Cross-Docking

JIN.BIN

May 5, 2006

Abstract

Cross-docking techniques are applied universally across a wide range of businesses working in many market sectors. This dissertation provides a solution method of solving the Cross-Docking problem to optimality by applying the Column Generation technique. The feature and computation complexity of the Cross-Docking problem are discussed in details. The master problem and pricing subproblem are set up according to the column generation structure. Three algorithms are developed to solve the subproblem from a closed optimal value to the optimality, in which brings the possibility of solving the Cross-Docking problem to optimality. The issues of implementation techniques are discussed including dertermining a start point, adding new columns, stop criteria and branching and bounding. The testing result shows that the application of column generation in solving the Cross-Docking problem is successful and promising. We also make a further study of how the time window constraint and capacity constraint affect the Cross-Docking problem.

Acknowledgements

The author benefited from fruitful discussions with Larsen Jesper . I am grateful to my supervisor, Larsen Jesper, for his experienced suggestion and helpful comments on Column Generation technolodgy, both in theory and in practical implementation. His constant support and encouragement inspired me to carry on this project through to the very last ending. And I gained great interest in every step forward during the project.

I also appricaited Clausen Jens for his help in dual problem discussion and Thomas K. Stidsen for his support in development environment setup.

Last, but not least, I would like to thank my friends, Hua Wang, TianShun Ye. They helped me getting through using the weird LaTex in writting the report.

Contents

Chapter 1

Introduction

Switching from a traditional stockholding supply chain system to a cross-docking system. Cross-docking techniques are applied universally across a wide range of businesses working in many market sectors. The basic business motivations for utilizing cross-docking techniques include increased stock flow, reduced stockholding, improved resource utilization and reduced delivery lead times.

A simple definition of cross-docking is as followings: receiving product from a supplier or manufacturer for several end destination and consolidating this product for common final delivery destinations. The key to the process is transshipping , not holding stick. Equally important is the process of turning expensive delivery consignments into economic loads through consolidation and resource sharing. For many businesses it is essential to keep track of product consignments as they progress along the supply chain. An increasingly topical theme for many bussiness is manipulating product into a user-friendly form for the end user.

The key benifits results from the adoption of cross-docking techniques relate to improvements in service levels, inventory levels, stocking returns and unit costs.

In our dissertation, the cross-docking problem is to deliver each customer's order from the pickup node to the delivery node through cross-docking at the depot. The objective is to achieve a minimal total time consuming of all the deliverys. In addition to find the shortest ways of delivery orders, the reduced time cost by improved resource utilization should also bring to consideration. Obviously, the cross-docking problem is an NP-hard problem.

Dantzig-Wolfe decomposition and column generation, devised for linear programs, is a success story in large scale integer programing. Column generation is nowadays a prominent method to cope with a huge number of variables. The embedding of column generation techniques within a linear programing based branch-and-bound framework was the key step in the design of exact algorithms for a large class of integer programs.

We apply column generation technique in solving the cross-docking problem to optimality. We merge promising contemporaty research works with more classical solution strategies in the implementation process. The key in solving the Cross-Docking problem to optimality is to find the shortest path with consideration of saving loading effort. Three algorithms are developed to solve the subproblem from closed optimal value to optimality. NDCA and CDSPA are to find the closed optimal value of single path or combination path, respectively. OCDSPA is to solve the subproblem into optimality. Issues of implementation to solve the master problem are tailored and discussed including selecting an initial start point, adding columns into the master problem, stop criteria and branch-and-bound.

This disseration is organized into five main parts.

1. Problem and analysis. Section2 gives the details of the problem description; section3 analyses the feature of the Cross-Docking problem and makes comparation with other VRP.

2. Column generation technique and modeling of the Cross-Docking problem. Section4 gives an overview of column generation technique; section5 builds an original model of the Cross-Docking problem and sets up the master problem and the pricing subproblem.

3. Solving the subproblem. Section6 to 10 are dealing with the subproblem. Section6 introduces the GPLA which is a foundation of development of the three new algorithms for the subproblem; section6 and 8 describ the NDCA and its adaption to the subproblem; section9 and 10 describ the CDSPA and the OCDSPA algorithm.

4. Implementation. Section11 is the implementation of the three algorithms of the subproblem; section12 is the implementation of the column generation technique in solving the master problem.

5. Numeric result and conclusion. Section13 is the numeric result and analysis; section14 is a conclusion.

Chapter 2

Problem Description

The Cross-Docking Problem is issued from Transvision and Jakob Birkedal Nielsen.

We have a set of pickup-and-delivery orders, that is, each order can be characterized by a place where we have to pickup the cargo and another place where it has to be delivered. For each order we furthermore have the size measured in number of pallets. Now the orders can not be driven directly from the pickup place to the delivery place. They first have to go to a central distribution facility denoted the depot. A node is either the depot, a pickup place or a delivery place.

At the depot we have a fleet of vehicles that leave at 6am in the morning and has to be back at latest at 10pm in the evening. All trucks are identical wrt. capacity. So each of the orders have to be transported from the pickup place to the depot and then from the depot to the delivery place. In order to complicate matters time windows on the customers can be added but this is not part of the initial problem. It is allowed for a vehicle first to collect orders drive to the depot offload some but not all orders, onload additional orders and drive out again on a "delivery route".

For the problem we have realistic data generated based on a Danish scenario. For simplicity we assume use Euclidean distances, and assume that the trucks can keep a constant speed of 60 km/h on the road. No order must be left overnight in the depot.

For each node it takes 10 minutes to dock the truck and a further 1 min for each pallet that needs to be on- or off-loaded. Our aim is to minimize the total time needed to deliver the orders. Additions to the objective like minimize the number of vehicles and equal load sharing can be discussed in a more elaborate project.

Figure 2.1 show a cross-docking problem instance. We have seven customers

A, B, C, D, E, F and G. O is the deport. Each customer has one pickup node p_i and one delivery node d_i. The task is to deliver every customer's order from p_i to d_i. In figure 2.1 the left part contains two pickup routes, the right part contains three delivery routes. A, B, C, D and E, F, G are first pickuped up by pickup route seperately and transported back to the deport. Some orders are offloaded and uploaded into different vehicles, others stay on the same vehicles. After reloading process at the deport, the seven orders are grouped into three delivering routes, A and G, B and D, C, E and F, and delivered to their delivery nodes.

If we call a route start from depot and end at depot as subroute, figure 2.1 also shows that each subroute only contains one type of node, pickup node and delivery node are never visited by a same subroute.

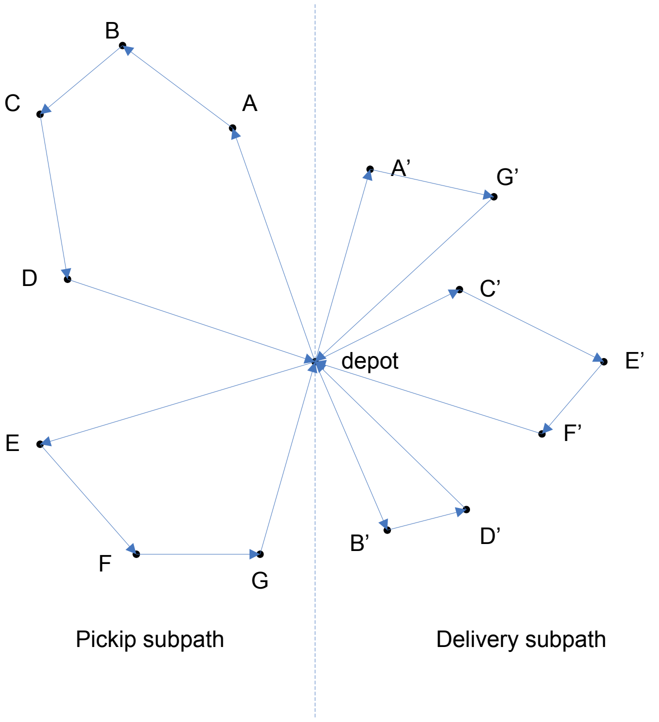


Figure 2.1: The Cross-Docking Problem

We would like a solution approach to the described problem (either heuristic or optimal) preferably with a mathematical model. If a heuristic is produced we would like an assesment of the quality of the solutions.

Chapter 3

Problem Analysis

3.1 Feature of Cross-Docking Problem

First we analyse the problem from the view of customer. Each customer has two service nodes, a pickup node and a delivery node. The service of a customer is to transport the order from its pickup node to its delivery node. The pickup node can be considered as a supply; the delivery node can be considered as a demand. Thus each customer has both supply and demand. The service is accomplished in two steps: 1)a customer's order is first picked up at the pickup node, then transported back to the deport and offloaded(if necessary). We denote this step as *pickup* and the transporting path as pickup subpath; then 2)From the deport, a customer's order is transported to the delivery node from the depot We denote this step as *delivery* and the transporting path as delivery subpath. Figure 3.1 show a whole trasportation process of service of a customer.

Second we analyse the problem from the view of vehicle routing. A vehicle can service more than one pickup orders in a pickup subpath, and can service more than one delivery orders in a delivery subpath. However, these two tasks is never inter-mixed. A vehicle can only pickup orders on the pickup path; and can only delivery orders on the delivery path. A vehicle can switch job between pickup and delivery or choose to do one type of job only. The job switching is only allowed at the deport only if a vehicle finishes its last job.

Tow situations of job switching process occuring at the deport: (a)a vehicle switches from pickup job to delivery job; (b)a vehicle swithes from delivery job to pickup job.

In (a) situation, after a vehicle has picked up some customers' order from pickup nodes and back to deport, it doesn't have to offload all the orders. In another word, if some part of the pickup orders are on the delivery path, they won't be offloaded at the deport. The vehicle will keep those orders onload, and upload

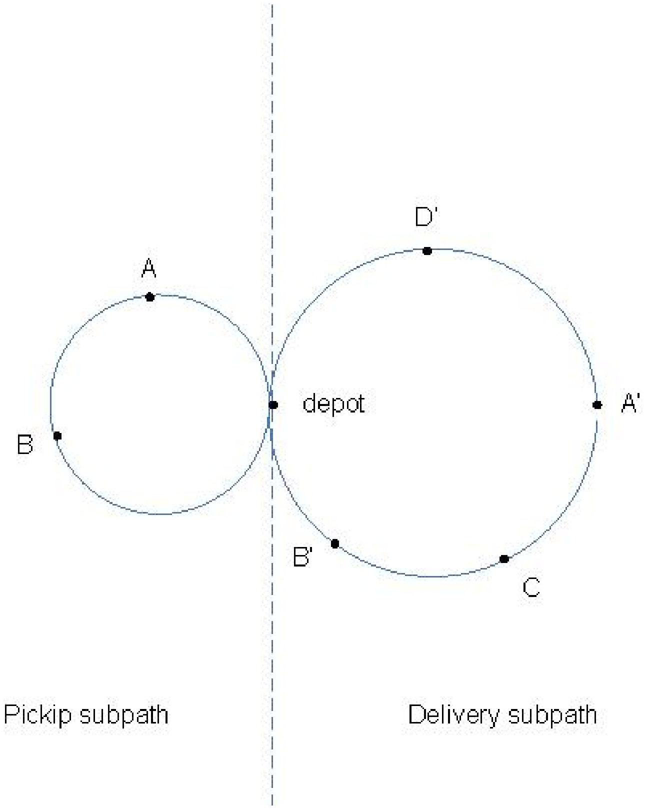


Figure 3.1: Transportation service of a customer

some additional delivery orders, which have already been transported into the deport. Thus on a delivery subpath, a delivery order is either a directive pickup order from the pickup subpath or a delivery order uploaded at the deport. During a job switching in situation (a), by keeping some orders onload on the vehicle at the depot, we can save time consuming on offloading and uploading directive pickup-delivery orders. This is the point of time saving in the cross-docking problem.

Obvioulsy, in (b) situation, a vehicle's onloading capacity is none after delivering the orders, no time saving can achieve from the job switching.

The Cross-Docking problem is to find the optimal path set which achieve delivering all orders in a minimum time cost. In addition to considering the set of shortest path covering each customer nodes, the time consuming on job switching at depot should also be taken into account.

3.2 Compared with other Pickup and Delivery VRP

The cross-docking problem is different from some other pick-up and delivery vehicle routing problems(PDVRP) studied before.

In dial-a-ride problem(DARP), where goods are picked up at one location and transported to an other location, the orders don't have to be transported back to the central deport as a connective node of the pickup part and delivery part. Thus DARP won't consider the time consuming of offloading and uploading at the deport.

In VRP with backhauls problem(VRPB), where goods are transported to and from the central depot, however, each customer has either a demand or supply, but not both. This model indicates that none directive pickup-delivery orders exist at all, so as that none time saving can be made from directive pickup-delivery orders, The most advantage in our cross-docking problem disappears.

In VRP with simultaneous delivery and pick-up points(VRPSP), where goods are transported to and from the central deport, and each customer has demand and a supply, and the operations must be made simultaneously. However, in cross-docking problem, as the pickup node and the delivery node of a customer are in seperate places, the pickup and delivery operation can't be made at a same time. Also as in DARP, there is none time consuming of offloading and uploading orders at the deport.

Thus, the cross-docking problem is clearly different from situations in the above pick-up and delivery vehicle routing problems. We can't use any solution method from those PDVRP. The feature of cross-docking problem is the time saving from loading(uploading/offloading) directive pickup-delivery orders. Our goal is to find more directive pickup-delivery orders; and also keep the shortest path set under a low value level. we starts from a study of the capacity constraints.

3.3 Single Path and Combination Path

In pickup subpart, a vehicle starts from deport with none capacity usage. The capacity usage is increased each time after visiting a customer pickup node. When back to the deport, the vehicle's onloading capacity reaches the maximum capacity usage of the pickup path.

In delivery subpart, a vehicle starts from deport with uploading all delivery orders of the delivery subpath. The vehicle's onloading capacity is the maximum capacity usage of the delivery path. Each time after visiting a customer, the capacity usage is decreased by the customer's order. When back to deport, the vehicle's capacity usage is zero.

Above are two conversed vehicle routing procedures. However they both possess a same capacity constraint property. That is, in each subpath, the maximum capacity usage should never excceed CAP, CAP is the maximum vehicle onloading capacity:

$$pickup\ subpath: \sum_{i \in PI} q_i \leq CAP \tag{3.1}$$

$$delivery\ subpath: \sum_{i \in DE} q_i \leq CAP \tag{3.2}$$

PI, DE is the node set covered by pickup path and delivery path, respectively.

Thus, regarding to capacity constraint, a pickup subpath and a delivery subpath are equivalent. A delivery subpath can be considered as a pickup subpath: starting from deport with zero capacity usage and increasing the capacity usage each time by visiting a customer. Now to find a shortest delivery path is the same as to find a shortest pickup path.

The same capacity constraint feature in both subpaths illuminate us a general way of solving the cross-docking problem. We can seperate the pickup part and delivery part into two equivalent subparts. Each subpart can be considered as an independent vehicle routing problem. In general solution method of VRP, a set of shortest paths will be generated in order to covery every customer nodes. In our problem, these shortest path will also satisfy capacity constraint and time window constraint, that is vehicles start routing after T_{start} and should finish all delivery before T_{end}. Moreover, we can't solve these two subparts completely seperately. We still have to think of the saving time by directive pickup-delivery order, which builds a relationship between a pickup subpath with a delivery subpath.

There are three types of paths defined by different kinds of nodes they're covering:

1. pickup path only covers pickup nodes;
2. delivery path only covers delivery nodes;
3. path covers both pickup nodes and delivery nodes;

The first two paths are the same as they contain only one job, we name it *single path* or *subpath*. The single path should meet time window constraint and capacity constraint. We will build a new algorithm for the negative reduced cost single path, named NDCA, which is developed based the GPLA.

The third path contains both pickup and delivery jobs, which is the point to construct the optimal solution in Cross-Docking problem. We name it *combination path*. The combination path should meet time window constraint and capacity constraint in both subpaths; and should try to achieve more saving cost from directive pickup-delivery orders. We will analysis the combination path in detail and form an algorithm CDSPA to find the optimal combination path.

We will apply Column Generation technique in the Cross-Docking problem to generate these two types of path with negative cost. After getting the optimal solution of liner relaxation of the Master problem, we will use branch and bound to searching the solution tree and find the optimal optimal integer solution.

3.4 Time Windows in Cross-Docking Problem

The time window constaint in our problem is not as usual situation in which each customer has its own service time window. Our loosey time window constraint indicates there will be more candidate paths existing during searching for the optimal shortest path, compared with a tight time window constraint.

The time windows constraint in Cross-Docking problem is different from other VRP. In addition to a same service time window for all nodes $[T_{start}, T_{end}]$, the time windows constraints build inter-connection between two or more subpaths makes a more complex situation.

Combined time window constraint

From the view of a vehicle which switches job at the depot, from pickup job to delivery job, the total time spending on both jobs should not excceed the maximum time window $T_{end} - T_{start}$. T_{start} and T_{end} are the opening and close time of depot. If such a job switching paths is generated in "one process", the time constraint is handled straightforwardly during generation. Otherwise, as the pickup subpath and the delivery subpath are generated in independant process, we can't ganratee the time window constraint is satisfied. This constraint builds connection between one pickup subpath and one delivery subpath.

Sequence time window constraint

From the view of a delivery path, a delivery path can't upload a delivery order until the order has been pickuped and transported into the depot. In another word, a delivery subpath should be waiting at the depot until all of its delivery orders being collected at depot. This constraint builds connection between one delivery subpath and several pickup subpaths.

Chapter 4

Overview of Column Generation

Column generation is clearly a success story in large scale integer programming. The linear programming bound obtained from an extensive reformulation is often stronger, the tailing off effect can be lessened or circumvented at all, and the knowledge of the original compact formulation provides us with a strong guide for branching and cutting decisions in the search tree. Today we are in a position that generic integer programming column generation codes solve many large scale problems of "industrial difficulty", no standard commercial MIP solver could cope with. This is all the more true since non-linearities occurring in practical problems can be taken care of in the subproblem.

Column generation may be most attractive in applications involving a huge number of columns since these problems present challenges to other methods. The goal is to formulate a decomposition that will allow the solution to the RMP to serve as a tight bound on the value of the optimal integer solution to facilitate the BB search.

4.1 Column Generation Technique

Let us call the following linear program the master problem (MP).

$$z^{\star}_{MP} := min \sum_{j \in J} c_j \lambda_j \tag{4.1}$$

$$subject\ to\ \sum_{j \in J} a_j \lambda_j \geq b \tag{4.2}$$

$$\lambda_j \geq 0,\ j \in J \tag{4.3}$$

In each iteration of the simplex method we look for a non-basic variable to price out and enter the basis. That is, given the non-negative vector π of dual variables we wish to find a $j \in J$ which minimizes $\bar{c}_j := c_j - \pi^t a_j$. This explicit pricing is a too costly operation when $|j|$ is huge. Instead, we work with a reasonably small subset $J' \subseteq J$ of columns, the restricted master problem (RMP), and evaluate reduced costs only by implicit enumeration. Let λ and π assume primal and dual optimal solutions of the current RMP, respectively. When columns $a_j, j \in J$, are given as elements of a set A, and the cost coefficient c_j can be computed from a_j via a function c then the subproblem

$$\bar{c}^\star := min\{c(a) - \pi^t a | a \in A\} \tag{4.4}$$

performs the pricing. If $\bar{c}^\star \geq 0$, there is no negative $\bar{c}_j, j \in J$, and the solution λ to the restricted master problem optimally solves the master problem as well. Otherwise, we add to the RMP the column derived from the optimal subproblem solution, and repeat with re-optimizing the RMP.

The advantage of solving an optimization problem in (4.4) instead of an enumeration in (4.1) becomes even more apparent when we remember that vectors $a \in A$ often encode combinatorial objects like paths, sets, or permutations. Then, A and the interpretation of cost are naturally defined on these structures, and we are provided with valuable information about what possible columns "look like".

In regards to convergence, note that each $a \in A$ is generated at most once since no variable in an optimal RMP has negative reduced cost. When dealing with some finite set A (as is practically always true), the column generation algorithm is exact.

In addition, we can make use of bounds. Let $\bar{z}$ denote the optimal objective function value to the RMP. When an upper bound $\kappa \geq \sum_{j \in J} \lambda_j$ holds for the optimal solution of the master problem, we have not only an upper bound $\bar{z}$ on $z^\star_{MP}$ in each iteration, but also a lower bound: we cannot reduce $\bar{z}$ by more than κ times the smallest reduced cost $\bar{c}^\star$:

$$\bar{z} + \kappa \bar{c}^\star \leq z^\star_{MP} \leq \bar{z} \tag{4.5}$$

Thus, we may verify the solution quality at any time. In the optimum of (4.1), $\bar{c}^\star = 0$ for the basic variables, and $\bar{z} = z^\star_{MP}$.

4.2 Dantzig and Wolfe Decomposition

We briefly review the classical decomposition principle in linear programming, due to Dantzig and Wolfe (1960). Consider a linear program (the original or compact formulation)

$$z^\star := min\ c^T x \tag{4.6}$$

$$subject\ to\ Ax \geq b \tag{4.7}$$

$$Dx \geq d \tag{4.8}$$

$$x \geq 0 \tag{4.9}$$

Let $P = \{x \in R^n_+ | Dx \geq d \neq \emptyset$. It is well known (Schrijver, 1986) that we can write each $x \in P$ as convex combination of extreme points $\{P_q\}_{q \in Q}$ plus non-negative combination of extreme rays $\{P_r\}_{r \in R}$ of P, i.e.,

$$x = \sum_{q \in Q} P_q \lambda_q + \sum_{r \in R} P_r \lambda_r,\ \sum_{q \in Q} \lambda_q = 1,\ \lambda \in R_+^{|Q|+|R|} \tag{4.10}$$

where the index sets Q and R are finite. Substituting for x in (4.6) and applying the linear transformations $c_j = c^T p_j$ and $a_j = Ap_j, j \in Q \cup R$ we obtain an equivalent extensive formulation

$$z^\star := min \sum_{q \in Q} c_q \lambda_q + \sum_{r \in R} c_r \lambda_r \tag{4.11}$$

$$subject\ to\ \sum_{q \in Q} a_q \lambda_q + \sum_{r \in R} a_r \lambda_r \geq b \tag{4.12}$$

$$\sum_{q \in Q} \lambda_q = 1 \tag{4.13}$$

$$\lambda \geq 0 \tag{4.14}$$

It typically has a large number $|Q| + |R|$ of variables, but possibly substantially fewer rows than (4.6). The equation $\sum_{q \in Q} \lambda_q = 1$ is referred to as the convexity constraint. If $x \equiv 0$ is feasible for P in (4.6) at zero cost it may be omitted in Q. The convexity constraint is then replaced by $\sum_{q \in Q} \lambda_q \leq 1$.

Although the compact and the extensive formulations are equivalent in that they give the same optimal objective function value $z^\star$, the respective polyhedra are not combinatorially equivalent. As (4.10) suggests, x uniquely reconstructs from a given λ, but not vice versa.

Given a dual optimal $\bar{u}$, $\bar{v}$ to the RMP obtained from (4.11), where variabel v corresponds to the convexity constraint, the subproblem (4.4) in Dantzig-WOlfe decomposition is to determine $min_{j\in Q\cup R}\{c_j - \bar{u}^T a_j - \bar{v}\}$. By our previous linear transformation this result in

$$\bar{c}^\star := min\{(c^T - \bar{u}^T A)x - \bar{v} | Dx \geq d, x \geq 0\} \tag{4.15}$$

This is a linear program again. We assumed $P \neq \emptyset$. When $\bar{c}^\star \geq 0$ no negative reduced cost column exists, and the algorithm terminates. When $\bar{c}^\star < 0$ and finit, the optimal solution to (4.15) is an extreme point p_q of P, and we added the column $[c^T p_q, (Ap_q)^T, 1]^T$ to the RMP. When $\bar{c}^\star = -$ we identify an extreme ray p_r of P as a homogeneous solution to (4.15), and we add the column $[c^T p_r, (Ap_q)^T, 0]^T$ to the RMP. From (4.5) together with the convexity constraint we obtain at each iteration

$$\bar{z} + \bar{c}^\star \leq z^\star \leq \bar{z} \tag{4.16}$$

where $\bar{z} = \bar{u}^T b + \bar{v}$ is again the optimal objective function value of the RMP. Note that the lower bound is also valid in the case the subproblem generate an extreme ray, that is, when $\bar{c}^\star = -$. Dantzig-Wolfe type approximation algorithms with guaranteed convergence rates have been propsed for certain liear programs.

If the original formulation is to obtain integer solution in x variables, $\bar{z}$ in (4.16) is not a valid upper bound on $z^\star$, except if the current x variables are integer. In general the generated set of columns may not contain an integer feasible solution. Branching and cutting constraints are added, the refomulation is re-applied, and the process continues with an updated master problem.

4.3 Integrality Property

Solving the subproblemas an integer programusually helps in closing part of the integrality gap of the master problem, except when the subproblem possesses the integrality property. This property means that solutions to the pricing problem are naturally integer when it is solved as a linear program. This is the case for our shortest path subproblem and this is why we obtained the value of the linear relaxation of the original problem as the value of the linear relaxation of the master problem.

Chapter 5

Modeling of Cross-Docking Problem

The necessary building blocks for a column generation based solution approach to integer programs: (1) an original formulation to solve which acts as the control center to facilitate the design of natural branching rules and cutting planes; (2) a master problem to determine the currently optimal dual multipliers and to provide a lower bound at each node of the branch-and-bound tree; (3) a pricing subproblem which explicitly reflects an embedded structure we wish to exploit.

Formulation may be the most crucial aspect an implementation. While it is difficult to reduce the formulation process to a simple series of steps, it is easy to describe desirable characteristics that a model should offer. Working to obtain these desirable characteristics typically leads to an iterative formulation process.

5.1 Modeling of Cross-Docking Problem

$$objective\ Z = min \sum_{k \in K^*} T_k^{pik} + T_k^{del} - saving_k \tag{5.1}$$

Subject to:

$$\sum_{k \in K^*} \sum_{j \in N \cup \{t\}} x_{ij}^k = 1 \qquad i \in N \tag{5.2}$$

$$\sum_{k \in K^*} \sum_{j \in N \cup \{t\}} y_{ij}^k = 1 \qquad i \in N \tag{5.3}$$

$$\sum_{i \in \{s\} \cup N} \sum_{j \in N \cup \{t\}} q_i x_{ij}^k \leq CAP \qquad k \in K^* \tag{5.4}$$

$$\sum_{i\in\{s\}\cup N}\sum_{j\in N\cup\{t\}} q_i y_{ij}^k \leq CAP \qquad k\in K^* \tag{5.5}$$

$$T_k^{pik} + T_k^{del} - saving_k \leq T_{max} \qquad k\in K^* \tag{5.6}$$

$$\sum_{k\in K^*}\sum_{j\in N\cup\{t\}} T_k^{pik} x_{ij}^k + \sum_{k\in K^*}\sum_{j\in N\cup\{t\}} T_k^{del} y_{ij}^k \leq T_{max} \qquad i\in N \tag{5.7}$$

$$\sum_{i\in N} x_{it}^k = 1 \qquad k\in K^* \tag{5.8}$$

$$\sum_{i\in N} y_{it}^k = 1 \qquad k\in K^* \tag{5.9}$$

$$\sum_{j\in N} x_{sj}^k = 1 \qquad k\in K^* \tag{5.10}$$

$$\sum_{j\in N} y_{sj}^k = 1 \qquad k\in K^* \tag{5.11}$$

$$\sum_{j\in\{s\}\cup N} x_{ji}^k = \sum_{j\in N\cup\{t\}} x_{ij}^k \qquad i\in N \qquad k\in K^* \tag{5.12}$$

$$\sum_{j\in\{s\}\cup N} y_{ji}^k = \sum_{j\in N\cup\{t\}} y_{ij}^k \qquad i\in N \qquad k\in K^* \tag{5.13}$$

$$\sum_{j\in N\cup\{t\}} x_{ij}^k \leq 1 \qquad i\in N \qquad k\in K^* \tag{5.14}$$

$$\sum_{j\in N\cup\{t\}} y_{ij}^k \leq 1 \qquad i\in N \qquad k\in K^* \tag{5.15}$$

$$T_k^{pik} = \sum_{i\in\{s\}\cup N}\sum_{j\in N\cup\{t\}} (dist_{ij}^{pik} + ser_j) x_{ij}^k \qquad k\in K^* \tag{5.16}$$

$$T_k^{del} = \sum_{i\in\{s\}\cup N}\sum_{j\in N\cup\{t\}} (dist_{ij}^{del} + ser_j) y_{ij}^k \qquad k\in K^* \tag{5.17}$$

$$ser_i = docking + loading * q_i \qquad i\in N \tag{5.18}$$

$$ser_s = 0 \tag{5.19}$$

$$ser_{kt}^{pik} = docking + loading \sum_{i\in N}\sum_{j\in N\cup\{t\}} q_i x_{ij}^k \qquad k\in K^* \tag{5.20}$$

$$ser_{kt}^{del} = docking + loading \sum_{i\in N}\sum_{j\in N\cup\{t\}} q_i y_{ij}^k \qquad k\in K^* \tag{5.21}$$

$$saving_k = 2loading \sum_{i \in N} q_i \Big(\sum_{j \in N \cup \{t\}} x_{ij}^k \sum_{j \in N \cup \{t\}} y_{ij}^k \Big) \qquad k \in K^* \tag{5.22}$$

$$x_{ij}^k \in \{0,1\} \qquad y_{ij}^k \in \{0,1\} \tag{5.23}$$

Parameters:

T_{start}: Depot open time.

