寻找最大带宽的独立路径对算法*

谢 政,张晓明,陈 挚
(国防科技大学 理学院,湖南 长沙 410073)

摘 要:独立多路径算法在多径算法研究中具有重要地位。最小时延多路径问题的研究已较为成熟,而最大带宽多路径问题的研究却刚刚起步。文章介绍了有向图中链路独立路径对问题,提供了一种复杂度为$O(mn\log n)$求解该问题的多项式算法。该算法不需要考虑最大带宽链路独立路径对上流值分配问题,能够更好地应用到现实网络中。

关键词:多路径;链路独立;最大带宽路径对;容量;WPAP
中图分类号:O224 **文献标志码**:A **文章编号**:1001-2486(2012)05-0158-06

Finding the widest pair of arc-disjiont paths

XIE Zheng, *ZHANG Xiaoming*, *CHEN Zhi*
(College of Science, National University of Defense Technology, Changsha 410073, China)

Abstract: Disjoint paths routing catches special attention in multipath research. Shortest disjoint paths problem has been studied maturely, while the research of widest disjoint paths problem is just under development. In this paper, we presented a problem of finding the widest pair of arc-disjoint paths in a directed graph. It differs from Multipath Flows, which may be transformed into the problem of Maximum Flow and Minimum Cut. Then we propose a polynomial algorithm for it with time complexity $O(mn\log n)$. It is not required to reassign flux at common vertices on the widest pair of paths, and this scheme can be implemented easier in real networks.

Key words: multipath; arc-disjoint; widest pair; capacity; WPAP(Widest Pair of Arc-disjoint Paths)

传统的单一最短路径算法解决的是网络中两节点间一条最短路径的选取问题。但随着网络的复杂化,链路与节点时常暂时或者完全失效;同时,节点的移动,链路的切换,使得网络拓扑结构具有不稳定性。因此,依照传统的最短路径算法求得的传输路径将因为网络拓扑结构的快速变化而频繁中断,这必然导致重路由开销增大,从而加重网络的运行负担,也将极大地降低信息传输效率。随着信息社会的到来,大容量数据传输已成为必然,使用单一路径传输方案将造成局部负载过重,从而引起网络拥塞以及整体利用率低下等问题。越来越多的研究表明,单路径机制难以满足现代网络应用的需求[6-9]。

多径路由可以大大减少路由开销,降低网络延时并提高数据传输速率,减少网络拥塞,均衡网络负载,增强可靠性。近年来,多路径路由渐渐引起人们的重视[4,7-9]。与单路径相比,多路径能够更好地平滑网络流量、减少网络拥塞、提高网络利用率和改善服务质量[8,10]。

独立多路径算法是多路径算法的重要方面,这是因为独立多路径具有较强的生存能力。多路径可以提供持续能力更强的服务,在某条链路或某个节点失效的情况下,备用路径能继续提供传输服务。

现有的关于给定源-宿节点对的独立多路径算法主要有以下几种:Suurballe 和 Tarjan 提出了基于最小时延独立路径对算法[11-12];Chekuri 和 Guruswami 分别提出了相近的链路独立多路径算法(EDP)[1,13];Nelakuditi 和 Zhu 初步解决了最大带宽独立多路径问题(WDP)[2-3,5];Kishimoto, Aggarwal 和 Kabadi 分别研究了 k-路径流问题[4,7,14]。然而,迄今为止,很少有人关注最大带宽的独立多路径对问题。

带宽(或者容量)是 QoS 问题最重要的技术指标之一,它体现出信息传输能力的强弱。链路独立路径比非独立路径具有更高的可靠性,同时相对节点独立路径具有更低的复杂性,在理论和应用领域具有很强的研究价值。

本文的目标是在所有链路独立路径对中找出带宽和最大的路径对,与最小时延路径对相似,上

* 收稿日期:2012-03-02
基金项目:国家部委资助项目
作者简介:谢政(1960—),男,教授,E-mail:xiezheng@nudt.edu.cn

述路径对定义为最大带宽链路独立路径对(Widest Pair of Arc-disjoint Paths,WPAP)。

1　概念和定义

有向图(或网络)D = (V,A,c),其中 c 是非负弧容量函数:A→$\mathbf{R}^+$,(s,t)是 V 中的一对源、宿节点,设$|V|=n$,$|A|=m$。对任意的弧$(i,j)\in A$,记 $c(i,j)=c_{ij}$。节点入邻域(出邻域)包含的节点数称为入度(出度)。路径 p 表示为 $p=v_1v_2\cdots v_i\cdots v_l=12\cdots l$,其中 v_i 是路径 p 中的节点,$1\leqslant i\leqslant l$,p 称为 $1-l$ 路径。称 $j(j+1)\cdots(k-1)k$ 为 p 的节,记为 $p(j,k)$,其中 $1\leqslant j<k\leqslant l$。全文中如无特别说明,一条路径是指的一条 $s-t$ 路径。

