QUEUEING NETWORKS WITH LIMITED FLEXIBILITY

Queueing Networks with Limited Flexibility

By
Liuxing Kan, B.Eng.

A Thesis
Submitted to the School of Graduate Studies
in Partial Fulfilment of the Requirements
for the Degree of

Master of Science

McMaster University

MASTER OF SCIENCE (2006) McMaster University
(Computing and Software) Hamilton, Ontario

TITLE: Queueing Networks with Limited Flexibility
AUTHOR: Liuxing Kan
B.Eng. (Wuhan University)
SUPERVISOR: Dr. George Karakostas
NUMBER OF PAGES: 1, 190

Abstract

Queueing network models have been widely adopted in the field of complex systems involving service. In this thesis, we study a queueing network model which consists of servers and classes with incoming customers. Customers are served by servers at classes, where a class of a customer is used to indicate the stage of processing. All the servers are flexible to switch their service between classes. Our objective is to choose an efficient assignment of servers to classes that maximizes the capacity of the given queueing network. By introducing limited flexibility, we restrict the maximum number of servers which can simultaneously work at a particular class and present a problem called the Total Discrete Capacity Constrained Problem (TDCCP). We also extend TDCCP to TDCCP with costs, where a cost is incurred when a server is working at a class.

We prove that both TDCCP and TDCCP with costs are NP-complete problems. However, for a special case where all servers are identical, we show that TDCCP and TDCCP with costs can be solved in polynomial time. Then we present approximation algorithms for another special case where all classes are identical. We also give approximation algorithms for solving the general case of TDCCP and TDCCP with costs.

Finally, we implement the approximation algorithms for solving TDCCP and TDCCP with costs. Numerical results on several experiments are reported. We compare and analyze the performance of the different algorithms. Several suggestions will be also given for choosing our algorithms and improving the results.

Acknowledgements

First of all, I would like to express my deep and sincere gratitude to my supervisor Dr. George Karakostas, for his constant support, insightful guidance and stimulating encouragement in all the time of two years' graduate studies at McMaster University. This thesis could not have been accomplished without his detailed comments, valuable suggestions and careful corrections. Honestly, I have learned a lot from him in both academic research and non-academic fields.

Thanks to Dr. Douglas Down and Dr. Spencer Smith, for their agreement to be my committee members and review this thesis. Your valuable suggestions and comments are highly acknowledged.

I thank all the professors and staffs in the Department of Computing and Software, especially Dr. Michael Soltys, who gave me a great lecture of computability and complexity.

I also thank Dr. Hu Zhang (Advanced Optimization Lab, McMaster University) for his helpful discussions on scheduling problems and all my colleagues in ITB-206 for the great working environment they made.

Finally, many thanks to my parents, who always supported me and encouraged me. Your love is the strong power for me to go forward. Also thanks to my girlfriend Emma, for the love and happiness you gave me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Queueing network models have many applications in the areas such as manufacturing, business, computer networks, and etc. First of all, take the example of a TV set factory. Various kinds of TV parts made in different workshops need to be assembled by workers; hence, the factory may set up a workstation for assembling. TV parts produced by other workshops may be sent to the assembling workstation with different arrival speeds. Workers in the workstation can work only on one assembly task at any time. When an assembly task is finished at the workstation, the assembled parts will be either sent to another workstation or sent out upon the completion of a whole TV set. There might exist a limit on the number of workers who can work at the same workstation due to specific regulations of the factory. Workers with different working experience may have different speeds of assembling. Therefore, if the manager of the factory wants to improve the daily production of TV sets, what strategy can he employ to make a more efficient assignment of workers?

Another example can be the Canadian immigration offices. Nowadays, the immigration department of Canada receives a flood of immigration applications; however, there is a limited number of immigration officers. After the applications are received, they are distributed to different case processing offices. If one case processing office finishes reviewing an application, the application is sent to another case processing office or is returned to applicants, if all the reviewing processes have been completed.

Every immigration officer can only process one application at a time. Officers can cooperate on the same application. But the number of officers working on the same application is limited because of the space of each immigration office. Similar to the last example, applications arriving in different offices may have different arrival rates and different officers may also have different case processing speeds due to their professional experience. Hence, how do we assign the officers to different offices more efficiently such that as many applications as possible can be processed everyday?

Furthermore, the above two examples can be extended to include the costs. In the first example, when workers are working in the factory, they are paid hourly. The cost of assembling a TV is the salaries paid to workers in the time of their work. The total incurred cost of assembling TVs in a day is the salaries paid to all the workers. The factory manager also has a budget that gives the maximum daily cost for assembly work. To improve the daily production of TVs, how can the manager make a more efficient assignment of workers while the daily cost of assembly does not exceed the given budget? Similar costs and budget can also be applied in the second example. When there is a budget for salaries of the immigration officers, how can we assign the officers to speed up processing applications while the given budget is still respected?

Similar models have arisen in other areas. Squillante, Xia, Yao and Zhang [SXYZ01] present examples in the area of parallel computer systems. F. S. Hillier and K. C. So [HS96] also propose a production line design problem in the area of production scheduling.

In this thesis, we study queueing network models which consists of servers and classes. A class of a customer is used to indicate the stage of processing. Customers will obtain service from servers at classes. When a customer has its service completed at a class, it can go to another class or leave the network. All the servers are flexible in our model, where we say that a server is flexible if it can switch its service between classes and the flexibility of a server is the number of classes which it can serve. We assume that each server can only work at one class at a time. However, several servers are capable of working at the same class. For a particular class, the maximum number of servers which can simultaneously work at this class is constrained. Let the service rate of a server be the mean number of customers it can serve in a unit of time and the customers arrival rate be the mean number of customers arriving at classes from outside the network in a unit of time. When the customers arrival rate at a class is

greater than the servers service rate at this class, network congestions will happen and customers at this class cannot be served in time. What can we do to maximize the customers arrival rate without any congestion in the network? Moreover, if the costs of servers working at classes and the budget of the maximum allowed total costs are defined, we will also consider the case for maximizing the customers arrival rate without any congestion and budget violation in the network. In the examples above, it is obvious that the workstations and immigration offices correspond to classes, the TV parts and applications correspond to customers, workers and immigration officers correspond to servers in our queueing network model.

In fact, most problems of queueing networks do require using servers with limited flexibility. Queueing networks with inflexibility (a server is forced to work at one class instead of several classes) or full flexibility (a server can work at any class) are just special cases of queueing networks with limited flexibility. Thus the study of queueing networks with limited flexibility will give us a more general way to explore the nature of queueing networks. In this thesis, we will build a mathematical model of queueing networks with limited flexibility and go through a thorough analysis of that model.

1.2 Previous work

In the area of operations research, earlier work on queueing networks studied work (resource) allocation problems and server (machine) allocation problems. Especially the work allocation problem has been studied extensively, e.g. knapsack problems and scheduling problems. However, the problem discussed in this thesis belongs to the server allocation problem.

F. S. Hillier and K. C. So [HS96] presented a production line model with a group of servers and a fixed amount of work to be partitioned among working stations. The objective in this paper is to maximize the throughput (production rate) of the given production line. They give numerical results for the case of a single server per station and the case of multiple servers per station. Several important observations are given based on their computational results. Their model is asking for a simultaneous assignment of servers and work to stations in the production line. Moreover, once the work is finished at a station, it cannot be sent to any other stations. This model is different from ours, since our model has interconnected stations (classes). But in

their work, they deal with flexibility, that is, multiple servers can work at a station.

M. S. Squillante, C. H. Xia, D. D. Yao and L. Zhang [SXYZ01] later proposed a parallel-server system model consisting of servers and job classes. Each job class has a queue for storing jobs from outside. When the servers are busy, the jobs will stay in the queue and wait for a server's execution. A threshold is assigned in each queue, and the threshold of a queue is the maximum number of jobs which can wait in this queue without processing by servers. When the number of jobs at a class is above the threshold of the queue, this queue is eligible to be processed by servers. Thus once a server is available for selecting the next job to process, it checks all the threshold values of queues and identifies qualified queues whose numbers of jobs are greater than their thresholds. A cost will be incurred when a job is waiting in the queue of a class. This paper develops threshold-based policies for allocating servers to those parallel queues such that the total incurred cost of this system is minimized.

S. Andradóttir, H. Ayhan and D. G. Down [AAD01] studied a dynamic server assignment to classes in order to obtain optimal (or near-optimal) throughput of queueing networks. Their model of queueing networks has common components (servers, classes and jobs) as the model of M. S. Squillante, C. H. Xia, D. D. Yao and L. Zhang [SXYZ01]. However, the classes in this model are assumed to be interconnected, where jobs can move from one class to another class upon the completion of the service. Also, servers in this model can travel between stations in a negligible amount of time and more than one servers can work at a class. The result of this paper shows that keeping all servers busy is very important to the maximization of the throughput.

In 2003, S. Andradóttir, H. Ayhan and D. G. Down [AAD03] showed a tight upper bound of the maximal capacity of queueing network models with flexible servers and constructed generalized round-robin policies for achieving a capacity arbitrarily close to the maximal capacity of a queueing network.

W. J. Hopp and M. P. van Oyen [HvO04] use the term cross-training instead of flexibility. The definitions of workers and workforce are also used instead of customers and servers. They refer to the overall framework as Agile Workforce Evaluation. In this framework, they outline approaches for accessing and classifying manufacturing and service operations. An extensive survey of the literature for workers' coordination and workforce's cross-training is also given.

J. G. Dai and W. Lin [DL05] proposed a family of policies called maximum pressure service policies for dynamically allocating service capacities in a stochastic network. Under some assumptions, they prove that both policies with flexible processors and policies with inflexible processors are throughput optimal. This paper also shows that those policies can be used in the queueing networks with interconnected stations.

To our knowledge, none of the previous work studies limited flexibility in queueing networks. This is the first attempt to address this problem. Actually, like the example of the TV set factory or Canadian immigration offices mentioned before, studying queueing networks with limited flexibility will help us to solve more general and practical problems in the area of queueing networks.

1.3 Model description

We consider a queueing network (henceforth we call it system) with servers and classes. A server will work at classes with given service rates. Customers with the arrival rates will enter the system from outside and be served at classes. Incoming customers will stay in the queues of classes to wait for service. The server assignment policy of queueing networks is the assignment of servers to classes. Our goal is to find an efficient server assignment policy such that the performance of the queueing network is maximized.

Many researchers use the throughput as a performance measure of queueing networks, e.g. S. Andradóttir, H. Ayhan and D. G. Down [AAD01], J. G. Dai and W. Lin [DL05], J. Ostalaza, J. McClain and J. Thomas [OMT90], L. Tassiulas and P. B. Bhattacharya [TB00]. The throughput of the queueing network is the number of customers which can be processed by this system under a specific server assignment policy in a unit of time. Therefore, if a queueing network has a larger throughput, it has a better performance. The maximal throughput of the queueing network is achieved when the number of customers processed by this system in a unit of time is maximized. The queueing network is stable if the customers' arrival rate is less than or equal to the maximal throughput of the system, which implies that all customers coming in a unit of time are processed by the system. Otherwise we say that this queueing network is unstable. Obviously, queue overflows will happen in an unstable queueing network with finite storage. In this thesis, we define the capacity of the

queueing network to be the arrival rate to this system so that the queueing network can be stabilized. Clearly, we are interested in computing the maximal capacity of a given queueing network. The way to increase the capacity of the queueing network is to find the a more efficient server assignment policy that increases the throughput of the system. We shall call the problem of maximizing the throughput of the queueing network with limited flexibility as the Total Discrete Capacity Constrained Problem (TDCCP). Further, if costs are incurred to the assignment of servers to classes and the total cost of the queueing network cannot exceed a given maximum budget, we extend the Total Discrete Capacity Constrained Problem (TDCCP) to TDCCP with costs. For more details of TDCCP and TDCCP with costs, please refer to Chapter 2.

When we are computing the maximal capacity of queueing networks, two important conditions must be satisfied. First, the queueing network must maintain stability; second, the limited flexibility of the queueing network cannot be violated.

Since the servers can switch their service among classes and several servers may work at a class in the same time, we define the total service rate of servers at a given class to be the number of customers that can be served at this class by servers in a unit of time. Similarly, the total arrival rate of customers at a given class is defined to be the number of customers arriving at this class from either outside of the network or from inside of the network in a unit of time. To maintain the stability of the queueing network, the total service rate of servers at each class must be greater than or equal to the total arrival rate of customers at this class. Thus, all customers coming to each class in a unit of time will be completely served by servers.

Another condition is the upper bound of the number of servers at every class. In other words, the number of servers assigned to a class cannot exceed the given maximum number.

The above conditions must be satisfied when we maximize the performance of queueing networks. More specifically, the objective of our problem is to find a server assignment policy without violating limited flexibility of queueing networks so that we can achieve the maximal capacity of the queueing network.

1.4 Our results

The most significant contribution of this thesis is the extension of the queueing network models in [AAD03] to the version with limited flexibility. By showing that queueing networks have a variety of applications in the areas of manufacturing, government administration, etc, we find that it is necessary to introduce flexibility to queueing networks. More importantly, the queueing networks with limited flexibility reflect more general models to the existing ones.

Our original queueing network model is a stochastic model. By Theorem 1 in [AAD03], we show that the stochastic optimization problem of the queueing network model can be converted into a deterministic optimization problem called the Total Discrete Capacity Constrained Problem (TDCCP). The solution of the deterministic problem TDCCP can be easily mapped to a solution of the original queueing network model. This result gives us a strong connection between maximizing capacity in a queueing network and the problem TDCCP. In TDCCP, the customers' arrival rates at classes are all given and each server has a fixed service rate at each class. The total associated service rate at a class depends on the proportion of time that each server is working at this class. To maintain the network stability, the total associated service rate at each class must be at least the customers' arrival rate at this class. Furthermore, there also exists limited flexibility which is a constraint on the number of servers which can work at a class in a unit of time. The objective of TDCCP is to compute the proportion of time of servers to classes so that the maximal capacity (the maximal customers' arrival rate to the system) is achieved and limited flexibility is respected.

TDCCP will also be extended to TDCCP with costs. In this problem, we define the cost per unit of proportion of time when servers are working at classes. The cost of a server is the sum of incurred cost of its assignment at all classes. The total cost in the system cannot exceed a given budget, which is the maximum allowed total cost in the system. By finding an optimal server assignment policy of the queueing network without violating the budget, the maximal capacity of the queueing network will be achieved and the throughput will be maximized. We give the detailed definitions of TDCCP and TDCCP with costs, as well as their mathematical models in the form of optimization problems. These mathematical models will bring us a useful and

quantitative way for further analyzing TDCCP and TDCCP with costs.

Our objective is to compute the maximal capacity for a given queueing network with limited flexibility. However, a result of this thesis is that the general case of TDCCP (with costs) is an NP-Complete problem. Since it is unlikely to find exact polynomial time algorithms for solving it exactly (unless P=NP), we try to develop approximation algorithms for it. We evaluate an approximation algorithm by its approximation factor. If the approximate solution is at least ρ ($\rho \leq 1$) times the optimum solution, we say that this approximation algorithm is a ρ-approximation algorithm with approximation factor ρ. The approximation factor is also called the relative performance guarantee. This thesis will be dedicated to designing polynomial time approximation algorithms for TDCCP (with or without costs).

Although the general case of TDCCP is NP-Complete, for the special case where service rates are independent of all servers (all servers are identical), we show that TDCCP can be solved in polynomial time. Moreover, we present a mathematical formula for directly computing the assignment of servers that achieves the maximal capacity of a given queueing network in this special case.

Another important case of the general TDCCP is the special case where service rates depend only on the servers (all classes are identical). We prove the NP-Completeness for this special case. Thus seeking good approximation algorithms for this special case will be our key issue.

To design approximation algorithms for the case of TDCCP where service rates depend only on servers, we first study a problem called the maximum concurrent multicommodity k-splittable flow problem which is generated from the maximum concurrent multicommodity flow problem in [SM90] and the k-splittable flow problem in [BKS02]. Given a network with capacitated edges, a source node and sink nodes, there are commodities that need to be sent from the source node to their sink node. The maximum concurrent multicommodity flow problem maximizes the factor by which we can multiply all demands and still achieve a feasible multicommodity flow without violating the given edge capacities. The k-splittable flow problem is the multicommodity flow problem in which the number of different paths for sending each commodity is bounded by a given number k. When $k = 1$, we call it the unsplittable flow (instead of 1-splittable flow) problem. We say that k is the splittability bound of each commodity. By a simple construction, we show that the case of TDCCP where

service rates depend only on the servers can be viewed as an instance of the maximum concurrent multicommodity k-splittable flow problem. Thus, by utilizing the ideas of algorithms in [BKS02], [DGG99] and [KS02], we present an approximation algorithm (Algorithm 1) based on solving the maximum concurrent multicommodity k-splittable flow problem. Algorithm 1 contains two approximation stages: During the first stage, instead of solving the maximum concurrent multicommodity k-splittable flow problem, we transform the TDCCP to the maximum concurrent multicommodity uniform exactly-k-splittable flow problem, in which all commodities are sent through exactly k paths and each path carries the same flow amount; during the second stage, we approximate the maximum concurrent multicommodity uniform exactly-k-splittable flow problem by solving the unsplittable flow problem (only one path allowed for sending each commodity) where we split each commodity into k sub-commodities and send each sub-commodity unsplittably. We prove that the approximation factor of this two-stage algorithm is $1/10$, which guarantees that we can achieve at least $1/10$ times the optimum capacity for this case.

For the second stage of Algorithm 1, we find another way to achieve its goal. To employ this different method, we study the scheduling of unrelated machines problem, which can be stated as follows: We consider a system with parallel machines and independent jobs. The processing time of scheduling a job on a machine is given. We define the total processing time of a machine to be the sum of processing times of all jobs which are scheduled on this machine. The makespan of a schedule is the maximum total processing time of any machine in the system. The objective of the scheduling is to find a schedule that minimizes the makespan. If we treat the sub-commodities as the jobs, we prove that the maximum concurrent multicommodity uniform exactly-k-splittable flow problem can be transformed to the scheduling of unrelated machines problem. Then a different approximation algorithm (Algorithm 2) based on an algorithm for solving (approximately) the scheduling of unrelated machines problem in [LST90] is proposed. Algorithm 2 has two stages, where the first stage is the same as Algorithm 1 for transforming TDCCP to the maximum concurrent multicommodity uniform exactly-k-splittable flow problem, and the second stage solves the maximum concurrent multicommodity uniform exactly-k-splittable flow problem by using the algorithm for the scheduling of unrelated machines problem. Let the maximal capacity of the fractional TDCCP be λ^*, where the fractional

TDCCP means that we have the full server flexibility at all classes. We define the customers' arrival rate at class k to be a_k and the server flexibility at class k to be c_k. Since in this case the service rates depend only on servers, we let the service rate of server j at all classes be μ_j. We show that Algorithm 2 can produce the throughput at least $1/2(1+\rho)$ of the optimum, where $\rho := \max_{j,k} \frac{a_k \lambda^*}{c_k \mu_j}$.

We analyze the problem TDCCP with costs in a way similar to the problem TDCCP. In this problem, we are given a budget and the cost per unit of assignment when servers are working at classes. The total cost of the queueing network is the sum of incurred cost of each server's assignment at all classes. Compared with TDCCP, there is another constraint in the system that says the total cost of the queueing network cannot exceed the budget we have. Obviously, TDCCP without costs is just a special case of TDCCP with costs by setting the budget and given costs in the queueing network to be zero. Thus it is easy to draw the conclusion that the general case of TDCCP with costs is also an NP-Complete problem.

We show that the case of the general TDCCP with costs where service rates are independent of all servers (all servers are identical) is polynomially solvable. Furthermore, we find a mathematical formula for computing the maximal capacity of the queueing network in this special case.

The case of the general TDCCP with costs where service rates depend only on the servers (all classes are identical) is also considered. Since this case contains the general TDCCP without costs where service rates depend only on the servers as a special case (by setting the budget and all the costs to zero), we see that it is also an NP-Complete problem. Two different approximation algorithms are designed for solving this special case, which correspond to Algorithms 1 and 2 above.

We present the constrained maximum concurrent multicommodity k-splittable flow problem, which is the cost version of the maximum concurrent multicommodity k-splittable flow problem. Similar to the maximum concurrent multicommodity k-splittable flow problem, we have a network with capacitated edges, a source node and sink nodes. There are commodities that need to be sent from the source node to their sink node. We define the cost per unit of flow on each edge. When the commodities go through an edge, the incurred cost of this edge will be the flow amount multiplied by the given cost per unit of flow. Thus the total cost of the network is the sum of costs on all edges. A budget of network is also given, which is the maxi-

mum allowed total cost in the network. Thus the constrained maximum concurrent multicommodity k-splittable flow problem is to find paths for sending all commodities concurrently without any edge capacity violation or splittability bound violation so that the maximum possible faction λ of all commodity demands is achieved simultaneously and the total cost of the network respects the budget constraint. We transform the special case of TDCCP where service rates depend only on the servers to an instance of the constrained maximum concurrent multicommodity k-splittable flow problem defined above. Therefore, the algorithms in [BKS02] and [Sku02] can be used to design an approximation algorithm (Algorithm 3) for solving this case of TDCCP with costs. Similar to Algorithm 1, Algorithm 3 is designed as follows: We first transform the TDCCP with costs to the constrained maximum concurrent multicommodity uniform k-splittable flow problem where all commodities are sent through exactly k paths with the same flow amount on each path; then we split each commodity into k sub-commodities and find the unsplittable flow for each sub-commodity without violating the given cost budget in the network. The approximation factor of Algorithm 3 is $1/12$, which guarantees that at least $1/12$ times the maximal capacity will be achieved for TDCCP with costs.

We also show that the second stage in Algorithm 3 can be viewed as the scheduling of unrelated machines problem with costs, which is a more general scheduling model compared with the model in [LST90]. In this problem, we are also given the cost per unit of assignment of servers to jobs and a budget of the maximum allowed total cost in the system. The total cost in this system is the sum of all the costs incurred in the servers' assignments. Again, the objective of the problem is to minimize the makespan while the total cost respects the given budget. We demonstrate a transformation from this case of TDCCP with costs where service rates depend only on the servers to the scheduling of unrelated machines problem with costs, and design an approximation algorithm (Algorithm 4) for solving TDCCP with costs. Algorithm 4 also has two stages: The first stage is the same as Stage 1 in Algorithm 3, where we solve the constrained maximum concurrent multicommodity uniform k-splittable flow problem; in the second stage, we transform the constrained maximum concurrent multicommodity uniform k-splittable flow problem to the scheduling of unrelated machines problem with costs and find its integer solution approximately. Let the maximal capacity of the fractional TDCCP be λ^*, the customers' arrival rate

at class k be a_k, the server flexibility at class k be c_k and the service rate of server j at all classes be μ_j. Similar to Algorithm 2, we show that Algorithm 4 will also produce throughput of at least $1/2(1+\rho)$ of the optimum, where $\rho := \max_{j,k} \frac{a_k \lambda^*}{c_k \mu_j}$.

We also design approximation algorithms for solving the general case of TDCCP (with or without costs). For the general TDCCP without costs, the algorithm includes Algorithm 1 or Algorithm 2 as a subroutine. For the general TDCCP with costs, the algorithm applies Algorithm 3 or Algorithm 4 as a subroutine. Thus, the theoretical approximation factor of the general approximation algorithm varies with the subroutines we use.

To test the efficiency of our algorithms, we implement in MATLAB the 4 algorithms for solving the special case of TDCCP (with costs) where service rates depend only on the servers. Further, algorithms for solving the general cases of TDCCP (with costs) are also implemented. The numerical results of several experiments demonstrate that in most cases, our algorithms can achieve very good approximate solutions compared with the optimum. However, in some general cases of TDCCP (with costs), our algorithms might produce bad results.

Finally, we mention that the algorithms in our thesis can be applied to solve other models in [AAD03], [AAD01], [HS96] and [DL05]. The model in [AAD03] and [AAD01] is actually the queueing network model with full flexibility, where there is no limit on the number of servers at any class. For the model in [HS96], the fixed amount of work can be treated as the work amount in one period of time in their production line system. In the case that we have a work allocation to classes, their problem of maximizing throughput is exactly the same as TDCCP with full flexibility. The models of stochastic processing networks in [DL05] have two different versions which are the preemptive server processing and the non-preemptive server processing. Their model of non-preemptive server processing is similar to our queueing network model. In this case, our algorithms can also be applied.

1.5 Thesis structure

This thesis is organized as follows.

Chapter 2 presents the detailed definitions and mathematical models for the Total Discrete Capacity Constrained Problem.

Chapter 3 presents several important network flow and scheduling problems which are related to this thesis.

Chapter 4 discusses the TDCCP without costs. We consider two special cases: The first one is the case when all servers are identical and the second one is the case when all classes are identical. We show that the first case can be solved in polynomial time and present two approximation algorithms for solving the second case. We also present the approximation algorithms for solving the general TDCCP without costs. Finally, we prove the NP-Completeness of TDCCP without costs.

Chapter 5 discusses the TDCCP with costs. We show that TDCCP with costs can be solved in polynomial time when all servers are identical. Two approximation algorithms are designed for solving TDCCP with costs when all classes are identical. We also present the approximation algorithms for solving the general TDCCP with costs.

Chapter 6 reports the testing results. We have tested 40 examples for each algorithm in this thesis. The detailed computational results are given. We discuss and analyze the testing results. In the end of this thesis, we give the summary of our experiments.

Chapter 2

Total Discrete Capacity Constrained Problem

In this chapter we present the Total Discrete Capacity Constrained Problem (TDCCP) for maximizing the capacity of the queueing network. Our queueing network model is derived from the model of S. Andradóttir, H. Ayhan and D. G. Down [AAD03]. Thus, we will first study their model in its entirety and introduce the limited flexibility. Then, detailed definitions and the mathematical definition of TDCCP will be given. We further establish the connection between the deterministic problem TDCCP and the original stochastic queueing network problem and show that the solution of TDCCP can be mapped to the solution of the queueing network model. Finally, we extend TDCCP to TDCCP with costs and give its mathematical definition.

2.1 Overview of the network

In our queueing network, the basic nodes are classes where servers provide service to customers. Servers can either work in parallel or work cooperatively at a class. Since the customers may be transferred from one class to another or be sent out upon completion of service, there are connections among classes, which denote the probabilities of switching. Originally, all customers come from outside. For convenience, we can set up a source node for sending new customers and a sink node for receiving

all customers that have been exited the network.

2.2 Network topology

Suppose there are K classes in the network. The classes in the queueing network indicate the stages of customers' processing status. Customers are served by servers in parallel. We do not allow that servers can pool their efforts on one customer when they are working together at a particular class. Each class has a buffer of infinite size. Thus if all servers are currently busy, new customers that enter a class must stay in the buffer and wait for service.

At any class k, customers enter k either from outside of the network or from some other class i. Let the probability that an arrival from outside of the network is routed to class k be $p_{0,k}$. We assume that each customer must be served before leaving the network, and this is equivalent to $\sum_{k=1}^{K} p_{0,k} = 1$. We assume that the arrival process of customers from outside of the network has independent and identically distributed (i.i.d) interarrival times $\{\xi(n)\}$, where $\xi(n)$ is the interarrival time of the $(n-1)^{th}$ and n^{th} customer arriving from outside of the network. By this assumption we know that the random variables $\{\xi(n)\}$ are all mutually independent and the probability distributions of $\{\xi(n)\}$ are identical. When customers complete service at class i, the probability of customers entering class k is $p_{i,k}$. Thus the customers exiting class i will have probability $1 - \sum_{k=1}^{K} p_{i,k}$ of leaving the network. We also assume that all the customers will leave the network eventually. Furthermore, by defining a matrix P to have (i,k) entry $p_{i,k}$ for $i,k = 1,...,K$ and I to be the $K \times K$ identity matrix, if $(I - P')$ is invertible (non-singular), then all the customers will leave the network eventually.

Since the interarrival times $\{\xi(n)\}$ have the same probability distribution, we will have the same expectation of each variable in $\{\xi(n)\}$. This can be written as follows,

$$E(\xi(1)) = E(\xi(2)) = E(\xi(3)) = ... = E(\xi(n)).$$

Note that the expectation of the customers' interarrival time $E(\xi(n))$ is the mean interarrival time of two successively incoming customers. Thus we define the associated arrival rate of customers from outside to be λ, where $\lambda = 1/E[\xi(1)]$. Let the total arrival rate to class k be λ_k. We know that the arrivals of customers at class

k are from either outside of the network or other classes. Given any class i, when customers at class i finish their service, they are going to enter class k with probability $p_{i,k}$. Therefore, the arrivals from class i to k have the rate $p_{i,k}\lambda_i$. Similarly, the rate of arrivals from outside of the network to class k is defined to be $p_{0,k}\lambda$. Suppose the network is stable with a given arrival rate λ. The total arrival rates at class k is $\lambda_k = p_{0,k}\lambda + \sum_{i=1}^{K} p_{i,k}\lambda_i$. Given $p_{0,k}$ and $p_{i,k}$ for any k and i, we can solve those K equations and get the unique solution λ_k. Thus, the customers' arrival rate at each class can be computed.

For technical reasons, we assume that the interarrival times of customers are unbounded and spread out. This technical assumption is needed for describing the system as a Markov process with a reachable origin. The Markov process is a stochastic process where the future of the process depends only upon the present state of it and this implies that the present state is a direct result of its history. For more details, see Appendix A in [AAD03].

2.3 Service mechanism

In the queueing network, we also have M servers which provide service to customers at classes. All servers use the First Come, First Served order (FCFS). When server j finishes its service at class i and switches to class k for the n^{th} time, it incurs a switching time $\zeta_{i,k}^{j}(n)$ (possibly zero). We assume that the switching times $\{\zeta_{i,k}^{j}(n)\}$ are i.i.d. for every $j = 1, ..., M$, $k = 1, ..., K$. Also, we assume that the server switch time exists only between two different classes, which means that $\{\zeta_{i,i}^{j}(n)\}$ is identically zero for all i and j. Several servers may be simultaneously working at a class. We define $\eta_{j,k}(n)$ to be the service time of the n^{th} customer served by server j at class k. Furthermore, if we assume that the service times $\{\eta_{j,k}(n)\}$ are i.i.d., then $\{\eta_{j,k}(n)\}$ have the same probability distribution, therefore $E(\eta_{j,k}(1)) = E(\eta_{j,k}(2)) = ... = E(\eta_{j,k}(n))$. Thus, the associated service rate of server j working at class k is defined to be $\mu_{j,k} = 1/E[\eta_{j,k}(1)]$. If server j cannot work at class k, we set $\mu_{j,k} = 0$.

If server j spends any time at class k, we say that server j is assigned to class k. We also define $\delta_{j,k}$ to be the long run average proportion of time that server j is working at class k in a unit of time. We say that $\delta_{j,k}$ is the server assignment policy for the queueing network.

We know that there are M servers in the queueing network. The difference between the model in this thesis and the model in [AAD03] is that we have limited flexibility described by c_k, where we define $c_k \leq M$ to be the maximum number of servers that can be assigned to class k in a unit of time.

The following figure shows an example of our model with two servers and three classes.

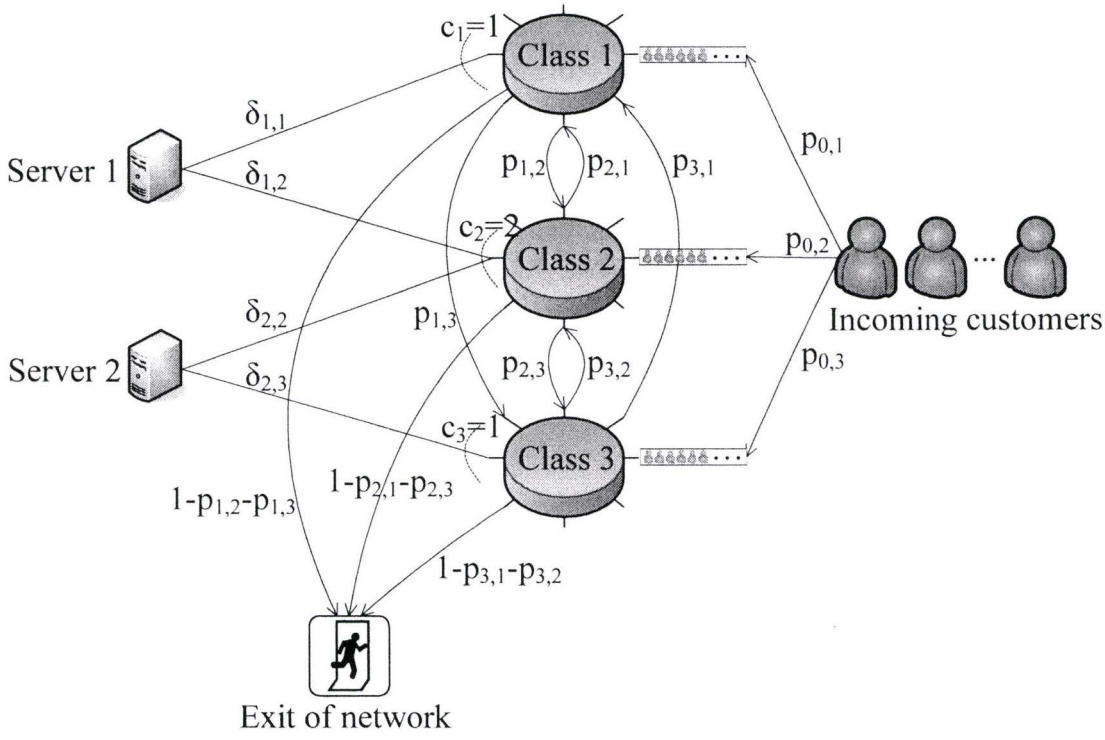


Figure 2.1: A queueing network model with 2 servers and 3 classes

2.4 Total Discrete Capacity Constrained Problem

To maximize throughput of the original queueing network, we introduce a problem called the Total Discrete Capacity Constrained Problem (TDCCP). The objective of TDCCP is to find a server policy to achieve the maximal capacity of the given

queueing network. Given the probabilities of customer arrivals (from either outside or inside network), we have the following K equations,

$$\lambda_k = p_{0,k}\lambda + \sum_{i=1}^{K} p_{i,k}\lambda_i, k = 1, ..., K.$$

We assume that the network is stable with a given arrival rate λ. Then, solving these K equations will give us a unique solution of each λ_k for $k = 1, ..., K$. Let $\lambda_k = \lambda \cdot a_k$ be the solution, where a_k is the solution of these equations with $\lambda = 1$. By this step, the customer arrival rate at class k can be expressed in terms of the arrival rate λ and the constant a_k. Then, the input of TDCCP are the service rates $\mu_{j,k}$, flexibility c_k and a_k while the output is the server assignment $\delta^*_{j,k}$ and maximal capacity λ^*. We formulate TDCCP as the following optimization problem with variables $\delta_{j,k}$ and λ:

$$\begin{aligned}
\max \quad & \lambda && (MP)\\
\text{s.t.} \quad & \sum_{j=1}^{M} \mu_{j,k}\delta_{j,k} \geq \lambda a_k, k = 1, ..., K && (1)\\
& \sum_{k=1}^{K} \delta_{j,k} \leq 1, j = 1, ..., M && (2)\\
& \sum_{j=1}^{M} \chi\{\delta_{j,k} > 0\} \leq c_k, k = 1, ..., K && (3)\\
& \delta_{j,k} \geq 0, k = 1, ..., K, j = 1, ..., M. && (4)
\end{aligned}$$

Constraint (1) says the sum of the associated service rate at class k is greater than or equal to the total customer arrival rate at class k, which will guarantee the stability of the original queueing network. Constraint (2) guarantees that the sum of proportions of time of server j working at all classes cannot exceed 1 so that no server in the system is overloaded. Constraint (3) is the flexibility limit where we do not allow the number of servers working at class k to be more than c_k. Constraint (4) says that a negative proportion $\delta_{j,k}$ is not permitted in the network. Given a class k, $\chi\{\cdot\}$ is the indicator function that indicates the membership of $\delta_{j,k}$ in the set $\{\delta_{j,k} > 0\}$. Let the optimal solution of (MP) be λ^* and $\{\delta^*_{j,k}\}$. We will see that $\{\delta^*_{j,k}\}$ is the

set of proportional assignments of servers to classes required to achieve the maximal capacity λ^*. Note that the solution $\{\delta^*_{j,k}\}$ may not be unique.

Let $Q_k(t)$ be the number of customers at class k present at time t and $Q(t)$ be a vector with kth entry $Q_k(t)$. The following theorem gives us the connection between maximizing capacity in the queueing network and the problem (MP) stated above.

Theorem 2.1. ***(i)*** *Any capacity less than λ^* may be achieved. More specifically, for an arrival process with rate $\lambda < \lambda^*$, there exists a dynamic server assignment policy such that the distribution of the queue length process $\{Q(t)\}$ converges to a steady-state distribution φ as $t \to \infty$.*
(ii) *A capacity larger than λ^* cannot be achieved. More specifically, for an arrival process with rate $\lambda > \lambda^*$, as $t \to \infty$,*

$$P(|Q(t)| \to \infty) = 1.$$

Theorem 2.1 is a trivial extension of Theorem 1 in [AAD03]. It says that the difficult stochastic queueing network problem can be converted into the deterministic optimization problem (MP). By solving the deterministic problem (MP), we can map its solution back to the solution of the original stochastic queueing network problem. This can be done by the generalized round-robin policies in [AAD03]. Suppose the optimal solution of (MP) is λ^* and $\delta^*_{j,k}$. Taking λ^* and $\delta^*_{j,k}$ as input, the generalized round-robin policies will output the servers' service policies in terms of service time of servers at each class. We know that the optimal server policy $\delta^*_{j,k}$ satisfies the flexibility constraint in (MP). According to the fact that the generalized round-robin policies will not assign server j to class k if $\delta^*_{j,k} = 0$ (e.g. see the server assignment algorithm in [AAD03]), it is easy to see that the flexibility of the original queueing network is still respected. In the remainder of this thesis, we will focus on solving the problem (MP). For the detailed proof of Theorem 2.1 and a discussion of generalized round-robin policies, refer to [AAD03].

There are two important special cases of (MP). We formulate the two cases as

mathematical programs (MP1) and (MP2).

$$\begin{aligned}
\max \quad & \lambda && (MP1)\\
\text{s.t.} \quad & \sum_{j=1}^{M} \mu_k \delta_{j,k} \geq \lambda a_k, k = 1, ..., K && (1)\\
& \sum_{k=1}^{K} \delta_{j,k} \leq 1, j = 1, ..., M && (2)\\
& \sum_{j=1}^{M} \chi\{\delta_{j,k} > 0\} \leq c_k, k = 1, ..., K && (3)\\
& \delta_{j,k} \geq 0, k = 1, ..., K, j = 1, ..., M. && (4)
\end{aligned}$$

In this case, the service rates are independent of the server (all servers are identical), i.e., $\mu_{j,k} = \mu_k$ for all j in (MP1). The constraints are the same as the constraints in (MP) except that we have μ_k instead of $\mu_{j,k}$ in (1). In Chapter 4, we will show that the maximal capacity of the queueing network in this case can be computed in polynomial time.

$$\begin{aligned}
\max \quad & \lambda && (MP2)\\
\text{s.t.} \quad & \sum_{j=1}^{M} \mu_j \delta_{j,k} \geq \lambda a_k, k = 1, ..., K && (1)\\
& \sum_{k=1}^{K} \delta_{j,k} \leq 1, j = 1, ..., M && (2)\\
& \sum_{j=1}^{M} \chi\{\delta_{j,k} > 0\} \leq c_k, k = 1, ..., K && (3)\\
& \delta_{j,k} \geq 0, k = 1, ..., K, j = 1, ..., M. && (4)
\end{aligned}$$

In this case, the service rate depends only on the server (all classes are identical), i.e., $\mu_{j,k} = \mu_j$ for all k. Constraints (1), (2), (3) and (4) are the same as the corresponding constraints in (MP). In Chapter 4, we will prove that this case of (TDCCP) is

NP-Complete. Two approximation algorithms will be designed for solving (MP2) approximately. Moreover, this case is important because the approximation algorithm for solving (MP) is actually based on the algorithms for solving (MP2).

2.5 Total Discrete Capacity Constrained Problem with Costs

In the original queueing network model, if we are given a cost per unit of server assignment and a budget of the queueing network, we call it the queueing network model with costs. The objective of this problem is to find a server assignment policy so that the maximal throughput of the queueing network is achieved and the total cost of the system respects the given budget. We define the cost per unit of server assignment to be $r_{j,k}$ when server j is working at class k and $\delta_{j,k}$ to be the proportion of time that server j is working at class k. Then it is easy to see that the incurred cost of server j at class k is $r_{j,k}\delta_{j,k}$ and the total cost of the system can be written as $\sum_{j=1}^{M}\sum_{k=1}^{K} r_{j,k}\delta_{j,k}$. We are also given a nonnegative budget C. Thus to respect the given budget C, we have the following additional constraint:

$$\sum_{j=1}^{M}\sum_{k=1}^{K} r_{j,k}\delta_{j,k} \leq C.$$

To decide the server assignment $\delta_{j,k}$ for maximizing the capacity λ, we are given the service rate $\mu_{j,k}$ when server j is working at class k, the number a_k which is used to derive the customers' arrival rate at class k, and servers' flexibility c_k at class k. By adding the above cost constraint, we formulate TDCCP with costs as follows,

$$\max \quad \lambda \qquad (MPC)$$

$$\text{s.t.} \quad \sum_{j=1}^{M} \mu_{j,k}\delta_{j,k} \geq \lambda a_k, k = 1, ..., K \qquad (1)$$

$$\sum_{k=1}^{K} \delta_{j,k} \leq 1, j = 1, ..., M \qquad (2)$$

$$\sum_{k=1}^{K}\sum_{j=1}^{M} r_{j,k}\delta_{j,k} \leq C \qquad (3)$$

$$\sum_{j=1}^{M} \chi\{\delta_{j,k} > 0\} \leq c_k, k = 1, ..., K \qquad (4)$$

$$\delta_{j,k} \geq 0, k = 1, ..., K, j = 1, ..., M. \qquad (5)$$

In (MPC), constraints (1), (2), (4) and (5) are exactly the same as the corresponding constraints in (MP). Constraint (3) says the total cost cannot exceed the budget C. Let the optimal solution of (MPC) be λ^* and $\{\delta^*_{j,k}\}$. Since the server assignment $\{\delta^*_{j,k}\}$ is the optimal assignments of servers to classes, it is required to achieve the maximal capacity λ^*. However, the solution $\{\delta^*_{j,k}\}$ is not necessarily unique.

Theorem 2.1 can also be applied to establish the connection between maximizing the throughput in the queueing network with costs and the solution to (MPC). Given the maximal capacity λ^* and server assignments $\{\delta^*_{j,k}\}$ respecting the given budget C, the generalized round-robin policies in [AAD03] can be employed to get arbitrarily close to the server assignments $\{\delta^*_{j,k}\}$. Note that in their construction, the assignments of servers to classes do not increase, thus the budget in the queueing network will not be violated. Then by Theorem 2.1, we know that the stochastic queueing network problem with costs can be converted into the deterministic optimization problem of TDCCP with costs. Also, the solution of TDCCP with costs can be directly mapped back to a solution of the original queueing network problem with costs using generalized round-robin policies. In the remainder of this thesis, we will also focus on solving the problem TDCCP with costs.

We consider the mathematical model (MP) for TDCCP and (MPC) for TDCCP

with costs. It is easy to see that (MP) is just a special case of (MPC) by setting $r_{j,k} = 0$. Then constraint (3) becomes trivial for any nonnegative budget C.

The difficulty in solving the problem (MP) and (MPC) comes from the integral constraints. In these constraints, although the server assignments $\delta_{j,k}$ can be fractional, the corresponding counted flexibility will be either 0 or 1 (depending on whether $\delta_{j,k}$ is 0 or not). Without the integral constraint (4), (MP) and (MPC) can be easily seen to be linear programming problems. However, in this thesis, we will show that even a special case of (MP) is NP-Complete, hence the problem TDCCP is NP-Complete. Since TDCCP is a special case of TDCCP with costs, we can conclude that TDCCP with costs is also an NP-Complete problem. In Chapter 4, we show two approximation algorithms for solving (MP), and in Chapter 5, we propose two different approximation algorithms for solving (MPC). Obviously, any algorithm for solving (MPC) can also be applied to solve the problem (MP), since (MP) is a special case of (MPC).