T_{close}: Depot close time.

T_{max}: The depot opeing period, $T_{max} = T_{close} T_{start}$

CAP: The vehicle capacity.

$loading$: The time consuming of loading one pallet.

$docking$: Docking time.

q_i: Order of customer i.

$dist_{ij}^{pik}$: Distance during time between (i, j) in pickup subpart.

$dist_{ij}^{del}$: Distance during time between (i, j) in delivery subpart.

s: Start deport.

t: Back deport.

N: Customer set.

Variables:

K^*: The optimal solution paths set. A path in K^* could be either a single path or a combination path.

T_k^{pik}: Time consuming of path k in pickup subpart. If a path has no pickup subpart, T_k^{pik} is zero.

$T_k^{delivery}$: Time consuming of path k in delivery subpart. If a path has no delivery subpart, T_k^{del} is zero.

$saving_k$: The saving time of path k. A single path's saving time is always zero.

ser_i: Service time of node i.

x_{ij}^k: 1 means path k covers arc (i, j) in pickup subpart; otherwise, 0.

y_{ij}^k: 1 means path k covers arc (i, j) in delivery subpart; otherwise, 0.

The objective function is to minimize total time consuming of delivery all customers' orders. In another word, it is to minimize the sum of time consuming of each path in K^*. $T_k^{pik} + T_k^{del} - saving_k$ is time consuming of path k.

(5.2), (5.3) make sure that each costomer pickup node and delivery node is visited exactly once.

(5.4), (5.5) is the capacity constraint in each subpart.

(5.6) is the combined time window constaint. Time consuming of each path, either single path or combination path, should not exceed the depot opening duration.

(5.7) makes the sequence time window constraint satisfied. For each customer i, there exist a combination path or two single paths to pickup and deliver i's order. In later situation, assume i's pickup subpath is k_i^{pik} and delivery subpath is k_i^{del}. And T_i^{pik}, T_i^{del} are the time consuming of k_i^{pik}, k_i^{del}, respectively. Obviously, $T_i^{pik} + T_i^{del}$ should not excced the depot opening duration T_{max}. $\sum_{k \in K^*} \sum_{j \in N \cup \{t\}} T_k^{pik} x_{ij}^k$ is the T_i^{pik}; $\sum_{k \in K^*} \sum_{j \in N \cup \{t\}} T_k^{del} y_{ij}^k$ is the T_i^{del}. If each customer satisfiy time window constraint (5.7), the sequence time window constraint are certainly satisfied. On each delivery single path, once the latest order has been pickuped and transported back to depot in time, combination time window constarint will be satisfied. In former situaion, when a customer is serviced by a combination path, (5.6) has covered this situation.

(5.8) to (5.15) are flow conservation constraints.

(5.8) to (5.11) make sure each path k start at start depot and end at back depot. (5.12), (5.13) make sure the incoming flow is equal to the outgoing flow.

(5.14), (5.15) make sure each node is at most visited once by path k.

(5.16), (5.17) are the calculation of time consuming, which include distance covering duration and service time at each customer node and depot.

(5.18) to (5.21) are the service time calculation. Service time is composed of docking time and loading time of order(s). The service time at a customer's pickcup node and delivery node is the same. The service time at back depot depend on the onloading capacity of a subpath. Required by our solution method, we still include the offloading and onloading service time of a directive pickup-delivery order at depot. However, this part of service time will be deducted in *saving*.

(5.22) is saving time calculation. Only a combination path could have none zero saving time. As we include loading time of directive pickup-delivery orders at depot, we have to deduct it here. A directive pickup-delivery order is an order existing on both subpaths of a combination path.

5.2 Master Problem

sec:master$_{p}roblem$
We write the time consuming of each path k as T_k,

$$C_k = T_k^{pik} + T_k^{del} - saving_k \qquad k \in K \tag{5.24}$$

The RMP is built as a set of paths k, $k \in K$, K is the candidate paths set, the objective is to minimize the sum of T_k, $k \in K$.

$$objective\ Z^{RMP} = min \sum_{k \in K} C_k \lambda_k \tag{5.25}$$

Subject to:

$$\sum_{k \in K} a_{ik} \lambda_k = 1 \qquad i \in N \tag{5.26}$$

$$\sum_{k \in K} b_{ik} \lambda_k = 1 \qquad i \in N \tag{5.27}$$

$$\sum_{k \in K} C_k (a_{ik} | b_{ik}) \leq T_{max} \qquad i \in N \tag{5.28}$$

$$\lambda \in [0,1] \tag{5.29}$$

a_{ik}: 1 if pickup node of customer i is on path k; other wise, 0;

b_{ik}: 1 if delivery node of customer i is on path k; other wise, 0;

(5.26) and (5.27) are constraints of one visiting at each customers' node.

(5.28) is sequence time window constraint. If a customer order is serviced by a combination path k$_{com}, then$

$$a_{ik_{com}} = 1$$

$$b_{ik_{com}} = 1$$

$$a_{ik_{com}} | b_{ik_{com}} = 1$$

(5.28) will be

$$C_{kcom} \leq T_{max}$$

If a customer order is serviced by two independent single paths k_{pik} and k_{del}, (5.28) will be

$$C_{k_{pik}} + C_{k_{del}} \leq T_{max}$$

u, v are dual variables of condition set (5.26) and (5.27), respectively. w is dual variables of condition set (5.28).

Before we continue to build the model of subproblem, first analysis condition (5.28). Condition (5.28) brings the following polynomial into the subproblem,

$$\sum_{i \in N} C_k(a_i|b_i)w_i \tag{5.30}$$

C_k is contained in (5.30), however C_k is the coefficient in RMP objective function. Thus (5.30) make the subproblem not satisfied the structure of being a shortest path problem. For easy implementation in the Cross-Docking Problem, we use a heuristic method.

We replace the condition (5.28) by two conditions,

$$T_{pik} \leq 0.25T_{max} \tag{5.31}$$

$$T_{del} \leq 0.75T_{max} \tag{5.32}$$

(5.31) make sure a pickup single path not excceed 0.25 of depot opening duration; (5.32) make sure a delivery single path not excceed 0.75 of depot opening duration.

These two conditions ensure that both the combination time window constraints and sequence time window constraints are satisfied, as any pickup subpath can be contacted with any delivery subpath. However, this condidtion replacement will reduce the feasible solution set, and increase the optimal objective value.

5.3 Subproblem

$$objective\ Z^{sub} = min\ C - \sum_{i \in N} \sum_{j \in N \cup \{t\}} x_{ij}u_i - \sum_{i \in N} \sum_{j \in N \cup \{t\}} y_{ij}v_i \tag{5.33}$$

The subproblem is to find the shortest single path or combination path k. If the cost of k is negative, new column built from k is inserted into RMP; if none path has negative reduced cost, the optimal solution of RMP is found.

We write (5.33) seperately for single path and combination path.

$$objective\ Z_{pik}^{sub} = min \sum_{i\in\{s\}\cup N}\sum_{j\in N\cup\{t\}}(dist_{ij}^{pik} + serv_j)x_{ij} - \sum_{i\in N} a_i u_i \tag{5.34}$$

let $u_s = 0$, (5.34) can be written as

$$objective\ Z_{pik}^{sub} = min \sum_{i\in\{s\}\cup N}\sum_{j\in N\cup\{t\}}(dist_{ij}^{pik} + serv_j - u_i)x_{ij} \tag{5.35}$$

(5.35) is the objective function for the shortest pickup single path. A pickup single path satisfy all flow conservation constraints, capacity constraint and time window constraint. The time window constraint is the time consuming of a pickup single path which should not exceed $0.25T_{max}$.

The shortest delivery single path has the same structure as (5.35), let $v_s = 0$:

$$objective\ Z_{del}^{sub} = min \sum_{i\in\{s\}\cup N}\sum_{j\in N\cup\{t\}}(dist_{ij}^{del} + serv_j - v_i)y_{ij} \tag{5.36}$$

A delivery single path satisfy all flow conservation constraints, capacity constraint and time window constraint. The time window constraint is the time consuming of a delivery single path which should not exceed $0.75T_{max}$.

The shortest combination path path objective function is

$$objective\ Z_{comb}^{sub} = min\ cost^{pik} + cost^{del} - saving \tag{5.37}$$

$$cost^{pik} = \sum_{i\in\{s\}\cup N}\sum_{j\in N\cup\{t\}}(dist_{ij}^{pik} + serv_j - u_i)x_{ij} \tag{5.38}$$

$$cost^{del} = \sum_{i\in\{s\}\cup N}\sum_{j\in N\cup\{t\}}(dist_{ij}^{del} + serv_j - v_i)y_{ij} \tag{5.39}$$

$$saving = 2loading \sum_{i\in N} q_i(\sum_{j\in N\cup\{t\}} x_{ij} \sum_{j\in N\cup\{t\}} y_{ij}) \tag{5.40}$$

Chapter 6

GPLA for SPPTW

6.1 SPPTW Problem

The shortest path problem with time windows (SPPTW) consists of finding the least cost route between a source p and a sink q in a network $G = (N, A)$ while respecting specified time windows $[a_i, b_i]$ at each visited node. The duration d_{ij} of each arc is restricted to positive values while the cost c_{ij} of each arc $(i, j) \in A$ is unrestricted.The SPPTW is NP-Hard.

GPLA presenets an efficient generaliezd permanent labelling algorithm to solve SPPTW. This algorithm is base on the defining of the concept of a generalized buckets and on a specific order of handling the labels.The algorithm runs in pseudo-polymonial time.

Even if the time windows constraints and the positive durations guarantee the finiteness of feasible paths, they do not guarantee that the feasible paths will be elementary (i.e. all nodes in a path visited only once).

6.2 SPP and RCSPP

The unconstrained shortest path problein has a considerable importance in transportation models and is the subject of an enormous number of papers. The solution to the shortest path problem is a directed spanning tree T of $G = (N, A)$ rooted at source p. Let label C_i be the cost of the unique path in T from p to i, $i \in N$. T is a shortest path tree with origin p if and only if Bellman's conditions hold:

$$C_i + c_{ij} - C_j \geq 0 \qquad \forall (i, j) \in A \tag{6.1}$$

All the algorithm of finding a solution of the shortest path problem use dynamic programming and perform the same operations:

1. Initilize a directed tree T rooted at p and for each $i \in N$, let C_i be the cost of the path in T from p to i;
2. Let $(i, j) \in A$ be an arc for which condition (6.1) is not satisfied. Then update the path cost accordingly, i.e.: $C_j := C_i + c_{ij}$ and adjust the tree T replacing the current arc incident into node j by arc (i, j);
3. Repeat step 2 until conditions (6.1) are satisfied for all arcs.

The main aspect in the implementation of this procedure is how to select an arc at step 2 to verify condition (6.1). In most algorithms, a node k is selected and treated, all all $k's$ successors are checked; i.e., for all arcs $(k, j) \in A$ step 2 is performed. Let Q be the set of candidate nodes; i.e., the set of nodes whose leaving arcs are not guaranteed to satisfy (6.1). There are many selection rules for node $k \in Q$. The most common being:

1. FIFO: the oldest element in Q is selected and treated. The set Q is represented by a queue. New nodes are inserted at the tail of the queue; the node to be treated is selected from the head of the queue;
2. LIFO: the newest element in Q is selected and treated. The set Q is represented by a stack. New nodes are inserted at the top of the stack; the node to be treated is selected from the top of the stack;
3. Best-First: the node $k \in Q$ with the least cost C_k is selected and treated.

In the resource constraint shortest path problem(RCSPP), constraints can be the vehicle max loading capacity, the vehicle longest travel distance/time; customer serviced in specific sequence order, and other constraints on either vehicles, customers or on both. All these additional constraints make SPPAC more complex in that a set of candidate routes associated with a node. For instance, vehicle loading capacity constraint occurs in most RCSPP.

SPPTW can be considered as a special RCSPP, i.e., time windows is a resource constraint of service time at customer. The SPPTW was formulated by Desrosiers, Pelletier and Soumis as a subproblem of a route construction problem. The authors formulate the following optimality principle: for a given path X_{pj} from source p to node j , if this path is efficient and if arc (i, j) is the last arc of X_{pj}, then the sub-path X_{pi} is an efficient path as defined in 6.3. This dynamic programming based method generally allows the efficient treatment of the nodes. This method is sensitive to the initial ordering of the data, and the algorithm can be accelerated by sorting the data in increasing order of starting times of the time windows. This algorithm can solve problems of low to medium density 15% to 30% with up to 300 nodes.

6.3 GPLA for SPPTW

The GPLA impove the SPPTW algorithms by using a different rule of node selection. The algorithm uses pushing method to only extend the efficient paths from source node to sink node. It reduces the operations down to a polynomial complexity of order $O(D^2)$. This is achieved by defining a Best-First rule of treatment of the efficients labels and by using generalized buckets.

The following sections are constructed by in setion 6.4 giving the terminologe and general treatment order of paths and labels inGPLA, in section 6.5 and 6.6 introducing two crucial ideas, lexicographically order and bucket, and how they can increase the operation in GPLA, in section 6.7 giving details of GPLA alrogithm.

6.4 Path and Label Treatment

With each path, i.e.: X_{pj} from the origin p to the node j satisfying time windows, is associated a (time, cost) label corresponding to the arrival time at node j and the cost of the path X_{pj}, respectively. There are always a set of possible path from node p to node j. These labels will be denoted by (T_i^k, C_i^k) to indicate the characteristics of the k^{th} path from p to i, where the indices k and i may be dropped when the context is unambiguous.

The visiting nodes sequence is also a very important feature of a path. Even two paths cover a same set of nodes visited, if the visiting sequence is different, the cost of these two paths is not certainly the same. Thus it is necessary to recored nodes sequence for each path. A sequence containts two types of infomation, coverd nodes and visiting sequence.

To find the optimal solution of SPPTW is to find the minimal cost of path fron origin p to destination q which is also satisfied time windows constraint. This is achived by extending any feasible paths from p to q. *Pushing* and *pulling* are two methods to extend a path. In *pushing* method, during the treatment of a node i, the set of paths of node i, i.e., $\{X_{pi}^1, X_{pi}^2, X_{pi}^3, \cdots\}$ is all pushed to i's successors respectively. If the extended path of a successor j meets the time window constraint of j, it becomes a new feasible path of j, i.e., $\{X_{pj}^1, X_{pj}^2, X_{pj}^3, \cdots\}$. In *pulling* method, during the treatment of node i, paths of predessors of node i are extended to i. GPLA use *pushing* method to extend path.

Each path is constructed by a sequence of nodes which are passed through by the path subsequently, i.e.: $X_{pj} = (i^0, i^1, i^2, \cdots, i^L)$. Where $i^0 = p$ and $i^L = j$. Labels of these nodes are calculated iteratively along the path, as follows:

$$T_{i^0} = 0 \tag{6.2}$$

$$C_{i^0} = 0 \tag{6.3}$$

$$T_{i^l} = max\{a_{i^l}, T_{i^{l-1}} + d_{i^{l-1}i^l}\} \qquad l = 1, \cdots, L \tag{6.4}$$

$$C_{i^l} = C_{i^{l-1}} + c_{i^{l-1}i^l} \qquad l = 1, \cdots, L \tag{6.5}$$

Start status (6.2), (6.3) means that no time consuming and cost from the start node. (6.4) indicate to arrive node i^l, time consuming is the sum of time consuming of preceed node i^{l-1} and time using on arc (i^{l-1}, i^l). (6.5) is the cost of path arriving node i^l. Additionally for time windows constraint, the arrival time T_{i^l} can't be early than the service time windows of node i^l.

Following, give definitions of terminologe used within the argurithm GPLA.

Definition 1 (Domination) *Let X^1 and X^2 be two different paths from p to j with associated labels (T^1, C^1) and (T^2, C^2). Then X^1 dominates X^2 or (T^1, C^1) dom (T^2, C^2) if and only if $(T^1, C^1) - (T^2, C^2) \leq (0, 0)$ and $(T^1, C^1) \neq (T^2, C^2)$*

Definition 2 (Efficient label/path) *A label (T, C) at a given node j is said to be efficient if no other label at j dominates it. A path X_{pj} from p to j is said to be efficient if its label is efficient.*

Obviously, a set of efficient labels may be linked to a node. The *domination* relation is not a total order and dose not allow all paths to be ordered. But, it does allow us to conclude that an efficient path X_{pj} is the shortest path arriving at node j at time T_j or before. In order to full order all efficient path, GPLA use the well-known *lexicographic ordering.* Details is presented in 6.5.

This relation allows the definition of the cost of the feasible path as a function of arrival time. Figure 6.1 shows this cost-time function for a given node. The vertical axis represents the paths cost and the horizontal axis represents the arrival time of the paths.

Figure 6.1 illustrates the relation between several different labels associated with efficient paths, i.e., X^1 to X^4; and others which are dominated, i.e., X^5 to X^8. I.e., analyse X_2 and X_7. From figure obviously show that both arrival time and cost of label X_2 than label X_7. Also in the sample from section 6.2, p_1 and p_2 are both efficient path of node i, as cost of p_2 is less than p_1, however used capacity of p_1 is less than p_2.

$EFF(Q)$ denote the set of efficient labels among Q. $EFF(Q_i)$ denote the set of efficient labels of node i. The process of extension of a path will be as following. Let (T_i^k, C_i^k) be an efficient label at node i and its associated path X_{pi}. When arc $(i, j) \in A$ is added to the path X_{pi}, a new path X_{pj} for node j is obtained if $T_j^k + d_{ij} \leq b_j$. The new label associated to this extended path is:

$$(T_j, C_j) = (max[a_j, T_i^k + d_{ij}], C_i^k + c_{ij}) \tag{6.6}$$

The label is added to the set of efficient label at node j , Q_j and this set is updated:

$$Q_j := EFF(Q_j \cup \{(T_j, C_j)\}) \tag{6.7}$$

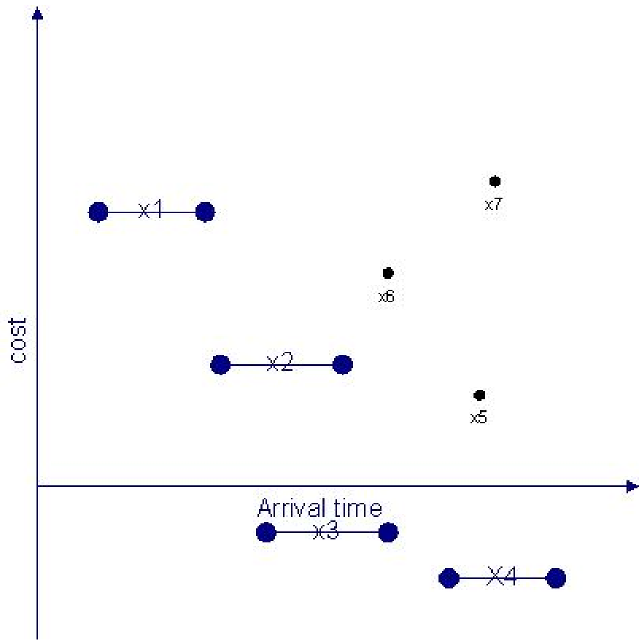


Figure 6.1: Domination Relation between labels Associated with Different Paths

This treatment is done for all arcs $(i, j) \in A$ incident to node i. One thing should be mentioned is, an efficient path, during the path extension, will possibly be dominated by other efficient path. Thus it is a continously process of new efficient paths gennerating and efficient paths, either original one or new extended one, being dominated. If to extend path in a way that the new path is less possible or impossible improved/dominated by other path, the algorithm operation times can be reduced. Next two sections intruduce two labels treatment methods to reduce the occuring of dominating/being dominated situation during extending process.

6.5 A New Order of Treatment of Labels

As Known in section 6.2, in a shortest path problems, in order to select an arc at step 2 to verify condition (6.1), first define an treatment order of candidate nodes, then apply the verification for all arcs of a selected node k. Three of the most common rules for selecting candidate node $k \in Q$ are FIFO, LIFO and Best-First. GLPA use Best-First rule of treatment of efficient labels.

In a unconstrained shortest path problem, an efficient label of a node k only depends on path cost. The notions of "node" and "label" are closely indication, i.e., only one efficient label linked with a node. The order of treatment is thus defined simultaneously for nodes and labels. When apply Best-First rule algorithm, select a minimun cost label from the set of untreated nodes, i.e., node k with lablel (C_i), then deal with path extension to all $k's$ successors.

However, this is not true in the case of constrained shortest path problem. In later situation, a set of efficient labels are possibly associated with each node, these two notions are distinct. The existence of an efficient Best-First rule algorithm for the SPPTW depends on the existence of a treatment order for the labels with a qualification condition on the arcs such that it is impossible to improve a label which has been previously treated.

The well-known lexicographic ordering is a total ordering in $\Re^2$, i.e. the following three propositions hold:

1. For all a, $b \in \Re^2$, if $a < b$, then $a < b$.
2. For all a, $b \in \Re^2$, if $a > b$, then $a > b$.
3. For all a, $b \in \Re^2$, if $a < b$, then $a \quad b$.

Lexicographic ordering is compatible with the dominance ordering, i.e., a label (T_i, C_i) is in $\Re^2$. As the time consuming t is always a positive number, all arcs have lexicographically positive label. The sign of cost c is unrestricted.

Using increasing lexicographic ordering of time consuming, we get a full ordering of the efficient labels. It is obviously that after sorting the efficient paths, the order of label's duration and cost is conversed. I.e., if labels are sorted in increasing duration order, their costs are in decreaing order. Assume efficient paths: $(10, 20)$, $(30, 5)$, $(20, 11)$, the increasing lexicographically order of time consuming is:

$$(10, 20), (20, 11), (30, 5)$$

their cost is in decreasing order.

Following is two *Theorem* of lexicographic ordering.

Theorem 1 *Let P be the set of labels already treated and let Q be the set of untreated labels. If*

(1) *all arcs $(i, j) \in A$ have lexicographically positive label, and*
(2) *the labels of Q are treated in increasing lexicographic order, then*

for all $a \in P$ and all $b \in Q$, $a < b$, the treatment of any label of Q can not improve a label in P.

Proof : The successors of any given label being treated are all lexicographically greater than this label. Using induction on the cardinality of P and that any element of Q is a successor of an element of P, we derive the conclusions easily.

Figure 6.2: Treatment order of labels

Figure 6.2 explain theorem more clearly. The rectangle represent the treated labels set P and untreated labels set Q. In set $\{P \cup Q\}$ all labels are sorted by lexicographically increasing order. Any treatment of label of Q won't effect a label in P. This means treatment of labels of larger lexicographical order won't improve labels of smaller lexicographical order. Thus treating the set of untreated labels in increasing lexicographic order, new labels won't improve a previously treated label. This label treatment order satisfies the application of Best-First rule algorithm for the SPPTW, and does help to reduce the operation times by avoiding doing unwise extending.

6.6 The Concept of A Generalized Bucket

In last section present the increasing lexicographically treatment order of labels from the overview of GPLA, this section will show that in some small range of lexicographical interval, arbitrary treatment order of labels can be used. These small interval are defined as bucket.

Desrochers improved the criterion for choosing the next label to be treated by using a generalization of the concept of a "bucket" introduced by Denardo and Fox. A bucket is a list of nodes whose label values lie within a specified intenrval. In the simple case, the p^{th} bucket is made up of all nodes whose label values are included in the semi-open interval $[pm, (p+1)m)$ where m is the width of the bucket. The set of buckets generated by the algorithm has the same bucket width spreading over different intervals.

$$m = min\{c_{ij}\}.$$

The search for the smallest temporary label is thus replaced by the search for an element of the first bucket to contain the temporary labels.

In $\Re^2$, GPLA use the same concept of a bucket by defining the width of the bucket as

$$(m_d, m_c) = minlex\{(d_{ij}, c_{ij})\}$$

Before analyse the treatment order property within a bucket, first construct a bucket. Let $F(Q)$ be the next label to be handled according to increasing lexicographically order, $F(Q)$ is the smallest label in the set of untreated labels:

$$F(Q) = minlex\{(T_i^k, C_i^k)\}$$

Construct a bucket $B(Q)$ which lower bound is at $F(Q)$, thus $B(Q)$ is the generalized bucket defined by:

$$B(Q) = \{(T_i^k, C_i^k) \in Q | F(Q) \leq (T_i^k, C_i^k) < F(Q) + (m_d, m_c)\}$$

The upper bound of $B(Q)$ is $F(Q) + (m_d, m_c)$.

Desrochers showed that replacing the treatment of $F(Q)$ by the treatment of an element in $B(Q)$ does not alter the optimality of the algorithm.

Theorem 2 *If* $(m_d, m_c) > (0, 0)$*, then the elements of the generalized bucket* $B(Q)$ *cannot be improved during the treatment of an element of* $B(Q)$*.*

Proof: Suppose it were possible to improve an element (T_j^k, C_j^k) of $B(Q)$ by treating another element (T_i^l, C_i^l) of $B(Q)$,

$$(T_i^l, C_i^l) + (d_{ij}, c_{ij}) < (T_j^k, C_j^k) \Rightarrow (T_i^l, C_i^l) + (d_{ij}, c_{ij}) < (T_j^k, C_j^k)$$

on the other hand,

$$(T_j^k, C_j^k) < F(Q) + (m_d, m_c) \qquad as (T_j^k, C_j^k) \in B(Q)$$

$$(T_i^l, C_i^l) + (d_{ij}, c_{ij}) < F(Q) + (m_d, m_c)$$

$$(T_i^l, C_i^l) < F(Q) + (m_d, m_c) - (d_{ij}, c_{ij})$$

$$(T_i^l, C_i^l) < F(Q) \qquad as\ (m_d, m_c) < (d_{ij}, c_{ij})$$

which contradicts the definition of $B(Q)$. It is therefore impossible to improve the elements of $B(Q)$.

In the case of the SPPTW, since the durations d_{ij} are strictly positive, it is sufficient to verify that the label (T, C) are lexicographically positive without any restrictions on the costs. Thus the value of m_d , which is only decided by d_{ij}, where

$$m_d = min\{d_{ij}\} \qquad (i, j) \in A$$

Using bucket method to collect a small set of untreated labels within a bucket, from *Theorem 2*, labels can be treated in any order without affecting the operation efficiency.

6.7 GPLA Algorithm

In previous sections, it has been sufficient to verify that in SPPTW, label (T, C) is lexicographically positive. The generalized permanent labelling algorithm (GPLA) can be described as follows:

Step1: Initialization.

$$\mathrm{P}_i = \{ \begin{array}{ll} \{(0,0)\} & i = p \\ \emptyset & \forall i \in N, i \neq p \end{array}$$
$$\mathrm{Q}_i = \emptyset \qquad \forall i \in N$$
$$m_d = mind_{ij} \qquad (i, j) \in A$$

P_i is the set of permament labels for node i.
Q_i is the set of candidate labels for node i.

Step2: Find the current bucket.

Find $F(Q)$ the label $\{(T_i^k, C_i^k)\}$ of lexcigraphically minimum cost from the set $Q = \cup_i(Q_i - P_i)$. If $Q = \emptyset$, stop.
Calculate the upper bound of $B(Q)$,

Step3: Find the next label to be treated.

Find one element of $B(Q)$.
If $B(Q)$ is empty, go to step 2.

