CHAPTER VI

PRIMAL-DUAL COLUMN GENERATION AND PARTITIONING METHODS

We have briefly introduced the primal-dual methods in Section 2.4 and key variable decomposition in Section 2.3.3. In this chapter, we will combine these two methods to solve MCNF problems.

6.1 Generic primal-dual method

The *primal-dual method* is a dual-ascent LP solution method which starts with a feasible dual solution, and then iteratively constructs a primal feasibility subproblem called the *restricted primal problem* (RPP) based on the complementary slackness (CS) conditions. It uses the optimal dual solution of the RPP to improve the current dual solution if primal infeasibility still exists. The algorithm terminates when all primal infeasibility disappears.

6.1.1 Constructing and solving the RPP

Suppose we are given a dual feasible solution $(\sigma, -\pi)$ where each σ_k is a free dual variable associated with the convexity constraint (or, in other words, flow balance constraint) for commodity k, and each $\pi_a \geq 0$ corresponds to the bundle constraint for arc a. Using the arc-path MCNF formulation as introduced in Section 2.3.2, we can identify the set of zero-reduced-cost columns $Q = \{Q^k, \forall k \in K\}$ and A^0, where $Q^k := \{p : PC_p^{c+\pi} = \sigma_k,\ p \in P^k\}$ for each commodity k and $A^0 := \{a : \pi_a = 0\}$ for each arc a. Define $A^+ := A \backslash A^0 = \{a : \pi_a > 0\}$. For each arc $a \in A$, we assign a nonnegative slack variable $s_a = u_a - \sum_{k \in K} \sum_{p \in Q^K} (B^k \delta_a^p) f_p$ and two nonnegative artificial variables w_a^+ and w_a^- representing the amount of primal infeasibility. The complementary slackness condition (CS.1) in Section 2.3.2 implies that $s_a = 0$ for each $a \in A^+$. The RPP is to minimize the sum of all artificial variables subject to the constraints whose variables have zero reduced cost in the dual feasible solution $(\sigma, -\pi)$.

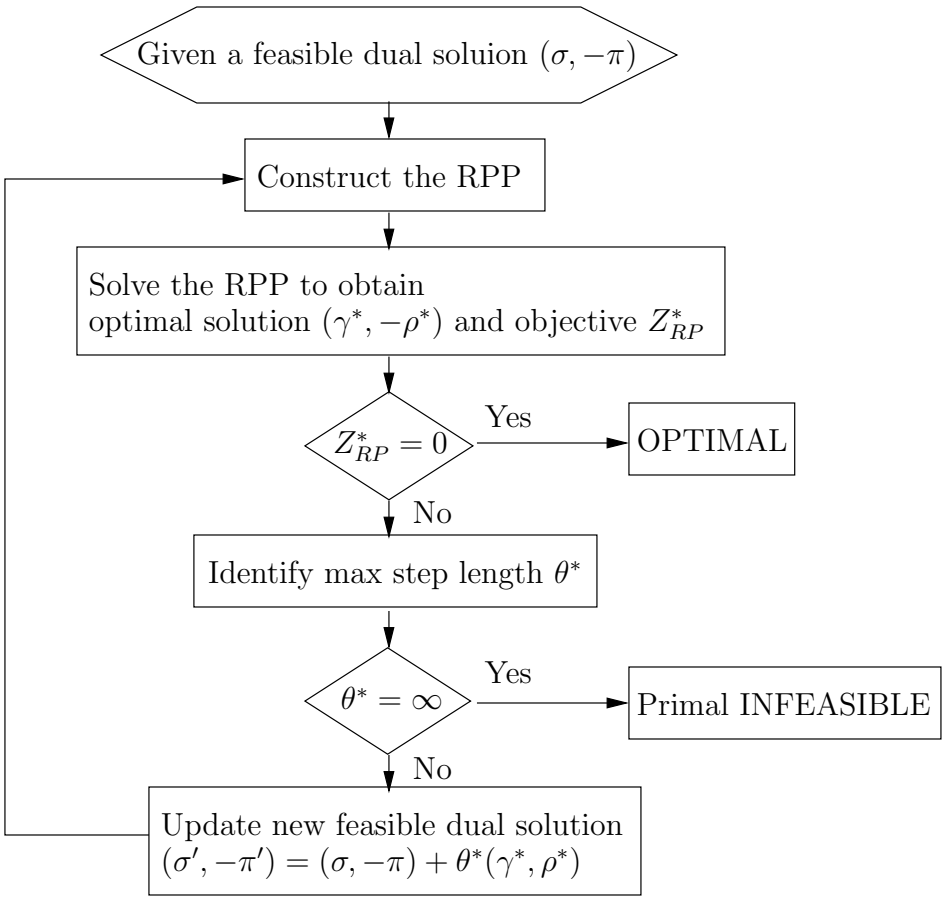


Figure 10: Generic primal-dual method for arc-path form of multicommodity flow problem

The RPP is expressed as follows:

$$\min \sum_{a \in A} (w_a^+ + w_a^-) = Z^*_{RP}(f, s, w) \qquad \text{(P\_RPP)}$$

$$s.t. \qquad \sum_{p \in Q^k} f_p = 1 \quad \forall k \in K \qquad (6.1)$$

$$\sum_{k \in K} \sum_{p \in Q^k} (B^k \delta_a^p) f_p + s_a + (w_a^+ - w_a^-) = u_a \quad \forall a \in A^0 \qquad (6.2)$$

$$\sum_{k \in K} \sum_{p \in Q^k} (B^k \delta_a^p) f_p + (w_a^+ - w_a^-) = u_a \quad \forall a \in A^+ \qquad (6.3)$$

$$f_p \geq 0 \quad \forall p \in Q^k, \forall k \in K$$

$$s_a \geq 0 \quad \forall a \in A^0 \; ; \; w_a^+ \geq 0, \; w_a^- \geq 0 \quad \forall a \in A$$

whose dual is

$$\max \sum_{k\in K} \gamma_k + \sum_{a\in A} u_a(-\rho_a) = Z^*_{RD}(\gamma,\rho) \tag{D_RPP}$$

$$s.t.\ \ \gamma_k + \sum_{a\in A} B^k \delta^p_a(-\rho_a) \le 0 \ \ \forall p \in Q^k,\ \forall k \in K \tag{6.4}$$

$$0 \le \rho_a \le 1 \ \ \forall a \in A^0$$

$$-1 \le \rho_a \le 1 \ \ \forall a \in A^+$$

$$\gamma_k : \text{free} \ \ \forall k \in K$$

where γ_k are the dual variables for the convexity constraint (6.1) and $-\rho_a$ are the dual variables for the bundle constraints (6.2) and (6.3). By defining $PC^\rho_p = \sum_{a\in A} B^k \delta^p_a \rho_a$, the first constraint (6.4) of D_RPP can be rewritten as $PC^\rho_p \ge \gamma_k$, for each $p \in Q^k$ and each $k \in K$.

RPP is a LP, thus can be solved by any LP method. Section 6.1.4 discusses the degeneracy issues when generic primal-dual methods are used to solve RPP. Section 6.2 will discuss how to use Rosen's key variable decomposition method [273] to solve the RPP.

If the optimal objective value of the RPP is positive, which means there must exist primal infeasibility when using only the current column set $\{Q \cup A^0\}$ to solve the original problem, then the optimal dual solution of the RPP can be used as an improving direction for the current dual solution (this will be explained in Section 6.1.2). On the other hand, if the optimal objective value of the RPP is zero, we have achieved primal feasibility while maintaining dual feasibility and complementary slackness; thus, we have the optimal solution.

6.1.2 Improving the current feasible dual solution

In this section, we will explain why the optimal dual solution of the RPP is an improving direction for the current feasible dual solution $(\sigma, -\pi)$.

Suppose $(\gamma^*, -\rho^*)$ is an optimal dual solution for the RPP. Let $(\sigma', -\pi') := (\sigma, -\pi) + \theta(\gamma^*, -\rho^*)$. If the optimal objective value of RPP is positive, that is, $Z^*_{RP}(f, s, w) > 0$, then moving along the direction $(\gamma^*, -\rho^*)$ will increase the dual objective value since $Z_D(\pi', \sigma')$

$= Z_D(\pi, \sigma) + \theta Z_{RD}(\rho^*, \gamma^*) > Z_D(\pi, \sigma)$ where $Z_{RD}(\rho^*, \gamma^*) = Z^*_{RP}(f, s, w) > 0$.