路径 p 的容量为 $c(p)=\min_{(i,j)\in A(p)}c(i,j)$,其中 $A(p)$是路径 p 的所有弧组成的集合;两条链路独立路径 p_1 和 p_2 的容量为 $c(p_1)+c(p_2)$。

在我们讨论的网络中,带宽和容量是等价的,后面将不作区分。

2　寻找 WPAP 的算法

要寻找最大带宽链路独立路径对,一个自然的想法是先在 D 中找出最大带宽路径 p,然后在 D 中去掉 p 的弧,得到的网络记为$\hat{D}$,再在$\hat{D}$中求出最大带宽路径$\hat{p}$。p 和$\hat{p}$显然是链路独立的,然而它们未必是带宽最大一对路径。另一种方法是模仿文献[8]中寻找最短链路独立路径对的方法,先求出 D 中最大带宽路径 p,在 D 中将 p 的弧反向得到 D',再找出 D'中最大带宽路径 p',然后将 p 和 p'结合起来,得到两条路径也是链路独立的,但同样未必是带宽最大的。必须寻找一种新的方法解决这个问题。

在寻找 WPAP 之前,先要找到网络的最大带宽路径。Edmonds 和 Karp 利用改进的 Dijkstra 算法在 $O(m\log n)$[15-16]的时间内解决了最大带宽路径问题。这种方法被广泛应用到最大流问题和 QoS 路由问题,特别是在模拟仿真当中,几乎都首先采用改进的 Dijkstra 算法来寻找 v_i 最大带宽路径。

2.1　p 和 WPAP 的关系

为方便,假设网络 D 中仅有一条带宽最大路径(如有多条,将其排序为 1,2,…,j,取第一条为最大带宽路径)。记最大带宽路径 $p=i_1i_2\cdots i_l$,其中 $v_{i1}=v_s$,$v_{il}=v_t$。

p 不能直接用于寻找 WPAP,但同样起到重要作用,从下面的定理,可以看到 p 和 WPAP 的密切关系。

定理 1　假设 p 是网络 D 中最大带宽路径,p_1 和 p_2 是 WPAP,$c(p_1)\geqslant c(p_2)$,那么或者 $c(p_2)=c(p)$,或者 $A(p)\cap A(p_1)\neq\varnothing$。

证明　假设 $A(p)\cap A(p_1)=\varnothing$,那么 p 和 p_1 是一组链路独立的路径对,若 $c(p_2)\neq c(p)$,那么 $c(p_2)<c(p)$,从而 $c(p)+c(p_1)>c(p_1)+c(p_2)$,这与 p_1 和 p_2 是 WPAP 矛盾。

记 p 与 WPAP 的公共弧为 $A_1(p)$,p 的其他弧为 $A_2(p)$。将 $A_2(p)$中的弧反向,容量不变,记为 $A_2'(p)$。

构造一个新图 $P=(V(p_1)\cup V(p_2),(A(p_1)\cup A(p_2)\cup A_2'(p))\setminus A_1(p))$,可以得到如下结论。

定理 2　删除 P 中所有度为 0 的节点,得到的图记为 p',那么 p'是 $D'=(V,(A\cup A(\hat{p}))\setminus A(p),c)$中的一条 $s-t$ 路径,其中 $\hat{p}$ 是将 p 的弧反向得到的一条 $t-s$ 路径。

证明　设 $G=(V(p_1)\cup V(p_2),A(p_1)\cup A(p_2))$,显然,$G$ 中节点的度或者为 2 或者为 4。例如在图 1 中,$v_s=v_{i1}$,$v_t=v_{il}=v_{10}$,其中 v_{i3}是 p 上的节点,但不在 G 中。

这个图有 3 类节点:a. 仅有 2 条出弧或者入弧,例如 v_s 和 v_t;b. 入弧和出弧均为 1 条,例如 v_{i2} 和 v_{i7};c. 入弧和出弧均为 2 条,例如 v_{i8}。