Similar to TDCCP, we will study two important special cases of TDCCP with costs, which are formulated as follows:

$$\begin{aligned}
\max \quad & \lambda && (MPC1)\\
\text{s.t.} \quad & \sum_{j=1}^{M} \mu_k \delta_{j,k} \geq \lambda a_k, k = 1, ..., K && (1)\\
& \sum_{k=1}^{K} \delta_{j,k} \leq 1, j = 1, ..., M && (2)\\
& \sum_{k=1}^{K} \sum_{j=1}^{M} r_{j,k} \delta_{j,k} \leq C && (3)\\
& \sum_{j=1}^{M} \chi\{\delta_{j,k} > 0\} \leq c_k, k = 1, ..., K && (4)\\
& \delta_{j,k} \geq 0, k = 1, ..., K, j = 1, ..., M. && (5)
\end{aligned}$$

In this case, the service rates are independent of the server (all servers are identical), i.e., $\mu_{j,k} = \mu_k$ for all j in (MPC1). Except for $\mu_{j,k} = \mu_k$ in constraint (1), all other constraints in (MPC1) are actually the same as the constraints in (MPC). In

Chapter 5, we will show that the maximal capacity λ^* in (MPC1) can be computed in polynomial time.

$$\max \quad \lambda \qquad (MPC2)$$

$$\text{s.t.} \quad \sum_{j=1}^{M} \mu_j \delta_{j,k} \geq \lambda a_k, k = 1, ..., K \qquad (1)$$

$$\sum_{k=1}^{K} \delta_{j,k} \leq 1, j = 1, ..., M \qquad (2)$$

$$\sum_{k=1}^{K} \sum_{j=1}^{M} r_{j,k} \delta_{j,k} \leq C \qquad (3)$$

$$\sum_{j=1}^{M} \chi\{\delta_{j,k} > 0\} \leq c_k, k = 1, ..., K \qquad (4)$$

$$\delta_{j,k} \geq 0, k = 1, ..., K, j = 1, ..., M. \qquad (5)$$

In (MPC2), the service rate depends only on the server (all classes are identical), i.e., $\mu_{j,k} = \mu_j$ for all k. The constraints in (MPC2) are the same as the corresponding constraints in (MPC). We will see that the problem (MPC2) is an NP-Complete problem since it contains (MP2) as a special case (by setting $r_{j,k} = 0$ for all j, k and $C = 0$). In Chapter 5, we will design two approximation algorithms for solving (MPC2). The approximation algorithm for solving the general (MPC) will use one of the approximation algorithms for solving (MPC2).

Chapter 3

Network Flow and Scheduling Problems

In previous chapters, we have shown that the original queueing network can be converted to the deterministic problems of TDCCP and TDCCP with costs. The algorithms for solving TDCCP and TDCCP with costs are based on the transformations of TDCCP and TDCCP with costs to network flow and scheduling problems. Before showing the detailed algorithms, we will study those important network flow and scheduling problems.

3.1 Multicommodity Flow Problem

In the multicommodity flow problem, we are given a directed or undirected graph $G = \langle V, E \rangle$ where V is a set of nodes and E is a set of edges. Each edge $e \in E$ has capacity u_e. There are n different commodities which co-exist in G and need to be sent from source nodes to terminal nodes. We define the source node to be s_i and the terminal node to be t_i for sending and receiving the commodity i, $i = 1, ..., n$, respectively. Then, a multicommodity flow on G can be represented as a set of node pairs $(s_1, t_1), (s_2, t_2), ..., (s_n, t_n)$, where the pair (s_i, t_i) denotes the commodity i from the source node s_i to the terminal node t_i. Let f_e denote the flow on the edge e, $e \in E$. There are mainly two optimization problems on multicommodity flow relevant to our work. One is the maximum multicommodity flow problem and the other one is the

maximum concurrent multicommodity flow problem,

First we study the maximum multicommodity flow problem. The objective of the maximum multicommodity flow problem is to send n commodities while respecting edge capacities so that the sum of all commodity flows is maximized. We define P_{s_i,t_i} to be the set of paths for sending commodity i from source s_i to terminal t_i, $i = 1, ..., n$. The maximum multicommodity flow problem can be formally written as follows,

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} f_{(s_i,t_i)} \\ \text{s.t.} \quad & \sum_{p \in P_e} f_p \leq u_e, \forall e \in E \\ & f_p \geq 0, \forall p \in P_{s_i,t_i}, i = 1, ..., n \end{aligned}$$

where $f_{(s_i,t_i)}$ denotes the flow amount of commodity i from s_i to t_i, P_e denotes the set of paths going through the edge e and f_p denotes the flow amount in the path where $p \in P_{s_i,t_i}$. Obviously, $\sum_{i=1}^{n} f_{(s_i,t_i)}$ is the total multicommodity flows we send.

Farhad Shahrokhi and D. W. Matula [SM90] proposed the maximum concurrent multicommodity flow problem, which is another version of the multicommodity flow problem. Here, for each commodity i, we are given a demand d_i at the terminal node t_i. The objective of the maximum concurrent multicommodity flow problem is to maximize the factor by which we can multiply all terminal node demands and still achieve a feasible multicommodity flow without violating the given edge capacities. More specifically, if we define a variable β and set all terminal node demands d_i ($i = 1, ..., n$) to be βd_i, the maximum concurrent multicommodity flow problem maximizes the number β while respecting all edge capacities in G, so that all the demands βd_i are satisfied simultaneously. We consider the following example of the multicommodity flow.

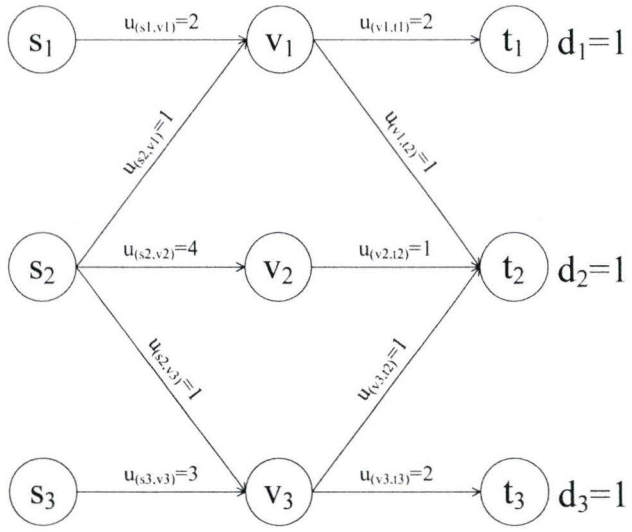


Figure 3.1: An example of the multicommodity flow problem with 3 commodities

In this example, the maximum possible fraction of routing all commodities simultaneously is 2 by sending 2 units of commodity 1 following the path $(s_1 \rightarrow v_1 \rightarrow t_1)$, 1 unit of commodity 2 following the paths $(s_2 \rightarrow v_1 \rightarrow t_2)$ and $(s_2 \rightarrow v_2 \rightarrow t_2)$ and 2 units of commodity 3 following the path $(s_3 \rightarrow v_3 \rightarrow t_3)$.

Let P_e denotes the set of paths going through the edge e. Given a path $p \in P_{s_i,t_i}$, f_p denotes the flow amount in the path. We can formulate the maximum concurrent multicommodity flow problem as follows,

$$\begin{aligned} \max \quad & \beta \\ \text{s.t.} \quad & \sum_{p \in P_{s_i,t_i}} f_p \geq \beta d_{t_i}, i = 1, ..., n \\ & \sum_{p \in P_e} f_p \leq u_e, \forall e \in E \\ & f_p \geq 0, \forall p \in P_{s_i,t_i}, i = 1, ..., n. \end{aligned}$$

Obviously, the maximum multicommodity flow problem and the maximum concurrent multicommodity flow problem are linear programming problems, which can be solved

in polynomial time. Moreover, our approximation algorithms in Chapter 4 will start by solving the maximum concurrent multicommodity flow problem.

3.2 Single Source Unsplittable Flow Problem

In the single source unsplittable flow problem, we are given a graph $G = \langle V, E \rangle$, where V is a set of nodes and E is a set of edges with capacities u_e, $e \in E$. There are several commodities to be sent from a source node to n terminal nodes. Each terminal node $t_i \in V$ has a demand $d_i, i = 1, ..., n$. A commodity flow is called an unsplittable flow if the commodity is sent to the terminal by following one single path. In the single source unsplittable flow problem, we want to route all commodities unsplittably. The following figure is an example of the single source unsplittable flow problem.

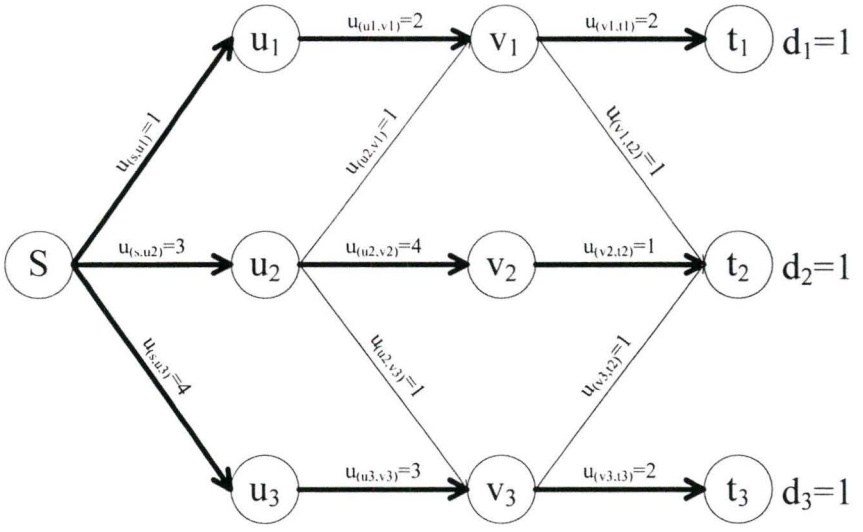


Figure 3.2: An example of the single source source unsplittable flow problem

In this example, the unsplittable flows for sending commodities follow the bold paths $(S \rightarrow u_1 \rightarrow v_1 \rightarrow t_1)$, $(S \rightarrow u_2 \rightarrow v_2 \rightarrow t_2)$ and $(S \rightarrow u_3 \rightarrow v_3 \rightarrow t_3)$, respectively.

The single source unsplittable flow problem was first studied by J. M. Kleinberg [Kle96]. In this paper, several questions were presented.

- Feasibility: can we find an unsplittable flow which satisfies all the demands and capacities for a given graph?
- Congestion: what is the smallest number α such that if we multiply all the capacities by α, there exists an unsplittable flow which satisfies all the demands?
- Number of rounds: if we are allowed to partition the set of commodities into several subsets (rounds), what is the minimum number of those subsets so that we can find a feasible unsplittable flow for each of them?
- Maximization: can we determine a subset of commodities so that we can route each commodity in this subset unsplittably and maximize $\sum d_n$?

Unfortunately, all of the above problems are NP-Complete. J. M. Kleinberg [Kle96] gives a 16-approximation for the congestion problem. He also shows that there is a $\lceil \ln(d_{max}/d_{min}) \rceil \cdot \lceil 2e\alpha^* \rceil$-approximation algorithm for the number of rounds problem and a $2e\lceil \ln(d_{max}/d_{min}) \rceil^{-1}$-approximation algorithm for the maximization problem, where d_{max} is the maximum demand in the graph, d_{min} is the minimum demand in the graph and α^* is the minimum congestion of any fractional routing from source node to terminal nodes (see Lemma 4.1 and Lemma 5.3 in [Kle96] for more details). We will consider the congestion problem in this thesis. The minimum congestion problem for the multicommodity flow is actually equivalent to the maximum concurrent multicommodity flow problem. Suppose we have the optimal congestion α^* for a multicommodity flow network. Instead of multiplying all edge capacities by α^*, we can divide the flow on each edge and every commodity demand by the factor α^*, and obtain a feasible multicommodity flow without violating edge capacities. Clearly, α^* is the smallest congestion, thus $\beta^* := 1/\alpha^*$ will be the largest fraction by which we can multiply all demands and still obtain a feasible multicommodity flow. By the NP-Completeness of the minimum congestion problem for unsplittable flows, it is easy to see that the maximum concurrent multicommodity unsplittable flow problem is also NP-Complete. Y. Dinitz, N. Garg and M. Goemans [DGG99] show a 2-approximation algorithm for the congestion problem, a 5-approximation al-

gorithm for the number of rounds problem and a 0.226-approximation algorithm for the maximization problem, when the maximum commodity demand is less than or equal to the minimum edge capacity in the network. We also refer to this condition as the balance condition. The algorithm for minimizing the congestion in [DGG99] will be used in this thesis.

3.3 Single Source Unsplittable Min-cost Flow Problem

The single source unsplittable min-cost flow problem is the cost version of the single source unsplittable flow problem. Suppose we have a graph $G = \langle V, E \rangle$. Let $f(e)$ be the flow amount on edge e, $e \in E$. We assume that there exists a cost function $c : E \rightarrow R^+$, where R^+ is the set of nonnegative reals and the total cost on edge e is $c(e)f(e)$, $e \in E$. All commodities are required to be sent from a single source node to n terminal nodes with demand $d_i, i = 1, ..., n$. There also exists a budget B, where we do not allow the total incurred cost of the network to be more than the budget B. This additional constraint can be written as follows:

$$\sum_{e \in E} f(e)c(e) \leq B.$$

The objective of the single source unsplittable min-cost flow problem is asking for paths to send all commodities to terminal nodes unsplittably without violating the given budget B.

For the single source unsplittable min-cost flow problem, M. Skutella [Sku02] studies the problems of congestion, number of rounds and maximization. For the congestion problem, if the balance condition is satisfied, a 3-approximation algorithm without violating the given budget was proposed. In the case of arbitrary demands, he also gave a $(3 + 2\sqrt{2})$-approximation algorithm while respecting the budget constraint. In this thesis, we only consider the problem of minimum congestion for the single source unsplittable min-cost flow problem, which is related to the concurrent multicommodity flow problem.

3.4 k-splittable Flow Problem (with Costs)

The k-splittable flow problem was first studied by G. Baier, E. Köhler and M. Skutella in [BKS02]. We consider the graph $G = \langle V, E \rangle$, with capacity u_e on each edge $e \in E$ and n terminal nodes. The k-splittable flow problem is a multicommodity flow problem where the number of paths for sending each commodity from the source node to the terminal node is bounded by the number k. More precisely, for arbitrary source nodes $s \in V$ and terminal nodes $t \in V$, let $P_{s,t}$ denote the set of paths from source nodes to terminal nodes. The k-splittable flow is specified by k paths $\{P_1, ..., P_k\} \subseteq P_{s,t}$ with flow amount f_i, $i = 1, ..., k$ on each path. We do not require all paths to be distinct. For a given number $k' \geq k$, any k-splittable flow is also a k'-splittable flow. The k-splittable flow is called feasible if the flow through each edge respects the edge capacity. Note that when $k = 1$, this is the unsplittable flow (instead of 1-splittable flow) problem. Thus, we see that the k-splittable flow problem is the general version of the unsplittable flow problem.

We generate the maximum concurrent multicommodity k-splittable flow problem by combining the k-splittable flow problem and the maximum concurrent multicommodity flow problem. The objective of the maximum concurrent multicommodity k-splittable flow problem is to send each commodity within k paths so that we maximize the factor by which we multiply each commodity demand and still get a feasible multicommodity flow instance respecting edge capacities.

We extend the maximum concurrent multicommodity k-splittable flow problem to the constrained maximum concurrent multicommodity k-splittable flow problem by introducing costs and a budget in the network. A cost function is given by $c : E \rightarrow R^+$. Let $f(e)$ be the flow on edge e. We define the cost on edge e to be $c(e)f(e)$, $e \in E$. We are also given a budget B, which is the maximum allowed cost of the network. Similar to the single source unsplittable min-cost flow problem, we have an additional cost constraint $\sum_{e \in E} f(e)c(e) \leq B$. The objective of this problem asks for a feasible maximum concurrent multicommodity k-splittable flow that does not violate the cost constraint.

Algorithms 1, 2, 3 ,4 in Chapters 4 and 5 will approximate the maximum concurrent multicommodity k-splittable flow problem by solving the maximum concurrent multicommodity uniform exactly-k-splittable flow problem. In the maximum concur-

rent multicommodity uniform exactly-k-splittable flow problem, we require exactly k paths for sending each commodity and each path in the network carries the same (uniform) flow amount. Thus the objective of this problem is to send each commodity in exactly k paths and each path carries the uniform flow amount while respecting edge capacities, so as to maximize the factor by which we multiply all demands. G. Baier, E. Köhler and M. Skutella [BKS02] proved that the value of a maximum concurrent multicommodity uniform exactly-k-splittable flow problem can guarantee $1/2$ of the optimal value of the corresponding maximum concurrent multicommodity k-splittable flow problem. Note that if we split each commodity by k sub-commodities, then sending the original commodity in exactly k paths is actually equivalent to routing each sub-commodity unsplittably.

3.5 Scheduling of Unrelated Machines Problem

We will show a transformation of TDCCP (with costs) to the scheduling of unrelated machines problem (with costs) in Chapters 4 and 5. Therefore, in this section, we introduce the definition and mathematical program of this problem.

The scheduling of unrelated machines problem is an integer optimization problem which can be stated as follows: Suppose we have m parallel machines and n independent jobs. Each job has to be processed by exactly one of m machines. We define $\tau_{i,j}$ to be the required processing time when job j is scheduled on machine i. If job j is scheduled on machine i, variable $y_{i,j} = 1$, otherwise $y_{i,j} = 0$. The total processing time of machine i is the sum of the processing times of all jobs which are scheduled on machine i, which can be written as $\sum_{j=1}^{n} \tau_{i,j} y_{i,j}$. The makespan of a schedule is the last finishing time of jobs in the given schedule. This problem asks for an optimal schedule to minimize the makespan. The following figure shows an example of this scheduling problem.

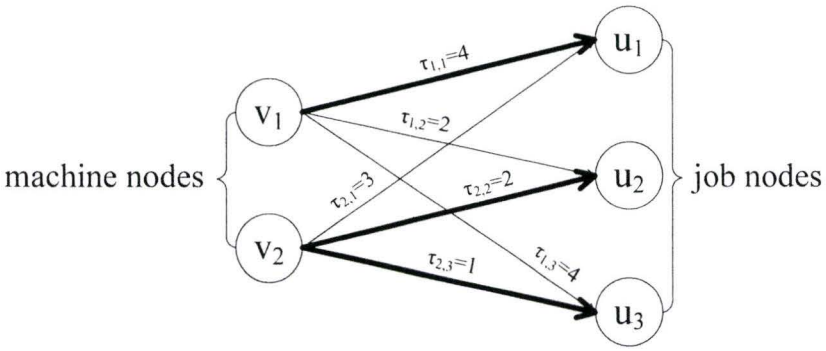


Figure 3.3: An example of the scheduling of unrelated machines problem

This is an example with two machines and three jobs. The bold lines in the figure are the optimal schedule with $y_{1,1} = y_{2,2} = y_{2,3} := 1$ and the remaining $y_{i,j} := 0$. The total processing time on machine 1 is $\tau_{1,1}y_{1,1} = 4$ and the total processing time on machine 2 is $\tau_{2,2}y_{2,2} + \tau_{2,3}y_{2,3} = 3$. Therefore the optimal makespan of this example is 4.

The mathematical program of the scheduling of unrelated machines problem is formulated as follows:

$$\min \quad T \tag{$M1$}$$

$$\text{s.t.} \quad \sum_{i=1}^{m} y_{i,j} = 1, j = 1, ..., n \tag{1}$$

$$\sum_{j=1}^{n} \tau_{i,j} y_{i,j} \leq T, i = 1, ..., m \tag{2}$$

$$y_{i,j} \in \{0, 1\}, i = 1, ..., m, j = 1, ..., n \tag{3}$$

where $\tau_{i,j}$ is the processing time of job j scheduled at machine i, T is the makespan we want to minimize and $y_{i,j}$ is the integer variable with value 0 or 1, where $y_{i,j} = 1$ means that job j is scheduled on machine i and $y_{i,j} = 0$ means that job j is not scheduled on machine i. Constraint (1) says that the job cannot be over assigned.

Constraint (2) says that the total processing time of all jobs on any machine cannot exceed the makespan. Constraint (3) says that we only permit integer assignments.

The scheduling of unrelated machines problem is NP-Complete (see [LST90]). But if we remove constraint (3) in (M1), the original scheduling problem is reduced to a linear programming problem, where the $y_{i,j}$ can be fractional numbers. It is well-known that one of the most natural strategies for solving integer optimization problems like (M1) is to drop the integrality constraints, get the solution of the linear relaxation problem and round its solution to an integer solution. J. K. Lenstra, D. B. Shmoys and E. Tardos [LST90] present a rounding scheme for computing an approximate solution of (M1). By a transformation from our problem (MP2) to the scheduling problem, their algorithm will be adopted in this thesis for solving the problem (MP2). For more details, refer to [LST90].

In the scheduling of unrelated machines problems with costs, the cost of job j on machine i is defined to be $c_{i,j}y_{i,j}$. A budget B is also given, which says the total cost on all machines cannot be more than B. We can write the additional constraint as follows:

$$\sum_{j=1}^{n}\sum_{i=1}^{m} y_{i,j}c_{i,j} \leq B.$$

The goal of this problem asks for an optimal schedule to minimize the makespan while the budget constraint is respected. Thus, the scheduling of unrelated machines problem with costs can be formulated as follows,

$$\begin{aligned}
&\min \quad T && (M2)\\
&\text{s.t.} \quad \sum_{i=1}^{m} y_{i,j} = 1, j = 1, ..., n && (1)\\
&\qquad \sum_{j=1}^{n} \tau_{i,j}y_{i,j} \leq T, i = 1, ..., m && (2)\\
&\qquad \sum_{j=1}^{n}\sum_{i=1}^{m} y_{i,j}c_{i,j} \leq B, i = 1, ..., m, j = 1, ..., n && (3)\\
&\qquad y_{i,j} \in \{0, 1\}, i = 1, ..., m, j = 1, ..., n && (4)
\end{aligned}$$

Note that (M1) is just a special case of (M2) if we set $c_{i,j} := 0$, $\forall i, j$ and $B := 0$.

Hence it is easy to see that the scheduling of unrelated machines problems with costs is also NP-Complete. D. B. Shmoys and E. Tardos [ST93] show an approximation algorithm for solving the scheduling of unrelated machines problem with costs which has the same approximation factor as the approximation algorithm of solving the non-cost scheduling of unrelated machines problem. For more details, refer to [ST93]. In Chapter 5 we will show a transformation of the problem (MPC2) to the scheduling of unrelated machines problem with costs, then the algorithm in [ST93] will be applied to solve (MPC2).

3.6 Minimum Cost Bipartite Matching Problem

In Algorithm 4 (refer to Chapter 5), the integer solution for TDCCP with costs is obtained by solving a minimum cost bipartite matching problem. In this problem, we are given a bipartite graph $G = \langle V_1, V_2, E \rangle$, where V_1, V_2 are two sets of disjoint nodes and all edges in E go between V_1 and V_2. A matching is a subset of edges $M \subseteq E$ such that for all nodes $v \in V_1 \cup V_2$, at most one edge in M is incident on v. We say that a node v is matched if an edge in M is incident on v; otherwise, we say that v is unmatched. Let $|V_1| = m$ and $|V_2| = n$. Without loss of generality, we assume $m \geq n$. We define $y_{i,j} = 1$ if node $j \in V_2$ is matched by edge (i, j), $i \in V_1$. We are also given a cost of the matching $c_{i,j}$, $\forall i \in V_1, j \in V_2$. The minimum cost bipartite matching asks for a matching which matches all the nodes in V_2 and minimizes the total cost of the matching. This problem can be formulated as follows,

$$\min \quad \sum_{i=1}^{m} \sum_{j=1}^{n} c_{i,j} y_{i,j} \qquad (MCM)$$

$$\text{s.t.} \quad \sum_{j=1}^{n} y_{i,j} \leq 1, i = 1, ..., m \qquad (1)$$

$$\sum_{i=1}^{m} y_{i,j} = 1, j = 1, ..., n \qquad (2)$$

$$y_{i,j} \in \{0, 1\}, i = 1, ..., m, j = 1, ..., n \qquad (4)$$

When $m = n$, this problem is called the assignment problem, which can be formulated as follows,

$$\begin{aligned}
\min \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{i,j}y_{i,j} && (AP)\\
\text{s.t.} \quad & \sum_{j=1}^{n} y_{i,j} = 1, i = 1, ..., n && (1)\\
& \sum_{i=1}^{n} y_{i,j} = 1, j = 1, ..., n && (2)\\
& y_{i,j} \in \{0, 1\}, i = 1, ..., n, j = 1, ..., n && (4)
\end{aligned}$$

The assignment problem (AP) can be solved by the Hungarian method with complexity at most $O(n^4)$. For details about the Hungarian method, see [Mur95] and [Lov86].

The assignment problem is just a special case of the minimum cost bipartite matching problem. We give an example of the assignment problem in the following figure.

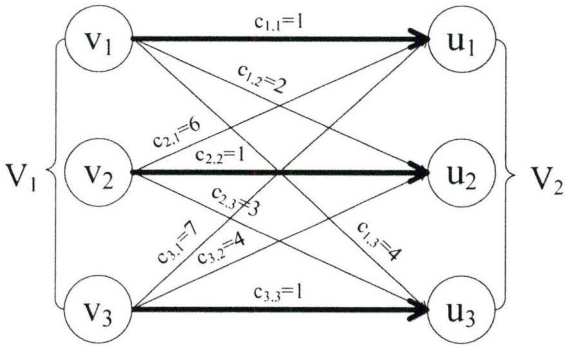


Figure 3.4: An example of the assignment problem

This is an example with $|V_1| = |V_2| = 3$. The bold lines in the figure denote the matching. It is easy to calculate the total cost of this bipartite graph is $c_{1,1}y_{1,1} + c_{2,2}y_{2,2} + c_{3,3}y_{3,3} = 3$, which is the minimum cost we can achieve.

When $m > n$, we can add $(m - n)$ new nodes v' in V_2 such that the cardinalities of V_1 and V_2 are same. Then for a new additional node v', we define the cost on all edges (v, v'), $v \in V_1$ to be zero. Note that by this construction, we do not change the essence of the original problem. By solving this assignment problem, we can still find the minimum cost matching in $G = \langle V_1, V_2, E \rangle$. Therefore, any minimum cost bipartite matching problem can be converted into an equivalent assignment problem. This observation will be applied to design Algorithm 4. Note that the matching found in the bipartite graph for the given assignment problem will be called a perfect matching since each node in the graph is exactly matched by one node. In Algorithm 4, we will transform the problem TDCCP with costs to an instance of the scheduling of unrelated machines problem. The perfect matching we found in a given bipartite graph will give us an integer schedule for the scheduling problem, by which we can obtain the original solution of TDCCP with costs.

Chapter 4

Total Discrete Capacity Constrained Problem without Costs

In this chapter, we discuss the Total Discrete Capacity Constrained Problem (TDCCP) without costs. First of all, we consider a special case of TDCCP where the service rates are independent of servers, and we show that the maximal capacity of TDCCP in this case can be computed in polynomial time. Secondly, we study another special case of TDCCP where the service rates are independent of classes, and design two approximation algorithms for solving this case. Then, we give an approximation algorithm for the general TDCCP. Finally, we prove the NP-Completeness of TDCCP by showing a polynomial time reduction to the Partition problem.

4.1 Overview of solving TDCCP

First we study two special cases of TDCCP. In the first case, the service rates are independent of all servers and $\mu_{j,k} = \mu_k$. We show that this case can be solved in polynomial time. In the second case, the service rates depend only on servers and $\mu_{j,k} = \mu_j$. We design two different approximation algorithms Algorithm 1 and Algorithm 2 for the second case. Given an $M \times 1$ vector α and a $K \times 1$ vector β, if the service rates $\mu = \alpha \cdot \beta^T$, then we show that this case will fall into the case

of $\mu_{j,k} = \mu_j$. Finally, we give the approximation algorithms for solving the general TDCCP by applying Algorithm 1 or Algorithm 2. Figure 4.1 shows the framework of our algorithms for solving TDCCP.

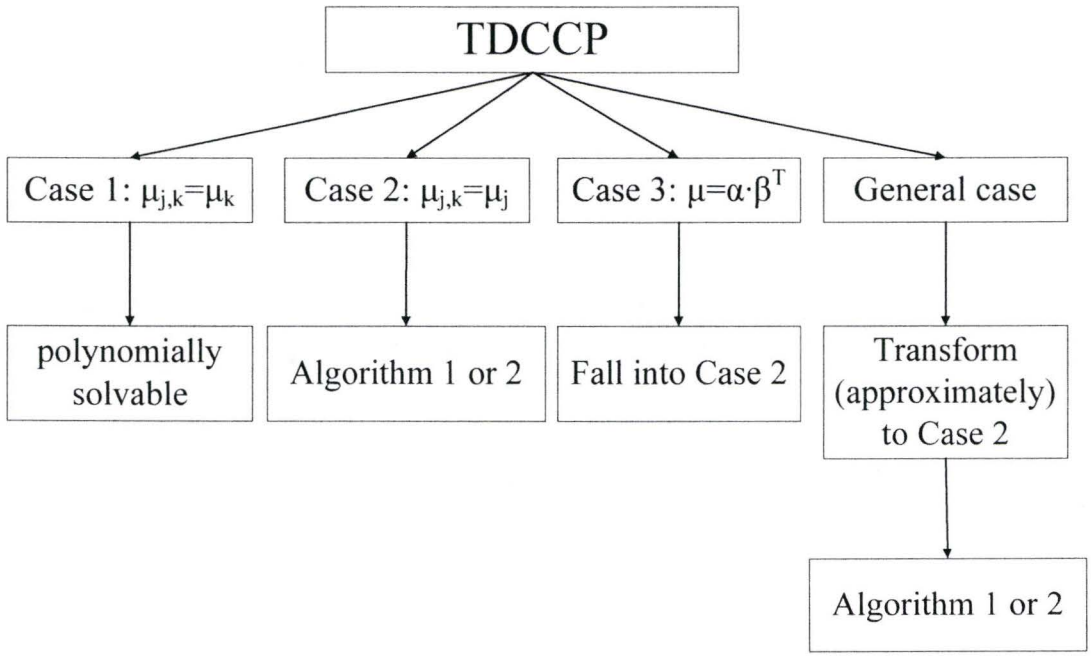


Figure 4.1: The Framework of our algorithms for solving TDCCP

4.2 Solving TDCCP in the case of $\mu_{j,k} = \mu_k$

In this case, for any class k, all servers have the same service rate. Thus we say the service rates are independent of all servers or all servers are identical.

In Chapter 2, we introduced the problem (MP1) for this case, which is as follows.

$$\begin{aligned}
\max \quad & \lambda && (MP1)\\
\text{s.t.} \quad & \sum_{j=1}^{M} \mu_k \delta_{j,k} \geq \lambda a_k, k = 1, ..., K && (1)\\
& \sum_{k=1}^{K} \delta_{j,k} \leq 1, j = 1, ..., M && (2)\\
& \sum_{j=1}^{M} \chi\{\delta_{j,k} > 0\} \leq c_k, k = 1, ..., K && (3)\\
& \delta_{j,k} \geq 0, k = 1, ..., K, j = 1, ..., M. && (4)
\end{aligned}$$

Let the optimal solution for (MP1) be $\{\delta^*_{j,k}\}$ and λ^*. We can prove the following theorem.

Theorem 4.1. *If $\mu_{j,k} = \mu_k$ for all j, the maximal capacity is*

$$\lambda^* = \min\left(\frac{M}{\sum_{k=1}^{K} a_k/\mu_k}, \min_{1\leq k\leq K} \frac{c_k \mu_k}{a_k}\right).$$

Proof. Suppose the optimal solution of (MP1) is λ^* and $\delta^*_{j,k}$. In constraint (3), the indicator function says that $\forall k$ if $\delta^*_{j,k} > 0$, then $\chi\{\delta^*_{j,k} > 0\} = 1$ otherwise $\chi\{\delta^*_{j,k} > 0\} = 0$. Because $\delta^*_{j,k} \leq 1$, we have for all k,

$$\delta^*_{j,k} \leq \chi\{\delta^*_{j,k} > 0\}.$$

Furthermore we can get for all k,

$$\sum_{j=1}^{M} \delta_{j,k} \leq \sum_{j=1}^{M} \chi\{\delta_{j,k} > 0\} \leq c_k. \tag{4.1}$$

Then we derive that for all k,

$$\sum_{j=1}^{M} \delta^*_{j,k} \leq c_k.$$

When we achieve the optimal, one of the following conditions must be satisfied:

(1) : $\sum_{j=1}^{M} \delta_{j,k}^* < c_k$, for $k = 1, ..., K$.

(2) : There exists at least one class l such that $\sum_{j=1}^{M} \delta_{j,l}^* = c_l$.

If condition (1) is satisfied, constraints (1) and (2) in (MP1) must be tight for all k and j, otherwise, λ^* and $\delta_{j,k}^*$ can be trivially increased. According to constraint (1) in (MP1), we have

$$\lambda^*(a_k/\mu_k) = \sum_{j=1}^{M} \delta_{j,k}^*, \forall k.$$

Then the sum of the above inequalities for all classes is:

$$\sum_{k=1}^{K} \lambda^*(a_k/\mu_k) = \sum_{k=1}^{K} \sum_{j=1}^{M} \delta_{j,k}^*.$$

Since $\sum_{k=1}^{K} \delta_{j,k}^* = 1$ we get:

$$\lambda^* = \frac{\sum_{k=1}^{K} \sum_{j=1}^{M} \delta_{j,k}^*}{\sum_{k=1}^{K} (a_k/\mu_k)} = \frac{M}{\sum_{k=1}^{K} (a_k/\mu_k)}. \tag{4.2}$$

If condition (2) is satisfied, by (4.1) we have the following:

$$\lambda^* \leq \frac{\sum_{j=1}^{M} \delta_{j,k}^*}{a_k/\mu_k} \leq \frac{c_k}{a_k/\mu_k}, \quad \forall k.$$

Let $k_0 := \text{argmin}_{1 \leq k \leq K}(\frac{c_k}{a_k/\mu_k})$. We know that either $\sum_{j=1}^{M} \delta_{j,k_0}^* = c_{k_0}$ or $\sum_{j=1}^{M} \delta_{j,k_0}^* < c_{k_0}$. If $\sum_{j=1}^{M} \delta_{j,k_0}^* < c_{k_0}$, we can increase δ_{j,k_0}^* and decrease $\delta_{j,k}^*$ for at least one j and $k \neq k_0$ until we have $\sum_{j=1}^{M} \delta_{j,k_0}^* = c_{k_0}$. Note that this transformation will not decrease λ^*. Then we know that $\frac{\sum_{j=1}^{M} \delta_{j,k_0}^*}{a_{k_0}/\mu_{k_0}}$ is the minimum value for all k. Since,

$$\lambda^* = \text{min}_{1 \leq k \leq K} \frac{\sum_{j=1}^{M} \delta_{j,k}^*}{a_k/\mu_k},$$

we have,

$$\lambda^* = \frac{\sum_{j=1}^{M} \delta_{j,k_0}^*}{a_{k_0}/\mu_{k_0}} = \frac{c_{k_0}}{a_{k_0}/\mu_{k_0}} = \text{min}_{1 \leq k \leq K} \frac{c_k \mu_k}{a_k}. \tag{4.3}$$

By (4.2) and (4.3), the maximal capacity λ^* of (MP1) is

$$\lambda^* = \min\left(\frac{M}{\sum_{k=1}^{K} a_k/\mu_k}, \min_{1\le k\le K}(\frac{c_k\mu_k}{a_k})\right).$$

□

Corollary 4.2. *There is a polynomial time algorithm for solving (MP1) in the case of $\mu_{j,k} = \mu_k$ for all j.*

Proof. We can calculate the optimal solution λ^* according the above theorem. Obviously, this calculation can be done in polynomial time. □

4.3 Solving TDCCP in the case of $\mu_{j,k} = \mu_j$

When $\mu_{j,k} = \mu_j$ for any server j, we say that the service rates depend only on servers or all classes are identical. We design two different approximation algorithms for solving this case.

4.3.1 Approximation Algorithm 1

Design of Algorithm 1

In Chapter 2, we introduced the problem (MP2) for this case, which is as follows.

$$\begin{aligned}
\max \quad & \lambda && (MP2)\\
\text{s.t.} \quad & \sum_{j=1}^{M} \mu_j\delta_{j,k} \ge \lambda a_k, k = 1, ..., K && (1)\\
& \sum_{k=1}^{K} \delta_{j,k} \le 1, j = 1, ..., M && (2)\\
& \sum_{j=1}^{M} \chi\{\delta_{j,k} > 0\} \le c_k, k = 1, ..., K && (3)\\
& \delta_{j,k} \ge 0, k = 1, ..., K, j = 1, ..., M. && (4)
\end{aligned}$$

Now we define a new variable $x_{j,k} = \mu_j \delta_{j,k}$, and rewrite (MP2) as (MP2').

$$\begin{aligned}
\max \quad & \lambda && (MP2') \\
\text{s.t.} \quad & \sum_{j=1}^{M} x_{j,k} \geq \lambda a_k, k = 1, ..., K && (1) \\
& \sum_{k=1}^{K} x_{j,k} \leq \mu_j, j = 1, ..., M && (2) \\
& \sum_{j=1}^{M} \chi\{x_{j,k} > 0\} \leq c_k, k = 1, ..., K && (3) \\
& x_{j,k} \geq 0, k = 1, ..., K, j = 1, ..., M. && (4)
\end{aligned}$$

Problem (MP2') can be viewed as an instance of the maximum concurrent multicommodity k-splittable flow problem with the following network topology. This network has a single source node $\{S\}$ for sending K commodities, $2M$ nodes $\{v_1, ..., v_M\}$, $\{u_1, ...u_M\}$ for transferring commodities, and K terminal nodes $\{t_1, ..., t_K\}$ with demand $a_k, k = 1, ..., K$ for receiving commodities. First we connect source node S with every node in the set $\{v_1, ..., v_M\}$, and set infinite capacity on the edge $(S, v_j), j = 1, ..., M$. Then we connect nodes v_j with u_j, and set the capacity on the edge (v_j, u_j) to be μ_j for $j = 1, ..., M$. Finally we connect node $u_j, j = 1, ..., M$ with every node in the set $\{t_1, ..., t_K\}$, and also set infinite capacity on the edge $(u_j, t_k), j = 1, ..., M, k = 1, ..., K$. The number of different paths allowed for each commodity k is bounded by c_k $(1 \leq c_k \leq M)$, $k = 1, ..., K$. Then the problem (MP2') asks for the flow assignment $\{x_{j,k}\}$ of sending K commodities to t_k while respecting capacity μ_j on edge (v_j, u_j), $j = 1, ..., M$ and splittablity bound c_k for each commodity k, $k = 1, ..., K$, so that the maximum possible fraction λ of all commodity demands a_k, $k = 1, ..., K$ is simultaneously achieved. Please refer to Figure 4.2 for the network topology of (MP2').

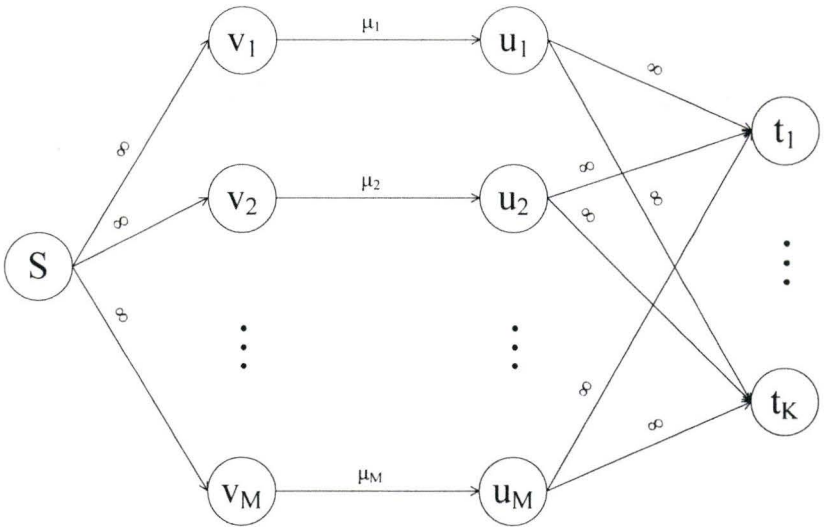


Figure 4.2: The network structure for our special case (MP2')

To obtain an approximate solution of (MP2'), Algorithm 1 does the following:

1. In the first stage, we approximate (MP2') by solving the maximum concurrent multicommodity uniform exactly-k-splittable flow problem.

2. In the second stage, we approximate the maximum concurrent multicommodity uniform exactly-k-splittable flow problem by computing the minimum congestion of an unsplittable flow problem.

3. Finally, we compute the approximate solution of the problem (MP2) by the unsplittable flow and congestion we found.

Description of Algorithm 1

In Algorithm 1, the stage for computing the minimum congestion of an unsplittable flow problem is based on the algorithms in [DGG99] and [KS02]. We first introduce some necessary definitions which will be used in our algorithm.

• Regular terminal: a terminal node t_k is regular if its demand a_k is greater than the flow on every edge entering t_k, otherwise it is an irregular terminal.

• Singular edge: an edge (u, v) is called singular if all the vertices which are reachable from v have out-degree at most 1.

The detailed description of Algorithm 1 is as follows.

Stage 1: Initialization. Construct a network (see Figure 4.1) with one source node $\{S\}$ for sending K commodities, $2M$ nodes $\{v_1, ..., v_M, u_1, ...u_M\}$ for transferring commodities, and K terminal nodes $\{t_1, ..., t_K\}$ with demand $a_k, k = 1, ..., K$ for receiving commodities. Set the capacity on the edge (v_j, u_j) to be μ_j for $j = 1, ..., M$.

Stage 2: Transformation of initial problem to a maximum concurrent uniform exactly-k-splittable flow problem. We replicate each terminal node t_k into c_k identical sub-terminals $t_{(k,i)}$, $i = 1, ..., c_k$. Then each commodity k with demand a_k is split into c_k sub-commodities (k, i), $i = 1, ..., c_k$, each with the same demand $a_{(k,i)} := a_k/c_k$ and with the same source. Further, for each edge connecting with terminal node t_k, we connect it with all of the sub-terminals $t_{(k,i)}$. Now sending the new commodity (k, i) unsplittably is equivalent to sending the original commodity k exactly in c_k paths with the same flow amount on each path.

Stage 3: Calculation of demand $a_{(k,i)}$. We use the ellipsoid method (see [Sch86]) to solve the following relaxation of the maximum concurrent uniform exactly-k-splittable flow problem.

$$\begin{aligned}
\max \quad & \lambda && (MP2'') \\
\text{s.t.} \quad & \sum_{j=1}^{M} x_{j,(k,i)} \geq \lambda a_k/c_k, \forall (k,i) && (1) \\
& \sum_{(k,i)} x_{j,(k,i)} \leq \mu_j, \forall j && (2) \\
& x_{j,(k,i)} \geq 0, \forall j, (k,i) && (3)
\end{aligned}$$

Let $\hat{\lambda}^*$ and $\hat{x}^*_{j,(k,i)}$ be the optimal fractional solution of (MP2"). Then we assign the flow amount $\hat{x}^*_{j,(k,i)}$ on the edge $(u_j, t_{(k,i)})$ and the flow amount $\sum_{(k,i)} \hat{x}^*_{j,(k,i)}$ on the edges (s, v_j) and (v_j, u_j). For each sub-terminal in the network, let its demand be $a^*_{(k,i)} := \hat{\lambda}^* a_k/c_k$. Note that the flow $\hat{x}^*_{j,(k,i)}$ computed in this stage is the fractional flow.