Furthermore, for each commodity k and path $p \in Q^k$, the associated dual constraint $\sum_{a \in A} B^k \delta^p_a(-\pi'_a) + \sigma'_k \leq PC^c_p + \theta(\sum_{a \in A} B^k \delta^p_a(-\rho^*_a) + \gamma^*_k) \leq PC^c_p$ always holds for any positive θ. Similarly, for each arc $a \in A^0$, the associated dual constraint $\pi' = \pi + \theta\rho^* \geq 0$ holds for any positive θ. That is, if we move the current dual solution $(\sigma, -\pi)$ to $(\sigma', -\pi')$ along the direction of $(\gamma^*, -\rho^*)$, then for any positive step length θ the dual objective is improving and columns of $\{Q \cup A^0\}$ will have nonnegative reduced costs in the next iteration.

On the other hand, for column $p \in P^k \backslash Q^k$ or arc $a \in A^+$, if we choose a large step length θ, the associated dual constraint $PC^c_p + \theta(\sum_{a \in A} B^k \delta^p_a(-\rho^*_a) + \gamma^*_k) \leq PC^c_p$ or $\pi' = \pi + \theta\rho^* \geq 0$ may not hold since $\sum_{a \in A} B^k \delta^p_a(-\rho^*_a) + \gamma^*_k > 0$ may be true for $p \in P^k \setminus Q^k$, and $\rho^* < 0$ may be true for $a \in A^+$.

Thus, $(\gamma^*, -\rho^*)$ is a feasible direction, but the step length θ needs to be smaller than a specific threshold to maintain dual feasibility. If such a threshold does not exist, that is, if we can increase θ indefinitely while dual feasibility is always preserved, then the dual is unbounded and the primal is infeasible.

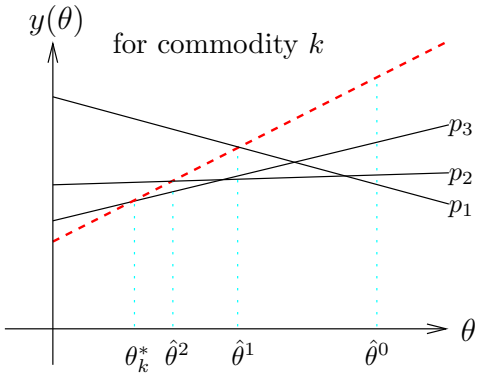
If there exists a finite optimal solution, then there must exist a finite threshold θ^* representing the maximum step length such that $(\sigma, -\pi) + \theta^*(\gamma^*, -\rho^*)$ is dual feasible but $(\sigma, -\pi) + \theta(\gamma^*, -\rho^*)$ is not, for any $\theta > \theta^*$. In particular, let $\theta_1 = \min_{k \in K}\{\theta_k := \min_{p \in P^k \backslash Q^k \text{ and } PC^{\rho^*}_p < \gamma^*_k} \frac{PC^{c+\pi}_p - \sigma_k}{\gamma^*_k - PC^{\rho^*}_p}\}$, and $\theta_2 = \min_{a \in A^+ \text{ and } \rho^*_a < 0}\{\theta_a := \frac{\pi_a}{-\rho^*_a}\}$. Then, $\theta^* := \min\{\theta_1, \theta_2\}$. That is, θ_1 and θ_2 are the threshold for columns in $P^k \setminus Q^k$ and A^+ respectively, and the maximum step length is the smaller of these two values.

6.1.3 Computing the step length θ^*

The step length $\theta^* := \min\{\theta_1, \theta_2\}$. θ_2 can be easily computed by scanning each arc a in A^+ to find the one satisfying $\rho^*_a < 0$ with the smallest value of $\frac{\pi_a}{-\rho^*_a}$. Computing θ_1 requires more consideration since it is not clear how to choose the path $p \in P^k \setminus Q^k$ satisfying $PC^{\rho^*}_p < \gamma^*_k$ with the smallest value of $\frac{PC^{c+\pi}_p - \sigma_k}{\gamma^*_k - PC^{\rho^*}_p}$.

Let $sp(k)$ denote the shortest path for commodity k. If θ is large enough that $PC^{c+\pi+\theta\rho^*}_{sp(k)} < \sigma_k + \theta\gamma^*_k$, then the dual vector $(\sigma', -\pi') = (\sigma, -\pi) + \theta(\gamma^*, -\rho^*)$ is infeasible for step length

θ. The relationship between step length θ and dual feasibility can be observed from Figure 11.



dased line: $y_k(\theta) = \sigma_k + \theta\gamma_k^*$

solid line: $y_p(\theta) = PC_p^{c+\pi} + \theta PC_p^{\rho^*}$ for path p

Figure 11: Illustraion on how to obtain θ_k^* for commodity k

In particular, we can draw a dashed line $y_k(\theta) = \sigma_k + \theta\gamma_k^*$ for commodity k, and draw a solid line $y_p(\theta) = PC_p^{c+\pi} + \theta PC_p^{\rho^*}$ for each path p (see Figure 11). Then, θ^1 makes $(\sigma', -\pi')$ $= (\sigma, -\pi) + \theta^1(\gamma^*, -\rho^*)$ infeasible if and only if there exists some path p and commodity k such that $y_p\left(\theta^1\right) = PC_{sp(k)}^{c+\pi+\theta^1\rho^*} < \sigma_k + \theta^1\gamma_k^* = y_k\left(\theta^1\right)$. In other words, to test whether a step length θ makes the dual solution infeasible, we simply compute the shortest path $sp(k)$ for each commodity k, and then compare the value of $y_{sp(k)}(\theta)$ with $y_k(\theta)$ for each commodity k. If there exists some commodity k such that $y_{sp(k)}(\theta) < y_k(\theta)$, then we know θ is still too large.

By setting $y_k(\theta) = y_p(\theta)$, we can solve for $\hat{\theta}_p$ so that $\sigma_k + \hat{\theta}_p\gamma_k^* = PC_p^{c+\pi} + \hat{\theta}_p PC_p^{\rho^*}$, and $\theta_1 = \min_p \hat{\theta}_p$. Since the number of paths connecting an OD pair k may be large, how to efficiently compute θ_1 is itself a challenging problem.

Polak [268] gives a Floyd-Warshall like algorithm to solve this problem, but its implementation is not clear. Barnhart and Sheffi [32, 38] use a shortest path procedure employing a multiple labeling scheme which first determines all possible value of $PC_p^{\rho^*}$ for each path $p \in P^k$ and for each OD pair k. Then, of all the possible paths with a specific value of $PC_p^{\rho^*}$, they determine the shortest one using arc length $c + \pi$. However, it is also not clear how to efficiently implement this multiple labeling scheme.

Here we propose a binary search method and an iterative method to compute θ^*.

6.1.3.1 Binary search method for finding θ^*

The idea of our binary search method is to identify the range $[\theta_l, \theta_u]$ for θ_1 such that $y_{sp(k)}(\theta_l) > y_k(\theta_l)$ for each commodity k, and $y_{sp(k)}(\theta_u) < y_k(\theta_u)$ for some commodity k. Then we iteratively shrink this range until its width is smaller than a threshold ϵ (the machine precision number, for example).

Since $\theta^* := \min\{\theta_1, \theta_2\}$, we can first identify θ_2, then use it to search for θ_1. In particular, let $sp(k)$ denote the shortest path for commodity k using $c + \pi'$ as the arc lengths. If $y_{sp(k)}(\theta_2) > y_k(\theta_2)$ for each commodity k, then it means $\theta_2 < \theta_1$ and thus $\theta^* := \theta_2$. Otherwise, we can use θ_2 as θ_u to start the binary search for $\theta^* := \theta_1$.

Let $K = \{(s_i, t_i)\}$ be the initial set of requested OD pairs. Procedure $bins\_theta(K, \pi, \sigma, \rho^*, \gamma^*)$ first finds for $\theta' = \theta_2 = \min_{a \in A^+ \text{ and } \rho_a^* < 0} \{\theta_a := \frac{\pi_a}{-\rho_a^*}\}$ if it exists. Furthermore, if $\theta' = \theta_2$ makes $(\sigma', -\pi') = (\sigma, -\pi) + \theta'(\gamma^*, -\rho^*)$ feasible, then $\theta^* = \theta_2$ has been identified.

If there exists no arc $a \in A^+$ such that $\rho_a^* < 0$, then we use any positive θ' to start the binary search. In particular, if θ' makes $(\sigma', -\pi')$ feasible (i.e., $y_{sp(k)}(\theta') > y_k(\theta')$ for each k), then we set $\theta_l = \theta'$, $\theta_u = 2\theta'$ and then test whether θ_u makes $(\sigma', -\pi')$ infeasible. If so, then we find a range $[\theta_l, \theta_u]$ for θ^* where $\theta_u = 2\theta_l$. Otherwise, we keep doubling θ_l and θ_u until θ_u makes $(\sigma', -\pi')$ infeasible.