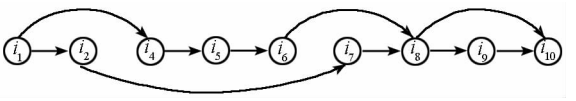

图 1　由 WPAP 组成的图 G

Fig. 1　Graph G consisting of WPAP

我们对 3 种情况分别给出证明。

对于 a 类型中的节点,根据定理 1,按照$(A(p_1)\cup A(p_2))\setminus A_1(p)$操作时,将删除 v_s 的一条出弧和 v_t 的一条入弧,而添加 $A_2'(p)$中的弧时对这两个节点没有影响。因此在 P 中 v_s 仅有一条出弧且 v_t 仅有一条入弧。

对于 b 类型中的节点,又有 3 种子类型:b1. 节点的入弧是 p 上的弧,而出弧不是,例如 v_{i2}和 v_{i6};b2. 节点的出弧是 p 上的弧,而入弧不是,例如 v_{i4};b3. 节点的入弧和出弧都是 p 上的弧,例如 v_{i5}和 v_{i9}。当删除 $A_1(p)$中的弧时,b1 类型中的节点失去入弧,但在添加 $A_2'(p)$中弧时得到新的入弧;b2 类型中的节点在上述操作中失去原有入弧并得到新的入弧。因而,这两种类型的节点仅有 1 条出弧和 1 条入弧。b3 类型中的节点在删除和添加弧的操作后不再有弧。

对于 c 类节点,每个节点有 2 条出弧和 2 条

入弧,其中一对在 p 上,另外 1 对不在 p 上(与引理 1 的证明类似,如果两对弧都不在 p 上,其中一对可用 p 上的弧代替)。通过删除 $A_1(p)$ 中的弧,每个节点仅有一条入弧和一条出弧,例如 v_{i8}。

注意到通过添加 $A'_2(p)$ 中的弧,某些节点重新出现,例如图 2 中的 v_{i3} 从 $A'_2(p)$ 中获得一条入弧和一条出弧。

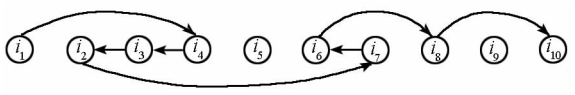

图 2 路径 p'

Fig. 2 Path p'

删除所有度为 0 的节点,我们得到一条不同于 p 的路径 p'。

从定理 2 可以看出,要找到 WPAP,首先要找到与之相关的 p'。

2.2 寻找与 WPAP 相关的 p'

我们当前的目标是寻找 D' 中与 p 结合可以直接构造出 WPAP 的 p'。

前面已经讨论过,p' 可能不是 D' 中的最大带宽路径,因而不能直接利用最大带宽路径算法求解。从定理 2 的证明可以看出,p' 被分成若干个 v_{i_j} 到 v_{i_k} 节,其中 $1 \leqslant j < k \leqslant l$,例如图 2 中的 $p'(i_1, i_4), p'(i_4, i_2), p'(i_2, i_7), p'(i_7, i_6), p'(i_6, i_8), p'(i_8, i_{10})$。进一步,这些节又能分成 A 和 B 两个部分。A 部分中的节,源、宿节点都在 $\hat{p}$ 上,而其他节点不在 $\hat{p}$ 上,例如 $p'(i_1, i_4), p'(i_2, i_7), p'(i_6, i_8), p'(i_8, i_{10})$;$B$ 部分中的节,节点和弧都在 $\hat{p}$ 上,例如 $p'(i_4, i_2), p'(i_7, i_6)$。也就是说,$D'$ 仅有两类路径对寻找 p' 有贡献:一类是弧都不在 $\hat{p}$ 的 $i_j - i_k$ 路径,另一类是 $\hat{p}(i_k, i_j)$,其中 $j < k$。因此,可以通过删除 D' 多余的节点和弧来简化 D',这将可以极大地简化寻找 p' 的工作。

我们的最终目标是寻找最大带宽链路独立路径对,因此 A 部分中每一个 $p'(i_j, i_k)$ 节都是 D' 中最大带宽 $i_j - i_k$ 路径。当前的任务是寻找 D' 中最大带宽 $i_j - i_k$ 路径,其中 $1 \leqslant j < k \leqslant l$。步骤如下:

(1)求出 D 中最大带宽 $s - t$ 路径 p,并标记 p 上的节点。

(2)对任意 $j, 1 \leqslant j \leqslant l-1$,在 $D - A(p) - \{v_{i_1}, v_{i_2}, \cdots, v_{i_{j-1}}\}$(当 $j = 1$ 时,网络为 $D - A(p)$)中,找出以 v_{i_j} 为根的最大带宽树 T_{i_j},检查 T_{i_j} 的叶子节点是否在 p 上,如果不在,删除节点及其入弧,并更新叶子节点集,最后 T_{i_j} 的剩余部分记为 T'_{i_j}。

