = q_{12} = q_{21} = 1$ and $\alpha_1 + \alpha_2 \neq 0$). This causes the inputs λ to increase until infinity. All 5 singular cases are shown in Table 2.1. For all other cases there is only one solution.

Region of solutions	Condition	Amount of solutions
I	$p_{12} = p_{21} = 1$ and $\alpha_1 + \alpha_2 = 0$	∞
III	$q_{12} = q_{21} = 1$ and $\alpha_1 + \alpha_2 = (1 - p_{12})\mu_1 + (1 - p_{21})\mu_2$	∞
II_b	$p_{12} = q_{21}$ and $(1 - p_{21})\mu_2 = \alpha_1 + \alpha_2$	∞
II_a	$p_{21} = q_{12}$ and $(1 - p_{12})\mu_1 = \alpha_1 + \alpha_2$	∞
-	$q_{12} = q_{21} = 1$ and $\alpha_1 + \alpha_2 \neq (1 - p_{12})\mu_1 + (1 - p_{21})\mu_2$ and there are no solutions in regions I, II_a and II_b	0

Table 2.1 the five singularities for a finite buffer fluid overflow Jackson network of 2 nodes

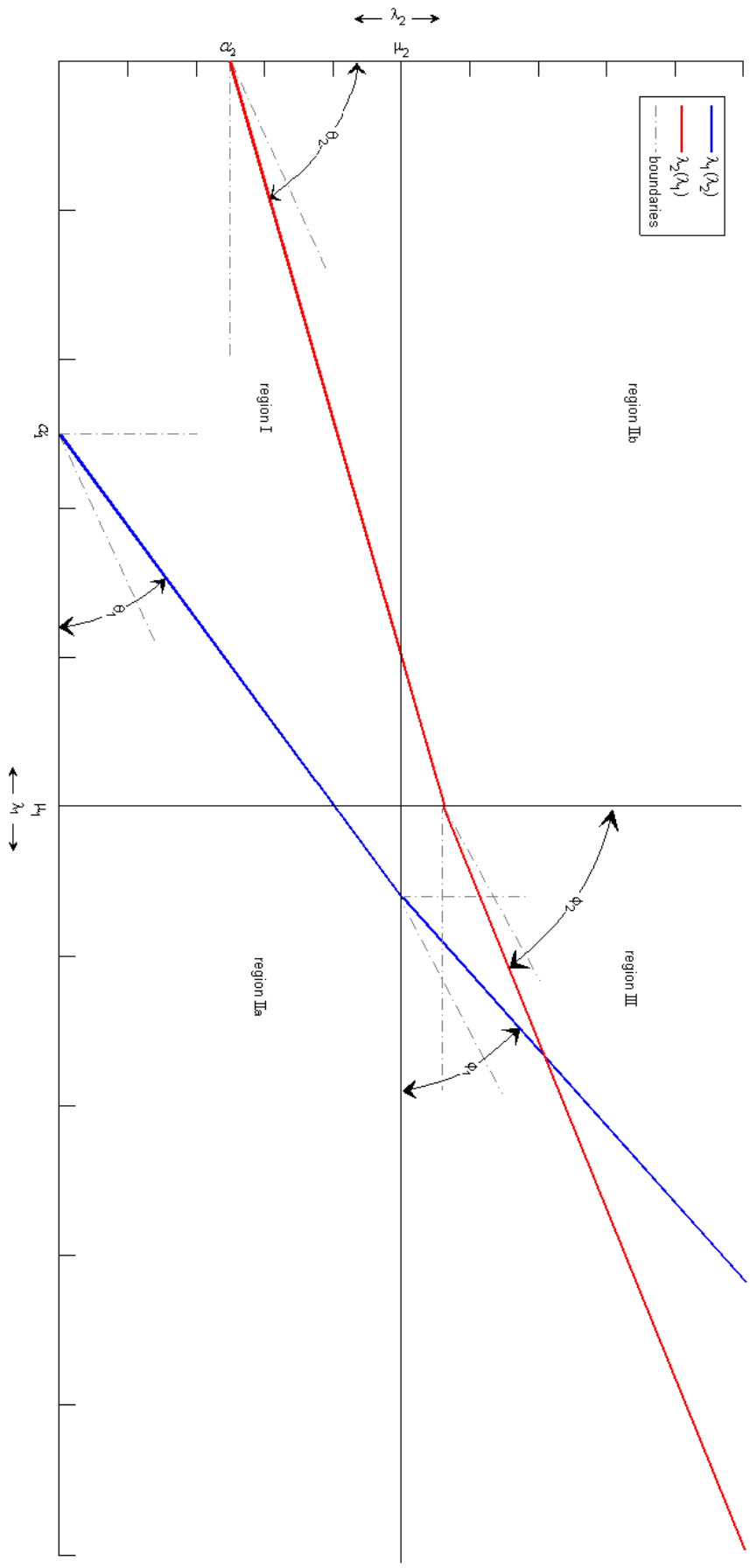

Figure 2.1 plotting $\lambda_1(\lambda_2)$ and $\lambda_2(\lambda_1)$

Chapter 3 A related non-linear system of equations

In this chapter first a few equations that concern the dynamic behavior of a finite buffer fluid overflow Jackson network of N nodes are given. Also an equation is given with which the steady state inputs λ can be calculated if it is known what nodes are empty and what nodes are full in steady state. Hereafter it is proven that the steady state inputs λ are monotonic in α_i and that there is only one steady state solution λ, for known steady state empty and full nodes, if $\det(I - S_{\mathbb{E}} P - S_{\bar{\mathbb{E}}} Q) \neq 0$. The definition of $\mathbb{E}$ is shown in the next section.

3.1 Dynamics of a N -node finite buffer fluid overflow Jackson network

Before a general system of equations can be found for the dynamics of a N -node finite buffer fluid overflow Jackson network it is useful to introduce some notation. The buffers are grouped by $\mathbb{F}(t) = \{i \mid X_i(t) = K_i\}$ and $\mathbb{E}(t) = \{i \mid X_i(t) = 0\}$. In other words the collection of full buffers at time t is referred to as subset $\mathbb{F}(t)$ and the collection of empty buffers at time t is referred to as subset $\mathbb{E}(t)$. If $X_i(t) = K_i$ (and thus if node i belongs to subset $\mathbb{F}(t)$) then the buffer does overflow for $\mu_i < \lambda_i(t)$, the amount of overflown fluid is equal to $(\lambda_i(t) - \mu_i)^+$ and the net input is equal to $\dot{X}_i(t) = (\lambda_i(t) - \mu_i) \wedge 0$. On the other hand if $X_i(t) = 0$ (and thus if node i belongs to subset $\mathbb{E}(t)$) then the output of the buffer is $(\lambda_i(t) \wedge \mu_i)$ and the net input of the buffer is equal to $\dot{X}_i(t) = (\lambda_i(t) - \mu_i)^+$. The remaining buffers, i.e. the buffers that satisfy $0 < X_i(t) < K_i$, have an output of μ_i and a net input of $\dot{X}_i(t) = (\lambda_i(t) - \mu_i)$. Now a general system of equations for a finite buffer fluid overflow Jackson network of N nodes can be given:

$$\lambda_i(t) = \alpha_i + \sum_{j \in \mathbb{E}(t)} p_{ji}(\lambda_j(t) \wedge \mu_j) + \sum_{j \notin \mathbb{E}(t)} p_{ji}\mu_j + \sum_{j \in \mathbb{F}(t)} q_{ji}(\lambda_j(t) - \mu_j)^+ \quad \text{for } i = 1, 2.., N \quad (3.1)$$

$$\dot{X}_i(t) = (\lambda_i(t) - \mu_i) \wedge 0 \qquad \text{for } i \in \mathbb{F}(t) \qquad (3.2)$$

$$\dot{X}_i(t) = (\lambda_i(t) - \mu_i)^+ \qquad \text{for } i \in \mathbb{E}(t) \qquad (3.3)$$

$$\dot{X}_i(t) = (\lambda_i(t) - \mu_i) \qquad \text{for } i \in (\bar{\mathbb{F}}(t) \cap \bar{\mathbb{E}}(t)) \qquad (3.4)$$

Equation 3.1 can also be written in matrix form, for this purpose selection matrices $S_{\bar{\mathbb{E}}(t)}$, $S_{\mathbb{E}(t)}$, and $S_{\mathbb{F}(t)}$ are introduced. Now (3.1) can be written as follows:

$$\lambda(t) = \alpha + (S_{\bar{\mathbb{E}}(t)} P)' \mu + (S_{\mathbb{E}(t)} P)'(\lambda(t) \wedge \mu) + (S_{\mathbb{F}(t)} Q)'(\lambda(t) - \mu)^+ \qquad (3.5)$$

Another subset is introduced: $\mathbb{G}(t) = \{i \mid \lambda_i(t) < \mu_i\}$. This means that all nodes that are underloaded are part of subset $\mathbb{G}(t)$. Furthermore selection matrix $S_{\mathbb{G}(t)}$ is used.

Now (3.5) can be written as:

$$\lambda(t) = \alpha + (S_{\bar{\mathbb{E}}(t)}P)'\mu + (S_{\mathbb{E}(t)}S_{\mathbb{G}(t)}P)'\lambda(t) + (S_{\mathbb{E}(t)}S_{\bar{\mathbb{G}}(t)}P)'\mu + (S_{\mathbb{F}(t)}S_{\bar{\mathbb{G}}(t)}Q)'(\lambda(t) - \mu) \quad (3.6)$$

By using the selection matrices $S_{\mathbb{E}(t)}$, $S_{\mathbb{F}(t)}$ and $S_{\mathbb{G}(t)}$ equations (3.2),(3.3) and (3.4) can also be written in matrix form. The result is:

$$\dot{X}(t) = (I - S_{\mathbb{G}(t)}S_{\mathbb{E}(t)} - S_{\bar{\mathbb{G}}(t)}S_{\mathbb{F}(t)})(\lambda - \mu) \quad (3.7)$$

This means that all nodes have an output of $(\lambda_i(t) - \mu_i)$ except for the empty underloaded nodes and the full overloaded nodes. These empty underloaded and full overloaded nodes have a net input of zero. The inputs λ that are used in this equation can be calculated with (3.6).

3.2 Assumptions on the network

The assumptions shown below are referred to in this report. From now on it is used that $\mathbb{E}$ is equal to $\mathbb{E}(t)$ in steady state en $\mathbb{F}$ is equal to $\mathbb{F}(t)$ in steady state.

assumption 1:
a unique solution for $\mathbb{E}$ exists.

This assumption is believed to be true for the finite buffer fluid overflow Jackson network. However this assumption is not yet shown mathematically

assumption 2:
In steady state $|\mathbb{E}| + |\mathbb{F}| = N$. This means that in steady state a buffer is either full or empty and not somewhere in between.

This assumption can only be not satisfied if the input of a node is exactly the same as the maximal output of this node. In that case the buffer can be neither full nor empty. A network does not satisfy this assumptions is the network shown in Figure 3.1 if $\alpha_1 > \mu_1$ and $\mu_2 = \mu_1 + \alpha_2$. In this case buffer 2 can be neither full nor empty.

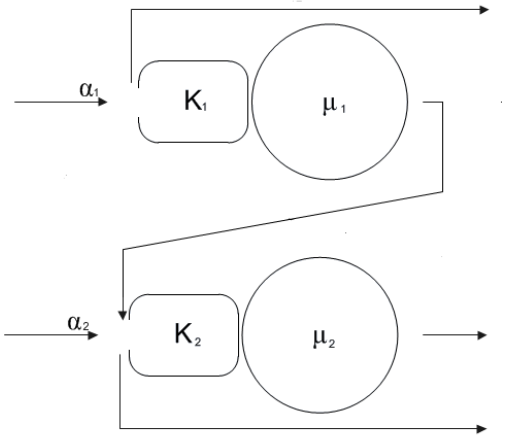

Figure 3.1 network of 2 nodes that does not satisfy assumption 2 if $\alpha_1 > \mu_1$ and $\mu_2 = \mu_1 + \alpha_2$

3.3 Calculating steady state inputs λ for known subset $\mathbb{E}$

In steady state all subsets are no longer a function of time and if assumption 2 (see section 3.2) is met then $\mathbb{E} = \mathbb{G}$ and $\mathbb{F} = \overline{\mathbb{E}} = \overline{\mathbb{G}}$ in steady state. This because if a buffer is empty but overloaded, the buffer would fill up and steady state has not yet been reached. In the same way if a buffer is full but underloaded the buffer would have a net output and steady state has not yet been reached as well. By using $S_{\mathbb{E}} = S_{\mathbb{G}}$ and $S_{\mathbb{F}} = S_{\overline{\mathbb{E}}} = S_{\overline{\mathbb{G}}}$ (3.5) and (3.6) can be reduced to:

$$\lambda = \alpha + P'(\lambda \wedge \mu) + Q'(\lambda - \mu)^{+} \tag{3.8}$$

$$\lambda = \alpha + (S_{\mathbb{E}}P)'\lambda + (S_{\overline{\mathbb{E}}}P)'\mu + (S_{\overline{\mathbb{E}}}Q)'(\lambda - \mu) \tag{3.9}$$

Substituting $S_{\mathbb{E}} = S_{\mathbb{G}}$ and $S_{\mathbb{F}} = S_{\overline{\mathbb{E}}} = S_{\overline{\mathbb{G}}}$ into (3.7) results in $\dot{X} = 0$. What indeed means that the network is in steady state.