If, on the other hand, the initial θ' makes $(\sigma', -\pi')$ infeasible, then we set $\theta_l = \frac{1}{2}\theta'$, $\theta_u = \theta'$ and then test whether θ_l makes $(\sigma', -\pi')$ feasible or not. Using a similar procedure as in the previous paragraph but searching along the opposite direction, a range $[\theta_l, \theta_u = 2\theta_l]$ contains θ^* can be identified.

After we obtain the range $[\theta_l, \theta_u]$ for θ^*, we can set $\theta' = \frac{\theta_l + \theta_u}{2}$ and check whether θ' can be used as a new upper bound θ_u (or a new lower bound θ_l) so that the range $[\theta_l, \theta_u]$ is cut in half. We iterate until the width is smaller than a threshold ϵ.

To guarantee that θ^* is sufficiently accurate, we may set the threshold ϵ to be the machine precision number, which is usually 10^{-9} or even smaller. However, such a small threshold value may require many iterations of binary search and makes the procedure inefficient. Next we propose an iterative method, which we will use for our implementation.

Procedure bins_theta($\mathbf{K} := \{(\mathbf{s_i}, \mathbf{t_i})\}, \pi, \sigma, \rho^*, \gamma^*$)
begin
$[\theta_l, \theta_u, \overline{OD}] = FindRange(K, \pi, \sigma, \rho^*, \gamma^*)$;
Let $\theta' = \theta_u$
while $\theta_u > \theta_l + \epsilon$
Let $\theta' = \frac{\theta_l + \theta_u}{2}$,
$\overline{OD} = UpdateOD(\overline{OD}, \pi, \sigma, \theta')$;
if $\overline{OD} \neq \emptyset$ ***then*** $\theta_u := \theta'$
else $\theta_l := \theta'$
return $\theta^* = \theta'$
end

Subprocedure $FindRange(K, \pi, \sigma, \rho^*, \gamma^*)$
begin
Set an initial $\theta_0 > 0$, let $\hat{A}^+ = \{a : a \in A^+ \text{ and } \rho_a^* < 0\}$ and $\overline{OD} = K$
if $\hat{A}^+ \neq \emptyset$ ***then*** $\theta' := \min_{a \in \hat{A}^+} \{\theta_a := \frac{\pi_a}{-\rho_a^*}\}$
else $\theta' = \theta_0$
Initialize $\theta_l = \theta_u = \theta'$
$\overline{OD} = UpdateOD(\overline{OD}, \pi, \sigma, \theta')$;
if $\overline{OD} \neq \emptyset$ ***then*** $\zeta_1 = 1$,
else if $\hat{A}^+ \neq \emptyset$
return $[\theta_l, \theta_u, \overline{OD}]$
if $\zeta_1 = 1$ ***then*** $\zeta_2 = 1$ and $[\omega_l, \omega', \omega_u] := [\frac{1}{2}, 1, \frac{1}{2}]$
else $\zeta_2 = 0$ and $[\omega_l, \omega', \omega_u] := [1, 2, 2]$
while $\zeta_1 = \zeta_2$
if $\zeta_2 = 0$ ***then*** reset $\overline{OD} = K$
$\overline{OD} = UpdateOD(\overline{OD}, \pi, \sigma, \theta')$;
if $\overline{OD} \neq \emptyset$ ***then*** $\zeta_2 = 1$
else $\zeta_2 = 0$
if $\zeta_1 = \zeta_2$ ***then*** $[\theta_l, \theta', \theta_u] := [\theta_l \times \omega_l, \theta' \times \omega', \theta_u \times \omega_u]$
return $[\theta_l, \theta_u, \overline{OD}]$
end

Subprocedure $UpdateOD(\overline{OD}, \pi, \sigma, \theta')$
begin
Compute $(\pi', \sigma') := (\pi, \sigma) + \theta'(\rho^*, \gamma^*)$
Using $c + \pi'$ as arc lengths, use a MPSP algorithm to compuate $sp(k)$ $\forall k \in \overline{OD}$
Remove those commodities k satisyfing $PC_{sp(k)}^{c+\pi'} \geq \sigma_k'$ from the set $\overline{OD}$
return $\overline{OD}$
end

6.1.3.2 Iterative method for finding θ^*

From Figure 11, we know that for each commodity k, θ_k^* can be determined by the intersection of $y_k(\theta) = \sigma_k + \theta_p \gamma_k^*$ and $y_p(\theta) = PC_p^{c+\pi} + \theta_p PC_p^{\rho^*}$ for some path p from origin s_k to destination t_k. The challenge will be to efficiently determine the right path p for each commodity k.

Procedure iter_theta($\mathbf{K} := \{(\mathbf{s_i}, \mathbf{t_i})\}, \pi, \sigma, \rho^*, \gamma^*$)
begin
  $[\theta_l, \theta_u, \overline{OD}] = FindRange(K, \pi, \sigma, \rho^*, \gamma^*)$;
  if $\theta_l = \theta_u$
    return $\theta^* = \theta_u$
  Set $\hat{\theta}^0 := \theta_u$, $t = 0$
  if $\overline{OD} = \emptyset$ ***then*** reset $\overline{OD} := K$
  while $\overline{OD} \neq \emptyset$
    Compute $(\pi', \sigma') := (\pi, \sigma) + \theta'(\rho^*, \gamma^*)$, $t := t + 1$
    Using $c + \pi'$ as arc lengths, solve the MPSP problem to find $sp(k)$ $\forall k \in \overline{OD}$
    Trace path $sp(k)$ to compute $PC_{sp(k)}^{\rho^*}$ for each commodity $k \in \overline{OD}$
    Compute $\hat{\theta}^t = \min_{k \in \overline{OD}} \frac{PC_{sp(k)}^{c+\pi'} - \sigma_k}{\gamma_k^* - PC_{sp(k)}^{\rho^*}}$
    $\overline{OD} = UpdateOD(\overline{OD}, \pi, \sigma, \hat{\theta}^t)$;
  return $\theta^* = \hat{\theta}^t$
end

Our iterative method uses an initial $\hat{\theta}^0$ that produces an infeasible dual solution $(\sigma', -\pi')$ $= (\sigma, -\pi) + \hat{\theta}^0(\gamma^*, -\rho^*)$, and then solves the MPSP problem to find $sp(k)$ and $PC_{sp(k)}^{c+\pi'}$ for each commodity $k \in \overline{OD}$. Using the shortest path $sp(k)$ for commodity k, we can also compute $PC_{sp(k)}^{\rho}$ and determine a better $\hat{\theta}^1 = \frac{PC_{sp(k)}^{c+\pi'} - \sigma_k}{\gamma_k^* - PC_{sp(k)}^{\rho^*}}$ by finding the intersection of the line $y_k(\theta)$ and the line $y_{sp(k)}(\theta)$.

Using the new $\hat{\theta}^1$, we can update $\overline{OD}$, the set of commodities that still satisfies $y_k(\hat{\theta}^1) > y_{sp(k)}(\hat{\theta}^1)$. Let $\hat{\theta}^t$ denote the step length in the t^{th} iteration computed by this iterative method. It is easy to observe that $\hat{\theta}^1 < \hat{\theta}^0$, and that $\hat{\theta}^{t+1} < \hat{\theta}^t$. Thus, the sequence $\hat{\theta}^t$ is decreasing. Furthermore, if $y_{k'}(\hat{\theta}^t) < y_{sp(k')}(\hat{\theta}^t)$ for commodity k', then $y_{k'}(\hat{\theta}^{t+1}) < y_{sp(k')}(\hat{\theta}^{t+1})$. These properties can be easily verified from Figure 11. Thus, our iterative method iteratively shrinks $\overline{OD}$ and $\hat{\theta}^t$ until finally $\overline{OD}$ becomes empty.