(3)构造一个新的网络 $D^* = (\cup_{j=1}^{l-1} V(T'_{i_j}), \cup_{j=1}^{l-1} A(T'_{i_j}) + A(\hat{p}), c)$。

定义 1 如果在 $D^* - A(\hat{p})$ 中存在 v_{i_1} 到 v_{i_j} 的路径,把 v_{i_j} 称为可达节点(RV, reachable vertex);如果 v_{i_1} 到 v_{i_j} 路径上不存在其他节点,称 v_{i_j} 为直接可达节点(DRV, directly - reachable vertex),路径称为直达路径。如果一条路径包含 D^* 中 $\hat{p}$ 的一个节,称它倒退一次;如果包含 $\hat{p}$ 上没有公共节点的两个节,称它倒退 2 次;余者类推。

在 D^* 中的 $i_j - i_k$ 路径上,$1 \leqslant j < k \leqslant l$,如果没有除 i_j, i_k 以外的节点在 $\hat{p}$,则将 $i_j - i_k$ 路径缩减成 (i_j, i_k) 弧,并记得到的新网络为 D_0。

在讨论如何寻找 p' 之前,定义一些符号。

$p'(i)$:D_0 中以 i 为终点的当前最优路径。

c':由 $p' = p'(d)$ 所决定的最大带宽独立路径对的带宽。

$Label_{ij} = 1$ 表示 $(i, j) \in A(\hat{p})$;$Label_{ij} = 0$ 表示 $(i, j) \notin A(\hat{p})$。

如果在路径 $p'(i)$ 中,v_k 的入弧和出弧都不在 $\hat{p}$ 上,而 $p'(k, i)$ 节上的其他任意节点的出弧或入弧在 $\hat{p}$ 上,称 v_k 是 v_i 前的最后一个公共节点。

如果节点 v_i 在 $p'(j, k)$ 上,而 v_j 和 v_k 都是公共节点,且 v_k 是 v_j 的邻居公共节点,称 $p'(j, k)$ 为 v_i 的生存节。

例如在图 3 中,假设当前最优路径为 $p'(i_7) = i_1 i_4 i_3 i_2 i_7$,可以得到一对链路独立路径 $p_1(i_7) = i_1 i_4 i_5 i_6 i_7$ 和 $p_2(i_7) = i_1 i_2 i_7$。易知,一条路径 $p'(i)$ 仅与一对路径相对应。

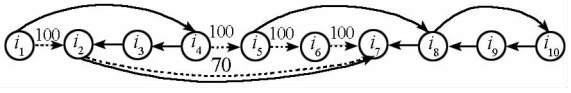

图 3 $p'(i)$ 与 $p_1(i)$ 和 $p_2(i)$ 之间的关系

(虚线表示 $p_1(i)$ 和 $p_2(i)$ 上的弧)

Fig. 3 Relation between $p'(i)$ and the pair of or $p_1(i)$, $p_2(i)$ (Dashed lines indicate arcs on $p_1(i)$ and $p_2(i)$)

给定一个网络 D 及其最大带宽 $s - t$ 路径 p,将 p 上所有弧反向,容量保持不变,可以得到一条 $t - s$ 路径 $\hat{p}$ 和一个新网络 D'。不妨设 D' 中存在一条 $s - t$ 路径 p',若不然,D 中必不存在链路独立路径对。按照如下方法构建一个子网络:首先,增加 $V(p) \cup V(p')$ 中的顶点和 $A(p) \cup A(p')$ 中的弧到空图;然后,检查是否存在方向相反的弧对,如果存在,删除之;最后将图中度为 0 的顶点删除,得到的图记为 $p \oplus p'$。

构建 $p'(j) \oplus p(s, j)$ 得到的两条路径可以被分解为若干个部分,每个部分由两条顶点独立的路径组成。以 v_j 之前最后一个公共节点 v_i 为起

点,经过不在 p 上的那条 v_i 的出弧的路径称为上路径,经过在 p 上的那条 v_i 的出弧的路径称为下路径。上路径(下路径)的容量记为 $u(j)$($d(j)$)。

对每个顶点 v_i,定义一个包含 9 个元素的向量,记为

$ARRAY(i)=(pr(i),in(i),t(i),u(i),d(i),\max l(i),\min l(i),\max(i),\min(i))$

其中

$pr(i)\in\{j\mid v_j\in V(D_0)\}$:$v_i$ 在 p'_i上的前一个顶点;