3.4 Proving that the steady state inputs λ are monotonic in α_i

Lemma: *the steady state inputs λ are monotonic in α_i* *(assumption 2 is used)*

Proof:

To prove this lemma, steady state equation 3.9 is used. Therefore, for this proof assumption 2 is used. If it is assumed that the matrix $(I - (S_{\mathbb{E}}P)' - (S_{\overline{\mathbb{E}}}Q)')^{-1}$ does exist, the solution of this steady state equation is:

$$\lambda = \underbrace{(I - (S_{\mathbb{E}}P)' - (S_{\overline{\mathbb{E}}}Q)')^{-1}}_{\text{matrix}} \underbrace{\left(\alpha + (S_{\overline{\mathbb{E}}}P)'\mu - (S_{\overline{\mathbb{E}}}Q)'\mu\right)}_{\text{constant}}$$

The inverse $(I - A)^{-1}$ can be written as $I + A + A^2 + A^3 + A^4 + \ldots$. Hence, because all entries of $S_{\mathbb{E}}P + S_{\overline{\mathbb{E}}}Q$ are non-negative numbers the (j,i) entry of $(I - (S_{\mathbb{E}}P)' - (S_{\overline{\mathbb{E}}}Q)')^{-1}$ is non-negative and the (i,i) entry of $(I - (S_{\mathbb{E}}P)' - (S_{\overline{\mathbb{E}}}Q)')^{-1}$ is positive. Therefore, the steady state input λ_i rises whenever α_i rises and the steady state input λ_j increases if α_i rises whenever there is a positive integer m such that the (i, j) entry of $(S_{\mathbb{E}}P_{\mathbb{E}} + S_{\overline{\mathbb{E}}}Q_{\mathbb{F}})^{(m)}$ is greater than zero. If the (i, j) entry of $(S_{\mathbb{E}}P + S_{\overline{\mathbb{E}}}Q)^{(m)}$ is greater than zero for some positive integer m then j is said to be accessible from node i. This means that there is a route in the matrix $S_{\mathbb{E}}P + S_{\overline{\mathbb{E}}}Q$ from node i to node j and therefore node j senses an increasing input λ_i. As a result the (j,i) entry of $(I - (S_{\mathbb{E}}P)' - (S_{\overline{\mathbb{E}}}Q)')^{-1}$ is greater than zero if node j is accessible from node i. The nodes $j \neq i$ that are not accessible from node i are not influenced by increasing α_i. Notice that in this meaning of accessibility fluid can still go from node i to node j if node j is not accessible from node i, it just means that increasing α_i is not noticeable in node j.

If $\lambda_j < \mu_j$ then buffer j is empty in steady state. But at some point raising α_i can result into the fact that $\lambda_j > \mu_j$. For $\lambda_j > \mu_j$ buffer j is full in steady state. The transition between an empty buffer and a full buffer j in steady state takes place when $\lambda_j = \mu_j$. For this transition $p_{jk}\mu_k$ fluid goes from node j to node k, independently of whether buffer j is full, empty or somewhere in between. Therefore, if the number of nodes in subset $\mathbb{E}$ decreases, then still increasing α_i results into an increasing input λ_i and also into an increasing input λ_j whenever node j is accessible from node i.

Nevertheless if subset $\mathbb{E}$ changes, then the matrix $S_{\mathbb{E}}P + S_{\bar{\mathbb{E}}}Q$ changes and therefore other nodes might be accessible from node i.

In conclusion, rising α_i causes λ_j to rise or stay the same and therefore the steady state inputs λ are monotonic in α_i. This because increasing α_i causes λ_i and the number of steady state full buffers to rise. The input λ_j only rises whenever node j is accessible from node i. If the number of nodes in subset $\mathbb{E}$ changes, still raising α_i causes λ_i to go up and causes λ_j to increase whenever node j is accessible from node i. The nodes that are accessible from node i might be different if subset $\mathbb{E}$ changes.

3.5 Proving that there is one steady state solution λ for known steady state subset $\mathbb{E}$

Lemma: *There is one steady state solution for λ, for given steady state subset $\mathbb{E}$, if* $\det(I - S_{\mathbb{E}}P - S_{\bar{\mathbb{E}}}Q) \neq 0$ *(assumption 2 is used)*

Proof:

The solution λ can be calculated by using (3.9) and thus for this proof assumption 2 is used. If assumed that the matrix $(I - (S_{\mathbb{E}}P)' - (S_{\bar{\mathbb{E}}}Q)')^{-1}$ does exist, the solution to (3.9) is:

$$\lambda = \underbrace{(I - (S_{\mathbb{E}}P)' - (S_{\bar{\mathbb{E}}}Q)')^{-1}}_{\text{matrix}} \underbrace{\left(\alpha + (S_{\bar{\mathbb{E}}}P)'\mu - (S_{\bar{\mathbb{E}}}Q)'\mu\right)}_{\text{constant}}.$$

This system of equations can only be solved if the matrix $(I - (S_{\mathbb{E}}P)' - (S_{\bar{\mathbb{E}}}Q)')$ is invertible. A matrix can be inverted if it has a determinant unequal to zero, and therefore if the matrix is nonsingular.

In the case where the matrix $(I - (S_{\mathbb{E}}P)' - (S_{\bar{\mathbb{E}}}Q)')$ is singular, solving (3.9) results in $\lambda_i = C_i + \lambda_i$. The reason for this singularity is that if node i is either empty or full the same amount of fluid that respectively arrives or overflows at node i eventually is routed back to node i according to matrix $S_{\mathbb{E}}P + S_{\bar{\mathbb{E}}}Q$. Therefore, for all possible routes from node i back to node i in matrix $S_{\mathbb{E}}P + S_{\bar{\mathbb{E}}}Q$, no fluid leaves the network. Thus, the sum of the j th row of $S_{\mathbb{E}}P + S_{\bar{\mathbb{E}}}Q$ is equal to one for all nodes j that node i can access (including itself). Therefore, there is a sub network consisting of node i and the nodes that node i can access. In this sub network all empty and full nodes route respectively all the fluid that is processed and all fluid that overflows to other nodes in this sub network. There are an infinite number of solutions in this sub network if the total input into this sub network is equal to the total output of this sub network. In that case $C_i = 0$. On the other hand if the total input into this sub network is not equal to the total output of this sub network then $C_i \neq 0$ and there are no solutions for λ at all. Therefore, there is a single steady state solution if $\det(I - (S_{\mathbb{E}}P)' - (S_{\bar{\mathbb{E}}}Q)') = \det(I - S_{\mathbb{E}}P - S_{\bar{\mathbb{E}}}Q) \neq 0$ (because $\det(A') = \det(A)$)) and there are an infinite number of steady state solutions or no steady state solutions if $\det(I - S_{\mathbb{E}}P - S_{\bar{\mathbb{E}}}Q) = 0$.

In chapter 2 it is shown that there is a single solution for steady state inputs λ for a finite buffer fluid overflow Jackson network of 2 nodes, except for 5 singular cases. The 4 singular cases where there are an infinite number of solutions are exactly the four possible cases where $\det(I - S_{\mathrm{E}}P - S_{\bar{\mathrm{E}}}Q) \neq 0$ and $C_i = 0$. For example in the case where $p_{12} = p_{21} = 1$ and $\alpha_1 + \alpha_2 = 0$ there are an infinite number of solutions in region I (see Figure 2.1) and therefore both buffers are empty in steady state. In this case S_{E} is equal to I and therefore $\det(I - S_{\mathrm{E}}P - S_{\bar{\mathrm{E}}}Q) = \det(I - P) = 0$. In this case the total input of the network is equal to the total output of the network which results in $C_i = 0$. In the case where there are no solutions at all, both buffers fill up and the inputs λ will increase until infinity because $q_{12} = q_{21} = 1$. Therefore, in steady state both buffers are full and $S_{\bar{\mathrm{E}}}$ is equal to I. Hence, $\det(I - S_{\mathrm{E}}P - S_{\bar{\mathrm{E}}}Q) = \det(I - Q) = 0$. In this case the total input into the network is not equal to the total output out of the network which results in $C_i \neq 0$.

If assumption 1 is indeed true, then all finite buffer fluid overflow Jackson networks, where no sub networks can occur wherein all empty and full nodes route respectively all the fluid that is processed and all fluid that overflows to other nodes in this sub network, have a single solution for steady state inputs λ.

Chapter 4 Algorithms to solve the non-linear system of equations

In this chapter two algorithms are explained that can find the steady state subsets $\mathbb{E} = \{i \mid \lambda_i < \mu_i\}$, $\mathbb{F} = \{i \mid \lambda_i < \mu_i\}$ and $\mathbb{U} = \{i \mid \lambda_i = \mu_i\}$. Here the nodes that are part of subset $\mathbb{E}$ are empty in steady state, the nodes that are part of subset $\mathbb{F}$ are full in steady state and for the nodes that are part of subset $\mathbb{U}$ it is unknown what amount of fluid is inside the buffer in steady state. In this chapter the subset $\mathbb{G}$ has a slightly different meaning than it has in the previous chapter. In this chapter $\mathbb{G} = \{i \mid \lambda_i \leq \mu_i\}$ and node i is said to be underloaded if $\lambda_i \leq \mu_i$ as opposed to $\lambda_i < \mu_i$ used in the previous chapter. The first algorithm that can find the steady state subsets $\mathbb{E}$, $\mathbb{F}$ and $\mathbb{U}$, calculates the steady state inputs λ for each combination of nodes in subset $\mathbb{G} = \{i \mid \lambda_i \leq \mu_i\}$ until the solution is found. The second algorithm finds steady state subsets in a more efficient manner. For both algorithms a Mathematica script is made so that the outcomes can be compared (see next chapter).

4.1 Assumptions on the input data for the two algorithms

These assumption are made on the input data for the two algorithms. These assumptions are referred to in this chapter.

assumption 3: $\det(I - S_{\mathbb{G}}P - S_{\bar{\mathbb{G}}}Q) \neq 0$ for $\forall \mathbb{G} \subseteq \mathbb{N}$ If this assumption is satisfied then it is guaranteed that a solution is found with brute force (see section 4.2).
assumption 4: $\det(I - S_{\mathbb{G}}P - S_{\mathbb{B}}Q) \neq 0$ for $\forall \mathbb{G}, \mathbb{B} \subseteq \mathbb{N}$. The meaning of subset $\mathbb{B}$ is explained in section 4.3. If this assumption is satisfied then it is satisfied that the efficient algorithm (see section 4.3) finds the solution.

4.2 Algorithm one: brute force

This algorithm finds the solution if assumption 1 and assumption 3 are satisfied. If assumption 3 is satisfied all inverses that can be needed with this algorithm are computable. In this algorithm for each combination of nodes in the steady state subset $\mathbb{G}$, the steady state inputs λ are calculated according to (3.9):

$$\lambda = \alpha + (S_{\mathbb{G}}P)'\lambda + (S_{\bar{\mathbb{G}}}P)'\mu + (S_{\bar{\mathbb{G}}}Q)'(\lambda - \mu)$$

If all nodes that where assumed to be in steady state subset $\mathbb{G}$ satisfy $\lambda_i \leq \mu_i$ and all other nodes satisfy $\lambda_i > \mu_i$ then this subset $\mathbb{G}$ is indeed the steady state subset and it can be determined whether the nodes in subset $\mathbb{G}$ are either in subset $\mathbb{E}$ or in subset $\mathbb{U}$. If a solution is found the algorithm is done. In appendix A a Mathematica script for brute force is shown.

Maximal number of iterations:

In the worst case a maximal number of combinations 2^N for subset $\mathbb{G}$ are verified and therefore λ can be calculated a maximum of 2^N times before the true solution is found. Because of this exponential increase this algorithm is inefficient.

Remark:
To guarantee that a solution is found for (3.9), λ must be computable for every combination of nodes in subset $\mathbb{G}$. To calculate λ for some combination of nodes in subset $\mathbb{G}$, the matrix $(I-(S_{\mathbb{G}}P)'-(S_{\bar{\mathbb{G}}}Q)')$ has to be inverted and therefore $\det(I-S_{\mathbb{G}}P-S_{\bar{\mathbb{G}}}Q)\neq 0$ has to hold. Hence, assumption 3 has to be satisfied to guarantee that a solution is found.

4.3 Algorithm two: efficient algorithm

In this algorithm two subsets are relevant: subset $\mathbb{B}$ contains the nodes that are assumed to have a finite buffer capacity (all other nodes are assumed to have an infinite buffer capacity) and subset $\mathbb{G}$ contains the nodes that are assumed to be underloaded. If a node is underloaded it has an output of λ_i instead of μ_i. In the end of this section an illustration is given for the efficient algorithm.

Before it is tried to explain the algorithm for solving (3.8) (is the same as (3.9)), first the method as shown in Goodman and Massey (1984) for solving (4.1) is explained.

$$\lambda=\alpha+P'(\lambda\wedge\mu) \tag{4.1}$$

In this paper Goodman and Massey have also shown that there is a single solution to (4.1)

Goodman and Massey's method (G&M method):
A loop with counter k is used and $\mathbb{G}(k)$ is the subset of nodes that are known to be underloaded at the start of iteration k. In short Goodman and Massey's method is to first assume that all nodes are overloaded and thus that $\mathbb{G}(1)$ is an empty set. In that case the output of the i th node is equal to the maximal output μ_i and the inputs λ can be calculated according to (4.2).

$$\lambda(1)=\alpha+P'\mu \tag{4.2}$$

This first guess is at worst an overestimate of the true inputs because all nodes where assumed to have a maximal output of μ_i. If in this case all nodes meet $\lambda_i(1)>\mu_i$ then all nodes are indeed overloaded and $\lambda_i(1)$ is the solution to the throughput equation. If on the other hand one or more nodes are stable (and thus underloaded) for this guess then these nodes are underloaded in the true situation. Now again the throughput equation can be solved, but this time using the fact that all nodes indexed by $\mathbb{G}(2)=\{i\mid\lambda_i(1)\leq\mu_i\}$ are underloaded. All other nodes are still assumed to be overloaded. Once again the solution $\lambda_i(2)$ is at worst an overestimate (though more conservative as $\lambda_i(1)$). Therefore, the set $\mathbb{G}(3)=\{i\mid\lambda_i(2)\leq\mu_i\}$ can be larger than or equal to the set $\mathbb{G}(2)$. By induction, the size of $\mathbb{G}(n)$ increases. Therefore, there is a first positive integer n_* less than or equal to N such that $\mathbb{G}(n_*)=\mathbb{G}(n_*+1)$. Hence, the true solution $\lambda_i(n_*)$ and $\mathbb{E}(n_*)=\mathbb{G}(n_*)$ can be found algorithmically.