We use Figure 11 to illustrate the procedure. Suppose commodity k has only three possible OD paths: p_1, p_2, and p_3. We start with an initial $\hat{\theta}^0$ that satisfies $y_k(\hat{\theta}^0) >$

$y_{sp(k)}(\hat{\theta}^0)$ for commodity k. Then, using $c + \pi + \hat{\theta}^0\rho^*$ as the arc lengths, we identify the shortest path $sp(k) = p_1$. We compute the intersection of $y_k(\theta)$ and $y_{sp(k)}(\theta)$ to determine a new step length $\hat{\theta}^1$. Using $c + \pi + \hat{\theta}^1\rho^*$ as the arc lengths, we identify the shortest path $sp(k) = p_2$ that still satisfies $y_k(\hat{\theta}^1) > y_{sp(k)}(\hat{\theta}^1)$. We compute the intersection of $y_k(\theta)$ and $y_{sp(k)}(\theta)$ to determine a new step length $\hat{\theta}^2$. From Figure 11, we find $y_k(\hat{\theta}^2) > y_{p_3}(\hat{\theta}^2)$ where path p_3 is the shortest path for commodity k using $c + \pi + \hat{\theta}^2\rho^*$ as the arc lengths. After computing $\hat{\theta}^3 = \frac{PC_{p_3}^{c+\pi'} - \sigma_k}{\gamma_k^* - PC_{p_3}^{\rho^*}}$ we find that $y_k(\hat{\theta}^3) = y_{sp(k)}(\hat{\theta}^3)$. Therefore, we have determined the step length $\theta_k^* = \hat{\theta}^3$ for commodity k.

In the worst case, this iterative method may have $\sum_k \left|P^k\right|$ iterations of MPSP computations, where P^k is the set of all possible paths for OD pair k. However, in practice, it requires only few iterations to converge to the exact θ^* since the size of $\overline{OD}$ is nonincreasing. We observe that it is faster than the binary search method because the binary search method is doing blind search and converges slowly near the end of the procedure. For the primal-dual method, we need to compute θ^* exactly so that the dual objective is strictly ascending. Numerical precision problems in the binary search method create difficulties in making improvements due to zero step length.

Both of our proposed methods for computing θ^* require many iterations of MPSP computation. Thus, an efficient MPSP algorithm will greatly improve the practical running time. There may exist other methods more efficient than our iterative method for computing the step length θ^*. We think this is an interesting topic for future research.

6.1.4 Degeneracy issues in solving the RPP

At each iteration, after identifying the step length θ^* a new, improved feasible dual solution $(\sigma', -\pi') = (\sigma, -\pi) + \theta^*(\gamma^*, -\rho^*)$ can be computed and used to construct a new RPP with only the zero-reduced-cost columns.

The RPP can be solved by any LP method. In our implementation, we use CPLEX 7.0 to solve the RPP. CPLEX has its own techniques to resolve degeneracy. However, our experiments show that sometimes degeneracy between different iterations of RPP may slow down the primal-dual methods even when CPLEX's techniques are used.

Suppose $Z^*_{RP_t}$ is the optimal objective for the t^{th} RPP (the RPP in the t^{th} primal-dual iteration). Since the primal infeasibility stays strictly positive until the last primal-dual iteration, the original dual objective will be strictly improving at each iteration. The total primal infeasibility $Z^*_{RP_t}$ should remain nonincreasing. That is, $Z^*_{RP_{t+1}} \leq Z^*_{RP_t}$. Equality holds whenever a primal degenerate pivot occurs between iteration t and iteration $t+1$.

Suppose we add columns to the RPP to construct a larger problem PP_A. That is, problem PP_A contains three sets of columns: (1) artificial columns with unit objective coefficients, (2) all shortest paths with zero objective coefficients for each commodity, and (3) all other paths which are not the shortest, but also have zero objective coefficients for each commodity. The original RPP can be thought as a "restricted master" problem of PP_A that uses only the first two groups of columns. A feasible basis and a basic feasible solution (b.f.s.) for RPP will also be a feasible basis and b.f.s. for PP_A. Therefore, when $Z^*_{RP_{t+1}} = Z^*_{RP_t}$, moving from iteration t to iteration $t+1$ in the primal-dual method can be thought as a degenerate pivot for the larger problem PP_A.

CPLEX does not avoid such degenerate pivots between primal-dual iterations since the new RPP is constructed from scratch and CPLEX solves the new RPP from scratch as well. Whatever degeneracy-resolving techniques CPLEX applies for solving the previous RPP do not carry over when solving the new RPP. In our experiments, we do experience such degeneracy. To illustrate the extent of the affect of degeneracy, we note a case in which the primal infeasibility remains unchanged for approximately 1000 iterations, causing the method to take several hours to terminate.

To avoid these degenerate pivots, we perturb the objective coefficients of the artificial variables. In particular, instead of using 1 as the objective coefficient for all artificial variables (as suggested in most textbooks), we assign random integers to be the new objective coefficients for artificial variables. This change will not affect the validity of the primal-dual method. The optimal dual solution for the new RPP will be different from the original one, but will still be a valid dual improving direction. All the other operations such as computing the step length θ^* and constructing the new RPP will still work. The algorithm will terminate since each primal-dual iteration strictly improves the primal infeasibility and

the dual objective value, and dual feasibility and complementary slackness conditions are still maintained throughout the whole procedure.

In our implementation, we check whether the primal infeasibility decreases between primal-dual iterations. If it remains unchanged for two iterations, then we perturb the objective coefficients of artificial variables once. This change does resolve the degenerate pivots and shortens the computational time. For example, the previous case, which requires several hours to perform 1000 degenerate pivots, solves in several seconds.

In next section, we describe a new method for solving the RPP which may be efficient for cases with many commodities.

6.2 Primal-dual key path method

As introduced in Section 2.3.3, the key variable decomposition method, first proposed by Rosen [273], has been used by Barnhart et al. [36] in conjunction with column generation with success.

Barnhart et al. [36] use key variable decomposition with primal simplex column generation. In this section, we apply a similar technique to solve the RPP from the primal-dual method. In particular, the primal-dual key path method follows the generic primal-dual steps (see Figure 10), except that it uses the key variable decomposition method to solve the RPP. Thus in this section we focus on solving the RPP by key variable decomposition.

6.2.1 Cycle and Relax(i) RPP formulations

Given a feasible dual solution $(\sigma, -\pi)$, we can construct P_RPP, the path formulation of the RPP (see page 156). For each commodity k, we choose a shortest path $key(k) \in Q^k$, where $Q^k := \{p : PC_p^{c+\pi} = \sigma_k,\ p \in P^k\}$. After performing column operations to eliminate

the key columns, we obtain the following key cycle formulation of the RPP:

$$\min \sum_{a \in A} (w_a^+ + w_a^-) = Z^*_{CRP}(f, s, w) \qquad \text{(C\_RPP)}$$

$$s.t. \quad \sum_{p \in Q^k} f_p = 1 \quad \forall k \in K \qquad (6.5)$$

$$\sum_{k \in K} \sum_{p \in Q^k} B^k(\delta_a^p - \delta_a^{key(k)}) f_p + s_a + (w_a^+ - w_a^-) = u_a - \sum_{k \in K} B^k \delta_a^{key(k,i)} \quad \forall a \in A^0 \qquad (6.6)$$

$$\sum_{k \in K} \sum_{p \in Q^k} B^k(\delta_a^p - \delta_a^{key(k)}) f_p \qquad + (w_a^+ - w_a^-) = u_a - \sum_{k \in K} B^k \delta_a^{key(k,i)} \quad \forall a \in A^+ \qquad (6.7)$$

$$f_p \geq 0 \quad \forall p \in Q^k, \forall k \in K$$

$$s_a \geq 0 \quad \forall a \in A^0 \; ; \; w_a^+ \geq 0, \; w_a^- \geq 0 \quad \forall a \in A$$

Note that C_RPP has the same objective function as P_RPP since all the shortest path columns have zero objective coefficients. C_RPP should be no easier or harder than P_RPP since the transformation does not change the problem structure.