Stage 4: Transformation of the fractional flow calculated in Stage 3 into an unsplittable flow.

Step 1: Let $a_{min} := \min_{(k,i)}\{a^*_{(k,i)}\}$ and $a_{max} := \max_{(k,i)}\{a^*_{(k,i)}\}$. If $a_{max} > 1$, we scale all sub-terminal demands and edge capacities by the factor a_{max} so that $a_{(k,i)} \in (0, 1]$ and $\mu_j \in (0, +\infty)$. Let $D := 1/a_{min}$, we know that all demands fall within the interval $[1/D, 1]$. We partition the interval $[1/D, 1]$ into sub-intervals as follows:

$$[1/D, 1/2^{\lfloor logD \rfloor}], ..., (1/2^{i+1}, 1/2^i], ..., (1/2, 1].$$

Obviously there are $\lfloor logD \rfloor + 1$ sub-intervals. We number them as sub-interval 1, 2,...,$\lfloor logD \rfloor$ from left to right.

Step 2: For each sub-interval above (in the increasing order of interval i), we find the commodities (k, i) whose demand $a^*_{(k,i)}$ falls within the ith sub-interval, $i = 1, ..., \lfloor logD \rfloor$. We assume there are m such commodities. For convenience, we name them as commodities $1, 2, ..., m$ and their corresponding sub-terminals are $t_1, t_2, ..., t_m$. Let x_{j,t_l} denote the flow on the edge (u_j, t_l), $j = 1, ..., M, l = 1, ..., m$. We define a new variable $y_{j,(k,i)}$, which is used to record the unsplittable flow we find on edge $(j, (k, i))$ for each commodity (k, i). Once we finish computing $y_{j,(k,i)}$ for all j and (k, i), by the network topology of (MP2"), we can easily determine the unsplittable flows for sending all (k, i). $y_{j,(k,i)}$ can be assigned as follows:

(a) Check each edge in the network, if no commodity l, $l = 1, ..., m$ goes through this edge, we remove this edge from the network.

(b) Eliminate irregular sub-terminals from the network (refer to Figure 4.1 for the network topology). We check each sub-terminal t_l in the network, $l = 1, ..., m$. If there is any sub-terminal t_l with only one incoming edge (u_j, t_l), we know that the flow amount x_{j,t_l} on edge (u_j, t_l) equals the demand $a^*_{t_l}$ of node t_l. We assign the commodity l to edge (v_j, u_j) and $y_{j,t_l} := a^*_{t_l}$. Now we know that the commodity l will be routed unsplittably by following the path

$$S \rightarrow v_j \rightarrow u_j \rightarrow t_l.$$

Then we remove commodity l from the network, decrease the flow on the edge (S, v_j), (v_j, u_j), (u_j, t_l) by x_{j,t_l}. Obviously, the flow on edge (u_j, t_l) will be zero. After this step, there is no irregular sub-terminal in the network.

Thus, each of the remaining sub-terminals must have at least two incoming edges.

(c) Find an alternating cycle for each of the remaining commodities. The alternating cycle is constructed by alternatively going forward or backward on the edges of the network until we form a cycle, which can be stated as follows. Starting from a node $u_{j_1} \in \{u_1, ..., u_M\}$ and stopping at a terminal node $t_{l_1} \in \{t_1, ..., t_m\}$, the forward path is constructed by selecting an edge (u_{j_1}, t_{l_1}) with the positive flow $x_{j_1,t_{l_1}}$ on it. Since there are at least two incoming edges for each sub-terminal after step (b), we can find at least one distinct path to follow back to a different node u_{j_2}. We call this path the backward path. Whenever we finish constructing a backward path and stop at a node u_{j_p}, we check the edge (u_{j_p}, t_{l_1}). If edge (u_{j_p}, t_{l_1}) is a singular edge, we continue constructing the backward path by adding edges (v_{j_p}, u_{j_p}), (s, v_{j_p}) in the backward path and then finish the alternating cycle by adding edges (s, v_{j_1}), (v_{j_1}, u_{j_1}) as the forward path in the cycle. If edge (u_{j_p}, t_{l_1}) is a non-singular edge, we can find one distinct edge (u_{j_p}, t_{l_2}) leaving the node u_{j_p} and this edge will be added into the forward path. Then we construct the backward path from the node t_{l_2} again. We perform the above steps for constructing the forward path and backward path alternatively until a cycle

$$u_{j_1} \rightarrow t_{l_1} \rightarrow u_{j_2} \rightarrow t_{l_2} \rightarrow ... \rightarrow u_{j_1}$$

is formed. Please also refer to Figure 2 in [DGG99] for an example of constructing the alternating cycle.

(d) Augment the alternating cycle found in (c). We first check the edge capacities in the backward path of the alternating cycle. If there is any edge with capacity less than the commodity demand in the backward path, we ignore this cycle and continue to find another alternating cycle. Otherwise, we decrease flow along the forward path and increase flow along the backward path by the same amount ϵ, where $\epsilon = \min\{\epsilon_1, \epsilon_2\}$. Suppose there are q sub-terminals in the alternating cycle, we define ϵ_1 to be the minimum flow amount along any edge in the forward paths, and ϵ_2 to be the minimum value of $a_{t_l} - x_{j,t_l}$ along the edge (u_j, t_l), $l = 1, ..., q$ in the backward path.

After we decrease and increase flow respectively in the forward and backward paths, we can get at least one edge with flow 0 or one edge (u_j, t_l) with flow equal to a sub-terminal's demand at node t_l. Then go to step (a). Clearly, by step (d), we will never send any unsplittable flow through an edge with capacity less than the corresponding commodity demand.

(e) We continuously repeat steps (a) to (d) until commodities l, $l = 1, ..., m$ are all removed from the network.

Stage 5: Calculation of the original solution $\delta_{j,k}$ and the capacity λ for (MP2). $y_{j,(k,i)}$ is the final unsplittable flow for sending sub-commodity (k, i). Note that if the original $a_{max} > 1$ in Step 1 of Stage 4, we need to restore the flow and capacities in the network by multiplying $y_{j,(k,i)}$ and μ_j with a_{max}, for all j, (k, i). The total flow through edge (v_j, u_j) is $\sum_{(k,i)} y_{j,(k,i)}$. We define $\alpha := \max_j \frac{\sum_{(k,i)} y_{j,(k,i)}}{\mu_j}$. If $\alpha > 1$, we know that edge capacities in our network are violated. Then we scale down the calculated unsplittable flow $y_{j,(k,i)}$ by the factor α so that the edge capacities in the network are respected. We know that the original commodity k is split into c_k sub-commodities, and the flow of commodity k through edge (v_j, u_j) is $\sum_{i=1}^{c_k} y_{j,(k,i)}$. Then we set the original assignments $\delta_{j,k}$ of servers to classes to be $\delta_{j,k} := \sum_{i=1}^{c_k} y_{j,(k,i)}/\mu_j$ for all j, k. Finally, according to constraint (1) of (MP2), the corresponding capacity λ can be computed by the server assignment $\delta_{j,k}$, i.e. $\lambda = \min_k \frac{\sum_{j=1}^{M} \mu_j \delta_{j,k}}{a_k}$.

Analysis of Algorithm 1

To evaluate the performance guarantee of Algorithm 1, first we show the proof of Theorem 5 in [BKS02].

Theorem 4.3. *Computing the maximum concurrent multicommodity uniform exactly-k-splittable flow problem in Stage 2 of Algorithm 1 guarantees at least* $1/2$ *of the optimal value for the maximum concurrent multicommodity k-splittable flow problem (MP2').*

Proof. Suppose the optimal solution of a maximum concurrent multicommodity k-splittable flow problem (MP2') is λ^* and $x^*_{j,k}$. We show that there exists a feasible maximum concurrent multicommodity uniform exactly-k-splittable flow with flow value $D := \lambda^* a_k/(2c_k)$ on each path for sending sub-commodity (k, i), $\forall(k, i)$.

In the given optimal solution of the maximum concurrent multicommodity k-splittable flow problem, we replace the edge (j,k) for sending commodity k with flow $x^*_{j,k}$ by $\lfloor x^*_{j,k}/D \rfloor$ copies, each carrying flow amount D. Let P be the set of edges with $x^*_{j,k} > 0$. By this construction, we can show that there are at least c_k copies of edges that we can produce for sending commodity k and each copy of an edge carries flow amount D, using the fact that the number of paths for sending commodity k is bounded by c_k

$$\begin{aligned}\sum_{P:x^*_{j,k}>0} \lfloor x^*_{j,k}/D \rfloor &\geq \sum_{P:x^*_{j,k}>0} (x^*_{j,k}/D - 1) \\ &\geq (\sum_{P:x^*_{j,k}>0} x^*_{j,k})/D - c_k \\ &\geq \frac{\lambda^* a_k}{D} - c_k = c_k.\end{aligned}$$

By the above inequality, we know that there are at least c_k paths carrying the flow amount D for sending commodity k, $k = 1, ..., K$. Thus the maximum flow achieved for commodity k is at least $D \cdot c_k = \lambda^* a_k/2$ and the maximum possible fraction for routing commodity k is at least $\lambda^*/2$ for (MP2'). □

By treating each sub-commodity separately and routing each sub-commodity unsplittably, we solve the maximum concurrent multicommodity unsplittable flow problem, instead of solving the maximum concurrent multicommodity uniform exactly-k-splittable flow problem. According to Theorem 5.1 in [KS02], we can prove the following theorem for bounding the minimum congestion in Stage 4 of Algorithm 1.

Theorem 4.4. *Stage 4 of Algorithm 1 routes each sub-commodity unsplittably such that the the total flow through (v_j, u_j) exceeds its edge capacity μ_j by less than $5\mu_j$, $j = 1, ..., M$.*

Proof. In our network, the capacitated edges are (v_j, u_j), $j = 1, ..., M$ and the other edges have infinite capacities. Hence in order to compute the minimum congestion, we only consider (v_j, u_j), $j = 1, ..., M$.

Let $D := 1/a_{min}$. We partition the interval $[1/D, 1]$ into sub-intervals

$$[1/D, 1/2^{\lfloor logD \rfloor}], ..., (1/2^{i+1}, 1/2^i], ..., (1/2, 1].$$

For an arbitrary sub-interval $(1/2^{i+1}, 1/2^i]$, by Step 1 in Stage 4, we know that we only consider commodity l with demand a_l which is located in this sub-interval and other commodities will be temporarily removed from the network. Let the initial flow on edge (v_j, u_j) be $f_{(v_j,u_j)}$ for sending the commodities (fractionally) with demands in this interval. Since the fractional flow $f_{(v_j,u_j)}$ computed in Stage 3 respects the edge capacity of (v_j, u_j), we have

$$f_{(v_j,u_j)} \leq \mu_j.$$

In Step 2 of Stage 4, if there is any edge with capacity less than the commodity demand in the backward path of the alternating cycle, we will ignore this cycle. This step is crucial because it guarantees that any commodity in this interval is only sent through edges with capacities at least its demand. Since the minimum commodity demand in this interval is $1/2^{i+1}$, we have

$$\mu_j \geq \frac{1}{2^{i+1}}.$$

Also, when we augment the alternating cycle, if the flow on the edge (v_j, u_j) is equal to a commodity demand and exceeds its capacity, this edge will be removed from the network in this interval. Thus we know that the flow through edge (v_j, u_j) exceeds its capacity by less than the maximal demand in this interval. Obviously, the maximal commodity demand in the interval $(1/2^{i+1}, 1/2^i]$ is at most $1/2^i$. Since the edge (v_j, u_j) may also be used in other sub-intervals before $(1/2^{i+1}, 1/2^i]$, the flow F on it is at most:

$$\begin{aligned} F &\leq f_{(v_j,u_j)} + \sum_{j=i}^{\lfloor logD \rfloor} 1/2^j \\ &= f_{(v_j,u_j)} + \frac{2}{2^i} - \frac{1}{2^{\lfloor logD \rfloor}} \\ &= f_{(v_j,u_j)} + \frac{4}{2^{i+1}} - \frac{1}{2^{\lfloor logD \rfloor}} \end{aligned}$$

We know that $f_{(v_j,u_j)} \leq \mu_j$ and $\frac{1}{2^{i+1}} \leq \mu_j$. Then, the total flow F on the edge (v_j, u_j) is bounded by,

$$\begin{aligned} F &\leq f_{(v_j,u_j)} + \frac{4}{2^{i+1}} - \frac{1}{2^{\lfloor logD \rfloor}} \\ &\leq \mu_j + 4\mu_j - \frac{1}{2^{\lfloor logD \rfloor}} \\ &= 5\mu_j - \frac{1}{2^{\lfloor logD \rfloor}} \leq 5\mu_j. \end{aligned}$$

Now we can get the conclusion that for any edge (v_j, u_j) in the graph, the flow through it is at most 5 times its edge capacity. Therefore, we know that there exists a 5-approximation algorithm for the minimum congestion of the unsplittable flow problem. So, we can achieve a 1/5-approximation algorithm for the maximum concurrent multicommodity unsplittable flow problem. □

Summarizing Theorems 4.3 and 4.4 we get the following theorem:

Theorem 4.5. *Algorithm 1 is a* 1/10*-approximation algorithm for solving (MP2').*

Proof. Let the optimal solution of (MP2') be λ^*. According to Theorem 4.3, solving the maximum concurrent multicommodity uniform exactly-k-splittable flow problem will achieve at least 1/2 of the optimal solution λ^*. By Theorem 4.4, Stage 4 in Algorithm 1 finds the minimum congestion for the unsplittable flow problem and achieves at least 1/5 of the optimal solution for the maximum concurrent multicommodity uniform exactly-k-splittable flow problem. Thus we can at least achieve 1/10 of the optimal capacity λ^* in (MP2'). □

4.3.2 Approximation Algorithm 2

Design of Algorithm 2

Approximation algorithm 2 is based on the algorithm in [LST90] for the scheduling of unrelated parallel machines problem. The first stage of Algorithm 2 is the same as Algorithm 1, where we transform (MP2') to the maximum concurrent multicommodity exactly-k-splittable flow problem and lose 1/2 of approximation factor.

By analyzing the network structure of (MP2'), we find that the flow on the edges (S, v_i) and (v_i, u_i), $i = 1, ..., M$ depends only on the flow amount of the edges (u_i, t_k), $k = 1, ..., K$. It is easy to see that $\{u_1, ..., u_M\}$ and $\{t_1, ..., t_K\}$ can be viewed as two

disjoint node sets in a bipartite graph. According to this observation, the second stage of Algorithm 2 computes the fractional solution of the maximum concurrent multicommodity exactly-k-splittable flow problem and converts its solution into the solution of an instance of the scheduling of unrelated machines problem.

In the third stage of Algorithm 2, we show how to round the fractional solution of the scheduling of unrelated machines problem to an integer solution by using the approximation algorithm studied in [LST90].

Finally, we map the integer solution of the scheduling problem back to the original solution $\delta_{j,k}$ and obtain the capacity λ of original problem (MP2).

Transformation of (MP2') to the scheduling of unrelated machines problem

When (MP2') is transformed into the maximum concurrent multicommodity exactly-k-splittable flow problem, we have the following relaxation of the multicommodity concurrent unsplittable flow problem:

$$\max \quad \lambda \qquad (MP2'')$$

$$\text{s.t.} \quad \sum_{j=1}^{M} x_{j,(k,i)} \geq \lambda a_k/c_k, \forall(k,i) \qquad (1)$$

$$\sum_{(k,i)} x_{j,(k,i)} \leq \mu_j, \forall j \qquad (2)$$

$$x_{j,(k,i)} \geq 0, \forall j, (k,i) \qquad (3)$$

where $x_{j,(k,i)} = \mu_j \delta_{j,(k,i)}$. Let $x^*_{j,(k,i)}$ and λ^* be the optimal solution (fractional) for (MP2").

First, we define $\lambda_{(k,i)} := \frac{\sum_{j=1}^{M} x^*_{j,(k,i)}}{a_k/c_k}$, $\forall k, i$. By constraint (1) in (MP2"), since $\sum_{j=1}^{M} x^*_{j,(k,i)} \geq \lambda^* a_k/c_k$, $\forall k, i$, it is easy to see that

$$\lambda_{(k,i)} = \frac{\sum_{j=1}^{M} x^*_{j,(k,i)}}{a_k/c_k} \geq \frac{\lambda^* a_k/c_k}{a_k/c_k} = \lambda^* > 0, \forall(k,i)$$

Then, we define $y_{j,(k,i)} := \frac{x^*_{j,(k,i)} c_k}{\lambda_{(k,i)} a_k}$ and $p_{j,(k,i)} := \frac{a_k}{c_k \mu_j}$, $\forall j, (k,i)$.

Using the above definitions, we can prove the following equality.

$$\begin{aligned}\sum_{j=1}^{M} y_{j,(k,i)} &= \sum_{j=1}^{M} \frac{x^*_{j,(k,i)} c_k}{\lambda_{(k,i)} a_k} \\ &= \frac{c_k \sum_{j=1}^{M} x^*_{j,(k,i)}}{\lambda_{(k,i)} a_k} \\ &= \frac{c_k \sum_{j=1}^{M} x^*_{j,(k,i)}}{\frac{\sum_{j=1}^{M} x^*_{j,(k,i)}}{a_k/c_k} a_k} \\ &= 1, \forall (k,i).\end{aligned}$$

Also, we have

$$\begin{aligned}\sum_{(k,i)} p_{j,(k,i)} y_{j,(k,i)} &= \sum_{(k,i)} \frac{a_k}{c_k \mu_j} \frac{x^*_{j,(k,i)} c_k}{\lambda_{(k,i)} a_k} \\ &= \sum_{(k,i)} \frac{x^*_{j,(k,i)}}{\mu_j \lambda_{(k,i)}} \\ &= \sum_{(k,i)} \frac{x^*_{j,(k,i)}}{\frac{c_k \mu_j}{a_k} \sum_{j=1}^{M} x^*_{j,(k,i)}} \\ &\leq \sum_{(k,i)} \frac{x^*_{j,(k,i)}}{\lambda^* \mu_j} \leq \frac{1}{\lambda^*}, \forall j.\end{aligned}$$

Summarizing the above, if we let $T = 1/\lambda^*$, we have the following linear programming problem:

$$\sum_{j=1}^{M} y_{j,(k,i)} = 1, \forall (k,i) \qquad (1) \quad (S1)$$

$$\sum_{(k,i)} p_{j,(k,i)} y_{j,(k,i)} \leq T, \forall j \qquad (2)$$

$$y_{j,(k,i)} \geq 0, \forall j, (k,i). \qquad (3)$$

This is exactly the relaxation version of the scheduling unrelated machines problem, where (k,i) can be viewed as jobs and j can be viewed as machines. The jobs in

(S1) correspond with sub-commodity nodes in (MP2") and the machines in (S1) correspond with edges (v_j, u_j), $j = 1, ..., M$ in (MP2"). The processing time for job (k, i) scheduled on machine j is $p_{j,(k,i)}$. Note that scheduling job (k, i) on machine j is actually equivalent to routing sub-commodity (k, i) on the edge (v_i, u_i) unsplittably. The fraction of job (k, i) scheduled on machine j is $y_{j,(k,i)}$. Given $x^*_{j,(k,i)}$ and λ^* in (MP2"), we can find a feasible (fractional) solution $y_{j,(k,i)}$ and the makespan $T = 1/\lambda^*$ for the linear programming problem (S1).

Let $\rho := \max_{j,(k,i)}\{p_{j,(k,i)}/T\}$. We consider the following linear programming problem:

$$\sum_{j=1}^{M} y_{j,(k,i)} = 1, \forall (k,i) \qquad (1) \quad (S2)$$

$$\sum_{(k,i)} p_{j,(k,i)} y_{j,(k,i)} \leq T, \forall j \qquad (2)$$

$$y_{j,(k,i)} = 0 \quad \text{if} \quad p_{j,(k,i)} > \rho \cdot T, \forall j, (k,i) \qquad (3)$$

$$y_{j,(k,i)} \geq 0, \forall j, (k,i). \qquad (4)$$

The constraint (3) in (S2) does not affect the value of $y_{j,(k,i)}$, since

$$p_{j,(k,i)} \leq \max_{j,(k,i)}\{p_{j,(k,i)}\} = \rho \cdot T$$

and we will not set any $y_{j,(k,i)}$ to zero. Thus the solution $y_{j,(k,i)}$ and $T = 1/\lambda^*$ in (S1) is still feasible in (S2).

Now we show the procedure *Int_Schedule* for rounding the fractional solution $y_{j,(k,i)}$ to the integer solution $\hat{y}_{j,(k,i)}$. This procedure is actually based on Theorem 1 (rounding theorem) in [LST90]. When we find an integer solution $\hat{y}_{j,(k,i)}$ for (S2) by *Int_Schedule*, the integer solution $\hat{x}_{j,(k,i)}$ can be computed by setting $\hat{x}_{j,(k,i)} = \hat{y}_{j,(k,i)} \lambda_{(k,i)} a_k / c_k$.

Procedure *Int_Schedule*
Input: the fractional schedule $y_{j,(k,i)}$ and makespan $T = 1/\lambda^*$ of (S2).
Output: the integer schedule $\hat{y}_{j,(k,i)}$.

Step 1: Consider the corresponding bipartite graph of (S2) and assign each edge

$(j,(k,i))$ with weight $y_{j,(k,i)}$. If $y_{j,(k,i)} = 0$, we remove the edge $(j,(k,i))$ from the graph and set $\hat{y}_{j,(k,i)} := 0$.

Step 2: Check all fractional solutions $y_{j,(k,i)}$. If $y_{j,(k,i)} = 1$, we adopt this integer solution, schedule job (k,i) on machine j, set $\hat{y}_{j,(k,i)} := 1$, $\hat{y}_{j_1,(k,i)} := 0, \forall j_1 \neq j$ and remove this edge $(j,(k,i))$ and the job node (k,i) from the graph.

Step 3: Scan the rest of the job nodes, if we cannot find a cycle starting from the node (k,i), we know there exists a tree rooted at the node $(k.i)$. Then we randomly choose a machine node j which is connected with the job node (k,i), schedule the job (k,i) at machine j, and set $\hat{y}_{j,(k,i)} := 1$, $\hat{y}_{j_1,(k,i)} := 0, \forall j_1 \neq j$. Finally we remove the job node (k,i) and all other edges connected with job (k,i).

Step 4: When we find a cycle starting from the node (k,i), we delete the alternate edges in this cycle. Then we check again whether we can find any cycle starting from node (k,i). If there is no cycle, (k,i) must be the root of a tree, otherwise, we continue constructing a cycle and delete the alternate edges in the cycle until we cannot find any cycle starting at (k,i). Now considering the tree rooted at (k,i), we randomly choose a machine node j connected with (k,i) and schedule (k,i) on j by setting $\hat{y}_{j,(k,i)} := 1$, $\hat{y}_{j_1,(k,i)} := 0, \forall j_1 \neq j$. Then we remove the job node (k,i) from the graph and any edge connected with (k,i).

Step 5: Repeat Steps 3 and 4 until all job nodes (k,i) are scheduled on machines and $\hat{y}_{j,(k,i)}$ is our final integer solution.

Description of Algorithm 2

The detailed description of Algorithm 2 is as follows. Note that Stages 1 and 2 in Algorithm 2 are the same as those in Algorithm 1.

Stage 1: Initialization. Construct a network with one source node $\{S\}$ for sending K commodities, $2M$ nodes $\{v_1,...,v_M,u_1,...u_M\}$ for transferring commodities, and K terminal nodes $\{t_1,...,t_K\}$ with demand $a_k, k = 1,...,K$ for receiving commodities. Set the capacity of the edge (v_j,u_j) to be μ_j for $j = 1,...,M$.

Stage 2: Transformation of initial problem to a maximum concurrent uniform exactly-k-splittable flow problem. We replicate each terminal node

t_k into c_k identical sub-terminals $t_{(k,i)}$, $i = 1, ..., c_k$. Then each commodity k with demand a_k is split into c_k commodities (k, i), $i = 1, ..., c_k$, each with the same demand $a_{(k,i)} := a_k/c_k$ and with the same source. Further, for each edge connecting with terminal node t_k, we connect it with all of the sub-terminals $t_{(k,i)}$. Note the new commodities (k, i) must be routed unsplittably.

Stage 3: Solving the fractional maximum concurrent uniform exactly-k-splittable flow problem and transformation of its solution to a solution of the scheduling of unrelated machines problem. We consider the relaxed version of the maximum concurrent uniform exactly-k-splittable flow problem (MP2") and get the solution $x_{j,(k,i)^*}$ and λ^* by the ellipsoid method. Then we construct an instance of scheduling of unrelated machines problem, with job nodes (k, i), $\forall k, i$ and machine nodes j, $j = 1, ..., M$. The processing time for a job (k, i) scheduled on machine j is defined to be $p_{j,(k,i)}$, where $p_{j,(k,i)} = \frac{a_k}{c_k \mu_j}$. Let a feasible solution of this scheduling problem be $y_{j,(k,i)}$ and the makespan T. Further we define $y_{j,(k,i)} = \frac{x^*_{j,(k,i)} c_k}{\lambda_{(k,i)} a_k}$ and $T = 1/\lambda^*$.

Stage 4: Calculation of the integer solution $\hat{y}_{j,(k,i)}$ by the fractional solution $y_{j,(k,i)}$. We use the procedure *Int_Schedule* to round the fractional solution $y_{j,(k,i)}$ to the integer solution $\hat{y}_{j,(k,i)}$.

Stage 5: Calculation of the integer solution $\hat{x}_{j,(k,i)}$ of (MP2'). Let $\hat{x}_{j,(k,i)} := \hat{y}_{j,(k,i)} \frac{\lambda_{(k,i)} a_k}{c_k}$. Then $\hat{x}_{j,(k,i)}$ is the solution for the reduced maximum concurrent multicommodity exactly-k-splittable flow problem. Similar to Stage 5 of Algorithm 1, we scale down the flow on every edge by a factor $\alpha := \max_j \frac{\sum_{(k,i)} \hat{x}_{j,(k,i)}}{\mu_j}$ such that the edge capacities on (v_j, u_j), $\forall j$ are respected.

Stage 6: Calculation of the original solution $\delta_{j,k}$ and the capacity λ in (MP2). The original $\delta_{j,k}$ in (MP2) can be computed by setting $\delta_{j,k} := \frac{\sum_{i=1}^{c_k} \hat{x}_{j,(k,i)}}{\mu_j}$, $j = 1, ..., M, k = 1, ..., K$. The capacity of (MP2) is $\lambda := \min_k \frac{\sum_{j=1}^{M} \mu_j \delta_{j,k}}{a_k}$.

Analysis of Algorithm 2

Before evaluating the approximation factor of Algorithm 2, by Theorem 1 in [LST90], we observe that the set $\{y_{j,(k,i)}\}$ has the following property found in Proposition 2 of [AAD03].

Lemma 4.6. *Let the total number K' of sub-terminal nodes be $K' = \sum_{k=1}^{K} c_k$. There*

exists a solution of $\{y_{j,(k,i)}\}$ *in which at least* $M \cdot K' + 1 - M - K'$ *elements are zero.*

Proof. The optimal solution $\{x^*_{j,(k,i)}\}$ is a vertex of the polyhedron of (MP2"). (MP2") has $M \cdot K' + 1$ variables and $M + K'$ constraints (excluding the constraint (3)). Therefore, there are $M + K'$ basic variables in any basic solution. The remaining $M \cdot K' + 1 - M - K'$ non-basic variables in $\{x^*_{j,(k,i)}\}$ are zero. Since $y_{j,(k,i)} = \frac{x^*_{j,(k,i)} c_k}{\lambda_{(k,i)} a_k}$, we know that in $\{y_{j,(k,i)}\}$, at least $M \cdot K' + 1 - M - K'$ elements are zero.

□

Corollary 4.7. *The bipartite graph with edge weights* $y_{j,(k,i)}$ *in the scheduling problem has no more edges (with positive weights) than nodes.*

Proof. There are $M + K'$ nodes in the bipartite graph. According to Lemma 4.6, the number of edges with positive weights is at most $M + K' - 1$, which is smaller than the number of nodes. □

Theorem 4.8. *The procedure Int_Schedule can round any feasible solution* $y_{j,(k,i)}$ *into an integer solution* $\hat{y}_{j,(k,i)}$ *with makespan at most* $(1 + \rho)T$.

Proof. According to the procedure *Int_Schedule*, it is straightforward that $\hat{y}_{j,(k,i)} \in \{0, 1\}$ and

$$\sum_{(k,i)} \hat{y}_{j,(k,i)} = 1, j = 1, ..., M.$$

For any machine j in the bipartite graph, since $\hat{y}_{j,(k,i)} \leq y_{j,(k,i)} + 1$ and $p_{j,(k,i)} \leq \rho \cdot T$, we have

$$\begin{aligned}
\sum_{(k,i)} p_{j,(k,i)} \hat{y}_{j,(k,i)} &\leq \sum_{(k,i)} p_{j,(k,i)} (y_{j,(k,i)} + 1) \\
&= \sum_{(k,i)} p_{j,(k,i)} y_{j,(k,i)} + p_{j,(k,i)} \\
&\leq \sum_{(k,i)} p_{j,(k,i)} y_{j,(k,i)} + \rho \cdot T \\
&\leq T + \rho \cdot T.
\end{aligned}$$

□

Theorem 4.9. *Algorithm 2 is a* $\frac{1}{2(1+\rho)}$*-approximation algorithm for solving (MP2').*

Proof. Let the optimal solution of (MP2') be λ^*. We know that solving the maximum concurrent multicommodity uniform exactly-k-splittable flow problem guarantees at least $1/2$ of the optimal solution λ^*. According to Theorem 4.8, the makespan is at most $(1+\rho)T$, thus we can achieve at least $\frac{1}{1+\rho}$ of the optimal capacity of the problem (MP2"). Summarizing the above arguments, at least $\frac{1}{2(1+\rho)}$ of the optimal solution λ^* can be achieved. □

4.4 The case $\mu = \alpha \cdot \beta^T$

Given an $M \times 1$ vector α and a $K \times 1$ vector β, if the $M \times K$ matrix $\mu = \alpha \cdot \beta^T$, we show that this case can be solved by Algorithm 1 or Algorithm 2.

Since $\mu = \alpha \cdot \beta^T$, the (j,k) entry $\mu_{j,k}$ of the matrix μ will be $\mu_{j,k} = \alpha_j \beta_k$. We define $x_{j,k} := \alpha_j \delta_{j,k}$ for all j, k and $b_k := a_k/\beta_k$ for all k. Now the problem (MP) can be written as follows,

$$
\begin{aligned}
\max \quad & \lambda && (MP') \\
\text{s.t.} \quad & \sum_{j=1}^{M} x_{j,k} \geq \lambda b_k, k = 1, ..., K && (1) \\
& \sum_{k=1}^{K} x_{j,k} \leq \alpha_j, j = 1, ..., M && (2) \\
& \sum_{j=1}^{M} \chi\{\delta_{j,k} > 0\} \leq c_k, k = 1, ..., K && (3) \\
& x_{j,k} \geq 0, k = 1, ..., K, j = 1, ..., M. && (4)
\end{aligned}
$$

Obviously, (MP') falls into the case of (MP2'). Therefore, we can apply Algorithm 1 or Algorithm 2 to calculate an approximate solution $x_{j,k}$ and generate the final solution of TDCCP by setting $\delta_{j,k} := x_{j,k}/\alpha_j$ for all j, k.

4.5 Solving the general case

When the service rates $\mu_{j,k}$ are arbitrary, we will have the general case of (MP). In this section, an approximation algorithm will be given for solving the general (MP).

The idea of designing the approximation algorithm for (MP) is as follows: First, we transform (approximately) the general (MP) to an instance of (MP2'). Then, we solve the reduced case by Algorithm 1 or Algorithm 2.

Let μ_j^{max} be the maximal service rate of server j over all classes and μ_j^{min} be the minimum service rate of server j over all classes. We define $\omega_j := \mu_j^{max}/\mu_j^{min}$ and $\omega_{max} := \max_j\{\omega_j\}$. Note that if $\mu_{j,k} = 0$ then server j cannot work at class k ($\delta_{j,k} = 0$). In this thesis, we will assume that $\mu_{j,k} > 0$ for all j, k. Now we try to solve the following problem:

$$\begin{aligned}
\max \quad & \lambda && (MP3)\\
\text{s.t.} \quad & \sum_{j=1}^{M} \mu_{j,k}\delta_{j,k} \geq \lambda a_k, k = 1, ..., K && (1)\\
& \sum_{k=1}^{K} \mu_{j,k}\delta_{j,k} \leq \mu_j^{max}, j = 1, ..., M && (2)\\
& \sum_{j=1}^{M} \chi\{\delta_{j,k} > 0\} \leq c_k, k = 1, ..., K && (3)\\
& \delta_{j,k} \geq 0, k = 1, ..., K, j = 1, ..., M. && (4)
\end{aligned}$$

Obviously, this problem is the same formulation as (MP2') in Section 4.2. Let the optimal solution of general (MP) be λ^*, δ^* and the optimal solution of (MP3) be $\hat{\lambda}^*$, $\hat{\delta}^*$. We can prove the following lemma.

Lemma 4.10. *For the optimal solution of (MP) and (MP3), we have* $\hat{\lambda}^* \geq \lambda^*$.

Proof. It is clear that λ^* and $\delta_{j,k}^*$ must satisfy all constraints in (MP). We show that λ^* and $\delta_{j,k}^*$ is also a feasible solution of (MP3).

Because constraints (1), (3) and (4) in (MP) are exactly the same as those corresponding constraints in (MP3), we only need to prove that λ^* and $\delta_{j,k}^*$ also satisfy

constraint (2) in (MP3). Since

$$\sum_{k=1}^{K} \frac{\mu_{j,k}}{\mu_j^{max}} \delta_{j,k}^* \leq \sum_{k=1}^{K} \delta_{j,k}^*,$$

and according to constraint (2) in (MP), we have

$$\sum_{k=1}^{K} \delta_{j,k}^* \leq 1,$$

therefore

$$\sum_{k=1}^{K} \mu_{j,k} \delta_{j,k}^* \leq \mu_j^{max}.$$

Hence we know $\delta_{j,k}^*$ and λ^* is also a feasible solution of (MP3). Thus we derive $\hat{\lambda}^* \geq \lambda^*$. □

Suppose the (approximate) solution of (MP3) is $\tilde{\lambda}$ and $\tilde{x}_{j,k}$. We define

$$\delta_{j,k} := \frac{\tilde{x}_{j,k}}{(\omega_j \mu_{j,k})}, \forall j, k. \tag{4.4}$$

We know that $\omega_{max} = \max_j\{\omega_j\}$. Then the following theorem can be proved by applying Algorithm 1 for (MP3):

Theorem 4.11. *Solution (4.4) is a feasible solution of (MP), and achieves a λ of value at least $\lambda^*/10\omega_{max}$.*

Proof. Suppose the optimal solution of (MP3) is $\hat{\lambda}^*$. From Algorithm 1 and Theorem 4.5, we know

$$\tilde{\lambda} \geq \frac{1}{10\hat{\lambda}^*}.$$

By (4.4), we have

$$\begin{aligned}\sum_{k=1}^{K}\delta_{j,k} &= \frac{1}{\omega_j}\sum_{k=1}^{K}\frac{\tilde{x}_{j,k}}{\mu_{j,k}}\\ &\leq \frac{1}{\omega_j}\sum_{k=1}^{K}\frac{\tilde{x}_{j,k}}{\mu_j^{min}}\\ &= \frac{1}{\omega_j}\cdot\frac{\sum_{k=1}^{K}\tilde{x}_{j,k}}{\mu_j^{min}}\\ &\leq \frac{1}{\omega_j}\cdot\frac{\mu_j^{max}}{\mu_j^{min}}\\ &= 1,\end{aligned}$$

which satisfies the constraint (2) of (MP).

Furthermore, we observe that,

$$\sum_{j=1}^{M}\mu_{j,k}\delta_{j,k} = \sum_{j=1}^{M}\frac{\tilde{x}_{j,k}}{\omega_j} \geq \frac{\sum_{j=1}^{M}\tilde{x}_{j,k}}{\omega_{max}} \geq \frac{\tilde{\lambda}a_k}{\omega_{max}} \geq \frac{\hat{\lambda}^* a_k}{10\omega_{max}}.$$

According to Lemma 4.10, we know

$$\sum_{j=1}^{M}\mu_{j,k}\delta_{j,k} \geq \frac{\hat{\lambda^*}}{10\omega_{max}} \geq \frac{\lambda^*}{10\omega_{max}}.$$

Therefore, we get the result $\lambda \geq \lambda^*/10\omega_{max}$. □

Similarly, by using Algorithm 2 and Theorem 4.9, we also have the following theorem:

Theorem 4.12. *Solution (4.4) is a feasible solution of (MP), and achieves a λ of value at least $\lambda^*/2(1+\rho)\omega_{max}$.*

4.6 NP-Completeness

Given a set A of even cardinality $n = |A|$ with n numbers $\{s_1, s_2, ..., s_n\}$, the PARTITION problem is to decide whether we can partition the set A into two sets with cardinality $n/2$, where the sums of numbers in two sets are the same. More precisely, we can define the PARTITION problem as follows,

PARTITION=$\{(A, I)|$ There is a subset $I \subseteq \{1, ..., n\}$ of cardinality $n/2$ such that $\sum_{i \in I} s_i = \sum_{i \in A \setminus I} s_i\}$.

The PARTITION problem is one of the NP-complete problems in Karp's 21 NP-Complete problems. For more details, refer to [Kar72].

We prove the NP-Completeness of TDCCP by reducing the PARTITION problem to the special case (MP2'), (which is equivalent to (MP2)). The decision version of problem (MP2') can be stated as follows: We are given an instance (MP2') of the problem TDCCP and a capacity $\lambda^* \in \mathbb{R}$. Is the solution of (MP2') greater than or equal to λ^*? More specifically, we define (MP2') as follows,

TDCCP=$\{((\text{MP2}'), \lambda^*)|$ The solution of (MP2') is greater than or equal to $\lambda^*\}$.

Now we can prove the following theorem and show a polynomial time reduction for the PARTITION problem.

Theorem 4.13. *PARTITION has a solution iff the solution λ of (MP2') satisfies $\lambda \geq \lambda^*$.*

Proof. We consider the PARTITION instance which has a set $A = \{s_1, s_2, ..., s_n\}$ with even cardinality $|A| = n$. Let $S := \sum_{j=1}^{n} s_j$ be the sum of all of the numbers. We construct an instance of (MP2') corresponding to the given PARTITION instance as follows: Let the number of classes $K := 2$, the number of servers $M := n$ and set $\mu_j := s_j$, $j = 1, ..., M$. Let $c_1 = c_2 := n/2$ and $a_1 = a_2 := 1$. Finally we define $\lambda^* := S/2$. Obviously, this construction is in polynomial time.

First we show that if PARTITION has a solution which splits the set A into two sets I and $A \setminus I$, each with the sum of numbers $S/2$, then the solution of the above instance of (MP2') has the solution $\lambda = \lambda^* = S/2$. By setting $x_{j,1} := \mu_j$ and $x_{j,2} := 0$, $\forall j \in I$ and $x_{j,1} := 0$ and $x_{j,2} := \mu_j$, $\forall j \in A \setminus I$, we have,

$$\lambda = (\sum_{j \in I} x_{j,1} + \sum_{j \in A \setminus I} x_{j,1})/a_1 = (\sum_{j \in I} x_{j,2} + \sum_{j \in A \setminus I} x_{j,2})/a_2 = S/2,$$

and the flexibility at each class is $n/2$. Thus, we find a feasible solution of the instance of (MP2') with $\lambda = \lambda^*$.

Now we show that if the solution of the instance of (MP2') achieves $\lambda = \lambda^* = S/2$, then set A can be partitioned into two sets I and $A \setminus I$, each with the sum of numbers

$S/2$. Let $x_{j,k}$, $j = 1, ..., n, k = 1, 2$ be the server assignment of server j to class k in the optimal solution λ. Because of the optimality of λ, we know that for any j, $x_{j,1}$ and $x_{j,2}$ cannot be both zero, otherwise λ is not the optimal solution.

Then we show either $x_{j,1} = 0$ or $x_{j,2} = 0$. Given the solution $x_{j,k}$, $j = 1, ..., n, k = 1, 2$, we choose any two servers j and l which have $x_{j,1}x_{j,2} > 0$ and $x_{l,1}x_{l,2} > 0$. Now we set,

$$\begin{aligned} x_{j,1} &:= x_{j,1} - \min\{x_{j,1}, x_{l,2}\} \\ x_{j,2} &:= x_{j,2} + \min\{x_{j,1}, x_{l,2}\} \\ x_{l,1} &:= x_{l,1} + \min\{x_{j,1}, x_{l,2}\} \\ x_{l,2} &:= x_{l,2} - \min\{x_{j,1}, x_{l,2}\} \end{aligned}$$

By the above step, we know that either $x_{j,1} = 0$ or $x_{l,2} = 0$. Also, the feasibility and the optimal solution λ will not be violated. We can continue this step until there is at most one server j with $x_{j,1}x_{j,2} > 0$.

Suppose there exists a server j with $x_{j,1}x_{j,2} > 0$. Then according to the flexibility constraint,

$$\sum_{j=1}^{n} \chi\{x_{j,k} > 0\} \leq n/2, k = 1, 2$$

we know that there are at most $n - 1$ servers that are used. Therefore there exists at least one on server l with $x_{l,1}x_{l,2} = 0$. Since in the optimal solution, we know that

$$\begin{aligned} \sum_{j=1}^{n} x_{j,1} &\geq \lambda a_1 = S/2, \\ \sum_{j=1}^{n} x_{j,2} &\geq \lambda a_2 = S/2. \end{aligned}$$

As server l is not used, we have

$$\sum_{j=1}^{n} \sum_{k=1}^{2} x_{j,k} \leq S - s_l.$$

which gives us a contradiction. Hence we know in the optimal solution, a server j with $x_{j,1}x_{j,2} > 0$ does not exist and for any j, we have either $x_{j,1} = 0$ or $x_{j,2} = 0$.

Since we have $x_{j,1}x_{j,2} = 0$, we know that if $x_{j,1} = 0$, then $x_{j,2} = \mu_j$ (otherwise $\lambda = S/2$ cannot be achieved when $x_{j,2} < \mu_j$). Finally we define the set $I := \{j | x_{j,1} = 0\}$. Thus, the sums of numbers in two sets are

$$\sum_{j \in I} s_j = \sum_{j \in A \setminus I} s_j = S/2.$$

□

Theorem 4.14. *(MP2') is NP-Complete.*

Proof. Clearly, the problem (MP2') is in NP. In Theorem 4.13, we have show a polynomial reduction from the PARTITION problem to an instance of (MP2'). Since the PARTITION problem is NP-Complete, we know that (MP2') is also NP-Complete.

□

Corollary 4.15. *The general TDCCP is NP-Complete.*

Chapter 5

Total Discrete Capacity Constrained Problem with Costs

In this chapter, we will discuss the Total Discrete Capacity Constrained Problem (TDCCP) with Costs. In this problem, a server incurs a cost when it is working at a class. The total cost of the system is the sum of costs of all servers. Given a budget, the objective of this problem is still to maximize the capacity of the queueing network while the total cost of the system respects the given budget. First, we will show if all the servers are identical, then we can compute the maximal capacity λ^* in polynomial time. Second, if all the classes are identical, we will design two approximation algorithms and give their approximation factors. Then, for the general case of TDCCP with cost, an approximation algorithm is also proposed. Since the NP-Complete problem of TDCCP without cost is a special case of TDCCP with cost (by setting all costs to be zero), we finally derive its NP-Completeness.

5.1 Overview of solving TDCCP with costs

Similar to the way of solving TDCCP, we first show a polynomial algorithm for solving TDCCP with costs when $\mu_{j,k} = \mu_k$. Then we design two approximation algorithms Algorithm 3 and Algorithm 4 for solving TDCCP with costs when $\mu_{j,k} = \mu_j$. For the case of $\mu = \alpha \cdot \beta^T$ where α is a $M \times 1$ vector and β is a $K \times 1$ vector, we show that this case will fall into the case of $\mu_{j,k} = \mu_j$. Finally, we also give the approximation

algorithms for solving the general TDCCP with costs by applying Algorithm 3 or Algorithm 4. Figure 5.1 shows the framework of our algorithms for solving TDCCP with costs.