Extending G&M method for a finite buffer fluid overflow Jackson network:

This efficient algorithm finds the solution if assumption 1 and 4 is satisfied. If assumption 4 is satisfied then all inverses that can be needed with this algorithm are computable. To find the steady state subset $\mathbb{E}$, $\mathbb{F}$ and $\mathbb{U}$ and the steady state inputs λ for a finite buffer fluid overflow Jackson network of N nodes two loops are used. The outer loop has counter k and the inner loop that runs within loop k has counter l. The nodes that are assumed to have a finite buffer capacity are part of subset $\mathbb{B}(k)$ (all other nodes are assumed to have an infinite buffer capacity). This subset $\mathbb{B}$ is constant throughout iteration k. The nodes that are known to be underloaded at the start of iteration k,l are part of subset $\mathbb{G}(k,l)$. Furthermore $\lambda(k,l)$ is the solution at the end of iteration k,l and $\lambda(k)$ is the solution at the end of iteration k.

Iteration $k = 1$:

In this iteration all nodes are assumed to have an infinite buffer capacity and thus $\mathbb{B}(1)$ is an empty set. If all nodes are assumed to have an infinite buffer capacity and thus no node can overflow then (3.8) reduces to (4.1). Therefore, Goodman and Massey's method can be used to algorithmically find inputs $\lambda(1)$. This is the first guess for the solution to (3.8).

This guess is at worst an underestimate because all nodes are assumed to have an infinite buffer capacity and therefore no buffer can overflow. Therefore, the true steady state inputs of all nodes can only be the same or greater than $\lambda(1)$. Hence, all nodes that satisfy $\lambda_i(1) > \mu_i$ for this first guess also overflow in the true situation. If all nodes satisfy $\lambda_i(1) > \mu_i$ then all nodes overflow in the true situation and therefore the steady state subset $\mathbb{F}$ is a full set. In that case λ is calculated in the next iteration $k = 2$. If all nodes satisfy $\lambda_i(1) \le \mu_i$ then no node overflows and it can be determined whether a node is in subset $\mathbb{E} = \{i \mid \lambda_i < \mu_i\}$ or in subset $\mathbb{U} = \{i \mid \lambda_i = \mu_i\}$. In that case the algorithm is done. On the other hand if some nodes are found to satisfy $\lambda_i(1) > \mu_i$ and some nodes are found to satisfy $\lambda_i(1) \le \mu_i$ for this situation then a next iteration is needed to see whether the underloaded nodes are underloaded or full in the true situation.

Iteration $k = 2$

The next iteration looks a lot like Goodman and Massey's method. The main difference is that in this iteration the nodes that satisfy $\lambda_i(1) > \mu_i$ are assumed to have a finite buffer capacity and thus $\mathbb{B}(2) = \{k \mid \lambda_i(1) > \mu_i\}$. The equation to solve for iteration 2 is:

$$\lambda = \alpha + P'(\lambda \wedge \mu) + (S_{\mathbb{B}}Q)'(\lambda - \mu) \qquad (4.3)$$

As a first guess for $\lambda(2)$ all nodes that are assumed to have an infinite buffer capacity are assumed to have a maximal output of μ_i and thus $\mathbb{G}(2,1)$ is an empty set. With this information inputs $\lambda(2,1)$ can be calculated according to (4.4).

$$\lambda(2,1) = \alpha + P'\mu + (S_{\mathbb{B}}Q)'(\lambda - \mu) \qquad (4.4)$$

$\lambda(2,1)$ is at worst an overestimate of $\lambda(2)$. If all nodes meet $\lambda_i(2,1) > \mu_i$ then all nodes indeed overflow and all nodes are full in the true situation. If a node satisfies $\lambda_i(2,1) \leq \mu_i$ then this node also satisfies $\lambda_i(2) \leq \mu_i$ and therefore this node is underloaded if only the nodes in subset $\mathbb{B}(2)$ are assumed to have a finite buffer capacity. Now again the throughput equation can be solved, but this time using the fact that all nodes indexed by $\mathbb{G}(2,2) = \{i \mid \lambda_i(2,1) \leq \mu_i\}$ are underloaded. The equation to solve for iteration $2,2$ is:

$$\lambda = \alpha + (S_{\mathbb{G}}P)'\lambda + ((I - S_{\mathbb{G}})P)'\mu + (S_{\mathbb{B}}Q)'(\lambda - \mu) \qquad (4.5)$$

Again $\lambda(2,2)$ is an overestimate of $\lambda(2)$ although a more conservative one then $\lambda(2,1)$ .Therefore, the set $\mathbb{G}(2,3) = \{i \mid \lambda_i(2,2) \leq \mu_i\}$ can be larger than or equal to the set $\mathbb{G}(2,2) = \{i \mid \lambda_i(2,1) \leq \mu_i\}$. For iteration $2,3$ the nodes in subset $\mathbb{G}(2,3)$ have an output of λ_i instead of μ_i and therefore all nodes have the same or a smaller steady state input in iteration $2,3$ then they had in iteration $2,2$. As a result $\mathbb{G}(2,4) = \{i \mid \lambda_i(2,3) \leq \mu_i\}$ can be larger than or equal to the set $\mathbb{G}(2,3)$. As can be seen, by induction the size of $\mathbb{G}(2,n)$ increases. Therefore, there is a first positive integer n_* smaller than or equal to N that holds $\mathbb{G}(2,n_*) = \mathbb{G}(2,n_*+1)$ and therefore the solution $\lambda_i(2,n_*)$ can be found algorithmically.

This guess $\lambda(2)$ is like $\lambda(1)$ at worst an underestimate because the nodes that were not part of $\mathbb{B}(2)$ cannot overflow and therefore the true inputs of all nodes can be the same as $\lambda(2)$ or greater than $\lambda(2)$. Hence, all nodes that satisfy $\lambda_i(2) > \mu_i$ for this guess also overflow in the true situation. If $|\mathbb{G}|$ is equal to zero and thus all nodes satisfy $\lambda_i(2) > \mu_i$ then all nodes overflow and the true steady state inputs λ are calculated in the next iteration $k = 3$. Further if all nodes that do not belong to $\mathbb{B}(2)$ are found to satisfy $\lambda_i(2) \leq \mu_i$ then it can be determined whether these underloaded nodes are part of $\mathbb{E} = \{i \mid \lambda_i < \mu_i\}$ or part of $\mathbb{U} = \{i \mid \lambda_i = \mu_i\}$ and the algorithm is done. On the other hand if some nodes that are not part of $\mathbb{B}(2)$ are found to satisfy $\lambda_i(2) > \mu_i$ and some others are found to satisfy $\lambda_i(2) \leq \mu_i$ for this situation then a next iteration is needed to see whether these underloaded nodes are full or underloaded in the true situation.

Iteration $k = 3$ till $k = N$:

All the nodes that satisfy $\lambda_i(k-1) > \mu_i$, and thus are part of $\mathbb{B}(k)$, overflow in the true situation. Therefore, in iteration k all these nodes are assumed to have a finite buffer capacity. The solution $\lambda(k)$ is found algorithmically in the same way as for $k = 2$. If all nodes satisfy $\lambda_i(k) > \mu_i$ and thus if $|\mathbb{G}| = 0$ then all nodes overflow and the solution λ is calculated in the next iteration. If all nodes that are not part of $\mathbb{B}(k)$ satisfy $\lambda_i(k) \leq \mu_i$ then it can be determined whether these underloaded nodes are part of $\mathbb{E} = \{i \mid \lambda_i < \mu_i\}$ or part of $\mathbb{U} = \{i \mid \lambda_i = \mu_i\}$ and the algorithm is done. If some nodes that are not part of $\mathbb{B}(k)$ are found to satisfy $\lambda_i(k) > \mu_i$ and some are found to satisfy $\lambda_i(k) \leq \mu_i$ then another iteration is needed until the solution is found. If a next iteration is needed then $|\mathbb{B}(k)| > |\mathbb{B}(k-1)|$ because if no extra node is found to overflow in iteration k then the solution is found.

On the next page the flowchart for this algorithm is shown.

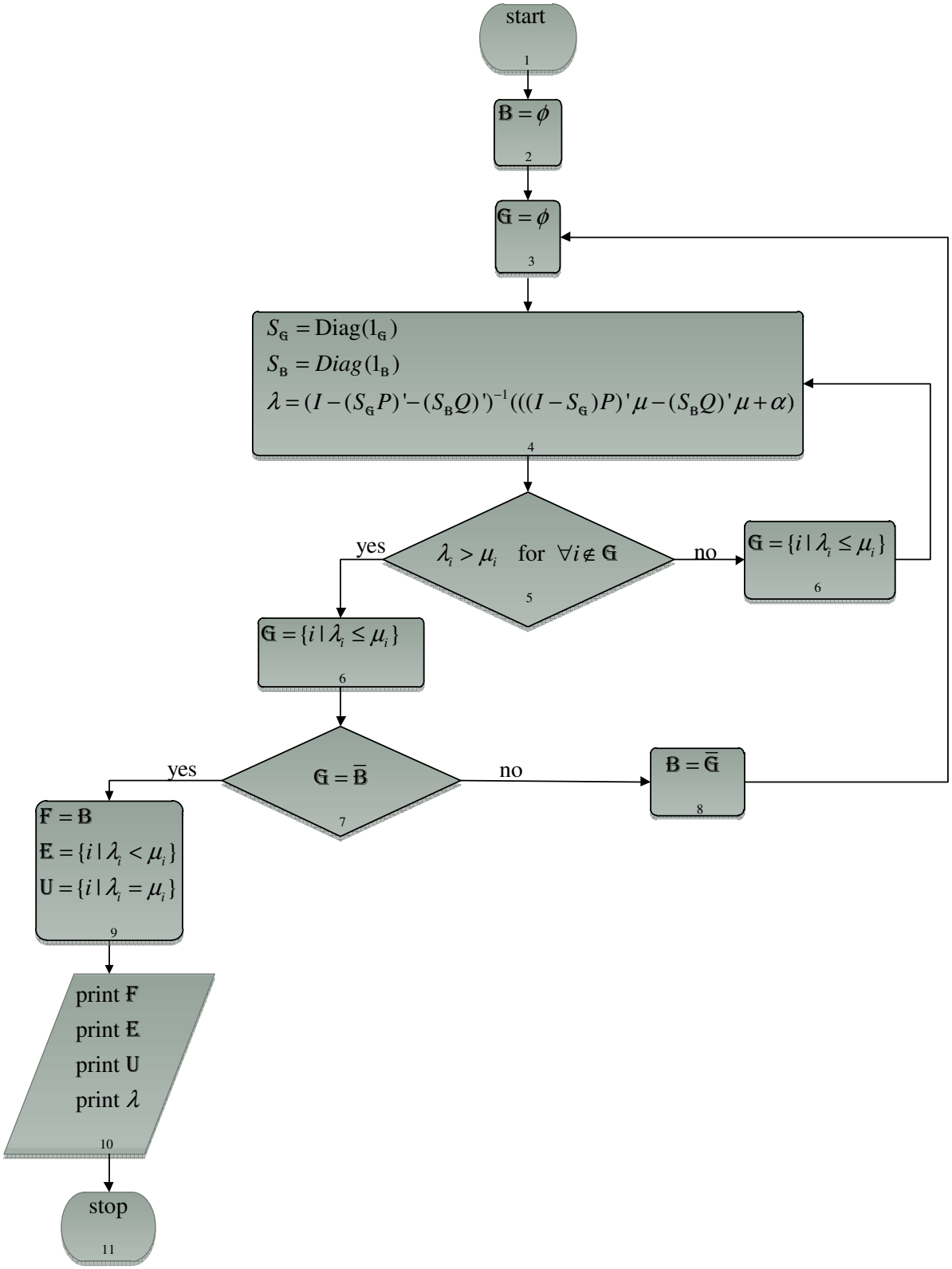

Figure 4.1 flowchart for the efficient algorithm

In appendix B the Mathematica script for this efficient algorithm is shown.

Illustration for the efficient algorithm:

In this illustration it is tried to make the efficient algorithm more clearly. This illustration is not based on a specific network (in terms of matrices P, Q and vectors α and μ). However it shows how the algorithm works by using an abstract finite buffer fluid overflow Jackson network of 4 nodes. In this network all nodes have a maximal process rate of μ.

Iterations $k = 1$:

In this iteration all buffers are assumed to have an infinite buffer capacity. Thus, $\mathbb{B}(1)$ is an empty set. In iteration $(k,l) = (1,1)$ every node is assumed to have a maximal output of μ, and thus the nodes are expected to have more input than their maximal process rate μ. Now the inputs can be calculated according to (4.3). A possible outcome for a network with 4 nodes is shown in Figure 4.2.