When the number of commodities (OD pairs) is huge, both C_RPP and P_RPP will have many constraints, which makes the RPP more difficult. The key variable transformation relaxes the nonnegativity constraints for each key path (i.e., it allows $f_{key(k)}$ to be negative) and results in an easier problem R_RPP(i), where i denotes the index of iteration:

$$\min \sum_{a \in A} (w_a^{i+} + w_a^{i-}) = Z^*_{RRP(i)}(f, s, w) \qquad \text{(R\_RPP}(i)\text{)}$$

$$s.t. \quad \sum_{k \in K} \sum_{p \in Q^k} B^k(\delta_a^p - \delta_a^{key(k,i)}) f_p^i + s_a^i + (w_a^{i+} - w_a^{i-}) = u_a - \sum_{k \in K} B^k \delta_a^{key(k,i)} \quad \forall a \in A^0 \qquad (6.8)$$

$$\sum_{k \in K} \sum_{p \in Q^k} B^k(\delta_a^p - \delta_a^{key(k,i)}) f_p^i \qquad + (w_a^{i+} - w_a^{i-}) = u_a - \sum_{k \in K} B^k \delta_a^{key(k,i)} \quad \forall a \in A^+ \qquad (6.9)$$

$$f_p^i \geq 0 \quad \forall p \in Q^k \setminus f^i_{key(k,i)}, \forall k \in K; \; f^i_{key(k,i)} \text{:free} \quad \forall k \in K$$

$$s_a^i \geq 0 \quad \forall a \in A^0 \; ; \; w_a^{i+} \geq 0, \; w_a^{i-} \geq 0 \quad \forall a \in A$$

whose dual is

$$\max \sum_{a \in A} (u_a - \sum_{k \in K} B^k \delta_a^{key(k,i)})(-\rho_a^i) = Z^*_{RRD(i)}(\rho) \qquad \text{(R\_RDP}(i)\text{)}$$

$$s.t. \quad \sum_{a \in A} B^k (\delta_a^p - \delta_a^{key(k,i)})(-\rho_a^i) \leq 0 \quad \forall p \in Q^k, \forall k \in K \qquad (6.10)$$

$$0 \leq \rho_a^i \leq 1 \quad \forall a \in A^0$$

$$-1 \leq \rho_a^i \leq 1 \quad \forall a \in A^+$$

6.2.2 Key variable decomposition method for solving the RPP

The RPP can be solved by iteratively application of the key variable decomposition method as illustrated in Figure 12. In each iteration of the key variable decomposition method, an easier problem, R_RPP(i), is solved to optimality.

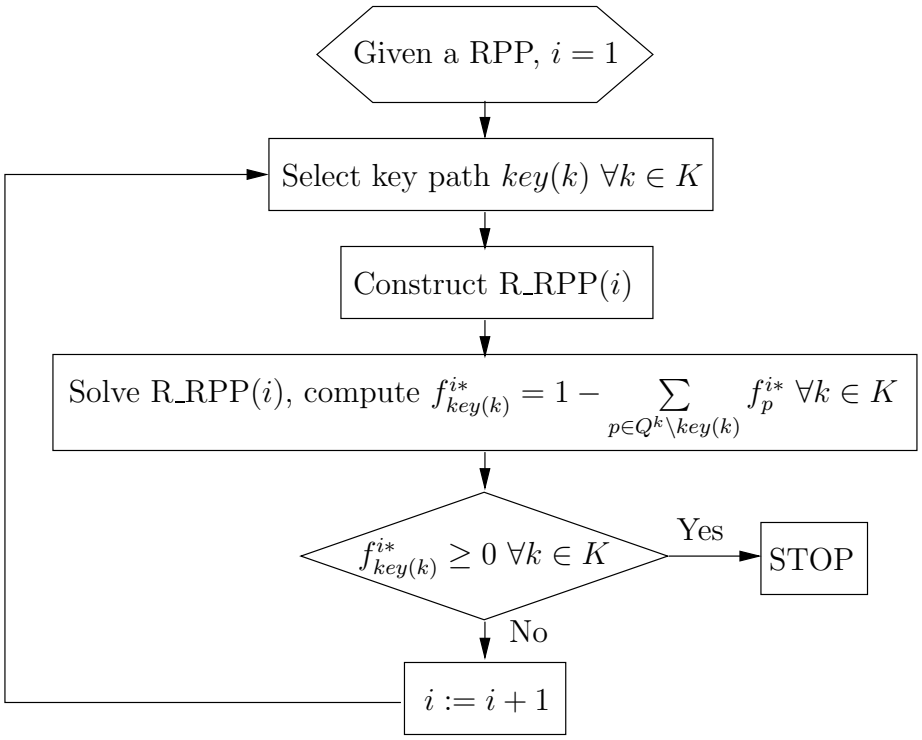


Figure 12: Key variable decomposition method for solving the RPP

After solving R_RPP(i), the algorithm will check the sign of key variables by calculating $f^{i*}_{key(k,i)} = 1 - \sum_{p \in Q^k \setminus key(k,i)} f_p^{i*}$. For those key variables with negative signs, the algorithm will perform a *key variable change* operation to replace the current key variable with a

new one. Among all positive shortest path variables eligible for selection, the one with the largest value is usually chosen; intuitively, that path is more likely to have positive flow in the optimal solution. That is, $key(k, i+1) = \arg\max_{q \in Q^k} f_q^{i*}$.

The proof of finiteness and optimality for the key variable decomposition method can be found in Barnhart et al. [36]. In particular, after solving the R_RPP(i), if there exists some key variable $f_{key(k,i)}^{i*} < 0$, there must also exist a shortest path $q(k)$ such that $f_{q(k)}^{i*} > 0$, and column $q(k)$ is in the optimal basis. The optimal dual solution of RPP(i), $-\rho^{i*}$, remains as a basic feasible solution for R_RDP(i+1). The swap of key path from $key(k, i)$ to $q(k)$ does not affect using $-\rho^{i*}$ as an initial basic feasible solution for R_RDP(i+1). Furthermore, $Z_{RRD(i)}^*$ $= Z_{RRD(i+1)}$, and the complementary slackness conditions are maintained after the swap of the key path. Since $f_{key(k,i)}^{i+1} < 0$, a dual simplex pivot has to be performed to achieve optimality for R_RPP($i+1$). Using the dual simplex method with degeneracy resolving techniques, the dual objective is strictly improved in next iteration, and the algorithm will terminate in a finite number of iterations. If the key variable decomposition method takes $\hat{i}$ iterations to solve the RPP, then $Z_{RRP(\hat{i})}^* = Z_{RP}^*$.

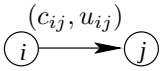
After the RPP is solved, we use the same procedure as introduced in Section 6.1.3 to identify the step length θ^*. Then, we can update $\pi' = \pi + \theta^* \rho^*$, compute $\sigma_k^* = PC_{sp(k)}^{c+\pi'}$ where $sp(k)$ is the shortest path of commodity k using $c + \pi'$ as the arc lengths, and finally construct a new RPP for next primal-dual iteration.

This algorithm maintains dual feasibility and complementary slackness conditions while trying to achieve primal feasibility (which will be attained when all key variables become nonnegative).

6.2.3 Degeneracy issues between key path swapping iterations

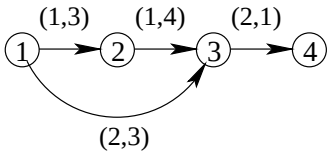
The finiteness of the key variable decomposition method for solving the relaxed problem R_RPP(i) relies on the assumption of techniques to resolve degeneracy. Degeneracy is not an issue when we solve the R_RPP(i), but it will become a crucial factor between iterations of solving the R_RPP(i).

Suppose we are to solve the ODMCNF problem shown in Figure 13, which requires us



c_{ij}: cost of arc (i,j)

u_{ij}: capacity of arc (i,j)



K^1: $(s^1, t^1) = (1, 4)$, $B^1 = 2$

initial $\pi_a = 0\ \forall a \in A$

arc (i,j)	index
$(1,3)$	1
$(1,3)$	2
$(2,3)$	3
$(3,4)$	4

Q^1 : $p_1 : 1 \to 2 \to 3 \to 4$
$p_2 : 1 \to 3 \to 4$

$$min \sum_{a=1}^{4} w_a^+ + \sum_{a=1}^{4} w_a^- \qquad \text{(RPP)}$$

subject to

$$\begin{aligned}
2f_{p_1} \qquad\qquad + s_1 + w_1^+ - w_1^- &= 3 \\
2f_{p_2} + s_2 + w_2^+ - w_2^- &= 3 \\
2f_{p_1} \qquad\qquad + s_3 + w_3^+ - w_3^- &= 4 \\
2f_{p_1} + 2f_{p_2} + s_4 + w_4^+ - w_4^- &= 1 \\
f_{p_1} + f_{p_2} &= 1
\end{aligned}$$

$f_p \geq 0\ \forall p \in Q^1$; $s_a \geq 0, w_a^+, w_a^-$: free $\forall a \in A$

Optimal solution:

$w_4^{-*} = 1$

$w_a^{-*} = 0,\ a = 1, 2, 3$

$w_a^{+*} = 0,\ a = 1, 2, 3, 4$

both $f_{p_1}^*, f_{p_2}^* \in [0, 1.5]$, but $f_{p_1}^* + f_{p_2}^* = 1$

Figure 13: A small ODMCNF example and its RPP formulation

to send 2 units of flow from node 1 to node 4. Using $-\pi = 0$ as an initial dual feasible solution, the restricted network consists of 4 arcs, $(1,2)$, $(1,3)$, $(2,3)$, and $(3,4)$, because there are only 2 paths ($1 \to 2 \to 3 \to 4$ and $1 \to 3 \to 4$) that have zero reduced cost. Note that there should exist other paths from node 1 to node 4 using other arcs; otherwise the problem is primal infeasible. It is easy to see that the optimal solution of this RPP is $w_4^{+*} = 1$, $w_a^{+*} = 0$ for $a = 1, 2, 3$; $w_a^{-*} = 0$ for $a = 1, 2, 3, 4$; and $f_{p_1}^{1*} \in [0, 1.5]$, $f_{p2}^{1*} \in [0, 1.5]$, with $f_{p_1}^{1*} + f_{p_2}^{1*} = 1$. That is, there exist infinitely many $f_{p_1}^{1*}$ and $f_{p_2}^{1*}$. The exact value of $f_{p_1}^{1*}$ and $f_{p_2}^{1*}$ will not affect the objective function.