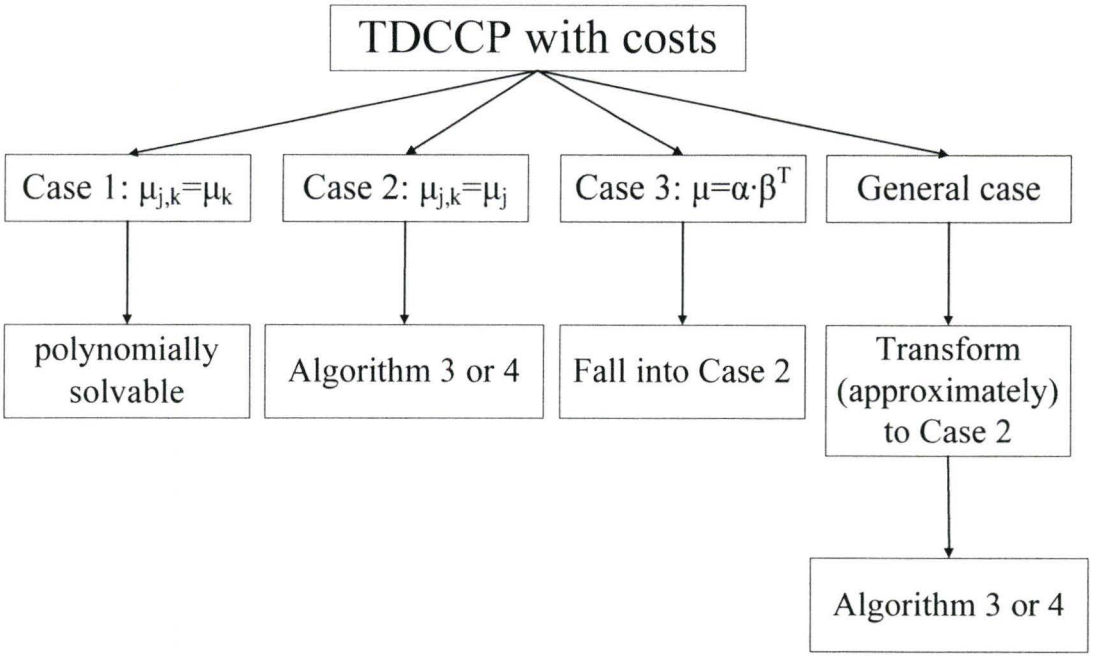


Figure 5.1: The Framework of our algorithms for solving TDCCP with costs

5.2 Solving TDCCP with costs in the case of $\mu_{j,k} = \mu_k$

In this case, the service rates are independent of the servers. In other words, all the servers are identical. We also assume that the costs of servers only depend on classes in the queueing network. Then we have $r_{j,k} = r_k$ for all j. We rewrite the original

problem (MPC1) as follows.

$$\begin{aligned}
\max \quad & \lambda && (MPC1') \\
\text{s.t.} \quad & \sum_{j=1}^{M} \mu_k \delta_{j,k} \geq \lambda a_k, k = 1, ..., K && (1) \\
& \sum_{k=1}^{K} \delta_{j,k} \leq 1, j = 1, ..., M && (2) \\
& \sum_{k=1}^{K} \sum_{j=1}^{M} r_k \delta_{j,k} \leq C, && (3) \\
& \sum_{j=1}^{M} \chi\{\delta_{j,k} > 0\} \leq c_k, k = 1, ..., K && (4) \\
& \delta_{j,k} \geq 0, k = 1, ..., K, j = 1, ..., M. && (5)
\end{aligned}$$

Let the optimal solution for (MPC1') be $\delta^*_{j,k}$ and λ^*. Similarly to Theorem 4.1, we can prove the following theorem.

Theorem 5.1. *If $\mu_{j,k} = \mu_k$ for all j, the maximal capacity of (MPC1') is*

$$\lambda^* = \min\left(\frac{M}{\sum_{k=1}^{K} a_k/\mu_k}, \min_{1 \leq k \leq K}\left(\frac{c_k \mu_k}{a_k}\right), \frac{C}{\sum_{k=1}^{K} a_k r_k/\mu_k}\right).$$

Proof. This theorem can be proved by a similar way to Theorem 4.1. When the optimal solution λ^* and $\delta^*_{j,k}$ is achieved, we consider the following two conditions, which cover all possibilities and one of which must be satisfied:

(1) : $\sum_{j=1}^{M} \delta^*_{j,k} < c_k$, for $k = 1, ..., K$.

(2) : There exists at least one class l such that $\sum_{j=1}^{M} \delta^*_{j,l} = c_l$.

If condition (1) is satisfied, either constraints (1) and (2) in (MPC1') are tight, or constraints (1) and (3) in (MPC1') are tight. Otherwise, λ^* and $\delta^*_{j,k}$ can be trivially

increased. According to Theorem 4.1, when constraints (1) and (2) in (MPC1') are tight, we know that

$$\lambda^* = \frac{M}{\sum_{k=1}^{K}(a_k/\mu_k)}. \tag{5.1}$$

Now suppose constraints (1) and (3) are tight, we have

$$\sum_{k=1}^{K} r_k \cdot \sum_{j=1}^{M} \delta_{j,k}^* = C.$$

According to constraint (1), we know $\sum_{j=1}^{M} \delta_{j,k}^* = \lambda^* \frac{a_k}{\mu_k}$. Then

$$\sum_{k=1}^{K} r_k \lambda^* \frac{a_k}{\mu_k} = \sum_{k=1}^{K} r_k \cdot \sum_{j=1}^{M} \delta_{j,k}^* = C,$$

thus we have

$$\lambda^* = \frac{C}{\sum_{k=1}^{K} a_k r_k/\mu_k}. \tag{5.2}$$

If condition (2) is satisfied, we have proved that

$$\lambda^* = \min_{1\le k\le K} \frac{c_k \mu_k}{a_k}. \tag{5.3}$$

Therefore, by inequality (5.1), (5.2) and (5.3), the maximal capacity λ^* of (MPC1') is,

$$\lambda^* = \min\left(\frac{M}{\sum_{k=1}^{K} a_k/\mu_k}, \min_{1\le k\le K}(\frac{c_k\mu_k}{a_k}), \frac{C}{\sum_{k=1}^{K} a_k r_k/\mu_k}\right).$$

□

Corollary 5.2. *There is a polynomial time algorithm for solving (MPC1) in the case of $\mu_{j,k} = \mu_k$ and $r_{j,k} = r_k$ for all j.*

5.3 Solving TDCCP with costs in the case of $\mu_{j,k} = \mu_j$

5.3.1 Approximation Algorithm 3

Design of Algorithm 3

In Chapter 2, we introduced the problem (MPC2) for this case, which is as follows.

$$\begin{aligned}
&\max \quad \lambda && (MPC2)\\
&\text{s.t.} \quad \sum_{j=1}^{M} \mu_j \delta_{j,k} \geq \lambda a_k, k = 1, ..., K && (1)\\
&\sum_{k=1}^{K} \delta_{j,k} \leq 1, j = 1, ..., M && (2)\\
&\sum_{k=1}^{K} \sum_{j=1}^{M} r_{j,k} \delta_{j,k} \leq C, && (3)\\
&\sum_{j=1}^{M} \chi\{\delta_{j,k} > 0\} \leq c_k, k = 1, ..., K && (4)\\
&\delta_{j,k} \geq 0, k = 1, ..., K, j = 1, ..., M. && (5)
\end{aligned}$$

Let $x_{j,k} := \mu_j \delta_{j,k}$ and $\hat{r}_{j,k} := r_{j,k}/\mu_j$, $\forall j, k$,then we can rewrite (MPC2) as (MPC2'):

$$\max \quad \lambda \qquad (MPC2')$$

$$\text{s.t.} \quad \sum_{j=1}^{M} x_{j,k} \geq \lambda a_k, k = 1, ..., K \qquad (1)$$

$$\sum_{k=1}^{K} x_{j,k} \leq \mu_j, j = 1, ..., M \qquad (2)$$

$$\sum_{k=1}^{K} \sum_{j=1}^{M} \hat{r}_{j,k} x_{j,k} \leq C, \qquad (3)$$

$$\sum_{j=1}^{M} \chi\{x_{j,k} > 0\} \leq c_k, k = 1, ..., K \qquad (4)$$

$$x_{j,k} \geq 0, k = 1, ..., K, j = 1, ..., M. \qquad (5)$$

We can treat (MPC2') as an instance of the constrained maximum concurrent multicommodity k-splittable flow problem, with exactly the same network structure as (MP2') (refer to Figure 4.2). The only thing we add is the cost on each edge, which is given as follows: First, we define the cost on edge (S, v_j) and the cost on edge (v_j, u_j), $j = 1, ..., M$ to be zero. Then, we define the cost on edge (u_j, t_k) to be $\hat{r}_{j,k}$, $\forall j, k$. Clearly, C is the maximum budget allowed in this network.

Without the integer constraint (4), (MPC2') is a linear programming problem, which can be viewed as the fractional version of the constrained maximum concurrent multicommodity flow problem. Let the optimal solution of (MPC2') without constraint (4) be $\hat{\lambda}^*$ and the optimal solution of (MPC2') be λ^* and $x^*_{j,k}$. We know that $\hat{\lambda}^*$ is an upper bound for λ^*. We define the demand of node t_k to be $\hat{\lambda}^* a_k, k \in \{1, ..., K\}$, and at most c_k paths are allowed for sending commodity k. Similar to Algorithm 1, by computing the minimum congestion, we can get the solution of the maximum concurrent multicommodity flow problem.

The idea of designing Algorithm 3 is as follows: First, we transform (MPC2') into the constrained maximum concurrent multicommodity uniform exactly-k-splittable flow problem by splitting each terminal node t_k into c_k sub-terminal nodes (k, i) $i = 1, ..., c_k$, each with the sub-commodity demand a_k/c_k. The cost $\hat{r}_{j,(k,i)}$ on the edge $(j, (k, i))$ is equal to the cost $\hat{r}_{j,k}$. We solve the fractional version of this reduced

problem without constraint (4) in (MPC2'), which is as follows:

$$\begin{aligned}
\max \quad & \lambda && (MPC2'')\\
\text{s.t.} \quad & \sum_{j=1}^{M} x_{j,(k,i)} \geq \lambda a_k/c_k, \forall (k,i) && (1)\\
& \sum_{(k,i)} \sum_{j=1}^{M} \hat{r}_{j,(k,i)} x_{j,(k,i)} \leq C, && (2)\\
& \sum_{(k,i)} x_{j,(k,i)} \leq \mu_j, \forall j && (3)\\
& x_{j,(k,i)} \geq 0, \forall j, (k,i) && (4)
\end{aligned}$$

Let the solution of (MPC2") be $\hat{\lambda}^*$ and $\hat{x}^*_{j,(k,i)}$, $\forall j, (k,i)$. Clearly, the number $\hat{\lambda}^*$ is the maximum fraction we can achieve for routing all sub-commodities simultaneously. Then we set each sub-terminal with a new demand $\hat{\lambda}^* a_k/c_k$ and try to find an unsplittable flow satisfying each sub-terminal demand without violating the cost budget.

The congestion α and the flow $x_{j,(k,i)} = a_k/c_k$ on edge $(j,(k,i))$ can be decided when we find the unsplittable flow for sending each sub-commodity $(k.i)$. If $\alpha > 1$, we know that the edge capacity on (v_j, u_j) will be violated when we route sub-commodities (k,i) unsplittably. Finally we scale down the flow on each edge by a factor α and obtain the original solution and capacity for the problem (MPC2').

Description of Algorithm 3

Algorithm 3 applies the algorithm for minimizing congestion in the unsplittable min-cost flow problem, which is studied in [Sku02]. First we give the following definitions.

Definition 5.3. Given $a, b \in \mathbb{R}^+$, b is *a-integral* if and only if $b \in a \cdot \mathbb{N}$.

Definition 5.4. A flow is called *a-integral* flow if the flow amount f_e on edge e is *a-integral* for all $e \in E$.

Then we give Theorem 1 from [Sku02]:

Theorem 5.5. *Let $G = \langle V, E\rangle$ be a directed graph with capacities and costs on the edges. Moreover, there is a source vertex $s \in V$ and k sinks $t_1, ..., t_k \in V$ with demands $a_1, ..., a_k$.*

(a) *There exists a feasible (splittable) flow satisfying all demands if and only if, for any subset $T \subseteq V \setminus \{S\}$, the sum of capacities of edges in the directed cut $(V \setminus T, T)$ is at least $\sum_{i:t_i \in T} d_i$. We refer to the condition that the sum of capacities of edges in the directed cut $(V \setminus T, T)$ is at least $\sum_{i:t_i \in T} d_i$ as the cut condition.*

(b) *If the cut condition is satisfied and all demands and capacities are a-integral for some $a \in \mathbb{R}^+$, then there exists a feasible (splittable) flow satisfying all demands with minimum cost such that the flow value on any edge is a-integral. Moreover, such a flow can be computed in polynomial time.*

Computing the *a-integral* flow without increasing the total cost can be easily performed. Suppose we are given a graph $G = \langle V, E\rangle$ with k commodities. We assume that all demands and capacities are *a-integral* in G. We also assume that each commodity is sent through edges with capacities greater than or equal to its demand (note that this assumption does not always hold in general). If the cut condition is satisfied and we have a feasible splittable flow f satisfying all demands, let $f_k(e)$ denote the flow amount of commodity k on edge e, $k = 1, ..., K$ and $e \in E$. Now consider a subgraph of G in which each $f_k(e)$ is not *a-integral*. Since in this subgraph, all the demands are *a-integral* but the edges entering demand nodes do not carry an *a-integral* flow amount, the degree of each demand node is at least two. Therefore, we can determine an alternating cycle by using the same step we applied in Algorithm 1. Then we augment the flow in the forward path and decrease the flow in the backward path until the flow on one edge becomes *a-integral* if the cost of flow is not increased after this augmentation. Otherwise, we augment the flow in the backward path and decrease the flow in the forward path until the flow on one edge becomes *a-integral*. We delete all edges with *a-integral* flow and continue iteratively until all edges are deleted from the subgraph. This process terminates after at most $|E|$ iterations. Clearly, there are $|V|$ nodes in G, thus, the running time is $O(|V| \cdot |E|)$.

The key step of Algorithm 3 is the approximation algorithm for finding the unsplittable min-cost flow, which also applies the above method for finding the *a-integral*

flow (see, e.g., [Sku02]). Let $a_{min} := \min_k\{a_k\}$ and $a_{max} := \max_k\{a_k\}$. We will round down each demand a_k in our problem to

$$\hat{a}_k := a_{min} \cdot 2^{\lfloor log(a_k/a_{min}) \rfloor}.$$

Obviously, $\hat{a}_k$, $k = 1, ..., K$ are a_{min}-*integral*. By applying Algorithm 1 in [Sku02], we first introduce the following procedure *Mincost_USP*. Taking a given feasible splittable flow as the input, procedure *Mincost_USP* generates the unsplittable min-cost flow for the problem (MPC2'). We know that only (v_j, u_j), $j = 1, ..., M$ in (MPC2') are capacitated edges, and we will further prove that the flow value on edge (v_j, u_j) is less than $2f^0_{(v_j,u_j)} + a_{max}$, where $f^0_{(v_j,u_j)}$ is the initial flow on edge (v_j, u_j).

Procedure *Mincost_USP*

Input: The graph $G = \langle V, E\rangle$ corresponding to the problem (MPC2') with non-negative cost $\hat{r}_{j,k}$ on the edge (u_j, t_k), for all j and k, and a feasible splittable flow $f^0_{e\in E}$ satisfying all demands and the budget C.

Output: An unsplittable min-cost flow $x_{j,k}$, which is the flow amount on edge (u_j, t_k) and indicates the paths for sending commodity k from source node S to each terminal node t_k, $j = 1, ..., M, k = 1, ..., K$.

Step 1: Let $a_{min} := \min_k\{a_k\}$ and $a_{max} := \max_k\{a_k\}$. Round each demand to be $\hat{a}_k = a_{min} \cdot 2^{\lfloor log(a_k/a_{min}) \rfloor}, 1 \le k \le K$.

Step 2: Decrease the flow along the path $S \to v_j \to u_j \to t_k$ with the most expensive cost $\hat{r}_{j,k}$ until the flow entering t_k has been decreased by $a_k - \hat{a}_k$, $k = 1, ..., K$. After this step, the resulting flow $f^0_{e\in E}$ will satisfy each rounded demand $\hat{a}_k$.

Step 3: Set $i := 0$.

Step 4: While $\hat{a}_{min} \cdot 2^i \le \hat{a}_{max}$ do

(a): Set $i := i + 1$ and $d_i := \hat{a}_{min} \cdot 2^{i-1}$.

(b): For each edge (v_j, u_j) in G, set its capacity μ_j to $f^{i-1}_{(v_j,u_j)}$ rounded up to the nearest multiple of d_i.

(c): Compute a feasible d_i-*integral* flow $f^i_{e\in E}$ satisfying all demands $\hat{a}_{min} \cdot 2^{i-1}$ without increasing the total cost.

(d): Remove all edges e with $f^i_{e\in E} = 0$ from graph G.

(e): Set $i_1 := 1$.

(f): While $i_1 \leq k$ and $\hat{a}_{i_1} = d_i$ do

(i): Arbitrarily determine an edge (v_j, u_j) and assign commodity i_1 with flow amount $\hat{a}_{i_1}$ on edge (v_j, u_j). Then we set $\hat{x}_{j,i_1} := \hat{a}_{i_1}$.

(ii): Decrease flow along path $S \rightarrow v_j \rightarrow u_j \rightarrow t_{i_1}$ by $\hat{a}_{i_1}$.

(iii): Remove all edges e with $f^i_{e\in E} = 0$ from graph G.

(iv): $i_1 := i_1 + 1$.

Step 5: $\hat{x}_{j,k}$ is the rounded unsplittable min-cost flow. Let the final unsplittable min-cost flow $x_{j,k} := a_k$ if $\hat{x}_{j,k} = \hat{a}_k$, $j = 1, ..., M, k = 1, ..., K$. Return $x_{j,k}$, $j = 1, ..., M, k = 1, ..., K$.

Let $f^0_{(v_j,u_j)}$ be the initial flow on edge (v_j, u_j) and $f_{(v_j,u_j)}$ be the final unsplittable flow through edge (v_j, u_j). Obviously, $f_{(v_j,u_j)} = \sum_{k=1}^{K} x_{j,k}$. Similar to Theorem 3 in [Sku02], we can prove the following theorem.

Theorem 5.6. *The procedure Mincost_USP finds an unsplittable flow whose cost is bounded by the budget C and the flow value on edge (v_j, u_j) is less than or equal to $2f^0_{(v_j,u_j)} + a_{max}$, $j = 1, ..., M$. More precisely, the sum of all but one demand routed across any edge (v_j, u_j) is less than twice the initial flow value on (v_j, u_j).*

Proof. Let $x^0_{j,k}$ be the initial fractional flow on edge (u_j, t_k), $\hat{x}^0_{j,k}$ be the rounded fractional flow on edge (u_j, t_k), $\hat{x}_{j,k}$ be the rounded unsplittable min-cost flow on edge (u_j, t_k), and $x_{j,k}$ be the final unsplittable min-cost flow on edge (u_j, t_k), $j = 1, ..., M$, $k = 1, ..., K$. According to the definition of our graph G, we know that the cost $\hat{r}_{j,k}$ only exists on edge (u_j, t_k), $j = 1, ..., M, k = 1, ..., K$. In each loop of Step 4, the total cost in G never increases and the cost of the rounded unsplittable min-cost flow $\hat{x}_{j,k}$ is bounded by the cost of the initial fractional flow $\hat{x}^0_{j,k}$. Thus we have

$$\sum_{j=1}^{M}\sum_{k=1}^{K} \hat{r}_{j,k}\hat{x}_{j,k} \leq \sum_{j=1}^{M}\sum_{k=1}^{K} \hat{r}_{j,k}\hat{x}^0_{j,k}. \tag{5.4}$$

After sending all rounded demand, by Step 2 we know that we still have $\Delta x^0_{j,k} = x^0_{j,k} - \hat{x}^0_{j,k}$ amount of flow remaining on the edge (u_j, t_k). Clearly, $\sum_{j=1}^{M} \Delta x^0_{j,k} = a_k - \hat{a}_k$

and the remaining flow must satisfy,

$$\sum_{j=1}^{M}\sum_{k=1}^{K}\Delta x^0_{j,k}\hat{r}_{j,k} \leq \sum_{j=1}^{M}\sum_{k=1}^{K}x^0_{j,k}\hat{r}_{j,k} - \sum_{j=1}^{M}\sum_{k=1}^{K}\hat{x}^0_{j,k}\hat{r}_{j,k}. \tag{5.5}$$

The remaining flow demand $a_k - \hat{a}_k$ will be routed on edge (u_j, t_k) where $\hat{x}_{j,k} > 0$. Thus we know that if $\hat{x}_{j,k} > 0$

$$x_{j,k} - \hat{x}_{j,k} = a_k - \hat{x}_{j,k}. \tag{5.6}$$

By Step 2 of *Mincost_USP*, it is easy to see that $\Delta x^0_{j,k}$ remains on the most expensive edges and the flow amount $\sum_{j=1}^{M}\Delta x^0_{j,k}$ will be finally routed on edge (u_j, t_k), whose cost is less than or equal to the cost of $\Delta x^0_{j,k}$. Therefore we have,

$$\sum_{j=1}^{M}\sum_{k=1}^{K}(x_{j,k} - \hat{x}_{j,k})\hat{r}_{j,k} \leq \sum_{j=1}^{M}\sum_{k=1}^{K}\Delta x^0_{j,k}\hat{r}_{j,k}. \tag{5.7}$$

By (5.5) and (5.7) we have,

$$\sum_{j=1}^{M}\sum_{k=1}^{K}(x_{j,k} - \hat{x}_{j,k})\hat{r}_{j,k} \leq \sum_{j=1}^{M}\sum_{k=1}^{K}x^0_{j,k}\hat{r}_{j,k} - \sum_{j=1}^{M}\sum_{k=1}^{K}\hat{x}^0_{j,k}\hat{r}_{j,k}. \tag{5.8}$$

Combining (5.4) and (5.8),

$$\sum_{j=1}^{M}\sum_{k=1}^{K}x_{j,k}\hat{r}_{j,k} \leq \sum_{j=1}^{M}\sum_{k=1}^{K}x^0_{j,k}\hat{r}_{j,k} \leq C, \tag{5.9}$$

which tells us that the cost of the final unsplittable flow is still bounded by the cost of the initial fractional flow $x^0_{j,k}$ and respects the given budget C.

Now we show that the flow value on edge (v_j, u_j) is less than or equal to $2f^0_{(v_j,u_j)} + a_{max}$. Obviously, the flow on edge (v_j, u_j) will be $f_{(v_j,u_j)} = \sum_{k=1}^{K}x_{j,k}$, $j = 1, ..., M$. In Step 4 of *Mincost_USP*, the capacity μ_j in the i^{th} loop is computed by rounding the flow value $f^{i-1}_{(v_j,u_j)}$ up to the nearest multiple of d_i. We compute a *d_i-integral* flow by increasing the flow on less expensive edges (u_j, t_k) if the flow on (u_j, t_k) is not *d_i-integral*. If the flows on all edges are *d_i-integral*, we choose an arbitrary path from S to t_k for sending commodity k if $\hat{a}_k = d_i$. Then the flow on this path will be decreased by d_i. Note that when the flow on an edge is decreased to 0, we will delete

this edge from G. Thus the capacity μ_j on edge (v_j, u_j) is violated at most once. In other words, we know that the sum of all but one rounded demand $\hat{a}_k$ routed across any edge (v_j, u_j) is less than the initial flow $f^0_{(v_j,u_j)}$ on edge (v_j, u_j), $j = 1, ..., M$.

Consider edge (v_j, u_j), let k_0 be a commodity with the maximal rounded demand $\hat{a}_{k_0}$ that is routed across (v_j, u_j). Except commodity k_0, the sum of other demands routed across (v_j, u_j) is less than the initial flow on (v_j, u_j). Note that $a_k \leq 2\hat{a}_k$ for all commodities k. Therefore we have,

$$\sum_{\substack{k:k \text{ is routed} \\ across\ (v_j,u_j)}} a_k = a_{k_0} + \sum_{\substack{k:k\neq k_0 \text{ and} \\ k \text{ is routed} \\ across\ (v_j,u_j)}} a_k \leq a_{max} + 2 \sum_{\substack{k:k\neq k_0 \text{ and} \\ k \text{ is routed} \\ across\ (v_j,u_j)}} \hat{a}_k \leq a_{max} + 2f^0_{(v_j,u_j)}.$$

□

Now we give the detailed description of Algorithm 3.

Stage 1: Initialization. Construct a network with one source node $\{S\}$ for sending K commodities, $2M$ nodes $\{v_1, ..., v_M, u_1, ...u_M\}$ for transferring commodities, and K terminal nodes $\{t_1, ..., t_K\}$ with demand $a_k, k = 1, ..., K$ for receiving commodities. Set the capacity on the edge (v_j, u_j) to be μ_j for $j = 1, ..., M$ and the cost on the edge (u_j, t_k) to be $\hat{r}_{j,k} := r_{j,k}/\mu_j$ for $j = 1, ..., M, k = 1, ..., K$.

Stage 2: Transformation of initial problem to a constrained maximum concurrent uniform exactly-k-splittable flow problem. We replicate each terminal node t_k into c_k identical sub-terminals $t_{(k,i)}$, $i = 1, ..., c_k$. Then each commodity k with demand a_k is split into c_k commodities (k, i), $i = 1, ..., c_k$, each with the same demand $a_{(k,i)} := a_k/c_k$ and with the same source. Further, for each edge connecting with terminal node t_k, we connect it with all of the sub-terminals $t_{(k,i)}$ and set the cost on edge $(u_j, t_{(k,i)})$ to be $\hat{r}_{j,k}$, $\forall j, (k, i)$. Note that the new commodities (k, i) must be routed unsplittably.

Stage 3: Calculation of demand $a_{(k,i)}$. We use the ellipsoid method to solve (MPC2"), which is the relaxation of (MP2'). Let $\hat{\lambda}^*$ and $\hat{x}^*_{j,(k,i)}$ be the optimal fractional solution. Then we assign the flow amount $\hat{x}^*_{j,(k,i)}$ on the edge $(u_j, t_{(k,i)})$ and the flow amount $\sum_{(k,i)} \hat{x}^*_{j,(k,i)}$ on the edges (s, v_j) and (v_j, u_j), $j = 1, ..., M$. For each sub-terminal in the network, let its demand be $a^*_{(k,i)} := \hat{\lambda}^* a_k/c_k$. Note the flow $\hat{x}^*_{j,(k,i)}$ computed in this stage is the fractional flow which satisfies the budget constraint and the new demand $a_{(k,i)}$.

Stage 4: Transformation of the fractional flow calculated in Stage 3 into an unsplittable min-cost flow.

Step 1: Let $a_{min} := \min_{(k,i)}\{a^*_{(k,i)}\}$ and $a_{max} := \max_{(k,i)}\{a^*_{(k,i)}\}$. If $a_{max} > 1$, we scale down all sub-terminal demands, edge capacities, flows and the budget by the factor a_{max} so that $a_{(k,i)} \in (0, 1]$ and $\mu_j \in (0, +\infty)$. Let $D := 1/a_{min}$, we know that all demands fall within the interval $[1/D, 1]$. We partition the interval $[1/D, 1]$ into sub-intervals as follows:

$$[1/D, 1/2^{\lfloor logD \rfloor}], ..., (1/2^{i+1}, 1/2^i], ..., (1/2, 1].$$

This step is the same as Step 1 of Stage 4 in Algorithm 3. There are $\lfloor logD \rfloor$ sub-intervals, which are numbered as sub-interval 1, 2,...,$\lfloor logD \rfloor$ from left to right.

Step 2: Similar to the flow augmentation techniques in Stage 4 of Algorithm 3, we do a slight modification on the procedure $Mincost\_USP$ by restricting to unsplittable min-cost flows where commodity (k, i) can be only sent on edges with capacity at least $a_{(k,i)}$. Then, we use the procedure $Mincost\_USP$ to produce the unsplittable min-cost flow $x_{j,(k,i)}$ for each commodity with demand in sub-interval i, $i = 1, ..., \lfloor logD \rfloor$. The total cost of the unsplittable flow will be bounded by the initial cost of the fractional flow $\hat{x}^*_{j,(k,i)}$.

Stage 5: Calculation of the original solution $\delta_{j,k}$ and the capacity λ for (MPC2). If the original $a_{max} > 1$ in Step 1 of Stage 4, we first restore the flow and capacities in the network by multiplying $y_{j,(k,i)}$ and μ_j with a_{max}, for all j, (k, i). Then we scale down flow in every edge by a factor $\alpha := \max_j \frac{\sum_{(k,i)} x_{j,(k,i)}}{\mu_j}$. The final solution $x_{j,(k,i)}$ tells us that each commodity (k, i) will be shipped through the edge (v_j, u_j) unsplittably. Thus the λ that corresponds to this solution is the capacity that we achieved. It is easy to see that $\lambda = \min_{(k,i)} \frac{\sum_{j=1}^{M} x_{j,(k,i)}}{a_{(k,i)}}$. Since $x_{j,k} := \mu_j \delta_{j,k}$, finally we set $\delta_{j,k} := x_{j,k}/\mu_j = \sum_{(k,i)} x_{j,(k,i)}/\mu_j$.

Analysis of Algorithm 3

Similar to Theorem 4.3, first we show that Theorem 5 in [BKS02] can also be applied for the constrained maximum concurrent multicommodity k-splittable flow problem.

Theorem 5.7. *Computing the constrained maximum concurrent multicommodity uniform exactly-k-splittable flow problem in Stage 2 of Algorithm 3 guarantees at least* $1/2$ *of the optimal value for the constrained maximum concurrent multicommodity k-splittable flow problem (MPC2').*

Proof. Suppose the optimal solution of a constrained maximum concurrent multicommodity k-splittable flow problem (MP2') is λ^* and $x^*_{j,k}$. We use the same idea adopted in Theorem 4.3 so that there exists a feasible constrained maximum concurrent multicommodity uniform exactly-k-splittable flow with flow value $D := \lambda^* a_k/(2c_k)$ on each path for sending sub-commodity (k,i), $\forall(k,i)$.

By the same construction in Theorem 4.3, we know that there are at least c_k paths carrying the flow amount D for commodity k, $k = 1, ..., K$. Since our construction does not increase flow on any edge, the cost of the network will never be increased. Clearly, the maximum possible fraction for routing commodity k is at least $\lambda^*/2$ for (MPC2'). □

Using the same steps as in the proof of Theorem 4.4, we can prove the following theorem, which is similar to Theorem 6 in [Sku02]. Note that the sub-intervals in Stage 4 of Algorithm 3 are the same as the sub-intervals used in Algorithm 1.

Theorem 5.8. *Stage 4 of Algorithm 3 finds the unsplittable min-cost flow for sending each sub-commodity without violating the budget C such that the total flow through* (v_j, u_j) *exceeds its edge capacity* μ_j *by less than* $6\mu_j$, $j = 1, ..., M$.

Proof. Because the capacitated edges in G are (v_j, u_j), $j = 1, ..., M$, we only consider flow on those edges. Let the initial flow on edge (v_j, u_j) be $f_{(v_j,u_j)}$ and the final unsplittable min-cost flow on edge (v_j, u_j) be $f'_{(v_j,u_j)}$.

We choose an arbitrary sub-interval $(1/2^{i+1}, 1/2^i]$ from

$$[1/D, 1/2^{\lfloor logD \rfloor}], ..., (1/2^{i+1}, 1/2^i], ..., (1/2, 1].$$

In this interval, we only consider commodity l whose demand a_l is located in this sub-interval. By the modification in Step 2 of Stage 4, the unsplittable min-cost flows for sending commodity l go through edges with capacities greater than or equal to the demand of commodity l. In this interval, the minimum commodity demand is $1/2^{i+1}$, thus we have $\mu_j \geq \frac{1}{2^{i+1}}$. Considering edge (v_j, u_j), by Theorem 5.6, the unsplittable

min-cost flow found in this interval is bounded by $a^i_{max} + 2f^i_{(v_j,u_j)}$, where $a^i_{max} \leq 1/2^i$ is the maximal demand in this interval and $f^i_{(v_j,u_j)}$ is the initial fractional flow in this interval. We know that edge (v_j, u_j) may also be used in other sub-intervals before $(1/2^{i+1}, 1/2^i]$, the flow $f'_{(v_j,u_j)}$ on it is at most:

$$\begin{aligned} f'_{(v_j,u_j)} &\leq \sum_{j=i}^{\lfloor logD \rfloor} (a^i_{max} + 2f^i_{(v_j,u_j)}) \\ &\leq \sum_{j=i}^{\lfloor logD \rfloor} (1/2^j) + \sum_{j=i}^{\lfloor logD \rfloor} 2f^i_{(v_j,u_j)} \\ &= \frac{1}{2^{\lfloor logD \rfloor}}(2^{\lfloor logD \rfloor - i + 1} - 1) + \sum_{j=i}^{\lfloor logD \rfloor} 2f^i_{(v_j,u_j)} \\ &= \frac{1}{2^{i-1}} - \frac{1}{2^{\lfloor logD \rfloor}} + \sum_{j=i}^{\lfloor logD \rfloor} 2f^i_{(v_j,u_j)} \\ &= 4 \cdot \frac{1}{2^{i+1}} - \frac{1}{2^{\lfloor logD \rfloor}} + \sum_{j=i}^{\lfloor logD \rfloor} 2f^i_{(v_j,u_j)}. \end{aligned}$$

Since $\frac{1}{2^{i+1}} \leq \mu_j$ and $\sum_{j=i}^{\lfloor logD \rfloor} f^i_{(v_j,u_j)} \leq f_{(v_j,u_j)} \leq \mu_j$, the total flow on the edge (v_j, u_j) is bounded by,

$$f'_{(v_j,u_j)} \leq 4 \cdot \frac{1}{2^{i+1}} - \frac{1}{2^{\lfloor logD \rfloor}} + \sum_{j=i}^{\lfloor logD \rfloor} 2f^i_{(v_j,u_j)} \leq 4\mu_j - \frac{1}{2^{\lfloor logD \rfloor}} + 2\mu_j \leq 6\mu_j.$$

It tells us that for any edge (v_j, u_j) in the graph, the sum of unsplittable flows through it is at most 6 times its edge capacity. According to Theorem 5.6, we know that the cost of unsplittable flow found in each interval is bounded by the cost of the initial flow. Thus the budget C is still respected. Therefore, there exists a 6-approximation algorithm for the minimum congestion of the min-cost unsplittable flow problem and we can achieve a 1/6-approximation algorithm for the constrained maximum concurrent multicommodity unsplittable flow problem. □

Theorem 5.9. *Algorithm 3 is a* 1/12*-approximation algorithm for solving the problem MPC2.*

Proof. Let the optimal solution of (MPC2') be λ^*. According to Theorem 5.7, solving the constrained maximum concurrent multicommodity uniform exactly-k-splittable flow problem will achieve at least $1/2$ of the optimal solution λ^*. By Theorem 5.8, Stage 4 of Algorithm 3 achieves at least $1/6$ of the optimal solution for the constrained maximum concurrent multicommodity uniform exactly-k-splittable flow problem without violating the budget C. Therefore we achieve at least $1/12$ of the optimal capacity λ^* in (MPC2') while respecting the budget C. □

5.3.2 Improvement of Algorithm 3

Suppose we are given a number $\beta > 1$. Then we can slightly modify Algorithm 3 in the following two steps:

Step 1: In Step 1 of Stage 4, we partition the interval $[1/D, 1]$ into sub-intervals as follows:

$$[1/D, 1/\beta^{\lfloor \log_\beta^D \rfloor}], \ldots. (1/\beta^{i+1}, 1/\beta^i], \ldots, (1/\beta, 1],$$

Clearly, the number of sub-intervals is $\lfloor \log_\beta^D \rfloor + 1$.

Step 2: In Step 1 of the procedure $Mincost\_USP$, we round each demand to be $\hat{a}_k = a_{min} \cdot \beta^{\lfloor \log_\beta a_k/a_{min} \rfloor}, 1 \leq k \leq K$.

Since we have modified the rounded demand in the procedure $Mincost\_USP$ for each commodity, we first prove a new version of Theorem 5.6.

Theorem 5.10. *The procedure $Mincost\_USP$ finds an unsplittable flow whose cost is bounded by the budget C and the flow value on edge (v_j, u_j) is less than or equal to $\beta f^0_{(v_j,u_j)} + a_{max}$, $j = 1, \ldots, M$. More precisely, the sum of all but one demand routed across any edge (v_j, u_j) is less than β times the initial flow value on (v_j, u_j).*

Proof. The proof is the same as the proof of Theorem 5.6, except that we have $a_k \leq \beta \hat{a}_k$ for all commodities k. As a result, we have

$$\sum_{\substack{k:k \text{ is routed} \\ \text{across } (v_j,u_j)}} a_k = a_{k_0} + \sum_{\substack{k:k \neq k_0 \text{ and} \\ k \text{ is routed} \\ \text{across } (v_j,u_j)}} a_k \leq a_{max} + \beta \sum_{\substack{k:k \neq k_0 \text{ and} \\ k \text{ is routed} \\ \text{across } (v_j,u_j)}} \hat{a}_k \leq a_{max} + \beta f^0_{(v_j,u_j)}.$$

□

By applying the same approach as in Theorem 5.8 and together with Theorem 5.10, we can prove the following theorem:

Theorem 5.11. *Stage 4 of Algorithm 3 finds the unsplittable min-cost flow for sending each sub-commodity without violating the budget C such that the total flow through (v_j, u_j) exceeds its edge capacity μ_j by less than $(3+2\sqrt{2})\mu_j$, $j = 1, ..., M$.*

Proof. Let the initial flow on edge (v_j, u_j) be $f_{(v_j,u_j)}$ and the final unsplittable min-cost flow on edge (v_j, u_j) be $f'_{(v_j,u_j)}$. First, we choose an arbitrary sub-interval $(1/\beta^{i+1}, 1/\beta^i]$ from

$$[1/D, 1/\beta^{\lfloor \log_\beta^D \rfloor}], ..., (1/\beta^{i+1}, 1/\beta^i], ..., (1/\beta, 1].$$

We only consider commodity l whose demand a_l is located in this sub-interval. Similar to Theorem 5.8, we know that $\mu_j \geq \frac{1}{\beta^{i+1}}$. By Theorem 5.10, the unsplittable min-cost flow found in this interval is at most $a^i_{max} + \beta f^i_{(v_j,u_j)}$, where $a^i_{max} \leq 1/\beta^i$ is the maximal demand in this interval and $f^i_{(v_j,u_j)}$ is the initial fractional flow in this interval. Since edge (v_j, u_j) may also be used in other sub-intervals, the flow $f'_{(v_j,u_j)}$ on it is bounded by:

$$\begin{aligned} f'_{(v_j,u_j)} &\leq \sum_{j=i}^{\lfloor \log_\beta^D \rfloor} (a^i_{max} + \beta f^i_{(v_j,u_j)}) \\ &\leq \sum_{j=i}^{\lfloor \log_\beta^D \rfloor} (1/\beta^j) + \sum_{j=i}^{\lfloor \log_\beta^D \rfloor} \beta f^i_{(v_j,u_j)} \\ &= \frac{\beta^2/\beta^{i+1}}{\beta - 1} - \frac{(1/\beta)^{\lfloor \log_\beta^D \rfloor}}{\beta - 1} + \sum_{j=i}^{\lfloor \log_\beta^D \rfloor} \beta f^i_{(v_j,u_j)} \end{aligned}$$

We know that $\frac{1}{\beta^{i+1}} \leq \mu_j$ and $\sum_{j=i}^{\lfloor \log_\beta^D \rfloor} f^i_{(v_j,u_j)} \leq f_{(v_j,u_j)} \leq \mu_j$. Thus the total flow on the edge (v_j, u_j) is at most,

$$f'_{(v_j,u_j)} \leq \frac{\beta^2 \mu_j}{\beta - 1} + \beta\mu_j - \frac{(1/\beta)^{\lfloor \log_\beta^D \rfloor}}{\beta - 1} = (\frac{2\beta^2 - \beta}{\beta - 1})\mu_j - \frac{(1/\beta)^{\lfloor \log_\beta^D \rfloor}}{\beta - 1}.$$

Now we know that the total flow through any edge (v_j, u_j) is bounded by $(\frac{2\beta^2-\beta}{\beta-1})\mu_j$, $j = 1, ..., M$. Then we compute the minimum value of $\frac{2\beta^2-\beta}{\beta-1}$. Since $\beta > 1$, the solution is $\beta = 1+\sqrt{2}/2$ and the value of $\frac{2\beta^2-\beta}{\beta-1}$ is $3+2\sqrt{2}$. Therefore, the sum of unsplittable flow through any edge is at most $3+2\sqrt{2}$ times its edge capacity. In each sub-interval, the cost of unsplittable flow is bounded by the cost of the initial flow. Thus the budget C is still respected. Finally, we conclude that there exists a $3+2\sqrt{2}$-approximation algorithm for the minimum congestion of the min-cost unsplittable flow problem and have a $1/(3+2\sqrt{2})$-approximation algorithm for the constrained maximum concurrent multicommodity unsplittable flow problem. □

Corollary 5.12. *There is a $1/(6+4\sqrt{2})$-approximation algorithm for solving the problem MPC2.*

5.3.3 Approximation Algorithm 4

Design of Algorithm 4

Algorithm 4 is based on the the algorithm in [ST93] for solving the scheduling of unrelated machines problem with costs. The first stage of Algorithm 4 is the same as Algorithm 3, where we convert (MPC2') into the constrained maximum concurrent multicommodity exactly-k-splittable flow problem and will lose $1/2$ of approximation factor.

In the second stage, we solve the relaxation (MPC2") of the constrained maximum concurrent multicommodity exactly-k-splittable flow problem and transform its solution to a feasible (fractional) solution of the scheduling of unrelated machines problem with costs.

In the third stage, we round the feasible (fractional) solution of the scheduling of unrelated machines problem with costs to an integer solution while respecting the cost constraint.

Finally, we map the integer solution back to the solution $\delta_{j,k}$ and the capacity λ in the original problem (MPC2).

Transformation of (MPC2') to the scheduling of unrelated machines problem with costs

When (MPC2') is transformed into the constrained maximum concurrent multicommodity exactly-k-splittable flow problem, we have the following relaxation (MPC2") of the constrained multicommodity concurrent unsplittable flow problem as follows,

$$\begin{aligned}
\max \quad & \lambda && (MPC2'') \\
\text{s.t.} \quad & \sum_{j=1}^{M} x_{j,(k,i)} \geq \lambda a_k/c_k, \forall (k,i) && (1) \\
& \sum_{(k,i)} \sum_{j=1}^{M} \hat{r}_{j,(k,i)} x_{j,(k,i)} \leq C, && (2) \\
& \sum_{(k,i)} x_{j,(k,i)} \leq \mu_j, \forall j && (3) \\
& x_{j,(k,i)} \geq 0, \forall j, (k,i) && (4)
\end{aligned}$$

Let the optimal solution of (MPC2") be $x^*_{j,(k,i)}$ and λ^*. We also define $\lambda_{(k,i)}$, where $\lambda_{(k,i)} = \sum_{j=1}^{M} x^*_{j,(k,i)}/(a_k/c_k)$. It is easy to check that $\lambda_{(k,i)} \geq \lambda^*$ for any (k,i). Also, we define $y_{j,(k,i)} = \frac{c_k}{\lambda_{(k,i)} a_k} x^*_{j,(k,i)}$, $p_{j,(k,i)} = \frac{a_k}{c_k \mu_j}$, and $\hat{r}'_{j,(k,i)} = \hat{r}_{j,(k,i)} \cdot \frac{\lambda_{(k,i)} a_k}{c_k}$, for all j, (k,i).

