การจัดเส้นทางแบบมัลติคาสต์บนโครงข่ายเอทีเอ็มโดยใช้ฮิวริสติกอัลกอริทึม

นายสุชัย โรจนวิไลกุล

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมไฟฟ้า ภาควิชาวิศวกรรมไฟฟ้า
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2543

ISBN 974 - 13 - 0198 - 7

MULTICAST ROUTING OVER AN ATM NETWORK USING AN HEURISTIC ALGORITHM

Mr. Suchai Rojnawilikul

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Engineering in Electrical Engineering
Department of Electrical Engineering
Faculty of Engineering
Chulalongkorn University
Academic Year 2000

ISBN 974 - 13 - 0198 - 7

หัวข้อวิทยานิพนธ์	การจัดเส้นทางแบบมัลติคาสต์บนโครงข่ายเอทีเอ็มโดยใช้ฮิวริสติกอัลกอริทึม
โดย	นายสุชัย โรจนวิไลกุล
สาขาวิชา	วิศวกรรมไฟฟ้า
อาจารย์ที่ปรึกษา	รองศาสตราจารย์ ดร. วาทิต เบญจพลกุล

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

.....คณบดีคณะวิศวกรรมศาสตร์
(ศาสตราจารย์ ดร. สมศักดิ์ ปัญญาแก้ว)

คณะกรรมการสอบวิทยานิพนธ์

.....ประธานกรรมการ
(ศาสตราจารย์ ดร. ประสิทธิ์ ประพิณมงคลการ)

.....อาจารย์ที่ปรึกษา
(รองศาสตราจารย์ ดร. วาทิต เบญจพลกุล)

.....กรรมการ
(ดร. ลัญฉกร วุฒิสิทธิกุลกิจ)

สุชัย โรจนวิไลกุล : การจัดเส้นทางแบบมัลติคาสต์บนโครงข่ายเอทีเอ็มโดยใช้ฮิวริสติกอัลกอริทึม (MULTICAST ROUTING OVER AN ATM NETWORK USING A HEURISTIC ALGORITHM) อ.ที่ปรึกษา : รศ. ดร.วาทิต เบญจพลกุล , 85 หน้า. ISBN 974-13-0198-7

วิทยานิพนธ์ฉบับนี้มีจุดมุ่งหมายเพื่อเสนออัลกอริทึมการจัดเส้นทางแบบมัลติคาสต์บนโครงข่ายเอทีเอ็มโดยอาศัยแนวคิดของโครงข่ายเสมือน (virtual network), การจัดเส้นทางแบบ shortest path ซึ่งเป็นการจัดเส้นทางสำหรับการสื่อสารข้อมูลแบบจุดต่อจุด (point-to-point) ที่ใช้บนโครงข่ายเอทีเอ็ม และการนำแนวคิด Steiner Tree มาใช้ช่วยในการสร้างเส้นทางแบบมัลติคาสต์ อัลกอริทึมที่เสนอมี 2 ชุดคืออัลกอริทึม p1 และ p2 โดยที่ p2 ไม่ได้ใช้ Steiner node วิธีที่เสนอจะนำไปเปรียบเทียบกับอัลกอริทึมที่เสนอโดย Jia (1995) ที่เป็นอัลกอริทึมที่อยู่บนพื้นฐานของ spanning tree โดยมีฟังก์ชันวัตถุประสงค์ประกอบด้วยต้นทุนของแบนด์วิดท์ ต้นทุนของการต่อถึงกัน และต้นทุนของการสวิตช์ VP และ VC

ผลการจำลองแบบแสดงให้เห็นว่าอัลกอริทึม p1 ให้ค่าต้นทุนของแบนดวิดท์ต่ำกว่าอัลกอริทึมของ Jia 0.35% -11.31% และต้นทุนการสวิตช์ VP ลดลง 13.51% - 45.16% ขณะเดียวกันต้นทุนการต่อถึงกัน เพิ่มขึ้น 0.09% - 24% และต้นทุนของการสวิตช์ VC เพิ่มขึ้น 0.09% - 20% ซึ่งช่วงของการเปลี่ยนแปลงต้นทุนเหล่านี้มีความสัมพันธ์กับจำนวน Steiner Node ที่ใช้ในแต่ละช่วง สำหรับอัลกอริทึม p2 ให้ต้นทุนใกล้เคียงกับต้นทุนที่ได้จากอัลกอริทึมของ Jia แต่ใช้เวลาในการคำนวณเร็วกว่า เมื่อเปรียบเทียบค่าความยาววิถีและเวลาที่ใช้ในการคำนวณ พบว่าอัลกอริทึมที่เสนอใหม่มีความยาววิถีสั้นกว่าและใช้เวลาในการคำนวณน้อยกว่าอัลกอริทึมของ Jia นอกจากนี้ได้ทำการปรับแต่งอัลกอริทึมที่เสนอใหม่ทั้งสองเพื่อไปใช้ในการทำ dynamic multicast พบว่าขนาดของโครงข่ายที่เหมาะสมสำหรับอัลกอริทึมที่ปรับแต่งแล้วควรจะต่ำกว่า 40 โนด

ภาควิชา.....ลายมือชื่อนิสิต.....

ปีการศึกษา.....ลายมือชื่ออาจารย์ที่ปรึกษา.....

4070465821 ELECTRICAL ENGINEERING
MULTICAST ROUTING / ATM / STEINER TREE
SUCHAI ROJNAWILIKUL : MULTICAST ROUTING OVER ATM NETWORK USING A HEURISTIC ALGORITHM. THESIS ADVISOR : ASSOC. PROF. DR.WATIT BENJAPOLAKUL. 85 pp. ISBN 974 - 13 - 0198 - 7.

This thesis proposes two heuristic algorithms for solving multicast routing over ATM network. Using virtual network concept and shortest path routing which are originally used in point-to-point ATM network. We adapted Steiner Tree to find routes for multicast. The proposed algorithms p1 and p2 are compared with Jia (1995) algorithm. The objective function used for this work consists of cost of bandwidth, cost of connection establishment and cost of VP and VC switching.

The simulation results show that the proposed algorithm p1 uses bandwidth less than Jia algorithms within 0.35 % - 11.31 % range and saves cost of VP switching in the range of 13.51 % - 45.16 %. While its cost of connection establishment has grown up to 0.09 % - 24 % and cost of VC switching is also increased within 0.09 % - 20 %. It is found that the changing intervals of such costs are relevant to the changing of number of Steiner nodes used. p2 algorithm gives comparable costs with Jia algorithm. When comparing path length and running time of all investigated algorithms, p1 and p2 algorithms take shorter path length and faster running time than Jia algorithm. Moreover, the modification of two proposed algorithms for using in dynamic multicast situation can be applied in the network which has the size of less than 40 nodes.

Department..... Student's signature.....

Academic year..... Advisor's signature.....