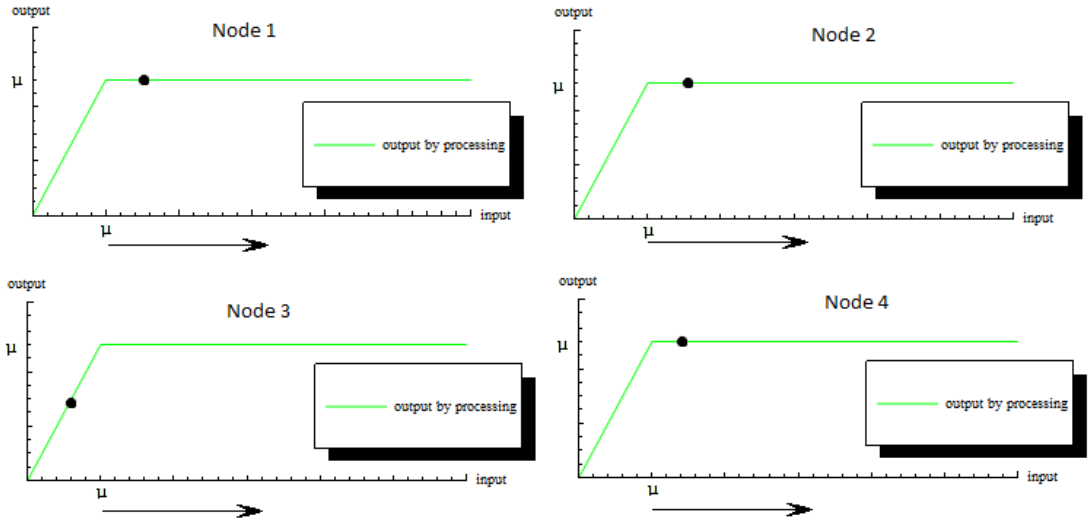

Figure 4.2: outcome of iteration 1,1

In this figure the arrows show where the inputs are assumed to be. And the black dots are the inputs that are calculated. It can be seen that node 3 does not satisfy $\lambda_3 > \mu$ and therefore has an output of λ_3 if all nodes are assumed to have an infinite buffer capacity. Now again the inputs are calculated but this time using that node 3 has an output of λ_3 instead of μ. A possible outcome is shown below. The gray dots are the previous results and the black dots are the new calculated inputs.

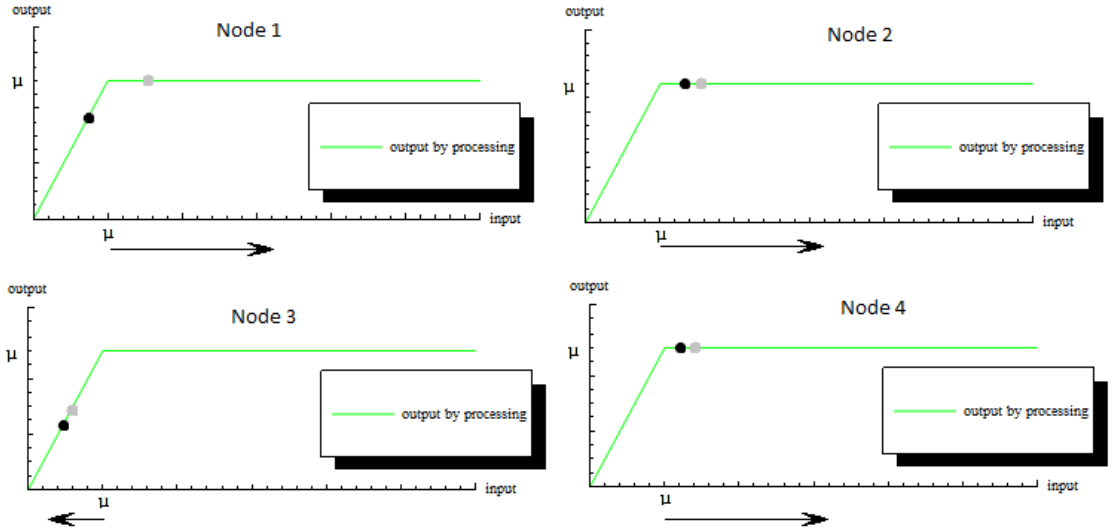

Figure 4.3: outcome of iteration 1,2

Notice that the inputs can only be the same or smaller than in the previous iteration because the output of node 3 has decreased and the other outputs have remained equal to μ. Now node 1 is found to have an input λ_1 smaller than its maximal process rate μ. Again the inputs can be calculated, this time using that nodes 1 and 3 both have an output λ_i instead of μ. A possible result is shown below.

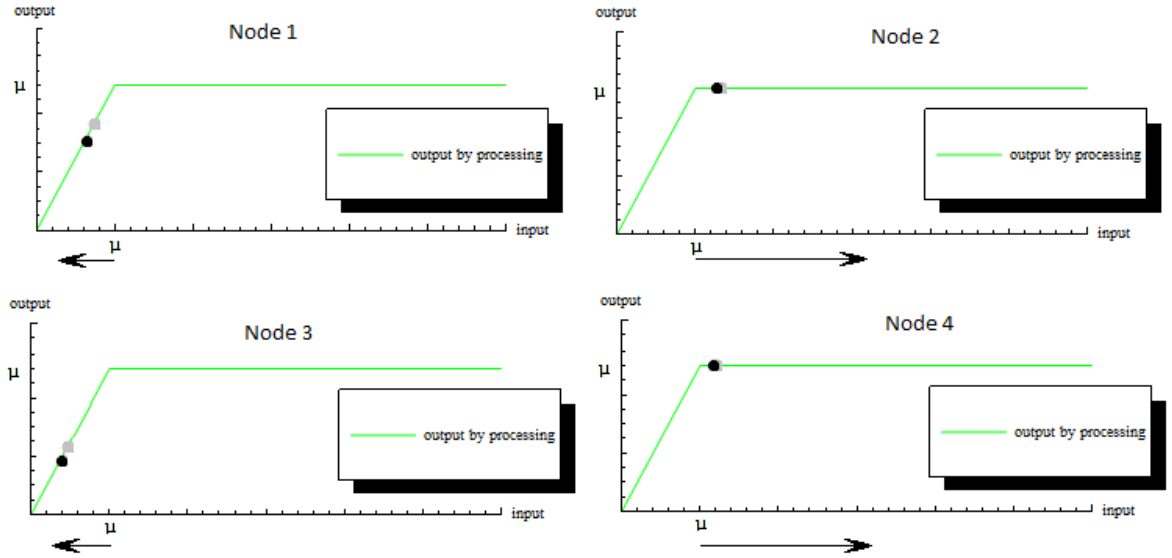

Figure 4.4: outcome of iteration 1,3

This time no extra node is found to have an input smaller than its maximal process rate and it is known that nodes 2 and 4 both overflow in the true situation. This because if buffers have a finite buffer capacity instead of an infinite buffer capacity the input cannot decrease.

Iterations $k = 2$:

Now another iteration is done but this time using that nodes 2 and node 4 both have a finite buffer capacity and therefore using that both buffers overflow. Node 1 and node 3 are again assumed to have a maximal output of μ. A possible solution for this iteration is shown below.

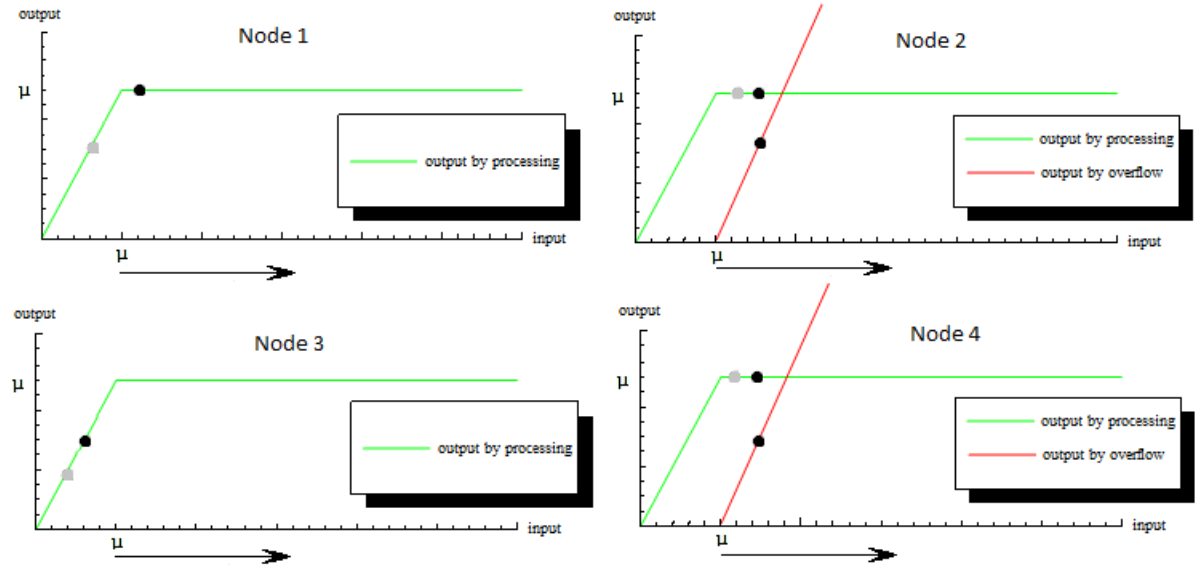

Figure 4.5: outcome of iteration 2,1

Because nodes 2 and 4 now have a greater output because both nodes overflow, all nodes have an input λ_i equal to or greater than was found in the previous iteration. In this iteration node 3 is found to have an input less than its maximal process rate if nodes 2 and 4 overflow. Therefore, node 3 has an output of λ_3. With this knowledge another iteration is done.

A possible result is shown below.

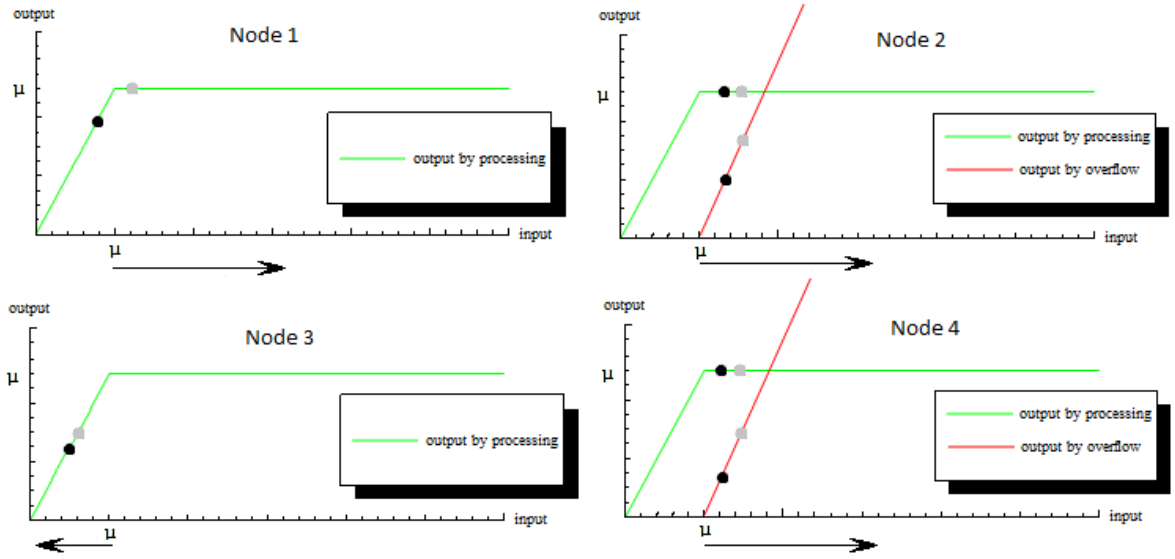

Figure 4.6: outcome of iteration 2,2

It can be seen that nodes 1 is also found to have an input smaller than its maximal output. And therefore this node is also underloaded if nodes 2 and 4 overflow. Now it is already known that node 1 and 3 are underloaded and nodes 2 and 4 are overloaded in steady state. The algorithm however does another iteration to obtain the true inputs λ as well. Thus, the algorithm continues until the dots agree with the arrows. A possible outcome of the last iteration is shown in Figure 4.7.

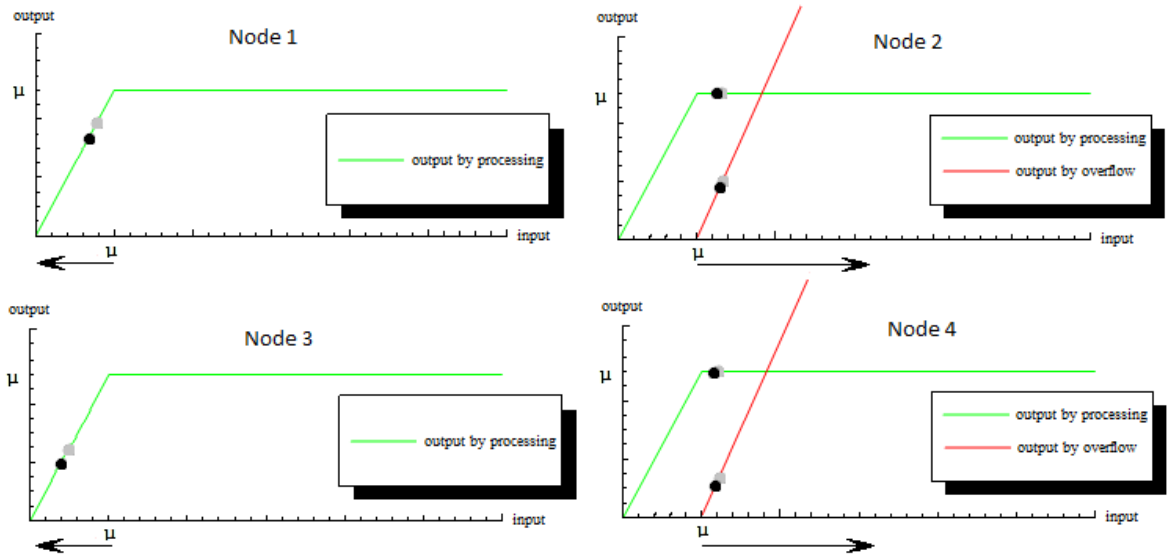

Figure 4.7: outcome of iteration 2,3

Now the algorithm is done!

Maximal number of iterations:

If no extra node is found to overflow in iteration k then this algorithm would be done. Therefore, the outer loop can be needed a maximum of N times. In this case per iteration only one node extra is found to overflow. If i nodes have been found to overflow before iteration k then during iteration k the inner loop can be needed a maximum of $N-i$ times to determine whether or not an extra node overflows. In the last big loop, the smaller loop can be needed twice to calculate the true inputs λ. Therefore, λ can be calculated a maximum of $\sum_{i=0}^{i=N}(N-i)+1=1/2N(N+1)+1$ times before the true solution is found. Brute force exceeds $1/2N(N+1)+1$ because $1/2N(N+1)+1=2^N$ for $N=1,2$ and $1/2N(N+1)+1<2^N$ for all other network sizes. The probability that the maximum number of iterations is needed for this efficient algorithm decreases for increasing network size N.