Obviously, we have the following equality

$$\sum_{j=1}^{M} y_{j,(k,i)} = \sum_{j=1}^{M} \frac{c_k}{\lambda_{(k,i)} a_k} x^*_{j,(k,i)} = 1, \forall k, i,$$

and the inequalities,

$$\begin{aligned}
\sum_{(k,i)} p_{j,(k,i)} y_{j,(k,i)} &= \sum_{(k,i)} \frac{a_k}{c_k \mu_j} \cdot \frac{c_k}{\lambda_{(k,i)} a_k} x^*_{j,(k,i)} \\
&\leq \frac{1}{\mu_j} \frac{\sum_{(k,i)} x_{j,(k,i)}}{\lambda^*} \leq 1/\lambda^*, \forall (k,i)
\end{aligned}$$

and,

$$\begin{aligned}\sum_{j=1}^{M}\sum_{(k,i)}\hat{r}'_{j,(k,i)}y_{j,(k,i)} &= \sum_{j=1}^{M}\sum_{(k,i)}\hat{r}_{j,(k,i)}\cdot\frac{\lambda_{(k,i)}a_k}{c_k}\cdot\frac{c_k}{\lambda_{(k,i)}a_k}x^*_{j,(k,i)}\\ &= \sum_{j=1}^{M}\sum_{(k,i)}\hat{r}_{j,(k,i)}x^*_{j,(k,i)}\\ &\leq C, \forall j, (k,i).\end{aligned}$$

We define $T = 1/\lambda^*$. Combining the above, we have the following linear programming problem:

$$\sum_{j=1}^{M} y_{j,(k,i)} = 1, \forall(k,i) \qquad (1) \quad (SC1)$$

$$\sum_{(k,i)} p_{j,(k,i)}y_{j,k} \leq T, \forall j \qquad (2)$$

$$\sum_{j=1}^{M}\sum_{(k,i)}\hat{r}'_{j,(k,i)}y_{j,(k,i)} \leq C \qquad (3)$$

$$y_{j,(k,i)} \geq 0, \forall j, (k,i). \qquad (4)$$

This is exactly the relaxed version of the scheduling unrelated machines problem with costs, where the job nodes in (SC1) correspond with the sub-commodity nodes $t_{(k,i)}$, $\forall(k,i)$ in (MPC2") and the machine nodes in (SC1) correspond with the edges (v_j, u_j), $\forall j$ in (MPC2"). The processing time for job (k,i) scheduled on machine j is defined to be $p_{j,(k,i)}$. We know that scheduling job (k,i) on machine j can be viewed as routing sub-commodity (k,i) on the edge (v_i, u_i) unsplittably. The fraction of job (k,i) scheduled on machine j is $y_{j,(k,i)}$. Given $x^*_{j,(k,i)}$ and λ^* in (MPC2"), we can find a feasible (fractional) solution $y_{j,(k,i)}$ with the makespan $T = 1/\lambda^*$ for the linear programming problem (SC1).

Similar to Algorithm 2, we define a variable $\rho := \max_{j,(k,i)}\{p_{j,(k,i)}/T\}$ and consider

the following linear programming problem (SC2).

$$\sum_{j=1}^{M} y_{j,(k,i)} = 1, \forall (k,i) \qquad (1) \quad (SC2)$$

$$\sum_{(k,i)} p_{j,(k,i)} y_{j,(k,i)} \leq T, \forall j \qquad (2)$$

$$\sum_{j=1}^{M} \sum_{(k,i)} \hat{r}'_{j,(k,i)} y_{j,(k,i)} \leq C \qquad (3)$$

$$y_{j,(k,i)} = 0 \quad \text{if} \quad p_{j,(k,i)} > \rho \cdot T, \forall j, (k,i) \qquad (4)$$

$$y_{j,(k,i)} \geq 0, \forall j, (k,i). \qquad (5)$$

Since

$$p_{j,(k,i)} \leq \max_{j,(k,i)} \{p_{j,(k,i)}\} = \rho \cdot T$$

We know that the constraint (4) in (SC2) will not set any $y_{j,(k,i)}$ to zero. Thus the solution $y_{j,(k,i)}$ and $T = 1/\lambda^*$ in (SC1) is still feasible in (SC2).

Description of Algorithm 4

D. B. Shmoys and E. Tardos [ST93] proposed an algorithm for rounding any fractional schedule $y_{j,(k,i)}$ to an integer schedule $\hat{y}_{j,(k,i)}$ which has makespan at most $T + \rho \cdot T$. When we find the integer solution $\hat{y}_{j,(k,i)}$for (SC2), we know that the integer solution $\hat{x}_{j,(k,i)}$ can be easily computed by setting $\hat{x}_{j,(k,i)} = \hat{y}_{j,(k,i)} \lambda_{(k,i)} a_k / c_k$. Thus the approximate solution $\delta_{j,k}$ and λ of (MPC2) will be derived.

The idea of the rounding algorithm is that we split each machine j into several sub-machines and guarantee the number of sub-machines are greater than or equal to the number of jobs (k,i) we have. Then we try to find a fractional matching in this bipartite graph with job (k,i) and the sub-machines. According to the fact that the min-cost integer matching can be computed in polynomial time, we finally get the integer matching and transform it to the solution of the scheduling of unrelated machines problem with costs.

It is easy to see that we have $K' := \sum_{k=1}^{K} c_k$ jobs and M machines in (SC2). Let $y_{j,k}$ $(j = 1, ..., M, k = 1, ..., K')$ be the solution of (SC2). We will follow the

steps in [ST93] to construct a bipartite graph $B(y) = (V, W, E)$ and a value $y'(v, w)$ (originally, $y'(v, w) = 0$) for each edge $(v, w) \in E$. This bipartite graph consists of the following two parts:

job nodes: $W = \{w_k : k = 1, ..., K'\}$.

sub-machine nodes: $V = \{v_{j,s} : j = 1, ..., M, s = 1, ..., k_j\}$, where $k_j := \lceil \sum_{k=1}^{K'} y_{j,k} \rceil$ and sub-machines $\{v_{j,s} : s = 1, ..., k_j\}$ correspond to machine j.

The cost of each edge $(v_{j,s}, w_k)$ equals the cost of machine j working on job k in (SC2), which is $\hat{r}'_{j,k}$.

We construct the graph $B(y)$ and the vector y' in the following way.

Step 1: We sort the jobs in the order of nonincreasing processing time $p_{j,k}$ such that $p_{j,1} \geq p_{j,2} \geq ... \geq p_{j,K'}$, $j = 1, ..., M$.

Step 2: Choose a machine j and compute the value $\sum_{k=1}^{K'} y_{j,k}$. If $\sum_{k=1}^{K'} y_{j,k} \leq 1$, we know that there is only one sub-machine $v_{j,1}$ corresponding to machine j. Then, for each $y_{j,k} > 0$, we set $y'(v_{j,1}, w_k) := y_{j,k}$.

Step 3: If $\sum_{k=1}^{K'} y_{j,k} \geq 1$, we know that there exists more than one sub-machine $v_{j,s}$. We find the minimum index k_1 such that $\sum_{k=1}^{k_1} y_{j,k} \geq 1$, and set $y'(v_{j,1}, w_k) := y_{j,k}$, $k = 1. ..., k_1 - 1$. Then, we set $y'(v_{j,1}, w_{k_1}) := 1 - \sum_{k=1}^{k_1-1} y'(v_{j,1}, w_k)$. This guarantees that the sum of y' for edges incident to $v_{j,1}$ is exactly 1. If $\sum_{k=1}^{k_1} y_{j,k} > 1$, we know that a fraction of the value y_{j,k_1} remains. Then we assign the remaining fraction on edge $(v_{j,2}, w_{k_1})$ and set $y'(v_{j,2}, w_{k_1}) := (\sum_{k=1}^{k_1} y_{j,k}) - 1$.

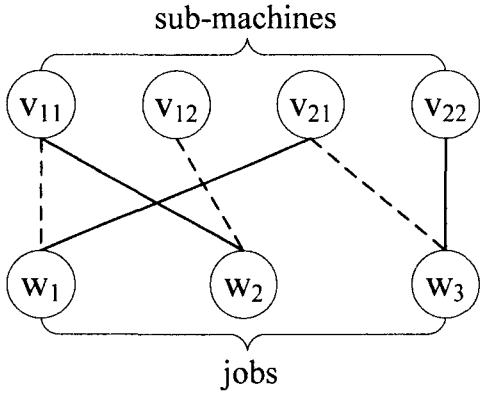
Step 4: For each $s = 2, ..., k_j - 1$, we find the minimum index k_s such that $\sum_{k=1}^{k_s} \geq s$. We set $y'(v_{j,s}, w_k) := y_{j,k}$, $k = k_{s-1} + 1, ..., k_s - 1$ for each $y_{j,k} > 0$, and set $y'(v_{j,s}, w_{k_s}) := 1 - \sum_{k=k_{s-1}+1}^{k_s-1} y'(v_{j,s}, w_k)$. Similarly, if $\sum_{k=1}^{k_s} y_{j,k} > s$, we also set $y'(v_{j,s+1}, w_{k_s}) := (\sum_{k=1}^{k_s} y_{j,k}) - s$.

We give a simple example to show this construction. Suppose we have 2 machines and 3 jobs. The matrix $y_{j,k}$ and $p_{j,k}$ are given as follows:

$$y_{j,k} = \begin{pmatrix} 1/4 & 1 & 0 \\ 3/4 & 0 & 1 \end{pmatrix}$$

$$p_{j,k} = \begin{pmatrix} 2 & 1 & 1 \\ 2 & 1 & 1 \end{pmatrix}$$

Then $B(y)$ is constructed as follows: The above steps can also be written as the



For solid edges, $y'(v_{j,s}, w_k) = 3/4$.
For dashed edges, $y'(v_{j,s}, w_k) = 1/4$.

Figure 5.2: An example of constructing $B(y)$

following procedure.

Procedure *Frac_Matching*
Input: the bipartite graph $B(y)$ and $y_{j,k}$.
Output: the fractional matching $y'(v_{j,s}, w_k)$.

Step 1: For each machine j, sort the jobs in the order of nonincreasing processing time $p_{j,k}$ such that we have $p_{j,1} \geq p_{j,2} \geq ... \geq p_{j,K'}$.

Step 2: For each machine j

(a): If $\sum_{k=1}^{K} y_{j,k} \leq 1$, we know that there is only one sub-machine $v_{j,1} \in V$ corresponding to machine j. Then, for each $y_{j,k} > 0$, include $(v_{j,1}, w_k) \in E$ and set $y'(v_{j,1}, w_k) := y_{j,k}$.

(b): If $\sum_{k=1}^{K} y_{j,k} > 1$, set $index(0) := 1$.
For $i = 1$ to $k_j - 1$

(i): Let $index(i)$ be the minimum index such that $\sum_{k=1}^{index(i)} y_{j,k} \geq i$.

(ii): Set $y'(v_{j,q}, w_k) := y_{j,k}$, $q = 1, ..., index(i) - 1$.

(iii): Set $y'(v_{j,index(i)}, w_k) := 1 - \sum_{q=index(i-1)}^{index(i)-1} y'(v_{j,q}, w_k)$.

(iv): If $\sum_{k=index(i-1)}^{index(i)} y_{j,k} > 1$, set

$y'(v_{j,index(i)+1}, w_{index(i)}) := (\sum_{k=index(i-1)}^{index(i)} y_{j,k}) - 1$.

D. B. Shmoys and E. Tardos present the definition of fractional matching in [ST93] as follows: A non-negative vector z on the edges of a bipartite graph is a fractional matching if, for each node u, the sum of the components of z corresponding to the edges incident to u is at most 1. The fractional matching exactly matches a node u if the corresponding sum is exactly 1. A fractional matching z is a matching if each component of z is 0 or 1.

Lemma 2.2 in [ST93] summarizes some simple properties of the bipartite graph $B(y)$ and the vector $y'(v_{j,s}, w_k)$, which is as follows:

Lemma 5.13. *The vector $y'(v_{j,s}, w_k)$ is a fractional matching in $B(y)$ of cost at most C. All job nodes w_k, $k = 1, ..., K'$ and sub-machine nodes $v_{j,s}$, $j = 1, ..., M, s = 1, ..., k_j - 1$ are exactly matched. Moreover, let $p_{j,s}^{max}$ denote the maximum of the processing times $p_{j,k}$ corresponding to edges $(v_{j,s}, w_k) \in E$ and $p_{j,s}^{min}$ denote the minimum of the processing times $p_{j,k}$ corresponding to edges $(v_{j,s}, w_k) \in E$. Finally we have $p_{j,s}^{min} \geq p_{j,s+1}^{max}$ for each $j = 1, ..., m$, $s = 1, ..., k_j - 1$.*

According to the steps constructing the fractional matching y' in $B(y)$, we have $y_{j,k} = \sum_{(v_{j,s}, w_k) \in E} y'(y_{j,s}, w_k)$ and the cost on edge $(v_{j,k}, w_k) \in E$ is equal to $\hat{r}'_{j,k}$ in (SC2). Therefore, the total cost of the fractional matching still respects the budget C in (SC2). By Theorem 7.3.3 in [Lov86], we know that there exists an integral solution of the optimal value for the minimum cost bipartite matching problem. Then we find a minimum cost (integer) matching M which matches all job nodes in $B(y)$. It is clear that the cost of this integer matching M is less than or equal to the cost of the fractional matching y' and also respects budget C. Finally, for each edge $(v_{j,s}, w_k)$ in M, we schedule job k on machine j in (SC2), $j = 1, ..., M$, $k = 1, ..., K'$.

The minimum cost integer matching problem can be solved by the Hungarian method for solving the assignment problem. (For more details about the Hungarian method, refer to [Mur95] and [Lov86].) The assignment problem is a special case of

the min-cost integer matching problem where we have the same number of machines and jobs. By our construction of the bipartite graph $B(y)$, we know that the number of sub-machine nodes $\sum_{j=1}^{M} k_j$ is greater than or equal to the number of job nodes K'. Thus we construct a new bipartite graph $B'(y)$ by adding $(\sum_{j=1}^{M} k_j) - K'$ dummy job nodes w'_k, $k = 1, ..., \sum_{j=1}^{M} k_j - K'$. The cost of each edge $(v_{j,s}, w'_k)$ is defined to be zero. By Lemma 5.13 we know that all job nodes w_k and sub-machine nodes $v_{j,s}$, $j = 1, ..., M$, $s = 1, ..., k_j - 1$ are exactly matched. Thus, the number of sub-machine nodes $\sum_{j=1}^{M} k_j$ is greater than or equal to the number of job nodes K'. The sum of remaining fractional matching incident to sub-machine v_{j,k_j}, $j = 1, ..., M$ is $(\sum_{j=1}^{M} k_j) - K'$. Then we fractionally match sub-machines v_{j,k_j}, $j = 1, ..., M$ with the dummy job nodes w'_k. By this construction, we can transform the minimum cost integer matching problem to an equivalent assignment problem with the initial fractional matching. This fractional matching we construct for $B'(y)$ is also called the fractional assignment and will be the initial input of the Hungarian method. Obviously, the optimal integer assignment in $B'(y)$ will give us the optimal integer matching in $B(y)$.

We use the Hungarian method to solve the reduced assignment problem and generate an integer assignment in $B'(y)$ with minimum cost. For each assignment corresponding to edge $(v_{j,s}, w_k)$, we schedule job k on machine j in (SC2), $j = 1, ..., M$, $k = 1, ..., K'$. Finally, the solution of (MPC2) can be generated by the integer schedule we find.

Algorithm 4 applies the same first two stages as Algorithm 3. The following is the detailed description.

Stage 1: Initialization. Construct a network with one source node $\{S\}$ for sending K commodities, $2M$ nodes $\{v_1, ..., v_M, u_1, ...u_M\}$ for transferring commodities, and K terminal nodes $\{t_1, ..., t_K\}$ with demand $a_k, k = 1, ..., K$ for receiving commodities. Set the capacity on edge (v_j, u_j) to be μ_j for $j = 1, ..., M$ and the cost on edge (u_j, t_k) to be $\hat{r}_{j,k} := r_{j,k}/\mu_j$ for $j = 1, ..., M, k = 1, ..., K$.

Stage 2: Transformation of initial problem to a constrained maximum concurrent uniform exactly-k-splittable flow problem. We replicate each terminal node t_k into c_k identical sub-terminals $t_{(k,i)}$, $i = 1, ..., c_k$. Then each commodity k with demand a_k is split into c_k commodities (k, i), $i = 1, ..., c_k$, each with the same demand $a_{(k,i)} := a_k/c_k$ and with the same source. Further, for each edge connecting

with terminal node t_k, we connect it with all of the sub-terminals $t_{(k,i)}$ and set the cost on edge $(u_j, t_{(k,i)})$ to be $\hat{r}_{j,k}$, $\forall j, (k,i)$. The new commodities (k,i) will be routed unsplittably.

Stage 3: Solving the fractional constrained maximum concurrent uniform exactly-k-splittable flow problem and transformation of its solution to a solution of the scheduling of unrelated machines problem with costs. We consider the relaxed version of the constrained maximum concurrent uniform exactly-k-splittable flow problem (MPC2") and get the solution $x^*_{j,(k,i)}$ and λ^* by the ellipsoid method. Then we construct the corresponding scheduling of unrelated machines problem, with job (k,i), $\forall(k,i)$ and machine j, $j = 1, ..., M$. The processing time for a job (k,i) scheduled on machine j is defined to be $p_{j,(k,i)}$, where $p_{j,(k,i)} := \frac{a_k}{c_k \mu_j}$. We assign the cost $\hat{r}'_{j,(k,i)} := \hat{r}_{j,(k,i)} \cdot \frac{\lambda_{(k,i)} a_k}{c_k}$ on edge $(j,(k,i))$, $\forall j, (k,i)$. Let a feasible solution of this scheduling problem be $y_{j,(k,i)}$ and the makespan T. Further we define $y_{j,(k,i)} := \frac{x^*_{j,(k,i)} c_k}{\lambda_{(k,i)} a_k}$ and $T := 1/\lambda^*$.

Stage 4: Computing a fractional matching $y'(v_{j,s}, w_k)$. Let $K' := \sum_{k=1}^{K} c_k$. We define $y_{j,k}$ to be the fraction of job k on machine j, $r_{j,k}$ to be the cost of job k on machine j, and $p_{j,k}$ to be the processing time of job k on machine j, where $k = 1, ..., K'$. Then we construct a bipartite graph $B(y) = (V, W, E)$ and a value $y'(v,w)$ and a cost $c(v,w)$ for each edge $(v,w) \in E$. One side of the bipartite graph consists of job nodes $W = \{w_k, k = 1, ..., K'\}$ and the other side consists of machine nodes $V = \{v_{j,s}\}, j = 1, ..., M, s = 1, ..., k_j$ where $k_j := \lceil \sum_{k=1}^{K} y_{j,k} \rceil$. We also set the cost of edge $(v_{j,s}, w_k) \in E$ to be $\hat{r}'_{j,k}$ in (SC2). Then we call procedure *Frac_Matching* to generate the fractional matching $y'(v_{j,s}, w_k)$ in $B(y)$.

Stage 5: Finding a min-cost integer matching which exactly matches all job nodes in $B(y)$. We construct a new bipartite graph $B'(y)$ by adding $k = 1, ..., \sum_{j=1}^{M} k_j - K'$ dummy job nodes w'_k, $k = 1, ..., \sum_{j=1}^{M} k_j - K'$. We set the cost on all edges $(v_{j,k}, w'_k)$ to be zero and randomly match those dummy job nodes with unmatched sub-machine nodes in $B'(y)$. Then we use the Hungarian method to find the min-cost integer matching $y(v_{j,s}, w_k)$ in $B'(y)$.

Stage 6: Calculation of the integer solution $\hat{x}_{j,(k,i)}$ of (MPC2'). We check the integer matching $y(v_{j,s}, w_k)$. If $y(v_{j,s}, w_k) = 1$, we find the node (k,i) corresponding with w_k and set $\hat{y}_{j,(k,i)} := 1$ otherwise $\hat{y}_{j,(k,i)} := 0$. Then let $\hat{x}_{j,(k,i)} := \hat{y}_{j,(k,i)} \frac{\lambda_{(k,i)} a_k}{c_k}$. $\hat{x}_{j,(k,i)}$ is the solution for the reduced constrained maximum concurrent multicommod-

ity exactly-k-splittable flow problem. Similar to Stage 5 of Algorithm 3, we scale down flow on every edge by a factor $\alpha := \max_j \frac{\sum_{(k,i)} \hat{x}_{j,(k,i)}}{\mu_j}$ such that the edge capacities on (v_j, u_j), $\forall j$ are respected.

Stage 7: Calculation of the original solution $\delta_{j,k}$ and the capacity λ in (MPC2). The original $\delta_{j,k}$ in (MPC2) can be computed by setting $\delta_{j,k} := \frac{\sum_{(k,i)} \hat{x}_{j,(k,i)}}{\mu_j}$, $j = 1, ..., M, k = 1, ..., K$. The capacity of (MPC2) is that $\lambda = \min_k \frac{\sum_{j=1}^{M} \mu_j \delta_{j,k}}{a_k}$.

Analysis of Algorithm 4

By applying Theorem 2.1 from [ST93], we first prove the following theorem.

Theorem 5.14. *Stages 4 and 5 in Algorithm 4 round any feasible solution $y_{j,(k,i)}$ into an integer solution $\hat{y}_{j,(k,i)}$ with makespan at most $(1+\rho)T$ and total cost at most C.*

Proof. By Lemma 5.13, we know that the cost of the fractional matching $y'(v_{j,s}, w_k)$ in the bipartite graph $B(y)$ is at most C. In Stage 5 of Algorithm 3, we can find a integer matching of the minimum cost in $B(y)$ by the Hungarian method. Clearly, the minimum cost in $B(y)$ is less than or equal to C. Thus the budget C is respected. Applying Theorem 2.1 in [ST93] directly, we know that the makespan is at most $(1+\rho)T$. □

Then we can prove the following theorem:

Theorem 5.15. *Algorithm 4 is a $\frac{1}{2(1+\rho)}$-approximation algorithm for solving (MPC2').*

Proof. Suppose the optimal solution of (MPC2') is λ^*. By Theorem 5.7, solving the constrained maximum concurrent multicommodity uniform exactly-k-splittable flow problem guarantees at least $1/2$ of the optimal solution λ^*. According to Theorem 5.14, we know that the makespan is as most $(1+\rho)T$, thus we can achieve at least $\frac{1}{1+\rho}$ of the optimal capacity of the constrained maximum concurrent multicommodity uniform exactly-k-splittable flow problem (MPC2") without violating the budget C. Therefore, at least $\frac{1}{2(1+\rho)}$ of the optimal solution λ^* can be achieved and the budget C is still respected, which proves the theorem. □

5.4 The case $\mu = \alpha \cdot \beta^T$

Given an $M \times 1$ vector α and a $K \times 1$ vector β, when the $M \times K$ matrix $\mu = \alpha \cdot \beta^T$, we show that this case will fall into the case of Section 5.2.

Since $\mu = \alpha \cdot \beta^T$, we know that in matrix μ, the (j,k) entry $\mu_{j,k} = \alpha_j \beta_k$. We define $x_{j,k} := \alpha_j \delta_{j,k}$ for all j, k and $b_k := a_k/\beta_k$ for all k. Moreover, we define the original cost $\hat{r}_{j,k} := r_{j,k}/\alpha_j$ for all j. Now the problem (MPC) can be written as,

$$\begin{aligned}
&\max \quad \lambda && (MPC') \\
&\text{s.t.} \quad \sum_{j=1}^{M} x_{j,k} \geq \lambda b_k, k = 1, ..., K && (1) \\
&\qquad \sum_{k=1}^{K} x_{j,k} \leq \alpha_j, j = 1, ..., M && (2) \\
&\qquad \sum_{k=1}^{K} \sum_{j=1}^{M} \hat{r}_{j,k} x_{j,k} \leq C, && (3) \\
&\qquad \sum_{j=1}^{M} \chi\{\delta_{j,k} > 0\} \leq c_k, k = 1, ..., K && (4) \\
&\qquad x_{j,k} \geq 0, k = 1, ..., K, j = 1, ..., M. && (5)
\end{aligned}$$

Obviously, (MPC') is exactly the same form as (MPC2'). Therefore, we can apply Algorithm 3 or Algorithm 4 to calculate an approximate solution $x_{j,k}$ and generate the final solution of TDCCP with costs by setting $\delta_{j,k} := x_{j,k}/\alpha_j$ for all j, k.

5.5 Solving the general case

When the servers or classes are not identical, we will have the general case of (MPC). In this section, an approximation algorithm will be given for solving the general (MPC).

The idea of this algorithm is similar to the algorithm for solving the general (MP). First, we transform (approximately) the general (MPC) to an instance of (MPC2'). Then, we solve the reduced case by Algorithm 3 or Algorithm 4.

We assume $\mu_{j,k} > 0$ for all j, k. Let μ_j^{max} be the maximum service rate of server j over all classes, i.e. $\mu_j^{max} = \max_k\{\mu_{j,k}\}$. Let μ_j^{min} be the minimum service rate of server j over all classes, i.e. $\mu_j^{min} = \min_k\{\mu_{j,k}\}$.

We try to solve the following problem first.

$$\begin{aligned}
\max \quad & \lambda && (MPC3)\\
\text{s.t.} \quad & \sum_{j=1}^{M} \mu_{j,k}\delta_{j,k} \geq \lambda a_k, k = 1, ..., K && (1)\\
& \sum_{k=1}^{K} \mu_{j,k}\delta_{j,k} \leq \mu_j^{max}, j = 1, ..., M && (2)\\
& \sum_{k=1}^{K}\sum_{j=1}^{M} r_{j,k}\delta_{j,k} \leq C, && (3)\\
& \sum_{j=1}^{M} \chi\{\delta_{j,k} > 0\} \leq c_k, k = 1, ..., K && (4)\\
& \delta_{j,k} \geq 0, k = 1, ..., K, j = 1, ..., M. && (5)
\end{aligned}$$

Obviously, (MPC3) is the same formulation as (MPC2') in Section 5.2. Let the optimal solution of general (MPC) be λ^*, δ^* and the optimal solution of (MPC3) be $\hat{\lambda}^*$, $\hat{\delta}^*$. We have the following lemma similar to Lemma 4.10.

Lemma 5.16. *For the optimal solution of (MPC) and (MPC3), we have* $\hat{\lambda}^* \geq \lambda^*$.

Suppose the (approximate) solution of (MPC3) is $\tilde{x}_{j,k}$ and $\tilde{\lambda}$. Let $\omega_j = \mu_j^{max}/\mu_j^{min}$ and $\omega_{max} = \max_j\{\omega_j\}$. By defining

$$\delta_{j,k} := \frac{\tilde{x}_{j,k}}{(\omega_j \mu_{j,k})}, \forall j, k. \tag{5.10}$$

We can prove the following two theorems by choosing the different algorithms for (MPC3). The proof is similar to the proof of Theorem 4.11.

Theorem 5.17. *Solution (5.10) is a feasible solution of (MPC), and achieves a* λ *of value at least* $\lambda^*/12\omega_{max}$.

Theorem 5.18. *Solution (5.10) is a feasible solution of (MPC), and achieves a* λ *of value at least* $\lambda^*/2(1+\rho)\omega_{max}$.

5.6 NP-Completeness

We consider the special case (MPC2'). By setting the cost $r_{j,k} := 0$, $\forall j, k$, the constraint (5) in (MPC2') will be trivial for any $C \geq 0$, then we know (MPC2') includes (MP2') as a special case. Therefore, we have the following theorem and corollary.

Theorem 5.19. *(MPC2') is NP-Complete.*

Corollary 5.20. *The general TDCCP with costs is NP-Complete.*

Chapter 6

Experiments

In our experiments, we design some examples for testing each algorithm studied in Chapter 4 and Chapter 5. When the service rates are independent of classes (all classes are identical), the testing results tell us that in most of the time, Algorithm 1, 2, 3 and 4 can produce very good approximate solutions for maximizing the throughput in queueing networks. However, in the general case, the approximation algorithms based on Algorithm 1, 2, 3 and 4 may give poor solutions. All tests are running in MATLAB 7.0. We will compare and analyze the performance among different algorithms.

6.1 Experiments of TDCCP without costs

6.1.1 TDCCP without costs when all classes are identical

Design of Examples

In our examples of TDCCP without costs and all classes identical, the input data are the service rates $\mu_{j,k}$ (μ_j), the numbers a_k and the flexibility bounds c_k. We define two integer M and K, where M is the number of servers and K is the number of classes. The values of M and K are integers in $(0, 20]$. Since all classes are identical and $\mu_{j,k} = \mu_j$ in this case, we let $\mu_{j,k}$ be a $M \times K$ matrix with the same value in each row ($\mu_{j,1} = \mu_{j,2} =, ..., = \mu_{j,K}$, $j = 1, ..., M$). In general, we can choose arbitrary values for each entry in $\mu_{j,k}$ and a_k. When the values of $\mu_{j,k}$ and a_k are very large,

we can always scale those values to a small range. Therefore, we set $\mu_{j,k}$ and a_k to be real numbers in $(0, 100]$. Each entry in c_k must be less than or equal to the number M $(0 < c_k \leq M)$.

Since it is very difficult to get the optimal solutions of TDCCP (without costs), we solve the relaxations of TDCCP (without costs) by abandoning the integer constraints. Originally, Algorithms 1 and 2 apply the ellipsoid method to solve those relaxations. However, in our experiments, the simplex method will be used. This is because the simplex method is more efficient and competitive than the ellipsoid method in practice, although it may have exponential running time in some artificial bad cases (e.g. see [Tod02]). MATLAB 7.0 provides us with a useful linear programming function 'linprog', which applies the revised simplex method (e.g. see [Sim72] and [Sch86]). Therefore, we will use the function 'linprog' to generate the relaxation results of our examples. We know that the optimal value of the relaxation is an upper bound for the optimal value of the original problem. Thus we will use the optimal values of the relaxed problems to evaluate the results of Algorithm 1 and Algorithm 2. Obviously, any solutions by our approximation algorithms cannot exceed the optimal of the relaxations. If Algorithm 1 or Algorithm 2 generates a throughput λ which is equal to the throughput of the relaxation problem, we can conclude that λ must also be the optimal for the original problem.

We partition the testing examples into two groups, depending on the following conditions:

Condition 1 : There exists a feasible fractional server assignment policy $\{\delta^*_{j,k}\}$ satisfying constraint (1), (2) and (4) in (MP2), so that the throughput λ^* is less than 1.

Condition 2 : There exists a feasible fractional server assignment policy $\{\delta^*_{j,k}\}$ satisfying constraint (1), (2) and (4) in (MP2), so that the throughput λ^* is greater than or equal to 1.

Note that Condition 1 and Condition 2 can also be defined in a similar way for TDCCP with costs.

The first test includes 20 examples that satisfy Condition 1 $(0 < \lambda^* < 1)$, and the second test includes 20 examples that satisfy Condition 2 $(\lambda^* \geq 1)$. The numbers M and K are not fixed. Thus, we have different $M \times K$ matrices $\mu_{j,k}$ in different

examples, as well as different vectors a_k and c_k. The values of each entry in $\mu_{j,k}$ and a_k are real numbers randomly selected in $(0, 100]$. Different entries of $\mu_{j,k}$ and a_k may have different values. The vector c_k includes K integers in $[1, M)$. (Please refer to the Appendix B for more details of testing examples.)

Finally, the examples will be tested by Algorithm 1, Algorithm 2 and 'linprog'. The testing environment is MATLAB 7.0.

Testing examples when Condition 1 is satisfied

We design 20 examples (see Appendix B.1) with different $\mu_{j,k}$, a_k and c_k. The output data are the server policy $\delta_{j,k}$ and the thoughput λ. First we run the relaxations of 20 examples by 'linprog' and let λ^* and $\delta^*_{j,k}$ be the corresponding optimal values. We know that $0 < \lambda^* < 1$ for all the examples. Let λ^1, $\delta^1_{j,k}$ be the solution of Algorithm 1, and similarly, let λ^2 and $\delta^2_{j,k}$ be the solution of Algorithm 2. We also define σ^1 to be the relative difference from λ^1 versus λ^* and σ^2 to be the relative difference from λ^2 versus λ^*. The detailed computational results are shown in Table 6.1. Table 6.2 and Figure 6.1 shows the relative differences σ^1 and σ^2. Note that for simplicity, we only show the throughput λ generated by the algorithms.

	λ^1	λ^2	λ^*
Example 1	0.1667	0.2500	0.2500
Example 2	0.3000	0.3000	0.3750
Example 3	0.0667	0.0667	0.0833
Example 4	0.2857	0.2500	0.3000
Example 5	0.5333	0.5714	0.6111
Example 6	0.2000	0.2000	0.2000
Example 7	0.1515	0.1515	0.2078
Example 8	0.5000	0.5000	0.5625
Example 9	0.2000	0.2000	0.7214
Example 10	0.1500	0.1500	0.2000
Example 11	0.3333	0.3333	0.4500
Example 12	0.1333	0.1333	0.4167
Example 13	0.1111	0.1111	0.1667
Example 14	0.2143	0.2143	0.2857
Example 15	0.3333	0.6667	0.8333
Example 16	0.4082	0.3571	0.5215
Example 17	0.0902	0.0789	0.1203
Example 18	0.2222	0.2222	0.2500
Example 19	0.5000	0.5000	0.8929
Example 20	0.3333	0.6667	0.6667

Table 6.1: Testing results of examples MP2 with $0 < \lambda^* < 1$

	σ^1	σ^2
Example 1	0.6667	1.0000
Example 2	0.8000	0.8000
Example 3	0.8000	0.8000
Example 4	0.9524	0.8333
Example 5	0.8727	0.9351
Example 6	1.0000	1.0000
Example 7	0.7292	0.7292
Example 8	0.8889	0.8889
Example 9	0.2772	0.2772
Example 10	0.7500	0.7500
Example 11	0.7407	0.7407
Example 12	0.3200	0.3200
Example 13	0.6667	0.6667
Example 14	0.7500	0.7500
Example 15	0.4000	0.8000
Example 16	0.7826	0.6848
Example 17	0.7500	0.6563
Example 18	0.8889	0.8889
Example 19	0.5600	0.5600
Example 20	0.5000	1.0000

Table 6.2: Relative differences from λ^1 and λ^2 versus λ^* for examples MP2 with $0 < \lambda^* < 1$

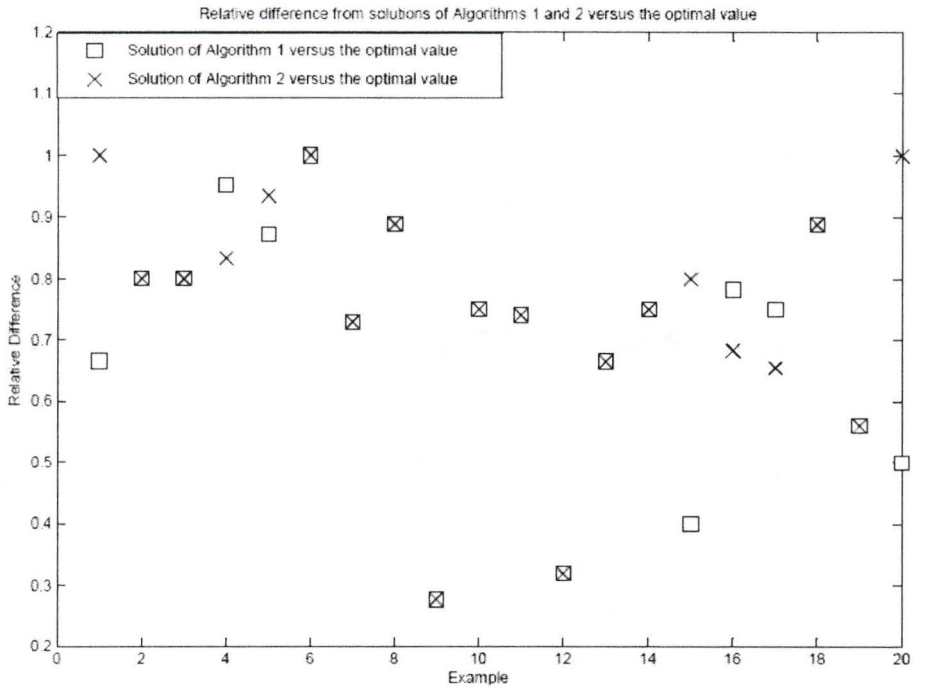


Figure 6.1: Relative differences from λ^1 and λ^2 versus λ^* for examples MP2 with $0 < \lambda^* < 1$

When the service rates are independent of classes and Condition 1 is satisfied, we find that both Algorithm 1 and Algorithm 2 can produce good solutions much better than the theoretical lower bounds. In example 6, their results are the same as the optimal of relaxation of TDCCP. Since the optimal of relaxation is the upper bound of the optimal of (MP2), obviously, the solutions of Algorithm 1 and 2 also achieve the optimal. In most examples, the solutions of Algorithm 1 and Algorithm 2 are very close. But in example 20, Algorithm 2 performs much better than Algorithm 1 and it even generates the optimal solution. By carefully analyzing the server assignment policies produced by Algorithm 1 and 2, we find the reason is that the proportion of time $\delta^1_{j,k}$ of a slow server j working at a class k is larger than $\delta^2_{j,k}$. In other words, a slow server in the system has been assigned too much work.

Testing examples when Condition 2 is satisfied

We design another 20 examples (see Appendix B.2) when Condition 2 is satisfied. The output data are $\delta_{j,k}$ and λ. The optimal solutions of relaxations λ^* and $\delta^*_{j,k}$ are also generated by 'linprog'. λ^1 and $\delta^1_{j,k}$ are the solutions of Algorithm 1. λ^2 and $\delta^2_{j,k}$ are the solutions of Algorithm 2. We define σ^1 to be the relative difference from λ^1 versus λ^* and σ^2 to be the relative difference from λ^2 versus λ^*. We give the detailed computational results in Table 6.3. Table 6.4 and Figure 6.2 shows the relative differences σ^1 and σ^2.

	λ^1	λ^2	λ^*
Example 1	1.0000	0.6250	1.0667
Example 2	2.0000	3.0000	3.3333
Example 3	2.0000	2.2222	2.3333
Example 4	1.3333	2.0000	2.4000
Example 5	2.0000	2.0000	2.2222
Example 6	10.0000	10.0000	10.0000
Example 7	1.5152	1.5152	2.0779
Example 8	4.0000	4.0000	4.5000
Example 9	1.0000	1.0000	1.4286
Example 10	3.0000	3.0000	4.0000
Example 11	3.3333	3.3333	4.5000
Example 12	1.3333	1.3333	4.1667
Example 13	3.3333	3.0000	3.4706
Example 14	3.0000	3.0000	4.0000
Example 15	1.0000	4.0000	4.5000
Example 16	4.4444	6.6667	8.3951
Example 17	1.5238	1.3333	2.5079
Example 18	5.0000	5.0000	5.6250
Example 19	3.0000	3.0000	6.9286
Example 20	1.4286	7.6923	8.6364

Table 6.3: Testing results of examples MP2 with $\lambda^* \geq 1$

	σ^1	σ^2
Example 1	0.9375	0.5859
Example 2	0.6000	0.9000
Example 3	0.8571	0.9524
Example 4	0.5556	0.8333
Example 5	0.9000	0.9000
Example 6	1.0000	1.0000
Example 7	0.7292	0.7292
Example 8	0.8889	0.8889
Example 9	0.7000	0.7000
Example 10	0.7500	0.7500
Example 11	0.7407	0.7407
Example 12	0.3200	0.3200
Example 13	0.9605	0.8644
Example 14	0.7500	0.7500
Example 15	0.2222	0.8889
Example 16	0.5294	0.7941
Example 17	0.6076	0.5316
Example 18	0.8889	0.8889
Example 19	0.4330	0.4330
Example 20	0.1654	0.8907

Table 6.4: Relative differences from λ^1 and λ^2 versus λ^* for examples MP2 with $\lambda^* \geq 1$

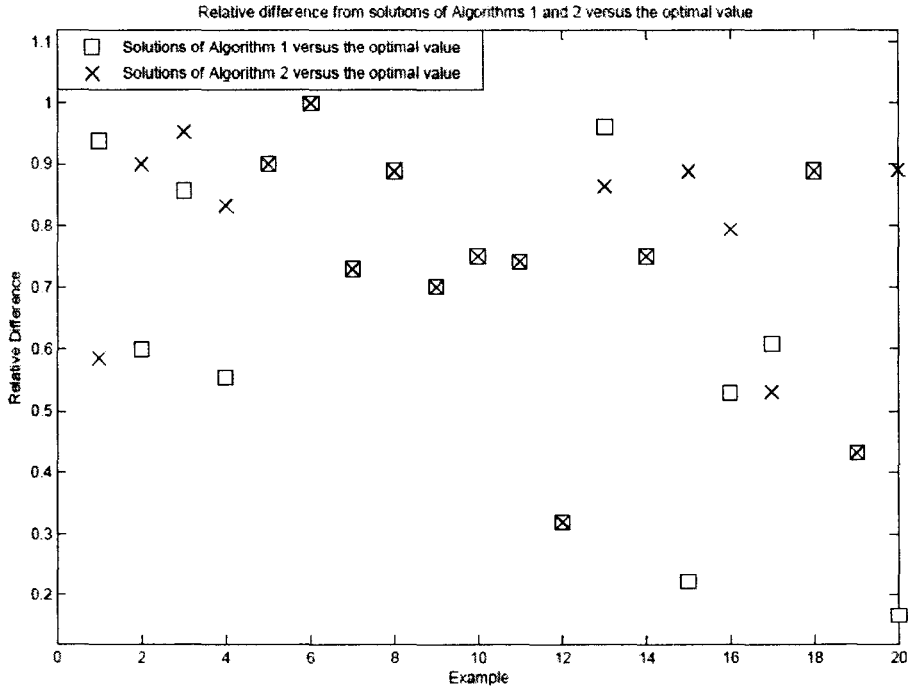


Figure 6.2: Relative differences from λ^1 and λ^2 versus λ^* for examples MP2 with $\lambda^* \geq 1$

It is easy to see that both Algorithm 1 and Algorithm 2 produce good solutions in most examples. We also observe that in several examples (e.g. examples 5, 6, 8, 9, 10, 11, 12, 14, 18 and 19), Algorithm 1 and Algorithm 2 produce the same throughput $\lambda^1 = \lambda^2$. However, by analyzing the server assignment policies generated by Algorithm 1 and Algorithm 2 in those examples, we find that $\delta^1_{j,k}$ and $\delta^2_{j,k}$ are not the same. In examples 2, 3, 15, 16 and 20, we see that the throughput estimate λ^2 produced by Algorithm 2 is better than the throughput estimate λ^1 produced by Algorithm 1. We also notice that in examples 15 and 20, Algorithm 2 performs much better than Algorithm 1.

6.1.2 TDCCP without costs in the general case

Design of Examples

In the general case of TDCCP without costs, the input data are also $\mu_{j,k}$, a_k and c_k. The difference is that the $M \times K$ matrix $\mu_{j,k}$ is not required to have the same entry value in each row vector. Similar to the tests in Section 6.1.1, we define the values of M and K to be integers in $(0, 20]$ and c_k to be integers in $(0, M]$. Each entry in $\mu_{j,k}$ and a_k is defined to be a real number in $(0, 100]$. All the numbers of $\mu_{j,k}$, a_k and c_k are defined randomly.

There will be two different tests. The first test includes 20 different examples that satisfy Condition 1 ($0 < \lambda^* < 1$), and the second test includes 20 different examples that satisfy Condition 2 ($\lambda^* \geq 1$).

The algorithm for solving the general (MP2) will apply Algorithm 1 or Algorithm 2 as a procedure. All of the examples will be tested by the algorithm of applying Algorithm 1, the algorithm of applying Algorithm 2 and 'linprog'. The testing environment is also MATLAB 7.0.