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงไปด้วยดี ด้วยความช่วยเหลืออย่างดียิ่งของรองศาสตราจารย์ ดร. วาทิต เบญจพลกุล อาจารย์ที่ปรึกษาวิทยานิพนธ์ ซึ่งท่านได้ให้คำแนะนำและข้อคิดเห็นต่าง ๆ ที่เป็นประโยชน์ต่อการวิจัยด้วยดีตลอดมา ขอขอบคุณ ผู้ช่วยศาสตราจารย์ ดร. ทับทิม อ่างแก้ว ที่ได้กรุณาให้การสนับสนุนทรัพยากรสำหรับการประมวลผล ขอขอบคุณคุณสุรเชษฐ์ สุขเจริญ และเพื่อน ๆ นิสิต สาขาระบบโทรคมนาคมทุก ๆ ท่าน ที่ได้กรุณาแลกเปลี่ยนความรู้และให้ความช่วยเหลือมาตลอด

สุดท้ายนี้ ผู้วิจัยใครขอกราบขอบพระคุณบิดา มารดา และผู้มีพระคุณทุกท่าน ซึ่งให้การสนับสนุนและให้กำลังใจแก่ผู้วิจัยเสมอมาจนสำเร็จการศึกษา

สุชัย โรจนวิไลกุล

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ(ต่อ)

หน้า

สารบัญรูป

หน้า

สารบัญรูป (ต่อ)

หน้า

สารบัญรูป (ต่อ)

หน้า

สารบัญรูป (ต่อ)

หน้า

สารบัญตาราง

หน้า

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

การจัดเส้นทางสำหรับส่งข้อมูลชนิดต่าง ๆ จากต้นทางหนึ่งไปยังปลายทางหลายจุดนั้นเป็นปัญหาที่มีความสำคัญ เนื่องมาจากปัญหาทรัพยากรโครงข่ายที่ใช้ทั่วไปมีอยู่อย่างจำกัด โดยวิธีดังกล่าวสามารถส่งชุดข้อมูลเพียงชุดเดียวแทนที่จะส่งข้อมูลเป็นจำนวนเท่ากับปลายทางที่จะส่งไป เป็นผลให้ทรัพยากรโครงข่ายเช่น แบนด์วิดท์ และเวลาที่ใช้ในการสร้างเส้นทางการต่อถึงกันลดลงอย่างมาก ทำให้โครงข่ายมีความสามารถที่จะรองรับการประยุกต์ใช้งานในลักษณะดังกล่าวในจำนวนผู้ใช้ปริมาณมากได้

โครงข่าย ATM (Asynchronous Transfer Mode Network) เป็นโครงข่ายการส่งข้อมูลในลักษณะที่มีการสร้างการต่อถึงกันระหว่างโนดต้นทางและปลายทางก่อนแล้วจึงส่งข้อมูล (connection-oriented) และสามารถรับประกันคุณภาพของบริการ (Qualtiy of Service, QOS) ได้หลายระดับ มีการนำไปใช้งานทั้งในระดับ Local Area Network (LAN) และ Wide Area Network (WAN) โดยเฉพาะอย่างยิ่งในโครงข่ายที่เป็นโครงข่ายหลัก (backbone) อย่างไรก็ตามในการส่งข้อมูลที่ใช้แบนด์วิดท์จำนวนมาก (bandwidth intensive) เช่น วิดีโอออนดีมานต์ ในกรณีที่มีผู้รับที่ปลายทางจำนวนมากและใช้การส่งข้อมูลในลักษณะจากจุดถึงจุด (point-to-point) นำไปสู่การใช้งานแบนด์วิดท์และทรัพยากรอื่น ๆ อย่างไม่มีประสิทธิภาพ ดังนั้นจึงมีความพยายามในการแก้ปัญหาการจัดเส้นทางแบบมัลติคาสต์บนโครงข่าย ATM โดยค่าที่ใช้ในการเปรียบเทียบการทำงานของอัลกอริทึมแบบต่าง ๆ คือ ค่าต้นทุนที่ใช้ในการต่อถึงกันแบบมัลติคาสต์, ค่าต้นทุนของแบนด์วิดท์ที่ใช้ และค่าต้นทุนในการสวิตช์ข้อมูลของโครงข่าย ATM

1.2 วัตถุประสงค์

เพื่อปรับปรุงวิธีการจัดเส้นทางแบบมัลติคาสต์ (Multicast Routing) บนโครงข่ายเอทีเอ็ม โดยให้ฟังก์ชันวัตถุประสงค์ประกอบด้วยต้นทุนของการสร้างการต่อถึงกัน (cost of establishing connection, C_E), ต้นทุนของแบนด์วิดท์ (cost of bandwidth, C_B) และต้นทุนของการสวิตช์ (cost of switchings, C_S) และพิจารณาการทำงานที่แตกต่างกันของ VP switch และ VC switch ในการจัดเส้นทาง

1.3 เป้าหมายและขอบเขตของวิทยานิพนธ์

ในการจำลองแบบจะเปรียบเทียบผลของต้นทุนในการสร้างการต่อถึงกัน C_E, ต้นทุนของแบนด์วิดท์ C_B, และต้นทุนของการสวิตช์ C_S เมื่อใช้วิธีการจัดเส้นทางของ Jia et al [1] และวิธีการที่นำเสนอใหม่

1.4 ขั้นตอนและวิธีดำเนินการ

1. ศึกษาความรู้พื้นฐานโครงข่ายเอทีเอ็ม
2. ศึกษาการจัดเส้นทางแบบมัลติคาสต์
3. ศึกษาการแก้ปัญหาด้วย Steiner Tree
4. ปรับปรุงวิธีการที่ใช้แก้ปัญหาการจัดเส้นทาง
5. เขียนโปรแกรมจำลองแบบ
6. ทดสอบโปรแกรมและทำการจำลองแบบ
7. ประเมินผลและสรุป
8. เขียนวิทยานิพนธ์

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1. ได้วิธีการจัดเส้นทางแบบมัลติคาสต์บนโครงข่ายเอทีเอ็มที่ประหยัดทรัพยากรโครงข่ายอันได้แก่ แบนด์วิดท์ และต้นทุนของการสวิตช์ ดีกว่าหรือเท่ากับวิธีที่ได้เสนอก่อนหน้า [1] โดยอาศัยอัลกอริทึมการจัดเส้นทางแบบ point-to-point ที่ใช้อยู่เดิมในการจัดเส้นทางของโครงข่าย ATM เป็นพื้นฐาน

2. เป็นแนวทางในการออกแบบอัลกอริทึมการจัดเส้นทางแบบมัลติคาสต์บนโครงข่ายเอทีเอ็มสำหรับกรณีที่เป็นการจัดเส้นทางมัลติคาสต์แบบพลวัต (dynamic multicast routing)ที่มีลักษณะที่ไม่รบกวนการต่อถึงกันที่เกิดขึ้นอยู่ก่อน