Step4: Treatment of label (T_i^k, C_i^k).

```
For all successors $j$ of node $i$ do
begin
if $T_i^k + d_{ij} < b_j$ (time windows satisfied) then
$(T_j, C_j) = (max(a_j, T_i^k + d_{ij}), C_i^k + c_{ij})$
$Q_j = EFF(Q_j \cup \{(T_j, C_j)\})$
end.
```

```
        $P_i = P_i \cup \{(T_i^k, C_i^k)\}$
        go to step 3.
```

Step 1 is initialization, path extension starts from root node p with none duration consuming and cost. Step 2 is to find the lexicographically minimum cost label $F(Q_i)$ from untreated labels. Step 3 is to construct bucket $B(Q)$ starting from $F(Q_i)$ and collect all untreated labels within $B(Q)$. Step 4 is how to extend each label in bucket $B(Q)$. New extended paths are added into efficient label set only if not being dominated by existing paths and without violating the time window constraint. After finishing all paths extension in $B(Q)$, go back step 2 and find the next untreated label.

By applying Theorem 1, step 2 shows that the untreated labels are selected to be treated in an increasing lexicographic order of time; by applying Theorem 2, Step 3 and 4 show untreated labels grouped by a bucket are treated in an arbitrary order. From an overview, the treatment of labels of a specific node still follow an increaing lexicographically order.

To verify the time window constraint in Step 4, two conditions are checked: 1) the finish time can't exceed the time window constraint b_i; 2) the start time can't be earlier than the time window constraint a_i.

Chapter 7

NDCA for SPPTW

From this section we will describe state-dominance criteria for the GPLA for solving the SPPTW on dense graphs. The new criteria markedly improve its performance. The new algorithm developed based on GPLA is new dominance criteria algorithm, in short NDCA.

Two types of dominance check are intruduced in the NDCA. One is a dominance test of label at the destination node, the other is at the original label being treated. At destination node dominance check, two criterias are used: 1) a minimun cost label dominance check; 2) a backward-looking dominance test. At treated node, the criteria of minimun cost label dominance test is used.

Both dominance checks are possible due to a new label arrangement and treatment order within each bucket: 1)labels are grouped by node to which they belong; 2)labels are stored and treated lexicographically in decreasing service ime order and increasing cost order. This treatement order allied with the suggested dominance criteria results in a significant time execution performance improvement with respect to the basic dense-graph GPLA.

7.1 Review of GPLA

Before continue the new dominance criterias, review some points in GPLA.

For each node i with time windows constraint $[a_i, b_i]$, which means for any label, the service time of custom i is no earlier than a_i and no later than b_i. Only consider integer solution, the number of different possible time value in a label is: $\sum(b_i - a_i) + 1$. Except labels having same time and cost value, for each node i, the maximun number of efficient labels is $\sum(b_i - a_i) + 1$. The total number of efficient labels in a SPPTW is:

$$\sum_{i \in N}(b_i - a_i) + 1$$

In GPLA, it is important to point out that Desrochers and Soumis (1988) concentrated their computational investigation of the implementation of GPLA to situations in which the number of labels of nodes actually present in the solution is small relative to the number of possible efficient labels ($\sum_i (b_i - a_i) + 1$). In their tests, they used a linear list, sorted in lexicographic order, to represent each set Q_i of temporary candidate labels for node i. At each iteration, for every successor j of node i, the corresponding label (T_j, C_j) was stored in the set Q_j if and only if it was not dominated. Thus, for each successor j the merger and reduction of the set of undominated labels associated with node j required the comparison of the labels with every other labels stored in Q_j.

In GPLA for dense graphs, Desrochers and Soumis (1988) also suggest a rather different implementation of the GPL algorithm (which for purposes of clarity we will call GPLd, with the lower case "d" referring to "dense") from the formal GPLA description in the same paper. As mentioned before, in the GPLA only non-dominated labels of node i are added to the set of efficient labels Q_i. In GPLd, however, the authors proposed to use a table to represent each set Q_i: Thus, all efficient paths as well as some dominated ones are included. According to Desrochers and Soumis, to eliminate the dominated paths it is sufficient to carry out a dominance test at each node at the end of the algorithm.

The dominance check in GPLA and GPLd are both time consuming, every new label should compared with existing labels of node i before adding to efficient set; or a whole dominance test is needed at a destination node at each iteration. Moreover during every dominance check, both time service value and path cost comparation are needed.

In our new dominance criteria algorithm, more dominance tests are intruduced into the process, as in GPLA and GPLd, these dominance tests improve algorithm's performance by sharply reducing the increasing of the amount of candidate labels. Moreover, these new dominance tests are more efficient and less time consuming. In some situations, dominance tests speeds up operations by avoiding applying check for every labels in a candidate set. Most of the dominance checks only need one comparation of the time value.

7.2 Dominance Test for the Label being Treated

In basic NDCA, the improvement is brought about by the intruduction of a dominance test for a new extended label before adding it into the successor's candidate label set. This test becomes feasible if the temporary labels for a given node are treated lexicographically in decreasing time order.

Instead of sorting the temporary labels lexicographically in decreasing service time directly, the same result can be achieved more easily by sorting buckets.

As with Desrochers and Soumis, the bucket width is constant and equal to the smallest inter-task duration from i to j plus duration of task i.

$$m = min(d_i + t_{ij}) \qquad \forall i, j \in N$$

.

If a label belongs to bucket p, its start service time lies within the bucket interval: $[mp, m(p+1))$. As the bucket index increases, the time interval increases. Thus labels belong to lower index buckets have earlier start service time than those belong to higher index buckets. Based on the relationship between bucket and service time, treating buckets in increasing index order naturally brings to a treatement order of labels in increasing start service time.

The following three points prove the feasiblity to treate buckets in an increasing index order.

1. First, from theorem 2, buckets can be swept in increasing order without affecting the optimality of the solution.
2. Second, extending labels (T_i, C_i) in bucket p to successor j, from theorem 2, the new label (T_j, C_j) won't belong to bucket p. As consuming time is strictively positive, $T_j > T_i$, new label belongs to bucket q, which $q > p$. Thus new label is always extended to a larger index bucket. This property enable an increasing treatment order of buckets. But it won't grand that new labels are extended in a sequent increasing bucket index order. Figure 7.1 shows how buckets are generated and treated.

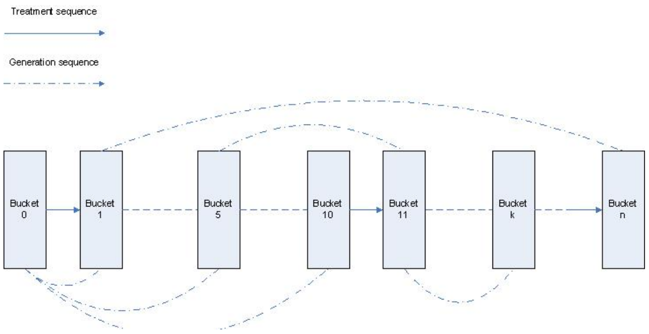


Figure 7.1: Bucket generating and treatment order

3. Third, from Theorem 2, as no inter-dominance improvement within a bucket, no specific treatment order of labels within a bucket. The implementation use a FIFO(Fist-In, First-Out) rule for storing and removing the labels to be treated.

Conclude, in basic NDCA, treating labels in an increasing service time order can be achieved by treating buckets in increasing index order. Following introduce calculation of bucket index.

The bucket index of label (T_i, C_i) is:

$$p = \lfloor T_i/m \rfloor \tag{7.1}$$

As we know the upper bound of time consuming T_{end}, which is the latest time back to deport, the maximum number of buckets is:

$$PMAX = \lfloor T_{end}/m \rfloor \tag{7.2}$$

The skeleton of basic NDCA is as following: as no time consuming and cost from start deport, initialize by adding label $(0, 0)$ of node 0 into the bucket 0. The algorithm starts treatment of labels from bucket 0, which has only one label. Next is to extend label $(0, 0)$ to all customer nodes. If new label meets the time window constraint of the corresponding successor, calculate new label's bucket index k using (7.1); add it into bucket k. After finishing treatment of bucket 0, go to bucket 1, and so on. If a bucket is empty, continue to the next bucket. The algorithm stops when finishing dealing with bucket $PMAX$.

Within a bucket, labels are grouped by nodes to which they belong. After treatment of one label group, the next group treatment is initiated. This procedure is repeated until all label groups have been treated. Figure 7.2 shows the structure of buckets, nodes and labels. Buckets are treated by increasing index order. Within each bucket, label groups are treated in an arbitrary order. Within each label group, labels are treated in an arbitrary order.

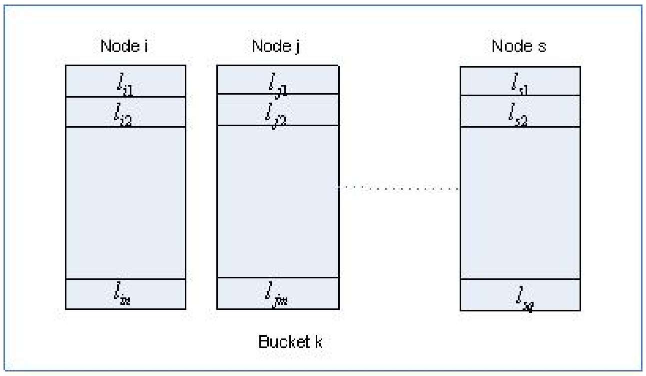
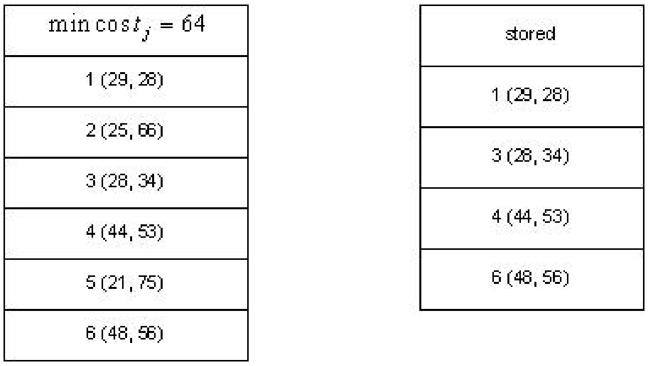


Figure 7.2: Bucket structure in basic NDCA

The basic NDCA, shown in figure A.1 brings improvement by introducing a dominance test for new extended labels. Each node $i \in N$ is associated a value: *mincost*$[i]$, which represents the minimun cost of currently treated labels of node i. Dominance test is done before adding new extended label (T_j, C_j) into a bucket. If C_j is less than *mincost*$[j]$, store the label into bucket; otherwise, discard it. This dominance test is excuted at destination node of all successors of a label being treated.

Following figure 7.3 is an example. On the left table, the first line is the current *mincost*$[j]$, below are new extended labels of node j, left is index to present for each label. On the right table show the stored new labels after dominance test.



$\min cost_j = 64$
1 (29, 28)
2 (25, 66)
3 (28, 34)
4 (44, 53)
5 (21, 75)
6 (48, 56)

stored
1 (29, 28)
3 (28, 34)
4 (44, 53)
6 (48, 56)

Figure 7.3: Dominance test at destination node

This minimal cost test only need cost comparation of two labels, without comparation of starting service time. The reason is: as the buckets are swept and the corresponding labels treated in an increasing time order, label earlier treated has earlier starting service time than a new extended label. This means the starting service time associated with current minimun cost label is always earlier than a new extended label. If new label of node j has greater or equal cost as *mincost*$[j]$, it is certainly dominated by the current minimun cost label; Thus, it is not necessary to keep the starting service time.

If a new extended label has smaller cost than *mincost*, it is only a candidate label, not necessarily an efficient label. As shown in figure 7.3, l_1, l_3, and l_4, l_6 are stored. Assume the bucket width is 10, using (7.1): l_1 and l_3 belong to bucket 2, l_4 and l_6 belong to bucket 4. Obviously, in bucket 4, $(44, 53) < (48, 56)$, l_6 is dominated by l_4. In bucket 2, both labels are efficient labels.

In figure A.1, see step 27, the *mincost* is updated after all labels have been treated within a bucket. As we grand that new labels are extended into a later index bucket, but not grand that new labels are generated in sequently increasing order, see figure 7.1. Suppose we update *mincost*$[j]$ at step 25 when a new label, i.e., (T_j^p, C_j^p), has smaller cost; assume in later, another new label is generated, i.e., (T_j^q, C_j^q). As the relation of T_j^q and T_j^p is not certain, it is not sufficient to do the dominance test by comparation $mincost[j]$ and C_j^q only. Thus in order to keep the validity of the minimal cost test, only update minimun cost after finishing treating all labels in a bucket. For example in figure 7.3, if update $mincost[j]$ to 28 once after treatment of $(29, 28)$, efficient l_3 won't be stored into bucket 2.

Figure A.1 present the basic NDCA. Step 3 initialize the minmum cost for all nodes as a super large number. Step 12 select a temperary label from a node group(labels seperated by group according to the node to which they belong). Step 10 to 26, labels are treated by using FIFO rule for storing into and removing from the buckets. step 13 to 19 extend label from i to successor j: 1) update starting service time; 2) check upper bound of time window constraint; 3) adjust starting service time by lower bound of time window; 4) update cost of path. Step 20 to 23 is the dominance test at the destination node. If new label is not dominated by minmum cost label, store it into calculated K^{th} bucket. Step 27 update $mincost[j]$ when finishing dealing with a bucket. Step 29 is a minmum test at back deport $N + 1$ to find the shortest path.

7.3 Dominance Test for the Original Label

The basic NCDA can be markedly improved by introduction of two additional state-dominance criterias. Each critera construct a dominance test at an original label before being extended to it's successors.

First dominance critera is a minimun cost test at an original node, which holds the same idea introduced in section 7.2. Once a label cost is equal or larger than the minmum cost of the node, we won't generate labels of its successors. See section 7.2 for details.

Second criteria introduce another dominance test at an origin node based upon a new order of storing labels within a bucket. Labels are stored in decreasing start service time order and increasing cost order in each group in a bucket, instead of FIFO rule used as treatment order in basic algorithm version. Before explain the dominance itself, we first disscuss insertion situations of the new label storing order.

Figure 7.4 show four possible situations of label insertion into a bucket. In (a) and (b), non-dominance occurs during insertion, new label $(50, 47)$ and $(57, 31)$ are stored orderly into bucket. In (c), new label is dominate by a stored one in the bucket, i.e., $(52, 35) < (55, 37)$. In (d), new label dominates an already stored label in the bucket, i.e., $(50, 33) < (52, 35)$.

This orderly insertion process is very likely an efficient labels domination test,

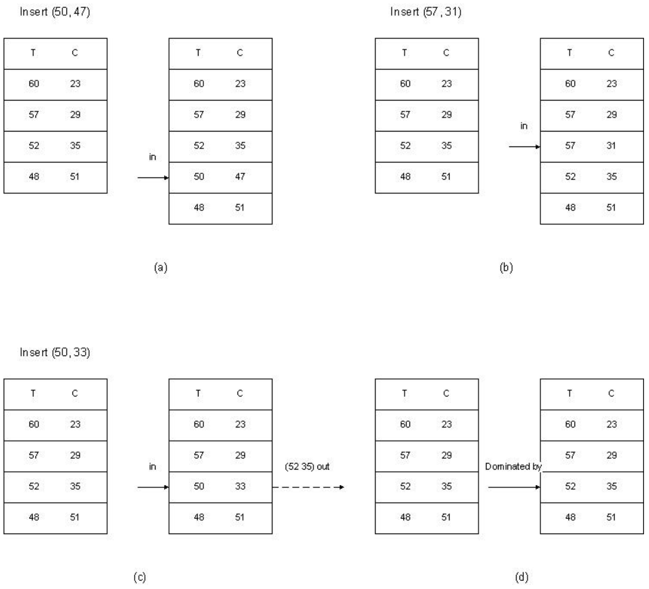


Figure 7.4: Label inserted in new storing order

except in situation (b), both label $(57, 29)$ and label $(57, 31)$ are stored. Once a label, either a new extended label or a stored label, is found to be dominated in a insertion, we discard it. Thus, this orderly label insertion itself lead to discarding quite a few unefficient labels. As a result, labels are stored in decreasing start service time order and increasing cost order.

The second dominance test is a previous time test based on above label storing order. For each node i in a bucket, introduce a previous time variable $previousT_i$: the start service time of most recent treated label of node i. We only need to treat a label (T_i, C_i), when $T_i < previousT_i$. Otherwise, we won't extend to its successors. As within a bucket, labels are sorted by decreasing cost order and increasing start service time order. The cost of previous label is no larger than the current treated label. Once the start service time of current label is no earlier than a previous one, it is certainly dominated by the previous label. This dominance test help discard unefficient labels described in figure 7.4 (b). Using previous time dominance test, unefficient label $(57, 31)$ won't be treated to extend to its successors, as it's start service time is no ealier than latest privious time 57.

The modified NDCA(NDCA II) resulting from the introduction of these two dominance tests at orginal label is stated in figure A.2.

Step 8 initialize the staring service time of previous treated label as a super large number. Step 14 apply the first dominance test by minmum cost label at destination node. Step 15 apply the second dominance test by previous service time variable. Step 20 store new label into K^{th} bucket in increasing service time order and decreasing cost order. This insertion procedure is likely a domination test of efficient labels. Step 22 update $previousT_i$ by service time of current label being treated after its treatment Step 32, as in basic algorithm, update the minmum cost after finishing treatment of all labels in a bucket.

7.4 Backward-looking Dominance Test

Besides the minimal cost dominance test, we introduce another backward-looking test at a destination node.

The backward-looking test is to compare a label cost with the minimal cost(s) in backward bucket(s). If bucket k stored label(s) of node i, there will be a minimal cost of i: $bckmincost_i^k$, It is a local minimal cost of node i, which is different from $mincost_i$. Before store a new extended label (T_j, C_j) into bucket k, first compare C_j with $bckmincost_j$ in each backward bucket $(k-1)$, $(k-2)$, $\cdots$, until the bucket contains lower bound of j's time window a_j, i.e., bucket k_j^{start}. In another word, k_j^{start} is the first feasible bucket index to store label of j. (7.3) show how to calculate k^{start} for a node:

$$k_i^{start} = \lfloor T_i/m \rfloor \tag{7.3}$$

A new label will be stored only if its cost is smaller than all those backward buckets minimal cost.

Following shows the validity of backward-looking dominance test. First, in a bucket, labels are stored in increasing cost order and decreasing starting service time order. Second, bucket definition and an its increasing index treatment order ensure that label stored in backward bucket has earlier start service time than a current treated label. Once the current label cost is larger than label of backward bucket, it is certainly dominated.

As the specific label storing order in a bucket discussed above, the first label of each label group is the minimal cost label in that bucket.

Following example, in figure 7.5, show six extended labels of node j. Assume the minimun cost is 64 and bucket width is 10. After minimun cost dominance check, four candidate labels left: l_1 and l_3 are stored in bucket 2; l_4 and l_6 are stored in bucket 4. It's obviously shown that l_6 and l_4 are already dominated by l_1. However the insertion can be avoided by the introduction of backward-looking dominance test, as l_4 and l_6 are both have larger cost than a backward bucket minimal cost 28, they won't be stord into bucket 4.

Even without the backward-looking dominance check, we still have a chance to discard l_4 and l_6 in a later stage. After finishing the treatement of bucket 2, $mincost_j$ is updated to 28. Thus during the minimun cost test at bucket 4, these two labels will be discarded as their cost are larger than 28.

However, backward-looking dominance test still has advantage in that it helps to dominate a candidate label in an earlier time, which prevent implementing a set of dominance check in order to orderly insert the label into a bucket. In above example, if l_4 and l_6 are dominated in an earlier stage, none time will consume on orderly insertion these two labels.

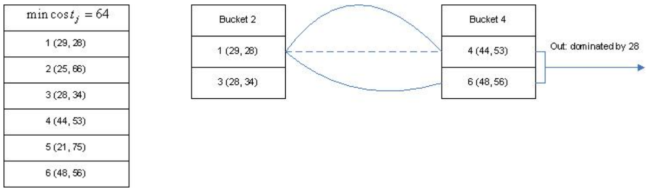


Figure 7.5: Backward-looking example

This dominance criterion is applied to all newly extended label at its destination node before insertion.

NDCA III, shown in figure A.3 is improved from NDCA II by introduction of backward-looking dominance test. In figure A.3 show that, step 22, the test is implemented after a minimun cost dominance test. Figure A.6 in appendix show the details of backward look test algorithm. Step 1 first calculate the first valid bucket for node j, $InBK$. Step 5 to 13, the while loop begin to compare the cost of label (T_j, C_j) with the minmum cost label in every precedence buckets, from bucket $(K-1)$ to bucket $InBk$. If C_j is smaller than all its precedence valid buckets, store (T_j, C_j) into buclet K; otherwise, discard the label.

Chapter 8

NDCA Adapting for Subproblem

The NDCA is expatiated based on SPPTW. The algorithm is also applicable for the one resource constrained shortest path problem(RCSPP).

To short we use NDCA refer to NDCAIII discussed in previous section.

8.1 Applying the NDCA in RCSPP

Time window can be considered as a time using resource constraint. $[a_i, b_i]$ can be explained in that at least $a_i - T_{start}$ time has been used before visiting customer i; and no more than $b_i - T_{start}$ time has been used when finishing service customer i.

In a capacity constraint SPP, each node has a same capacity window constraint,

$$[0, CAP] \tag{8.1}$$

CAP is maximum capacity of vehicle.

(8.1) means no more than CAP order is onload after visiting a customer. In another word, the onloading capacity should never excceed the maximum capacity of vehicle. In (8.1), the minimal onloading is 0. Obveriously, a vehicle could load nothing.

Above show SPPTW and one recourse RCSPP can be explained with each other, thus the NDCA can also be applied to find optimal solution of capacity constraint RCSPP.

8.2 Applying the NDCA in Subproblem

In our CG subproblem, one of the objective function is to find the shortest single path with capacity constraint and time window constraint. Obviously, NDCA is not sufficient to find the optimal solution of the single path in subproblem. The dominance test hasn't considered time consuming of path. A dominated path is still possible to be extended to the destination having less cost in the subproblem if it consumes less time. Less time consuming makes a path have more possibility of extension.
Thus the optimal single path is possibly dominated during the extension process in NDCA. However NDCA is still applicable in the subproblem:

1. the time window constraint is very loose in subproblem. Instead of having a tight time window for each node, all nodes share one big time window from T_{start} to T_{end}. This condition leads the subproblem won't be sensition to the time window constraint. Instead, the capacity constraint mainly affeacts the labels extending. Relaxing time consuming in the dominance test won't affect too much of the solution value;

2. instead of finding the optimal single path in the subproblem, it's only necessary to find negative reduced cost columns to insert into RMP. NDCA is an efficient algorithm to find the negative reduced cost single path with good quality.

Thus NDCA is sufficiently applicable in the subproblem of finding insertion columns. However, if the destination label list doesn't contain any negative cost path, NDCA can't guarantee that no more negative reduced cost single path exists. This situation will be solved in another algorithm CDSPA discussed in section

8.3 Adapting the NDCA in Subproblem

To apply NDCA for capacity RCSPP, some modification is needed. We use NDCA' to indicate the modified NDCA.

In NDCA' for RCSPP, label is presended as (cap_i, C_i), cap_i is the capacity used finishing service of customer i, C_i is the path cost finishing service of customer i. The first label of depot is initialized in (8.2). (8.3), (8.5) are label extension calculation:

$$(cap_0, c_0) = (0, 0) \tag{8.2}$$

$$cap_{i^l} = cap_{i^{l-1}} + q_{i^{l-1}} \qquad l = 1, \cdots, L \tag{8.3}$$

$$C_{i^l} = C_{i^{l-1}} + dist^{pik}_{i^{l-1}i^l} + ser_{i^l} + u_{i^l} \qquad l = 1, \cdots, L \tag{8.4}$$

If an extended label's capacity exceed CAP, discard it.

During the label extension, the time window constraint should be checked.

$$T_{i^l} = T_{i^{l-1}} + dist^{pik}_{i^{l-1}i^l} + ser_{i^l} \qquad l = 1, \cdots, L \tag{8.5}$$

T_i is the time consuming finishing service of customer i. If an extended label's time consuming excceed the maximum working time,

$$pickup\ single\ path:\ T_i \geq 0.25T_{max} \tag{8.6}$$

$$delivery\ single\ path:\ T_i \geq 0.75T_{max} \tag{8.7}$$

, discard it.

Above is extension calculation of pickup subpath, the calculation of delivery subpath has the same constructure.

Path capacity usage and path cost are used in dominance test. As each customer order is strictively positive, capacity usage will also be strictively lexicographic positive. In NDCA', bucket width is defined by the minimun customer order, see (8.8). The bucket index of a label is calculated using capacity usage, see (8.9), (8.10). (8.11) is calculation for maximum bucket index. Treating buckets in increasing index order leads to a treatment order of labels in increasing lexicographic order of capacity usage. Within each bucket, labels are treated in decreasing capacity usage order and increasing cost order.

$$m = min\ q_i \qquad \forall i \in N \tag{8.8}$$

$$k = \lfloor q/m \rfloor + 1 \tag{8.9}$$

$$k_{start} = \lfloor 0/m \rfloor \tag{8.10}$$

$$k_{max} = \lfloor CAP/m \rfloor + 1 \tag{8.11}$$

There is a difference in NDCA' from NDCA during the inital label extending process. At the begining in both NDCA' and NDCA, only one label of node 0, $(0, 0)$, in bucket 0. After extension of depot in NDCA, all newly generated labels will be stored into a forward buckets. However, in NDCA', as deport has no order, $q_0 = 0$, using (8.3), each generated label still has none capacity usage. Thus after extension of $(0, 0)$, all generated labels will be stored in bucket 0.

Chapter 9

CDSPA for Subproblem

NDCA is applied to find the negative reduced cost single path containing one type of job, either pickup orders or delivery orders. A combination path is distinguish from single path. It consists both pickup and delivery job, which is possible to achieve saving more time in cross-docking routing. In this section, we will develop an algorithm to find negative reduced cost combination path in the subproblem. The solution is developed based on the NDCA discussed in previous sections. Additional modification is made to satisfy new constraint in the combination situation. To distinguish from NDCA, we use ”cross-docking shortest path algorithm”, shortly CDSPA, to indicate the new algorithm.

9.1 Combination Path and Connective Path

In this section, we analyse the property of connective path, and how it can save time. Then we give out the saving time calculation of a connective path. First we give the terminologe used in the discussion.

Definition 3 (Combination path) *Combination path is a path covering both pickup and delivery order(s).*

Definition 4 (Connective delivery) *If a customer's order is picked up and delivered by a same vehicle, without being offloaded and uploaded at the depot, it is a connective delivery, or a connective order.*

Definition 5 (Connective path) *A combination path is a connective path if it contains at least one connective delivery.*

Property 1 (Combination path) *A connective path is a combination path; a combination path is not necessarily a connective path.*

Obviously, not each orders on a connective path is necessary to be a connective order.

Following is a study of these two path types. To simplicity, we analyse a situation of one customer only, i.e., customer A, A_p and A_d are the pickup and delivery node, respectively.

The figure 9.1(a) show a none connective order delivery process. A is a none connective delivery order, delivered from A_p to A_d by a combination path p_1.

A is uploaded at A_p on vehicle v and delivered back to depot. A is first offloaded at depot, then uploaded onto vehicle w and delivered to A_d. Finally, A is offloaded at A_d.

The figure 9.1(b) show a connective order delivery process. A is a connective delivery order, delivered from A_p to A_d by a connective path p_2.

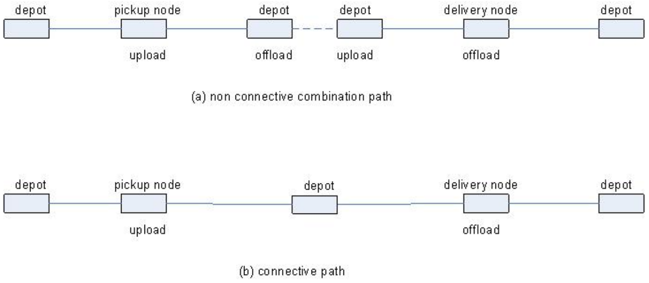


Figure 9.1: Delivery process of a none connective delivery

Comparing figure 9.1(a) and 9.1(b), the routing distance of p_1 and p_2 is equal, and each path docks two times at depot, one time at A_d and one time at A_d. p_1 has uploaded or offloaded A four times; however, p_2 has only two times. Thus, p_2 is able to save time from skipping an offload and an upload of order at depot.

To conclude from above instance, a connective path achieves saving more loading time than a combination path from connective deliverys. The reason is by beging picked up and delivered on a same vehicle, a connective order won't be offloaded or uploaded at the depot, in which the consuming time of offloading and uploading will be saved.