Remark:

In calculation 4 in the flowchart (see Figure 4.1) $(I-(S_{\mathbb{G}}P)'-(S_{\mathbb{B}}Q)')^{-1}$ has to be computed. To guarantee that a solution is found, λ must be computable for every possible combination of nodes in $\mathbb{G}$ and $\mathbb{B}$. Hence, assumption 4 has to hold to guarantee that a solution is found. Note that a node cannot be part of subset $\mathbb{G}$ and $\mathbb{B}$ at the same time and that a node can be part of neither one of them.

4.4 Comparing brute force and the efficient algorithm

In this section the brute force algorithm and the efficient algorithm are compared to each other on account of efficiency. To do so, for both algorithms the number of iterations that is needed to obtain the steady state subsets $\mathbb{E}$, $\mathbb{F}$ and $\mathbb{U}$ and the steady state inputs λ is acquired. The number of iterations is equal to the number of times λ is calculated. Every time λ is calculated an inverse has to be calculated which has a time complexity of $O(N^3)$. This means that if the network size doubles it takes about $2^3 = 8$ times as much steps to calculate the inverse. For each network size the algorithms are done a thousand times. Every time new random matrices P and Q and random vectors α and μ where used. Below in Figure 4.8 the Mathematica code that makes these random matrices P and Q and random vectors α and μ can be seen.

```
NN = 400 ; (*network size*)
(*randomly choose matrix P*)
Ptemp = Table[Random[], {NN}, {NN}];
totalP = Map[Total, Ptemp];
P = Ptemp / totalP * Table[Random[], {NN}];

(*randomly choose matrix P*)
Qtemp = Table[Random[], {NN}, {NN}];
totalQ = Map[Total, Qtemp];
Q = Qtemp / totalQ * Table[Random[], {NN}];

(*randomly choose a: 0 ≤ a ≤ 1 *)
a = Table[Random[], {NN}];

(*randomly choose mu: 0 ≤ mu ≤ 2 *)
mu = Table[Random[] * 2, {NN}];
```

Figure 4.8: Mathematica code for generating random matrices and random vectors

In Figure 4.9 the results are shown for the brute force algorithm.

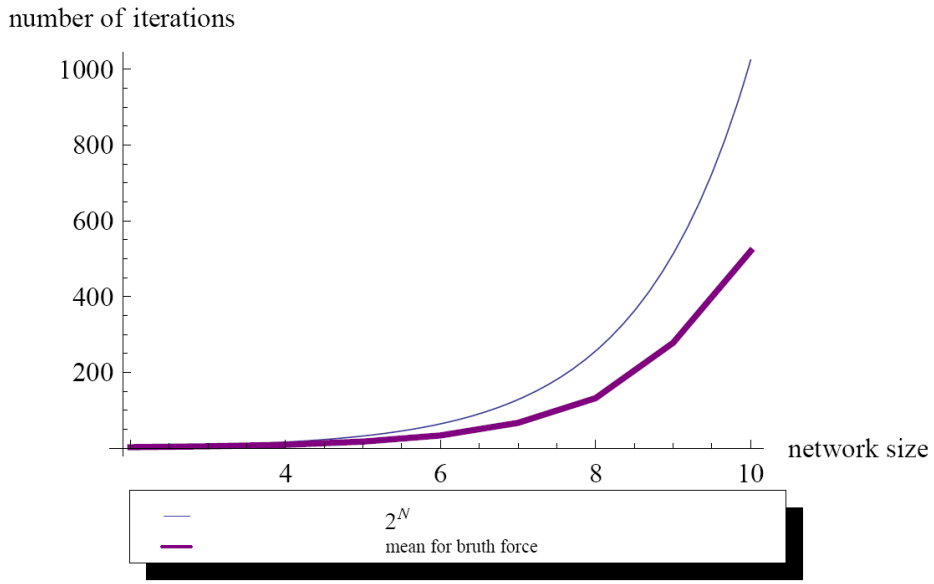

Figure 4.9: Results for brute force algorithm

As said the maximal number of iterations needed is 2^N. For these thousand results per network size about every number of iterations between zero and the maximum 2^N has occurred. Further it can be seen that the average number of iterations increases exponentially when the network size goes up. As said, in every iteration λ has to be calculated which has a time complexity of $O(N^3)$. Therefore, the average-case complexity and the worst-case complexity of this algorithm is $O(2^N N^3)$. Which means that the average number of steps needed and the maximal number of steps that can be needed to find the solution increases exponentially with increasing network size.

In the figure shown below the results for the efficient algorithm can be seen for small network sizes.

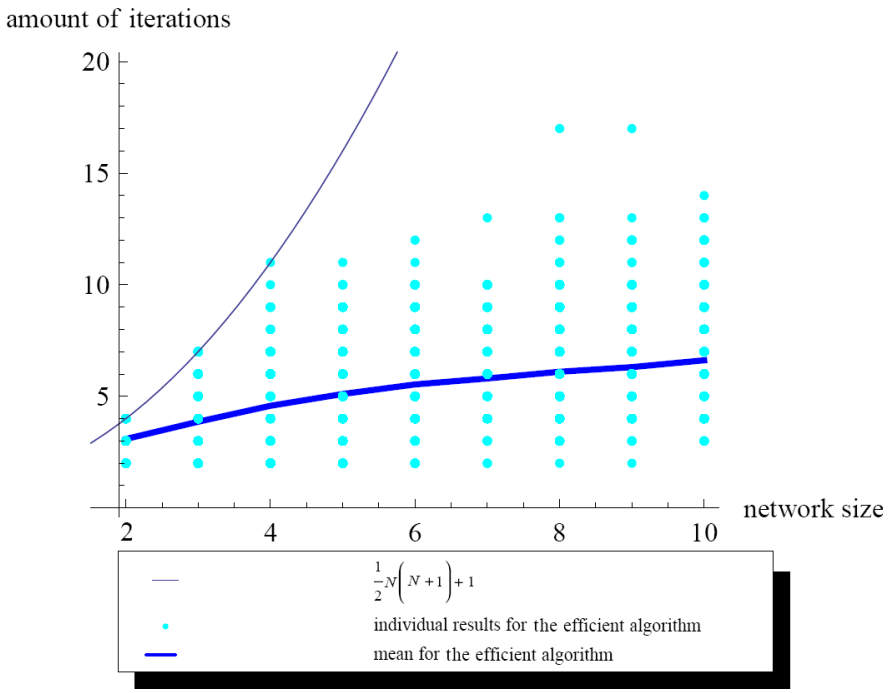

Figure 4.10: Results for the efficient algorithm for small network sizes

Only for $N = 2,3,4$ the measured maximal number of iterations is equal to the maximal number of iterations $(N/2+1/2)N+1$ that can be needed. For network sizes $N = 4$ until $N = 10$ the highest measured number of iterations needed is only 1 of the 1000 measurements.

In the figure below, the average number of iterations needed for the brute force algorithm and the efficient algorithm are shown in the same figure.

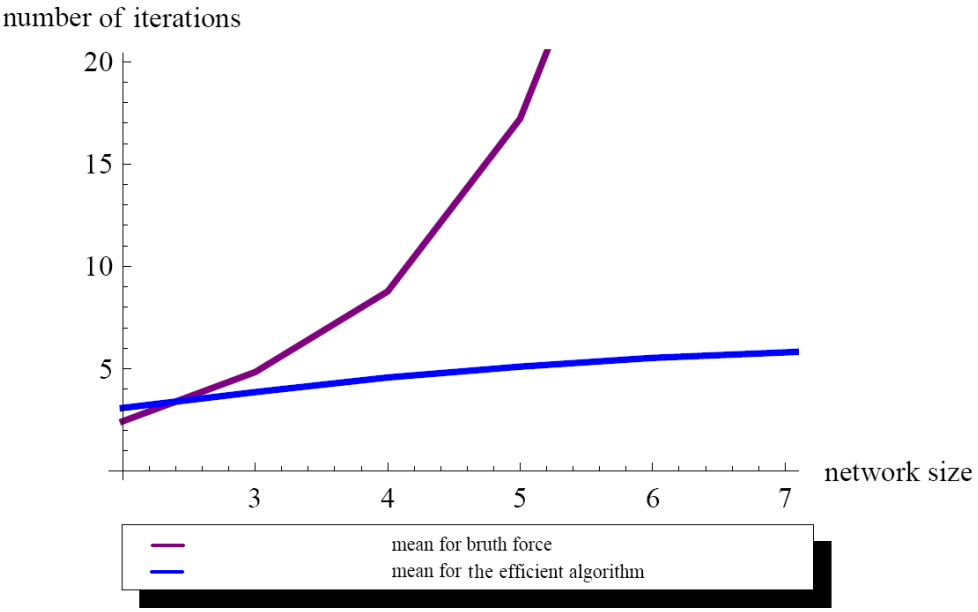

Figure 4.11: Results for the efficient algorithm and brute force

Apparently the brute force algorithm is more efficient than the other algorithm for a network size of $N = 2$. However for all other network sizes the efficient algorithm finds the results in less iterations than brute force. The difference between these two lines increases very quickly for increasing network size N which means that with the efficient algorithm the solutions can be calculated for much greater networks then is possible with brute force.

The efficient algorithm is also tested for larger network sizes. The results for all tested network sizes is shown in Figure 4.12.

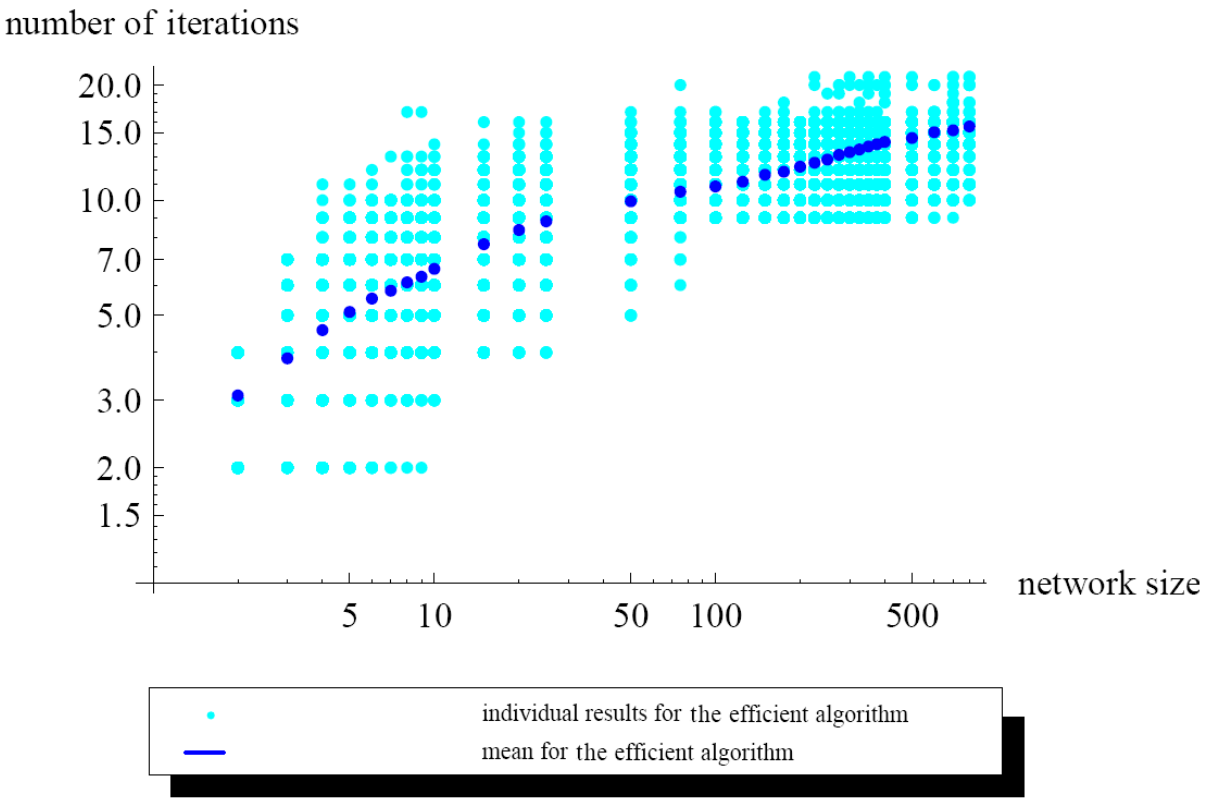

Figure 4.12: results for the efficient algorithm

In this figure it can be seen that very few iterations are needed for large network. The solutions for a network of 800 nodes can be obtained in an average of about 16 iterations. For all networks size greater than 4 nodes the maximum number of iterations $(N/2+1/2)N+1$ that can maximally be needed is not obtained in these thousand measurement.

Time complexity of the efficient algorithm:

To determine the time complexity of this algorithm (for networks that are generated with the Mathematica code in Figure 4.8) first it must be known how the number of iterations increases for increasing network size. A function $g(N)$ must be found such that: $\lim_{N\to\infty} \frac{f(N)}{g(N)} = \text{constant}$, where $f(N)$ is the number of iterations needed for a network size of N nodes.