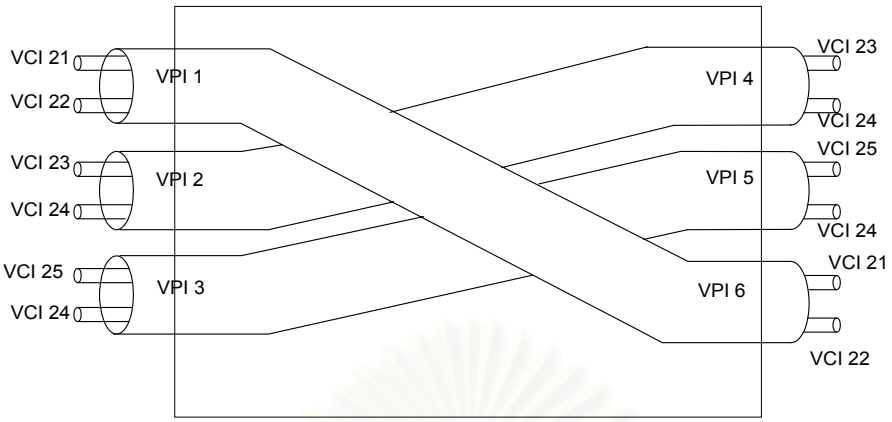
บทที่ 2

แนวคิด ผลงานที่ผ่านมา และการจัดเส้นทางแบบมัลติคาสต์บนโครงข่าย ATM

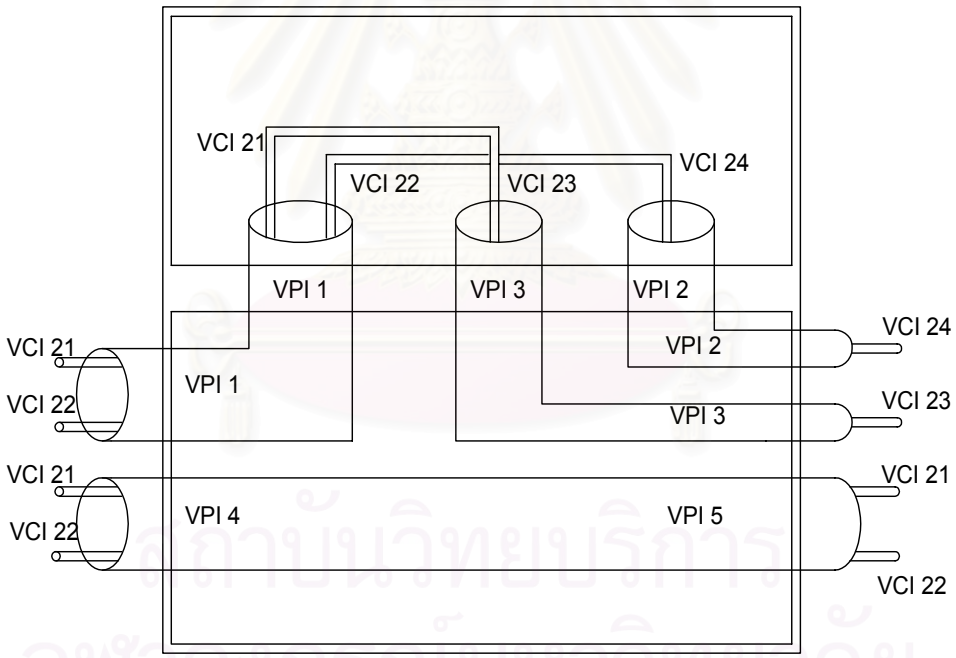
2.1 โครงข่าย ATM

โครงข่าย ATM ประกอบไปด้วย ATM switches และข่ายเชื่อมโยงกายภาพ (physical link) ATM เป็น protocol การสื่อสารแบบ connection-oriented แพ็กเกตข้อมูล (cells) ของ ATM จะถูกส่งผ่านช่องสัญญาณเสมือน (Virtual Channels, VCs) VC จะถูกสร้างขึ้นเมื่อมีการร้องขอการต่อถึงกันระหว่างปลาย (end-points) ทั้ง 2 ด้านเกิดขึ้น โดยที่ VCsจะถูกส่งผ่านไปในวิถีเสมือน (Virtual Paths, VPs) VC หนึ่ง ๆ สามารถส่งผ่าน cell ข้อมูลผ่าน VP เพียงชุดเดียวแล้วถึงปลายทาง หรืออาจส่งผ่าน VP หลายชุดที่ต่อถึงกัน (Virtual Path Connection, VPC, VPs) กว่าจะถึงปลายทางก็ได้ และ VC หลาย ๆ ชุดสามารถที่จะใช้ VP (หรือ VPC) ร่วมกัน เพื่อที่จะลดเวลาที่ใช้สร้างการต่อถึงกันของ VC และลดเวลาที่ใช้ในการสวิตช์ (switching time) ระหว่างทำการส่งผ่านข้อมูล ทั้งนี้รูปแบบการส่ง cell ข้อมูลของ ATM ผ่าน VP และ VC อาจอยู่ในลักษณะการส่งผ่าน VP หลายชุดต่อกัน (concatenated) ที่เรียกว่า VPC หรือ VPs และในลักษณะของการส่งผ่าน VC หลายชุดต่อกันที่เรียกว่า VCC (Virtual Channel Connection, VCs) ไปใน VPC หลายชุดต่อกัน

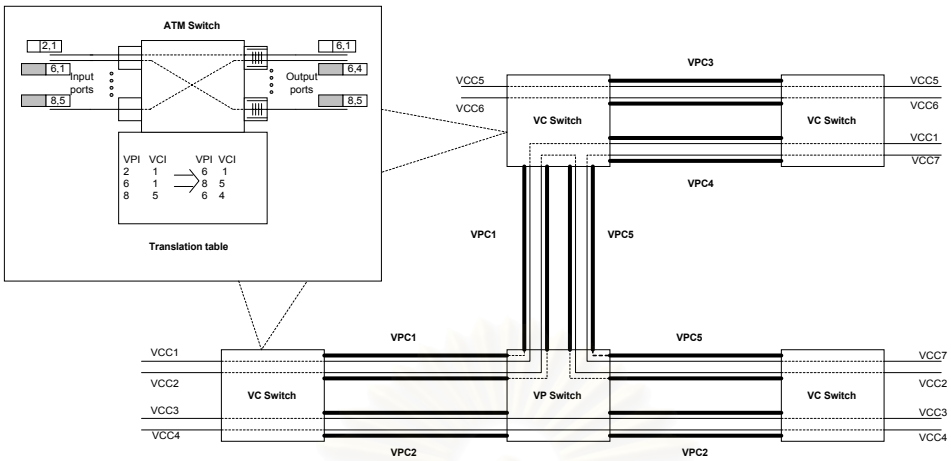
ATM switch ในโครงข่าย ATM จะประกอบด้วยสวิตช์ 2 ประเภทคือ VP switch และ VC switch แสดงดังรูปที่ 2.1 และรูปที่ 2.2 VP switch สามารถที่จะคงค่าหรือเปลี่ยนค่า VP Identifier (VPI) ในส่วน header ของ cell ซึ่งจะทำให้ cell ถูกส่งผ่านไปยัง VP ชุดเดิมหรือชุดใหม่ แต่ยังคงรักษาค่าของ VC Identifier (VCI) ไว้ไม่เปลี่ยนแปลงทำให้ cell ยังคงถูกส่งผ่าน VC ชุดเดิม ในส่วนของ VC switch สามารถเป็นจุดสิ้นสุดของ VPC (VPs), ทำการจัดกลุ่ม VCs รวมถึงอาจกำหนดค่า VCI, VPI ใหม่ให้กับ cell ที่ออกจากสวิตช์ ดังนั้น VP จะต้องเริ่มสร้างจาก VC switch และไปสิ้นสุดยัง VC switch เท่านั้น ส่วน VP switch เป็นโนดระหว่างทางในการส่ง cell ข้อมูล ดังแสดงในรูปที่ 2.3 นอกจากนี้จะเห็นว่า VPC ชุดหนึ่งสามารถรองรับ VCC ได้หลายชุด



รูปที่ 2.1 VP switch [2]



รูปที่ 2.2 VC switch [2]



รูปที่ 2.3 ATM switch และตัวอย่างของการใช้งาน [2]

2.2 **การจัดเส้นทางแบบมัลติคาสต์** (Multicast Routing)