$in(i)\in\{0,1\}$:$in(i)$描述 v_i 的入弧(j,i)是否在 $\tilde{p}$ 上,有 $in(i)=Label_{ij}$;

$t(i)\in\{0,1\}$:$t(i)=0$ 表示当前最优路径 p' 反向偶数次,否则表示反向奇数次;

$u(i)\in\{c_{jk}\mid(j,k)\in A(D_0)\}$:由 $p'(i)$决定的路径对中上路径的容量;

$d(i)\in\{c_{jk}\mid(j,k)\in A(D_0)\}$:由 $p'(i)$决定的路径对中下路径的容量;

$\max l(i)\in\{c_{jk}\mid(j,k)\in A(D_0)\}$:由源节点到 v_i 之前最后一个公共节点的当前最优路径决定的路径对中容量较大路径的容量;

$\min l(i)\in\{c_{jk}\mid(j,k)\in A(D_0)\}$:由源节点到 v_i 之前最后一个公共节点的当前最优路径决定的路径对中容量较小路径的容量;

$\max(i)\in\{c_{jk}\mid(j,k)\in A(D_0)\}$:由 $p'(i)$决定的路径对中容量较大路径的容量;

$\min(i)\in\{c_{jk}\mid(j,k)\in A(D_0)\}$:由 $p'(i)$决定的路径对中容量较小路径的容量。

设置 $Access(i)$ 描述顶点 v_i 是否被访问过,$Access(i)=0$ 表示没有被访问过,而 $Access(i)=1$ 表示已被访问过。若 $Access(i)=1$,而 v_i 被再次访问,我们用新向量 $ARRAY'(i)$来存储新的元素值并选择较优的向量。

寻找 p'的算法步骤如下:

Step 0　在 D_0 中,删去 v_s 的所有入弧,初始化: $pr(s)=in(s)=t(s)=0$,$d(s)=c(p)$,其余元素为∞。$c'=c(p)$,p'为空。对 v_s,设置 $Access(s)=1$;对其他顶点 v_i 设置 $Access(i)=0$。记 DRV 集合为 S_{DRV}。

Step 1　从 S_{DRV} 中选取一个顶点 v_{i_j},转 Step 2。

Step 2　删去除 v_{i_j}以外的所有从 v_s 到 S_{DRV}的入弧。

Step 3　设置一个 FIFO 队列 Q 用于存储待检查顶点。初始化 $Q\leftarrow v_s$,$inQ(s)=1$,对任意其他顶点 v_i,$inQ(i)=0$。

Step 4　提取 Q 的第一个顶点 v_i,检查它的每一条出弧(i,j):

如果 $Access(j)=0$,计算出 $ARRAY(j)$,将 v_j 添加到 Q,并令 $Access(j)=1$,$inQ(j)=1$;

如果 $Access(j)=1$,计算出 $ARRAY'(j)$;

如果 $\min(i)'+\max(i)'>\min(i)+\max(i)$,令 $ARRAY(j)=ARRAY'(j)$;

如果 $inQ(j)=0$,将 v_j 添加到 Q,令 $inQ(j)=1$;

当 v_i 的所有出弧检查完毕,转 Step 5。

Step 5　如果 Q 为空,转 Step 6;否则转 Step 4。

Step 6　如果 $\min(d)+\max(d)>c'$,令 $c'=\min(d)+\max(d)$并更新 p'。转 Step 7。

Step 7　如果 S_{DRV}为空,结束;否则转 Step 1。

在 Step 4 中求解 $ARRAY(j)$具体如下:首先,可以从 $Label_{ij}$得到 $in(j)$,设 $pr(j)=i$;然后分三部分进行讨论。

(1)第一个部分

如果 $in(i)=0$,$in(j)=1$,那么 $t(j)=[t(i)+1]_2$。

如果 $in(i)=0$,$in(j)=0$,那么 $t(j)=t(i)$。

如果 $in(i)=1$,那么 $t(j)=t(i)$。

(2)第二个部分

如果 $in=0$,$in(j)=0$,那么 $\min l(j)=\min(i)$,$\max l(j)=\max(i)$,

$u(j)=\min\{c(i,j),u(i)\}$,$d(j)=d(i)$;

否则,$\min l(j)=\min l(i)$,$\max l(j)=\max l(i)$

如果 $in(i)=0$,$t(j)=0$,

那么 $u(j)=\min\{u(i),c(i,j)\}$,

$d(j)=\min\{d(i),c(\tilde{p})\}$;

如果 $in(i)=0$,$t(j)=1$,

那么 $u(j)=u(i)$,$d(j)=\min\{d(i),c(i,j)\}$;