Only a connective path can save loading time. A combination path, but none connective path, will not save loading time. Below is the general calculation of saving time of connective path. First we give the definition of symbols:

Assume a combination path p, PI and DE present its pickup nodes set and delivery nodes set, respectively. CON is the connective delivery set of p: $CON = PI \cap DE$. If nodes set CON is not empty, p is a connective path.

If nodes set $CON = PI$ or $CON = DE$, p is a super connective path.

Definition 6 (Super connective path) *If all pickup orders, or all delivery orders of a combination path are connective orders, the combination path is a super connective path.*

Property 2 (Super connective path) *The connective delivery set of a super connective path is a smaller set of pickups set and deliverys set:*

$$CON = \{ \begin{matrix} PI\ if PI \subseteq DE \\ DE\ if DE \subseteq PI \end{matrix}$$

One connective delivery can make saving time:

$$saving_i = 2loading * q_i \qquad i \in CON \tag{9.1}$$

The saving time of a connective path p is the sum of all its connective orders' saving time:

$$saving = \sum_{i \in CON} 2loading * q_i \tag{9.2}$$

9.2 CDSPA Solution Method

A combination path contains both pickup and delivery jobs. These jobs are not intersected: a combination path can not begin its deliverys until finishing all its pickups and back to the depot.

The initial delivery onloading status, $initload_{del}$, is capacity usage before a vehicle start to upload new deliverys at the depot, which can be empty(zero) or not. Additional uploading delivery capacity, $addiload_{del}$, is the capacity of how much additional orders can be uploaded at depot for delivery.

For a none connective combination path, all the pickup orders will be offloaded at depot: $initload_{del} = 0$, $addiload_{del} = CAP$. At most CAP additional deliverys can be uploaded.

For a connective path, the connective delivery orders won't be offloaded at depot.

$$initload_{del} = \sum_{i \in CON} q_i \tag{9.3}$$

$$addiload_{del} = CAP - \sum_{i \in CON} q_i \tag{9.4}$$

Both none connective path and connective path meet the following constraints:

$$initload_{del} + addiload_{del} \leq CAP \tag{9.5}$$

$$\sum_{i \in CON} q_i + addiload_{del} \leq CAP \tag{9.6}$$

$$\sum_{i \in DE} q_i \leq CAP \tag{9.7}$$

For non connective path, $\sum_{i \in CON} q_i$ is zero.

(9.5) to (9.7) indicate that neither subpath will excceed the onloading capacity. A combination path can be considered as a composition of two independent subpaths each from one subpart: a pickup subpath *pik* and a delivery subpath *del*. See figure 9.2.

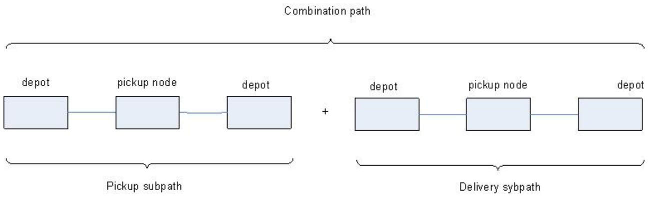


Figure 9.2: Seperate delivery process of a combination path

Refer to the time window constraint (5.31), (5.32) in section **??**, any two subpaths from each subpart could be contacted without violating the time window constraint:

$$T_{pik} \leq 0.25T_{max}$$

$$T_{del} \leq 0.75T_{max}$$

$$T_{pik} + T_{del} \leq 0.25T_{max} + 0.75T_{max} = T_{max}$$

Thus the combination path will certainly subject to the capacity constraint, time window constraint and flow conservation constarints.

During the indpendent subpath generation processm we've assumed every order is a non-connective delivery and will be offloaded and uploaded at the depot. After the combination operation, an order becomes a connective delivery if it exists on both subpaths. Therefore, the connective delivery's loading time at the depot should be deducted from the combination path time consuming and cost:

$$T_{comb} = T_{pik} + T_{del} - saving \tag{9.8}$$

$$C_{comb} = C_{pik} + C_{del} - saving \tag{9.9}$$

T_{pik} and T_{del} are time consuming in pickup and delivery subpath respectively. C_{pik} and C_{del} are the cost in pickup and delivery subpath, respectively. *saving* is calculated by (9.2).

Conclude, a combination path can be constructed in following steps:

1. generate a subpath *pik* in pickup subpart, *pik* is a single path;
2. generate a subpath *del* in delivery subpart, *del* is a single path;
3. construct a combination path by contacting *pik*, *del*, $p_{comb} = \{pik, del\}$;
4. find all connective deliverys on p_{comb};
5. the time consuming and cost of *pcomb* are calculated using (9.9), (9.8), respectively.

Now we can outline the solution nethod of CDSPA:

1. generate a candidate set of pickup subpaths, $PIKS$;
2. generate a candidate set of delivery subpaths, $DELS$;
3. build combination paths using subpaths from $PIKS$ and $DELS$;
4. collect combination paths with negative cost.

Each subpath building a combination path is alled the *pickup subpath* and the *delivery subpath*, respectively.

The algorithm should be efficient to generate these two candidate sets. As in the NDCA, only the subpaths which are possible to construct a negative cost combination path will be extended in the CDSPA. In another word, each candidate set is generated as tiny as possible, but still sufficient to constain the required subpaths.

9.3 Prominent Label

The NDCA will keep any efficient label and extend it toward the destination node. If a label is dominated by another label, it will be dropped. This section we will analyse the dominance situation to decide when to keep a dominated label.

A dominated label will be dropped as it has neither advantage in less-cost nor in less capacity usage. A destination label extended from a dominated label will always be dominated by the one extended from the dominating label. Following we'll study whether a dominated label has advantage in construction a combination label.

Assume label p_1 is dominated by label p_2, $(cap_1, C_1) > (cap_2, C_2)$. Now p_1 and p_2 both build a combination path with label p_3,

$$p_{comb1} = \{p_1, P_3\}$$

$$p_{comb2} = \{p_2, P_3\}$$

The cost of p_{comb1} and p_{comb2},

$$C_{comb1} = C_1 + C_3 - saving_{comb1}$$

$$C_{comb2} = C_2 + C_3 - saving_{comb2}$$

If $C_{comb1} < C_{comb2}$, p_1 still has advantage in having less-combination cost than p_2:

$$C_{comb1} < C_{comb2} \tag{9.10}$$

$$C_1 + C_3 - saving_{comb1} < C_2 + C_3 - saving_{comb2}$$

$$C_1 - saving_{comb1} < C_2 - saving_{comb2}$$

$$C_1 > C_2$$

Therefore in order to satisfy condition (, the saving condition will be

$$saving_{comb1} > saving_{comb2} \tag{9.11}$$

(9.11) shows a dominated label still has advantage of having less-combination cost against the dominating label only if it can construct a combination path to save more time from the connective deliverys. Following we study the property of connective deliverys in a connective path.

A connective path has at least one connective delivery:

$$CON = PI \cap DE \neq \emptyset \tag{9.12}$$

$$CON \subseteq PI \tag{9.13}$$

$$CON \subseteq DE \tag{9.14}$$

The saving time will satisfy following constraints:

$$saving = 2 * load * \sum_{i \in CON} q_i \leq 2 * load * \sum_{i \in PI} q_i \tag{9.15}$$

$$saving = 2 * load * \sum_{i \in CON} q_i \leq 2 * load * \sum_{i \in DE} q_i \tag{9.16}$$

cap_{pik}, cap_{del} is the onloading capcacity of path pik, del, respectively:

$$cap_{pik} = \sum_{i \in PI} q_i$$

$$cap_{del} = \sum_{i \in DE} q_i$$

Constraints (9.15), (9.16) can be written as:

$$saving \leq 2 * load * cap_{pik} \tag{9.17}$$

$$saving \leq 2 * load * cap_{del} \tag{9.18}$$

(9.17), (9.18) leads to the maximal saving time theorem:

Theorem 3 *The maximal saving time of a connective path is the total loading time of the subpath with less capacity. The loading time constains time consuming of uploading and offloading all orders of the subpath.*

$$saving_{max} = 2 * load * min\{cap_{pik}, cap_{del}\} \tag{9.19}$$

Obviously, the saving time of a connective path is always larger than zero.

Using (9.19) in the connective path cost (**??**), get:

$$C_{pik} + C_{del} - saving_{max} \leq C_{pik} + C_{del} - saving = C_{\{pik,del\}} \tag{9.20}$$

Using (9.17), (9.18) in (9.20),

$$C_{pik} + C_{del} - 2 * load * cap_{pik} \leq C_{\{pik,del\}} \tag{9.21}$$

$$C_{pik} + C_{del} - 2 * load * cap_{del} \leq C_{\{pik,del\}} \tag{9.22}$$

Above (9.21), (9.22) leads to the minimal combination path cost theorem:

Theorem 4 *A connective path will not have less cost than the sum of each subpath cost substract the loading time of either subpath.*

Remove C_{del} in (9.21),

$$C_{pik} - 2 * load * cap_{pik} \leq C_{pik} - saving \tag{9.23}$$

(9.23) leads to the combination path cost theorem:

Theorem 5 *Subpaths p_1 and p_2 are in the same subpart, if*

$$C_1 < C_2 - 2 * load * cap_2$$

, then any combination path formed by p_1 will always have less cost than a combination path formed by p_2.

Prove:

Assume pik' and pik are both from pickup subpart, and satisfy condition of theorem5,

$$C_{pik'} < C_{pik} - 2 * load * cap_{pik}$$

Now pik' and pik build a combination path with any subpath del of the delivery subpart, respectively:

$$C_{\{pik',del\}} \leq C_{pik'} + C_{del} < C_{pik} - 2 * load * cap_{pik} + C_{del} \leq C_{\{pik,del\}}$$

(5) got proved.

Theorem 5 draws the first rule of dropping off a dominated label:

Rule 1 (Dropping rule) *If p_2 is dominated by p_1, and the cost satisfy the following condition*

$$C_1 < C_2 - 2 * load * cap_2$$

we will discard p_2.

As from theorem 5, any combination path formed by dominated path p_2 is impossible to have less cost than p_1. Thus p_1 has none advantage in less-combination cost, either. It will be dropped.

Now we analyse the reversed situation of (5): $C_1 \geq C_2 - 2 * load * CAP_2$. Assume pik' dominate pik, and satisfy the reversed condition in Theorem5,

$$C_{pik'} \geq C_{pik} - 2 * load * cap_{pik}$$

Now pik' and pik build a combination path with any subpath del of the delivery subpart, respectively.

$$C_{pik'} + C_{del} \geq C_{pik} - 2 * load * cap_{pik} + C_{del}$$

as left hand side

$$C_{pik'} + C_{del} \geq C_{\{pik',del\}}$$

and right hand side

$$C_{pik} - 2 * load * cap_{pik} + C_{del} \leq C_{\{pik,del\}}$$

Conclusion: if p_1 dominate p_2, and $C_1 \geq C_2 - 2 * load * CAP_2$, the relationship of combination cost formed by p_1 and p_2 is not certain.

Following we analyse the condition of when to store and extend a dominated label.

Assume pik is dominated by pik', if we can find a subpath del^o in another part which makes path $\{pik, del^o\}$ have smaller cost than path $\{pik', del^o\}$, pik still has advantage in having less-combination cost against pik':

$$C_{\{pik,del^o\}} < C_{\{pik',del^o\}}$$

$$C_{pik} + C_{del^o} - saving_{\{pik,del^o\}} < C_{pik'} + C_{del^o} - saving_{\{pik',del^o\}}$$

$$C_{pik} - saving_{\{pik,del^o\}} < C_{pik'} - saving_{\{pik',del^o\}}$$

$$C_{pik} - C_{pik'} < saving_{\{pik,del^o\}} - saving_{\{pik',del^o\}} \tag{9.24}$$

(9.24) leads to the first keeping rules:

Rule 2 (Keeping rule) *p_1 is dominated by p_2. In the construction of a connective path path with p_3, if p_1 archieves saving more cost from the connective deliverys than the more cost it has,*

$$C_1 - C_2 < saving_{\{1,3\}} - saving_{\{2,3\}}$$

p_1 is able to build a combination path having less combination cost than p_2, it will be stored.

In order to prove that there exists such a path del^o satisfy the keeping Rule 2, we'll study the RHS of Rule 2: the saving difference bwtween a dominance pair.

As before, PI, PI' are nodes set covered by pik, pik', respectively. SH is the nodes set covered by both PI and PI':

$$SH = PI \cap PI'$$

$$PI_{ex} = PI - SH$$

$$PI'_{ex} = PI' - SH$$

$$PI_{ex} \cap PI'_{ex} = \emptyset$$

PI_{ex} and PI'_{ex} are the rest nodes set in PI, PI', respectively. We define them as exclusive nodes set of pik, pik', respectively.

Now pik' and pik both build a combination path with subpath del, respectively. DE is the nodes set of del. The connective delivery nodes set will be:

$$CON_{\{pik,del\}} = PI \cap DE = \{PI_{ex} \cup SH\} \cap DE = \{PI_{ex} \cap DE^o\} \cup \{SH \cap DE\}$$

$$CON_{\{pik',del\}} = PI' \cap DE = \{PI'_{ex} \cup SH\} \cap DE = \{PI'_{ex} \cap DE^o\} \cup \{SH \cap DE\}$$

The saving of connective deliverys will be:

$$saving_{\{pik,del\}} = 2load \sum_{i \in \{PI_{ex} \cap DE\}} q_i + 2load \sum_{i \in \{SH \cap DE\}} q_i \tag{9.25}$$

$$saving_{\{pik',del\}} = 2load \sum_{i \in \{PI'_{ex} \cap DE\}} q_i + 2load \sum_{i \in \{SH \cap DE\}} q_i \tag{9.26}$$

(9.25)-(9.26), we get

$$saving_{\{pik,del\}} - saving_{\{pik',del\}} = 2load \sum_{i \in \{PI_{ex} \cap DE\}} q_i - 2load \sum_{i \in \{PI'_{ex} \cap DE\}} q_i \tag{9.27}$$

(9.27) indicate that the saving difference between the two combination paths will not be affected by the shared nodes set SH. This can be explained as: if an order of the shared set is a connective delivery on *pik*, it will also be a connective delivery on *pik*′; if an order of the shared set is not a connective delivery on *pik*, it will not be a connective delivery on *pik*′, either.

In order to satisfy the saving condition in Rule 2, the method is to increase the RHS. From the exclusive nodes set definition, $PI_{ex} \cap PI'_{ex} = \emptyset$, we could assume a path del^o that makes every node of PI_{ex} as a connective delivery; and none of PI'_{ex}. (9.27) can be written as:

$$saving_{\{pik,del^o\}} - saving_{\{pik',del^o\}} = 2load \sum_{i \in PI_{ex}} q_i \tag{9.28}$$

In another word, by combination with path del^o, *pik* will achieve the maximal saving difference against *pik*′.

$$2load \sum_{i \in PI_{ex}} q_i \geq saving_{\{pik,del\}} - saving_{\{pik',del\}} \tag{9.29}$$

$$del \in DELS$$

If LHS of (9.29) meets the saving condition in Rule 2,

$$2load \sum_{i \in PI_{ex}} q_i > C_{pik} - C_{pik'} \tag{9.30}$$

that is: the more saving of {*pik*, del^o} against {*pik*′, del^o} could make compensation for the more cost of *pik* against *pik*′. Otherwise, we could not find a path del^o to meet the saving condition in Rule 2. As (9.29) show that the LHS is the maximal more saving *pik* could achieve, no combination path having more saving exists. Thus, (9.30) draw the second keeping rule of dominated label:

Rule 3 (Keeping rule) *p_1 and p_2 are subparts in the same subpart. p_2 is dominated by p_1. If they meet the following condition:*

$$C_2 - C_1 < 2load \sum_{i \in EX_{p_2}} q_i$$

EX_{p_2} is the exclusive nodes set of p_2 against p_1.

The dominated p_2 is able to build a combination path having less combination cost than p_1. It will be stored. Otherwise, it will be dropped.

Following we give the definitions of *prominent label* and *master label*.

Definition 7 (Prominent Label and Master Label) *Label A and B are in the same subpart, A is dominated by B. If A and B meet the saving condition in Rule 2, then A is a prominent label of B, B is a master label of A.*

Here *prominent* means a dominated label is prominent to have less combination cost than the dominating label.

A prominent label is build up from a pair of dominating label and dominated label. A master label is a dominating label; and a prominent label is certainly dominated by the master label. As dominated label, a prominent label is stick to a master label: a master label can have none or more than one prominent labels; a prominent label has only one master label.

For consistent discussion, we extended the master label definition:

Definition 8 (Master label) *An undominated efficient label is a master label.*

Now a master label can have none prominent label. The efficient labels in NDCA are master labels.

Prominenet labels are stored in the prominent list of the master label, see figure 9.3.

The CDSPA keeps the framework of the NDCA to collect efficient labels and applys Rule 2 and Rule 3 to collect prominent labels. We can now give a complete definition of the candidate set: the candidate subpaths set is composed of master labels and the prominent labels.

In order to collect prominent labels, we have to calculate the exclusive order capacity of a dominated label by comparing each order with the dominating

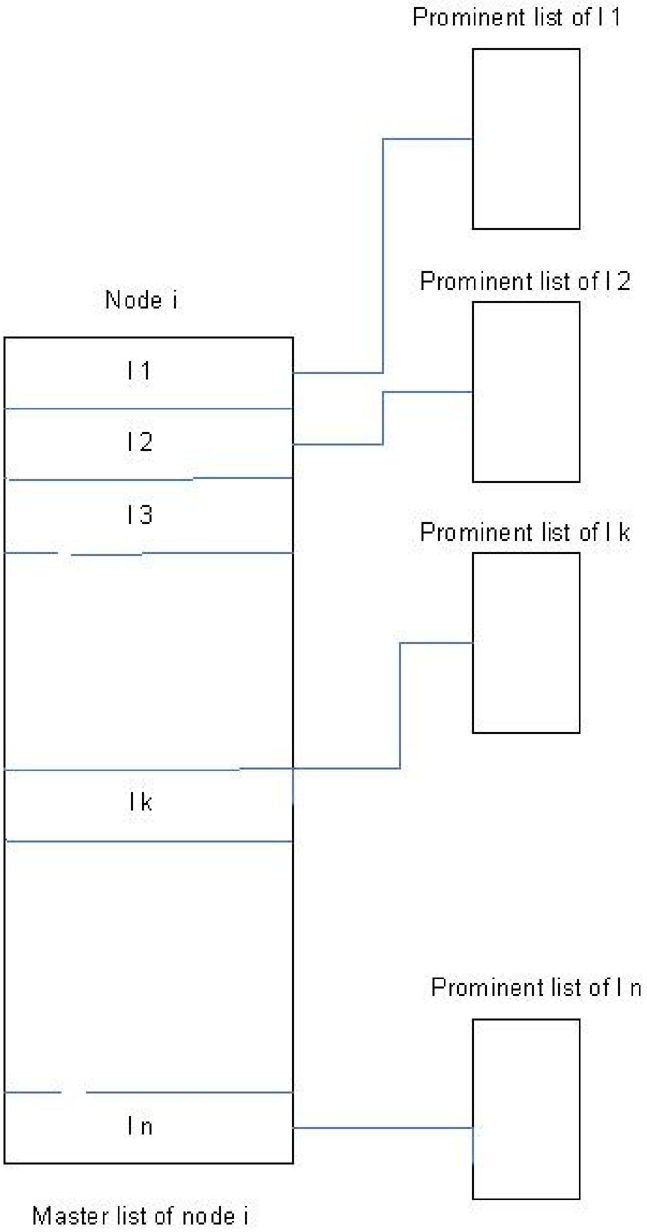


Figure 9.3: Prominent labels storeage

label, which is a time consuming operation. Observe tge saving condition, the result is decided by two factors: more cost(LHS); and more saving(RHS). In the next two sections, we'll introduce two method to short cut the operation time using in judging a prominent label. Each is developed according to one of the factor in the saving condition.

Section 9.4 introduce a new label presentation helping to avoid the prominence test in some situtation; section 9.5 introduce a new fast keeping rule without calculating the exclusive order capacity.

9.4 New Objective Function

In this section, we'll study a new label presentation which helps to avoid a prominence test by making a dominated label become an efficient label.

Update Objective Function

Our objective is to minimize the total time consuming of delivery orders. Time consuming contains three parts: travel distance, load order, dock at node. Assume the optimal solution paths set is K and $|K|$ is the number of paths in set K. The objective function can be written as

$$MIN\ Z = \sum_{k \in K} dist_k + \sum_{k \in K} load_k + \sum_{k \in K} dock_k \tag{9.31}$$

First, we discuss the total time of loading orders:

$$Total\ loading\ time = \sum_{k \in K} load_k \tag{9.32}$$

An order is either a connective delivery or none connective delivery, and the path set K covers each customer(one pickup node and one delivery node) once, (9.32) can be written as:

$$\begin{aligned} Total\ loading\ time &= \sum_{i \in N} loadorder_i && (9.33) \\ &= \sum_{i \in CON} loadorder_i + \sum_{i \in NCON} loadorder_i && (9.34) \end{aligned}$$

$loadorder_i$ is time consuming of loading an order i. N is customers set, CON is connective delivery set, $NCON$ is none connective delivery set. $N = CON \cup NCON$.

For a none connective delivery,

$$loadorder_i = 4loadingq_i \tag{9.35}$$

For a connective delivery,

$$loadorder_i = 2loadingq_i \tag{9.36}$$

Use (9.35), (9.36) into (9.37), we get

$$\begin{aligned} Total\ loading\ time &= 4loading \sum_{i \in NCON} q_i + 2loading \sum_{i \in CON} q_i & (9.37) \\ &= 4loading \sum_{i \in N} q_i - 2loading \sum_{i \in CON} q_i & (9.38) \end{aligned}$$

(9.37) shows the variety of total loading time is only affected by the connective delivery set. More connective deliverys save more loading time so as to reduce the cost of objective function.

Second, we discuss the total time of docking at nodes.

$$Total\ docking\ time = \sum_{k \in K} dock_k \tag{9.39}$$

Docking happens at two types of nodes: a customer node(pickup node or delivery node) and the depot. The optimal solution path set K covers each customer once, both pickup node and delivery node, thus the total docking time spending at customer nodes is fixed:

$$Total\ customer\ docking\ time = \sum_{k \in K} 2docking|N| \tag{9.40}$$

The number 2 means, for each customer, docking at the pickup node and the delivery node each one time.

The total depot docking time is decided by

$$Total\ depot\ docking\ time = \sum_{k \in K} docking_k \tag{9.41}$$

If a path is a combination path, it docks at depot two times: each sub path docks once. If a path is a single path, it docks at depot one time. Using (9.40), (9.41) into (9.39), we get

$$Total\ docking\ time = 2docking|N| + \sum_{k \in K} docking_k \tag{9.42}$$

(9.42) shows the variety of total docking time is only affected by the optmial solution paths number and type.

Now, using (9.37), (9.42) into (9.31), the objective function is changed as following:

$$
\begin{aligned}
MIN\ Z &= \sum_{k\in K} dist_k + A - 2loading \sum_{i\in CON} q_i + B + \sum_{k\in K} docking_k \qquad (9.43)\\
&= \sum_{k\in K} dist_k - 2loading \sum_{i\in CON} q_i + \sum_{k\in K} docking_k + A + B \qquad (9.44)\\
A &= 4loading \sum_{i\in N} q_i \qquad (9.45)\\
B &= 2docking|N| \qquad (9.46)
\end{aligned}
$$

Remove the scalar A and B from (9.43),

$$
\begin{aligned}
MIN\ Z' &= \sum_{k\in K} dist_k - 2loading \sum_{i\in CON} q_i + \sum_{k\in K} docking_k \qquad (9.47)\\
&= \sum_{k\in K} dist_k - 2loading \sum_{k\in K} Cap_k^{con} + \sum_{k\in K} docking_k \qquad (9.48)\\
&= \sum_{k\in K} dist_k - 2loading * Cap_k^{con} + docking_k \qquad (9.49)\\
Cap_k^{con} &= \sum_{i\in CON_k} q_i \qquad (9.50)
\end{aligned}
$$

CON_k is the connective delivery set of path k.

(9.47) shows three features in fact affect the objective fucntion:

1. total traveling distance;
2. total connective delivery capacity;
3. total solution path number .

Observe (9.47), we can rewrite each path cost C'.

If a solution path k is a non combination path:

$$C'_k = dist + docking \qquad (9.51)$$

If a solution path k is a combination path composed by k_{pik} and k_{del}:

$$
\begin{aligned}
C'_k &= dist_{k_{pik}} + dist_{k_{del}} + 2docking - 2loading * Cap_k^{con} \qquad (9.52)\\
&= C'_{k_{pik}} + C'_{k_{del}} - 2loading * Cap_{\{pik,del\}}^{con} \qquad (9.53)
\end{aligned}
$$

(9.52) indicate that a combination cost is still the sum of each subpath cost subtracting the connective order loadint time.

Thus we can use the traveling distance as the label cost in the label presenting in NDCA and CDSPA:

$$label\ of\ node\ i : (cap, dist)$$

cap is onloading order capacity arriving node *i*, *dist* is the traveling distance arriving node *i*. As each subpath consumes one docking time, exclude the docking time in label cost(9.51) won't affect the dominance and prominance testing. Once a label is extended back into depot, the final cost of the label is the distance time plus another docking time. During a combination operation, a combination path cost is the sum of each subpath cost substract the saving time from connective deliverys.

Next section, we show the advantage of using the new objective function (9.47) and new label presentation.

New Label Presentation

Figure 9.4 show two path p_1 and p_2,

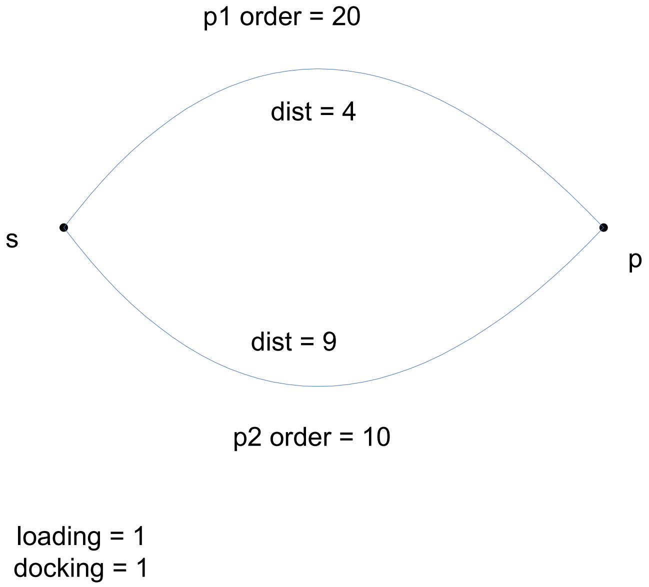


Figure 9.4: Label dominance condition

Using (cap, C) to present label p_1 and p_2:

$$p_1 : (20, 25)$$

$$p_2 : (10, 20)$$

P_1 is dominated by p_2, we need a prominence test to check whether p_1 is a prominent label of p_2. However, if Using $(cap, dist)$ to present label p_1 and p_2:

$$p_1 : (20, 4)$$

$$p_2 : (10, 9)$$

P_1 is not dominated by p_2 anymore, both p_1 and p_2 are efficient labels and will be stored. The reason is p_1 has a larger capacity consuming more loading time. It has more cost using (cap, C). Once change to $(cap, dist)$, the advantage of short distance of p_1 stands out. In later situation, a prominent test is unnecessary in which saves the operation of exclusive order calculation.

Now the inital label status and extension rule is changed as following:

$$label\ at\ start\ depot : (0, 0) \tag{9.54}$$

Extend label from node i to node j:

$$cap_j = cap_i + q_j \tag{9.55}$$

$$C_j = C_i + dist_{ij} - u_j \tag{9.56}$$

$$T_j = T_i + dist_{ij} + ser_j \tag{9.57}$$

In (9.55), if $cap_j > CAP$, drop the infeasible extended label. In (9.57), if T_j excceed the maximum working hours ($0.25T_{max}$ for pickup path; $0.75T_{max}$ for delivery path), drop the infeasible extended label.

Conclusion: by rewritting the objective function in form (, the new label presentation can be used, in which the three real factors affecting the objective function stand out: 1)traveling distance; 2) connective deliverys' capacity; 3) the solution paths type and number.

Through using the new label presentation, efficient labels are stored avoiding any prominence test, in which the operation time is saved.

9.5 Fast Dominance Test

In the saving condition, the other factor is the more saving time of dominated label. In this section, we'll study the property of the more saving time.