In the figure below the result is shown for $g(N) = \log(N)$

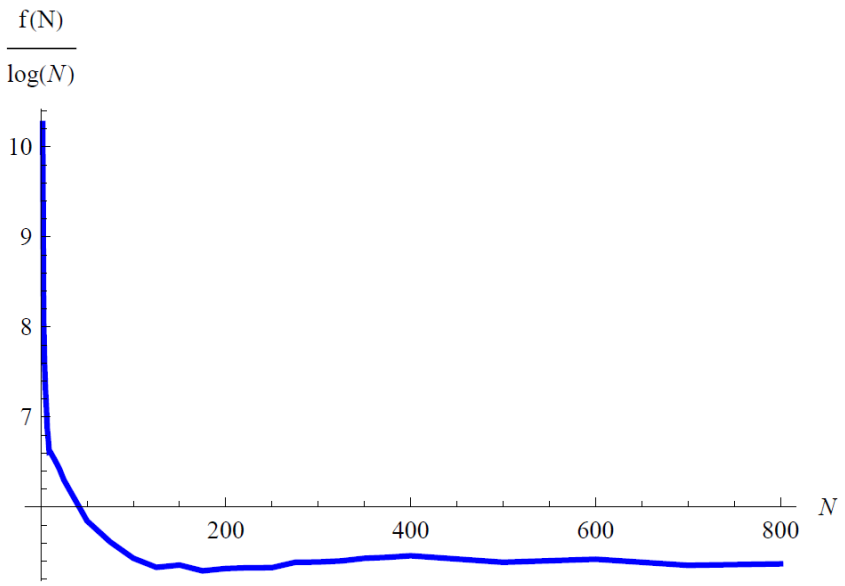

Figure 4.13: $\frac{f(N)}{g(N)}$ **for** $g(N)=log(N)$

In the figure below the result is shown for $g(N) = N^{0.17}$

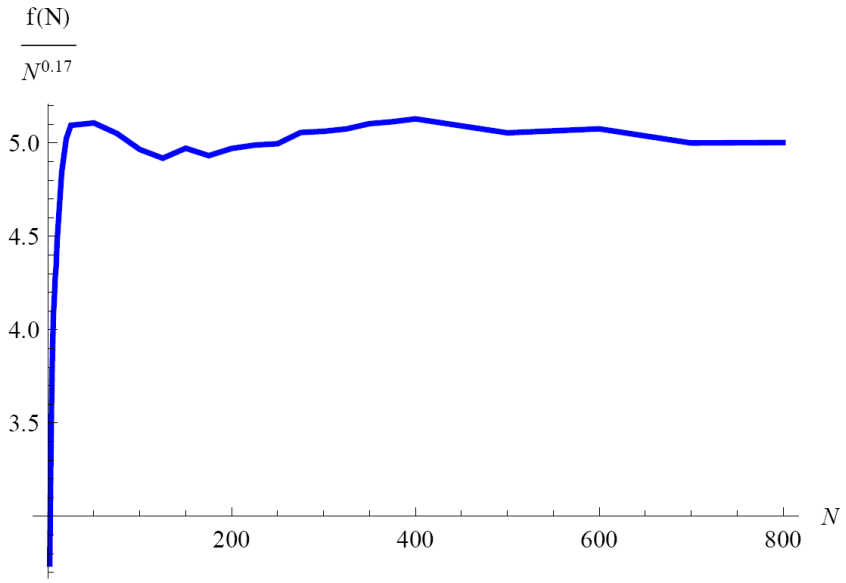

Figure 4.14: $\frac{f(N)}{g(N)}$ **for** $g(N)=N^{0.17}$

It can be seen that for both functions $\lim_{n\to\infty} f(N)/g(N) \approx 5$. Thus, the function $g(N)$ can be both $\log(N)$ and $N^{0.17}$. Probably the amount of iterations needed increases with $\log(N)$ but for small network sizes $N^{0.17}$ behaves the same as $\log(N)$. To determine which function is the right one, the algorithm has to be tested for larger network sizes. As said inverting a matrix has a time complexity of $O(N^3)$. Therefore, the average-case time complexity for the algorithm is probably $O(N^3 \log(N))$. In worst-case $O(N^2)$ iterations are needed. Thus, the worst-case time complexity of the algorithm is $O(N^3 N^2) = O(N^5)$ and therefore this is a polynomial time algorithm.

Chapter 5 Trajectory calculation

In this chapter it is shown that the efficient algorithm can also be used to calculate exact trajectories. In the end an interactive graphical representation for a network of 3 nodes and a program that calculates and visualizes exact trajectories for networks with general network sizes are shown.

5.1 Proving that the efficient algorithm can be used to calculate exact trajectories

Lemma: *the efficient algorithm can be used to calculate exact trajectories*

Proof:

In the previous chapter an algorithm is shown that can find the solution for the equation $\lambda = \alpha + P'(\lambda \wedge \mu) + Q'(\lambda - \mu)^+$.

If you want to calculate a trajectory then (3.5) has to be solved to find the inputs $\lambda(t)$ for some mode (a mode is a certain combination of subsets $\mathbb{E}(t)$ and $\mathbb{F}(t)$). Below (3.5) is again shown:

$$\lambda(t) = \alpha + (S_{\bar{\mathbb{E}}(t)}P)'\mu + (S_{\mathbb{E}(t)}P)'(\lambda(t) \wedge \mu) + (S_{\mathbb{F}(t)}Q)'(\lambda(t) - \mu)^+$$

For each mode, the subsets $\mathbb{E}(t)$ and $\mathbb{F}(t)$ are known and constant. There are 3^N different combinations of subsets $\mathbb{E}(t)$ and $\mathbb{F}(t)$ because a node can be grouped in neither $\mathbb{E}(t)$ nor $\mathbb{F}(t)$. The modes are numbered $m = 1, 2, .., 3^N$ and subsets $\mathbb{E}(t)$ and $\mathbb{F}(t)$ in mode m are referred to as respectively $\mathbb{E}_m(t)$ and $\mathbb{F}_m(t)$. It can be seen that (3.5) and (3.8) have the same form. Therefore, the inputs $\lambda(t)$ in mode m are the same as the steady state inputs λ for a network that has routing matrix $S_{\mathbb{E}_m(t)}P$ for the fluid that is processed by a node, routing matrix $S_{\mathbb{F}_m(t)}Q$ for the fluid that overflows at a node and $\alpha + (S_{\bar{\mathbb{E}}_m(t)}P)'\mu$ as exogenous inputs. Thus, the same efficient algorithm can also be used to calculate the inputs $\lambda(t)$ in mode m.

5.2 Assumptions on the input data for the trajectory calculation programs

These assumptions are made on the input data for the trajectory calculation program. These assumptions are referred to in this chapter.

Assumption 5: $\det(I - S_{\mathbb{G}(t)} S_{\mathbb{E}_m(t)} P - S_{\bar{\mathbb{G}}(t)} S_{\mathbb{F}_m(t)} Q) \neq 0$ for each mode m and $\forall \mathbb{G} \subseteq \mathbb{N}$ If this assumption is satisfied then the trajectory can always be calculated with brute force
Assumption 6: $\det(I - S_{\mathbb{E}_m(t)} S_{\mathbb{G}(t)} P - S_{\mathbb{F}_m(t)} S_{\bar{\mathbb{G}}(t)} Q) \neq 0$ for each mode m and $\forall \mathbb{G}(t), \mathbb{B}(t) \subseteq \mathbb{N}$ If this assumption is satisfied then the trajectory can always be calculated with the efficient algorithm

5.3 programs that can calculate exact trajectories

In Figure 5.1 the flowchart for calculating an exact trajectory is given.

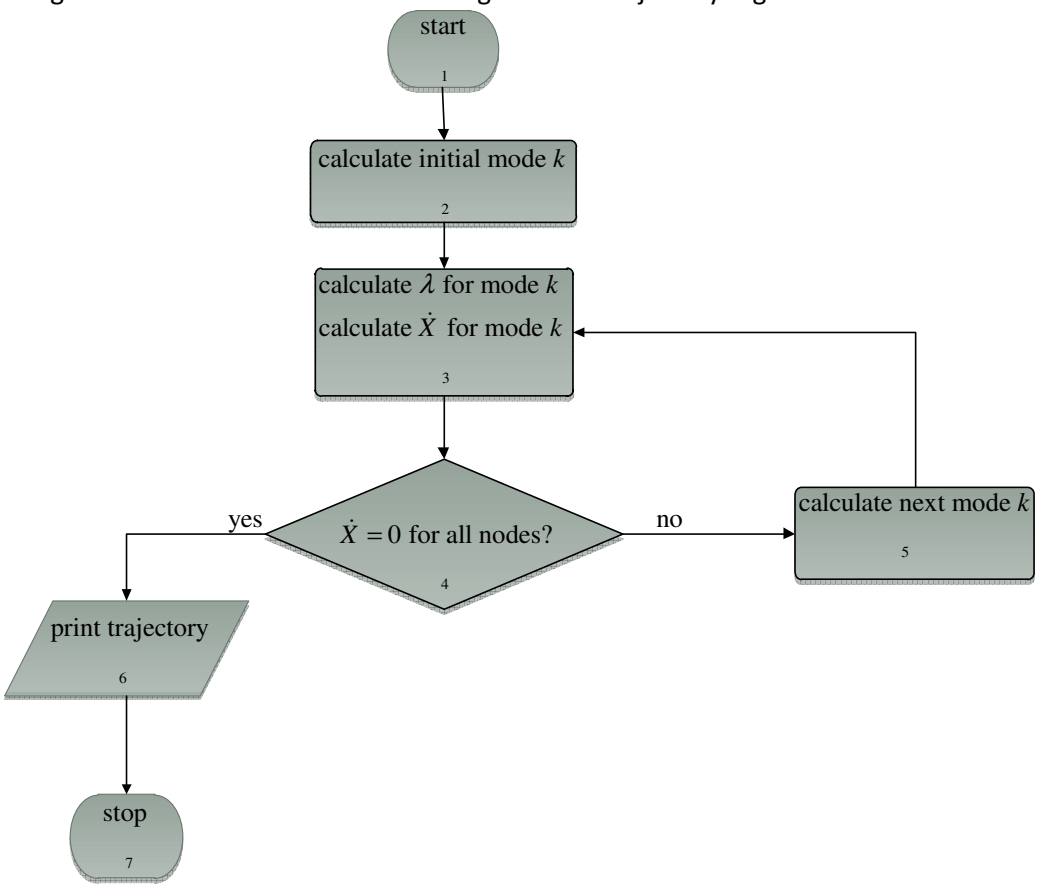

Figure 5.1: flowchart for calculating an exact trajectory

In calculation 3 λ can be calculated with brute force or the efficient algorithm and $\dot{X}$ can be calculated with (3.7). To guarantee that the trajectory can be calculated, λ should be computable for every mode. In chapter 4 it is shown that assumption 3 and assumption 4 have to hold to guarantee that a solution is found by respectively brute force and the efficient algorithm. As shown previously the steady state equation can be transformed to calculate dynamic inputs $\lambda(t)$ by replacing P, Q and α by respectively $S_{\mathrm{E}_m(t)}P$, $S_{\mathrm{F}_m(t)}Q$ and $\alpha+(S_{\bar{\mathrm{E}}_m(t)}P)'\mu$. Therefore, assumptions 3 and 4 can be transformed to assumptions 5 and 6 as well. If assumption 5 and assumption 6 are satisfied a trajectory can always be calculated by respectively brute force or the efficient algorithm.

Two Mathematica programs that can calculate exact trajectories are made. In appendix C a Mathematica script is shown for a finite buffer fluid overflow Jackson network of 3 nodes. On http://demonstrations.wolfram.com/DynamicsOfADeterministicOverflowFluidNetwork/ the interactive demonstration and more information is shown. In this program λ is calculated with brute force and therefore this program can calculate a exact trajectories if assumption 5 is satisfied. A disadvantage of this program is that it only works with buffer sizes equal to 1. However with this program it can easily be verified what the effects of changing exogenous input α_i for node i are, as well as the effects of changing the initial amount of fluid inside the buffers. In Figure 5.2 the visualization of the results is shown. On the axes it is shown how full a buffer is. For all modes an arrow points in the direction of $\dot{X}$. The sizes of these arrows give an indication of the relative sizes of $\dot{X}$. The blue arrow shows the trajectory for a given starting point. These starting values X_i^0 as well as the external inputs α_i can be changed with the slide bars for all 3 nodes.

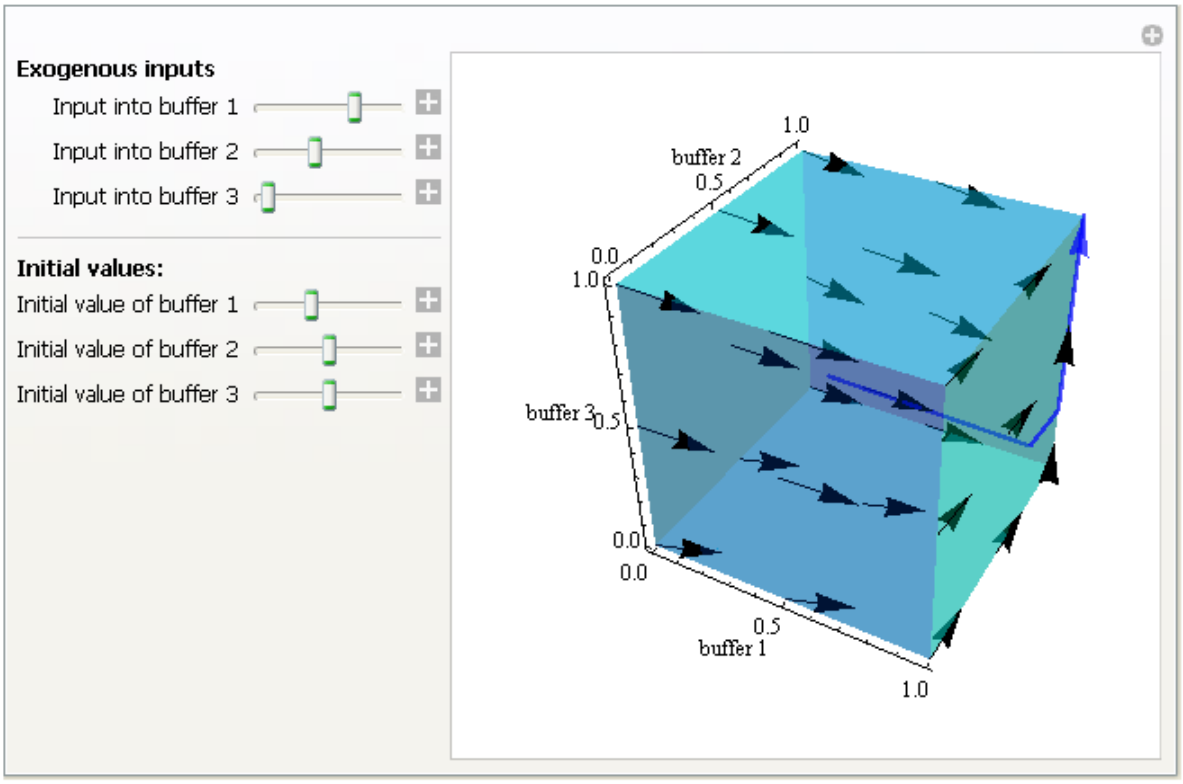

Figure 5.2: Visualization of the results of the program for 3 nodes

In appendix D a Mathematica scripts is shown that can calculate trajectories for a finite buffer fluid overflow Jackson network of N nodes. In this program λ is calculated with the efficient algorithm and therefore a trajectory can always be calculated if assumption 6 is satisfied. In Figure 5.3 four screenshots of the visualization are shown for a network of 20 nodes. The matrices P and Q and vectors α and μ are randomly generated with the Mathematica code in Figure 4.8. The initial amount of fluid in the buffers is randomly chosen equal to 1,2 or 3 and the buffer capacities are randomly chosen equal to 3,4 or 5. The empty buffers are visualized with a green color, the full buffers with a red color and the buffers that are neither empty nor full are shown in orange. With + and - you can skip forward and backward through the trajectory.