Testing examples when Condition 1 is satisfied

We design 20 examples (see Appendix B.3) with different $\mu_{j,k}$, a_k and c_k. The output data are $\delta_{j,k}$ and λ. Let the optimal solutions of the relaxations be λ^* and $\delta^*_{j,k}$, the solutions by applying Algorithm 1 be $\lambda^{1'}$ and $\delta^{1'}_{j,k}$, the solutions by applying Algorithm 2 be $\lambda^{2'}$ and $\delta^{2'}_{j,k}$. We define σ^1 to be the relative difference from $\lambda^{1'}$ versus λ^* and σ^2 to be the relative difference from $\lambda^{2'}$ versus λ^*. We will first solve the relaxation of each example by the function 'linprog' and get the optimal solutions λ^*. Then, we compare λ^* with the approximate solutions $\lambda^{1'}$ and $\lambda^{2'}$.

The detailed computational results are shown in Table 6.5. Table 6.6 and Figure 6.3 shows the relative differences $\sigma^{1'}$ and $\sigma^{2'}$.

	$\lambda^{1'}$	$\lambda^{2'}$	λ^*
Example 1	0.1000	0.1000	0.3835
Example 2	0.1415	0.0550	0.3203
Example 3	0.1600	0.1455	0.6250
Example 4	0.1667	0.1905	0.5400
Example 5	0.3500	0.2381	0.5259
Example 6	0.0408	0.0714	0.7895
Example 7	0.1111	0.0741	0.2532
Example 8	0.2560	0.4610	0.5590
Example 9	0.0143	0.0286	0.4607
Example 10	0.4646	0.4873	0.6324
Example 11	0.0667	0.0667	0.2416
Example 12	0.0730	0.0286	0.1388
Example 13	0.1600	0.1333	0.6923
Example 14	0.1000	0.0889	0.3319
Example 15	0.3056	0.5500	0.6689
Example 16	0.0375	0.0625	0.4802
Example 17	0.0373	0.0533	0.2726
Example 18	0.5051	0.7299	0.8710
Example 19	0.0400	0.0217	0.2932
Example 20	0.2226	0.1228	0.6426

Table 6.5: Testing results of examples MP with $0 < \lambda^* < 1$

	$\sigma^{1'}$	$\sigma^{2'}$
Example 1	0.2607	0.2607
Example 2	0.4418	0.1717
Example 3	0.2560	0.2327
Example 4	0.3086	0.3527
Example 5	0.6656	0.4528
Example 6	0.0517	0.0905
Example 7	0.4389	0.2926
Example 8	0.4580	0.8246
Example 9	0.0310	0.0620
Example 10	0.7346	0.7707
Example 11	0.2759	0.2759
Example 12	0.5261	0.2059
Example 13	0.2311	0.1926
Example 14	0.3013	0.2678
Example 15	0.4568	0.8222
Example 16	0.0781	0.1302
Example 17	0.1370	0.1957
Example 18	0.5799	0.8381
Example 19	0.1364	0.0742
Example 20	0.3464	0.1910

Table 6.6: Relative differences from $\lambda^{1'}$ and $\lambda^{2'}$ versus λ^* for examples MP with $0 < \lambda^* < 1$

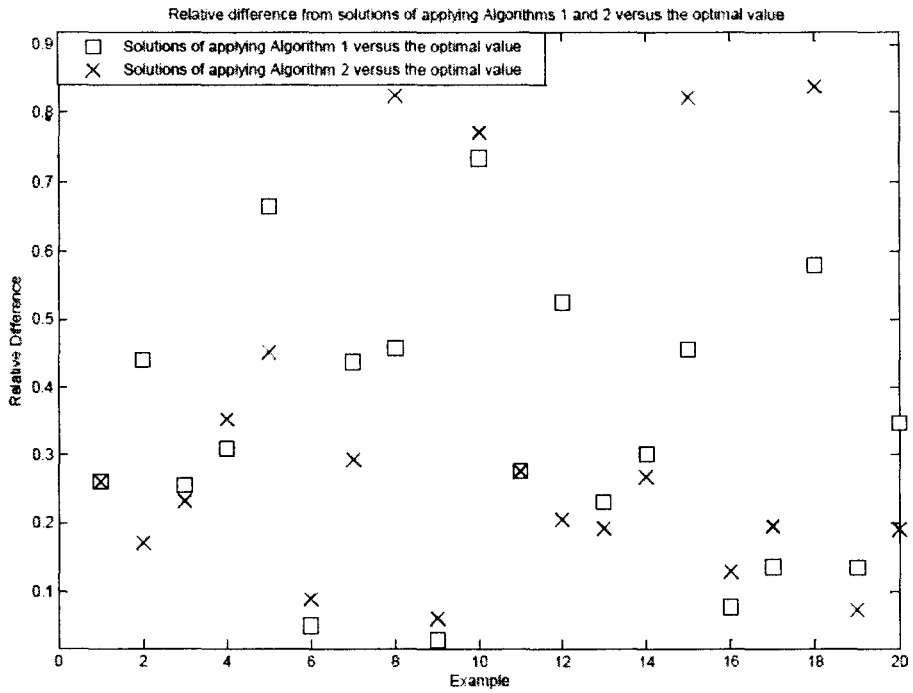


Figure 6.3: Relative differences from $\lambda^{1'}$ and $\lambda^{2'}$ versus λ^* for examples MP with $0 < \lambda^* < 1$

In the above examples with $0 < \lambda^* < 1$, we find that some solutions produced by our approximation algorithms are very bad (e.g. examples 6, 9, 16 and 19). Remember in the general case of TDCCP, we use the formula $\delta_{j,k} := \frac{\tilde{x}_{j,k}}{(\omega_j \mu_{j,k})}$ to generate the server assignments and the throughput, where $\omega_j := \mu_j^{max}/\mu_j^{min}$. We observe that in most of our examples, ω_j is a large number. Obviously, this is the reason for those bad approximation results. We also notice that our results of the algorithms in examples 5, 8, 10, 15 and 18 are much better. By analyzing the input data $\mu_{j,k}$, we find that it is because ω_j, $j = 1, ..., M$ in those examples are close to 1. In example 2, the throughput produced by applying Algorithm 1 is much better than applying Algorithm 2. By analyzing the server assignment $\delta_{j,k}$, we find two explanations. First, the server assignment in Algorithm 2 shows that slower servers are over assigned at a class; second, it shows that a fast server is working at a class with lower customer arrival rates.

Testing examples when Condition 2 is satisfied

We design another 20 examples (see Appendix B.4) when Condition 2 is satisfied. The output data are $\delta_{j,k}$ and λ. The optimal solutions of relaxations λ^* and $\delta^*_{j,k}$ are also generated by 'linprog'. $\lambda^{1'}$ and $\delta^{1'}_{j,k}$ are the solutions by applying Algorithm 1. $\lambda^{2'}$ and $\delta^{2'}_{j,k}$ are the solutions by applying Algorithm 2. We also define $\sigma^{1'}$ to be the relative difference from $\lambda^{1'}$ versus λ^* and $\sigma^{2'}$ to be the relative difference from $\lambda^{2'}$ versus λ^*. Table 6.7 is the detailed computational results. Table 6.8 and Figure 6.4 show the relative differences $\sigma^{1'}$ and $\sigma^{2'}$.

	$\lambda^{1'}$	$\lambda^{2'}$	λ^*
Example 1	0.1000	0.1250	1.1289
Example 2	0.0300	0.0300	1.4552
Example 3	0.0513	0.1111	1.3302
Example 4	2.5000	2.6471	3.3811
Example 5	0.3636	0.4959	2.6012
Example 6	0.7407	1.1111	1.9888
Example 7	0.0075	0.0333	1.2673
Example 8	0.5119	1.8750	2.1610
Example 9	0.1074	0.1111	2.1195
Example 10	0.0645	0.1400	1.5820
Example 11	0.2000	0.2000	1.6796
Example 12	0.0100	0.0800	2.1912
Example 13	0.1000	0.1250	1.8703
Example 14	0.0667	0.0952	2.3590
Example 15	1.8000	1.6200	2.3315
Example 16	0.1185	0.1619	2.2881
Example 17	0.0944	0.0577	4.0462
Example 18	3.5866	3.5866	4.8380
Example 19	0.0969	0.0286	1.9708
Example 20	0.3822	1.1483	2.6244

Table 6.7: Testing results of examples MP with $\lambda^* \geq 1$

	$\sigma^{1'}$	$\sigma^{2'}$
Example 1	0.0886	0.1107
Example 2	0.0206	0.0206
Example 3	0.0386	0.0835
Example 4	0.7394	0.7829
Example 5	0.1398	0.1906
Example 6	0.3725	0.5587
Example 7	0.0059	0.0263
Example 8	0.2369	0.8677
Example 9	0.0507	0.0524
Example 10	0.0408	0.0888
Example 11	0.1191	0.1191
Example 12	0.0046	0.0365
Example 13	0.0535	0.0668
Example 14	0.0283	0.0404
Example 15	0.7720	0.6948
Example 16	0.0518	0.0708
Example 17	0.0233	0.0143
Example 18	0.7413	0.7413
Example 19	0.0492	0.0145
Example 20	0.1456	0.4376

Table 6.8: Relative differences from $\lambda^{1'}$ and $\lambda^{2'}$ versus λ^* for examples MP with $\lambda^* \geq 1$

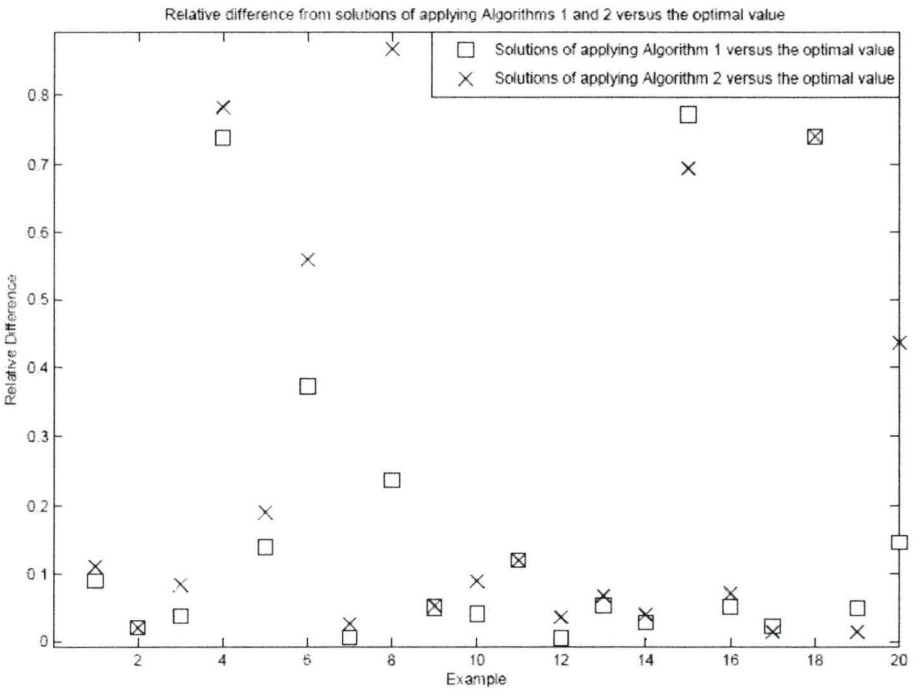


Figure 6.4: Relative differences from $\lambda^{1'}$ and $\lambda^{2'}$ versus λ^* for examples MP with $\lambda^* \geq 1$

In the above tests, the performances of our algorithms for solving the general TDCCP are very poor. As we discussed before, the reason is also because of the parameter ω_j. For those examples with ω_j close to 1, we can still obtain good approximate results, e.g. examples 4, 6, 8, 15 and 18. In most examples, we observe that the throughput $\lambda^{2'}$ produced by applying Algorithm 2 is better than $\lambda^{1'}$. In example 12, $\lambda^{2'}$ is much better. We also notice that the solutions in examples 7 and 19 are extremely bad. The reason is that there exists a big 'gap' (i.e. $\omega_{max} >= 50$) between the highest service rate and the lowest service that for a particular server j working at classes. It is also because the server assignments produced by the two algorithms use the servers with low service rates to work.

6.1.3 TDCCP with costs when all classes are identical

Design of Examples

In our examples of TDCCP with costs when all classes are identical, the input data are the service rates $\mu_{j,k}$, the numbers a_k, the costs $r_{j,k}$, the budget C and the flexibility bounds c_k. We define two integers M and K, where M is the number of servers and K is the number of classes. The values of M and K are integers in $(0, 20]$. Since all classes are identical, we let $\mu_{j,k}$ and $r_{j,k}$ be the $M \times K$ matrices with the same value in each row vector. The values of each entry in $\mu_{j,k}$, $r_{j,k}$ and a_k are randomly defined to be a real number in $(0, 100]$. Each entry in c_k is an integer which satisfies $0 < c_k \leq M$. Let $r_j^{max} := \max_j r_{j,k}$. The budget C is a real number randomly selected in $(0, 10\sum_{j=1}^{M} r_j^{max}]$, by which we guarantee that the number C cannot be arbitrarily large. We define this upper bound for C because when C is large enough, the cost constraint (5) in (MPC2) will be redundant.

Similar to Section 6.1.1, there will be two different tests. The first test includes 20 examples that satisfy Condition 1 ($0 < \lambda^* < 1$), and the second test includes another 20 examples that satisfy Condition 2 ($\lambda^* \geq 1$). The examples will be tested by Algorithm 3, Algorithm 4 and 'linprog'. The testing environment is MATLAB 7.0.

Testing examples when Condition 1 is satisfied

There are 20 examples (see Appendix B.5) with different input data $\mu_{j,k}$, a_k, $r_{j,k}$, C and c_k. The output data are the server policy $\delta_{j,k}$ and the throughput λ. First we run the relaxations of 20 examples by 'linprog' and let λ^* and $\delta_{j,k}^*$ be the optimal values. We know that $0 < \lambda^* < 1$ for all of the examples. Let λ^3, $\delta_{j,k}^3$ be the solution of Algorithm 3, and similarly, let λ^4 and $\delta_{j,k}^4$ be the solution of Algorithm 4. We define σ^3 and σ^4 to be the relative differences from λ^3 and λ^4 versus λ^*.

Table 6.9 is the detailed computational results. Table 6.10 and Figure 6.5 show the relative differences σ^3 and σ^4.

	λ^3	λ^4	λ^*
Example 1	0.1250	0.2222	0.2500
Example 2	0.3000	0.3000	0.3281
Example 3	0.6667	0.6667	0.8333
Example 4	0.1500	0.2857	0.3000
Example 5	0.4444	0.4444	0.5000
Example 6	0.2000	0.2000	0.2000
Example 7	0.1852	0.1852	0.2540
Example 8	0.4167	0.4167	0.4556
Example 9	0.6667	0.1333	0.7667
Example 10	0.4068	0.5854	0.6429
Example 11	0.0625	0.1250	0.1384
Example 12	0.2000	0.5161	0.5500
Example 13	0.4444	0.4444	0.5556
Example 14	0.4110	0.8333	0.9184
Example 15	0.6250	0.6667	0.7432
Example 16	0.2000	0.2581	0.2840
Example 17	0.2000	0.5882	0.7368
Example 18	0.5556	0.8654	0.8765
Example 19	0.6364	0.8000	0.9412
Example 20	0.3333	0.5000	0.5077

Table 6.9: Testing results of examples MPC2 with $0 < \lambda^* < 1$

	σ^3	σ^4
Example 1	0.5000	0.8889
Example 2	0.9143	0.9143
Example 3	0.8000	0.8000
Example 4	0.5000	0.9524
Example 5	0.8889	0.8889
Example 6	1.0000	1.0000
Example 7	0.7292	0.7292
Example 8	0.9146	0.9146
Example 9	0.8696	0.1739
Example 10	0.6328	0.9106
Example 11	0.4516	0.9032
Example 12	0.3636	0.9384
Example 13	0.8000	0.8000
Example 14	0.4475	0.9074
Example 15	0.8409	0.8970
Example 16	0.7043	0.9088
Example 17	0.2714	0.7983
Example 18	0.6338	0.9873
Example 19	0.6761	0.8500
Example 20	0.6556	0.9848

Table 6.10: Relative differences from λ^3 and λ^4 versus λ^* for examples MPC2 with $0 < \lambda^* < 1$

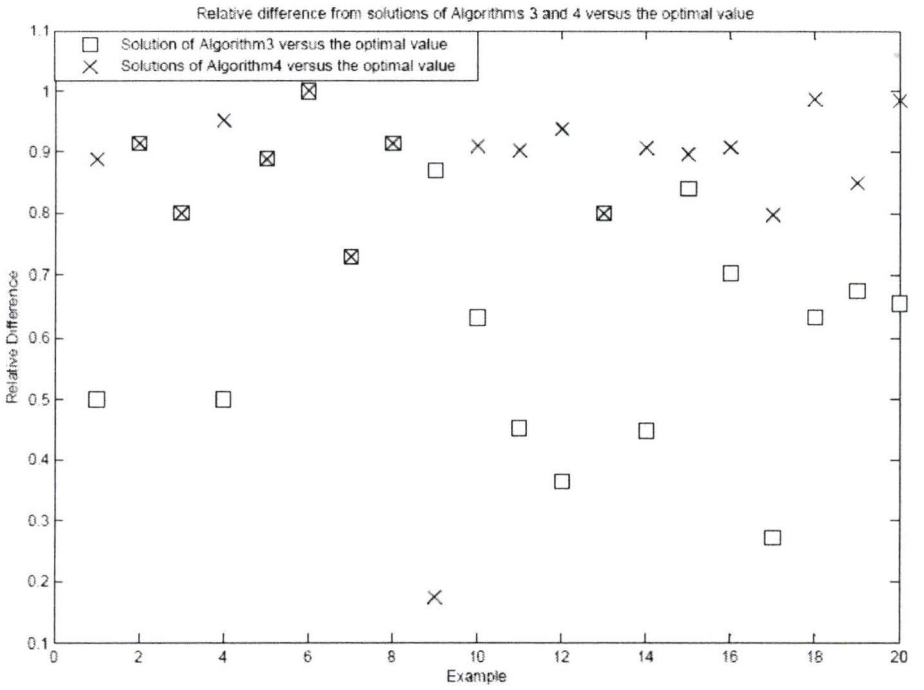


Figure 6.5: Relative differences from λ^3 and λ^4 versus λ^* for examples MPC2 with $0 < \lambda^* < 1$

According to the approximation results above, we observe that Algorithm 3 and Algorithm 4 perform very well in almost all of the 20 testing examples, especially in examples 2, 3, 6, 16 and 18. Comparing the results of Algorithm 3 and Algorithm 4, it is easy to see that most of the time, λ^4 is better than λ^3 (examples 1, 4, 10, 11, 12, 14, 15, 16, 18, 19 and 20). However, in example 9, the throughput produced by Algorithm 3 is 5 times better than the throughput produced by Algorithm 4. By analyzing the solution of example 9, we find that in the server assignment $\delta^3_{j,k}$ by Algorithm 3, the slowest server does not work at all, in other words, we say that this server is idle. However, in $\delta^4_{j,k}$, we find that the corresponding slowest server is assigned to a class by Algorithm 4, and as a result decreases the capacity of the queueing network. This is the reason why the solution of Algorithm 3 is better.

Testing examples when Condition 2 is satisfied

We design another 20 examples (see Appendix B.6) when Condition 2 is satisfied in TDCCP with costs. The output data are $\delta_{j,k}$ and λ. The optimal solutions of relaxations λ^* and $\delta^*_{j,k}$ are also generated by 'linprog'. λ^3 and $\delta^3_{j,k}$ are the solutions of Algorithm 3. λ^4 and $\delta^4_{j,k}$ are the solutions of Algorithm 4. We define σ^3 and σ^4 to be the relative differences from λ^3 and λ^4 versus λ^*. Table 6.11 is the detailed computational results. Table 6.12 and Figure 6.6 show the relative differences σ^3 and σ^4.

	λ^3	λ^4	λ^*
Example 1	1.4286	2.5000	3.0000
Example 2	1.8462	3.0000	3.7500
Example 3	1.5625	2.8571	3.1250
Example 4	2.5000	4.4444	4.7368
Example 5	1.3333	1.0000	1.5000
Example 6	4.0000	4.0000	4.0000
Example 7	2.3810	2.3810	3.2653
Example 8	3.7500	3.7500	4.1000
Example 9	8.0000	1.6000	9.2000
Example 10	1.6667	4.2105	4.8649
Example 11	0.6250	1.2500	1.2917
Example 12	5.0000	6.0000	8.2500
Example 13	1.6667	1.6667	2.0833
Example 14	4.2857	6.6667	6.9231
Example 15	4.7619	7.6923	9.1667
Example 16	3.0000	3.8710	4.2593
Example 17	4.4444	4.6154	5.1852
Example 18	3.3333	3.3333	3.5500
Example 19	1.7073	3.0000	3.9024
Example 20	5.0000	5.7143	7.3333

Table 6.11: Testing results of examples MPC2 with $\lambda^* \geq 1$

	σ^3	σ^4
Example 1	0.4762	0.8333
Example 2	0.4923	0.8000
Example 3	0.5000	0.9143
Example 4	0.5278	0.9383
Example 5	0.8889	0.6667
Example 6	1.0000	1.0000
Example 7	0.7292	0.7292
Example 8	0.9146	0.9146
Example 9	0.8696	0.1739
Example 10	0.3426	0.8655
Example 11	0.4839	0.9677
Example 12	0.6061	0.7273
Example 13	0.8000	0.8000
Example 14	0.6190	0.9630
Example 15	0.5195	0.8392
Example 16	0.7043	0.9088
Example 17	0.8571	0.8901
Example 18	0.9390	0.9390
Example 19	0.4375	0.7688
Example 20	0.6818	0.7792

Table 6.12: Relative differences from λ^3 and λ^4 versus λ^* for examples MPC2 with $\lambda^* \geq 1$

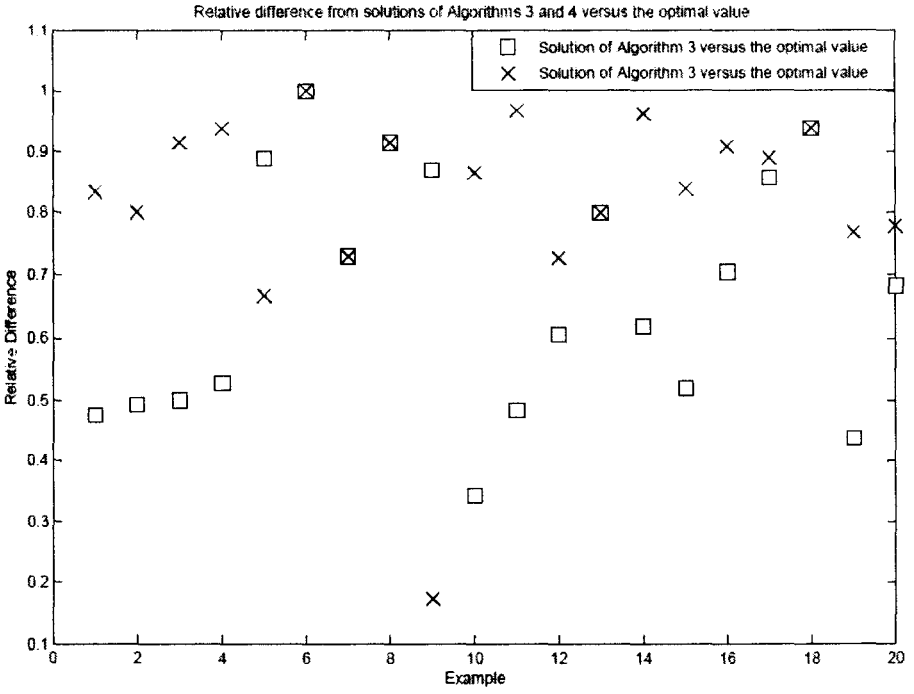


Figure 6.6: Relative differences from λ^3 and λ^4 versus λ^* for examples MPC2 with $\lambda^* \geq 1$

The results in the above examples by Algorithm 3 and Algorithm 4 are very good. In examples 6, 8 and 18, the throughputs produced by the two algorithms are close to λ^*. We also observe that the solution of Algorithm 4 are better than the solution of Algorithm 3 in most examples (examples 1, 2, 3, 4, 10, 11, 12, 14, 15, 16, 17, 19 and 20). In example 9, the results show that the throughput produced by Algorithm 3 is much better than Algorithm 4. By analyzing the server assignments $\delta^3_{j,k}$ and $\delta^4_{j,k}$, we find that the reason is the same as we discussed before: the slow servers are over assigned.

6.1.4 TDCCP with costs in the general case

Design of Examples

In the general case of TDCCP with costs, the input data are $\mu_{j,k}$, a_k, $r_{j,k}$, C and c_k. But we do not require that the matrix $\mu_{j,k}$ has the same entry value in each row

vector. Similar to the tests in Section 6.1.3, we define the values of M and K to be integers in $(0, 20]$ and c_k to be integers in $(0, M]$. The values of each entry in $\mu_{j,k}$, $r_{j,k}$ and a_k are random real numbers in $(0, 100]$. Let $r_j^{max} := \max_j r_{j,k}$. We also define the budget C to be a real number randomly selected in $(0, 10\sum_{j=1}^{M} r_j^{max}]$.

There will be two different tests. The first test includes 20 examples that satisfy Condition 1 ($0 < \lambda^* < 1$), and the second test includes 20 examples that satisfy Condition 2 ($\lambda^* \geq 1$).

The algorithm for solving the general (MPC2) will apply Algorithm 3 or Algorithm 4. All of the examples will be tested by applying Algorithm 3, Algorithm 4 and 'linprog'. The testing environment is also MATLAB 7.0.

Testing examples when Condition 1 is satisfied

We have 20 examples (see Appendix B.7) with different $\mu_{j,k}$, a_k, $r_{j,k}$, C and c_k. The output data are $\delta_{j,k}$ and λ. First we run the relaxations of 20 examples by 'linprog' and let λ^* and $\delta_{j,k}^*$ be the optimal. We know that $0 < \lambda^* < 1$ for all the examples. Let $\lambda^{3'}$ and $\delta_{j,k}^{3'}$ be the solutions by applying Algorithm 3. Let $\lambda^{4'}$ and $\delta_{j,k}^{4'}$ be the solutions by applying Algorithm 4. We also define $\sigma^{3'}$ and $\sigma^{4'}$ to be the relative differences from $\lambda^{3'}$ and $\lambda^{4'}$ versus λ^*.

Table 6.13 is the detailed computational results. Table 6.14 and Figure 6.7 show the relative differences $\sigma^{3'}$ and $\sigma^{4'}$.

	$\lambda^{3'}$	$\lambda^{4'}$	λ^*
Example 1	0.0400	0.0286	0.2288
Example 2	0.0333	0.0536	0.4200
Example 3	0.0077	0.0183	0.0952
Example 4	0.0313	0.0551	0.4712
Example 5	0.1569	0.1600	0.5274
Example 6	0.0708	0.1620	0.2085
Example 7	0.0117	0.0332	0.3683
Example 8	0.0671	0.0941	0.4601
Example 9	0.0476	0.0804	0.4381
Example 10	0.3563	0.6405	0.8378
Example 11	0.0154	0.0190	0.1837
Example 12	0.0192	0.0313	0.3115
Example 13	0.0038	0.0087	0.1373
Example 14	0.0357	0.1081	0.5485
Example 15	0.2833	0.4435	0.5445
Example 16	0.0091	0.0291	0.2230
Example 17	0.0202	0.0624	0.4058
Example 18	0.0851	0.1250	0.5409
Example 19	0.0554	0.1080	0.5430
Example 20	0.0185	0.0455	0.4272

Table 6.13: Testing results of examples MPC with $0 < \lambda^* < 1$

	$\sigma^{3'}$	$\sigma^{4'}$
Example 1	0.1748	0.1249
Example 2	0.0794	0.1276
Example 3	0.0808	0.1924
Example 4	0.0663	0.1170
Example 5	0.2974	0.3033
Example 6	0.3398	0.7773
Example 7	0.0318	0.0902
Example 8	0.1459	0.2045
Example 9	0.1087	0.1834
Example 10	0.4252	0.7644
Example 11	0.0838	0.1037
Example 12	0.0617	0.1003
Example 13	0.0280	0.0633
Example 14	0.0651	0.1971
Example 15	0.5203	0.8144
Example 16	0.0408	0.1303
Example 17	0.0498	0.1537
Example 18	0.1573	0.2311
Example 19	0.1019	0.1989
Example 20	0.0433	0.1064

Table 6.14: Relative differences from $\lambda^{3'}$ and $\lambda^{4'}$ versus λ^* for examples MPC with $0 < \lambda^* < 1$

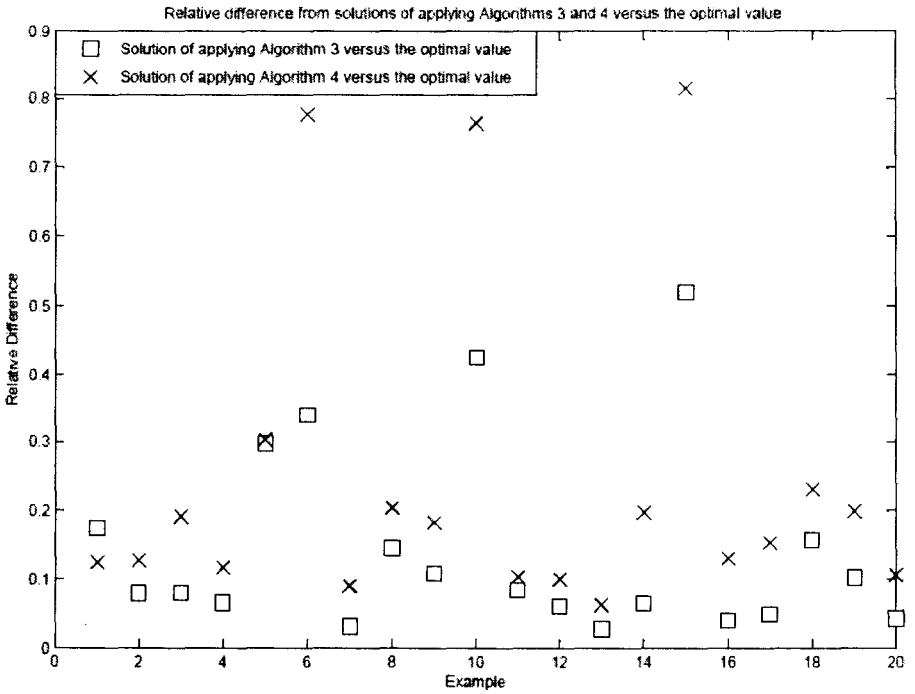


Figure 6.7: Relative differences from $\lambda^{3'}$ and $\lambda^{4'}$ versus λ^* for examples MPC with $0 < \lambda^* < 1$

In those examples with $0 < \lambda^* < 1$, we apply the same formula $\delta_{j,k} := \frac{\tilde{x}_{j,k}}{(\omega_j \mu_{j,k})}$ to generate the approximate solutions, where $\omega_j := \mu_j^{max}/\mu_j^{min}$. Thus it is reasonable that we may get the bad approximate solutions again. Also, when ω_j is close to 1, the approximation result we get will be much better (examples 6, 10 and 15). It is easy to see that in all examples except example 1, applying Algorithm 4 is better than applying Algorithm 3. In example 16, $\lambda^{4'}$ is much better than $\lambda^{3'}$. We analyze the server assignment policy produced by applying Algorithm 3 and find that the slower servers are assigned to classes with larger proportions of working time.

Testing examples when Condition 2 is satisfied

We design another 20 examples (see Appendix B.8) when Condition 2 is satisfied in the general TDCCP with costs. The optimal solutions of the relaxations λ^* and $\delta^*_{j,k}$ are generated by 'linprog'. $\lambda^{3'}$ and $\delta^{3'}_{j,k}$ are the solutions by applying Algorithm 3. $\lambda^{4'}$

and $\delta_{j,k}^{4'}$ are the solutions by applying Algorithm 4. We also define $\sigma^{3'}$ and $\sigma^{4'}$ to be the relative differences from $\lambda^{3'}$ and $\lambda^{4'}$ versus λ^*. The detailed computational results are shown in Table 6.15. Table 6.16 and Figure 6.8 show the relative differences $\sigma^{3'}$ and $\sigma^{4'}$.

	$\lambda^{3'}$	$\lambda^{4'}$	λ^*
Example 1	0.5000	0.6667	3.0682
Example 2	0.6000	0.5357	5.7143
Example 3	2.0571	2.4000	3.9916
Example 4	0.3571	0.5882	5.2632
Example 5	1.8667	3.5000	4.9148
Example 6	0.2143	0.4390	5.7852
Example 7	0.5208	0.5870	6.7518
Example 8	0.4878	0.7018	3.2648
Example 9	0.3571	0.8333	4.5830
Example 10	0.6400	0.9412	5.7398
Example 11	0.4444	0.5714	5.7447
Example 12	1.1250	3.2727	6.4060
Example 13	0.0645	0.0789	2.5263
Example 14	2.1538	4.4167	9.9785
Example 15	0.5556	0.9524	7.1909
Example 16	0.3226	0.9143	5.9669
Example 17	0.1307	0.1449	2.3374
Example 18	0.4000	0.9091	4.9437
Example 19	0.5536	1.0652	5.5583
Example 20	0.2083	0.4255	4.2722

Table 6.15: Testing results of examples MPC with $\lambda^* \geq 1$

	$\sigma^{3'}$	$\sigma^{4'}$
Example 1	0.1630	0.2173
Example 2	0.1050	0.0938
Example 3	0.5154	0.6013
Example 4	0.0679	0.1118
Example 5	0.3798	0.7121
Example 6	0.0370	0.0759
Example 7	0.0771	0.0869
Example 8	0.1494	0.2149
Example 9	0.0779	0.1818
Example 10	0.1115	0.1640
Example 11	0.0774	0.0995
Example 12	0.1756	0.5109
Example 13	0.0255	0.0313
Example 14	0.2158	0.4426
Example 15	0.0773	0.1324
Example 16	0.0541	0.1532
Example 17	0.0559	0.0620
Example 18	0.0809	0.1839
Example 19	0.0996	0.1916
Example 20	0.0488	0.0996

Table 6.16: Relative differences from $\lambda^{3'}$ and $\lambda^{4'}$ versus λ^* for examples MPC with $\lambda^* \geq 1$

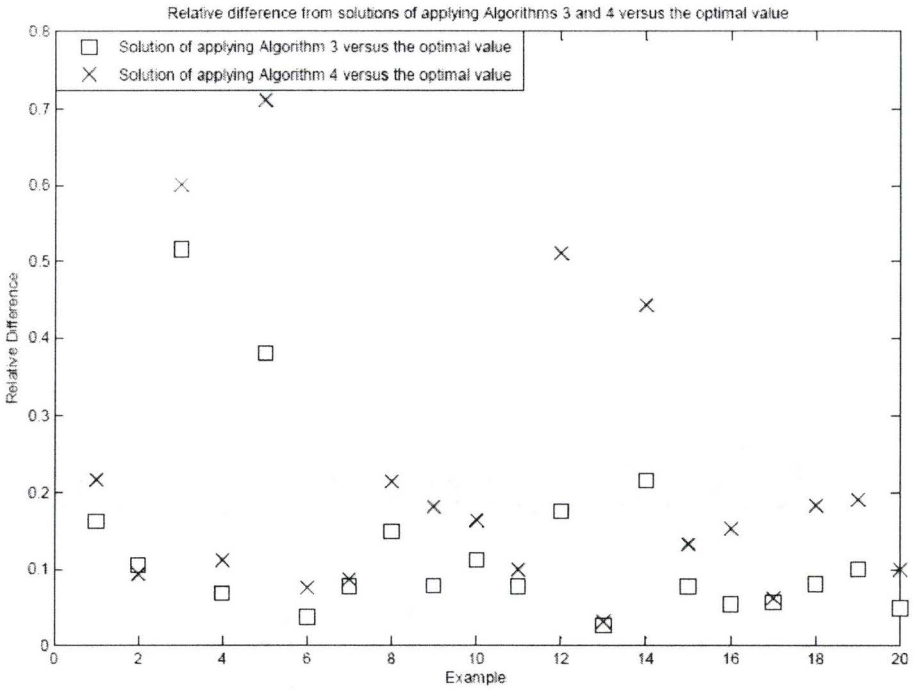


Figure 6.8: Relative differences from $\lambda^{3'}$ and $\lambda^{4'}$ versus λ^* for examples MPC with $\lambda^* \geq 1$

The above results show the poor performance of the algorithms for solving the general case of TDCCP with costs. It is also clear that in all examples except example 2, applying Algorithm 4 generates a better solution. Only the throughputs produced in examples 3, 5, 12 and 14 are good, while in others, especially in example 13, both algorithms perform poorly. The reason is also because the number ω_j is too big. Therefore, we finally obtain the very small $\delta_{j,k}$, which gives us the bad approximation results.

6.2 Summary of Experiments

By the observations from our experiments, we can conclude that in the case when service rates are independent of classes (all classes are identical), all algorithms can usually produce very good approximation results for both TDCCP and TDCCP with

costs. Let $\omega_{max} := \max_j\{\omega_j\}$. In the general case of TDCCP and TDCCP with costs, the approximation results will depend on the value ω_{max}. Our experiment results tell us that when $\omega_{max} \approx 1$, we can still obtain good approximate solutions. However, if ω_{max} is very large, we will achieve bad approximate solutions.

In the tests of TDCCP without costs, we find that in most examples, the approximate solutions of Algorithm 2 are at least as good as the approximate solutions of Algorithm 1. Thus, Algorithm 2 is suggested for solving TDCCP without costs. In the tests of TDCCP with costs, we find that Algorithm 4 can beat Algorithm 3 in most examples. Therefore, Algorithm 4 is strongly recommended for solving TDCCP with costs.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

We have described an entire model of queueing networks with limited flexibility. By using some parameters of queueing networks, we presented the deterministic optimization problem TDCCP. Since the solution of the problem TDCCP can be directly mapped back to a solution of the original queueing network problem, we established the connections between both problems. Furthermore, we extended the problem TDCCP to the problem TDCCP with costs by introducing costs into the system and gave its mathematical definition.

After obtaining the mathematical programs TDCCP and TDCCP with costs, we showed that when all servers are identical, both TDCCP and TDCCP with costs can be solved in polynomial time. However, when all classes are identical, we showed that both TDCCP and TDCCP with costs are NP-Complete. Thus we designed two approximation algorithms Algorithm 1, Algorithm 2 for solving TDCCP, and two approximation algorithms Algorithm 3, Algorithm 4 for solving TDCCP with costs. Then we proved that Algorithm 1 is a 1/10-approximation algorithm, Algorithm 3 is a 1/12-approximation algorithm, Algorithm 2 and Algorithm 4 are $\frac{1}{2(1+\rho)}$-approximation algorithms, where $\rho := \max_{j,k} \frac{a_k \lambda^*}{c_k \mu_j}$. By applying Algorithms 1, 2, 3 and 4, we also gave approximation algorithms for solving the general case of TDCCP and TDCCP with costs.

We implemented the approximation algorithms for solving TDCCP and TDCCP

with costs. Detailed computational results on several experiments showed that Algorithms 1, 2, 3 and 4 can both produce good approximate results in the case of $\mu_{j,k} = \mu_j$. However, applying Algorithms 1, 2, 3 and 4 for solving the general cases of TDCCP and TDCCP with costs might incur bad approximate results (though they are still much better than theoretical lower bounds).

7.2 Future works

The numerical results show that our algorithms do not perform very well in the general cases of TDCCP and TDCCP with costs. Thus, designing a better approximation algorithm for solving the general cases of TDCCP and TDCCP with costs remains to be studied.

Based on our experiments, we also observe three methods which may improve the throughput of the system. The first method is that we try to prevent idle servers, in other words, we should keep each server as busy as possible. The second method is that we can increase the proportion of server working time $\delta_{j,k}$ for faster servers and decrease $\delta_{j,k}$ for those slower servers. The third method is that we can allow preemption of activities, i.e. let faster servers have preemption to work and classes with larger customer arrival rates have preemption to be processed. Future work will also include enabling the above three methods applicable and designing better approximation algorithms in queueing networks with limited flexibility.

Bibliography

[AAD01] S. Andradóttir, H. Ayhan, and D. G. Down. Server assignment policies for maximizing the steady-state throughput of finite queueing systems. *Management Science*, 47:1421–1439, 2001.

[AAD03] S. Andradóttir, H. Ayhan, and D. G. Down. Dynamic server allocation for queueing networks with flexible servers. *Operations Research*, 51:952–968, 2003.

[BKS02] G. Baier, E. Köhler, and M. Skutella. On the k-splittable flow problem. *Proceedings of ESA '02*, 2002.

[DGG99] Y. Dinitz, N. Garg, and M. Goemans. On the single-source unsplittable flow problem. *Combinatorica*, 19:17–41, 1999.

[DL05] J. G. Dai and W. Lin. Maximum pressure policies in stochastic processing networks. *Operations Research*, 53:197–218, 2005.

[HS96] F. S. Hillier and K. C. So. On the simultaneous optimization of server and work allocations in production line systems with variable processing times. *Operations Research*, 44:435–443, 1996.

[HvO04] W. J. Hopp and M. P. van Oyen. Agile workforce evaluation: A framework for cross-training and coordination. *IIE Transactions*, 36(10):919–940, 2004.

[Kar72] R. M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, pages 85–103, 1972. Plenum Press.

[Kle96] J. M. Kleinberg. Single-source unsplittable flow. *Proceedings 37th Annual Symposium on Foundation of Computer Science (FOCS'96)*, pages 68–77, 1996.

[KS02] S. G. Kolliopoulos and C. Stein. Approximation algorithms for single-source unsplittable flow. *SIAM J. Computing*, 31:919–946, 2002.

[Lov86] László Lovász. *Matching theory.* 1986.

[LST90] J. K. Lenstra, D. B. Shmoys, and E. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46(3):259–271, 1990.

[Mur95] Katta G. Murty. *Operations Research Deterministic Optimization Models.* 1995.

[OMT90] J. Ostalaza, J. McClain, and J. Thomas. The use of dynamic (state-dependent) assembly-line balancing to improve throughput. *Journal of Manufacturing and Operations Management*, 3:105–133, 1990.

[Sch86] Alexander Schrijver. *Theory of Linear and Integer Programming.* 1986.

[Sim72] Donald M. Simmons. *Linear Programming for Operations Research.* 1972.

[Sku02] M. Skutella. Approximating the single source unsplittable min-cost flow problem. *Mathematical Programming B*, 91:493–514, 2002.

[SM90] Farhad Shahrokhi and D. W. Matula. The maximum concurrent flow problem. *Journal of the ACM*, 37(2):318–334, 1990.

[ST93] D. B. Shmoys and E. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming A*, 62(3):461–474, 1993.

[SXYZ01] M. S. Squillante, C. H. Xia, D. D. Yao, and L. Zhang. Threshold-based priority policies for parallel-server systems with affinity scheduling. In *Proceedings of the 2001 American Control Conference*, pages 2992–2999, 2001.

[TB00] L. Tassiulas and P. B. Bhattacharya. Allocation of independent resources for maximal throughput. *Stochastic Models*, 16:27–48, 2000.

[Tod02] Michael J. Todd. The many facets of linear programming. *Mathematical Programming*, 91(3):417–436, 2002.