Figure 14 illustrates the steps of the key variable decomposition method for solving this example. Suppose we choose p_1 to be the key path in the first iteration when solving R_RPP(1). There exist multiple optimal primal solutions to R_RPP(1); that is, $0 \leq f_{p_2}^{1*} \leq 1.5$. If we specify $f_{p_2}^{1*}$ to be any value in $[0, 1]$, then $f_{p_1}^{1*} = 1 - f_{p_2}^{1*}$ will be feasible. However, if we choose $1 < f_{p_2}^{1*} \leq 1.5$, then $-0.5 \leq f_{p_1}^{1*} < 0$ is not primal feasible for RPP, and we

Q^1 : p_1 : $1 \to 2 \to 3 \to 4$: $key(1,1)$
p_2 : $1 \to 3 \to 4$

$$min \sum_{a=1}^{4} w_a^+ + \sum_{a=1}^{4} w_a^- \qquad \text{(R\_RPP(1))}$$

subject to

$$-2f_{p_2}^1 + s_1 + w_1^+ - w_1^- = 1$$
$$2f_{p_2}^1 + s_2 + w_2^+ - w_2^- = 3$$
$$-2f_{p_2}^1 + s_3 + w_3^+ - w_3^- = 2$$
$$s_4 + w_4^+ - w_4^- = 1$$

$f_{p_1}^1$:free, $f_{p_2}^1 \geq 0$; s_a, w_a^+, w_a^-: free $\forall a \in A$

Optimal solution:

$w_4^{-*} = 1$, $f_{p_2}^{1*} \in [0, 1.5]$
$w_a^{-*} = 0$, $a = 1, 2, 3$
$w_a^{+*} = 0$, $a = 1, 2, 3, 4$
$f_{p_1}^{1*} \in [-0.5, 1]$

CPLEX gives $f_{p_2}^{1*} = 1.5$, thus $f_{p_1}^{1*} = -0.5 < 0$

Therefore, we have to swap $key(1,2) = p_2$

Q^1 : p_1 : $1 \to 2 \to 3 \to 4$
p_2 : $1 \to 3 \to 4$: $key(1,2)$

$$min \sum_{a=1}^{4} w_a^+ + \sum_{a=1}^{4} w_a^- \qquad \text{(R\_RPP(2))}$$

subject to

$$2f_{p_1}^2 + s_2 + w_1^+ - w_1^- = 3$$
$$-2f_{p_1}^2 + s_2 + w_2^+ - w_2^- = 1$$
$$2f_{p_1}^2 + s_3 + w_3^+ - w_3^- = 4$$
$$s_4 + w_4^+ - w_4^- = -1$$

$f_{p_2}^2$:free, $f_{p_1}^2 \geq 0$; s_a, w_a^+, w_a^-: free $\forall a \in A$

Optimal solution:

$w_4^{-*} = 1$, $f_{p_1}^{2*} \in [0, 1.5]$
$w_a^{-*} = 0$, $a = 1, 2, 3$
$w_a^{+*} = 0$, $a = 1, 2, 3, 4$
$f_{p_2}^{2*} \in [-0.5, 1]$

CPLEX gives $f_{p_1}^{2*} = 0$, thus $f_{p_2}^{2*} = 1 > 0$, STOP

If the solver gives $f_{p_1}^{2*} = 1.5$, then $f_{p_1}^{2*} = -0.5 < 0$,
Therefore, we have to swap $key(1,3) = p_1$ again.

Figure 14: Infinite iterations of key path swapping due to dual degeneracy

have to swap key paths.

Unfortunately, CPLEX (with default settings) will give $f_{p_2}^{1*} = 1.5$, and thus we have to set the key path for the second iteration, $key(1,2)$, to be path p_2 and construct R_RPP(2).

Again, R_RPP(2) has multiple optimal primal solutions $0 \leq f_{p_1}^{2*} \leq 1.5$, and thus the situation is exactly the same as the first iteration. Fortunately, CPLEX with default settings will give us $f_{p_1}^{2*} = 0$. However, there is no guarantee that CPLEX or any other solver will have good luck all the time. If the solver happens to choose $1 < f_{p_1}^{2*} \leq 1.5$ as its optimal solution, then $-0.5 \leq f_{p_2}^{2*} < 0$ is not primal feasible for RPP, and we have to swap the key path back to p_1. For CPLEX, since it will only choose the one that are basic feasible, so $f_{p_2}^{2*} = 0$ or $f_{p_2}^{2*} = 1.5$, a 50% chance to terminate earlier. In other words, since the LP

solver does not distinguish between multiple optimal solutions, it is possible that a "bad" optimal solution will be chosen, resulting in infinite loops of key path swapping. Indeed, in our computational experiments, this happens very frequently because the RPP is very dual degenerate.

Using 1 as an upperbound for each path flow f_p^i may avoid some cycling, since such upperbound forces each non-key-path flow less than 1 which in turn makes the sum of their flows smaller, and thus more likely to make the flow on key path nonnegative. However, it comes with the difficulty that we will have extra dual variables for these upper bound constraints. Bookkeeping for these extra dual variables, one for each path, may be difficult since there may be many of them and for each we require a record of its associated path. Furthermore, it is still possible that each $f_p^i < 1$ but their sum exceeds 1. Thus, cycling may still occur.

Because the objective function in RPP and R_RPP(i) contains only artificial variables with no other restriction on the path flows, the LP solver may give the optimal path flow "blindly" even if there does exist a better optimal flow solution. Here, we propose a perturbation method that assigns a very small random positive objective coefficient for all path variables. This idea is inspired by observing the example in Figure 14. In particular, after we relax the nonnegativity constraints for the key variables, we will prefer the optimal path flow solution to be as small as possible so that the convexity constraints will be more likely to make the key variable positive.

Therefore, instead of assigning a zero objective coefficient for each path flows, we give a very small positive random cost coefficient for each path flow. Thus, the optimal flow solution will tend to be smaller and key path flows will be more likely to be nonnegative. Note that such a change will affect dual feasibility of the R_RDP(i), the dual of the R_RPP(i). Thus we have to add an extra loop to compensate for the effect of the perturbation.

In particular, the optimal dual solution of R_RPP(i), $-\rho^*$, will be in the set $\{-1, 0, 1\}$. After the perturbation, $-\rho^*$ may become $-1 \pm \epsilon$, $\pm\epsilon$, or $1 \pm \epsilon$, where ϵ is a very small number. Thus we reset each deviated $-\rho_a^*$ to be its nearest integer, -1, 0, or 1, so that the affect of perturbation to $-\rho^*$ is compensated and corrected.

Table 30: Four methods for solving ODMCNF

Algorithm	Problem formulation	Solution method
PD	arc-path P_RPP formulation (see page 156)	generic primal-dual method
KEY	arc-path R_RPP(i) formulation (see page 166)	primal-dual key path method
DW	arc-path P_PATH formulation (see page 21)	Dantzig-Wolfe decomposition
NA	node-arc formulation (see page 13)	CPLEX

Table 31: Problem characteristics for 17 randomly generated test problems

Problem	$\|N\|$	$\|A\|$	$\|K\|$	$\|N\|\|K\|+\|A\|$	$\|M\|\|K\|$
$\mathbf{P}_1$	200	517	323	65,117	166,991
$\mathbf{P}_2$	200	501	323	65,117	161,823
$\mathbf{P}_3$	200	508	323	65,117	164,084
$\mathbf{P}_4$	200	520	317	63,917	164,840
$\mathbf{P}_5$	300	760	561	168,817	426,360
$\mathbf{P}_6$	300	811	530	159,517	429,830
$\mathbf{P}_7,\ldots,\mathbf{P}_{17}$	49	130	427	21,440	55,510

6.3 Computational results

We implemented four algorithms for solving the multicommodity flow problems. Each algorithm solves a different multicommodity flow formulation (see Table 30). All of our tests were run using CPLEX 7.0 on a Sun Sparc machine with 512MB RAM. We tested 17 problem sets obtained from Hane [165] (see Table 31). $\mathrm{P}_1,\ldots,\mathrm{P}_6$ are directed graphs with different topologies and commodities, and $\mathrm{P}_7,\ldots,\mathrm{P}_{17}$ are undirected graphs with different commodities but the same topology. All of these 17 problems use OD pairs as commodities, but the commodity costs are uniform on each arc. That is, $c_a^k = c_a$, for each arc a in A and each commodity k in K. Our MPSP algorithm $DLU2$ are used for computing shortest paths in MCNF algorithms PD, KEY, and DW.