Assume label p_1 is dominated by label p_2. I_1, I_2 are nodes set covered by p_1, p_2 respectively. The more capacity of p_1 against p_2 is presented as Δcap; the exclusive order capacity of p_1 against p_2 is presented as cap^{ex},

$$\Delta cap = cap_1 - cap_2$$

The relationship between Δcap and cap^{ex} is analysed in following three situations.

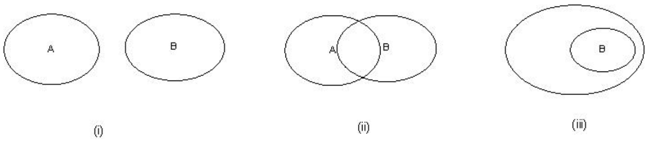


Figure 9.5: Three situations of covering nodes set

i) $I_1 \cap I_2 = \emptyset$, as shown in figure **??**(a).

$$\Delta cap = A$$

$$cap^{ex} = A - B$$

$$\Delta cap < cap^{ex}$$

In the situation p_1 and p_2 have no shared node, p_1's exclusive order is larger than Δcap.

ii) $I_1 \cap I_2 \neq \emptyset$ and $I_1 \cup I_2 \neq I_1$, as shown in figure 9.5(b).

$$\Delta cap = (A + C) - (B + C) = A - B$$

$$cap^{ex} = A$$

$$\Delta cap < cap^{ex}$$

In the situation p_1 and p_2 have shared nodes; and p_1 and p_2 both have exclusive order, p_1's exclusive order is larger than Δcap.

iii) $I_1 \cup I_2 = I_1$, as shown in figure 9.5(c).

$$\Delta cap = A$$

$$cap^{ex} = A$$

$$\Delta cap = cap^{ex}$$

In the situation p_1 covers all the nodes in p_2, p_1's exclusive order is equal to Δcap.

From (i) to (iii), we conclude the property of cap^{ex} as:

Theorem 6 *Subpath p_1 and p_2 are in the same subpart, p_1 is dominated by p_2. The exclusive order capacity of p_1 is larger or equal than the more capacity of of p_1 against p_2:*

$$\Delta cap = cap_1 - cap_2 \leq cap_1^{ex}$$

cap_1^{ex} is p_1's exclusive order against p_2.

Above property provides a new saving rule to speed up the prominence test.

Rule 4 *Subpath p_1 and p_2 are in the same subpart, p_1 is dominated by p_2. If*

$$C_1 - C_2 < 2loading(cap_1 - cap_2) \tag{9.58}$$

p_1 is a prominent label of p_1.

Prove:

From Theorem 6, $cap_1 - cap_2 \leq cap_1^{ex}$

$$2loading(cap_1 - cap_2) \leq 2loading * cap_1^{ex} \tag{9.59}$$

Using (9.59) into (9.58),

$$C_1 - C_2 < 2loading(cap_1 - cap_2) \leq 2loading * cap_1^{ex}{}_1 \tag{9.60}$$

(9.60) satisfied Rule3, p_1 is a prominent label of p_2.

Got proved.

If a dominated label satisfy condition in Rule 4, it certainly becomes a prominent label of the dominated label. Otherwise, an additional prominent test is necessary. Rule 4 extremely speeds up the opertaion of prominence test by avoiding the calculation of the exclusive order capacity.

Now we can conclude the complete procedure of a prominence test of a dominance pair: assume label p is dominated by label q:

1. Using Rule 1, if condition satisfied, go to step (4); otherwise, go to step (2);
2. Using Rule 4, if condition satisfied, go to step (5); otherwise, go to step (3);
3. Using Rule 3, if condition satisfied, go to step (5); otherwise, to to step (4);
4. p is not a prominent label, finish.
5. p is a prominent label, finish.

This test procedure delay the exclusive order calculation at a later step in order to save operation time.

9.6 Master Label Extension

Master label extension is much like the label extension described in section 9.4. A difference is once to extend a master label, its prominent label also needs to be extended.

Theorem 7 *Extend a master label and its prominent label to a same node, the new extended pair still holds prominence relationship.*

Prove:

For simplicity, assume master label l has only one prominent label l_p. l and l_p are labels of node i. Now extend these two labels from i to node j.

Step 1, extend l to l'. Using (9.55), (9.56):

$$cap_{l'} = cap_l + q_j$$

$$C_{l'} = C_l + dist_{ij}$$

Step 2, extend l_p to l'_p:

$$cap_{l'_p} = cap_{l_p} + q_j$$

$$C_{l'_p} = C_{l_p} + dist_{ij}$$

During extension, we have to check the capacity of new extended label won't exceed the maximum capacity of vehicle CAP. If the capacity constraint is not satisfied, we drop the infeasible label. So is the time window constraint.

The relationship of the new extended label l' and l'_p will be:

$$cap_{l'_p} - cap_{l'} = cap_{l_p} - cap_p \geq 0 \tag{9.61}$$

$$C_{l'_p} - C_{l'} = C_{l_p} - C_l \geq 0 \tag{9.62}$$

(9.61) and (9.62) shows that l'_p is dominated by l'. The more cost of l'_p against l' is the same as that between l_p against l. As l and l_p are extended to the same node, the exclusive order of l'_p against l' is the same as that l_p against l. $l_p^{'}$ and l' still hold the saving condition in Rule2, thus l'_p is a prominent label of l'. Got proved!

This property shows that a prominent relationship still holds during a master label extension. No additional prominence test is needed. This is why we store prominent labels sticking to the master label.

The master label extension rule is concluded as following:

1. Extend a master label itself using (9.55) to (9.57);
2. Extend all the prominent labels using (9.55) to (9.57);
3. Store the extended labels in step 2 in the prominent list of the label extended in step 1.
4. Drop labels violating the maximum capacity constraint and time window constraint.

9.7 Master Label Dominance

In this section, we will study dominance relationship between master labels, which contains dominance relationship between the master labels themselves; and dominance relationship between a master label and the prominent labels of another master label.

For simplicity, we assume two master labels p_1 and p_2 of a same node. Each of them has one prominent label, $p_1^{'}$ and $p_2^{'}$:

$$cap_1' \geq cap_1 \tag{9.63}$$

$$C_1' \geq C_1 \tag{9.64}$$

$$cap_2' \geq cap_2 \tag{9.65}$$

$$C_2' \geq C_2 \tag{9.66}$$

$cap_{p_1'}^{ex}$ is the exclusive order capacity of p_1' against p_1. $cap_{p_2'}^{ex}$ is the exclusive order capacity of p_2' against p_2.

We're interested in the relatioship between $p_1^{'}$ and p_2(or $p_2^{'}$ and p_1) under different situations.

Situation i) If p_1, p_2 are both efficient labels, and

$$cap_1 > cap_2 \tag{9.67}$$

$$C_1 < C_2 \tag{9.68}$$

Using (9.63) and (9.67), the capacity of $p_1^{'}$ is larger than p_2.

$$cap_1' \geq cap_1 > cap_2 \tag{9.69}$$

However, the relationship of C_1' and C_2 are not certain by (9.64) and (9.68). Thus the relationship of p_1' and p_2 is not certain. If $C_2 \geq C_1'$, p_2 can't dominate $p_1^{'}$; if $C_2 < C_1'$, p_2 dominate $p_1^{'}$.

Using the same method, we can also prove that the relationship of p_1 and $p_2^{'}$ is not certain, either.

Conclusion: given two efficient master labels, the relationship between the prominent labels of each master label is not certain.

Situation ii) If p_1 dominates p_2, but p_2 is not a prominent label of p_1:

$$cap_1 < cap_2 \tag{9.70}$$

$$C_1 < C_2 \tag{9.71}$$

$$2loading * cap^{ex}_{p'_2} \leq C_2 - C_1 \tag{9.72}$$

Using (9.65), (9.70),

$$cap'_2 \geq cap_2 > cap_1 \tag{9.73}$$

Using (9.66) , (9.71),

$$C'_2 \geq C_2 > C_1 \tag{9.74}$$

(9.73), (9.74) shown, p_1 also dominate $p_2^{'}$. However, the exclusive order capacity of $p_2^{'}$ against p_1 is not the same as $cap^{ex}_{p'_2}$. The more cost $C'_2 - C_1$ is lager than $C_2 - C_1$. An additional prominence test is necessary to check whether $p_2^{'}$ is a prominent label of p_1.

Situation iii) If label p_2 is a prominent label of p_1. From prominent label definition, p_2 is certainly dominated by p_1. Thus (9.73), (9.74) still holds; label p_1 dominates label $p_2^{'}$. However, the exclusive order of $p_2^{'}$ against p_1 is not the same as $cap^{ex}_{p'_2}$. The more cost $C'_2 - C_1$ is larger than $C_2 - C_1$. An additional prominence test is necessary to check whether $p_2^{'}$ is a prominent label of p_1.

From i) to iii), the dominance relationship between master labels can be conclude as

Theorem 8 *If label A dominates master label B, then label A also dominates B's prominent label(s).*

To check whether the prominent label of B is prominent label of A, an additional prominence test is necessary. Based on these dominance and prominence realationship property, in next section, we construct the rule of how to insert a new label into the bucket.

9.8 Insertion of Master Label

In this section, we deal with the five situations of inserting a new extended master label into a calculated bucket.

In NDCA, everytime insert a new label into a bucket, a dominace test is applied. If the new label is dominated by a label already stored in bucket, the new label will be discarded. If the new label is dominating a label already stored in buket, the old label will be removed from the bucekt.

In CDSPA, a dominance test is still applied before inserting a master label is still needed. Moreover, once a dominance occurs, an additional prominence test is applied, which is in order to check whether the dominanted label is a prominante label. There are five situations of relationship between a new extended label with a label stored in the bucket.

Assume the new extended master label is p_1, we are going to insert p_1 into bucket k. p_2 is a master label already stored in bucket k. For simplicity, both master labels have one prominent label, p_1' and p_2', respectively. The five situations will be:

Situation A: p_1 is not dominated by labels stored in bucket; and none label in bucket is dominated by the new label p_1.

Situation B: p_1 dominate p_2, and p_2 is not prominent label of p_1.

Situation C: p_1 dominate p_2, and p_2 is prominent label of p_1.

Situation D: p_2 dominate p_1, and p_1 is not prominent label of p_2.

Situation E: p_2 dominate p_1, and p_1 is prominent label of p_2.

Above situations are the same as we discussed in previous section. Following is the corresponding operations under each situation.

Situation A)
Insert p_1 into bucket k by increasing lexicographic order of used capacity.

Situation B)
Append $p_2^{'}$ into p_1's prominent list if it is a prominent label if p_1. Insert p_1 into bucket k by increasing lexicographic order of used capacity. Remove p_2 from bucket k.

Situation C)
Append p_2 into p_1's prominent list. Append $p_2^{'}$ into p_1's prominent list if it is a prominent label if p_1. Insert p_1 into bucket k by increasing lexicographic order of used capacity. Remove p_2 from bucket k.

Situation D)
This situation is the same type as situation B). P_1 won't be inserted into bucket k. Append $p_1^{'}$ into p_2's prominent list if it is a prominent label if p_2.

Situation E)
This situation is the same type as situation C). Append p_1 into p_2's prominent list. Append $p_1^{'}$ into p_2's prominent list if it is a prominent label if p_2.

We've finished discuss the method of master label extension and how to insert a master label into a bucket. Now we conclude the advantage of storing prominent labels in a master label's prominent list:

1. From the definition, a prominent label is possible to have less combination cost than its master label.
2. During master label extension, the prominence relationship property still holds, none additional prominence test is necessary.
3. The most advantage of storing prominent label in the prominent list, instead of storing them in the master list, is that the NDCA framework won't be destroyed. Efficient labels are stored and treated seperately from prominent labels. The result of the extended efficient labels is the same as in the NDCA and the computation effort of efficent labels keeps unchanged.
4. Some modification and adaptions are employed to keep the prominent labels.
5. Using Theorem 8, it's efficient to decide whether and where to store a prominent label of a dominated master label.

9.9 Waiting Label and Waiting List

In the NDCA, dominance occurs at four operation situations:

1. Insert an extended label into a bucket. Dominance occurs between the extended label and a stored label in bucket.
2. Minimal cost test before the treatment of a label. Dominance occurs between the treated label and a mininal cost label.
3. Minimal cost test before inserting an extended label. Dominance occurs between the new extended label and a mininal cost label.
4. Backward-look test before inserting an extended label. Dominance occurs between the new extended label and a set of bucket mininal cost labels.

The situations 2 to 4 are minimal cost tests. In NDCA, once a label cost is larger or equal than a minimal cost, it is dominated by the minimal cost label. It will be discarded.

In the CDSPA, we also invite these minimal cost tests. A difference is once a master label is dominated by a minimal cost label, an additional prominence test is necessary. Furthermore, if the master label has prominent labels, by Theorem5, its prominent label will also be dominated by the minimal cost label. A prominence test is necessary for each of the prominent label between the minimal cost label.

The prominent label of the minimal cost label, either a master label itself or its prominent label, will be stored in a seperate list named *waiting list.* Otherwise, it will be discarded. We will discuss the *waiting list* later.

In order to apply a prominence test with a minimal cost label, in addition to recored the minimal cost $mincost_i$, a complete information of the minimal cost label is necessary for the the exclusive order capacity calculation.

A node's minimal cost label is stored in one of the previous treated bucket. In each label list of a bucket, master labels are stored in increasing lexicographically order of cost; and decreasing lexicographically order of capacity. Obviously, the first label is the minimal cost label. A minimal cost label can be retrieved using its bucket index. I.e., $k_{i_{min}}$ is the bucket index storing the minimal cost label of node i. Following shows how to retrieve the minimal cost label of node i:

1. find bucket $k_{i_{min}}$;
2. find the label list of node i in bucket $k_{i_{min}}$;
3. the first label of the label list is the minimal cost label of node i.

The backward-look test is much alike the minimal cost test. In a backward-look check, a treated label will have a set of minimal cost test with bucket minimal cost labels in backward buckets from k_{start} to $k - 1$.

If a label is a prominent label of a minimal cost label, we will store it. Suppose we store it in the prominent list of the minimal cost label, it never gets chance to be extended. As the NDCA never retreates a label having been treated before.

In order to get further extension, we move the prominent label(dominated by either a minimal cost label or a bucket minimal cost label) into an independent list,named waiting list, stored in the current label list. Figure 9.6 shows the new data structure.

A bucket consists of a set of label lists of different nodes. Each label list consists of two sublists: 1) a master label list(master list); and 2) a waiting label list(waiting list). Label stored in a waiting list is a *waiting label.* A waiting label is a prominent label of a minimal cost label.

Figure 9.7 show the storage status change of storing a waiting label.

Now the label extension operation contains:

1. extend the master labels in the master list;
2. extend the prominent labels of each master label, if exist;

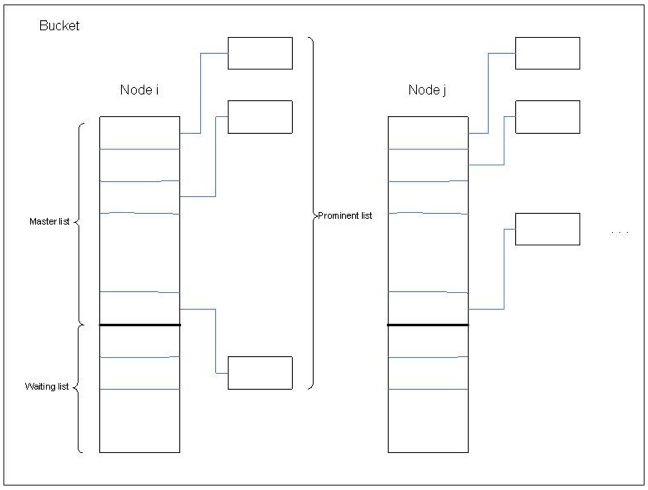


Figure 9.6: Bucket data structure

3. extend waiting labels in the waiting list.

Waiting label extension calculation is the same as master label extension. A new extended waiting label is to be stored in the waiting list in the label list in a calculated bucket.

The waiting labels in a same waiting list have none sorting order; there will be no dominance test between two waiting labels. In order to control the increasing number of waiting labels, as master label, before inserting a new extended waiting label into a calculated bucket, the minimal cost test and the backward-looking check are introduced. A new extended waiting label has dominance test with the minimal cost label and a set of bucket(s) minimal cost label in previous buckets. A waiting label will be discarded if it is dominated by any of these minimal cost label.

In addition to minimal cost test, another Master-Waiting dominance test is introduced. The test is applied between a waiting label and a the master label.

For each waiting label, introduce a dominance test with each master label the same label list. Once a waiting label is dominated by a master label, it is discarded from the waiting list. This dominance test can efficiently reduce a larger set of waiting labels.

The improved algorithm is shown in figure A.4. step 16 to 18: introduce the minimal cost test before treatment of a label; steps 24 to 25: introduce the minimal cost test before inserting a new label into a bucket; steps 27: introduce backward-look test before inserting a new label into a bucket.

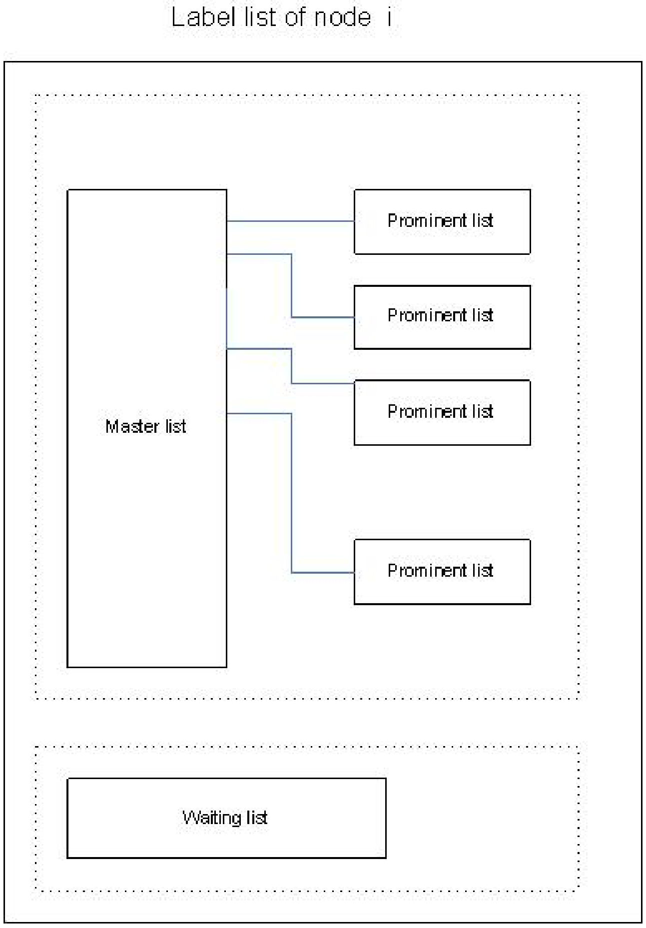


Figure 9.7: Store a waiting label

In CDSPA, another improvement is made to speed up the algorithm operation by sorting customers by increasing order capacity. This increasing capacity order is helpful in label extension treatment. A label is extended to each successor by this order. Once an extension is infeasible due to violation maximum capacity constraint, we discard rest of successors. As each of the rest successor has order capacity no less than the current one, none extension will be feasible, either.

9.10 Store Destination Label

In the NDCA, each destination label(depot label) having negative cost will be stored. In the CDSPA, a combination path having negative cost is probably composed by subpaths either or both having nonnegative cost. Thus some destination labels with nonnegative cost are required to be stored in the CDSPA. A cost check test is introduced before storing a destination label in order to avoid the unwise storage.

Theorem 9 *If a destination label has cost larger or equal than its double loading time,*

$$C \geq 2loading * cap$$

, by contacting with any other label, the combination cost won't be kess than the cost of the other label.

Prove:

Assume a label p with cost C_p, and $C_p \geq 2loading * cap_p$. Combine p with any other label q, the combination cost is

$$C_{\{p,q\}} = C_p + C_q - saving$$

As $saving \leq 2loading * cap_p$,

$$C_{\{p,q\}} = C_p + C_q - saving \geq C_p + C_q - 2loading * cap_p$$

As $C_p \geq 2loading * cap_p$,

$$C_p - 2loading * cap_p \geq 0$$

$$C_p - 2loading * cap_p + C_q \geq C_q$$

$$C_{\{p,q\}} \geq C_p - 2loading * cap_p + C_q \geq C_q$$

Got proved!

Theorem 9 indicate that when to construct a combination path, a path with cost $C \geq 2loading * cap$ can't help in decreasing the combination path cost. Therefore we won't store such destination path.

Rule 5 (Destination label cost check) *If a destination label has a cost larger or equal than its double loading time,*

$$C \geq 2loading * cap$$

it will be discarded; otherwise, it will be stored.

Another difference from the NDCA is extended destination labels are stored in seperate list according to their types. A master destination label is stored in a master label list. A waiting destination label is stored in a waiting label list. When finishing the calculation in CDSPA, we have two destination label lists.

9.11 Final Dominance Test

In this section, we will discuss the final dominance test at node $N + 1$. The minimal cost combination path is found by an enumerative combination of destination labels from each candidate set. Sometimes the number of destination labels will be very huge. The final dominance test helps to reduce the number of destination label sharply. It is to deal with labels in the two label list at destination: master list and waiting list. When finishing the final dominance test, a candidate solution label list is returned.

As destination labels have been pushed back into depot and no more order needs to be onloaded, the empty capacity of a vehicke won't be considered in a dominance test any more. A label becomes an efficient label if it has smaller cost. A label becomes a prominent label once it is possible to save more time by combination.

Definition 9 *A, B are both destination labels, if A cost is less than B, A dominates B.*

Definition 10 *If a final label won't be dominated by any other labels, it becomes a final efficient label.*

Definition 11 *If label A final dominate label B, and*

$$C_B - C_A < 2loading * cap_B^{ex}$$

cap_B^{ex} is the exclusive order capacity of B against A

B is prominent label of A.

The final dominance test is only to keep final efficient labels and final prominent labels.

A final efficient label is the minimal cost label of destination, which is stored in the master list. It's probable to have more than one final efficient label. All of them will be stored into the candidate solution list.

A prominent label of a finial efficient label certainly becomes a final prominent label. For the rest labels in either master list or waiting list, a prominence test is necessary. The prominence test is described in section 9.5. A promininent label will be inserted into candidate solution list.

After a final dominance test in each subpart, we get two solution label lists: pickup solution list and delivery solution list. The combination operation is to form combination paths by enumerative combination of subpaths from each solution list. See section 9.1 for details of combination operation and relative calculations. We only collect the combination paths with negative cost.

9.12 Conclusion of CDSPA

Using CDSPA, we're able to collect either negative cost single path or negative cost combination path. The solution of the CDSPA can completely covers that of the NDCA. On the other handm the computation effort increases.

In the CDSPA, we exclude the consideration of time consuming in dominance and prominence criteria between two labels. This relaxation indicates that the shortest path found by the CDSPA is possible not an optimal solution path of the original subproblem with time window constraint. Thus the CDSPA can't prove the optimality of RMP when no more negative cost path can be found.

However, CDSPA is an efficient algorithm of subproblem in finding negative reduced cost columns to be added into RMP.

In order the prove the optimality of RMP, another optimal algorithm of the subproblem is drawn, which is built based on the CDSPA by adding one prominence condition.

Chapter 10

The Optimal Cross-Docking Shortest Path Algorithm

The optimal algorithm of subproblem(OCDSPA) is easy to develop based on the CDSPA. We only have to include the time consuming factor into the dominance test criteria.

A dominated label has advantage against the dominating label if it has less time consuming. Like the less onloading capacity usage, less time consuming increases a label's extension possibility, which helps in reducing the path cost by selecting more attractive paths in extension. Thus we should have to keep those dominated labels with less time consuming compared to the dominating label.

In order to keep the framework of treatment order of master labels, we let the original dominance test unchanged and make modification in the prominence test criteria. Another new rule is added:

Rule 6 (Keeping rule) *Label A is dominated by label B. A is a prominent label of B if A has less time consuming of B.*

Rule7 will be applied between a dominated pair before any other rules listed at the end of section 9.5. Once a dominated label has less time consuming, it becomes a prominent label immediately.

By adding Rule7 into the CDSPA, the OCDSPA is able to find the optimal solution of the subproblem, either the shortest single path or the shortest combination path. When the OCDSPA can't find any negative cost path, it is sufficient to prove the RMP has reached the optimality.

Chapter 11

Implementation of Subproblem

All algorithm is implemented using C++ language calling ILog CPLEX callerable library. The implement of NDCA and CDSPA follows the direction of 1)not distrubing the algorithm structure and 2)operation efficience. Several classes are designed as container of the treatment according to the algorithms.

The class nodeLabel used to present a label information. It contains node index, path cost, and path details. The visiting nodes sequence is stored as a list. We again create a covered nodes set stored as a set. This makes constant access time during calculation exclusive order.

A master label and a prominent label present using MasterLabel class and PromLabel class respectively. The MasterLabel class is derived from nodeLabel class, as it has a prominent list storing the prominent labels. The PromLabel class is also derived from nodeLabel.

The class myBucket present a bucket information. As the number of buckets in one problem is fixed, a list of buckets is created initially at the begining. For constant access time of one specific bucket during a minimal cost label test, this buckets set is stored in a vector.

The class labelList present the list of a node labels stored in a bucket. In NDCA, a labelList store the Master label list of a node. In CDSPA, a labelList store the Master label list and the waiting label list of a node. Both label lists are stored as list type.

Each bucket may contain many label lists of different nodes. The treatment of label lists in a bucket has no special requirement. However, during a minimal cost label test or a backward-looking check, we have to retrieve a specific node's label list. For constant access time, we build a map of node's index with the label list address.

When finishing computing, each algorithm return one label list as candidate set. We use combLabel class to present a candidate single path or a combination path constructed by enumerative combination labels from these two candidate sets. After sorting, we select the set of best cost(minimal negative cost) paths to add into RMP.

Order to improve the operation efficiency, we sorted all customers by increasing order. In this way, during treatment of a label extension, once an extension is infeasible due to excceeding the capacity, we immediately stop its extension and switch to a next label treatment. As all sequent successor order is no less than the current successor, the extension will be infeasible.

Chapter 12

Implementation of Column Generation

12.1 Start Point

The well known simplex first phase carries over to column generation. Artificial variables, one for each constraint, penalized by a *bigM* cost, are kept in the RMP to ensure feasibility in a branch-and-bound algorithm. A smaller M gives a tighter upper bound on the respective dual variables, and may reduce the headinging effect (Vanderbeck, 2004) of initially producing irrelevant columns.

In some applications, the unit basis is already feasible. Then, an estimate of the actual cost coefficients should be used instead of M. Heuristic estimates of the optimal dual variable values are imposed as artificial upper bounds by Agarwal, Mathur, and Salkin (1989) by introducing unit columns with appropriate cost. The bounds are gradually relaxed until they are no longer binding.

We use the unit basis as a start point in the CG implementation. For each customer's node, we make a route start from depot to visit the node; after service the node, the route directly goes back to the depot. Each customer will have two routes, one pickup subpath and one delivery subpath. The initial start point will have 2N routes. The cost of each route is calculated by the actual distance, $2dist_{si}$, s refers to the depot.

This unit basis start point make sure that we can always find an integer optimal solution of the original problem. However, this start point is very far away from the integer optimal solution. It's the worst feasible integer solution. This bad start point indicate the most time of solving the cross-docking problem will be spent in searching a huge branch and bound tree.

12.2 Add Column into RMP

We are free to choose a subset of non-basic variables, and a criterion according to which a column is selected from the chosen set. According to the classical Dantzig rule, one chooses among all columns the one with the most negative reduced cost. Various schemes are proposed in the literature like full, partial, or multiple pricing (Chvatal, 1983). Column generation is a pricing scheme for large scale linear programs.

The role of the pricing subproblem is to provide a column that prices out profitably or to prove that none exists. It is important to see that any column with negative reduced cost contributes to this aim. In particular, there is no need to solve subproblem exactly; an approximation sufficiency until the last iteration. The ideal SP should have a structure that can be solved effectively since it must be solved repetitively. One method is to use a heuristic to generate good solutions quickly. If the heuristic fails to identify an improving column, an optimizing algorithm that requires more run time but is guaranteed of identifying an improving column (if one exists) must be employed.

We may add many negative reduced cost columns from a subproblem, even positive ones are sometimes used. We may solve a temporary restriction of the subproblem, or a relaxation, which is the case for the vehicle routing problem with time windows (Desrochers, Desrosiers, and Solomon, 1992). The RMP may require longer solution times due to the enlarged problem size. Conversely, columns which are of no use for the linear relaxation may be required for the integer feasibility of the RMP.