如果 $in(i)=1$,$t(j)=0$,

那么 $u(j)=u(i)$,$d(j)=d(i)$;

如果 $in(i)=1$,$t(j)=1$,

那么 $u(j)=u(i)$,$d(j)=d(i)$

(3)第三个部分

$\min\_c(j)=\min\{\min l(j),\min\{u(j),d(j)\}\}$

$\max\_c(j)=\min\{\max l(j),\max\{u(j),d(j)\}\}$

如果 $Access(j)=1$,我们计算并比较 $\min(i)'+\max(i)'$与 $\min(i)+\max(i)$,选择这两个值较大者所在向量。

2.3　算法的证明

2.3.1　计算向量 $ARRAY(j)$的证明

假设弧(i,j)被检测。$t(j)$描述 $p'(j)$反向的次数。如果 $in(j)=1$,那么$(i,j)\in A(\tilde{p})$,有两种情况:(Ⅰ)$in(i)=0$;(Ⅱ)$in(i)=1$。对情况(Ⅰ),$p'(j)$

反向的次数增加 1,因而 $t(j)=[t(i)+1]_2$;对情况(Ⅱ),v_i 和 v_j 的入弧在 $\hat{p}$ 的同一个节上,不增加反向次数,因而 $t(j)=t(i)$。如果 $in(j)=in(i)=0$,那么 v_i 是一个公共顶点,一个新的节出现,反向次数不增加,$t(j)=t(i)$;如果 $in(j)=0$ 而 $in(i)=1$,那么 $t(j)$不变,即 $t(j)=t(i)$。

如果 $in(j)=in(i)=0$,这意味着 v_i 是 v_j 之前的最后一个公共顶点,例如,图 3 中 $v_i=v_{i_8}$,$v_j=v_{i_{10}}$,因而 $\min l(j)$ 等于 v_i 的最小容量,$\max l(j)$ 类似。否则,v_i 与 v_j 有相同的最后公共节点,即 $\min l(j)=\min l(i)$,$\max l(j)=\max l(i)$。

检查 v_j 时,通过跟踪从 v_j 开始每个节点的入弧上的节点,可以得到一条当前最优路径 $p'(j)$。检验当前最优路径 $p'(j)$ 是否得到优化的标准是 $\min(j)+\max(j)$。

如果 $in(j)=0$,如图 3 所示,通过构造 $p'(j)\oplus p(s,j)$,可以得到两条链路独立的 $s-j$ 路径。如果 $in(j)=1$,同样可以得到两条链路独立的路径,但是他们终点不同。例如,设 $v_j=v_{i_3}$,$p'(3)=i_1i_4i_3$,那么得到的两条路径分别为 i_1i_4 和 $i_1i_2i_3$。对于这两种情况,记容量小(大)的一个为 $\min(j)$ $(\max(j))$。

下面,给出计算 $\min(j)$ 和 $\max(j)$ 的方法。

$$\min(j)=\min\{\min l(j),\min\{u(j),d(j)\}\}$$

$$\max(j)=\min\{\max l(j),\max\{u(j),d(j)\}\}$$

接下来计算 $u(j)$ 和 $d(j)$。

如果 $in=in(i)=0$,即 v_i 是一个公共顶点,可以得到 $u(j)=c(i,j)$,$d(j)=c(p)$。

否则,将 D_0 中的其他顶点分为 4 种情况:

(1)$in(j)=0$,$t(j)=0$,例如图 3 中的 v_{i_4},v_{i_8}。由于 (i,j) 是 v_j 的上路径,有 $u(j)=\min\{u(i),c(i,j)\}$,v_j 的下路径可以通过连接 p 的一节和 v_i 的下路径得到,因此 $d(j)=\min\{d(i),c(p)\}=d(i)$。

(2)$in(j)=0$,$t(j)=1$,例如图 3 中的 v_{i_7}。(i,j) 出现在 v_j 的下路径,因此 $d(j)=\min\{d(i),c(i,j)\}$,类似于情况(1),有 $u(j)=u(i)$。

(3)$in(j)=1$,$t(j)=0$,例如图 3 中的 v_{i_5},v_{i_6}。v_j 的下路径与 v_i 的相同,因此 $d(j)=d(i)$。与 v_i 相比,v_j 的上路径失去了弧 (i,j),但其容量没有减少,因而 $u(j)=u(i)$。例如,假设 $v_i=v_{i_7}$,$v_j=v_{i_6}$,那么 v_{i_7}的上路径是 $i_1i_4i_5i_6i_7$,而 v_{i_6}的上路径是 $i_1i_4i_5i_6$。$i_1i_2i_7$ 是 v_{i_7}的下路径,同时也是 v_{i_6}的下路径。