การส่งข้อมูลแบบมัลติคาสต์เป็นความสามารถของระบบโครงข่ายสื่อสารที่จะรับสำเนาข้อมูลเพียง 1 สำเนา แล้วทำการส่งสำเนาข้อมูลนั้นออกไปหลาย ๆ สำเนายังกลุ่มผู้รับปลายทาง ซึ่งวิธีนี้จะช่วยให้ประหยัดแบนด์วิดท์มากกว่าการส่งข้อมูลที่เหมือนกันเป็นจำนวนสำเนาเท่ากับจำนวนสมาชิกของผู้รับปลายทางดังที่พบในการส่งข้อมูลที่ต่อถึงกันแบบจุดถึงจุด (point-to-point connections) เนื่องจากรูปแบบการส่งข้อมูลที่ประหยัดจำนวนเสำเนาของข้อมูลมากที่สุดคือการส่งบนเส้นทางที่มีลักษณะเป็นต้นไม้ (Tree) [3] ถ้าเราสร้างแบบจำลองให้โครงข่ายสื่อสารแทนด้วยกราฟ G(V, E)เมื่อ V คือเซตของโนดหรือสวิตช์ และ E คือเซตของข่ายเชื่อมโยง (links) การจัดเส้นทางแบบมัลติคาสต์สามารถเทียบกับการค้นหาต้นไม้ T จากกราฟ G ที่ซึ่งทอดข้าม (span) ทุก ๆ โนดในกลุ่มมัลติคาสต์ M (M คือกลุ่มมัลติคาสต์ที่ประกอบด้วยผู้ส่งและผู้รับ) เข้าด้วยกัน ต้นไม้นั้นถูกเรียกว่าต้นไม้ของการส่งข้อมูลแบบมัลติคาสต์ (multicast tree)

ปัญหาการทำให้เหมาะที่สุด (optimization) ของ multicast tree สามารถพิจารณาเป็นปัญหา Steiner tree Problem in Networks (SPN) และมีนิยามดังนี้ กำหนดให้กราฟ G=(V, E) เป็นกราฟที่ไม่มีทิศทาง (undirected) และมีฟังก์ชันต้นทุน (cost function) ที่คำนวณต้นทุนจากต้นทุนของข่ายเชื่อมโยงโนด u และ v ที่เป็นค่าจริงบวก และเซตของ $M \subseteq V$ ที่เป็นเซตของกลุ่มมัลติคาสต์ ทำการค้นหาต้นไม้ $T = (V_T, E_T)$ ที่เชื่อมโยงสมาชิกใน M และทำให้ต้นทุนของการเชื่อมต่อ $C_T=\sum_{(u,v)\in ET} C_{uv}$ มีค่าน้อยที่สุดต้นไม้ที่ได้เรียกว่าต้นไม้แบบสไตเนอร์ (Steiner Tree) และให้โนดใด ๆ ที่ไม่เป็นสมาชิกของ M แต่เป็นสมาชิกของ G และมี link เชื่อมโยงโนดนั้นมากกว่า 2 ขึ้นไปถูกเรียกว่า Steiner node

แม้ว่าปัญหา SPN เป็นปัญหา NP-complete [4] [5] แต่ยังมีบางกรณีที่สามารถหาคำตอบได้ภายในเวลาที่เป็นโพลีโนเมียล (polynomial time) ได้แก่

กรณีที่ $|M| = 2$ เป็นการส่งข้อมูลแบบ unicast จะเป็นการคำนวณหา shortest path

และกรณี $|M| = |V|$ เป็นการส่งข้อมูลแบบ broadcast จะเป็นการคำนวณหา spanning tree

ดังนั้นในทางปฏิบัติสมควรหาฮิวริสติกอัลกอริทึมในการประมาณคำตอบของปัญหา SPN ซึ่งจะทำให้ได้คำตอบใกล้เคียงจุดที่เหมาะที่สุดภายในเวลาโพลีโนเมียล

สำหรับการจัดเส้นทางในกรณีที่เป็น Dynamic Multicast ในสภาวะใช้งานจริงการส่งข้อมูลในรูปแบบมัลติคาสต์ควรจะสามารถรองรับการร้องขอการต่อถึงกันเข้ากับกลุ่มมัลติคาสต์เดิมหรือการร้องขอออกจากกลุ่ม เอกสาร [4] และ [6] ชี้ให้เห็นว่า อัลกอริทึมการจัดเส้นทางในสภาวะ dynamic multicast ที่ดีควรจะรับประกันว่าในกรณีที่มีการร้องขอการต่อถึงกันหรือออกจากกลุ่มต้องไม่ก่อให้เกิดผลกระทบต่อเนื่องในวงกว้างของการเปลี่ยนแปลง routing table ของโนดต่าง ๆ ในโครงข่าย กล่าวคือจะต้องไม่ส่งผลกระทบในส่วนของ multicast tree ที่ถูกใช้งานโดยสมาชิกมัลติคาสต์ส่วนที่เหลือ

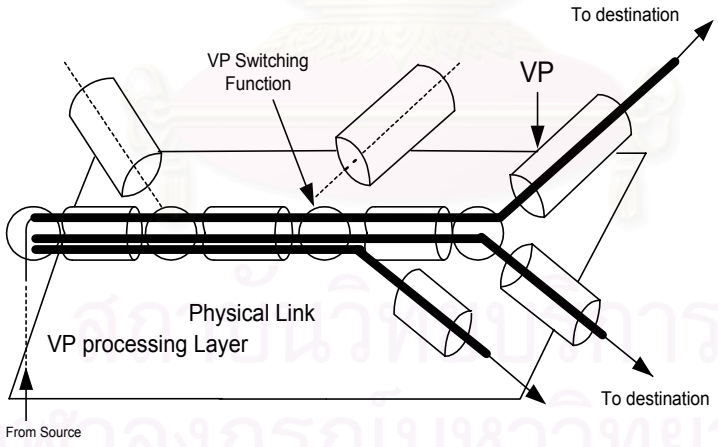
2.3 การจัดเส้นทางแบบมัลติคาสต์บนโครงข่าย ATM (Multicast Routing over ATM network)

วิธีการในการจัดเส้นทางบนโครงข่าย ATM สามารถแบ่งออกได้เป็น 3 วิธี

2.3.1 การจัดเส้นทางโดยใช้ Point-to-point VP

อาศัยพื้นฐานของการส่งข้อมูลแบบ unicast สามารถนำมาใช้กับการส่งข้อมูลแบบมัลติคาสต์ได้โดยใช้การซ้อนทับ (superposition) ของเส้นทางแบบ point-to-point ดังแสดงในรูปที่ 2.4

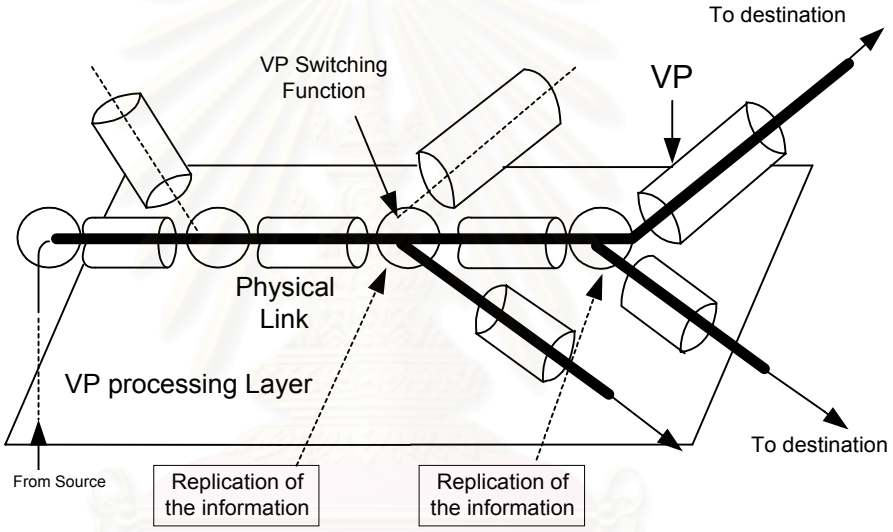
เมื่อมีการเรียก (call) เกิดขึ้น VCs หลาย ๆ ชุดจะถูกสร้างขึ้นพร้อมกับ VPs ที่ต่อแต่ละ source node และ destination node cell เข้าด้วยกัน ข้อมูลจะถูกทำสำเนาที่ source และถูกส่งไปยังแต่ละ VC บนเส้นทางที่จะทำการสื่อสารแบบมัลติคาสต์ โครงสร้างทาง hardware ของการทำมัลติคาสต์โดยอาศัยการส่ง unicast สามารถนำมาใช้ได้ง่ายเนื่องจากโครงข่าย ATM เป็นการสื่อสารในรูปแบบ point-to-point อยู่แล้ว แต่ในกรณีที่มีจำนวน destination node เพิ่มมากขึ้นจากรูปที่ 2.4 จะเห็นว่าจะมีข้อมูลชุดเดียวกันส่งผ่าน physical link ชุดเดียวกันมากขึ้นทำให้ระบบเกิดการคับคั่งของทราฟฟิกได้ง่าย