Appendix A

Symbols and Acronyms

Name	Definition
AP	Mathematical program of the assignment problem
MCM	Mathematical program of the minimum cost bipartite matching problem
MP	Mathematical program of TDCCP
MP'	Mathematical program of TDCCP when $\mu = \alpha \cdot \beta^T$
MPC	Mathematical program of TDCCP with costs
MPC'	Mathematical program of TDCCP with costs when $\mu = \alpha \cdot \beta^T$
M1	Mathematical program of the scheduling of unrelated machines problem
MP1	Mathematical program of TDCCP when $\mu_{j,k} = \mu_k$
MPC1	Mathematical program of TDCCP with costs when $\mu_{j,k} = \mu_k$
MP2	Mathematical program of TDCCP when $\mu_{j,k} = \mu_j$
MPC2	Mathematical program of TDCCP with costs when $\mu_{j,k} = \mu_j$
MP2'	An Instance of the maximum concurrent multicommodity k-splittable flow problem
MP2"	Relaxation of MP2'

MPC2'	An instance of the constrained maximum concurrent Multicommodity k-splittable flow problem
MPC2"	Relaxation of MPC2'
MP3	An instance of the maximum concurrent multicommodity k-splittable flow problem
MPC3	An instance of the constrained maximum concurrent multicommodity k-splittable flow problem
SC1	Relaxation of an instance of the scheduling of unrelated machines problem with costs
SC2	Relaxation of an instance of the scheduling of unrelated machines problem with costs
S1	Relaxation of an instance of the scheduling of unrelated machines problem
S2	Relaxation of an instance of the scheduling of unrelated machines problem
TDCCP	Total discrete capacity constrained problem

Appendix B

Testing Examples

B.1 TDCCP without costs when all classes are identical and Condition 1 is satisfied

Example 1:

$$\mu_{j,k} = \begin{pmatrix} 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 1 & 1 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Example 2:

$$\mu_{j,k} = \begin{pmatrix} 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.8 & 0.8 & 0.8 & 0.8 & 0.8 & 0.8 & 0.8 & 0.8 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \end{pmatrix}$$

Example 3:

$$\mu_{j,k} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

Example 4:

$$\mu_{j,k} = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 4 & 4 & 4 & 4 & 4 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \end{pmatrix}$$

Example 5:

$$\mu_{j,k} = \begin{pmatrix} 3 & 3 & 3 & 3 & 3 & 3 \\ 2 & 2 & 2 & 2 & 2 & 2 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 1.5 & 1.5 & 1.5 & 1.5 & 1.5 & 1.5 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

Example 6:

$$\mu_{j,k} = \begin{pmatrix} 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \end{pmatrix}$$

Example 7:

$$\mu_{j,k} = \begin{pmatrix} 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 \\ 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 1.1 & 1.1 & 1.1 & 1.1 & 1.1 & 1.1 & 1.1 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 5 & 5 & 5 & 5 & 5 & 5 & 5 \end{pmatrix}$$

Example 8:

$$\mu_{j,k} = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 0.8 & 0.8 & 0.8 & 0.8 & 0.8 & 0.8 & 0.8 & 0.8 & 0.8 & 0.8 \\ 0.8 & 0.8 & 0.8 & 0.8 & 0.8 & 0.8 & 0.8 & 0.8 & 0.8 & 0.8 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Example 9:

$$\mu_{j,k} = \begin{pmatrix} 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 4 & 4 & 4 & 4 & 4 & 4 & 4 \end{pmatrix}$$

Example 10:

$$\mu_{j,k} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 \\ 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 10 & 10 & 10 & 10 & 10 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 3 & 3 & 3 & 3 & 3 \end{pmatrix}$$

Example 11:

$$\mu_{j,k} = \begin{pmatrix} 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

Example 12:

$$\mu_{j,k} = \begin{pmatrix} 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.8 & 0.8 & 0.8 & 0.8 & 0.8 & 0.8 & 0.8 & 0.8 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

Example 13:

$$\mu_{j,k} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

Example 14:

$$\mu_{j,k} = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \end{pmatrix}$$

Example 15:

$$\mu_{j,k} = \begin{pmatrix} 3 & 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 & 4 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 1.5 & 1.5 & 1.5 & 1.5 & 1.5 & 1.5 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Example 16:

$$\mu_{j,k} = \begin{pmatrix} 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 \\ 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 4.9 & 4.9 & 4.9 & 4.9 & 4.9 & 4.9 & 4.9 & 4.9 & 4.9 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

Example 17:

$$\mu_{j,k} = \begin{pmatrix} 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 \\ 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 1.9 & 1.9 & 1.9 & 1.9 & 1.9 & 1.9 & 1.9 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 4 & 4 & 4 & 4 & 3 & 3 & 3 \end{pmatrix}$$

Example 18:

$$\mu_{j,k} = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 1.8 & 1.8 & 1.8 & 1.8 & 1.8 & 1.8 & 1.8 & 1.8 & 1.8 & 1.8 \\ 1.8 & 1.8 & 1.8 & 1.8 & 1.8 & 1.8 & 1.8 & 1.8 & 1.8 & 1.8 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Example 19:

$$\mu_{j,k} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 9 & 9 & 9 & 9 & 9 & 9 & 9 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 7 & 7 & 7 & 7 & 7 & 7 & 7 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 4 & 4 & 4 & 4 & 4 & 4 & 4 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 2 & 2 & 2 & 1 & 1 & 1 \end{pmatrix}$$

Example 20:

$$\mu_{j,k} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 \\ 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 3 & 3 & 3 & 3 & 3 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 1 & 1 & 2 & 2 \end{pmatrix}$$

B.2 TDCCP without costs when all classes are identical and Condition 2 is satisfied

Example 1:

$$\mu_{j,k} = \begin{pmatrix} 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Example 2:

$$\mu_{j,k} = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 1 & 1 & 0.5 & 0.5 & 0.5 & 0.25 & 0.25 & 0.5 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \end{pmatrix}$$

Example 3:

$$\mu_{j,k} = \begin{pmatrix} 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 \\ 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

Example 4:

$$\mu_{j,k} = \begin{pmatrix} 20 & 20 & 20 & 20 & 20 & 20 & 20 & 20 & 20 & 20 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \end{pmatrix}$$

Example 5:

$$\mu_{j,k} = \begin{pmatrix} 3 & 3 & 3 & 3 & 3 & 3 \\ 2 & 2 & 2 & 2 & 2 & 2 \\ 5 & 5 & 5 & 5 & 5 & 5 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 1 & 1 & 0.5 & 0.5 & 0.5 & 1 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

Example 6:

$$\mu_{j,k} = \begin{pmatrix} 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \end{pmatrix}$$

Example 7:

$$\mu_{j,k} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 1.1 & 1.1 & 1.1 & 1.1 & 1.1 & 1.1 & 1.1 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 5 & 5 & 5 & 5 & 5 & 5 & 5 \end{pmatrix}$$

Example 8:

$$\mu_{j,k} = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Example 9:

$$\mu_{j,k} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 \\ 10 & 10 & 10 & 10 & 10 & 10 & 10 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 4 & 4 & 4 & 4 & 4 & 4 & 4 \end{pmatrix}$$

Example 10:

$$\mu_{j,k} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 \\ 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 3 & 3 & 3 & 3 & 3 \end{pmatrix}$$

Example 11:

$$\mu_{j,k} = \begin{pmatrix} 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

Example 12:

$$\mu_{j,k} = \begin{pmatrix} 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.8 & 0.8 & 0.8 & 0.8 & 0.8 & 0.8 & 0.8 & 0.8 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

Example 13:

$$\mu_{j,k} = \begin{pmatrix} 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 \\ 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 \\ 30 & 30 & 30 & 30 & 30 & 30 & 30 & 30 & 30 & 30 & 30 & 30 \\ 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 & 1 & 1 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

Example 14:

$$\mu_{j,k} = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \end{pmatrix}$$

Example 15:

$$\mu_{j,k} = \begin{pmatrix} 9 & 9 & 9 & 9 & 9 & 9 \\ 4 & 4 & 4 & 4 & 4 & 4 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Example 16:

$$\mu_{j,k} = \begin{pmatrix} 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 \\ 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 50 & 50 & 50 & 50 & 50 & 50 & 50 & 50 & 50 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 0.9 & 0.9 & 0.9 & 0.9 & 0.9 & 0.9 & 0.9 & 0.9 & 0.9 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

Example 17:

$$\mu_{j,k} = \begin{pmatrix} 0.8 & 0.8 & 0.8 & 0.8 & 0.8 & 0.8 & 0.8 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 0.9 & 0.9 & 0.9 & 0.9 & 0.9 & 0.9 & 0.9 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 4 & 4 & 4 & 4 & 3 & 3 & 3 \end{pmatrix}$$

Example 18:

$$\mu_{j,k} = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 0.08 & 0.08 & 0.08 & 0.08 & 0.08 & 0.08 & 0.08 & 0.08 & 0.08 & 0.08 \\ 0.08 & 0.08 & 0.08 & 0.08 & 0.08 & 0.08 & 0.08 & 0.08 & 0.08 & 0.08 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Example 19:

$$\mu_{j,k} = \begin{pmatrix} 10 & 10 & 10 & 10 & 10 & 10 & 10 \\ 9 & 9 & 9 & 9 & 9 & 9 & 9 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 70 & 70 & 70 & 70 & 70 & 70 & 70 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 2 & 2 & 2 & 1 & 1 & 1 \end{pmatrix}$$

Example 20:

$$\mu_{j,k} = \begin{pmatrix} 10 & 10 & 10 & 10 & 10 \\ 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 \\ 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 0.3 & 0.4 & 1 & 0.2 & 0.3 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 1 & 1 & 2 & 2 \end{pmatrix}$$

B.3 TDCCP without costs in the general case when Condition 1 is satisfied

Example 1:

$$\mu_{j,k} = \begin{pmatrix} 0.5 & 0.5 & 0.5 & 0.6 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 2 & 1 & 1 & 3 & 1 & 2 & 1 & 2 & 1 & 1 \\ 1 & 1 & 4 & 1 & 1 & 2 & 1 & 3 & 1 & 1 \\ 2 & 5 & 2 & 3 & 1 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 1 & 1 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Example 2:

$$\mu_{j,k} = \begin{pmatrix} 0.2 & 0.2 & 0.1 & 0.2 & 0.2 & 0.2 & 0.2 & 0.3 \\ 1 & 0.8 & 0.8 & 0.8 & 0.8 & 1.2 & 0.8 & 0.8 \\ 1 & 1 & 1 & 0.2 & 1 & 1 & 0.2 & 1 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 3 & 3 & 3 & 3 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \end{pmatrix}$$

Example 3:

$$\mu_{j,k} = \begin{pmatrix} 1 & 2 & 1 & 1 & 1 & 1 & 1 & 4 & 1 \\ 4 & 3 & 3 & 1 & 3 & 3 & 3 & 3 & 1 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 1 & 1 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 2 & 1 & 1 & 2 & 3 & 3 & 2 & 1 & 1 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

Example 4:

$$\mu_{j,k} = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 1 & 3 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 3 & 3 & 3 & 2 & 3 & 2 & 1 & 1 & 3 & 3 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 2 & 2 & 2 & 2 & 1 & 1 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \end{pmatrix}$$

Example 5:

$$\mu_{j,k} = \begin{pmatrix} 3 & 3 & 2.8 & 3 & 3 & 3 \\ 1.8 & 1.8 & 2 & 1.9 & 1.8 & 1.8 \\ 1.7 & 2 & 1.9 & 1 & 1.8 & 1.9 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 3 & 1.8 & 2 & 1.8 & 3 & 1.5 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

Example 6:

$$\mu_{j,k} = \begin{pmatrix} 7 & 7 & 7 & 7 & 1 & 7 & 7 & 7 & 7 \\ 1 & 1 & 2 & 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 3 & 2 & 2 & 2 & 2 & 2 \\ 5 & 5 & 5 & 4 & 5 & 5 & 4 & 5 & 5 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 5 & 2 & 2 & 3 & 4 & 2 & 1 & 1 & 1 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Example 7:

$$\mu_{j,k} = \begin{pmatrix} 0.1 & 0.1 & 0.1 & 0.2 & 0.1 & 0.1 & 0.1 \\ 0.2 & 0.2 & 0.3 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.1 & 0.2 & 0.2 & 0.1 & 0.3 & 0.2 \\ 0.3 & 0.2 & 0.3 & 0.3 & 0.3 & 0.3 & 0.4 \\ 0.5 & 0.3 & 0.5 & 0.3 & 0.5 & 0.5 & 0.5 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 0.9 & 0.7 & 0.9 & 0.9 & 0.9 & 0.9 & 0.9 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 1 & 2 & 2 & 3 & 3 & 3 \end{pmatrix}$$

Example 8:

$$\mu_{j,k} = \begin{pmatrix} 1.9 & 1.89 & 2 & 2 & 2 & 2 & 2 \\ 4 & 4 & 3.92 & 3.9 & 4 & 4 & 4 \\ 3 & 3 & 3 & 3 & 2.9 & 2.95 & 3 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 1.8 & 1.9 & 1.9 & 0.6 & 1.2 & 5.8 & 2.9 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 2 & 2 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Example 9:

$$\mu_{j,k} = \begin{pmatrix} 0.1 & 0.1 & 0.2 & 0.1 & 0.1 & 0.1 & 0.1 \\ 2 & 1 & 2 & 2 & 2 & 1 & 2 \\ 3 & 3 & 3 & 4 & 3 & 3 & 3 \\ 4 & 1 & 1 & 1 & 4 & 4 & 4 \\ 7 & 2 & 1 & 2 & 3 & 3 & 1 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 8 & 2 & 9 & 4 & 2 & 2 & 2 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 1 & 1 & 2 & 2 & 2 & 2 \end{pmatrix}$$

Example 10:

$$\mu_{j,k} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 3 & 3 & 3 & 2.9 & 3.1 \\ 2.5 & 2.6 & 2.6 & 2.6 & 2.7 \\ 2 & 2 & 2 & 1.9 & 2 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 2 & 4 & 3 & 2 & 2.7 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 2 & 3 & 2 & 3 \end{pmatrix}$$

Example 11:

$$\mu_{j,k} = \begin{pmatrix} 0.5 & 0.8 & 0.5 & 0.5 & 0.8 & 0.5 & 0.5 & 0.7 & 0.5 & 0.5 \\ 2 & 1 & 1 & 3 & 1 & 4 & 1 & 1 & 1 & 1 \\ 2 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 1 & 1 \\ 2 & 4 & 2 & 3 & 3 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

Example 12:

$$\mu_{j,k} = \begin{pmatrix} 0.2 & 1 & 1 & 0.2 & 0.2 & 1 & 0.2 & 0.8 \\ 0.6 & 0.8 & 0.8 & 0.8 & 0.8 & 0.3 & 0.8 & 0.8 \\ 1 & 1 & 1 & 2 & 1 & 1 & 2 & 1 \\ 4 & 4 & 4 & 6 & 4 & 4 & 4 & 4 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \end{pmatrix}$$

Example 13:

$$\mu_{j,k} = \begin{pmatrix} 1 & 2 & 1 & 1 & 1 & 2 & 1 & 4 & 1 \\ 4 & 3 & 3 & 5 & 3 & 3 & 3 & 3 & 2 \\ 1 & 5 & 5 & 5 & 3 & 5 & 5 & 3 & 1 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 2 & 1 & 2 & 2 & 2 & 2 & 2 & 1 & 1 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Example 14:

$$\mu_{j,k} = \begin{pmatrix} 2 & 2 & 2 & 2 & 1 & 2 & 2 & 2 & 2 & 2 \\ 1 & 3 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & 1 \\ 1 & 1 & 1 & 5 & 1 & 1 & 1 & 1 & 1 & 1 \\ 3 & 3 & 3 & 2 & 3 & 2 & 1 & 1 & 3 & 3 \\ 2 & 2 & 4 & 2 & 2 & 2 & 7 & 2 & 2 & 2 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 3 & 3 & 2 & 2 & 1 & 1 & 6 & 6 & 6 & 6 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 2 & 2 & 2 & 1 & 1 & 1 & 4 & 4 & 4 \end{pmatrix}$$

Example 15:

$$\mu_{j,k} = \begin{pmatrix} 5.9 & 5.8 & 5.7 & 5.6 & 6 & 5.5 \\ 2.9 & 3 & 2.8 & 2.9 & 2.8 & 2.8 \\ 4.8 & 4.9 & 5 & 4.9 & 4.9 & 4.8 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 6 & 1.5 & 2 & 8 & 1.5 & 1.5 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Example 16:

$$\mu_{j,k} = \begin{pmatrix} 15 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 \\ 1 & 1 & 2 & 1 & 1 & 1 & 3 & 1 & 1 \\ 2 & 2 & 2 & 1 & 2 & 2 & 8 & 2 & 2 \\ 5 & 1 & 5 & 5 & 5 & 5 & 5 & 1 & 5 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 8 & 15 & 2 & 3 & 1 & 2 & 7 & 1 & 1 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 2 \end{pmatrix}$$

Example 17:

$$\mu_{j,k} = \begin{pmatrix} 0.5 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 1 & 0.2 \\ 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.9 \\ 0.5 & 0.5 & 0.7 & 0.5 & 0.5 & 0.5 & 0.5 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 1.9 & 0.7 & 1.9 & 0.9 & 0.9 & 0.9 & 0.9 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 1 & 2 & 2 & 3 & 3 & 3 \end{pmatrix}$$

Example 18:

$$\mu_{j,k} = \begin{pmatrix} 10 & 10 & 10.5 & 11 & 11 & 10 & 10.5 \\ 9 & 8.5 & 8.7 & 8.9 & 8.8 & 8.7 & 8.8 \\ 3 & 2.9 & 3 & 3 & 3 & 2.8 & 2.9 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 1.8 & 9 & 11 & 0.9 & 0.9 & 0.9 & 0.9 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 2 & 2 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Example 19:

$$\mu_{j,k} = \begin{pmatrix} 0.8 & 0.1 & 0.2 & 0.1 & 0.1 & 0.1 & 0.1 \\ 2 & 1 & 2 & 2 & 2 & 3 & 2 \\ 3 & 3 & 3 & 4 & 3 & 3 & 3 \\ 4 & 1 & 1 & 1 & 4 & 4 & 4 \\ 9 & 2 & 2 & 2 & 3 & 3 & 1 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 1 & 2 & 20 & 4 & 7 & 2 & 2 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 2 & 1 & 2 & 2 & 2 & 2 \end{pmatrix}$$

Example 20:

$$\mu_{j,k} = \begin{pmatrix} 1 & 1 & 2 & 1 & 1 \\ 3 & 3 & 3 & 2 & 3.5 \\ 1 & 1 & 1 & 3 & 4.7 \\ 2 & 2 & 4 & 5 & 2 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 3.5 & 4.7 & 1.5 & 5 & 1 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 3 & 3 & 1 & 3 \end{pmatrix}$$

B.4 TDCCP without costs in the general case when Condition 2 is satisfied

Example 1:

$$\mu_{j,k} = \begin{pmatrix} 5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 20 & 10 & 9 & 3 & 9 & 2 & 9 & 2 & 1 & 1 \\ 10 & 10 & 40 & 10 & 1 & 2 & 8 & 3 & 8 & 1 \\ 20 & 5 & 20 & 3 & 10 & 2 & 20 & 20 & 20 & 2 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Example 2:

$$\mu_{j,k} = \begin{pmatrix} 2 & 0.2 & 0.1 & 0.2 & 0.2 & 0.2 & 0.2 & 0.3 \\ 8 & 8 & 8 & 8 & 8 & 1.2 & 0.8 & 0.8 \\ 1 & 8 & 8 & 2 & 10 & 8 & 0.2 & 8 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 1 & 1 & 3 & 3 & 1 & 2 & 2 & 2 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \end{pmatrix}$$

Example 3:

$$\mu_{j,k} = \begin{pmatrix} 1 & 2 & 1 & 10 & 1 & 1 & 1 & 4 & 1 \\ 4 & 3 & 30 & 1 & 3 & 30 & 3 & 3 & 1 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 1 & 1 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 0.5 & 1 & 1 & 2 & 0.3 & 0.3 & 2 & 1 & 1 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

Example 4:

$$\mu_{j,k} = \begin{pmatrix} 9.7 & 10 & 9 & 10 & 9.8 & 9.6 & 9.6 & 10 & 9.7 & 9.8 \\ 4.5 & 4.7 & 5 & 4.8 & 5 & 4.9 & 5 & 4.5 & 4.5 & 4.5 \\ 4.7 & 4.8 & 5 & 4.7 & 5 & 4.8 & 4.5 & 5 & 4.9 & 4.9 \\ 8.9 & 8.5 & 9 & 8.7 & 8.7 & 9 & 8.7 & 9 & 8.8 & 8.6 \\ 19.7 & 20 & 19 & 19.5 & 19.6 & 19.3 & 19.9 & 19.4 & 19.8 & 19.5 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 2 & 2 & 0.2 & 2 & 1 & 1 & 2 & 0.2 & 2 & 2 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \end{pmatrix}$$

Example 5:

$$\mu_{j,k} = \begin{pmatrix} 3 & 30 & 3 & 30 & 3 & 30 \\ 1 & 10 & 2 & 10 & 10 & 1 \\ 0.5 & 2 & 5 & 1 & 0.5 & 0.5 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 1 & 0.5 & 1.5 & 0.8 & 3 & 1.5 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

Example 6:

$$\mu_{j,k} = \begin{pmatrix} 14 & 14.5 & 14 & 15 & 14.7 & 14.7 & 14.8 & 14.9 & 14.5 \\ 8 & 9 & 8.8 & 8.6 & 8 & 9 & 8.7 & 8 & 8 \\ 4.6 & 5 & 4.7 & 4.9 & 5 & 5 & 4.7 & 5 & 4.6 \\ 5 & 5 & 5 & 4.5 & 5 & 5 & 4.9 & 5 & 5 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 1 & 2 & 2 & 3 & 4 & 2 & 1 & 1 & 1 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Example 7:

$$\mu_{j,k} = \begin{pmatrix} 1 & 1 & 1 & 0.2 & 1 & 0.1 & 0.1 \\ 0.2 & 0.2 & 0.3 & 0.2 & 0.2 & 0.2 & 0.2 \\ 2 & 0.1 & 2 & 0.2 & 1 & 0.3 & 0.2 \\ 0.3 & 0.2 & 0.3 & 3 & 10 & 0.3 & 0.4 \\ 5 & 0.3 & 5 & 0.3 & 5 & 5 & 1 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 0.9 & 0.7 & 0.9 & 0.9 & 0.9 & 0.9 & 0.9 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 1 & 2 & 2 & 3 & 3 & 3 \end{pmatrix}$$

Example 8:

$$\mu_{j,k} = \begin{pmatrix} 10 & 10 & 9.5 & 9.6 & 9.7 & 9.9 & 10 \\ 19 & 18 & 20 & 19 & 18 & 18.5 & 18.9 \\ 6 & 5.5 & 5.7 & 5.7 & 5.6 & 5.8 & 5.9 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 1.8 & 1.9 & 1.9 & 0.6 & 1.2 & 5.8 & 2.9 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 2 & 2 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Example 9:

$$\mu_{j,k} = \begin{pmatrix} 1 & 1 & 2 & 0.1 & 0.1 & 1 & 0.1 \\ 2 & 1 & 2 & 2 & 2 & 1 & 2 \\ 3 & 3 & 9 & 4 & 3 & 5 & 3 \\ 4 & 9 & 9 & 1 & 4 & 4 & 4 \\ 7 & 2 & 1 & 2 & 8 & 3 & 1 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 1.8 & 2 & 1 & 0.4 & 1 & 2 & 2 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 1 & 1 & 2 & 2 & 2 & 2 \end{pmatrix}$$

Example 10:

$$\mu_{j,k} = \begin{pmatrix} 10 & 10 & 10 & 1 & 1 \\ 3 & 7 & 3 & 2 & 3.5 \\ 9 & 5 & 6 & 2 & 2.7 \\ 2 & 5 & 2 & 0.2 & 2 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 2 & 0.4 & 1 & 2 & 2.7 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 2 & 3 & 2 & 3 \end{pmatrix}$$

Example 11:

$$\mu_{j,k} = \begin{pmatrix} 5 & 0.8 & 5 & 0.5 & 8 & 0.5 & 5 & 7 & 0.5 & 0.5 \\ 2 & 8 & 8 & 9 & 9 & 4 & 9 & 9 & 8 & 1 \\ 20 & 10 & 10 & 15 & 8 & 20 & 2 & 20 & 8 & 1 \\ 2 & 40 & 20 & 30 & 30 & 20 & 20 & 20 & 20 & 2 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 2 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 2 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

Example 12:

$$\mu_{j,k} = \begin{pmatrix} 0.2 & 1 & 1 & 0.2 & 0.2 & 1 & 0.2 & 0.8 \\ 6 & 8 & 8 & 8 & 0.8 & 30 & 0.8 & 0.8 \\ 10 & 1 & 1 & 2 & 10 & 10 & 20 & 1 \\ 4 & 40 & 4 & 60 & 4 & 40 & 40 & 4 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 3 & 3 & 1 & 1 & 3 & 5 & 6 & 1 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \end{pmatrix}$$

Example 13:

$$\mu_{j,k} = \begin{pmatrix} 10 & 20 & 10 & 10 & 10 & 20 & 10 & 20 & 1 \\ 4 & 3 & 3 & 50 & 30 & 30 & 2 & 3 & 2 \\ 1 & 5 & 5 & 50 & 30 & 5 & 4 & 15 & 1 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 2 & 1 & 2 & 2 & 2 & 2 & 2 & 2 & 1 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Example 14:

$$\mu_{j,k} = \begin{pmatrix} 2 & 2 & 8 & 2 & 10 & 2 & 8 & 8 & 2 & 2 \\ 10 & 3 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & 1 \\ 1 & 1 & 1 & 5 & 1 & 1 & 1 & 1 & 1 & 1 \\ 3 & 30 & 3 & 2 & 30 & 2 & 1 & 1 & 30 & 30 \\ 2 & 2 & 4 & 2 & 2 & 2 & 5 & 2 & 2 & 2 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 3 & 3 & 2 & 2 & 1 & 1 & 1 & 1 & 3 & 2 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 2 & 2 & 2 & 1 & 1 & 1 & 4 & 4 & 4 \end{pmatrix}$$

Example 15:

$$\mu_{j,k} = \begin{pmatrix} 8.9 & 8.9 & 9 & 9 & 8.8 & 8.8 \\ 8.6 & 8 & 7.8 & 7.9 & 8 & 7.9 \\ 5 & 4.9 & 5 & 4.5 & 4.8 & 5 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 1 & 1.5 & 2 & 2 & 1.5 & 1.5 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Example 16:

$$\mu_{j,k} = \begin{pmatrix} 15 & 10 & 15 & 7 & 17 & 7 & 17 & 7 & 7 \\ 10 & 10 & 2 & 10 & 10 & 1 & 3 & 1 & 1 \\ 2 & 8 & 2 & 1 & 8 & 2 & 8 & 2 & 2 \\ 5 & 10 & 5 & 5 & 15 & 5 & 15 & 1 & 5 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 0.8 & 1 & 2 & 3 & 1 & 2 & 3 & 1 & 1 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 2 \end{pmatrix}$$

Example 17:

$$\mu_{j,k} = \begin{pmatrix} 5 & 1 & 0.1 & 1 & 1 & 0.1 & 0.1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 4 \\ 0.2 & 0.2 & 0.2 & 2 & 0.2 & 1 & 0.2 \\ 0.3 & 3 & 0.3 & 3 & 3 & 0.3 & 0.9 \\ 5 & 5 & 7 & 0.5 & 5 & 5 & 0.5 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 0.9 & 0.7 & 1.1 & 0.3 & 0.3 & 0.4 & 0.9 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 1 & 2 & 2 & 3 & 3 & 3 \end{pmatrix}$$

Example 18:

$$\mu_{j,k} = \begin{pmatrix} 19 & 19 & 20 & 19.5 & 20 & 20 & 19.9 \\ 9 & 8.8 & 8.9 & 8.6 & 8.5 & 8.7 & 8.6 \\ 6.5 & 6.9 & 7 & 6.6 & 6.8 & 6.7 & 6.9 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 1.8 & 0.9 & 1.1 & 0.9 & 0.9 & 0.9 & 0.9 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 2 & 2 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Example 19:

$$\mu_{j,k} = \begin{pmatrix} 8 & 0.1 & 2 & 0.1 & 1 & 10 & 0.1 \\ 2 & 10 & 2 & 2 & 2 & 3 & 2 \\ 3 & 3 & 3 & 4 & 3 & 3 & 3 \\ 4 & 1 & 14 & 1 & 4 & 4 & 4 \\ 9 & 3 & 2 & 2 & 3 & 3 & 1 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 1 & 2 & 2 & 0.4 & 3 & 2 & 2 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 2 & 1 & 2 & 2 & 2 & 2 \end{pmatrix}$$

Example 20:

$$\mu_{j,k} = \begin{pmatrix} 10 & 10 & 20 & 10 & 10 \\ 3 & 3 & 3 & 2 & 3.5 \\ 10 & 10 & 10 & 3 & 4.7 \\ 2 & 3 & 4 & 5 & 2 \end{pmatrix}$$

$$a_k = \begin{pmatrix} 3.5 & 4.7 & 0.5 & 1 & 1 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 3 & 3 & 1 & 3 \end{pmatrix}$$

B.5 TDCCP with costs when all classes are identical and Condition 1 is satisfied

Example 1:

$$\mu_{j,k} = \begin{pmatrix} 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$C = 6$$

$$a_k = \begin{pmatrix} 1 & 1 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Example 2:

$$\mu_{j,k} = \begin{pmatrix} 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.8 & 0.8 & 0.8 & 0.8 & 0.8 & 0.8 & 0.8 & 0.8 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.4 & 0.4 & 0.4 & 0.4 & 0.4 & 0.4 & 0.4 & 0.4 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \end{pmatrix}$$

$$C = 4$$

$$a_k = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \end{pmatrix}$$

Example 3:

$$\mu_{j,k} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \end{pmatrix}$$

$$C = 16$$

$$a_k = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

Example 4:

$$\mu_{j,k} = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

$$C = 5$$

$$a_k = \begin{pmatrix} 4 & 4 & 4 & 4 & 4 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \end{pmatrix}$$

Example 5:

$$\mu_{j,k} = \begin{pmatrix} 3 & 3 & 3 & 3 & 3 & 3 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 4 & 4 & 4 & 4 & 4 & 4 \end{pmatrix}$$

$$C = 9$$

$$a_k = \begin{pmatrix} 1.5 & 1.5 & 1.5 & 1.5 & 1.5 & 1.5 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

Example 6:

$$\mu_{j,k} = \begin{pmatrix} 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \end{pmatrix}$$

$$C = 32$$

$$a_k = \begin{pmatrix} 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \end{pmatrix}$$

Example 7:

$$\mu_{j,k} = \begin{pmatrix} 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 \\ 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.4 & 0.4 & 0.4 & 0.4 & 0.4 & 0.4 & 0.4 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 \end{pmatrix}$$

$$C = 2$$

$$a_k = \begin{pmatrix} 0.9 & 0.9 & 0.9 & 0.9 & 0.9 & 0.9 & 0.9 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 5 & 5 & 5 & 5 & 5 & 5 & 5 \end{pmatrix}$$

Example 8:

$$\mu_{j,k} = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \end{pmatrix}$$

$$C = 10$$

$$a_k = \begin{pmatrix} 0.9 & 0.9 & 0.9 & 0.9 & 0.9 & 0.9 & 0.9 & 0.9 & 0.9 & 0.9 \\ 0.9 & 0.9 & 0.9 & 0.9 & 0.9 & 0.9 & 0.9 & 0.9 & 0.9 & 0.9 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Example 9:

$$\mu_{j,k} = \begin{pmatrix} 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 \\ 7 & 7 & 7 & 7 & 7 & 7 & 7 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

$$C = 20$$

$$a_k = \begin{pmatrix} 3 & 3 & 3 & 3 & 3 & 3 & 3 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 4 & 4 & 4 & 4 & 4 & 4 & 4 \end{pmatrix}$$

Example 10:

$$\mu_{j,k} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 \\ 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 \\ 5 & 5 & 5 & 5 & 5 \\ 4 & 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

$$C = 10$$

$$a_k = \begin{pmatrix} 1.5 & 1.5 & 3.5 & 2.5 & 5 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 3 & 3 & 3 & 3 & 3 \end{pmatrix}$$

Example 11:

$$\mu_{j,k} = \begin{pmatrix} 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$C = 5$$

$$a_k = \begin{pmatrix} 4 & 4 & 2 & 2 & 2 & 2 & 2 & 2 & 4 & 4 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Example 12:

$$\mu_{j,k} = \begin{pmatrix} 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.4 & 0.4 & 0.4 & 0.4 & 0.4 & 0.4 & 0.4 & 0.4 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$C = 6$$

$$a_k = \begin{pmatrix} 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 2 & 2 & 3 & 3 & 3 & 3 & 3 \end{pmatrix}$$

Example 13:

$$\mu_{j,k} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \end{pmatrix}$$

$$C = 10$$

$$a_k = \begin{pmatrix} 1.5 & 1.5 & 1.5 & 1.5 & 1.5 & 1.5 & 1.5 & 1.5 & 1.5 & 1.5 & 1.5 & 1.5 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

Example 14:

$$\mu_{j,k} = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

$$C = 5$$

$$a_k = \begin{pmatrix} 2 & 1 & 1 & 2 & 2 & 1 & 0.2 & 0.2 & 0.2 & 0.2 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \end{pmatrix}$$

Example 15:

$$\mu_{j,k} = \begin{pmatrix} 3 & 3 & 3 & 3 & 3 & 3 \\ 2 & 2 & 2 & 2 & 2 & 2 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 4 & 4 & 4 & 4 & 4 & 4 \end{pmatrix}$$

$$C = 8$$

$$a_k = \begin{pmatrix} 1 & 1 & 1.2 & 1.3 & 1.4 & 1.5 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

Example 16:

$$\mu_{j,k} = \begin{pmatrix} 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 \\ 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

$$C = 10$$

$$a_k = \begin{pmatrix} 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 3 & 3 & 3 & 3 & 4 & 4 & 4 & 4 & 4 \end{pmatrix}$$

Example 17:

$$\mu_{j,k} = \begin{pmatrix} 0.7 & 0.7 & 0.7 & 0.7 & 0.7 & 0.7 & 0.7 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 \\ 0.9 & 0.9 & 0.9 & 0.9 & 0.9 & 0.9 & 0.9 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.4 & 0.4 & 0.4 & 0.4 & 0.4 & 0.4 & 0.4 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 \end{pmatrix}$$

$$C = 6$$

$$a_k = \begin{pmatrix} 0.6 & 0.6 & 0.6 & 0.8 & 0.8 & 0.2 & 0.2 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 3 & 3 & 3 & 4 & 4 & 4 & 4 \end{pmatrix}$$

Example 18:

$$\mu_{j,k} = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \\ 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \end{pmatrix}$$

$$C = 10$$

$$a_k = \begin{pmatrix} 0.4 & 0.4 & 0.4 & 0.4 & 0.4 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.2 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

Example 19:

$$\mu_{j,k} = \begin{pmatrix} 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 \\ 7 & 7 & 7 & 7 & 7 & 7 & 7 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

$$C = 11$$

$$a_k = \begin{pmatrix} 3 & 3 & 3 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

Example 20:

$$\mu_{j,k} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 \\ 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 \\ 5 & 5 & 5 & 5 & 5 \\ 4 & 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

$$C = 6$$

$$a_k = \begin{pmatrix} 2 & 2 & 3 & 3 & 3 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 1 & 3 & 3 & 3 \end{pmatrix}$$

B.6 TDCCP with costs when all classes are identical and Condition 2 is satisfied

Example 1:

$$\mu_{j,k} = \begin{pmatrix} 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$C = 6$$

$$a_k = \begin{pmatrix} 0.2 & 0.1 & 0.1 & 0.1 & 0.1 & 0.2 & 0.1 & 0.2 & 0.2 & 0.2 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Example 2:

$$\mu_{j,k} = \begin{pmatrix} 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.8 & 0.8 & 0.8 & 0.8 & 0.8 & 0.8 & 0.8 & 0.8 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.4 & 0.4 & 0.4 & 0.4 & 0.4 & 0.4 & 0.4 & 0.4 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \end{pmatrix}$$

$$C = 4$$

$$a_k = \begin{pmatrix} 0.1 & 0.1 & 0.3 & 0.5 & 0.1 & 0.1 & 0.1 & 0.1 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \end{pmatrix}$$

Example 3:

$$\mu_{j,k} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \end{pmatrix}$$

$$C = 16$$

$$a_k = \begin{pmatrix} 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.1 & 0.1 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

Example 4:

$$\mu_{j,k} = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

$$C = 5$$

$$a_k = \begin{pmatrix} 0.1 & 0.3 & 0.3 & 0.1 & 0.1 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \end{pmatrix}$$

Example 5:

$$\mu_{j,k} = \begin{pmatrix} 3 & 3 & 3 & 3 & 3 & 3 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 4 & 4 & 4 & 4 & 4 & 4 \end{pmatrix}$$

$$C = 9$$

$$a_k = \begin{pmatrix} 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

Example 6:

$$\mu_{j,k} = \begin{pmatrix} 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \end{pmatrix}$$

$$C = 32$$

$$a_k = \begin{pmatrix} 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \end{pmatrix}$$

Example 7:

$$\mu_{j,k} = \begin{pmatrix} 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 \\ 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.4 & 0.4 & 0.4 & 0.4 & 0.4 & 0.4 & 0.4 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 \end{pmatrix}$$

$$C = 2$$

$$a_k = \begin{pmatrix} 0.07 & 0.07 & 0.07 & 0.07 & 0.07 & 0.07 & 0.07 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 5 & 5 & 5 & 5 & 5 & 5 & 5 \end{pmatrix}$$

Example 8:

$$\mu_{j,k} = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \end{pmatrix}$$

$$C = 10$$

$$a_k = \begin{pmatrix} 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Example 9:

$$\mu_{j,k} = \begin{pmatrix} 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 \\ 7 & 7 & 7 & 7 & 7 & 7 & 7 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

$$C = 20$$

$$a_k = \begin{pmatrix} 0.25 & 0.25 & 0.25 & 0.25 & 0.25 & 0.25 & 0.25 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 4 & 4 & 4 & 4 & 4 & 4 & 4 \end{pmatrix}$$

Example 10:

$$\mu_{j,k} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 \\ 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 \\ 5 & 5 & 5 & 5 & 5 \\ 4 & 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

$$C = 10$$

$$a_k = \begin{pmatrix} 0.5 & 0.5 & 0.2 & 0.5 & 0.15 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 3 & 3 & 3 & 3 & 3 \end{pmatrix}$$

Example 11:

$$\mu_{j,k} = \begin{pmatrix} 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$C = 5$$

$$a_k = \begin{pmatrix} 0.4 & 0.4 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.5 & 0.5 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Example 12:

$$\mu_{j,k} = \begin{pmatrix} 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.4 & 0.4 & 0.4 & 0.4 & 0.4 & 0.4 & 0.4 & 0.4 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$C = 6$$

$$a_k = \begin{pmatrix} 0.3 & 0.3 & 0.3 & 0.3 & 0.1 & 0.1 & 0.1 & 0.1 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 2 & 2 & 3 & 3 & 3 & 3 & 3 \end{pmatrix}$$

Example 13:

$$\mu_{j,k} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \end{pmatrix}$$

$$C = 10$$

$$a_k = \begin{pmatrix} 0.4 & 0.4 & 0.4 & 0.4 & 0.4 & 0.4 & 0.4 & 0.4 & 0.4 & 0.4 & 0.4 & 0.4 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

Example 14:

$$\mu_{j,k} = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

$$C = 5$$

$$a_k = \begin{pmatrix} 0.2 & 0.1 & 0.1 & 0.2 & 0.2 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \end{pmatrix}$$

Example 15:

$$\mu_{j,k} = \begin{pmatrix} 3 & 3 & 3 & 3 & 3 & 3 \\ 2 & 2 & 2 & 2 & 2 & 2 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 4 & 4 & 4 & 4 & 4 & 4 \end{pmatrix}$$

$$C = 8$$

$$a_k = \begin{pmatrix} 0.1 & 0.1 & 0.12 & 0.05 & 0.1 & 0.13 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

Example 16:

$$\mu_{j,k} = \begin{pmatrix} 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 \\ 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

$$C = 10$$

$$a_k = \begin{pmatrix} 0.6 & 0.6 & 0.6 & 0.6 & 0.6 & 0.6 & 0.6 & 0.6 & 0.6 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 3 & 3 & 3 & 3 & 4 & 4 & 4 & 4 & 4 \end{pmatrix}$$

Example 17:

$$\mu_{j,k} = \begin{pmatrix} 0.7 & 0.7 & 0.7 & 0.7 & 0.7 & 0.7 & 0.7 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 \\ 0.9 & 0.9 & 0.9 & 0.9 & 0.9 & 0.9 & 0.9 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.4 & 0.4 & 0.4 & 0.4 & 0.4 & 0.4 & 0.4 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 & 0.3 \end{pmatrix}$$

$$C = 6$$

$$a_k = \begin{pmatrix} 0.06 & 0.06 & 0.06 & 0.09 & 0.09 & 0.09 & 0.09 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 3 & 3 & 3 & 4 & 4 & 4 & 4 \end{pmatrix}$$

Example 18:

$$\mu_{j,k} = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \\ 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \end{pmatrix}$$

$$C = 10$$

$$a_k = \begin{pmatrix} 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

Example 19:

$$\mu_{j,k} = \begin{pmatrix} 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 \\ 7 & 7 & 7 & 7 & 7 & 7 & 7 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

$$C = 11$$

$$a_k = \begin{pmatrix} 1 & 1 & 0.1 & 0.5 & 0.5 & 0.5 & 0.5 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

Example 20:

$$\mu_{j,k} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 \\ 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 \\ 5 & 5 & 5 & 5 & 5 \\ 4 & 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

$$C = 6$$

$$a_k = \begin{pmatrix} 0.1 & 0.05 & 0.15 & 0.3 & 0.3 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 1 & 3 & 3 & 3 \end{pmatrix}$$

B.7 TDCCP with costs in the general case when Condition 1 is satisfied

Example 1:

$$\mu_{j,k} = \begin{pmatrix} 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 1 & 0.2 & 1 & 1 & 1 & 1 & 1 & 0.8 & 1 & 1 \\ 0.8 & 1 & 1 & 1 & 0.8 & 1 & 1 & 1 & 1 & 1 \\ 3 & 1 & 2 & 2 & 2 & 2 & 2 & 1.5 & 2 & 2 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 0.5 & 0.5 & 0.5 & 0.5 & 1 & 0.5 & 0.5 & 2 & 0.5 & 0.5 \\ 1 & 2 & 1 & 2 & 1 & 1 & 1 & 1 & 0.5 & 1 \\ 1 & 3 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 0.2 & 2 & 1 & 2 & 2 & 2 & 4 & 3 & 2 \end{pmatrix}$$

$$C = 8$$

$$a_k = \begin{pmatrix} 1 & 1 & 2 & 2 & 2 & 2 & 3 & 3 & 2 & 2 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 1 & 2 & 1 & 1 & 1 & 2 & 1 & 1 & 1 \end{pmatrix}$$

Example 2:

$$\mu_{j,k} = \begin{pmatrix} 0.2 & 0.5 & 0.2 & 0.2 & 0.1 & 0.2 & 0.3 & 0.2 \\ 0.8 & 0.8 & 0.8 & 0.8 & 0.2 & 0.8 & 0.3 & 0.8 \\ 1 & 1 & 4 & 1 & 1 & 1 & 0.5 & 1 \\ 4 & 4 & 2 & 1 & 1 & 4 & 4 & 4 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 0.1 & 0.2 & 0.1 & 0.1 & 0.2 & 0.1 & 0.1 & 0.1 \\ 0.8 & 0.4 & 0.4 & 0.4 & 0.7 & 0.4 & 0.4 & 0.4 \\ 2 & 2 & 3 & 2 & 2 & 1 & 2 & 1 \\ 3 & 10 & 2 & 3 & 1 & 3 & 3 & 3 \end{pmatrix}$$

$$C = 10$$

$$a_k = \begin{pmatrix} 1 & 1 & 2 & 2 & 2 & 2 & 3 & 3 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 3 & 3 & 3 & 1 & 2 & 3 & 3 \end{pmatrix}$$