(4)$in(j)=1$,$t(j)=1$,例如 v_{i_2},v_{i_3}。类似于情况(3),$u(j)=u(i)$,$d(j)=d(i)$。

以上过程得到 $ARRAY(j)$ 的 9 个元素。

2.3.2 寻找 p'的证明

在 2.1 节,通过两条链路独立路径构造 p'。这里,需要反过来由 p'构造 p_1 和 p_2。对于 $A(p)+A(p')$,如果 (i_{j+1},i_j) 在 p'上,删去 (i_j,i_{j+1})。剩下的弧生成的图记为 D_{2_path},它由两条链路独立路径组成。

值得注意的是,在 D_{2_path} 中,可能有多种方法组合两条链路独立路径。如图 4:$\alpha_{11}+\alpha_{12}+\alpha_{13}$ 与 $\alpha_{21}+\alpha_{22}+\alpha_{23}$,$\alpha_{21}+\alpha_{12}+\alpha_{13}$ 与 $\alpha_{11}+\alpha_{22}+\alpha_{23}$ 等。

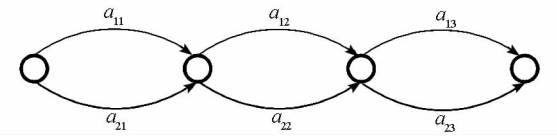

图 4 由最大带宽路径对组成的 D_{2_path}

Fig. 4 Graph D_{2_path} consisting of the widest pair

因此,需要在 D_{2_path} 中选出带宽最大的路径对 p_1 和 p_2。

假设度为 4 的顶点有 k 个,那么 p_1 和 p_2 都被分成 $k+1$ 个部分,并被分成 $k+1$ 对。每一对里面的 α_{1j},α_{2j} 节都是链路独立的,并且有相同的端点,其中 $1\leqslant j\leqslant k+1$。选择每一对里面带宽较大的节组成 p_1,其余的节组成 p_2。

定理 3 按照上述方法组成的 p_1 和 p_2,是 D_{2_path} 中带宽最大的路径对。

证明 在上述方法中,

$$c(p_1)=\min\{\max\{c(\alpha_{11}),c(\alpha_{21})\},\cdots,\max\{c(\alpha_{1(k+1)}),c(\alpha_{2(k+1)})\}\}$$

$$c(p_2)=\min\{\min\{c(\alpha_{11}),c(\alpha_{21})\},\cdots,\min\{c(\alpha_{1(k+1)}),c(\alpha_{2(k+1)})\}\}$$

显然,p_1 是所有 $s-t$ 路径中容量最大者。假设存在另外一对链路独立路径 p_x 和 p_y,其中 $c(p_x)\geqslant c(p_y)$。设 $c(\alpha_{i_0j_0})=c(p_2)$,其中 $i_0\in\{1,2\}$,$j_0\in\{1,2,\cdots,k+1\}$,即 $\alpha_{i_0j_0}$是容量最小的一条弧。易知,$c(p_y)=c(\alpha_{i_0j_0})=c(p_2)$,$c(p_1)\geqslant c(p_x)$,因而 $c(p_1)+c(p_2)\geqslant c(p_x)+c(p_y)$。

定理 4 在 D'中,将 p'上的 (i_j,i_k) 弧还原成 i_j-i_k 路径,其中 $1\leqslant j<k\leqslant l$,得到一条 D 中的与构建 WPAP 关联的 $s-t$ 路径。

证明 将 i_j-i_k 路径缩减为 (i_j,i_k) 弧后,D_0 中的 $s-t$ 路径 p'也是 D'中的 $s-t$ 路径,为方便起见,仍记为 p'。同样地,将 (i_j,i_k) 弧还原成 i_j-i_k 路径,D_{2_path}中得到的 p_1 和 p_2 是 D 中容量最大的链路独立路径对。

3　复杂度分析

定理 5　在 $O(mn\log n)$ 的时间内可以得到 D 中的 WPAP。

证明　首先,构建 D^* 的时间复杂度为 $O(mn)$。这是因为,寻找最大带宽路径 p 的时间复杂度为 $O(m\log n)$。对任意 $j, 2 \leqslant j \leqslant l-1$,寻找最大带宽树 T_{i_j} 并检查所有叶子节点的时间复杂度为 $O(m)+O(n)$。由于有 $O(l)$ 次循环,$l \leqslant n$,因此时间复杂度为 $O(mn\log n)$。