In our CG solution, we will add columns into RMP built by single paths or combination paths with negative reduced cost. NDCA is for finding negative reduced cost single path. If NDCA find negative cost single paths in both subparts, obviously we can construct negative cost combination path from these single paths.

Our column insertion stratage is first applying NDCA to find negative cost single paths in each subparts. These paths are stored in pickup path list and delivery path list, seperately. Second, we construct combination paths by enumerate combination single paths from these two path list. As each single path in the path list has negative cost, the built combination path also has negative cost. Third, we collect the built combination paths and all found single paths, and select 100 paths with least cost to insert into RMP.

The process continue till no good quality negative cost path, either single path or combination path, can be found or constructed by NDCA. To identify the good quality path generated by NDCA, we use two criteria: 1) if the negative cost is not smaller enough; 2) if the actual number of columns to be added into RMP is not big enough; As NDCA dosen't consider the candidate subpath of construction a connective path, the built combination path usually has a smaller saving cost. In the later stage of using NDCA to generate combination path, the paths negative cost are not small enough. These columns make very slow converge to the optimal solution. This is also the result in situation 2), not enough columns added into RMP makes slow converge. Thus, either these two situation occurs, we will switch to CDSPA.

By alppying CDSPA, the candidate single paths to construct negative cost connective path are found. Also the dominated negative cost single path could be found and stored in prominent list or waiting list. The insertion stratage of columns into RMP by using CDSPA is the same as NDCA.

The complexity of CDSPA is higher than NDCA, as much more candidate labels are generated in CDSPA. We use NDCA as a heuristic method to approach the optimal solution first, without solving the subproblem exactly. Thus we achieve saving time consuming of applying CDSPA at the very begining. This method efficiently brings us close to the optimal solution

We will swithch from CDSPA to OCDSPA when no good negative column generated by CDSPA, see above. As the CDSPA hasn't taken time consuming of path into dominance test consideration, the feasible solution set of original problem is not completely included into CDSPA. By applying OCDSPA, the complete feasible solution set is searched and it is probably to find better negative columns insert into RMP.

12.3 Stop Criteria

One would expect that the tailing off effect be amplied by the multitude of linear programs to solve. However, the contrary is true. Early termination makes the algorithm effective for integer programs in contrast to linear programs. The need for integer solutions provides us with a very simple amendment: Stop generating columns when tailing off occurs and take a branching decision.

we terminate solving the subproblem earlier to save the computational efforts. When applying OCDSPA to generate negative cost path, if the path cost is not small enough, we stop the subproblem and switch to branch and bound. Here is a tradeoff between computational efforts and the quality of the obtained lower bound upon premature termination. As the start point is not good quality, which will bring astray columns into RMP. The result is there is probable more columns with fraction coefficient in RMP, which make the search tree very huge. Thus it is not necessay to get an exclusive lower bound in the situation.

Note that monitoring the relative decrease of the objective function value over a predefined number of iterations (Gilmore and Gomory, 1963) is not robust against temporary stalls.

12.4 Branch and Bound

After get a lower bound of the RMP, we begin the branch and bound process to search the integer optimal solution. In out CG implementation, we use the depth-first branch and bound stratage.

We choose to branch on sum of x-variables, $x_{ij} = \sum_{k \in K} x_{ij}^k$. This kind if decisions are characterized by fixing arcs (i, j) of the network for all vehicles k. As in our model, all subproblems stay identical.

Our approach to selecting a variable on which to branch is to determine the total flow associated with each x_{ij} variable and branch on the fractional part f_{ij} closest to 0.5, where x_{ij} is the fractional part of $\sum_{k \in K_{ij}^f} \lambda_i^k$, K_{ij}^f is the set of columns that incorporate x_{ij}, $K_{ij}^f \in K_i$.

Branch on $\sum_{k \in K} x_{ij}^k$ is equivalent to branch on single flow variables f_{ij}. In the VRPTW context this was proposed by Desrochers, Desrosiers and Solomon(1922). All subproblems remains identical. Fixing f_{ij} to 0 simply done by removing the arc (i, j) from the network. In the master problem, all columns covering arc (i, j) will be removed; in the subproblem, arc (i, j) is removed from network.

Fixing f_{ij} to 1 is more complex than branch to 0. In the master problem, remove all columns start from i except terminating at j. If i is depot, keep those columns. And remove all columns terminate at j except starting at i. If j is depot, keep those columns. In the subproblem, remove all arcs start from i except arc (i, j). If i is depot, keep those arcs. And remove all arcs terminate at j excpet start from i. If j is depot, keep those arcs.

It will be the situation that a RMP has infeasible solution during the branching process. The reason is after muti-times of branching, some arcs are removed which is necessary in order to build a feasible solution. To avoid this problem, we keep the start point solution untouched. The first 2N columns are always keeping stayed in the RMP. As these columns form an integer solution of cross-docking problem, we can always find a feasible solution within these columns included in.

During the branch and bound process, we can use the updated upper bound to prone the infeasible branches. Each feasible integer solution we find during the branch and bound process is an upper bound of the original problem solution. These integer solutions are not generated by decresing objective value order. However, we can still use them to update the global upper bound, which will be the smallest integer solution objective value. During branching, once a lower bound of a branching node is larger than the global upper bound, this branch node is proned. As each integer solution found under this branching node is larger than the node's lower bound, which is obviously larger than the global bound.

Chapter 13

Numeric Result

13.1 Subproblem Algorithm

Subproblem has to be repeatedly solved in finding negative reduced cost columns. The efficiency of a column generation implementation is mainly decided by the efficiency of the algorithms of solving subproblem. We include the testing of the three algorithims developed to solve the subproblem.

1. NDCA is the algorithm to find the optimal path in a capacity resource constrained shortest path problem.
2. CDSPA is the algorithm to find the optimal combination path in a capacity resource constrained shortest path problem.
3. OCDSPA is the algorithm to find the optiaml path in a capacity resourse constrained shortest path problem with time window constraint(RCSPPTW).

Without any time window constraint

The NDCA is very fast algorithm to find good negative reduced cost single path in the subproblem. However the CDSPA is only efficient to find good negative reduced cost path within very few nodes situation, usually under 10 customers. Testing the CDSPA with 10 customers, it becomes extremely time consuming. The reason is the number of candidate path of destination node becomes very huge. The algorithm try to push every label which is possible to become an optimal subpath in construction of the shortest combination path. As during the label pushing process in one subpart, we have no information of how the label pushing situation is in the other part, in which no condition can be used to prone candidate labels being pushed further.

Apply a tight time window constraint

There are several ways to make the testing more practical and more realistic.

We choose to add a tight service time window constraint for each customer's node. For each customer, the pikcup node and the delivery node have a service time window $[a_i, b_i]$, respectively. A vehicle can't begin docking at a customer's node before the start serving time a_i; and should finish uploading or offloading a customer's order before the end serving time b_i. This tight time windows constraint sharply cut down the feasible set of candidate labels.

The time windows are generated as following. As a delivery node is usually two to three times far away from the depot than a pickup node, and the depot opens from 6am to 22pm, we assume all pickup nodes' start seving time between 7am to 14pm and all delivery nodes' start serving time between 10am to 21pm. For each customer, the start service time between delivery node and pickup node is far enough to ensure that a vehicle has enough time to transport the order back to depot first, then delivery it to the delivery node. For each delivery nodes' end service time, we make sure that a vehicle can return to depot and offload all the orders before 22pm. The width of time window is around one to two hours depending on the actual order capacity of each customer.

The CDSPA efficiency has been sharply improved by adding time windows constraints.

A test with N customers indicates there are total 2N service nodes, N pickup nodes and N delivery nodes. Testing case is generated by selecting the nodes from the provided data. First introduce the content in each column:

1. column 1: name of the testing case;
2. column 2: objective value of NDCA;
3. column 3: objective value of CDSPA;
4. column 4: objective value of OCDSPA;
5. column 5: gap between NDCA and OCDSPA;
6. column 6: gap between CDSPA and OCDSPA;
7. column 7: cpu time using of NDCA;
8. column 8: cpu time using of CDSPA;
9. column 9: cpu time using of OCDSPA;
10. column 10: cpu time ratio between NDCA and OCDSPA;
11. column 11: cpu time ratio between CDSPA and OCDSPA;

The number in case name indicates customer number. The charactor p or d at the end of case name indicate pickup subpart or delivery subpart. For instance , $ds50.a4.p$ refer to a data file containing 50 customers's pickup nodes information.

Table 13.1 show the three algorithms testing result of customer number from 10 to 200. Under each testing case, the NDCA used least cpu time to find the optimal solution and the OCDSPA used the most cpu time. The optimal objective value found by the NDCA is no better than the other two algorithms, and the CDSPA is no better than the OCDSPA. As the cumstomer number increases, computation effort grows within all the three algorithms.

The difference of objective solution value between the three algorithms show variety in customers number. When customer number is under 20(include 20), for the NDCA, there is 1 case less than optimal gap 0.5%, 1 case less than optimal gap 5%, 2 cases less than optimal gap 15%, and 3 cases larger than optimal gap 15%; for the CDSPA, there is only one case less than optimal gap 0.5%.

When customer number is 50, for the NDCA, there're 4 cases less than optimal gap 0.5%, 8 cases less than optimal gap 5%, 2 cases less than optimal gap 15%, and 1 cases larger than optimal gap 15%; for the CDSPA, there is 1 case less than optimal gap 0.5%, 6 cases less than 5%, and 2 cases less than 15%.

When customer number is from 100 to 200, for the NDCA, there're 6 cases less than optimal gap 0.5%, 5 cases less than optimal gap 5%, 4 cases less than optimal gap 15%, and 1 cases larger than optimal gap 15%; for the CDSPA, there're 5 cases less than optimal gap 0.5%, 5 cases less than 5%, and 1 case less than 15%.

Conclude, in 10 to 20 customers, the NDCA has 12.5% case larger than the optimal gap 5%; the CDSPA all reach optimal solution. in 50 customers, the NDCA has 9.4% case larger than the optimal gap 5%; the CDSPA has 6.2%. in 100 to 200 customers, the NDCA has 14% case larger than the optimal gap 5%; the CDSPA 3%. The result show that the NDCA is capable of finding close solution to the optimality, the CDSPA is capable of finding very close solution to the optimality. And as the customer number increase, the gap of NDCA shows increasing tendence, however the CDSPA shows the opposite result.

In addition to the objective value, the computation effort also show difference variety between the three algorithms. In order to find the optimal solution, the OCDSPA always consumed much more cpu time. In the 10 to 20 customers, the average cpu time of OCDSPA is 3.3 to 5 times of the NDCA; 1.7 to 2 times of the CDSPA. In the 50 customers, the average cpu time of OCDSPA is 7 to 10 times of the NDCA; 2.5 times of the CDSPA. In the 10 to 20 customers, the average cpu time of OCDSPA is 17 to 50 times of the NDCA; 3 to 10 times of the CDSPA.

The difference of computation effort shows that the NDCA achieves to saving computation effort by the possibility of increasing optimal gap, and the CDSPA and OCDSPA cost more computation effort to close the optimal gap.

As the customer number increased, the difference of computation effort between the NDCA and the OCDSPA also increased drasticly. We can expect that the incresing cpu time is due to the increasing treated candidate labels, thus the above result show that:

1. the candidate paths treated in the CDSPA or the OCDSPA are much more than the NDCA;
2. the candidate paths treated in the CDSPA or the OCDSPA grow drasticly fast than the NDCA as the customer number increases;
3. the candidate paths treated in the CDSPA or the OCDSPA are more sensitive to the nodes distribution situation as the customer number increases. In the testing case from 100 to 200 customers, the computation effort comparation betwee the NDCA and the OCDSPA shows a broad variety from 17 to 50. In another word the efficiency of OCDSPA and CDSPA is not only dependent on nodes number but also dependent on the nodes location.

Conclusion of the subproblem algorithms testing: the NDCA is fast to find a close optimal solution of negative single path, the CDSPA is efficient to find a very close optimal solution of subproblem with less time cost compared to the optimal algorithm. The result clearly shows the complexity in solving the Cross-Docking problem into optimality. An successful algorithm of the Cross-Docking problem results in an efficient subproblem algorithm capable to handle the drastic increasing candidate labels.

Compare with a loose time window constraint

We also study the result of NDCA, CDSPA and OCDSPA under a loose time windows constraints. Table how the solution result under a loose time windows of 3-hours service time for each customer node, $b_i - a_i \leq 3h$. As above, the comparation was made under different customers number.

Under each testing case, the optimal objective value found by the NDCA is no better than the other two algorithms, and the CDSPA is no better than the OCDSPA. Within 10 to 20 customers, the NDCA and the CDSPA are both none of optimal gap larger than 5%. With 50 customers, the NDCA has 7.5% optimal gap larger than 5%; the CDSPA has 5%. With larger than 50 customers, the NDCA increase to 22%; the CDSPA increase to 11%. Under each testing case, the NDCA used least cpu time to find the optimal solution and the OCDSPA used the most cpu time. As the cumstomer number increases, computation effort grows within all the three algorithms.

In addition to the comparation of testing result of the three algorithms under a loose time window constraint, we also make comparation with the result of 1-hour service time window got above. As the time window constraint for each node was regenerated, the comparation only include the computation effort under tight/loose time window situations; the objective value is not included in the comparation.

Compared with 1-hour service time, the result of 3-hours show that with less customers' situation, a loose time window constraint is prone to close the optimal gap of the NDCA and the CDSPA; however when the customers number increases, a loose time window constraint is prone to increase the optimal gap of the NDCA and CDSPA.

The result obviously show the difference of dominance criteria of the three algorithms. In the situation of less customers and loose time windows constraint, we can consider that the time window constraint is almost relaxed. Thus the dominance criteria of the NDCA approximatedly presented for the optimal solution dominance criteria. The possibility of the candidate label of optimal solution being cut off decreases; and the optimal gap closes. However in the situation of density network and loose time windows, the time window constraint can't be relaxed. A wide service time window increase the number of candidate labels, thus the possibility of candidate label of optimal solution being dominated increase also; and the optimal gap increases.

Compared the computation effort with 1-hour service time, 3-hours service time hadn't increased the cpu time drasticly as we expected. On the opposite, the computation effort didn't changed too much. The result is most probably due to the time window constraint defined for pickup subpath(4 hours) and delivery subpath(12 hours).

13.2 Column Generation Testing

The computation effort of column generation implementation without time windows constraints is extremely time consuming, as the bad performance of the algorithm in solving the subproble. Even under the situation of less than 10 customers, the result is rather very unsatisfied, either. Thus we won't include the testing result here.

In addition to the low efficiency of subproblem algorithm discussed in previous section, the reason for the bad performance of extremely time consuming can be explained in several ways:

1. First is the start point. As we have no warm start points, the one-way-out-and-in solution is very far away from the optimal solution. Poorly chosen initial columns lead the algorithm astray, when they do not resemble the structure of a possible optimal solution at all. They must then be interpreted as a misleading bound on an irrelevant linear combination of the dual variables. Thus at the begining of column generation process,

case	NDCA sol	CSPA sol	CDSPA sol	gap1	gap2	cpu time1	cpu time2	cpu time3	cpu rat1	cpu rat2
da10.a1.p	-213	-213	-213	-0	-0	9999.99	20000	30000	0.333333	0.666667
da10.a1.d	-1155	-1155	-1155	-0	-0	20000	30000	40000	0.5	0.75
da10.a2.p	-201	-206	-206	0.0242718	-0	0	9999.99	20000	0	0.5
da10.a2.d	-766	-766	-766	-0	-0	10000	20000	30000	0.333334	0.666667
da10.a3.p	-257	-257	-257	-0	-0	9999.99	20000	20000	0.5	1
da10.a3.d	-951	-951	-951	-0	-0	9999.99	20000	30000	0.333333	0.666666
da10.a4.p	-329	-329	-329	-0	-0	0	9999.99	30000	0	0.333333
da10.a4.d	-891	-891	-891	-0	-0	10000	20000	30000	0.333334	0.666667
da10.a5.p	-192	-192	-192	-0	-0	0	9999.99	20000	0	0.5
da10.a5.d	-638	-638	-638	-0	-0	10000	9999.99	20000	0.500001	0.5
da10.b1.p	-334	-334	-334	-0	-0	0	9999.99	30000	0	0.333333
da10.b1.d	-705	-705	-705	-0	-0	10000	20000	40000	0.25	0.5
da10.b2.p	-198	-212	-212	0.0660377	-0	0	9999.99	20000	0	0.5
da10.b2.d	-706	-706	-706	-0	-0	10000	9999.99	20000	0.500001	0.5
da10.b3.p	-132	-132	-132	-0	-0	0	9999.99	20000	0	0.5
da10.b3.d	-1124	-1124	-1124	-0	-0	10000	20000	30000	0.333334	0.666667
da10.b4.p	-204	-204	-204	-0	-0	10000	20000	30000	0.333334	0.666667
da10.b4.d	-981	-981	-981	-0	-0	9999.99	20000	30000	0.333333	0.666667
da10.b5.p	-147	-147	-147	-0	-0	9999.99	20000	30000	0.333333	0.666667
da10.b5.d	-689	-689	-689	-0	-0	9999.99	20000	30000	0.333333	0.666667

Table 13.1: Shortest path of 10 customers

case	NDCA sol	CSPA sol	CDSPA sol	gap1	gap2	cpu time1	cpu time2	cpu time3	cpu rat1	cpu rat2
da20.a1.p	-230	-310	-310	0.258065	-0	10000	40000	89999.9	0.111111	0.444444
da20.a1.d	-1039	-1039	-1039	-0	-0	30000	70000	120000	0.25	0.583333
da20.a2.p	-221	-274	-274	0.193431	-0	10000	50000	99999.9	0.1	0.5
da20.a2.d	-866	-866	-866	-0	-0	30000	79999.9	170000	0.176471	0.470588
da20.a3.p	-386	-386	-386	-0	-0	30000	79999.9	160000	0.1875	0.5
da20.a3.d	-1053	-1053	-1053	-0	-0	50000	110000	190000	0.263158	0.578947
da20.a4.p	-384	-427	-427	0.100703	-0	20000	60000	140000	0.142857	0.428571
da20.a4.d	-1117	-1117	-1117	-0	-0	40000	89999.9	190000	0.210526	0.473684
da20.a5.p	-325	-417	-417	0.220624	-0	10000	60000	110000	0.0909092	0.545455
da20.a5.d	-1616	-1616	-1618	0.00123609	0.00123609	30000	79999.9	160000	0.1875	0.5
da20.b1.p	-224	-224	-224	-0	-0	9999.99	40000	69999.9	0.142857	0.571429
da20.b1.d	-1163	-1163	-1163	-0	-0	30000	60000	89999.9	0.333333	0.666667
da20.b2.p	-223	-223	-223	-0	-0	10000	40000	89999.9	0.111111	0.444444
da20.b2.d	-1626	-1626	-1626	-0	-0	30000	70000	170000	0.176471	0.411765
da20.b3.p	-216	-216	-216	-0	-0	20000	50000	89999.9	0.222222	0.555556
da20.b3.d	-1026	-1026	-1026	-0	-0	30000	69999.9	120000	0.25	0.583333
da20.b4.p	-216	-216	-216	-0	-0	20000	50000	89999.9	0.222222	0.555556
da20.b4.d	-1121	-1121	-1121	-0	-0	40000	79999.9	130000	0.307692	0.615385
da20.b5.p	-300	-300	-300	-0	-0	10000	50000	99999.9	0.1	0.5
da20.b5.d	-1131	-1131	-1131	-0	-0	40000	79999.9	140000	0.285714	0.571429

Table 13.2: Shortest path of 20 customers

case	NDCA sol	CSPA sol	CDSPA sol	gap1	gap2	cpu time1	cpu time2	cpu time3	cpu rat1	cpu rat2
da50.a1.p	-422	-433	-488	0.135246	0.112705	70000	330000	770000	0.0909091	0.428571
da50.a1.d	-1574	-1592	-1592	0.0113065	-0	160000	480000	1.1e+06	0.145455	0.436364
da50.a2.p	-330	-425	-425	0.223529	-0	70000	320000	779999	0.0897437	0.410256
da50.a2.d	-1399	-1399	-1399	-0	-0	190000	560000	1.18e+06	0.161017	0.474576
da50.a3.p	-318	-325	-325	0.0215385	-0	50000	270000	570000	0.0877194	0.473684
da50.a3.d	-1665	-1665	-1665	-0	-0	180000	470000	829999	0.216868	0.566265
da50.a4.p	-438	-438	-438	-0	-0	60000	260000	560000	0.107143	0.464286
da50.a4.d	-1368	-1368	-1368	-0	-0	150000	420000	809999	0.185185	0.518519
da50.b1.p	-395	-395	-408	0.0318627	0.0318627	60000	290000	640000	0.0937501	0.453125
da50.b1.d	-1404	-1404	-1422	0.0126582	0.0126582	190000	480000	1.09e+06	0.174312	0.440367
da50.b2.p	-441	-441	-441	-0	-0	60000	290000	639999	0.0937501	0.453125
da50.b2.d	-1864	-1864	-1906	0.0220357	0.0220357	180000	510000	1.22e+06	0.147541	0.418033
da50.b3.p	-316	-319	-319	0.00940439	-0	50000	230000	500000	0.1	0.46
da50.b3.d	-1324	-1324	-1324	-0	-0	140000	370000	800000	0.175	0.4625
da50.b4.p	-382	-383	-383	0.00261097	-0	60000	350000	789999	0.0759494	0.443038
da50.b4.d	-1886	-1886	-1977	0.0460293	0.0460293	230000	620000	1.7e+06	0.135294	0.364706
da50.c1.p	-426	-426	-426	-0	-0	70000	290000	640000	0.109375	0.453125
da50.c1.d	-1492	-1492	-1492	-0	-0	180000	470000	1.38e+06	0.130435	0.34058
da50.c2.p	-443	-443	-443	-0	-0	60000	330000	719999	0.0833334	0.458333
da50.c2.d	-2066	-2066	-2066	-0	-0	210000	550000	1.15e+06	0.182609	0.478261
da50.c3.p	-457	-459	-459	0.0043573	-0	70000	390000	899999	0.0777778	0.433333
da50.c3.d	-1528	-1528	-1528	-0	-0	200000	590000	1.61e+06	0.124224	0.36646
da50.c4.p	-407	-407	-407	-0	-0	70000	320000	719999	0.0972223	0.444445
da50.c4.d	-1429	-1429	-1429	-0	-0	210000	580000	1.93e+06	0.108808	0.300518
da50.e1.p	-336	-336	-362	0.0718232	0.0718232	60000	260000	600000	0.1	0.433333
da50.e1.d	-1512	-1512	-1512	-0	-0	170000	450000	1.02e+06	0.166667	0.441176
da50.e2.p	-442	-442	-445	0.00674157	0.00674157	70000	350000	879999	0.0795455	0.397727
da50.e2.d	-1399	-1399	-1399	-0	-0	170000	560000	1.53e+06	0.111111	0.366013
da50.e3.p	-445	-445	-450	0.0111111	0.0111111	90000	460000	1.37e+06	0.0656935	0.335766
da50.e3.d	-1635	-1635	-1635	-0	-0	250000	800000	2.05e+06	0.121951	0.390244
da50.e4.p	-426	-426	-426	-0	-0	50000	320000	729999	0.0684932	0.438356
da50.e4.d	-1307	-1307	-1324	0.0128399	0.0128399	210000	590000	1.29e+06	0.162791	0.457364

Table 13.3: Shortest path of 50 customers

case	NDCA sol	CSPA sol	CDSPA sol	gap1	gap2	cpu time1	cpu time2	cpu time3	cpu rat1	cpu rat2
da100.a1.p	-474	-500	-510	0.0705882	0.0196078	260000	1.54e+06	4.72e+06	0.0550848	0.326271
da100.a1.d	-2428	-2428	-2430	0.000823045	0.000823045	850000	2.64e+06	2.078e+07	0.0409047	0.127045
da100.a2.p	-446	-446	-446	-0	-0	180000	1.09e+06	2.66e+06	0.0676692	0.409774
da100.a2.d	-1761	-1761	-1766	0.00283126	0.00283126	590000	1.79e+06	8.73e+06	0.067583	0.20504
da100.b1.p	-456	-457	-457	0.00218818	-0	230000	1.49e+06	3.98e+06	0.057789	0.374372
da100.b1.d	-1861	-1861	-1965	0.0529262	0.0529262	920000	2.97e+06	1.162e+07	0.0791738	0.255594
da100.b2.p	-448	-510	-514	0.128405	0.0077821	220000	1.46e+06	4.28e+06	0.0514019	0.341122
da100.b2.d	-2082	-2082	-2082	-0	-0	750000	2.45e+06	1.043e+07	0.071908	0.234899
da100.c1.p	-465	-465	-465	-0	-0	230000	1.32e+06	3.4e+06	0.0676471	0.388235
da100.c1.d	-2077	-2077	-2077	-0	-0	760000	2.23e+06	8.41e+06	0.0903686	0.26516
da100.c2.p	-526	-526	-528	0.00378788	0.00378788	240000	1.61e+06	5.14e+06	0.0466926	0.31323
da100.c2.d	-1827	-1827	-1882	0.0292242	0.0292242	820000	3.03e+06	2.085e+07	0.0393285	0.145324
da100.d1.p	-545	-566	-566	0.0371025	-0	270000	1.92e+06	6.81e+06	0.0396476	0.281938
da100.d1.d	-2011	-2011	-2011	-0	-0	760000	3.37e+06	1.29e+07	0.0589147	0.26124
da100.d2.p	-443	-443	-443	-0	-0	240000	1.57e+06	3.92e+06	0.0612245	0.40051
da100.d2.d	-1890	-1890	-1890	-0	-0	870000	2.79e+06	1.449e+07	0.0600414	0.192547

Table 13.4: Shortest path of 100 customers

case	NDCA sol	CSPA sol	CDSPA sol	gap1	gap2	cpu time1	cpu time2	cpu time3	cpu rat1	cpu rat2
da150.a.p	-567	-567	-567	-0	-0	610000	4.4e+06	1.695e+07	0.0359882	0.259587
da150.a.d	-2303	-2303	-2313	0.00432339	0.00432339	2.06e+06	7.34e+06	7.938e+07	0.0259511	0.0924666
da150.b.p	-568	-572	-646	0.120743	0.114551	570000	4.43e+06	2.196e+07	0.0259563	0.20173
da150.b.d	-2249	-2249	-2249	-0	-0	2e+06	9.18e+06	8.703e+07	0.0229806	0.105481
da150.c1.p	-528	-528	-528	-0	-0	510000	3.31e+06	1.095e+07	0.0465754	0.302283
da150.c1.d	-2072	-2072	-2156	0.038961	0.038961	1.97e+06	7.24e+06	6.833e+07	0.0288307	0.105956
da150.c2.p	-444	-519	-519	0.144509	-0	510000	3.96e+06	1.405e+07	0.0362989	0.281851
da150.c2.d	-1992	-1992	-1992	-0	-0	1.76e+06	7.33e+06	8.98e+07	0.0195991	0.0816258
da150.d1.p	-552	-552	-552	-0	-0	510000	3.89e+06	1.136e+07	0.0448944	0.34243
da150.d1.d	-2201	-2201	-2201	-0	-0	1.66e+06	6.49e+06	4.026e+07	0.041232	0.161202
da150.d2.p	-516	-516	-516	-0	-0	550000	4.07e+06	1.252e+07	0.0439297	0.32508
da150.d2.d	-2172	-2172	-2172	-0	-0	1.65e+06	6.48e+06	3.271e+07	0.0504433	0.198105
da200.a1.p	-460	-576	-576	0.201389	-0	800000	6.47e+06	2.485e+07	0.0321932	0.260362
da200.a1.d	-2524	-2524	-2524	-0	-0	3.14e+06	1.129e+07	1.18308e+09	0.00265408	0.00954286
da200.b.p	-562	-563	-563	0.0017762	-0	860000	7.24e+06	2.379e+07	0.0361497	0.30433
da200.b.d	-2259	-2299	-2376	0.0492424	0.0324074	3.85e+06	1.414e+07	1.15452e+09	0.00333472	0.0122475
da200.c.p	-582	-582	-582	-0	-0	1.09e+06	8.31e+06	3.771e+07	0.0289048	0.220366
da200.c.d	-2329	-2329	-2329	-0	-0	3.3e+06	1.319e+07	2.4716e+08	0.0133517	0.0533662
da200.d.p	-531	-531	-531	-0	-0	940000	8.29e+06	3.055e+07	0.0307692	0.271358
da200.d.d	-2323	-2323	-2323	-0	-0	3.15e+06	1.324e+07	6.2356e+08	0.00505164	0.0212329