รูปที่ 2.4 การส่งข้อมูลแบบมัลติคาสต์โดยใช้การสื่อสารแบบ point-to-point VP [7]

2.3.2 VP multicasting

วิธีนี้เมื่อมีการเรียกแบบมัลติคาสต์ถูกสร้างขึ้นมา VP ในรูปแบบต้นไม้ (Tree-shaped VP, T-VP) จะถูกสร้างขึ้นตามแอดเดรสของ source และ destination node หรือในอีกกรณีหนึ่ง ก่อนที่จะมีการเรียกเกิดขึ้นจะทำการสร้าง T-VP ล่วงหน้าสำหรับชุดของ source และ destination หนึ่ง ๆ ข้อมูลจะถูกสำเนาโดย switching node ที่เป็น VP switch (หรือ ATM cross-connect) ที่เป็น branch point โดยพิจารณาจากหมายเลข VPI ดังแสดงในรูปที่ 2.5 วิธีนี้ข้อมูลชุดเดียวกันจะไม่วิ่งผ่าน physical link เดียวกันเหมือนอย่างวิธี point-to-point VP

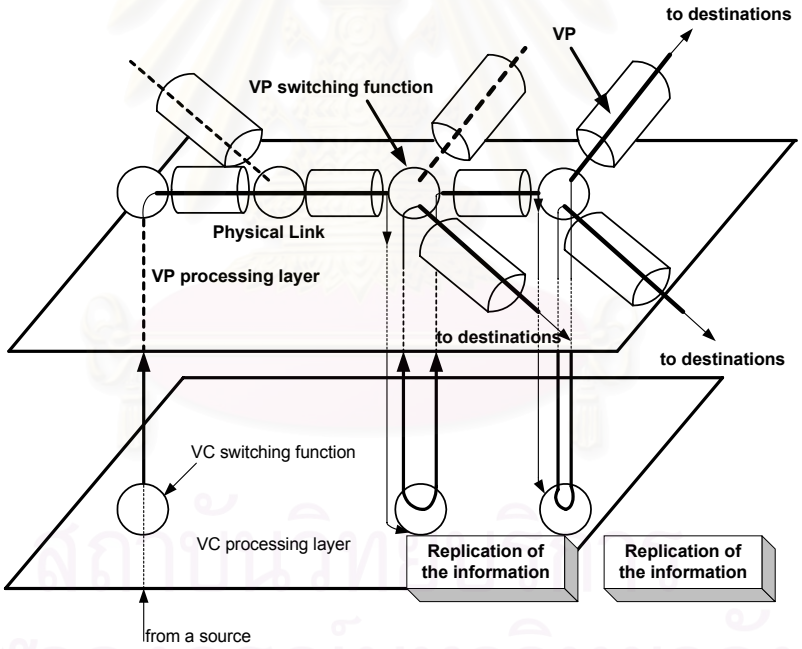


รูปที่ 2.5 การส่งข้อมูลแบบมัลติคาสต์โดยวิธี VP multicasting [7]

ปัญหาที่เกิดกับวิธี VP multicasting คือการจองหมายเลข VPI จำนวนมากเพื่อที่จะสร้าง multicast tree ของ VP ก่อนที่จะมีการเรียกแบบมัลติคาสต์เกิดขึ้น ซึ่งการจองหมายเลข VPI นี้จะจองไว้สำหรับทุก ๆ combination ของ source และ destination node โดยเป็นชุด VPI ที่จองแยกจาก VPI ที่ใช้สำหรับการส่งข้อมูลแบบ point-to-point ตามปกติ เมื่อพิจารณาถึงจำนวน VPI ที่มีอยู่อย่างจำกัด โครงสร้าง multicast tree แบบนี้จึงไม่เหมาะสมสำหรับรองรับ destination node จำนวนมาก ๆ

2.3.3 VC multicasting

วิธีนี้สร้าง multicast tree ในระดับของ VC โดยการผสม point-to-point VP เข้าด้วยกันหลาย ๆ hop เมื่อมีการเรียกแบบมัลติคาสต์เกิดขึ้น หรือเรียกว่าวิธี combined VP (C-VP) เส้นทางส่งข้อมูลแบบมัลติคาสต์ถูกสร้างขึ้นโดยการต่อกันของ VPs หลาย ๆ ชุด จากนั้น cell ข้อมูลที่มาจาก source node จะถูกส่งไปยัง VP และ VC processing layer (ซึ่งก็คือ VC swtich) ที่อยู่ตรงจุดกิ่ง (branch point) ของ point-to-point VPs หลังจากสำเนาข้อมูลที่ VC processing layer แล้ว cell ที่ได้จากการทำสำเนาจะถูกส่งขึ้นไปยัง VP processing layer และถูกส่งไปยัง VP ที่ต้องการ ดังนั้นในกรณีนี้ VP switch หรือ cross-connect switch ไม่สามารถทำสำเนาข้อมูลได้ จึงทำให้ประหยัดหมายเลข VPI เนื่องจากไม่ต้องสร้าง T-VP เฉพาะสำหรับทราฟฟิกแบบมัลติคาสต์ ดังแสดงในรูปที่ 2.6



รูปที่ 2.6 การส่งข้อมูลแบบมัลติคาสต์โดยวิธี VC multicasting [7]

ในการจัดเส้นทางแบบมัลติคาสต์บนโครงข่าย ATM นั้นจะกำหนดให้โครงข่าย ATM แทนด้วยกราฟ G(V, E) ให้ source node s $\in$ V และเซตของ destination nodes D $\subseteq$ V ทั้ง s และ D เป็น VC switches multicast tree ที่ใช้จัดเส้นทางเป็น subtree ของ G(V, E) ซึ่งมี root ที่ s และประกอบด้วยทุกสมาชิกใน D และอาจมีบางสมาชิกใน V - D และมีฟังก์ชันวัตถุประสงค์ดังนี้

$$C= \alpha C_E+\beta C_B+\gamma C_S \qquad (2.1)$$

α, β, γ เป็นค่า weight ของ C_E, C_B และ C_S

C คือ ต้นทุนทั้งหมดของการจัดเส้นทางแบบมัลติคาสต์

C_E คือ จำนวนทั้งหมดของ VP ที่สร้างการต่อถึงกัน (connection) จาก source ไปยัง destination ถ้ามี VC หลายชุดใช้ VP ร่วมกัน การคำนวณจะนับ VP นั้นเพียงครั้งเดียว

C_B คือ จำนวนแบนด์วิดท์ทั้งหมดจาก source ไปยังทุก ๆ destination แทนด้วยผลรวมของจำนวนข่ายเชื่อมโยง (link) ทั้งหมดจาก source ไปยัง destination ถ้ามี VC หลายชุดใช้ VP ร่วมกัน จะคำนวณต้นทุนแบนด์วิดท์ของ VP นั้นครั้งเดียว