其次,当 FIFO 队列 Q 为空时,在 $O(m)$ 的时间内得到一条 $s-t$ 路径 $p'(d)$。对每个 DRV,需要检查 $O(m)$ 条弧。事实上,在缩减原始网络 D 后,得到 D_0 的弧远远少于 m。对 D_0 中的每条弧,需要为其头顶点存储一个包含 9 个元素的向量,最多需计算 13 次。根据文献[18],当 Q 为空时,每条弧检查 $O(1)$ 次并且所有顶点都被检查完毕。因此,在 $O(m)$ 的时间内我们得到一条 $s-t$ 路径 $p'(d)$。每个 DRV 执行一个循环需要 $O(m)$ 的时间,DRV 的个数不超过 n 个,因此得到 p' 的时间为 $O(mn)$。

在 D_{2_path} 中,寻找所有度为 4 的顶点,并把 p_1 和 p_2 分成 $k+1$ 个部分需要 $O(n)$ 的时间 。计算所有节 α_{ij} 容量的复杂度为 $O(m)$,找出每一对节容量较大者需要 $O(k)$,$k \leqslant n$ 的时间。因此,构造 WPAP 的时间复杂度为 $O(m)$。加上构造 D_0 以及寻找 D_0 中 p' 所需的时间 $O(mn\log n)$,找到 WPAP 的时间复杂度为 $O(mn\log n)$。

4　结束语

在本文中,分析了最大带宽路径对问题,并给出了一个在有向图中求解 WPAP 的多项式算法,其复杂度为 $O(mn\log n)$。首先,找出最大带宽路径 $p=i_1 i_2 \cdots i_l$,并将其弧反向;然后,找出所有 $i_j - i_k$ 路径并将原图简化成 D_0;最后,找出 D_0 中一条特殊的 $s-t$ 路径 p',并求出与之相关联的 WPAP。

参考文献(References)

[1] Chekuri C, Khanna S. Edge-disjoint paths revisited[J]. ACM Transactions on Algorithms, 2007, 3(4): Article 46.

[2] Zhu S M, Gao D Q. Improvement and analysis of widest disjoint paths algorithm for proportional routing[J]. Journal of East China University of Science and Technology: Nature Science Edition, 2007, 33(3): 389 - 393.

[3] Zhu S M, Zhang Z L, Zhuang X H. Modified widest disjoint paths algorithm for multipath routing [J]. International federation for Information Processing, 2007:212 - 219.

[4] Kabadi S N, Chandrasekaran R, et al. Multiroute flows: Cut-trees and realizability [J]. Discrete Optimization, 2005, 2(3): 229 - 240.

[5] Nelakuditi S, Zhang Z L. On selection of candidate paths for proportional routing[J]. Computer Networks, 2004, 44: 79 - 102.

[6] Sidhu D, Nair R, Abdallah S. Finding disjoint paths in networks [J]. ACM SIGCOMM Computer Communication Review, 1991, 21(4): 43 - 51.

[7] Aggarwal C C, Orlin J B. On multi-route maximum flows in networks[J]. Networks, 1997, 39(1): 43 - 52.

[8] Cidon I, Rom R, Shavitt Y. Analysis of multi-path routing [J]. Transactions on Networking, 1999, 7(6): 885 - 896.

[9] Shen B H, Hao B, Arunabha S. On multipath routing using widest pair of disjoint paths[J]. Hight Performance Switching and Routing, 2004: 134 - 140.

[10] Taft-Plotkin N, Bellur B, Ogier R G. Quality of service routing using maximally disjoint paths[C]//Proc. of IEEE/IFIP IQQoS, 1999.

[11] Suurballe S. Disjoint paths in networks[J]. Networks, 1974, 4: 125 - 145.

[12] Suurballe S, Tarjan R. A quick method for finding shortest pair of disjoint paths[J]. Networks, 1984, 14: 325 - 336.

[13] Guruswami V, Khanna S, Rajaraman R. Near-optimal hardness results and aprooximation algorithms for edge-disjoint paths and related problems[J]. Comput. Syst. Sci, 2003, 67(3): 473 - 496.

[14] Takeuchi M, Kishimoto W. On two-route flows in an undirected network[R]. IEICE Technical Report, CAS90 - 19, DSP - 90 - 23. 1990.

[15] Edmonds J, Karp R M. Theoretical improvements in algorithmic efficiency for network flow problems[J]. Journal of ACM, 1972, 19(2): 248 - 264.

[16] Edmonds J, Fulkerson D R. Bottleneck extrema[R]. RAND Corp. Memorandum RM - 5375 - PR, 1968.