Table 13.5: Shortest path of 150-200 customers

case	NDCA sol	CSPA sol	CDSPA sol	gap1	gap2	cpu time1	cpu time2	cpu time3	cpu rat1	cpu rat2
ds10.a1.p	-210	-210	-210	-0	-0	9999.99	20000	30000	0.333333	0.666667
ds10.a1.d	-946	-946	-946	-0	-0	9999.99	20000	30000	0.333333	0.666667
ds10.a2.p	-218	-218	-218	-0	-0	0	9999.99	20000	0	0.5
ds10.a2.d	-799	-799	-799	-0	-0	10000	20000	30000	0.333334	0.666667
ds10.a3.p	-200	-200	-200	-0	-0	10000	9999.99	20000	0.500001	0.5
ds10.a3.d	-633	-633	-633	-0	-0	9999.99	20000	30000	0.333333	0.666667
ds10.a4.p	-223	-228	-228	0.0219298	-0	0	10000	20000	0	0.5
ds10.a4.d	-615	-628	-646	0.0479876	0.0278638	0	9999.99	20000	0	0.5
ds10.a5.p	-155	-155	-155	-0	-0	10000	9999.99	20000	0.500001	0.5
ds10.a5.d	-501	-501	-501	-0	-0	9999.99	20000	30000	0.333333	0.666667
ds10.b1.p	-310	-310	-310	-0	-0	0	9999.99	30000	0	0.333333
ds10.b1.d	-713	-713	-713	-0	-0	10000	20000	40000	0.25	0.5
ds10.b2.p	-233	-233	-233	-0	-0	0	9999.99	20000	0	0.5
ds10.b2.d	-946	-946	-946	-0	-0	10000	20000	30000	0.333334	0.666667
ds10.b3.p	-95	-95	-95	-0	-0	9999.99	20000	20000	0.5	1
ds10.b3.d	-672	-672	-672	-0	-0	9999.99	20000	30000	0.333333	0.666666
ds10.b4.p	-155	-155	-155	-0	-0	10000	20000	30000	0.333334	0.666667
ds10.b4.d	-926	-926	-926	-0	-0	9999.99	20000	30000	0.333333	0.666667
ds10.b5.p	-172	-172	-172	-0	-0	10000	20000	20000	0.500001	1
ds10.b5.d	-1046	-1046	-1046	-0	-0	9999.99	20000	30000	0.333333	0.666666

Table 13.6: Shortest path of 3-hours service window

case	NDCA sol	CSPA sol	CDSPA sol	gap1	gap2	cpu time1	cpu time2	cpu time3	cpu rat1	cpu rat2
ds20.a1.p	-326	-338	-338	0.035503	-0	20000	50000	89999.9	0.222222	0.555556
ds20.a1.d	-882	-882	-882	-0	-0	30000	69999.9	120000	0.25	0.583333
ds20.a2.p	-226	-226	-226	-0	-0	20000	60000	110000	0.181818	0.545455
ds20.a2.d	-885	-885	-885	-0	-0	40000	89999.9	150000	0.266667	0.6
ds20.a3.p	-279	-293	-293	0.0477816	-0	20000	69999.9	150000	0.133333	0.466667
ds20.a3.d	-979	-979	-979	-0	-0	50000	110000	200000	0.25	0.55
ds20.a4.p	-315	-315	-315	-0	-0	20000	60000	99999.9	0.2	0.6
ds20.a4.d	-1314	-1314	-1314	-0	-0	40000	79999.9	130000	0.307692	0.615385
ds20.a5.p	-315	-315	-315	-0	-0	20000	60000	110000	0.181818	0.545455
ds20.a5.d	-1445	-1445	-1448	0.00207182	0.00207182	40000	89999.9	190000	0.210526	0.473684
ds20.b1.p	-216	-216	-216	-0	-0	9999.99	40000	69999.9	0.142857	0.571429
ds20.b1.d	-1072	-1072	-1072	-0	-0	30000	60000	99999.9	0.3	0.6
ds20.b2.p	-340	-340	-340	-0	-0	10000	40000	89999.9	0.111111	0.444445
ds20.b2.d	-1397	-1397	-1397	-0	-0	30000	80000	150000	0.2	0.533333
ds20.b3.p	-211	-211	-211	-0	-0	10000	40000	79999.9	0.125	0.5
ds20.b3.d	-1032	-1032	-1032	-0	-0	30000	70000	110000	0.272727	0.636364
ds20.b4.p	-218	-218	-218	-0	-0	9999.99	40000	69999.9	0.142857	0.571429
ds20.b4.d	-1077	-1077	-1077	-0	-0	30000	70000	99999.9	0.3	0.7
ds20.b5.p	-315	-315	-315	-0	-0	10000	50000	99999.9	0.1	0.5
ds20.b5.d	-1295	-1295	-1295	-0	-0	40000	89999.9	150000	0.266667	0.6

Table 13.7: Shortest path of 3-hours service window

case	NDCA sol	CSPA sol	CDSPA sol	gap1	gap2	cpu time1	cpu time2	cpu time3	cpu rat1	cpu rat2
ds50.a1.p	-444	-444	-444	-0	-0	70000	380000	979999	0.0714286	0.387755
ds50.a1.d	-1784	-1784	-1784	-0	-0	180000	590000	1.63e+06	0.110429	0.361963
ds50.a2.p	-324	-337	-337	0.0385757	-0	70000	320000	719999	0.0972223	0.444445
ds50.a2.d	-1322	-1322	-1334	0.0089955	0.0089955	200000	550000	1.4e+06	0.142857	0.392857
ds50.a3.p	-332	-332	-332	-0	-0	50000	240000	510000	0.0980393	0.470588
ds50.a3.d	-1356	-1474	-1596	0.150376	0.0764411	140000	410000	979999	0.142857	0.418367
ds50.a4.p	-325	-325	-325	-0	-0	50000	230000	490000	0.102041	0.469388
ds50.a4.d	-1426	-1426	-1426	-0	-0	150000	390000	760000	0.197368	0.513158
ds50.b1.p	-444	-444	-444	-0	-0	80000	360000	799999	0.1	0.45
ds50.b1.d	-1611	-1611	-1611	-0	-0	250000	650000	1.55e+06	0.16129	0.419355
ds50.b2.p	-353	-353	-353	-0	-0	60000	250000	590000	0.101695	0.423729
ds50.b2.d	-1501	-1501	-1501	-0	-0	160000	450000	1.09e+06	0.146789	0.412844
ds50.b3.p	-340	-340	-340	-0	-0	50000	230000	510000	0.0980393	0.45098
ds50.b3.d	-1451	-1451	-1451	-0	-0	150000	400000	1.01e+06	0.148515	0.39604
ds50.b4.p	-444	-444	-444	-0	-0	70000	290000	650000	0.107692	0.446154
ds50.b4.d	-1863	-1863	-1863	-0	-0	180000	480000	1.17e+06	0.153846	0.410256
ds50.c1.p	-385	-400	-400	0.0375	-0	70000	320000	699999	0.1	0.457143
ds50.c1.d	-1484	-1501	-1634	0.0917993	0.0813953	190000	520000	1.82e+06	0.104396	0.285714
ds50.c2.p	-434	-434	-440	0.0136364	0.0136364	80000	370000	879999	0.0909091	0.420455
ds50.c2.d	-1677	-1677	-1677	-0	-0	230000	610000	1.76e+06	0.130682	0.346591
ds50.c3.p	-462	-462	-462	-0	-0	70000	260000	560000	0.125	0.464286
ds50.c3.d	-1443	-1443	-1443	-0	-0	160000	420000	889999	0.179775	0.47191
ds50.c4.p	-423	-449	-449	0.0579065	-0	60000	280000	590000	0.101695	0.474576
ds50.c4.d	-1494	-1494	-1494	-0	-0	160000	440000	989999	0.161616	0.444444

Table 13.8: Shortest path of 3-hours service window

case	NDCA sol	CSPA sol	CDSPA sol	gap1	gap2	cpu time1	cpu time2	cpu time3	cpu rat1	cpu rat2
ds50.d1.p	-358	-371	-371	0.0350404	-0	60000	330000	849999	0.0705883	0.388235
ds50.d1.d	-1500	-1500	-1500	-0	-0	220000	700000	2.14e+06	0.102804	0.327103
ds50.d2.p	-336	-427	-427	0.213115	-0	60000	260000	590000	0.101695	0.440678
ds50.d2.d	-1546	-1546	-1546	-0	-0	150000	430000	1.01e+06	0.148515	0.425743
ds50.d3.p	-332	-334	-334	0.00598802	-0	70000	270000	580000	0.12069	0.465517
ds50.d3.d	-1716	-1716	-1716	-0	-0	160000	430000	919999	0.173913	0.467391
ds50.d4.p	-229	-229	-229	-0	-0	50000	200000	440000	0.113636	0.454546
ds50.d4.d	-1165	-1165	-1165	-0	-0	140000	360000	750000	0.186667	0.48
ds50.e1.p	-337	-337	-337	-0	-0	50000	250000	570000	0.0877194	0.438597
ds50.e1.d	-1507	-1507	-1507	-0	-0	170000	470000	1.22e+06	0.139344	0.385246
ds50.e2.p	-438	-438	-438	-0	-0	60000	300000	630000	0.0952381	0.476191
ds50.e2.d	-1415	-1415	-1415	-0	-0	160000	450000	979999	0.163265	0.459184
ds50.e3.p	-416	-416	-416	-0	-0	70000	350000	859999	0.0813954	0.406977
ds50.e3.d	-1390	-1390	-1390	-0	-0	220000	650000	1.75e+06	0.125714	0.371429
ds50.e4.p	-443	-443	-443	-0	-0	50000	280000	600000	0.0833334	0.466667
ds50.e4.d	-1436	-1442	-1467	0.0211316	0.0170416	190000	490000	1.2e+06	0.158333	0.408333

Table 13.9: Shortest path of 3-hours service window

case	NDCA sol	CSPA sol	CDSPA sol	gap1	gap2	cpu time1	cpu time2	cpu time3	cpu rat1	cpu rat2
ds100.a1.p	-553	-553	-553	-0	-0	250000	1.37e+06	3.87e+06	0.0645995	0.354005
ds100.a1.d	-2206	-2206	-2206	-0	-0	710000	2.22e+06	1.131e+07	0.0627763	0.196286
ds100.a2.p	-437	-448	-448	0.0245536	-0	170000	1.01e+06	2.44e+06	0.0696722	0.413934
ds100.a2.d	-1727	-1727	-1727	-0	-0	530000	1.64e+06	4.86e+06	0.109054	0.337449
ds100.b1.p	-451	-451	-451	-0	-0	250000	1.44e+06	3.91e+06	0.0639387	0.368286
ds100.b1.d	-2078	-2078	-2078	-0	-0	770000	2.37e+06	9.33e+06	0.0825295	0.254019
ds100.b2.p	-443	-530	-530	0.164151	-0	200000	1.3e+06	3.37e+06	0.0593472	0.385757
ds100.b2.d	-1936	-1936	-2113	0.0837672	0.0837672	720000	2.25e+06	7.34e+06	0.0980927	0.306539
ds100.c1.p	-468	-468	-468	-0	-0	230000	1.39e+06	3.81e+06	0.0603675	0.364829
ds100.c1.d	-2127	-2127	-2127	-0	-0	820000	2.56e+06	1.008e+07	0.0813492	0.253968
ds100.c2.p	-529	-529	-529	-0	-0	230000	1.36e+06	3.69e+06	0.0623307	0.368564
ds100.c2.d	-2112	-2112	-2112	-0	-0	830000	2.62e+06	1.513e+07	0.0548579	0.173166
ds100.d1.p	-461	-546	-546	0.155678	-0	240000	1.78e+06	5.17e+06	0.0464217	0.344294
ds100.d1.d	-2169	-2169	-2169	-0	-0	890000	3.65e+06	1.072e+07	0.0830224	0.340485
ds100.d2.p	-446	-446	-446	-0	-0	220000	1.53e+06	4.12e+06	0.0533981	0.371359
ds100.d2.d	-1783	-1783	-1783	-0	-0	900000	3.02e+06	1.03e+07	0.0873787	0.293204

Table 13.10: Shortest path of 3-hours service window

case	NDCA sol	CSPA sol	CDSPA sol	gap1	gap2	cpu time1	cpu time2	cpu time3	cpu rat1	cpu rat2
ds150.a.p	-461	-501	-542	0.149446	0.0756458	510000	3.47e+06	1.406e+07	0.0362731	0.246799
ds150.a.d	-1983	-2036	-2226	0.109164	0.0853549	1.72e+06	6.02e+06	6.627e+07	0.0259544	0.0908405
ds150.b.p	-558	-558	-558	-0	-0	500000	3.37e+06	1.011e+07	0.049456	0.333333
ds150.b.d	-2185	-2185	-2185	-0	-0	1.82e+06	6.03e+06	7.373e+07	0.0246847	0.0817849
ds150.c1.p	-562	-562	-562	-0	-0	550000	3.5e+06	1.19e+07	0.0462185	0.294118
ds150.c1.d	-2004	-2004	-2157	0.0709318	0.0709318	1.56e+06	5.49e+06	5.961e+07	0.0261701	0.0920986
ds150.c2.p	-457	-523	-542	0.156827	0.0350554	480000	3.35e+06	1.147e+07	0.0418483	0.292066
ds150.c2.d	-2199	-2199	-2339	0.0598546	0.0598546	1.88e+06	7.29e+06	4.1768e+08	0.00450105	0.0174535
ds150.d1.p	-571	-572	-572	0.00174825	-0	540000	3.77e+06	1.117e+07	0.0483438	0.337511
ds150.d1.d	-2245	-2245	-2245	-0	-0	1.86e+06	6.24e+06	4.086e+07	0.0455213	0.152717
ds150.d2.p	-450	-450	-450	-0	-0	500000	4e+06	1.146e+07	0.04363	0.34904
ds150.d2.d	-2105	-2105	-2105	-0	-0	1.58e+06	6.39e+06	2.916e+07	0.0541838	0.219136
ds200.c.p	-548	-554	-554	0.0108303	-0	820000	7.69e+06	3.353e+07	0.0244557	0.229347
ds200.c.d	-2339	-2339	-2339	-0	-0	3.29e+06	1.402e+07	5.9476e+08	0.00553164	0.0235725
ds200.d.p	-455	-527	-527	0.136622	-0	920000	7.08e+06	2.686e+07	0.0342517	0.263589
ds200.d.d	-2248	-2248	-2248	-0	-0	3.32e+06	1.658e+07	1.77237e+09	0.0018732	0.0093547
ds200.a1.p	-555	-561	-564	0.0159574	0.00531915	870000	6.05e+06	2.133e+07	0.0407876	0.283638
ds200.a1.d	-2239	-2240	-2262	0.010168	0.00972591	3.18e+06	1.129e+07	2.0336e+08	0.0156373	0.0555173
ds200.b.p	-562	-564	-564	0.0035461	-0	920000	7.82e+06	3.542e+07	0.025974	0.220779
ds200.b.d	-2438	-2438	-2438	-0	-0	3.51e+06	1.47e+07	1.04743e+09	0.00335106	0.0140343

Table 13.11: Shortest path of 3-hours service window

much time is consumed on finding a better solution, but not converge to the optimal solution.

2. Second is the subproblem solution quality. Besides the increasing number of candidate labels in CDSPA, the columns which have been generated have very closed reduced cost. It's quite often a situation that a number of generated columns have very little difference value bwteen 1 or 2.

   A even worse situation is, as no time window constarints in the shortest path subproblem, the generated paths are more likely to be symmetric. For instance, a negative cost path visits customer 1,3,5,6 sequently. Thus we will have another path 6,5,3,1 of the same reduced cost. When doing combination operation to construct a combination path, this symmetry situation gets enlarged. By enumerative combination, more symmertic combination paths will be generated.

   Affected by these closed value columns and similar form columns, the master problem will spend more time to price out the non-basis variables, which makes the convergence very slow.

3. Another reason in that the generated similar columns bring more variables into the master problem, which leads to the situation of having more fraction solution variables. Thus most of the computation effor will be consumed on branch and bound process.

To make the testing practical and more realistic, as the method used in the subproblem algorithm testing, We generated a tight service time window constraint for each customer's node. See the generation method in the previous section. We introduce each column first.

1. column 1: name of testing case;
2. column 2: objective value of start point;
3. column 3: the lower bound;
4. column 4: the first integer solution objective value;
5. column 5: the final solution objective value;
6. column 6: cpu time using to find the lower bound;
7. column 7: cpu time using to find the first integer solution;
8. column 8: cpu time using to find the final integer solution;
9. column 9: the optimal gap of low bound solution;
10. column 10: the optimal gap of first integer solution;
11. column 11: the iterations to find the lower bound;

12. column 12: the branch nodes number;

Table how the result from 10 to 20 customers. With 10 customers, 3 cases got the optimal integer solution without branching and bound and 2 cases reached the optimality as the first integer solution. With 20 customers, 3 cases reached the optimality as the first integer solution. The two gaps are both small. With 10 customers, the lower bound gap is around 0.01 to 0.03, the 1st integer solution gap is around 0.03. With 20 customere, the lower bound gap becomes smaller from 0.0003 to 0.01, the 1st integer solution gap is from 0.0005 to 0.01. It runed 3 to 7 iterations to reach the lowe bound in 10 customers, 7 to 15 iterations in 20 customers. The branching and bound nodes number is rather different between 10 and 20 customers. With 10 customers, average 1 to 30 nodes were branched and bound. With 20 customers, the branching nodes grow from 30 to more than 600, the most case reached 2469. However the cpu time using to find the 1st integer solution and optimal solution aren't very long, which is 1 to 2 times of finding the lower bound. These result show that the two algorithm work efficiently to reach a good lower bound and fast to find the optimal solution in the situtation with less customers.

The testing result of 50 customer is shown in table The algorithm can't stop runing in 2 hours as the huge searching tree. We stop the run after 2 hours. To be consistent, we use the best integer solution value as the "optimal" objective value to calculate gaps in columns 9 and 10.

As the customer increase, the two gaps increased a little bit. The lower bound optimal gap increases to 0.04 to 0.10; the first integer solution optimal gap increaed to 0.02 to 0.10. It runed 25 to 35 iterations to reach the lower bound.

However the searching tree growed fast, the branching and bound nodes increased drasticly. On average 2000 to 6000 nodes have to be branched. Most of the time was spent in branch and bound operation, which makes the convergence very slow. The cpu time using to find the first integer solution grows to 9 to 11 times of finding the lower bound. Although the increasing computaion effort in branching, the solution value improvement is different. Averagely the objective value can get reduced around 200 to 300. In the best case the reduced value is 700; yet in the worst case only 43.

When the customers increased to 100, we can't find a integer solution within 2 hours. At the end of Table we also include the testing result of 100 customers. The lower bound found was average one third of the start point solution.

Above result show that the column generation solution of the cross-docking problem is able to reach the lower bound efficient. And it can also find the first integer solution without a rather long time consuming process by branching and bound operation. However the convergence is slow after reach the first integer solution. The new integer solutions found later after are having closed objective value below or above the upper bound. The occurence of upper bound updating is not frequently and reduced objective value is very small. This make extremely slow convergence in the later computaion.

In addition to the bad start point applied as we discussed in section another two reasons seem most probably lead to such situation.

1. The quality of the adding cloumns set. Each time we selected the 100 best columns(if exists) of the minimal cost to add into RMP. However we haven't consider the variety between each column in the adding set. The column generation technique is built on the simplex method of using dural variables in the subproblem in order to lead to find negative reduced cost columns to price out non-basis variables in the RMP. Thus the dual variables are crucial in deciding the quality of the columns found in subproblem. The similarity of the adding columns increases the chance of more columns becoming redudant in getting the dual solution. Less effective columns make slow change in the dual variables, in which misleads the subproblem solving.
2. The quality of each geneated cloumn. As we don't have a upper bound of the total paths required to delivery all orders, we can't use the bound check condition(. It will be the situation that the best or good negative cost paths found are infeasible solution as they already exceed the lower bound of the shortest path problem. Thus without the upper bound k, $k = \sum_i \lambda_i$, we can't control the quality of a generated column.
3. The distribution of pickup nodes and delivery nodes. It is average situation that a customer's delivery node is far away than the pickup node from the depot. The distance of a delivery node to the depot is around 2 to 3 times of a pickup node. Thus we assigned the maximum path duration of 4-hours and 12-hours for each subpart, respectively.

   This distribution situation leads to different convergence of finding negative cost path in the subproblem. The pickup subpart is always fast than the delivery subpart. When there is no negative cost path found in pickup subpart, more attractive negative cost paths are existing in the delivery subpart. This unbalanced convergence reduced the chance of generating good quality combination paths. As one subpart has none candidate path, we can only add single paths into RMP.

13.3 Increase Maximum Capacity Resource

A cross-docking delivery is to find the shortest path to deliver orders and saving loading time of connective deliverys at depot. It's a straightforward expectation that with an increased maximum capacity of vehicle, the This section we

case	start	LB	1st	last	cpu time LB	cpu time 1st	cpu time obj	gap1	gap2	it	bound nodes
ds10.a1	4184	1942.67	1978	1968	50000	70000	90000	0.0128726	-0.0050813	3	2
ds10.a2	3886	1368.67	1482	1423	90000	120000	410000	0.0381822	-0.0414617	8	18
ds10.a3	4394	2152	2154	2154	50000	50000	60000	0.000928505	0	4	1
ds10.a4	4148	1808	2032	1899	80000	190000	360000	0.04792	-0.0700369	7	13
ds10.a5	3978	2024	2024	2024	60000	60000	60000	0	0	4	0
ds10.b1	3808	1611.89	1793	1640	100000	170000	780000	0.0171409	-0.0932927	5	32
ds10.b2	4630	2407.5	2441	2441	30000	40000	50000	0.0137239	0	3	1
ds10.b3	3906	1765.83	1922	1850	60000	80000	440000	0.0454955	-0.0389189	5	17
ds10.b4	4262	1944	1944	1944	60000	60000	60000	0	0	4	0
ds10.b5	4180	1939	1939	1939	50000	50000	50000	0	0	5	0
ds20.a1	8578	3528.33	3598	3585	360000	870000	3.17e+06	0.0158066	-0.00362622	7	35
ds20.a2	8058	3225.8	3238	3223	560000	900000	1.47e+06	-0.000868756	-0.00465405	8	7
ds20.a3	8192	2464.5	2469	2469	600000	630000	660000	0.0018226	0	9	1
ds20.a4	8324	3059.63	3359	3122	440000	1.86e+06	1.4407e+08	0.0199792	-0.0759129	8	2469
ds20.a5	8790	3115.3	3481	3173	440000	1.28e+06	4.477e+07	0.0181847	-0.097069	9	648
ds20.b1	9050	3460	3603	3534	410000	720000	3.5e+06	0.0209394	-0.0195246	10	36
ds20.b2	8788	3677.25	3694	3694	290000	410000	580000	0.00453438	0	6	6
ds20.b3	8464	2968	3340	2976	400000	900000	3.82e+06	0.00268817	-0.122312	7	50
ds20.b4	8498	3477.75	3486	3486	360000	520000	710000	0.00236661	0	7	4
ds20.b5	8426	2951.5	3013	2980	390000	580000	4.05e+06	0.00956376	-0.0110738	7	33

Table 13.12: Cross-Docking solution

case	start	LB	1st	last	cpu time LB	cpu time 1st	gap1	gap2	it	bound nodes
ds50.a1	21172	6757.42	8241	7976	6.367e+07	7.772e+08	0.152781	-0.0332247	31	2133
ds50.a2	20492	7379.52	8408	8245	3.4e+07	3.6021e+08	0.10497	-0.0197696	27	2918
ds50.a3	21220	8281.99	9333	8570	1.272e+07	8.916e+07	0.0336064	-0.0890315	27	4545
ds50.a4	21704	9061.7	10142	9437	1.244e+07	1.1391e+08	0.0397687	-0.0747059	22	5030
ds50.b1	21274	7806.35	9325	8368	1.693e+07	2.1356e+08	0.0671188	-0.114364	24	3601
ds50.b2	22122	8463.18	8999	8896	1.863e+07	1.7041e+08	0.048653	-0.0115782	23	3859
ds50.b3	21018	7764.26	8549	8290	2.176e+07	1.6157e+08	0.0634186	-0.0312425	25	5361
ds50.b4	20888	7572.66	8390	8267	4.272e+07	4.2901e+08	0.083989	-0.0148784	34	2829
ds50.c1	20454	8334.49	9717	9232	9.451e+07	7.0283e+08	0.0972178	-0.0525347	30	2714
ds50.c2	20174	7192.1	8385	8176	4.831e+07	4.9573e+08	0.12034	-0.0255626	28	2607
ds50.c3	21466	7794.21	8438	8158	2.3e+07	1.7699e+08	0.0445926	-0.0343221	23	3940
ds50.c4	20336	7680.83	8470	8427	4.622e+07	2.3403e+08	0.0885451	-0.00510265	25	4250
ds50.d1	21518	7952.97	9885	8558	1.4606e+08	9.7854e+08	0.0706973	-0.15506	35	2757
ds50.d2	22592	8046.21	8850	8713	6.622e+07	4.7937e+08	0.0765287	-0.0157236	29	2838
ds50.d3	22050	8295.88	9472	8855	1.901e+07	1.6544e+08	0.0631417	-0.0696781	27	4747
ds50.d4	20892	8424.19	9802	9107	2.08e+07	1.7212e+08	0.0749766	-0.0763149	24	5955
ds100.a1	41664	12805.8	-1	41664	5.98735e+08	0	0.692642	1.00002	69	1
ds100.a2	42924	16044.6	-1	42924	1.01571e+09	0	0.626209	1.00002	57	27
ds100.b1	43396	15029.7	-1	43396	-1.73782e+09	0	0.653661	1.00002	60	22
ds100.b2	41906	14081.3	-1	41906	-1.09234e+09	0	0.663978	1.00002	74	1
ds100.c1	41754	14450.7	-1	41754	-2.12457e+09	0	0.653909	1.00002	65	20
ds100.c2	41746	14582.5	-1	41746	-3.42727e+08	0	0.650684	1.00002	65	22
ds100.d1	43948	14484.9	-1	43948	1.43102e+09	0	0.670407	1.00002	77	1
ds100.d2	41956	14510.8	-1	41956	-1.89646e+09	0	0.654142	1.00002	76	1

Table 13.13: Cross-Docking solution

made testing of increasing the maximum capacity of each vehicle and see how it affected the cross-docking problem.

To keep the testing consistent, we use the same time window in table nd only increase one time of capacity of each vehicle. Now the maximum capacity is $2CAP$, the testing result is shown in talbe Due to the optimality reason, we only include the testing case with 10 to 20 customers. The solution of each testing case reached the optimality.

The objective value decreased as the maximum capacity increased. Within 10 customers, the average objective value decreased 40. Within 20 customers, the average objective value decreased 130 to 180, and in some cases the decreased value reached to 260 and 230. The result show that increasing the capacity helps to decrease the total delivery cost. As the capacity increased, a vehicle tries to pickup or deliver more orders by one route within the time window constraint, in which the total traveling distance decreases. This also increased the possibility of generation of more connective deliverys, in which the loading time of connective deliverys is saved.

The computation effort increased as the maximum capacity increased. With 10 customers, the iterations of finding the lower bound didn't change much. With 20 customers, the iterations increased 1.5 to 2 times. The cpu time also showed this change. From 10 to 20 customers, the cpu time used to find the lowe bound increased from 1.5 to 2 time, respectively. The coveragence didn't change too much. However with 20 customers, the branching and bound nodes number increased 2 to 7 times. This shows that an increased maximum capacity brought more fraction solution variables into the master problem.

13.4 Time Constraint of Subpath

As the unbalanced distance distribution of pickup nodes and delivery nodes, we assigned the maximum duration of pickup subpath and delivery subpath with 4-hours and 12-hours, respectively. In the section, we'll study the sensitivity of this time constraint assignment and how it affects the cross-docking problem.

We made testing of two time constraint situtations. One is 6-10 time constraint, in which the maximum duration of pickup subpath is 6-hours and the maximum duration of delivery subpath is 10-hours. The other is 8-8 time constraint. Tablend howed the testing result, respectively. We made comparation of these two situations of time constraint assignment with 4-12 situation.