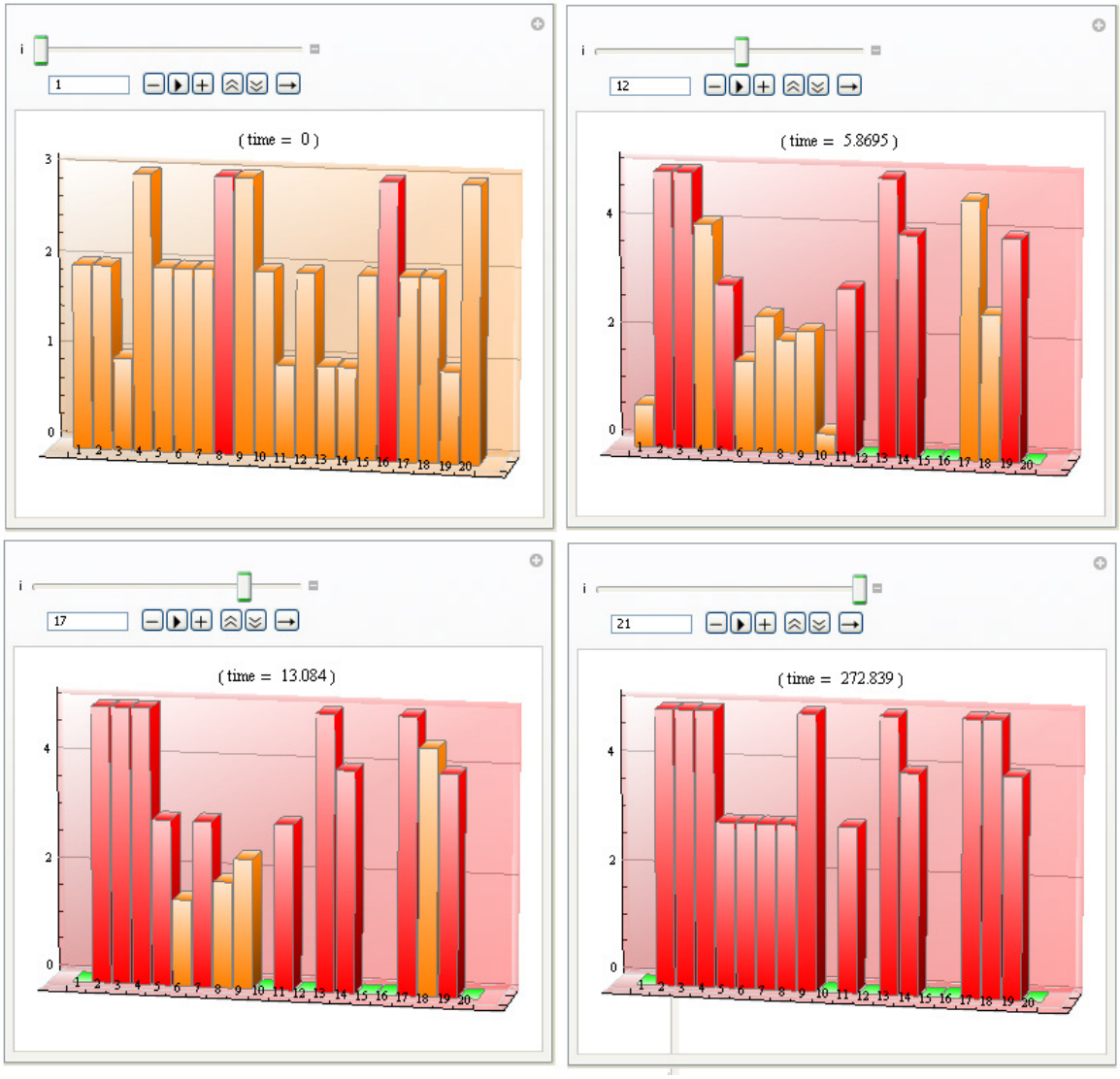

Figure 5.3: Visualization of the results of the program for N nodes

Observations:

These observations are made with the trajectory calculating programs:

- No cyclic behavior is observed: There is no buffer that switches between being empty and full until infinity.
- Before steady state is reached buffers can switch between being empty and full.
- If the buffer capacities and the initial amount of fluid inside the buffer increase with a factor f and the other conditions stay the same the time at which a certain buffer becomes full or empty increases with a factor f as well.
- In steady state a buffer is observed to be either empty or full. Buffer i can only be neither full nor empty in steady state if the input λ_i is exactly the same as the maximal processing rate μ_i. This is very unlikely to happen for the random matrices and random vectors it is tested on (though not impossible, see the example in section 3.2).

Chapter 6 Conclusion

Nowadays a wide variety of different manufacturing networks exist. These factories can be modeled by using network models. To optimize these factories, insight for these network models is needed. In this report insight is gained for a finite buffer fluid overflow Jackson network. Because very little was known about this kind of networks only continuous and deterministic fluid flows and deterministic processes are considered in this report.

In this paper it is shown that for a finite buffer fluid overflow Jackson network of 2 nodes there is a single solution for steady state inputs λ except for five singular cases. For a network with general network size N there is a single solution for steady state inputs λ for known steady state subset $\mathbb{E}$ if $\det(I - S_{\mathbb{E}}P - S_{\bar{\mathbb{E}}}Q) \neq 0$ and if every node is either full or empty in steady state (assumption 2). This means that if there is a unique solution for steady state subset $\mathbb{E}$, like assumed in this report (assumption 1), then there is a single solution if $\det(I - S_{\mathbb{E}}P - S_{\bar{\mathbb{E}}}Q) \neq 0$ and if assumption 2 is met. Furthermore for a network of N nodes, the steady state inputs λ are monotonic in α_i. Which means that increasing the exogenous input α_i for node i cannot result into the fact that the steady state input λ_j decreases for node j.

There are two algorithms for finding the nodes that are empty and full in steady state and the steady state inputs λ. The brute force algorithm tries every possible solution until the true solution is found. For this algorithm the average-case and the worst-case time complexity are both $O(2^N N^3)$. The other algorithm can find the solution in less iterations than brute force for a network with more than 2 nodes. This algorithm has a polynomial time complexity. The average-case time complexity is probably $O(N^3 \log(N))$ and the worst-case time complexity is $O(N^5)$ .Therefore, this algorithm can calculate the solution for much greater networks than is possible with brute force. This efficient algorithm can also be used to calculate exact trajectories. During these trajectory calculations the following observations have been made:

- No cyclic behavior is observed
- Before steady state is reached buffers can switch between being empty and full.
- If the buffer capacities and the initial amount of fluid inside the buffer increase with a factor f and the other conditions stay the same the time at which a certain buffer becomes full or empty increases with a factor f as well.
- In steady state a buffer is observed to be either empty or full

In this report two assumptions, that are not yet mathematically shown, have been made. In the future it can be proven that there is indeed a unique solution for steady state subset $\mathbb{E}$, and that a buffer is indeed either empty or full in steady state. Furthermore to know whether the average-case time complexity of the efficient algorithm is $O(N^3 \log(N))$ or $O(N^{3.17})$ the algorithm has to be tested for larger network sizes. In future research, one can try to write this network model as a linear complementary problem (LCP) like Hong and Mandelbaum (1991) did for the network model given in Goodman and Massey (1984). If this network model can be rewritten as an LCP then the algorithm for solving a LCP can be compared to the efficient algorithm given in this report. In future research, one can also try to include stochastic material flows and processes.

References

[1] Goodman, J.B, & Massey, W.A. (1984). The Non-Ergodic Jackson Network. Journal of Applied Probability, 21, 860-869.

[2] Hong, C & Mandelbaum, A. (1991). Discrete Flow Networks: Bottleneck Analysis and Fluid Approximations. Mathematics of Operations Research, 16(2), 508-446

Appendix A code for brute force

With this algorithm the steady state inputs λ can be calculated. This algorithm can also calculate what nodes are empty and full in steady state and what nodes have an unknown amount of fluid in their buffers in steady state. For more information about brute force see section 4.2.

Brute force algorithm for solving the equation $\lambda = P(\lambda \wedge \mu) + Q(\lambda - \mu)^+$ for a material overflow processing network

Written by Stijn Fleuren Tu/e Mechanical Engineering 21 – 05 – 2010

Input needed : routingmatrices P and Q, external input a and processrate mu
Output : set E (underloaded nodes) and inputs λ_i (lambda)

```
NN = Length[P]; (*size of the network*)
Nodes = Table[i, {i, 1, NN}];
powerset = Subsets[Nodes]; (*every combination of nodes in subset E*)

(*try every combination of nodes in subset E until solution is found*)
For[j = 1, j ≤ Length[powerset], j++,
 {
  G = Table[0, {NN}];
  For[i = 1, i ≤ NN, i++, If[Count[powerset[[j]], i] == 0, G[[i]] = 1, G[[i]] = 0]];
  SG = DiagonalMatrix[G];
  SB = IdentityMatrix[NN] - SG;
  NodesinG = Flatten[Position[G, 1]];
  lambda = Inverse[IdentityMatrix[NN] - (SG.P)^T - (SB.Q)^T].
    (((IdentityMatrix[NN] - SG).P)^T.mu - (SB.Q)^T.mu + a);

  y = 0;
   For[i = 1, i <= NN, i++, If[G[[i]] == 1, If[lambda[[i]] <= mu[[i]], y++, Break[]],
    If[lambda[[i]] >= mu[[i]], y++, Break[]]]];
   If[y == NN, Break[]]}]

(* make a list of full, empty and unknown buffers*)
EmptyB = {};
FullB = {};
UnknownB = {};
For[i = 1, i ≤ NN, i++, If[G[[i]] == 1 && lambda[[i]] == mu[[i]], AppendTo[UnknownB, i],
  If[G[[i]] == 1 && lambda[[i]] < mu[[i]], AppendTo[EmptyB, i], AppendTo[FullB, i]]]]

(*printing the results*)
Print["Empty buffers", Tab, "=" Tab, EmptyB];
Print["Full buffers ", Tab, "=" Tab, FullB];
Print["Unknown buffers   =", Tab, UnknownB];
Print["λ(lambda) ", Tab, Tab, "=", Tab, lambda];
Print["μ(mu) ", Tab, Tab, Tab, "=", Tab, mu];
```

Appendix B code for efficient algorithm

With this efficient algorithm the steady state inputs λ can be calculated. This efficient algorithm can also calculate what nodes are empty and full in steady state and what nodes have an unknown amount of fluid in their buffers in steady state. For more information about this efficient algorithm see section 4.3.

Efficient algorithm for solving the equation $\lambda = P(\lambda \wedge \mu) + Q(\lambda - \mu)^+$ for a material overflow processing network

Written by Stijn Fleuren Tu/e Mechanical Engineering 21 – 05 – 2010

Input needed: routingmatrices P and Q, external input a and processrate mu
Output: Empty nodes, full nodes, unknown nodes and inputs λ_i (lambda)

```
NN = Length[P];      (*network size*)
(* calculation 2 in flowchart*)
 B = Table[0, {NN}];

 endloop1 = False;

 While[endloop1 == False, {

  (*calculation 3 in flowchart*)
  G = Table[0, {NN}];

  endloop2 = False;
  While[endloop2 == False, {

   (*calculation 4 in flowchart*)
    SG = DiagonalMatrix[G];
    SB = DiagonalMatrix[B];
   lambda = Inverse[IdentityMatrix[NN] - (SG.P)^T - (SB.Q)^T].
      (((IdentityMatrix[NN] - SG).P)^T.mu - (SB.Q)^T.mu + a);

   (*check 5 in flowchart*)
     y = 0;
     For[i = 1, i <= NN, i++,
     If[(G[[i]] == 1 && lambda[[i]] <= mu[[i]]) || (G[[i]] == 0 && lambda[[i]] > mu[[i]]),
      y++, Break[]]];
     If[y == NN, endloop2 = True];

   (*calculation 6 in flowchart*)
     For[i = 1, i <= NN, i++, If[lambda[[i]] > mu[[i]], G[[i]] = 0, G[[i]] = 1]];
  }];

  (*check 7 in flowchart*)
 If[(G + B) == Table[1, {NN}], endloop1 = True];

  (*calculation 8 in flowchart*)
 For[i = 1, i <= NN, i++, If[G[[i]] == 0, B[[i]] = 1, B[[i]] = 0]];

}]
```

```
(*calculation 9 in flowchart*)
EmptyB = {};
FullB = {};
UnknownB = {};
For[i = 1, i ≤ NN, i++, If[G[[i]] == 1 && lambda[[i]] == mu[[i]], AppendTo[UnknownB, i],
  If[G[[i]] == 1 && lambda[[i]] < mu[[i]], AppendTo[EmptyB, i], AppendTo[FullB, i]]]]


(* output 10 in flowchart*)
Print["Empty buffers", Tab, "=" Tab, EmptyB];
Print["Full buffers ", Tab, "=" Tab, FullB];
Print["Unknown buffers   =", Tab, UnknownB];
Print["λ(lambda) ", Tab, Tab, "=", Tab, lambda];
Print["μ(mu) ", Tab, Tab, Tab, "=", Tab, mu];
```

Appendix C code for the interactive demonstration

This is the Mathematica code for an interactive demonstration of a finite buffer fluid overflow Jackson network consisting of 3 nodes. With this demonstration the effect of changing exogenous inputs α_i and the initial amount of fluid inside the buffers can be shown. For more information about this demonstration see page 34 or visit the website: http://demonstrations.wolfram.com/DynamicsOfADeterministicOverflowFluidNetwork/.