Example 3:

$$\mu_{j,k} = \begin{pmatrix} 0.8 & 2 & 1 & 0.5 & 1 & 1 & 1 & 1 & 1 & 0.5 & 1 & 1 \\ 0.2 & 1 & 1 & 1 & 1 & 1 & 0.8 & 1 & 1 & 1 & 0.9 & 1 \\ 3 & 3 & 3 & 3 & 3 & 3 & 2 & 3 & 3 & 3 & 4 & 3 \\ 5 & 5 & 5 & 5 & 4 & 1 & 1 & 5 & 5 & 3 & 5 & 5 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 1 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 \\ 2 & 2 & 2 & 3 & 2 & 2 & 2 & 2 & 2 & 3 & 2 & 2 \\ 3 & 1 & 3 & 3 & 3 & 3 & 2 & 3 & 3 & 3 & 3 & 3 \\ 4 & 1 & 4 & 5 & 4 & 4 & 1 & 4 & 4 & 4 & 5 & 4 \end{pmatrix}$$

$$C = 50$$

$$a_k = \begin{pmatrix} 10 & 1 & 10 & 9 & 10 & 8 & 10 & 10 & 10 & 10 & 10 & 10 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 3 & 1 & 2 & 2 & 2 & 2 & 2 & 1 & 3 & 2 & 2 \end{pmatrix}$$

Example 4:

$$\mu_{j,k} = \begin{pmatrix} 2 & 2 & 2 & 2 & 1 & 1 & 2 & 2 & 2 & 2 \\ 2 & 1 & 1 & 2 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0.2 & 1 & 1 \\ 4 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 2 & 3 \\ 5 & 2 & 2 & 0.5 & 2 & 1 & 2 & 2 & 1 & 2 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 1 & 1 & 1 & 3 & 1 & 1 & 2 & 1 & 1 & 1 \\ 2 & 1 & 1 & 1 & 1 & 1 & 1 & 4 & 1 & 1 \\ 1 & 1 & 3 & 1 & 1 & 1 & 1 & 2 & 1 & 1 \\ 1 & 3 & 1 & 1 & 5 & 1 & 1 & 1 & 1 & 1 \\ 1 & 3 & 1 & 1 & 6 & 1 & 1 & 1 & 2 & 1 \end{pmatrix}$$

$$C = 20$$

$$a_k = \begin{pmatrix} 4 & 3 & 3 & 1 & 1 & 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 4 & 5 & 4 & 5 & 4 & 3 & 4 & 2 & 1 & 4 \end{pmatrix}$$

Example 5:

$$\mu_{j,k} = \begin{pmatrix} 3 & 1 & 2 & 2 & 3 & 3 \\ 1 & 2 & 1 & 1 & 1 & 2 \\ 1 & 1 & 1 & 0.5 & 0.5 & 0.5 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 1 & 2 & 2 & 3 & 2 & 2 \\ 1 & 2 & 1 & 2 & 1 & 1 \\ 4 & 3 & 4 & 1 & 4 & 4 \end{pmatrix}$$

$$C = 10$$

$$a_k = \begin{pmatrix} 2 & 2 & 2 & 1.5 & 1.5 & 1.5 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 1 & 2 & 1 & 2 & 2 \end{pmatrix}$$

Example 6:

$$\mu_{j,k} = \begin{pmatrix} 7 & 6.5 & 6.5 & 6.6 & 7 & 6.9 & 7 & 7 & 7 \\ 1.9 & 1.8 & 1.7 & 1.8 & 1.9 & 2 & 1.8 & 1.9 & 1.7 \\ 3.8 & 4 & 3.9 & 3.8 & 3.7 & 3.8 & 3.9 & 3.7 & 3.9 \\ 2.9 & 2.8 & 2.7 & 2.8 & 2.9 & 3 & 2.7 & 2.8 & 2.9 \\ 5 & 4.7 & 4.8 & 5 & 5 & 4.9 & 5 & 5 & 4.9 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 3 & 2 & 2 & 2 & 1 & 2 & 4 & 2 & 2 \\ 1 & 1 & 3 & 1 & 1 & 1 & 1 & 2 & 1 \\ 3 & 2 & 3 & 3 & 2 & 3 & 3 & 5 & 3 \\ 1 & 2 & 3 & 2 & 7 & 2 & 2 & 6 & 2 \\ 6 & 4 & 4 & 5 & 4 & 4 & 4 & 2 & 4 \end{pmatrix}$$

$$C = 100$$

$$a_k = \begin{pmatrix} 8 & 10 & 10 & 14 & 10 & 10 & 16 & 12 & 10 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 4 & 4 & 3 & 4 & 3 & 4 & 4 & 1 & 4 \end{pmatrix}$$

Example 7:

$$\mu_{j,k} = \begin{pmatrix} 0.1 & 0.05 & 0.1 & 0.1 & 1 & 0.2 & 0.1 \\ 0.5 & 0.2 & 0.6 & 0.2 & 0.2 & 0.3 & 0.5 \\ 0.2 & 0.1 & 0.2 & 0.2 & 0.1 & 0.2 & 0.2 \\ 0.3 & 0.2 & 0.3 & 0.9 & 0.2 & 0.1 & 0.3 \\ 0.7 & 0.5 & 0.3 & 0.3 & 0.2 & 0.1 & 0.3 \\ 0.6 & 0.3 & 0.2 & 0.5 & 0.5 & 0.3 & 0.1 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 1 & 0.1 & 0.2 & 0.1 & 0.4 & 0.1 & 0.1 \\ 0.3 & 0.2 & 0.2 & 0.7 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.9 & 0.4 & 0.8 & 0.4 & 0.6 & 0.4 & 0.4 \\ 0.5 & 0.5 & 0.3 & 0.7 & 0.5 & 0.5 & 0.5 \\ 0.2 & 0.3 & 0.3 & 0.3 & 0.1 & 0.3 & 0.3 \end{pmatrix}$$

$$C = 3$$

$$a_k = \begin{pmatrix} 0.9 & 0.8 & 0.9 & 0.9 & 1.2 & 0.9 & 1.8 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 5 & 2 & 1 & 5 & 3 & 5 & 4 \end{pmatrix}$$

Example 8:

$$\mu_{j,k} = \begin{pmatrix} 2 & 1 & 2 & 1 & 2 & 1 & 2 & 2 & 2 & 2 & 1 & 2 & 2 & 2 & 2 & 1 & 1 & 1 & 1 & 1 \\ 4 & 5 & 4 & 4 & 4 & 3 & 4 & 4 & 4 & 3 & 4 & 4 & 4 & 1 & 4 & 4 & 4 & 4 & 4 & 4 \\ 3 & 1 & 3 & 3 & 3 & 2 & 1 & 1 & 1 & 2 & 3 & 1 & 3 & 1 & 3 & 3 & 3 & 3 & 3 & 2 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 5 & 3 & 5 & 3 & 5 & 3 & 5 & 7 & 9 & 5 & 5 & 5 & 5 & 5 & 9 & 5 & 1 & 1 & 3 & 10 \\ 2 & 4 & 2 & 4 & 3 & 4 & 3 & 4 & 4 & 1 & 1 & 1 & 4 & 4 & 2 & 4 & 3 & 4 & 4 & 4 \\ 1 & 3 & 3 & 3 & 2 & 3 & 3 & 3 & 2 & 2 & 3 & 3 & 3 & 2 & 3 & 3 & 2 & 3 & 3 & 3 \end{pmatrix}$$

$$C = 12$$

$$a_k = \begin{pmatrix} 0.9 & 0.7 & 0.9 & 0.9 & 1.5 & 0.9 & 0.9 & 0.9 & 0.9 & 1.3 \\ 0.6 & 0.9 & 0.9 & 0.9 & 0.9 & 0.9 & 0.9 & 0.9 & 0.9 & 1.8 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Example 9:

$$\mu_{j,k} = \begin{pmatrix} 0.2 & 0.1 & 0.1 & 0.1 & 0.2 & 0.2 & 0.1 \\ 2 & 2 & 1 & 1 & 2 & 3 & 2 \\ 1 & 1 & 2 & 3 & 3 & 2 & 3 \\ 4 & 1 & 4 & 2 & 3 & 5 & 4 \\ 7 & 6 & 1 & 4 & 5 & 4 & 7 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 1 & 1 & 8 & 2 & 1 & 1 & 2 \\ 2 & 2 & 3 & 2 & 3 & 2 & 2 \\ 3 & 4 & 3 & 3 & 4 & 5 & 3 \\ 5 & 5 & 9 & 6 & 5 & 2 & 5 \\ 1 & 2 & 2 & 1 & 3 & 1 & 1 \end{pmatrix}$$

$$C = 6$$

$$a_k = \begin{pmatrix} 1 & 8 & 4 & 6 & 0.2 & 1 & 0.2 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 3 & 3 & 2 & 2 & 1 & 4 & 4 \end{pmatrix}$$

Example 10:

$$\mu_{j,k} = \begin{pmatrix} 2 & 1.99 & 1.98 & 2 & 1.96 \\ 3.9 & 3.8 & 4 & 3.9 & 3.8 \\ 5 & 4.9 & 5 & 4.9 & 4.8 \\ 4 & 3.9 & 3.9 & 3.9 & 4 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 1 & 2 & 1 & 4 & 2 \\ 4 & 2 & 1 & 5 & 5 \\ 2 & 3 & 10 & 4 & 4 \\ 8 & 1 & 3 & 2 & 1 \end{pmatrix}$$

$$C = 4$$

$$a_k = \begin{pmatrix} 2 & 8 & 0.6 & 0.4 & 1 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 3 & 3 & 1 & 2 & 2 \end{pmatrix}$$

Example 11:

$$\mu_{j,k} = \begin{pmatrix} 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 1 & 0.2 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 \\ 0.8 & 1 & 1 & 1 & 0.8 & 1 & 1 & 1 & 1 & 1 \\ 3 & 3 & 2 & 2 & 2 & 2 & 2 & 1.5 & 2 & 2 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 0.5 & 0.5 & 0.5 & 0.5 & 1 & 0.5 & 0.5 & 2 & 0.5 & 0.5 \\ 1 & 2 & 3 & 2 & 1 & 1 & 1 & 1 & 0.5 & 1 \\ 1 & 3 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 0.2 & 2 & 1 & 2 & 2 & 2 & 4 & 3 & 2 \end{pmatrix}$$

$$C = 20$$

$$a_k = \begin{pmatrix} 3 & 3 & 3 & 3 & 2 & 2 & 3 & 3 & 3 & 3 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 2 \end{pmatrix}$$

Example 12:

$$\mu_{j,k} = \begin{pmatrix} 0.2 & 0.5 & 0.2 & 0.2 & 0.1 & 0.2 & 0.1 & 0.2 \\ 0.8 & 2 & 0.8 & 0.8 & 1 & 0.8 & 1 & 0.8 \\ 1 & 1 & 4 & 1 & 1 & 1 & 3 & 1 \\ 4 & 4 & 3 & 1 & 7 & 8 & 4 & 4 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 0.1 & 0.2 & 0.1 & 0.1 & 0.2 & 0.1 & 0.1 & 0.1 \\ 0.8 & 0.4 & 0.4 & 0.4 & 0.7 & 0.4 & 0.4 & 0.4 \\ 2 & 2 & 3 & 6 & 2 & 1 & 2 & 1 \\ 3 & 10 & 2 & 7 & 1 & 3 & 3 & 3 \end{pmatrix}$$

$$C = 15$$

$$a_k = \begin{pmatrix} 1 & 1 & 5 & 5 & 5 & 5 & 3 & 3 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 2 & 3 & 3 & 1 & 2 & 1 & 3 \end{pmatrix}$$

Example 13:

$$\mu_{j,k} = \begin{pmatrix} 0.2 & 2 & 1 & 2 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 2 & 1 \\ 3 & 3 & 3 & 3 & 3 & 3 & 2 & 3 & 3 & 3 & 4 & 3 \\ 6 & 5 & 5 & 5 & 4 & 1 & 1 & 5 & 5 & 4 & 5 & 5 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 1 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 \\ 2 & 2 & 2 & 3 & 2 & 2 & 2 & 2 & 2 & 3 & 2 & 2 \\ 3 & 1 & 3 & 6 & 3 & 7 & 2 & 3 & 3 & 3 & 3 & 3 \\ 4 & 1 & 4 & 5 & 4 & 4 & 1 & 4 & 4 & 4 & 5 & 4 \end{pmatrix}$$

$$C = 100$$

$$a_k = \begin{pmatrix} 10 & 1 & 10 & 9 & 8 & 8 & 10 & 10 & 7 & 1 & 2 & 10 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 3 & 1 & 2 & 1 & 1 & 2 & 2 & 1 & 3 & 2 & 2 \end{pmatrix}$$

Example 14:

$$\mu_{j,k} = \begin{pmatrix} 2 & 2 & 2 & 2 & 1 & 2 & 2 & 1 & 2 & 2 \\ 2 & 1 & 1 & 6 & 7 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 \\ 4 & 1 & 3 & 3 & 3 & 3 & 3 & 3 & 2 & 3 \\ 5 & 2 & 2 & 4 & 5 & 1 & 2 & 2 & 1 & 2 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 1 & 1 & 1 & 3 & 1 & 1 & 2 & 1 & 1 & 1 \\ 2 & 1 & 1 & 1 & 1 & 1 & 1 & 4 & 1 & 1 \\ 1 & 1 & 3 & 1 & 1 & 1 & 1 & 2 & 1 & 1 \\ 1 & 3 & 1 & 1 & 5 & 1 & 1 & 1 & 1 & 1 \\ 1 & 3 & 1 & 1 & 6 & 1 & 1 & 1 & 2 & 1 \end{pmatrix}$$

$$C = 5$$

$$a_k = \begin{pmatrix} 4 & 3 & 3 & 4 & 1 & 1 & 1 & 3 & 1 & 2 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 4 & 2 & 4 & 1 & 4 & 3 & 4 & 2 & 1 & 4 \end{pmatrix}$$

Example 15:

$$\mu_{j,k} = \begin{pmatrix} 3 & 2.9 & 2.8 & 2.99 & 3 & 3 \\ 2.8 & 2.9 & 3 & 2.7 & 2.8 & 2.9 \\ 2 & 1.9 & 2 & 1.7 & 1.9 & 1.8 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 1 & 2 & 2 & 3 & 2 & 2 \\ 1 & 2 & 1 & 2 & 1 & 1 \\ 4 & 3 & 1 & 1 & 4 & 4 \end{pmatrix}$$

$$C = 16$$

$$a_k = \begin{pmatrix} 1 & 3 & 3 & 3 & 3 & 1.5 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 1 & 2 & 1 & 1 & 2 \end{pmatrix}$$

Example 16:

$$\mu_{j,k} = \begin{pmatrix} 7 & 6 & 5 & 1 & 7 & 3 & 7 & 7 & 7 \\ 1 & 0.5 & 1 & 1 & 1 & 2 & 5 & 1 & 1 \\ 3 & 4 & 1 & 3 & 3 & 2 & 3 & 3 & 3 \\ 1 & 2 & 1.5 & 2 & 2 & 3 & 2 & 2 & 1 \\ 5 & 3 & 4 & 5 & 5 & 4 & 5 & 5 & 4 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 3 & 2 & 2 & 2 & 1 & 2 & 4 & 2 & 2 \\ 1 & 1 & 3 & 1 & 1 & 1 & 0.1 & 2 & 1 \\ 3 & 2 & 1 & 3 & 2 & 3 & 3 & 5 & 3 \\ 1 & 2 & 3 & 2 & 7 & 2 & 2 & 6 & 2 \\ 2 & 4 & 4 & 5 & 4 & 4 & 3 & 2 & 4 \end{pmatrix}$$

$$C = 70$$

$$a_k = \begin{pmatrix} 7 & 10 & 10 & 13 & 10 & 10 & 17 & 11 & 10 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 4 & 4 & 3 & 4 & 3 & 4 & 4 & 1 & 4 \end{pmatrix}$$

Example 17:

$$\mu_{j,k} = \begin{pmatrix} 0.1 & 0.05 & 0.1 & 0.1 & 0.2 & 0.2 & 0.1 \\ 0.5 & 0.2 & 0.6 & 0.2 & 0.2 & 0.3 & 0.1 \\ 0.2 & 0.1 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.3 & 0.2 & 0.3 & 0.9 & 0.2 & 0.1 & 0.3 \\ 0.7 & 0.5 & 0.3 & 0.3 & 0.2 & 0.3 & 0.3 \\ 0.6 & 0.4 & 0.2 & 0.5 & 0.5 & 0.3 & 0.1 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 1 & 0.1 & 0.2 & 0.1 & 0.4 & 0.1 & 0.1 \\ 0.3 & 0.2 & 0.2 & 0.7 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.9 & 0.4 & 0.8 & 0.4 & 0.6 & 0.4 & 0.4 \\ 0.5 & 0.5 & 0.3 & 0.7 & 0.5 & 0.5 & 0.5 \\ 0.2 & 0.3 & 0.3 & 0.3 & 0.1 & 0.3 & 0.3 \end{pmatrix}$$

$$C = 6$$

$$a_k = \begin{pmatrix} 0.1 & 0.5 & 0.9 & 0.7 & 1.1 & 0.1 & 1.8 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 2 & 1 & 3 & 3 & 5 & 4 \end{pmatrix}$$

Example 18:

$$\mu_{j,k} = \begin{pmatrix} 2 & 1 & 2 & 1 & 2 & 1 & 2 & 2 & 2 & 1 & 1 & 2 & 2 & 2 & 2 & 1 & 1 & 1 & 1 & 2 \\ 4 & 5 & 4 & 4 & 4 & 1 & 4 & 4 & 4 & 3 & 4 & 4 & 4 & 2 & 4 & 4 & 4 & 4 & 4 & 4 \\ 3 & 1 & 4 & 3 & 3 & 2 & 1 & 3 & 3 & 2 & 3 & 1 & 3 & 1 & 3 & 3 & 3 & 3 & 3 & 1 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 5 & 3 & 5 & 3 & 5 & 3 & 5 & 7 & 9 & 5 & 5 & 5 & 2 & 5 & 3 & 5 & 1 & 1 & 3 & 10 \\ 2 & 4 & 0 & 4 & 3 & 4 & 3 & 4 & 4 & 1 & 1 & 1 & 4 & 4 & 2 & 4 & 3 & 4 & 4 & 4 \\ 1 & 3 & 3 & 3 & 2 & 3 & 3 & 3 & 2 & 1 & 3 & 3 & 3 & 2 & 3 & 3 & 2 & 3 & 3 & 3 \end{pmatrix}$$

$$C = 30$$

$$a_k = \begin{pmatrix} 0.9 & 0.1 & 0.9 & 0.9 & 1.5 & 0.3 & 0.9 & 0.7 & 0.9 & 1.3 \\ 0.6 & 0.9 & 0.9 & 0.9 & 0.9 & 0.2 & 0.9 & 0.5 & 0.3 & 1.8 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 2 & 1 & 2 & 1 & 2 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Example 19:

$$\mu_{j,k} = \begin{pmatrix} 0.7 & 0.1 & 0.1 & 0.1 & 0.2 & 0.6 & 0.1 \\ 2 & 2 & 1 & 1 & 2 & 3 & 2 \\ 1 & 1 & 3 & 4 & 3 & 2 & 3 \\ 4 & 1 & 4 & 3 & 3 & 5 & 4 \\ 7 & 6 & 2 & 2 & 3 & 4 & 7 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 1 & 1 & 8 & 2 & 1 & 1 & 1 \\ 2 & 2 & 3 & 2 & 3 & 2 & 2 \\ 3 & 4 & 9 & 3 & 4 & 3 & 3 \\ 5 & 5 & 9 & 1 & 5 & 2 & 5 \\ 1 & 2 & 1 & 1 & 3 & 1 & 1 \end{pmatrix}$$

$$C = 11$$

$$a_k = \begin{pmatrix} 1 & 3 & 2 & 8 & 2 & 12 & 3 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 3 & 3 & 2 & 2 & 1 & 4 & 4 \end{pmatrix}$$

Example 20:

$$\mu_{j,k} = \begin{pmatrix} 2 & 2 & 1 & 2 & 1 \\ 1 & 3 & 3 & 2 & 3 \\ 6 & 2 & 1 & 5 & 1 \\ 1 & 3 & 2 & 2 & 4 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 1 & 2 & 3 & 4 & 2 \\ 4 & 2 & 1 & 2 & 5 \\ 2 & 3 & 10 & 4 & 4 \\ 8 & 7 & 3 & 2 & 1 \end{pmatrix}$$

$$C = 15$$

$$a_k = \begin{pmatrix} 7 & 1 & 9 & 2 & 10 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 3 & 3 & 1 & 2 & 2 \end{pmatrix}$$

B.8 TDCCP with costs in the general case when Condition 2 is satisfied

Example 1:

$$\mu_{j,k} = \begin{pmatrix} 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 1 & 0.2 & 1 & 1 & 1 & 1 & 1 & 0.8 & 1 & 1 \\ 0.8 & 1 & 1 & 1 & 0.8 & 1 & 1 & 1 & 1 & 1 \\ 3 & 1 & 2 & 2 & 2 & 2 & 2 & 1.5 & 2 & 2 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 0.5 & 0.5 & 0.5 & 0.5 & 1 & 0.5 & 0.5 & 2 & 0.5 & 0.5 \\ 1 & 2 & 1 & 2 & 1 & 1 & 1 & 1 & 0.5 & 1 \\ 1 & 3 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 0.2 & 2 & 1 & 2 & 2 & 2 & 4 & 3 & 2 \end{pmatrix}$$

$$C = 8$$

$$a_k = \begin{pmatrix} 0.1 & 0.1 & 0.1 & 0.1 & 0.3 & 0.2 & 0.1 & 0.1 & 0.2 & 0.2 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 1 & 2 & 1 & 1 & 1 & 2 & 1 & 1 & 1 \end{pmatrix}$$

Example 2:

$$\mu_{j,k} = \begin{pmatrix} 0.2 & 0.5 & 0.2 & 0.2 & 0.1 & 0.2 & 0.3 & 0.2 \\ 0.8 & 0.8 & 0.8 & 0.8 & 0.2 & 0.8 & 0.3 & 0.8 \\ 1 & 1 & 4 & 1 & 1 & 1 & 0.5 & 1 \\ 4 & 4 & 2 & 1 & 1 & 4 & 4 & 4 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 0.1 & 0.2 & 0.1 & 0.1 & 0.2 & 0.1 & 0.1 & 0.1 \\ 0.8 & 0.4 & 0.4 & 0.4 & 0.7 & 0.4 & 0.4 & 0.4 \\ 2 & 2 & 3 & 2 & 2 & 1 & 2 & 1 \\ 3 & 10 & 2 & 3 & 1 & 3 & 3 & 3 \end{pmatrix}$$

$$C = 10$$

$$a_k = \begin{pmatrix} 0.1 & 0.1 & 0.2 & 0.2 & 0.1 & 0.1 & 0.2 & 0.2 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 3 & 3 & 3 & 1 & 2 & 3 & 3 \end{pmatrix}$$

Example 3:

$$\mu_{j,k} = \begin{pmatrix} 1.8 & 2 & 1.9 & 1.8 & 2 & 2 & 2 & 1.9 & 1.9 & 1.8 & 2 & 2 \\ 0.9 & 1 & 1 & 1 & 1 & 1 & 0.9 & 1 & 1 & 1 & 0.9 & 1 \\ 3.8 & 3.9 & 3.7 & 3.6 & 3.8 & 3.9 & 4 & 4 & 3.8 & 3.9 & 4 & 3.9 \\ 5 & 5 & 5 & 5 & 4.8 & 4.9 & 4.9 & 5 & 5 & 5 & 5 & 5 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 1 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 \\ 2 & 2 & 2 & 3 & 2 & 2 & 2 & 2 & 2 & 3 & 2 & 2 \\ 3 & 1 & 3 & 3 & 3 & 3 & 2 & 3 & 3 & 3 & 3 & 3 \\ 4 & 1 & 4 & 5 & 4 & 4 & 1 & 4 & 4 & 4 & 5 & 4 \end{pmatrix}$$

$$C = 50$$

$$a_k = \begin{pmatrix} 0.25 & 0.25 & 0.25 & 0.25 & 0.25 & 0.25 & 0.25 & 0.25 & 0.25 & 0.25 & 0.25 & 0.25 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 3 & 1 & 2 & 2 & 2 & 2 & 2 & 1 & 3 & 2 & 2 \end{pmatrix}$$

Example 4:

$$\mu_{j,k} = \begin{pmatrix} 2 & 2 & 2 & 2 & 1 & 1 & 2 & 2 & 2 & 2 \\ 2 & 1 & 1 & 2 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0.2 & 1 & 1 \\ 4 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 2 & 3 \\ 5 & 2 & 2 & 0.5 & 2 & 1 & 2 & 2 & 1 & 2 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 1 & 1 & 1 & 3 & 1 & 1 & 2 & 1 & 1 & 1 \\ 2 & 1 & 1 & 1 & 1 & 1 & 1 & 4 & 1 & 1 \\ 1 & 1 & 3 & 1 & 1 & 1 & 1 & 2 & 1 & 1 \\ 1 & 3 & 1 & 1 & 5 & 1 & 1 & 1 & 1 & 1 \\ 1 & 3 & 1 & 1 & 6 & 1 & 1 & 1 & 2 & 1 \end{pmatrix}$$

$$C = 20$$

$$a_k = \begin{pmatrix} 0.4 & 0.3 & 0.3 & 0.1 & 0.1 & 0.1 & 0.1 & 0.2 & 0.2 & 0.2 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 4 & 5 & 4 & 5 & 4 & 3 & 4 & 2 & 1 & 4 \end{pmatrix}$$

Example 5:

$$\mu_{j,k} = \begin{pmatrix} 3 & 2.9 & 2.8 & 2.9 & 3 & 3 \\ 1.9 & 2 & 1.9 & 1.9 & 1.9 & 2 \\ 1 & 1 & 1 & 0.97 & 0.99 & 0.98 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 1 & 2 & 2 & 3 & 2 & 2 \\ 1 & 2 & 1 & 2 & 1 & 1 \\ 4 & 3 & 4 & 1 & 4 & 4 \end{pmatrix}$$

$$C = 10$$

$$a_k = \begin{pmatrix} 0.2 & 0.1 & 0.2 & 0.5 & 0.1 & 0.1 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 1 & 2 & 1 & 2 & 2 \end{pmatrix}$$

Example 6:

$$\mu_{j,k} = \begin{pmatrix} 7 & 6.5 & 6.5 & 0.6 & 7 & 6.9 & 7 & 7 & 7 \\ 1.9 & 1.8 & 1.7 & 1.8 & 1.9 & 2 & 1.8 & 1.9 & 1.7 \\ 3.8 & 4 & 3.9 & 3.8 & 3.7 & 3.8 & 3.9 & 3.7 & 3.9 \\ 2.9 & 2.8 & 2.7 & 1 & 2.9 & 3 & 2.7 & 2.8 & 2.9 \\ 5 & 1 & 1 & 5 & 5 & 1 & 5 & 5 & 4.9 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 3 & 2 & 2 & 2 & 1 & 2 & 4 & 2 & 2 \\ 1 & 1 & 3 & 1 & 1 & 1 & 1 & 2 & 1 \\ 3 & 2 & 3 & 3 & 2 & 3 & 3 & 5 & 3 \\ 1 & 2 & 3 & 2 & 7 & 2 & 2 & 6 & 2 \\ 6 & 4 & 4 & 5 & 4 & 4 & 4 & 2 & 4 \end{pmatrix}$$

$$C = 100$$

$$a_k = \begin{pmatrix} 0.4 & 0.4 & 0.4 & 0.4 & 0.4 & 0.4 & 0.4 & 0.4 & 0.4 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 4 & 4 & 3 & 4 & 3 & 4 & 4 & 1 & 4 \end{pmatrix}$$

Example 7:

$$\mu_{j,k} = \begin{pmatrix} 0.1 & 0.1 & 0.1 & 0.1 & 1 & 0.2 & 0.1 \\ 0.5 & 0.2 & 0.6 & 0.2 & 0.2 & 0.3 & 0.5 \\ 0.2 & 0.1 & 0.2 & 0.2 & 0.1 & 0.2 & 0.2 \\ 0.3 & 0.2 & 0.3 & 0.9 & 0.2 & 0.1 & 0.3 \\ 0.7 & 0.5 & 0.3 & 0.3 & 0.2 & 0.1 & 0.3 \\ 0.6 & 0.3 & 0.2 & 0.5 & 0.5 & 0.3 & 0.1 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 1 & 0.1 & 0.2 & 0.1 & 0.4 & 0.1 & 0.1 \\ 0.3 & 0.2 & 0.2 & 0.7 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.9 & 0.4 & 0.8 & 0.4 & 0.6 & 0.4 & 0.4 \\ 0.5 & 0.5 & 0.3 & 0.7 & 0.5 & 0.5 & 0.5 \\ 0.2 & 0.3 & 0.3 & 0.3 & 0.1 & 0.3 & 0.3 \end{pmatrix}$$

$$C = 3$$

$$a_k = \begin{pmatrix} 0.06 & 0.06 & 0.06 & 0.06 & 0.06 & 0.06 & 0.06 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 5 & 2 & 1 & 5 & 3 & 5 & 4 \end{pmatrix}$$

Example 8:

$$\mu_{j,k} = \begin{pmatrix} 2 & 1 & 2 & 1 & 2 & 1 & 2 & 2 & 2 & 2 & 1 & 2 & 2 & 2 & 2 & 1 & 1 & 1 & 1 & 1 \\ 4 & 5 & 4 & 4 & 4 & 3 & 4 & 4 & 4 & 3 & 4 & 4 & 4 & 1 & 4 & 4 & 4 & 4 & 4 & 4 \\ 3 & 1 & 3 & 3 & 3 & 2 & 1 & 1 & 1 & 2 & 3 & 1 & 3 & 1 & 3 & 3 & 3 & 3 & 3 & 2 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 5 & 3 & 5 & 3 & 5 & 3 & 5 & 7 & 9 & 5 & 5 & 5 & 5 & 5 & 9 & 5 & 1 & 1 & 3 & 10 \\ 2 & 4 & 2 & 4 & 3 & 4 & 3 & 4 & 4 & 1 & 1 & 1 & 4 & 4 & 2 & 4 & 3 & 4 & 4 & 4 \\ 1 & 3 & 3 & 3 & 2 & 3 & 3 & 3 & 2 & 2 & 3 & 3 & 3 & 2 & 3 & 3 & 2 & 3 & 3 & 3 \end{pmatrix}$$

$$C = 12$$

$$a_k = \begin{pmatrix} 0.05 & 0.05 & 0.05 & 0.1 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.1 \\ 0.1 & 0.05 & 0.1 & 0.1 & 0.1 & 0.2 & 0.2 & 0.1 & 0.1 & 0.3 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 2 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Example 9:

$$\mu_{j,k} = \begin{pmatrix} 0.2 & 0.1 & 0.1 & 0.1 & 0.2 & 0.2 & 0.1 \\ 2 & 2 & 1 & 1 & 2 & 3 & 2 \\ 1 & 1 & 2 & 3 & 3 & 2 & 3 \\ 4 & 1 & 4 & 2 & 3 & 5 & 4 \\ 7 & 6 & 1 & 4 & 5 & 4 & 7 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 1 & 1 & 8 & 2 & 1 & 1 & 2 \\ 2 & 2 & 3 & 2 & 3 & 2 & 2 \\ 3 & 4 & 3 & 3 & 4 & 5 & 3 \\ 5 & 5 & 9 & 6 & 5 & 2 & 5 \\ 1 & 2 & 2 & 1 & 3 & 1 & 1 \end{pmatrix}$$

$$C = 9$$

$$a_k = \begin{pmatrix} 0.4 & 0.4 & 0.4 & 0.4 & 0.4 & 0.4 & 0.4 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 3 & 3 & 2 & 2 & 1 & 4 & 4 \end{pmatrix}$$

Example 10:

$$\mu_{j,k} = \begin{pmatrix} 2 & 1 & 1.98 & 2 & 1.96 \\ 3.9 & 0.8 & 4 & 3.9 & 3.8 \\ 5 & 4.9 & 5 & 4.9 & 4.8 \\ 4 & 3.9 & 3.9 & 3.9 & 4 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 1 & 2 & 1 & 4 & 2 \\ 4 & 2 & 1 & 5 & 5 \\ 2 & 3 & 10 & 4 & 4 \\ 8 & 1 & 3 & 2 & 1 \end{pmatrix}$$

$$C = 4$$

$$a_k = \begin{pmatrix} 0.2 & 0.8 & 0.05 & 0.4 & 0.05 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 3 & 3 & 1 & 2 & 2 \end{pmatrix}$$

Example 11:

$$\mu_{j,k} = \begin{pmatrix} 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 1 & 0.2 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 \\ 0.8 & 1 & 1 & 1 & 0.8 & 1 & 1 & 1 & 1 & 1 \\ 3 & 3 & 2 & 2 & 2 & 2 & 2 & 1.5 & 2 & 2 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 0.5 & 0.5 & 0.5 & 0.5 & 1 & 0.5 & 0.5 & 2 & 0.5 & 0.5 \\ 1 & 2 & 3 & 2 & 1 & 1 & 1 & 1 & 0.5 & 1 \\ 1 & 3 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 0.2 & 2 & 1 & 2 & 2 & 2 & 4 & 3 & 2 \end{pmatrix}$$

$$C = 20$$

$$a_k = \begin{pmatrix} 0.1 & 0.1 & 0.1 & 0.05 & 0.05 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 2 \end{pmatrix}$$

Example 12:

$$\mu_{j,k} = \begin{pmatrix} 0.49 & 0.5 & 0.49 & 0.48 & 0.49 & 0.48 & 0.49 & 0.48 \\ 1.8 & 2 & 1.8 & 1.8 & 1.9 & 1.8 & 1.95 & 1.8 \\ 3.9 & 3.8 & 4 & 4 & 4 & 3.7 & 3.9 & 4 \\ 7.8 & 7.8 & 8 & 7.9 & 7.9 & 8 & 8 & 7.8 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 0.1 & 0.2 & 0.1 & 0.1 & 0.2 & 0.1 & 0.1 & 0.1 \\ 0.8 & 0.4 & 0.4 & 0.4 & 0.7 & 0.4 & 0.4 & 0.4 \\ 2 & 2 & 3 & 6 & 2 & 1 & 2 & 1 \\ 3 & 10 & 2 & 7 & 1 & 3 & 3 & 3 \end{pmatrix}$$

$$C = 15$$

$$a_k = \begin{pmatrix} 0.1 & 0.1 & 0.15 & 0.5 & 0.5 & 0.5 & 0.3 & 0.1 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 2 & 3 & 3 & 1 & 2 & 1 & 3 \end{pmatrix}$$

Example 13:

$$\mu_{j,k} = \begin{pmatrix} 0.2 & 2 & 1 & 2 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 2 & 1 \\ 3 & 3 & 3 & 3 & 3 & 3 & 2 & 3 & 3 & 3 & 4 & 3 \\ 6 & 5 & 5 & 5 & 4 & 1 & 1 & 5 & 5 & 4 & 5 & 5 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 1 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 \\ 2 & 2 & 2 & 3 & 2 & 2 & 2 & 2 & 2 & 3 & 2 & 2 \\ 3 & 1 & 3 & 6 & 3 & 7 & 2 & 3 & 3 & 3 & 3 & 3 \\ 4 & 1 & 4 & 5 & 4 & 4 & 1 & 4 & 4 & 4 & 5 & 4 \end{pmatrix}$$

$$C = 100$$

$$a_k = \begin{pmatrix} 1 & 0.1 & 1 & 0.15 & 0.8 & 0.1 & 0.1 & 0.05 & 0.7 & 0.1 & 0.1 & 0.2 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 3 & 1 & 2 & 1 & 1 & 2 & 2 & 1 & 3 & 2 & 2 \end{pmatrix}$$

Example 14:

$$\mu_{j,k} = \begin{pmatrix} 2 & 2 & 2 & 2 & 1.9 & 2 & 2 & 1.9 & 2 & 2 \\ 6.8 & 6.9 & 6.8 & 6.9 & 7 & 7 & 6.9 & 6.9 & 6.8 & 7 \\ 1.9 & 1.8 & 2 & 2 & 2 & 1.9 & 2 & 2 & 1.9 & 1.8 \\ 4 & 4 & 3.7 & 3.8 & 3.95 & 3.92 & 3.75 & 3.8 & 4 & 4 \\ 5 & 2 & 5 & 4.9 & 5 & 4.8 & 5 & 5 & 4.9 & 5 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 1 & 1 & 1 & 3 & 1 & 1 & 2 & 1 & 1 & 1 \\ 2 & 1 & 1 & 1 & 1 & 1 & 1 & 4 & 1 & 1 \\ 1 & 1 & 3 & 1 & 1 & 1 & 1 & 2 & 1 & 1 \\ 1 & 3 & 1 & 1 & 5 & 1 & 1 & 1 & 1 & 1 \\ 1 & 3 & 1 & 1 & 6 & 1 & 1 & 1 & 2 & 1 \end{pmatrix}$$

$$C = 5$$

$$a_k = \begin{pmatrix} 0.4 & 0.3 & 0.3 & 0.1 & 0.1 & 0.1 & 0.1 & 0.3 & 0.1 & 0.2 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 4 & 2 & 4 & 1 & 4 & 3 & 4 & 2 & 1 & 4 \end{pmatrix}$$

Example 15:

$$\mu_{j,k} = \begin{pmatrix} 3 & 1 & 2.8 & 2.99 & 3 & 3 \\ 2.8 & 2.9 & 3 & 0.5 & 2.8 & 2.9 \\ 2 & 1.9 & 2 & 1.7 & 1.9 & 1.8 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 1 & 2 & 2 & 3 & 2 & 2 \\ 1 & 2 & 1 & 2 & 1 & 1 \\ 4 & 3 & 1 & 1 & 4 & 4 \end{pmatrix}$$

$$C = 16$$

$$a_k = \begin{pmatrix} 0.1 & 0.25 & 0.25 & 0.3 & 0.1 & 0.1 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 2 & 1 & 2 & 1 & 1 & 2 \end{pmatrix}$$

Example 16:

$$\mu_{j,k} = \begin{pmatrix} 7 & 6 & 5 & 1 & 7 & 3 & 7 & 7 & 7 \\ 1 & 0.5 & 1 & 1 & 1 & 2 & 5 & 1 & 1 \\ 3 & 4 & 1 & 3 & 3 & 2 & 3 & 3 & 3 \\ 1 & 2 & 1.5 & 2 & 2 & 3 & 2 & 2 & 1 \\ 5 & 3 & 4 & 5 & 5 & 4 & 5 & 5 & 4 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 3 & 2 & 2 & 2 & 1 & 2 & 4 & 2 & 2 \\ 1 & 1 & 3 & 1 & 1 & 1 & 0.1 & 2 & 1 \\ 3 & 2 & 1 & 3 & 2 & 3 & 3 & 5 & 3 \\ 1 & 2 & 3 & 2 & 7 & 2 & 2 & 6 & 2 \\ 2 & 4 & 4 & 5 & 4 & 4 & 3 & 2 & 4 \end{pmatrix}$$

$$C = 70$$

$$a_k = \begin{pmatrix} 0.7 & 0.5 & 0.9 & 0.1 & 0.3 & 0.3 & 0.2 & 0.1 & 0.1 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 4 & 4 & 3 & 4 & 3 & 4 & 4 & 1 & 4 \end{pmatrix}$$

Example 17:

$$\mu_{j,k} = \begin{pmatrix} 0.1 & 0.05 & 0.1 & 0.1 & 0.2 & 0.2 & 0.1 \\ 0.5 & 0.2 & 0.6 & 0.2 & 0.2 & 0.3 & 0.1 \\ 0.2 & 0.1 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.3 & 0.2 & 0.3 & 0.9 & 0.2 & 0.1 & 0.3 \\ 0.7 & 0.5 & 0.3 & 0.3 & 0.2 & 0.3 & 0.3 \\ 0.6 & 0.4 & 0.2 & 0.5 & 0.5 & 0.3 & 0.1 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 1 & 0.1 & 0.2 & 0.1 & 0.4 & 0.1 & 0.1 \\ 0.3 & 0.2 & 0.2 & 0.7 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.9 & 0.4 & 0.8 & 0.4 & 0.6 & 0.4 & 0.4 \\ 0.5 & 0.5 & 0.3 & 0.7 & 0.5 & 0.5 & 0.5 \\ 0.2 & 0.3 & 0.3 & 0.3 & 0.1 & 0.3 & 0.3 \end{pmatrix}$$

$$C = 6$$

$$a_k = \begin{pmatrix} 0.01 & 0.05 & 0.09 & 0.07 & 0.5 & 0.01 & 0.1 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 2 & 1 & 3 & 3 & 5 & 4 \end{pmatrix}$$

Example 18:

$$\mu_{j,k} = \begin{pmatrix} 2 & 1 & 2 & 1 & 2 & 1 & 2 & 2 & 2 & 1 & 1 & 2 & 2 & 2 & 2 & 1 & 1 & 1 & 1 & 2 \\ 4 & 5 & 4 & 4 & 4 & 1 & 4 & 4 & 4 & 3 & 4 & 4 & 4 & 2 & 4 & 4 & 4 & 4 & 4 & 4 \\ 3 & 1 & 4 & 3 & 3 & 2 & 1 & 3 & 3 & 2 & 3 & 1 & 3 & 1 & 3 & 3 & 3 & 3 & 3 & 1 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 5 & 3 & 5 & 3 & 5 & 3 & 5 & 7 & 9 & 5 & 5 & 5 & 2 & 5 & 3 & 5 & 1 & 1 & 3 & 10 \\ 2 & 4 & 0 & 4 & 3 & 4 & 3 & 4 & 4 & 1 & 1 & 1 & 4 & 4 & 2 & 4 & 3 & 4 & 4 & 4 \\ 1 & 3 & 3 & 3 & 2 & 3 & 3 & 3 & 2 & 1 & 3 & 3 & 3 & 2 & 3 & 3 & 2 & 3 & 3 & 3 \end{pmatrix}$$

$$C = 30$$

$$a_k = \begin{pmatrix} 0.09 & 0.01 & 0.09 & 0.09 & 0.5 & 0.03 & 0.09 & 0.07 & 0.09 & 0.3 \\ 0.06 & 0.01 & 0.01 & 0.01 & 0.01 & 0.02 & 0.09 & 0.05 & 0.03 & 0.08 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 1 & 2 & 1 & 2 & 1 & 2 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Example 19:

$$\mu_{j,k} = \begin{pmatrix} 0.7 & 0.1 & 0.1 & 0.1 & 0.2 & 0.6 & 0.1 \\ 2 & 2 & 1 & 1 & 2 & 3 & 2 \\ 1 & 1 & 3 & 4 & 3 & 2 & 3 \\ 4 & 1 & 4 & 3 & 3 & 5 & 4 \\ 7 & 6 & 2 & 2 & 3 & 4 & 7 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 1 & 1 & 8 & 2 & 1 & 1 & 1 \\ 2 & 2 & 3 & 2 & 3 & 2 & 2 \\ 3 & 4 & 9 & 3 & 4 & 3 & 3 \\ 5 & 5 & 9 & 1 & 5 & 2 & 5 \\ 1 & 2 & 1 & 1 & 3 & 1 & 1 \end{pmatrix}$$

$$C = 11$$

$$a_k = \begin{pmatrix} 0.1 & 0.3 & 0.3 & 0.8 & 0.2 & 0.9 & 0.3 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 3 & 3 & 2 & 2 & 1 & 4 & 4 \end{pmatrix}$$

Example 20:

$$\mu_{j,k} = \begin{pmatrix} 2 & 2 & 1 & 2 & 1 \\ 1 & 3 & 3 & 2 & 3 \\ 6 & 2 & 1 & 5 & 1 \\ 1 & 3 & 2 & 2 & 4 \end{pmatrix}$$

$$r_{j,k} = \begin{pmatrix} 1 & 2 & 3 & 4 & 2 \\ 4 & 2 & 1 & 2 & 5 \\ 2 & 3 & 10 & 4 & 4 \\ 8 & 7 & 3 & 2 & 1 \end{pmatrix}$$

$$C = 15$$

$$a_k = \begin{pmatrix} 0.7 & 0.1 & 0.8 & 0.2 & 1.1 \end{pmatrix}$$

$$c_k = \begin{pmatrix} 3 & 3 & 1 & 2 & 2 \end{pmatrix}$$