C_S คือ จำนวน VP switching และ VC switching จาก source ไปยัง destination ถ้ามี VC หลายชุดใช้ VP ร่วมกัน จะคำนวณ switching บน VP นั้นเพียงครั้งเดียว

โดยพิจารณาปัญหาการทำให้เหมาะที่สุดนี้เป็นปัญหาที่ไม่มีเงื่อนไขบังคับ (unconstrained optimization)

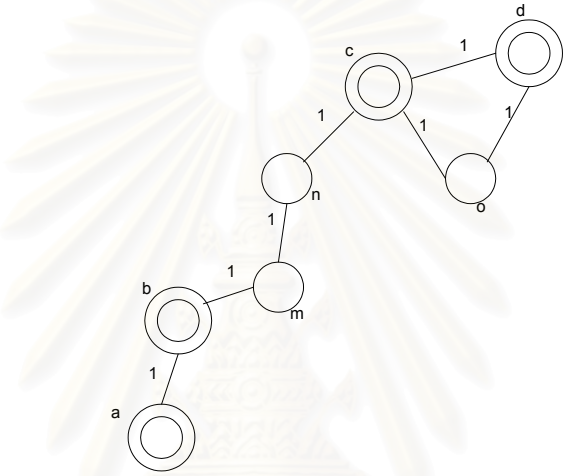
งานวิจัยที่ผ่านมาเช่น Sun et al. [6] และ Ammar et al. [8] ทำการจัดเส้นทางแบบมัลติคาสต์บนโครงข่าย ATM บนพื้นฐานของวิถีเสมือน (VP) แต่ไม่ได้พิจารณาความแตกต่างของการทำงานของ VP switch และ VC switch ทำให้ต้นทุนการจัดเส้นทางที่ได้ไม่ตรงกับความเป็นจริง เนื่องจากไม่คำนึงถึงจุดเริ่มต้นและจุดสิ้นสุดของ VP และการใช้ VP ร่วมกัน

Jia, et al. [1] ได้เสนออัลกอริทึมการจัดเส้นทางแบบมัลติคาสต์บนโครงข่าย ATM โดยพิจารณาถึงความแตกต่างของการทำงานของสวิตช์ทั้ง 2 แบบ และได้เสนอแนวคิดของ Virtual Network ซึ่งประกอบไปด้วย Virtual Link ที่ได้จากการทำการยุบ link ระหว่าง VC switches ใด ๆ บนโครงข่าย ATM เดิม และ VC switch ทุก ๆ โนดจากโครงข่ายเดิม โดยมีรายละเอียดการหา Virtual Network ดังนี้

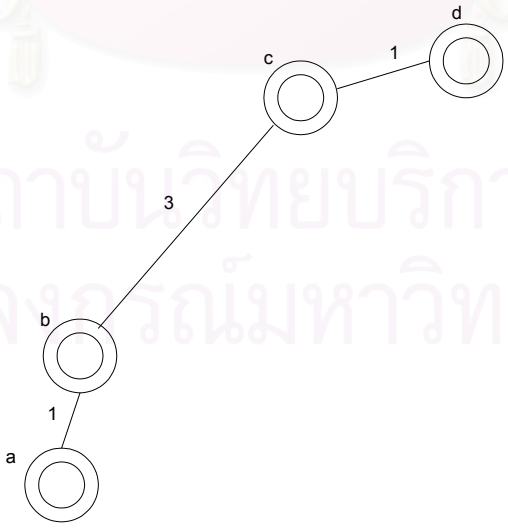
ขั้นตอนการยุบ VP switch ออกจากโครงข่าย ATM

1. ทำการลบ VP switch ที่อยู่ระหว่าง VC switch คู่ใด ๆ แล้วแทนที่ด้วย virtual link ที่เชื่อมโยงระหว่าง VC switch คู่นั้น ๆ
2. weight ของ virtual link เท่ากับผลรวมของจำนวน VP switch บวกหนึ่ง เมื่อกำหนดให้ทุก ๆ link บนโครงข่าย ATM เดิมมี weight เป็น 1

ผลการยุบ VP switch แสดงดังตัวอย่างรูปที่ 2.7



รูปที่ 2.7 a โครงข่ายตัวอย่างก่อนหา Virtual network



รูปที่ 2.7 b Virtual network ของโครงข่ายในรูปที่ 2.7 a

จากตัวอย่างจะพบว่าเนื่องจาก virtual link เริ่มต้นและสิ้นสุดที่ VC switch และโนดระหว่างทางเป็น VP switch ทั้งหมด VP หนึ่ง ๆ สามารถพิจารณาเป็น virtual link 1 link หรือ หลาย ๆ virtual link ต่อกัน ทำให้ virtual link หนึ่ง ๆ สามารถถูกใช้ร่วมกันได้ระหว่าง VCC หลายชุด

พิจารณาฟังก์ชันวัตถุประสงค์ใน 1 virtual link ของ virtual network

$$C(VL) = \alpha(1) + \beta(N_{vps}+1) + \gamma(T_{vp}*N_{vps}+2T_{vc})$$
$$= (\beta+\gamma T_{vp})N_{vps} + (\alpha+\beta+2\gamma T_{vc})$$
$$= KN_{vps} + H \qquad (2.2)$$

เมื่อ N_{vps} คือจำนวน VP switch ที่อยู่ระหว่าง VC switch คู่ใด ๆ

T_{vp} และ T_{vc} คือ เวลาที่ใช้ในการสวิตช์ของ VP swich และ VC switch

$K = (\beta+\gamma T_{vp})$

$H = (\alpha+\beta+2\gamma T_{vc})$

K และ H เป็นค่าคงที่สำหรับทุก ๆ link ใน virtual network และ C(VL) แปรผันตาม N_{vps} เพื่อที่จะเปรียบเทียบการจัดเส้นทางให้ได้ต้นทุน (cost) น้อยที่สุดระหว่างวิธีการต่าง ๆ กันจะพิจารณาเฉพาะต้นทุนสัมพัทธ์ (relative cost) แทนที่จะพิจารณาต้นทุนที่แท้จริง ดังนั้นจะใช้ค่า $N_{vps}+1$ เป็น weight ของ virtual link ดังเอกสาร [1] และทำให้ต้นทุนชนิดต่าง ๆ เป็นค่าที่ไม่มีหน่วย

$$C(VL) = N_{vps}+1 \qquad (2.3)$$

อัลกอริทึมการจัดเส้นทางแบบมัลติคาสต์ที่เสนอโดย Jia เป็นอัลกอริทึมที่อยู่บนพื้นฐานของ minimum spanning tree ของ Kruskal [9] ซึ่งต้องการการจัดเรียงลำดับของ link และต้องระวังการเลือกลำดับของ link ไม่ให้เกิดลูปใน multicast tree จึงไม่เหมาะกับการทำ dynamic multicast [4] ซึ่งการจัดเส้นทางแบบมัลติคาสต์สำหรับแต่ละ destination node ต้องไม่รบกวนการต่อถึงกันที่มีอยู่ก่อน ดังนั้นการสร้าง multicast tree โดยอาศัย partial tree จึงได้รับการสนับสนุนว่ามีความเหมาะสมสำหรับ การทำ dynamic multicast และเป็นคุณสมบัติที่อัลกอริทึมควรจะมี นอกจากนี้ Jia ยังไม่ได้พิจารณา Steiner node ซึ่งอยู่นอกเซต {s} U D ดังนั้นคำตอบที่ได้จึงยังไม่เข้าใกล้จุดที่เหมาะสมมากพอ [4] [5]