6.3.1 Algorithmic running time comparison

Table 32 lists the overall running time of each algorithm on each problem, as well as percentage of time spent by different procedures. In particular, t_{all} denotes the total running time. We divide the running time of algorithm PD into three components: t_{sp}, t_θ and $\overline{t}$, where t_{sp} represents the time spent on generating all shortest paths for constructing the RPP, t_θ is the time spent on computing the step length θ^*, and $\overline{t}$ is the remaining time mostly spent

Table 32: Total time (ms) of four algorithms on problems P_7,..., P_{17}

	PD				KEY					DW			NA
	$\mathbf{t}_{sp}$%	$\mathbf{t}_{\theta}$%	$\overline{t}$%	$\mathbf{t}_{all}$	$\mathbf{t}_{sp}$%	$\mathbf{t}_{\theta}$%	$\mathbf{t}_{key}$%	$\overline{t}$%	$\mathbf{t}_{all}$	$\mathbf{t}_{sp}$%	$\overline{t}$%	$\mathbf{t}_{all}$	$\mathbf{t}_{all}$
$\mathbf{P}_1$	54%	28%	18%	**291**	43%	22%	26%	9%	**366**	31%	69%	**57**	**5532**
$\mathbf{P}_2$	39%	46%	15%	**1444**	19%	22%	56%	3%	**2584**	31%	69%	**65**	**5934**
$\mathbf{P}_3$	42%	44%	15%	**987**	24%	25%	48%	4%	**1713**	29%	71%	**58**	**5611**
$\mathbf{P}_4$	42%	42%	16%	**699**	22%	22%	53%	4%	**1334**	27%	73%	**62**	**5479**
$\mathbf{P}_5$	50%	34%	16%	**2276**	12%	9%	78%	1%	**9769**	28%	72%	**124**	**34753**
$\mathbf{P}_6$	44%	46%	10%	**3214**	25%	26%	46%	3%	**5630**	36%	64%	**144**	**30460**
$\mathbf{P}_7$	67%	20%	13%	**65**	70%	22%	3%	5%	**59**	57%	43%	**13**	**363**
$\mathbf{P}_8$	61%	26%	12%	**102**	66%	28%	4%	3%	**97**	51%	49%	**15**	**369**
$\mathbf{P}_9$	59%	29%	11%	**1075**	60%	30%	8%	2%	**1038**	43%	57%	**27**	**592**
$\mathbf{P}_{10}$	59%	29%	12%	**1416**	58%	29%	12%	2%	**1397**	40%	60%	**27**	**812**
$\mathbf{P}_{11}$	61%	28%	12%	**209**	64%	30%	4%	2%	**195**	51%	49%	**18**	**416**
$\mathbf{P}_{12}$	61%	28%	11%	**311**	65%	30%	4%	2%	**292**	52%	48%	**20**	**437**
$\mathbf{P}_{13}$	60%	29%	11%	**384**	64%	30%	4%	2%	**359**	56%	44%	**19**	**426**
$\mathbf{P}_{14}$	60%	29%	11%	**451**	63%	30%	5%	2%	**434**	56%	44%	**22**	**465**
$\mathbf{P}_{15}$	61%	28%	11%	**533**	64%	29%	5%	2%	**517**	52%	48%	**17**	**445**
$\mathbf{P}_{16}$	61%	28%	11%	**697**	63%	29%	7%	2%	**653**	54%	46%	**20**	**504**
$\mathbf{P}_{17}$	60%	29%	11%	**805**	62%	29%	8%	2%	**780**	45%	55%	**23**	**518**

on CPLEX operations. The total running time of algorithm KEY can be divided into four parts: t_{sp}, t_{θ}, t_{key} and $\overline{t}$. Three of them, t_{sp}, t_{θ} and $\overline{t}$, are as described before. t_{key} is the time spent for key path swapping and for CPLEX to solve the relaxed RPP. Algorithm DW spent t_{sp} in computing shortest paths for generating columns with negative reduced cost. Algorithm NA simply uses CPLEX to directly solve the problem in node-arc form.

In all of the tests, DW is the fastest algorithm. For larger cases P_1,...,P_6, NA is the slowest algorithm and usually is many times slower than the other algorithms. PD is the second fast algorithm. KEY usually spends no more than twice the time of PD, with the exception of problem P_5.

For smaller cases P_7,...,P_{17}, KEY performs slightly better than PD. It is difficult to conclude whether KEY is faster than NA or not from our limited tests. However, it is observed that the performance of NA seems to be consistent when the topology and number of commodities are fixed. For example, we may roughly classify the 17 problems into 3 groups: P_1,...,P_4, P_5 and P_6, and P_7,...,P_{17}, by their number of nodes, arcs and commodities. We observe that NA performs similarly for problems in the same group. KEY and PD, on the other hand, seem to be more sensitive to what the commodities are, instead of how many commodities there are, and may perform drastically differently even on the same network with the same number of commodities.

Table 33: Comparisons on number of iterations for PD, KEY and DW

	PD	KEY		DW
	number of iterations	number of iterations	number of key changes	number of iterations
$\mathbf{P}_1$	8	8	100	7
$\mathbf{P}_2$	29	25	1770	7
$\mathbf{P}_3$	21	21	973	6
$\mathbf{P}_4$	15	15	797	7
$\mathbf{P}_5$	21	21	3102	6
$\mathbf{P}_6$	24	24	1788	6
$\mathbf{P}_7$	4	4	0	4
$\mathbf{P}_8$	6	6	4	5
$\mathbf{P}_9$	61	60	142	7
$\mathbf{P}_{10}$	79	77	391	6
$\mathbf{P}_{11}$	12	12	5	5
$\mathbf{P}_{12}$	18	18	5	6
$\mathbf{P}_{13}$	22	22	11	6
$\mathbf{P}_{14}$	26	26	26	7
$\mathbf{P}_{15}$	31	31	29	5
$\mathbf{P}_{16}$	40	39	71	6
$\mathbf{P}_{17}$	46	46	122	6

6.3.1.1 Generic primal-dual method (PD) vs. key path decomposition method (KEY)

In this section we take a closer look at algorithms PD and KEY. KEY and PD should have a similar number of primal-dual iterations since they only differ in using different techniques to solving the RPP, where KEY uses the key path decomposition method and PD uses CPLEX. Table 33 compares the number of iterations. The results show that both PD and KEY do take almost same number of primal-dual iterations. They are not identical since there exist multiple optimal dual solutions (i.e., feasible dual improving directions) for the RPP.

The reason KEY performs much worse than PD for problems $\mathrm{P}_1,\ldots,\mathrm{P}_6$ is that there are many iterations of key path swapping when solving the relaxed RPP. Table 32 shows that t_{key}, the time spent on swapping key paths and solving the relaxed RPP, is usually the most time-consuming component in KEY. This fact can also be confirmed by observing that both PD and KEY have about the same amount of t_{sp} and t_θ. In other words, the key path decomposition method that iteratively solves smaller relaxed LPs seems to be less efficient than the ordinary LP methods that solve a larger and more difficult LP, at least for the larger cases we have tested.

However, this does not mean that KEY is always less efficient than PD. Indeed, for example, in all problems $P_7,\ldots,P_{17}$, KEY performs slightly better than PD. In particular, the key path decomposition method is designed to be more efficient for problems with a large number of OD commodities so that the time saved by solving the relaxed RPP will outweigh the time spent in swapping key paths. Problems $P_7,\ldots,P_{17}$ have more commodities than problems $P_1,\ldots,P_6$. Thus, it is not surprising that algorithm KEY is more efficient in solving problems $P_7,\ldots,P_{17}$, than algorithm PD.