Under 6-10 situaion, the objective value of almost every testing case got increased. With 10 customers, the objective value averagely increased 100 to 300. In some cases, the value decreased 60 to 100. With 20 custimers, the objective value averagely increased 400 to 600. Under 8-8 situation, the increased

case	start	LB	1st	last	cpu time LB	cpu time 1st	cpu time obj	gap1	gap2	it	bound nodes
ds10.a1	4184	1903.67	1927	1927	80000	120000	150000	0.0121086	0	5	2
ds10.a2	3886	1336.25	1426	1384	150000	330000	1.51e+06	0.0345014	-0.0303468	10	25
ds10.a3	4394	2070	2070	2070	80000	90000	90000	0	0	6	0
ds10.a4	4148	1735	1878	1814	120000	230000	610000	0.0435502	-0.0352811	9	10
ds10.a5	3978	1977	1977	1977	100000	110000	110000	0	0	7	0
ds10.b1	3808	1522	1535	1535	170000	250000	300000	0.00846906	0	8	1
ds10.b2	4630	2400.5	2441	2441	40000	60000	60000	0.0165916	0	4	1
ds10.b3	3906	1727.33	1810	1746	110000	150000	280000	0.0106911	-0.0366552	7	3
ds10.b4	4262	1874.5	1879	1871	100000	140000	250000	-0.00187066	-0.00427579	7	3
ds10.b5	4180	1913	1913	1913	70000	70000	70000	0	0	7	0
ds20.a1	8578	3365.5	3762	3408	700000	3.65e+06	1.1759e+08	0.0124707	-0.103873	11	670
ds20.a2	8058	2966	2966	2966	1.65e+06	1.66e+06	1.66e+06	0	0	14	0
ds20.a3	8192	2335	2335	2335	1.79e+06	1.81e+06	1.81e+06	0	0	16	0
ds20.a4	8324	2941.83	3222	2973	660000	2.99e+06	-1.09494e+09	0.0104832	-0.0837538	11	16451
ds20.a5	8790	2980.25	3069	3053	740000	2.07e+06	2.3039e+08	0.023829	-0.00524075	13	1098
ds20.b1	9050	3295.4	3422	3402	740000	1.25e+06	6.77e+06	0.0313345	-0.00587889	15	31
ds20.b2	8788	3458.67	3466	3460	780000	1.15e+06	1.58e+06	0.000385356	-0.0017341	11	2
ds20.b3	8464	2814.12	2815	2815	1.15e+06	1.27e+06	1.36e+06	0.000313447	0	13	1
ds20.b4	8498	3357.75	3320	3320	550000	1.52e+06	3.07e+06	-0.0113705	0	9	10
ds20.b5	8426	2837.55	2927	2872	1e+06	1.86e+06	2.08e+07	0.0119951	-0.0191504	15	62

Table 13.14: Cross-Docking with 2CAP

objective value got enlarged. With 10 customers, the objective value averagely increased 300 to 1000. With 20 custimers, the objective value averagely increased 1300 to 2000.

In addition to the increased objective value, the computation effort also got increased. Under 6-10 situation, the cpu time used to find the lower bound increased 1.5 to 2 times; and the iterations increased 1 to 1.5 times. Under 8-8 situation, the cpu time used to find the lower bound increased 2 to 4 times; and the iterations increased 1 to 2 times. In ordert to find the integer solution, the increasing of branching and bound nodes number is diverse. With 10 customers, both two situations didn't change too much, except one case under 8-8 situation, the branching nodes increased to 13 times. With 20 customers, the branching nodes increased drasticly. Under 6-10 situation, average 2000 or 10000 nodes are branched. Under 8-8 situation, average 300 to 400 nodes are branched, in some cases reached to more than 1000, the extreme case reached 13680 branching nodes.

Above result show that the maximum duration time constraint assigment of pickup and delivery subpath should be selected delicatedly. As an unlogical time constraint assignment will not only increase the objective value but also increase the computation effort drasticly. A wise maximum duration assignment is selected according to the real distance distribution of the pickup nodes and delivery nodes.

13.5 Solve Cross-Docking Problem Using NDCA Only

We include the testing result of solving Cross-Docking problem using the NDCA only. Without the CDSPA and the OCDSPA, some parts of the feasible solution set are obviously cut off. The objective solution using the NDCA only can't be proved optimality. Table howed the testing result.

With 10 customers, the objective value averagely increased 50 to 300; with 20 customers, the objective value averagely increased 500 to 900. As the feasible solution set was cut off smaller, the computation effort decreased also. The cpu time used to find the lower bound decreased to 0.5 to 1 time. However, in order to find the integer solution, the cpu time increased a little bit. The cpu time used to find the optimal integer solution increased 13 to 30 times. The drasticly increased branching nodes number also shows the increasing computation effort.

Table howed the testing result of 2-times of the maximum capacity constraint situation. As showed in previously sections, the objective value decreased and the computation effort increased as the maximum capacity increased.

case	start	LB	1st	last	cpu time LB	cpu time 1st	cpu time obj	gap1	gap2	it	bound nodes
ds10.a1	4184	1943	1990	1950	110000	290000	1.31e+06	0.00358974	-0.0205128	5	23
ds10.a2	3886	1650	1650	1650	200000	210000	210000	0	0	10	0
ds10.a3	4394	2001.38	2048	2031	110000	220000	1.37e+06	0.0145864	-0.00837026	7	23
ds10.a4	4148	1801	1833	1810	130000	300000	960000	0.00497238	-0.0127072	7	15
ds10.a5	3978	2351	2351	2351	90000	100000	100000	0	0	6	0
ds10.b1	3808	1703.75	1795	1730	140000	280000	1.49e+06	0.0151734	-0.0375723	7	26
ds10.b2	4630	2356.5	2385	2385	70000	90000	120000	0.0119497	0	5	2
ds10.b3	3906	1988	1988	1988	70000	80000	80000	0	0	6	0
ds10.b4	4262	2160.33	2369	2183	100000	170000	340000	0.0103833	-0.0852038	7	7
ds10.b5	4180	2165.71	2174	2174	70000	90000	110000	0.00381128	0	6	1
ds20.a1	8578	3874	4132	3955	860000	4.86e+06	-1.10034e+09	0.0204804	-0.0447535	11	16439
ds20.a2	8058	3551.43	3953	3631	1.16e+06	7.05e+06	9.4002e+08	0.0219152	-0.0886808	14	4320
ds20.a3	8192	3076.87	3448	3103	1.26e+06	1.219e+07	-8.36597e+08	0.00842149	-0.111183	13	13170
ds20.a4	8324	3450.92	3490	3448	970000	3.39e+06	2.13e+07	-0.00084776	-0.012181	11	56
ds20.a5	8790	3651	3970	3777	2.48e+06	1.449e+07	7.2889e+08	0.0333598	-0.0510988	16	3197
ds20.b1	9050	3866.36	4364	3961	470000	2.45e+06	1.35628e+09	0.0238937	-0.101742	9	10543
ds20.b2	8788	4136.4	4141	4129	530000	650000	1.47e+06	-0.0017922	-0.00290627	9	6
ds20.b3	8464	3170.81	3341	3208	1.12e+06	5.42e+06	5.367e+07	0.0115917	-0.0414589	13	218
ds20.b4	8498	3637.98	4207	3688	650000	2.92e+06	3.0268e+08	0.013563	-0.140727	12	1840

Table 13.15: Cross-Docking solution of 6-10 situation

case	start	LB	1st	last	cpu time LB	cpu time 1st	cpu time obj	gap1	gap2	it	bound nodes
ds10.a1	4184	2220.5	2271	2258	160000	430000	1.05e+06	0.0166076	-0.00575731	8	12
ds10.a2	3886	2165	2165	2165	180000	190000	190000	0	0	8	0
ds10.a3	4394	2678.25	2784	2693	100000	280000	980000	0.00547716	-0.0337913	4	17
ds10.a4	4148	2215	2263	2238	80000	180000	730000	0.010277	-0.0111707	5	20
ds10.a5	3978	2473.43	2479	2476	110000	150000	200000	0.00103854	-0.00121163	8	2
ds20.a1	8578	4723.55	4741	4726	1.13e+06	2.92e+06	9.26e+06	0.000519371	-0.00317393	11	20
ds20.a2	8058	4118.67	4217	4133	960000	3.93e+06	8.624e+07	0.00346802	-0.0203242	11	299
ds20.a3	8192	3821.5	3894	3825	1.15e+06	5.02e+06	3.137e+07	0.000915033	-0.0180392	10	100
ds20.a4	8324	4386.6	4492	4460	1.73e+06	3.62e+06	-2.02073e+09	0.0164574	-0.00717489	13	13680
ds20.a5	8790	5103.47	5217	5092	7.32e+06	4.141e+07	2.0161e+08	-0.0022519	-0.0245483	12	312
ds10.b1	3808	1865	2014	1968	110000	330000	1.212e+07	0.0523374	-0.023374	6	357
ds10.b2	4630	3498.9	3541	3516	70000	150000	470000	0.00486348	-0.00711035	6	14
ds10.b3	3906	2208.5	2224	2216	70000	100000	150000	0.00338448	-0.00361011	6	3
ds10.b4	4262	2837	2837	2837	90000	90000	90000	0	0	6	0
ds10.b5	4180	3043.33	3194	3066	60000	260000	690000	0.00739291	-0.0417482	4	18
ds20.b1	9050	5482.67	5807	5531	350000	2.06e+06	8.226e+07	0.00873862	-0.0499006	8	729
ds20.b2	8788	5400.88	5577	5450	620000	2.21e+06	1.695e+08	0.00901376	-0.0233028	10	1080
ds20.b3	8464	4227.89	4465	4332	1.01e+06	3.96e+06	9.263e+07	0.0240317	-0.0307018	11	348
ds20.b4	8498	4637.88	5039	4766	720000	3.98e+06	5.125e+07	0.0268831	-0.0572807	11	339
ds20.b5	8426	4311.75	4528	4337	2.2e+06	9.96e+06	3.8176e+08	0.005822	-0.0440397	15	1288

Table 13.16: Cross-Docking solution of 8-8 situation

Above testing results both showed the advantage of applying the CDSPA and the OCDSPA in solving the Cross-Docking problem.

case	start	LB	1st	last	cpu time LB	cpu time 1st	cpu time obj	gap1	gap2	it	bound nodes
ds10.a1	4184	1946	1979	1965	40000	70000	110000	0.00768836	-0.00922604	3	5
ds10.a2	3886	1667.67	1826	1718	70000	140000	1.44e+06	0.0292976	-0.0628638	4	79
ds10.a3	4394	2152	2154	2154	60000	60000	60000	0	0	4	0
ds10.a4	4148	1885	1920	1905	60000	90000	190000	0.0104987	-0.00787402	4	6
ds10.a5	3978	2099	2099	2099	50000	50000	50000	0	0	4	0
ds10.b1	3808	1736.5	1868	1849	50000	110000	670000	0.0608437	-0.0102758	4	35
ds10.b2	4630	2549.83	2588	2588	40000	50000	60000	0.0147476	0	4	1
ds10.b3	3906	1968	2059	2012	60000	100000	320000	0.0218688	-0.0233598	5	11
ds10.b4	4262	2018	2087	2031	70000	110000	160000	0.00640079	-0.0275726	5	4
ds10.b5	4180	2070.33	2241	2227	50000	70000	360000	0.0703488	-0.00628648	4	24
ds20.a1	8578	3951.3	4044	3974	260000	500000	4.18e+06	0.00571213	-0.0176145	6	67
ds20.a2	8058	3724.86	3919	3766	340000	1.03e+06	2.506e+07	0.0109231	-0.0406267	7	376
ds20.a3	8192	3322.05	3623	3362	430000	2.07e+06	8.543e+07	0.0118838	-0.0776324	9	1076
ds20.a4	8324	3282.18	3634	3399	300000	1.07e+06	7.6584e+08	0.0343677	-0.069138	7	11425
ds20.a5	8790	3177	3317	3191	330000	650000	1.86e+06	0.00438734	-0.0394861	7	22
ds20.b1	9050	4294.14	4545	4447	220000	600000	1.9319e+08	0.0343731	-0.0220373	7	4330
ds20.b2	8788	4147.17	4263	4164	270000	670000	7.86e+06	0.00404259	-0.0237752	7	109
ds20.b3	8464	3453.67	3663	3490	290000	1.16e+06	5.041e+07	0.0104107	-0.0495702	8	694
ds20.b4	8498	3790.27	3940	3830	270000	550000	1.971e+07	0.0103736	-0.0287206	6	296
ds20.b5	8426	3400	3771	3475	440000	1.47e+06	1.5139e+08	0.0215827	-0.0851799	9	1679

Table 13.17: Cross-Docking solution using NDCA only

case	start	LB	1st	last	cpu time LB	cpu time 1st	cpu time obj	gap1	gap2	it	bound nodes
ds10.a1	4184	1913.67	1949	1949	60000	90000	130000	0.018129	0	4	2
ds10.a2	3886	1604.33	1846	1679	110000	400000	6.11e+06	0.0444709	-0.099464	6	159
ds10.a3	4394	2070	2070	2070	70000	70000	70000	0	0	6	0
ds10.a4	4148	1844.5	2003	1876	90000	310000	1.29e+06	0.016791	-0.0676972	6	31
ds10.a5	3978	2061	2061	2061	90000	90000	90000	0	0	7	0
ds10.b1	3808	1708.5	1886	1779	100000	280000	5.62e+06	0.039629	-0.0601461	6	160
ds10.b2	4630	2525.5	2558	2558	70000	90000	110000	0.0127052	0	7	1
ds10.b3	3906	1921	1935	1935	120000	160000	200000	0.00723514	0	8	1
ds10.b4	4262	1927	1927	1927	100000	100000	100000	0	0	8	0
ds10.b5	4180	2005	2166	2130	80000	150000	740000	0.0586854	-0.0169014	6	21
ds20.a1	8578	3751.74	3882	3811	560000	1.47e+06	1.0083e+08	0.0155495	-0.0186303	11	682
ds20.a2	8058	3514.09	3626	3551	580000	1.95e+06	1.635e+07	0.0103951	-0.0211208	10	82
ds20.a3	8192	3145.29	3649	3189	1.09e+06	6.45e+06	5.8232e+08	0.013707	-0.144246	15	2603
ds20.a5	8790	3028.33	3081	3059	670000	1.21e+06	6.41e+06	0.0100251	-0.00719189	11	23
ds20.b1	9050	4063.57	4313	4217	410000	1.21e+06	5.8327e+08	0.0363833	-0.022765	11	5771
ds20.b2	8788	3943.71	3986	3963	510000	1.64e+06	4.997e+07	0.00486644	-0.00580368	10	277
ds20.b3	8464	3223.69	3405	3319	620000	2.27e+06	2.9994e+08	0.0287179	-0.0259114	12	1558
ds20.b4	8498	3619.82	3838	3701	460000	2.17e+06	8.1068e+08	0.0219345	-0.037017	9	5772
ds20.b5	8426	3234.47	3453	3324	770000	2.68e+06	3.4886e+08	0.0269354	-0.0388087	13	1534

Table 13.18: Cross-Docking solution of 2CAP using NDCA only

Chapter 14

Conclusion and Recommendation for Future Research

In this dissertation we apply column generation techniques in solving the Cross-Docking problem into optimality. The numeric result shows that the solution method is promising for the complex NP-hard problem. The linear programing bound obtained from an extensive reformulation is stronger. And the knowledge if the original compact formulation provides with a strong guide from branching and cutting decisions in the searching tree. It is a successful implemenation of column generation for the Cross-Docking problem.

The result shows that for very hard problems, it always exists a tradeoff between computation effort and solution quality. As the subproblem needs to be solved repeatedly, it's crucial to focus on the algorithm efficiency by developing a heuristic method or algorithm of finding a closed optimal solution, instead of an exact optimal solution.

In the Cross-Docking problem, three algorithms of solving the subproblem are efficient. The NDCA and the CDSPA can find closed optimal solution of single path or combination path. The later type of path is the key in solving the Cross-Docking problem. These two algorithms efficiently add new negative reduced columns into the restricted master problem. The optimal algorithm OCDSPA is only necessarily applied at a final stage to prove the optimality.

The further testing case shows that the Cross-Docking problem is sensitive to the maximum duration time window constraint of each subpart. The sensitivity is shown both in objective value and in computation effort. A good maximum duration time constraint assignment is selected logically corresponding to the distance distribution of pickup and delivery nodes, which helps to decrease the objective value and save the computation effort.

Another factor directively affects the objective value is the maximum capacity resource. Increaseing the vehicle onloading capacity leads to a decreased objective value by the increasing possiblity of finding a better shorest paths set or making more connective deliverys.

During the implementation of column generation to solve the Cross-Docking problem, the numeric result shows that the selected initial start point will affect the performance of the convergence. A rough start point leads astray the constructure of the basis in which makes the searching become huge. An improvement can be achieved by using a warm start.

As the unbalanced optimality convergence in each subpart, we expect that solving each subpart as an independent VRP to a closed optimal solution first, then combining two subparts together to find the negative cost single path or combination path, will improve the situation. After solving each subpart independently, the columns of each subpart are very closed to the optimal solution of the Cross-Docking problem. Thus the combination paths found by the NDCA or CDSPA, OCDSPA lead the convergence to the optimal solution.

Another aspect for further research is to deal with the time window constraint. The combined time constraint and the sequence time constraint affect both the objective value and the computation effort.

Bibliography

[1] .Desrochers, F.Soumis, "A generalized permanent labelling algorithm for the shortest path problem with time window", *INFOR* vol.26 (1988), no.3

[2] .B.Cunha,J.Swait, "New dominance criteria for the generalized permanent labelling algorithm for the shortest path problem with time windows on dense graphs", *INTERNATIONAL TRANSACTIONS IN OPERATION RESEARCH*, 7 (2000), 139-157

[3] .Righini, M.Salani, "Dynamic programming algorithms for the elementary shortest path problem with resource constraints", *ELectronic Notes in DESCRETE MATHEMATICS*, 17 (2004) 247-249

[4] .H.Hartel, H.Glaser, "The resrouce constrained shortest path problem implemented in a lazy functional language", *J.Functional Programming*, 1 (1): 1-100, January 1996,

[5] .Feillet,P.Dejax,M.Gendreau,C.Gueguen, "An exact algorithm for the elementary shortest path problem with resource constraint: application to some vehicle routing problems", *Networks*, 2004

[6] .Kjerrstrom, *The resource constrained shortest path problem*,

[7] .Larsen, "The dynamic vehicle routing problem", published by IMM DTU, 2000

[8] .Cook,J.L.Rich, "A parallel cutting-plane algorithm for the vehicle routing problem with time windows", ,

[9] .Desrosiers,M.E.Lubbecke, "A primer in column generation", *Column Generation*, Spring, 2005

[10] .Villeneuve,J.Desrosiers,M.E.Lubbecke,F.Soumis "On Compact Formulations",

[11] Vehicle routing with time windows and time-dependent rewards:a problem from the American Red Cross", November 19, 2003

[12] .A.Person,M.G.Lundgren, "Shipment planning at oil refineries using column generation and valid inequaliites", *Europe Journal of operation research*, 163 (2005) 631-652

[13] .Desrosiers,M.E.Lubbecke, "Selected topic in column generation", *Operation Research*, Revised March 9 2004, October 21 2004

[14] .Wilhelm, "A technical review of column generation in integer programming", *Optimization and Engineering*, 2 2001, 159-200

[15] .Brenninger-Gothe, "Two vehicle routing problems: Mathematical programming approaches", 1989

Appendix A

Algorithm

```
begin ▷ NDCA I implementation for dense graphis
▷ calculate the domension of the bucket (dim_bck)
$dim\_bck \leftarrow minimum(d_i + t_{ij})$, all$(i, j) \in A$
▷ initialize the variables which contain the current minimum cost to each node
for $i \leftarrow 0$ to $n + 1$
    do $min\_cost[i] \leftarrow \propto$
▷ generate the label corrensponding to node 0
$T_0 \leftarrow 0$; $C_0 \leftarrow 0$
while there are buckets with temporary labels to be treated
    do
        ▷ minimum cost of the labels being treated for node $i$ in the current bucket
        $aux \leftarrow \propto$
        while current bucket not empty
            do
                ▷ find the next label to be treated
                select and remove label$(T_i, C_i)$ from the bucket
                for all successors $j$ of node $i$
                    do
                        ▷ determine arrival time at $j$
                        $T_j \leftarrow T_i + d_j + t_{ij}$
                        ▷ verify time window constraint
                        if $T_j \leq b_j$
                          then
                                $T_j \leftarrow max(T_j, a_j)$
                                $C_j \leftarrow C_i + c_{ij}$
                                ▷ verify if label $(T_j, C_j)$ is not dominated:
                                ▷ compare the cost of the new path with the old path
                                if $C_j < min\_cost[j]$
                                  then
                                        calculate the bucket $K$ to store the label $(T_j, C_j)$
                                        store $(T_j, C_j)$ in $K^{th}$ bucket
                store $(T_j, C_j)$ as a permanent label
                $aux \leftarrow min(aux, C_j)$
                find the next label in the current bucket to be treated
        ▷ update the current minimum cost to node $i$
        $min\_cost[i] \leftarrow min(min\_cost[i], aux)$
        find the next bucket with temporary labels to be treated
carray out the dominance test at node $n + 1$
end
```

Figure A.1: labels treatment order in basic NDCA(NDCA I)

```
begin ▷ NDCA II implementation for dense graphis
▷ calculate the domension of the bucket (dim_bck)
$dim\_bck \leftarrow minimum(d_i + t_{ij})$, all$(i, j) \in A$
▷ initialize the variables which contain the current minimum cost to each node
for $i \leftarrow 0$ to $n + 1$
    do $min\_cost[i] \leftarrow \propto$
▷ generate the label cornesponding to node 0
$T_0 \leftarrow 0$; $C_0 \leftarrow 0$
while there are buckets with temporary labels to be treated
    do
       ▷ minimum cost of the labels being treated for node $i$ in the current bucket
       $previous_{T_j} \leftarrow \propto$
       ▷ receives the cost of the least cost label within the bucket for node i
       $aux \leftarrow C_i$
       while current bucket not empty
           do
              ▷ find the next label to be treated
              select and remove label$(T_i, C_i)$ from the bucket
              ▷ dominance test at origin i
              if $C_i < min\_cost[i]$
                then
                     if $T_i < previous\_T_i$
                       then
                            for all successors $j$ of node $i$
                                do
                                   extend label from node i to node j
                                   ▷ verify if label $(T_j, C_j)$ is not dominated:
                                   ▷ compare the cost of the new path with the old path
                                   if $C_j < min\_cost[j]$
                                     then
                                     calculate the bucket $K$ to store the label $(T_j, C_j)$
                                     store $(T_j, C_j)$ in $K^{th}$ bucket
                            store $(T_i, C_i)$ as a permanent label
                            $previous\_T_i \leftarrow T_i$
              if $T_i = a_i$
                then
                     disregard the remaining labels for node $i$
                else
                     find the next label in the current bucket to be treated
       ▷ update the current minimum cost to node $i$
       $min\_cost[i] \leftarrow min(min\_cost[i], aux)$
       find the next bucket with temporary labels to be treated
carray out the dominance test at node $n + 1$
end
```

Figure A.2: NDCA II

```
begin ▷ NDCA III implementation for dense graphis
▷ calculate the domension of the bucket (dim_bck)
$dim\_bck \leftarrow minimum(d_i + t_{ij})$, all$(i, j) \in A$
▷ initialize the variables which contain the current minimum cost to each node
for $i \leftarrow 0$ to $n + 1$
    do $min\_cost[i] \leftarrow \propto$
▷ generate the label corrensponding to node 0
$T_0 \leftarrow 0$; $C_0 \leftarrow 0$
while there are buckets with temporary labels to be treated
    do
        ▷ minimum cost of the labels being treated for node $i$ in the current bucket
        $previous_{T_j} \leftarrow \propto$
        ▷ receives the cost of the least cost label within the bucket for node i
        $aux \leftarrow C_i$
        while current bucket not empty
            do
                ▷ find the next label to be treated
                select and remove label$(T_i, C_i)$ from the bucket
                ▷ dominance test at origin i
                if $C_i < min\_cost[i]$
                  then
                        if $T_i < previous\_T_i$
                          then
                                for all successors $j$ of node $i$
                                    do
                                        if $C_j < min\_cost[j]$
                                          then
                                          calculate the bucket $K$ to store the label $(T_j, C_j)$
                                          backward look check
                                store $(T_i, C_i)$ as a permanent label
                                $previous\_T_i \leftarrow T_i$
                if $T_i = a_i$
                  then
                        disregard the remaining labels for node $i$
                  else
                        find the next label in the current bucket to be treated
        ▷ update the current minimum cost to node $i$
        $min\_cost[i] \leftarrow min(min\_cost[i], aux)$
        find the next bucket with temporary labels to be treated
carray out the dominance test at node $n + 1$
end
```

Figure A.3: NDCA III algorithm

```
begin ▷ CDSPA implementation for dense graphis
$dim\_bck \leftarrow minimum(cap_i)$, $i \in N$ ▷ calculate the domension of the bucket (dim_bck)
generate bucket list
for $i \leftarrow 0$ to $n+1$ ▷ initialize the variables which contain the current minimum cost to each node
    do $min\_cost[i] \leftarrow \propto$
for $i \leftarrow 0$ to $n+1$ ▷ initialize the variables
    do $min\_cost\_k[i] \leftarrow -1$
$storedList \leftarrow empty$ ▷ initialize master list
$waitingList \leftarrow empty$ ▷ initialize waiting list
$T_0 \leftarrow 0$; $C_0 \leftarrow 0$ ▷ generate the label corrensponding to node 0
while there are buckets with temporary labels to be treated
    do
      while there are label lists within current treated bucket
          do
            $aux \leftarrow \propto$
            $previous\_T_i \leftarrow \propto$ ▷ initialize start time of labels being treated for node $i$ in current buck
            find current minimal cost label of node $i$: $min\_cost[i]$
            while there are labels within current masterlabel list
                do minimal cost test of the current label$(T_i, C_i)$
            minimal cost test of each waiting label
            while there are labels within current masterlabel list
                do
                  select and remove label$(T_i, C_i)$ from the bucket
                  for all successors $j$ of node $i$
                      do
                        extend to new label$(T_j, C_j)$
                        store label$(T_j, C_j)$ into storedList if it is destination label
                        if $C_j \leftarrow min\_cost[j]$
                          then
                              minimal cost test of label$(T_j, C_j)$
                          else
                              calculate the bucket $K$ to store the label $(T_j, C_j)$
                              backward look check
                              store label$(T_j, C_j)$ into masterList of bucket $K$, if it not domi
                              store label$(T_j, C_j)$ into waitingList of bucket $K$, if it is promin
                  $previous\_T_i \leftarrow T_i$
                  $aux \leftarrow min(aux, C_i)$
            $min\_cost[i] \leftarrow min(min\_cost[i], aux)$
            update bucket index of current minimum cost of node $i$
            while there are labels within current waitinglabel list
                do
                  for all successors $j$ of node $i$
                      do
                        extend to new label$(T_j, C_j)$
                        store label$(T_j, C_j)$ into waitingList if it is destination label
                        if $C_j \leftarrow min\_cost[j]$
                          then
                              minimal cost test of label$(T_j, C_j)$ against minimal cost label
                          else
                              calculate the bucket $K$ to store the label $(T_j, C_j)$
                              backward look check
                              store label$(T_j, C_j)$ into masterList of bucket $K$, if it not domi
                              store label$(T_j, C_j)$ into waitingList of bucket $K$, if it is promin
            find the next bucket with temporary labels to be treated
      carray out the dominance test at node $n+1$
      end
```

Figure A.4: CDSPA algorithm

```
prominent test for each prominent label of label($T_i, C_i$)
store it into waitingList of the corresponding bucket if it is prominent label of minimal cost label
remove it from prominent list if it is not prominent label of minimal cost label
if $C_i \geq min\_cost[i]$
  then
    if label($T_i, C_i$) is prominent label of minimal cost label
      then
        store label($T_i, C_i$) into waitingList of the corresponding bucket
      else
        remove label($T_i, C_i$)
```

Figure A.5: minimal cost test algorithm

```
$InBk \leftarrow$ address of the first valid bucket for node $j$
$x \leftarrow K - 1$
$isProm \leftarrow false$
$isDomed \leftarrow false$
while not $isDomed$ and $x \geq InBk$
  do
    if $(T_j, C_j)$ dominated
      then
        $isDomed \leftarrow true$
        if $(T_j, C_j)$ is prominent label
          then
            $isProm \leftarrow true$
    $x \leftarrow x - 1$
```

Figure A.6: backward look check