จากเหตุผลข้างต้นเนื่องจากอัลกอริทึมที่มีพื้นฐานจาก spanning tree มีลักษณะของการรบกวนการต่อของกลุ่มมัลติคาสต์ที่เกิดขึ้นอยู่ก่อน อีกทั้งโครงข่าย ATM ใช้วิธี shortest path ในการหาเส้นทางจากจุดถึงจุดโดยทั่วไป ดังนั้นจึงได้นำเสนออัลกอริทึมที่อาศัยการจัดเส้นทางสำหรับส่งข้อมูลจากจุดถึงจุดมาใช้ในการจัดเส้นทางเพื่อส่งข้อมูลในลักษณะจากจุดถึงหลายจุด (point - to - multipoint) โดยเป็นการรวมกันของการใช้ virtual network และการจัดเส้นทางแบบมัลติคาสต์โดยอาศัย shortest path อัลกอริทึมที่นำเสนอแบ่งเป็น 2 ชุดสำหรับกรณี static multicast และอีก 2 ชุดสำหรับ dynamic multicast ที่แก้ไขมาจาก 2 ชุดแรก

วิทยานิพนธ์ฉบับนี้ทำการเปรียบเทียบอัลกอริทึมการจัดเส้นทางแบบมัลติคาสต์ที่เสนอโดย Jia [1] , อัลกอริทึมที่เสนอขึ้นใหม่ชุดที่ 1 (p1) และอัลกอริทึมที่เสนอขึ้นใหม่วิธีที่ 2 (p2) ในสภาวะที่เป็น static multicast และเปรียบเทียบอัลกอริทึมที่ดัดแปลงจาก p1 (p1 partial), อัลกอริทึมที่ดัดแปลงจาก p2 (p2 partial) กับ p1, p2 และอัลกอริทึมของ Jia ในสภาวะ dynamic multicast เพื่อเป็นแนวทางในการออกแบบอัลกอริทึม ในกรณีของ static multicast จะศึกษาผลของ C_E, C_B, C_S และเวลาที่ใช้ในการประมวลผล นอกจากนี้ยังพิจารณาความยาวของเส้นทางจากโนดต้นทางไปยัง
โนดปลายทางซึ่งมีผลต่อเวลาประวิง (delay) โดยวัดออกมาเป็นจำนวน hop ของข่ายเชื่อมโยงที่ใช้

รูปแบบการส่งข้อมูลมัลติคาสต์เป็นแบบ VC multicasting โดยสมมติ (assume) ให้มี VPs เริ่มต้นที่ source node และไปสิ้นสุดที่ branch points หรือสิ้นสุดที่ destination node และให้มี VPs ที่เริ่มต้นที่ branch points ใด ๆ ไปสิ้นสุดที่ branch points อื่นหรือ destination nodes

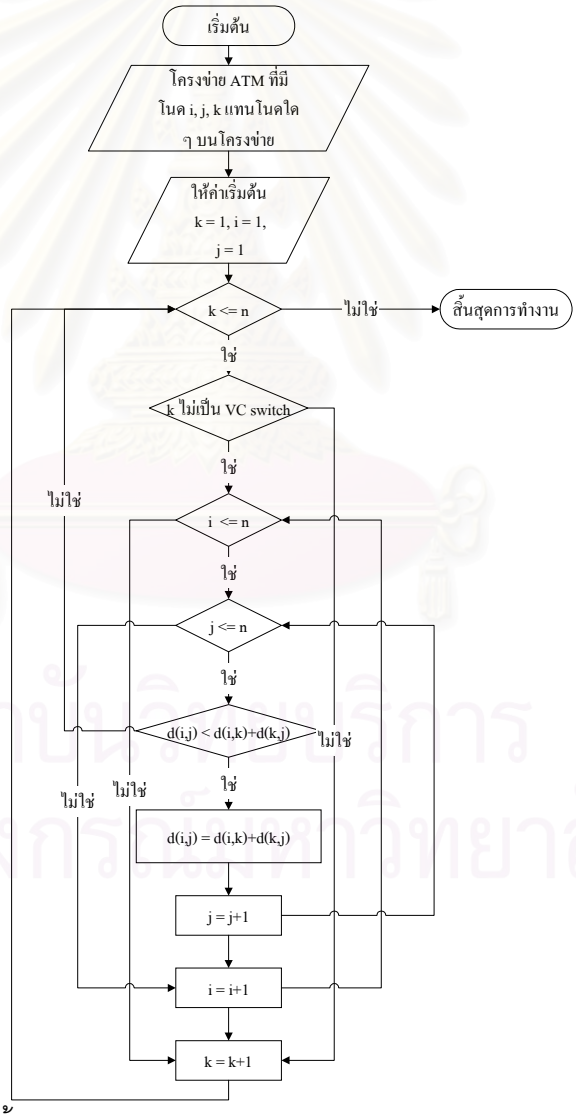
2.4 อัลกอริทึมในการจัดเส้นทางมัลติคาสต์บนโครงข่าย ATM

2.4.1 การหา virtual network

ในการคำนวณหา virtual network สามารถอาศัยวิธี dynamic programming [9] ดังนี้

$$D_k[i,j] = \min\left(D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\right) \text{ เมื่อ k} \notin \text{VC switches} \qquad (2.4)$$

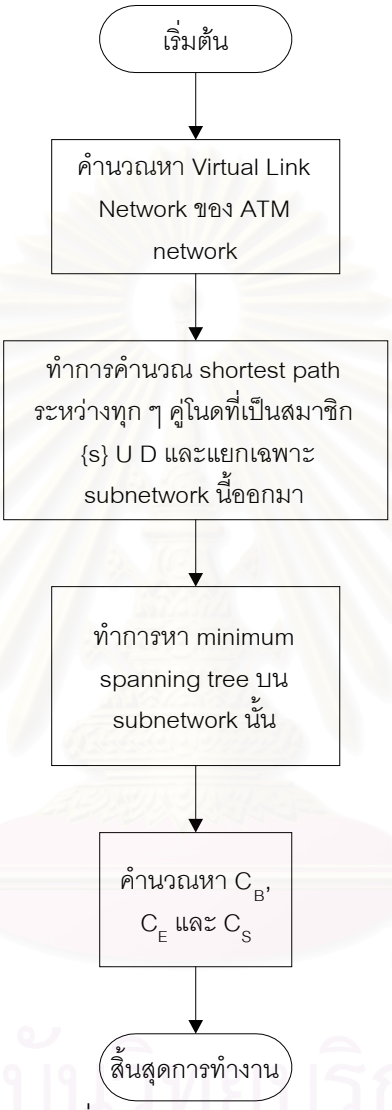
เมื่อ $D_k[i,j]$ เป็นเมตริกซ์ของระยะทางที่สั้นที่สุดระหว่างโนด i และ j สำหรับการวนรอบการทำงานที่ k แสดงขั้นตอนการหา virtual network ดังรูปที่ 2.8



รูปที่ 2.8 ขั้นตอนการหา virtual network

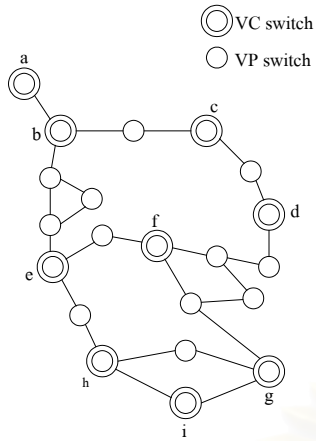
2.4.2 **อัลกอริทึมที่เสนอโดย** Jia

อัลกอริทึมที่เสนอโดย Jia มีขั้นตอนการทำงานดังรูปที่ 2.9

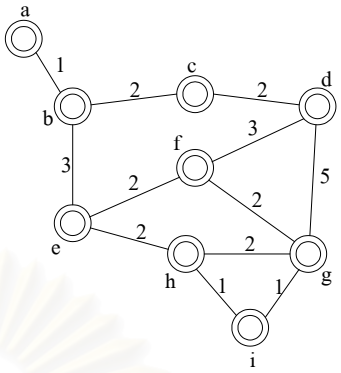


รูปที่ 2.9 Jia's algorithm

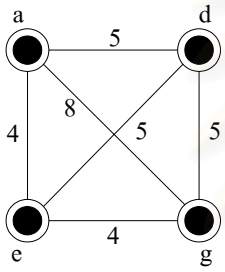
ซึ่งอธิบายได้ในรูปที่ 2.10 a, 2.10 b และ 2.10 c