To improve the relative efficiency of algorithm KEY, techniques to shorten t_{key} are required. In this thesis, we have proposed the method of adding positive perturbations on the objective coefficients of the path flows in the relaxed RPP to avoid degenerate pivots and cycling problems, but more work is necessary.

6.3.1.2 Generating shortest paths for Dantzig-Wolfe decomposition

Because DW performs so efficient, in all of our tests, we try to find out the best of four implementations of DW: DW_{OO}, DW_{OA}, DW_{AO} and DW_{AA}: DW_{OO} generates only a single shortest path for each commodity in all iterations; DW_{OA} generates only a single shortest path for each commodity in the initial iteration, but then generates all shortest paths for each commodity in every other iterations; DW_{AO} generates all shortest paths for each commodity in the initial iteration, but then generates a single shortest path for each commodity in the latter iterations; and DW_{AA} generates all shortest paths in all iterations.

The goal here is to see whether generating all shortest paths for each commodity in some iterations will shorten the total running time. In other words, we try to see how many good columns should be generated to shorten the overall running time in the column generation scheme of DW. Intuitively, if we can generate many good columns earlier, we may achieve optimality earlier. However, generating all of the columns with negative reduced cost (i.e., all shortest paths, in our case) can be time consuming. Compromises in the number of columns generated (i.e., generating at most a specific number of columns) might be worth investigating.

Note that in our DW implementation, we initially add $|N|+|K|$ artificial variables with

Table 34: Total time (ms) of four DW implementations

	$\mathbf{DW}_{OO}$	$\mathbf{DW}_{OA}$	$\mathbf{DW}_{AO}$	$\mathbf{DW}_{AA}$
$\mathbf{P}_1$	57	92	63	96
$\mathbf{P}_2$	65	104	79	122
$\mathbf{P}_3$	58	102	73	107
$\mathbf{P}_4$	62	114	80	136
$\mathbf{P}_5$	124	294	155	308
$\mathbf{P}_6$	144	277	208	325
$\mathbf{P}_7$	13	21	22	30
$\mathbf{P}_8$	15	28	25	36
$\mathbf{P}_9$	27	47	35	55
$\mathbf{P}_{10}$	27	48	33	56
$\mathbf{P}_{11}$	18	28	24	36
$\mathbf{P}_{12}$	20	34	27	40
$\mathbf{P}_{13}$	19	33	28	43
$\mathbf{P}_{14}$	22	39	32	48
$\mathbf{P}_{15}$	17	28	25	38
$\mathbf{P}_{16}$	20	35	29	44
$\mathbf{P}_{17}$	23	39	31	48

large cost so that initial basic feasible solutions can be easily identified by CPLEX.

Table 34 compares running time for four different DW implementations. It shows that DW_{OO}, the implementation that only generates single shortest path for each commodity in all iterations, is the most efficient one, DW_{AO} is the second fastest, DW_{OA} is third, and DW_{AA} is the slowest implementation, taking more than twice the time of DW_{OO}. Because of the results, we use DW_{OO} for the comparisons in Table 32 and 33.

Table 34 shows that, at least for our limited tests, generating all shortest paths is not a good idea. Generating all shortest paths may reduce the number of total iterations, but DW_{OO} requires fewer than 10 iterations for all test problems (as shown in Table 33). Generating all shortest paths may indeed be a time-consuming operation. This may also explain why the other methods, PD and KEY, which requires all shortest paths, may be less efficient.

6.3.2 Impact of good shortest path algorithms

Three of the algorithms, especially PD and KEY, require extensive computations of shortest paths. Thus, an efficient shortest path algorithm is crucial for practical efficiency.

In particular, both PD and KEY generate all shortest paths to construct the RPP (this

takes time t_{sp}). Also, they need to solve many iterations of MPSP to determine the step length θ^* (this takes time t_θ). Table 33 shows that PD spends approximately 85% of its total running time doing shortest path computations. KEY spends about the same amount of time as PD in shortest path computations. DW spends approximately 30% to 50% of its time computing shortest paths.

There are at least two ways to shorten the time spent on shortest path computations in PD and KEY: one is to find a better dual improving direction so that the total number of primal-dual iterations can be reduced, and the other is to design more-efficient shortest path algorithms.

All of the shortest path problems encountered in these algorithms are, in fact, MPSP problems. In particular, each commodity corresponds to an OD pair, and shortest paths between several OD pairs are repeatedly computed either to test whether they are dual infeasible (when determining θ^*) or to generate all shortest paths for each requested OD pair (when constructing the RPP). These MPSP problems share the same topology. All of these issues point to opportunities in the design of efficient MPSP algorithms for problems with fixed topology, which is the topic we have discussed in Chapter 2, Chapter 3, and Chapter 4 of this thesis.

6.4 Summary

In this chapter we first discussed generic primal-dual methods (PD) for solving origin-destination multicommodity network flow (ODMCNF) problems. We proposed two methods to determine the step length θ^* and choose the latter one for our implementation. We perturbed the objective coefficients of the artificial variables to resolve the degenerate pivots caused by primal degeneracy.

We then proposed a new method, the primal-dual key path method (KEY), to solve ODMCNF problems. We discussed its properties and difficulties, and proposed techniques to resolve cycling problems caused by dual degeneracy.

We compared our algorithm KEY with other three algorithms, PD, DW, and NA. We found DW, the Dantzig-Wolfe decomposition method, was the fastest algorithm in all of our

tests. As expected, solving the node-arc formulation (NA) without any special techniques such as resource-directive methods or the basis-partitioning methods introduced in Chapter 2, will be time-consuming and very inefficient, except for small networks.

KEY follows the generic PD steps but uses the key path decomposition method to solve a relaxed RPP at each iteration. It is designed for problems with large number of commodities. In our limited tests, KEY does perform better than PD for cases with more commodities. However, for larger cases with few commodities, it performs much worse than PD. The reason for its inefficiency is that it requires many iterations of key path swapping operations. Although our proposed technique, which perturbs the objective coefficients for path flows, has reduced the chances of key path swapping and resolved the cycling problem caused by dual degeneracy, it is still not efficient enough for some larger cases.

More tests should be performed, especially using cases with a large number of commodities, to draw more solid conclusions on the efficiency of KEY.

We also found that generating all shortest paths for the same commodity may not save time in Dantzig-Wolfe decomposition with a column generation scheme. That is, generating a single shortest path for each commodity is good enough.

To explain why DW performs so well compared to PD and KEY, we give the following reasons:

First, DW generates fewer shortest paths than DW and KEY. In particular, DW will not generate a path twice, if we keep all of the previously generated paths in the restricted master problem. PD and KEY, on the other hand, generate a new set of all shortest paths at every iteration using the new dual solutions. Note that between two primal-dual iterations, many paths may remain the shortest and do not need to be altered. The reason that we remove the entire previous set of shortest paths and add a new set of shortest paths is that keeping old paths would require bookkeeping on each specific path, which is difficult and inefficient. In particular, to check whether a path is in the current set takes exponential time when there are (exponentially) many paths to be added or removed. Furthermore, using CPLEX to solve the RPP usually requires the sparse representation of the constraint matrix. In this case, each path corresponds to a column. Removing and adding columns

will require a new sparse representation. Although we can add easily columns at the end of the matrix, identifying the columns to be removed requires sparse matrix operations. Thus, PD and KEY invoke many more operations than DW does, making the algorithms less efficient.

Degeneracy is an important issue, especially for primal-dual based methods like PD and KEY. Multiple optimal dual or primal solutions may cause implementation problems. Techniques that choose a "good" optimal solution are desired, and may drastically shorten the total running time.

KEY has one more issue of concern than PD: the memory requirement. To implement the key path swapping operations more efficiently, we allocate memory to store the initial relaxed RPP at each primal-dual iteration. Inside a primal-dual iteration, KEY may perform many iterations of key path swapping. The column corresponding to the key path is empty, and an empty column has no sparse representation at all. Thus, it is clumsy to do the key path swapping and column operations only using the original sparse representation of the RPP. Although we have an efficient way to swap key paths and update each column for the commodities that have new key paths, it still requires much time, especially when there are many iterations of key path swapping operations. Table 33 and Table 34 show that more iterations of swapping key paths results in a longer running time compared to PD.

To make KEY more competitive, new techniques that identify better dual improving directions can be developed. Our results show that efficient MPSP algorithms play a crucial role in improving all of the path-oriented multicommodity network flow algorithms, especially primal-dual based algorithms like PD and KEY which 85% (PD) or 95% (KEY) of their total running time computing shortest paths.