Interactive demonstration for a 3 node finite buffer fluid overflow Jackson network

written by Stijn Fleuren Tu / e Mechanical Engineering 11 – 06 – 2010

input : external inputs (a) and startingpositions (X0)

```
s = 1 * 10^-10;
P = {{0, 0.1, 0.1}, {0.1, 0, 0.1}, {0.1, 0.1, 0}};
Q = {{0, 0.7, 0.2}, {0.3, 0, 0.4}, {0.6, 0.0, 0}};
mu = {1.1, 1.1, 1.1};
NN = 3;
Modes = Flatten[Table[{k, l, m}, {k, 0, 1, 0.5}, {l, 0, 1, 0.5}, {m, 0, 1, 0.5}], 2];

CalcXdot[a_] :=
 Module[{mode = 1, EmptyB, FullB, k, powerset, j, G, SG, SB, Sfull, Sempty, lambda,
   y, Xdot = Table[0, {27}]},
  While[mode ≤ 27, {
    EmptyB = Table[0, {NN}];
    FullB = Table[0, {NN}];
    For[k = 1, k ≤ 3, ++k, {If[Modes[[mode, k]] == 0, EmptyB[[k]] = 1],
      If[Modes[[mode, k]] == 1, FullB[[k]] = 1]}];
    powerset = Subsets[{1, 2, 3}];
    For[j = 1, j ≤ Length[powerset], j++,
     {
      G = Table[0, {NN}];
      For[l = 1, l ≤ 3, ++l, If[Count[powerset[[j]], l] == 0, G[[l]] = 1]];
      SG = DiagonalMatrix[G];
      SB = IdentityMatrix[NN] - SG;
      Sfull = DiagonalMatrix[FullB];
      Sempty = DiagonalMatrix[EmptyB];

       lambda = Inverse[IdentityMatrix[NN] - (SG.Sempty.P)^T - (SB.Sfull.Q)^T].
         (((IdentityMatrix[NN] - SG).Sempty.P)^T.mu - (SB.Sfull.Q)^T.mu + a +
           ((IdentityMatrix[NN] - Sempty).P)^T.mu);
      y = 0;
      If[SG.(mu - lambda) + (IdentityMatrix[NN] - SG).(lambda - mu) ==
        SG.Abs[(mu - lambda)] + (IdentityMatrix[NN] - SG).Abs[(lambda - mu)], Break[]];
     }];

    Xdot[[mode]] = (IdentityMatrix[NN] - SG.Sempty - (IdentityMatrix[NN] - SG).Sfull).
      (lambda - mu);
    mode++;
   }];
  Xdot
]
```

```
CalcTempMode[X1_, X2_, X3_] := Which[X1 < s && X2 < s && X3 < s, 1,
  X1 > (1 - s) && X2 < s && X3 < s, 19,
  X1 < s && X2 > (1 - s) && X3 < s, 7,
  X1 < s && X2 < s && X3 > (1 - s), 3,
  X1 > (1 - s) && X2 > (1 - s) && X3 < s, 25,
  X1 > (1 - s) && X2 < s && X3 > (1 - s), 21,
  X1 < s && X2 > (1 - s) && X3 > (1 - s), 9,
  X1 > (1 - s) && X2 > (1 - s) && X3 > (1 - s), 27,
  X1 < s && X2 < s, 2, X1 > (1 - s) && X2 < s, 20,
  X1 < s && X2 > (1 - s), 8, X1 > (1 - s) && X2 > (1 - s), 26, X1 < s && X3 < s, 4,
  X1 > (1 - s) && X3 < s, 22, X1 < s && X3 > (1 - s), 6, X1 > (1 - s) && X3 > (1 - s), 24,
  X2 < s && X3 < s, 10, X2 > (1 - s) && X3 < s, 16, X2 < s && X3 > (1 - s), 12,
  X2 > (1 - s) && X3 > (1 - s), 18, X1 < s, 5, X1 > (1 - s), 23, X2 < s, 11, X2 > (1 - s),
  17, X3 < s, 13, X3 > (1 - s), 15,
  True, 14]

CalcTrajectory[Xdotlocal_, Xinit_] :=
 Module[{mindtime, m, dtime, Xtemp, TempModelocal, Xlocal = {}},
  AppendTo[Xlocal, Xinit];
  TempModelocal = CalcTempMode[Last[Xlocal][[1]], Last[Xlocal][[2]], Last[Xlocal][[3]]];
  While[Xdotlocal[[TempModelocal]] ≠ Table[0, {3}], {
    mindtime = Infinity;
    For[m = 1, m ≤ 3, ++m,
     {dtime = Infinity,
      If[ Xdotlocal[[TempModelocal, m]] < 0 && Modes[[TempModelocal, m]] ≠ 0,
       dtime = -Last[Xlocal][[m]] / Xdotlocal[[TempModelocal, m]]],
      If[Xdotlocal[[TempModelocal, m]] > 0 && Modes[[TempModelocal, m]] ≠ 1,
       dtime = (1 - Last[Xlocal][[m]]) / Xdotlocal[[TempModelocal, m]]],
      If[dtime < mindtime, mindtime = dtime]}];
    Xtemp = Last[Xlocal] + Xdotlocal[[TempModelocal]] * mindtime;
    AppendTo[Xlocal, Xtemp];

    TempModelocal = CalcTempMode[Last[Xlocal][[1]], Last[Xlocal][[2]], Last[Xlocal][[3]]];
   }];
  Xlocal]
```

```
Manipulate[
 Xdot = CalcXdot[{a1, a2, a3}];
 X = CalcTrajectory[Xdot, {X01, X02, X03}];
 normfactor = Norm[Xdot];
 Graphics3D[{Opacity[0.4], Cuboid[], Opacity[0.7], Thick, Blue, Arrowheads[Large],
   Arrow[X], Opacity[1], Thin, Black,
   Table[Arrow[{Modes[[n]], (Modes[[n]] + Xdot[[n]] / normfactor)}], {n, 27}]},
  Axes → True, AxesLabel → {"buffer 1", "buffer 2", "buffer 3"}, Boxed → False,
  ImageMargins → 10, AxesStyle → Black, BaseStyle → {Cyan, Opacity[1], EdgeForm[None]},
  ImageSize → {350, 350}],
 Style["exogenous inputs", Bold],
 {{a1, 0.5, "buffer 1"}, 0, 4, .01, Appearance → "Labeled", ImageSize → Tiny},
 {{a2, 0.5, "buffer 2"}, 0, 4, .01, Appearance → "Labeled", ImageSize → Tiny},
 {{a3, 0.5, "buffer 3"}, 0, 4, .01, Appearance → "Labeled", ImageSize → Tiny},
 Delimiter,
 Style["initial values", Bold],
 {{X01, 0.5, "buffer 1"}, 0, 1, .01, Appearance → "Labeled", ImageSize → Tiny},
 {{X02, 0.5, "buffer 2"}, 0, 1, .01, Appearance → "Labeled", ImageSize → Tiny},
 {{X03, 0.5, "buffer 3"}, 0, 1, .01, Appearance → "Labeled", ImageSize → Tiny},
 {{Xdot, Table[0, {27}]}, ControlType → None},
 {{X, {{.5, .5, .5}}}, ControlType → None},
 {{normfactor, 0}, ControlType → None},
 {{Modes, Flatten[Table[{k, l, m}, {k, 0, 1, 0.5}, {l, 0, 1, 0.5}, {m, 0, 1, 0.5}], 2]},
  ControlType → None},
 AutorunSequencing → {1, 2, 3, 4, 5, 6},
 TrackedSymbols :> {a1, a2, a3, X01, X02, X03}, SaveDefinitions → True]
```

Appendix D code for trajectory calculation for N -nodes

With this program exact trajectories can be calculated for a network of N nodes. For more information about this program see section 5.3 35.

Algorithm for calculating the trajectory for a N – node material overflow processing network

Written by Stijn Fleuren Tu/e Mechanical Engineering 29 – 05 – 2010

input needed : routingmatrices P and Q, external input vector a, processrate vector mu, buffersizes vector K and startingposition vector X0
Output : trajectory as a function of time

```
NN = Length[P];
Xdot = {};
s = 1 * 10^-10;
X = {X0};
time = {0};
While[Xdot ≠ Table[0, {NN}], {

  (* classify empty and full buffers*)
  EmptyB = Table[0, {NN}];
  FullB = Table[0, {NN}];
  For[i = 1, i <= NN, i++, If[Last[X][[i]] < s, {X[[Length[X], i]] = 0, EmptyB[[i]] = 1},
    If[Last[X][[i]] > (K[[i]] - s), {X[[Length[X], i]] = K[[i]], FullB[[i]] = 1}]]];
  Sfull = DiagonalMatrix[FullB];
  Sempty = DiagonalMatrix[EmptyB];

  (*_____Calculating lambda with eficient algorithm_____*)

     (* calculation 2 in flowchart of efficient algorithm*)
     B = Table[0, {NN}];
     endloop1 = False;
     endloop2 = False;
     While[endloop1 == False, {

     (*calculation 3 in flowchart of efficient algorithm*)
     G = Table[0, {NN}];
     endloop2 = False;

     While[endloop2 == False, {

  (*calculation 4 in flowchart of efficient algorithm*)
    SG = DiagonalMatrix[G];
    SB = DiagonalMatrix[B];
    lambda = Inverse[IdentityMatrix[NN] - (SG.Sempty.P)^T - (SB.Sfull.Q)^T].
      (((IdentityMatrix[NN] - SG).Sempty.P)^T.mu - (SB.Sfull.Q)^T.mu + a +
        ((IdentityMatrix[NN] - Sempty).P)^T.mu);
```

```
 (*check 5 in flowchart of efficient algorithm*)
    y = 0;
    For[i = 1, i <= NN, i++,
     If[(G[[i]] == 1 && lambda[[i]] <= mu[[i]]) || (G[[i]] == 0 && lambda[[i]] > mu[[i]]), y++,
    If[y == NN, endloop2 = True];

 (*calculation 6 in flowchart of efficient algorithm*)
    For[i = 1, i <= NN, i++, If[lambda[[i]] > mu[[i]], G[[i]] = 0, G[[i]] = 1]];
 }];

 (*check 7 in flowchart of efficient algorithm*)
 If[(G + B) == Table[1, {NN}], endloop1 = True];

  (*calculation 8 in flowchart of efficient algorithm*)
 For[i = 1, i <= NN, i++, If[G[[i]] == 0, B[[i]] = 1, B[[i]] = 0]];
 }];

(*_____end of efficient algorithm_____*)


(*calculating Xdot*)
Xdot = (IdentityMatrix[NN] - SG.Sempty - (IdentityMatrix[NN] - SG).Sfull).(lambda - mu);

  (*calculating first modechange*)
  If[Xdot ≠ Table[0, {NN}], {
    mindtime = Infinity;
    For[i = 1, i ≤ NN, i++, {
      dtime = Infinity,
      If[EmptyB[[i]] == 1 && G[[i]] == 0, dtime = K[[i]] / (lambda[[i]] - mu[[i]])],
      If[FullB[[i]] == 1 && G[[i]] == 1, dtime = K[[i]] / (mu[[i]] - lambda[[i]])],
      If[FullB[[i]] ≠ 1 && EmptyB[[i]] ≠ 1 && G[[i]] == 1, dtime = Last[X][[i]] / (mu[[i]] - lambda[[i]])],
      If[FullB[[i]] ≠ 1 && EmptyB[[i]] ≠ 1 && G[[i]] == 0,
       dtime = (K[[i]] - Last[X][[i]]) / (lambda[[i]] - mu[[i]])],
      If[dtime < mindtime, {mindtime = dtime, imin = i}] }];

    (*updating X and time*)
    AppendTo[X, Last[X] + mindtime * Xdot];
    AppendTo[time, Last[time] + mindtime];
   }]
 }]

(*assigning colors to buffers: red for the full ones,
green for the empty ones and yellow for the remainging nodes*)
Color = Table[0, {Length[X]}, {NN}];
For[i = 1, i <= Length[X], i++,
  For[k = 1, k ≤ NN, k++, If[X[[i]][[k]] > (K[[k]] - s), Color[[i]][[k]] = Red,
    If[X[[i]][[k]] < s, Color[[i]][[k]] = Green, Color[[i]][[k]] = Orange]]];

(*plotting barchart*)
Manipulate[If[i > Length[X], i = Length[X], If[i < 1, i = 1]];
 BarChart3D[X[[i]], ChartStyle → Color[[i]], ChartLabels → Table[j, {j, NN}],
  PlotLabel → {{"time =", time[[i]]}}], {i, 1, Length[X], 1}]
```