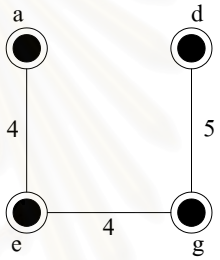
รูปที่ 2.10 a ATM network



รูปที่ 2.10 b virtual network



รูปที่ 2.10 c Induced network ของ {a, d, e, g}



รูปที่ 2.10 d ผลของ spanning tree

จากรูปที่ 2.10 d จะเห็นว่าต้นทุนของการต่อถึงกัน C_E จะเท่ากับ $|D|$ ขณะที่ต้นทุนของแบนด์วิดท์ (C_B) จะเท่ากับผลรวมของ weight บน link ที่ต่อระหว่างโนด u กับ v ใด ๆ (w(u,v)) ของเส้นทางที่เป็นผลลัพธ์ จากรูปที่ 2.10 d จะได้ $C_B = 13$ ต้นทุนของการสวิตช์ VC เท่ากับ 4 (เกิดขึ้นที่โนดต้นทางและปลายทาง) ส่วนต้นทุนของการสวิตช์ VP เท่ากับผลรวมของจำนวนโนดที่อยู่ระหว่างโนดที่เป็น end points ทั้งสองในรูปที่ 2.10 d ซึ่งก็คือ (4 - 1) + (4 - 1) + (5 - 1) =10

ต้นทุนการจัดเส้นทางของอัลกอริทึมของ Jia แสดงได้ดังนี้ [1]

$$C_E = |D| \quad (2.5)$$

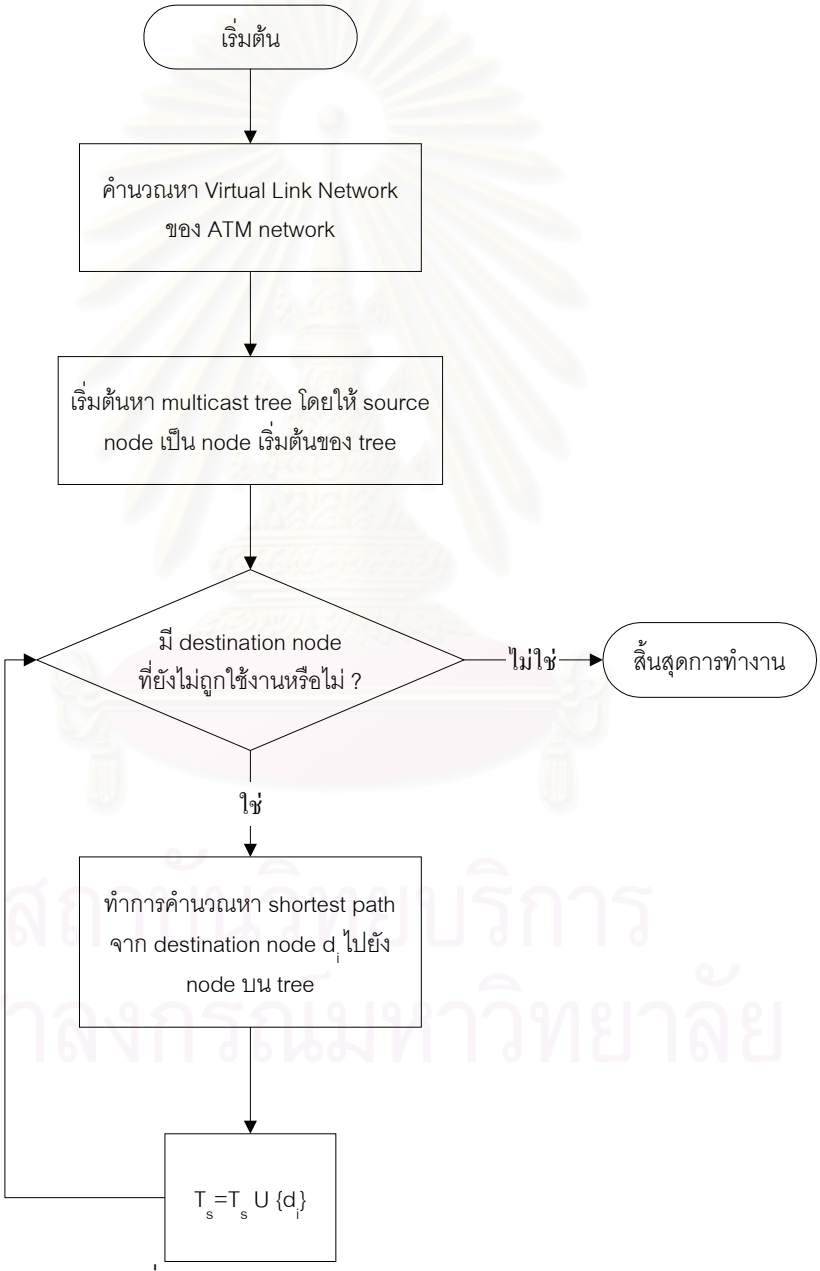
$$C_B = \sum_{(u,v)\in R} w(u,v) \quad (2.6)$$

$$C_S(\text{VC switching}) = |D|+1 \quad (2.7)$$

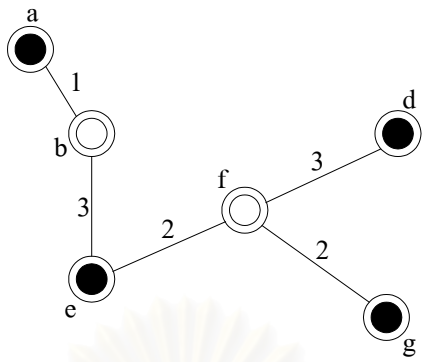
$$C_S(\text{VP switching}) = \sum_{(u,v)\in R} (w(u,v) -1) \quad (2.8)$$

2.4.3 **อัลกอริทึมที่เสนอวิธีที่** 1 (p1)

อัลกอริทึมที่เสนออยู่บนพื้นฐานของการคำนวณ shortest path ซึ่งเป็นวิธีการหาเส้นทางที่ใช้กันอยู่ในการส่งข้อมูลแบบ point-to-point และมีการนำ Steiner node เข้ามาเกี่ยวข้องด้วยทำให้ต้นทุนที่ได้มีพจน์ที่เป็นจำนวนของ Steiner node เข้ามารวมด้วย อัลกอริทึม p1 มีขั้นตอนการทำงานดังรูปที่ 2.11 และเมื่อนำไปใช้กับโครงข่ายในรูปที่ 2.10 a จะได้คำตอบดังรูปที่ 2.12



รูปที่ 2.11 p1 algorithm



รูปที่ 2.12 เส้นทางที่ได้จากอัลกอริทึม p1

ซึ่งมีต้นทุนของการจัดเส้นทางดังนี้

$$C_E = |D| + nS \quad (2.9)$$

$$C_B = \sum_{(u,v)\in R} w(u,v) \quad (2.10)$$

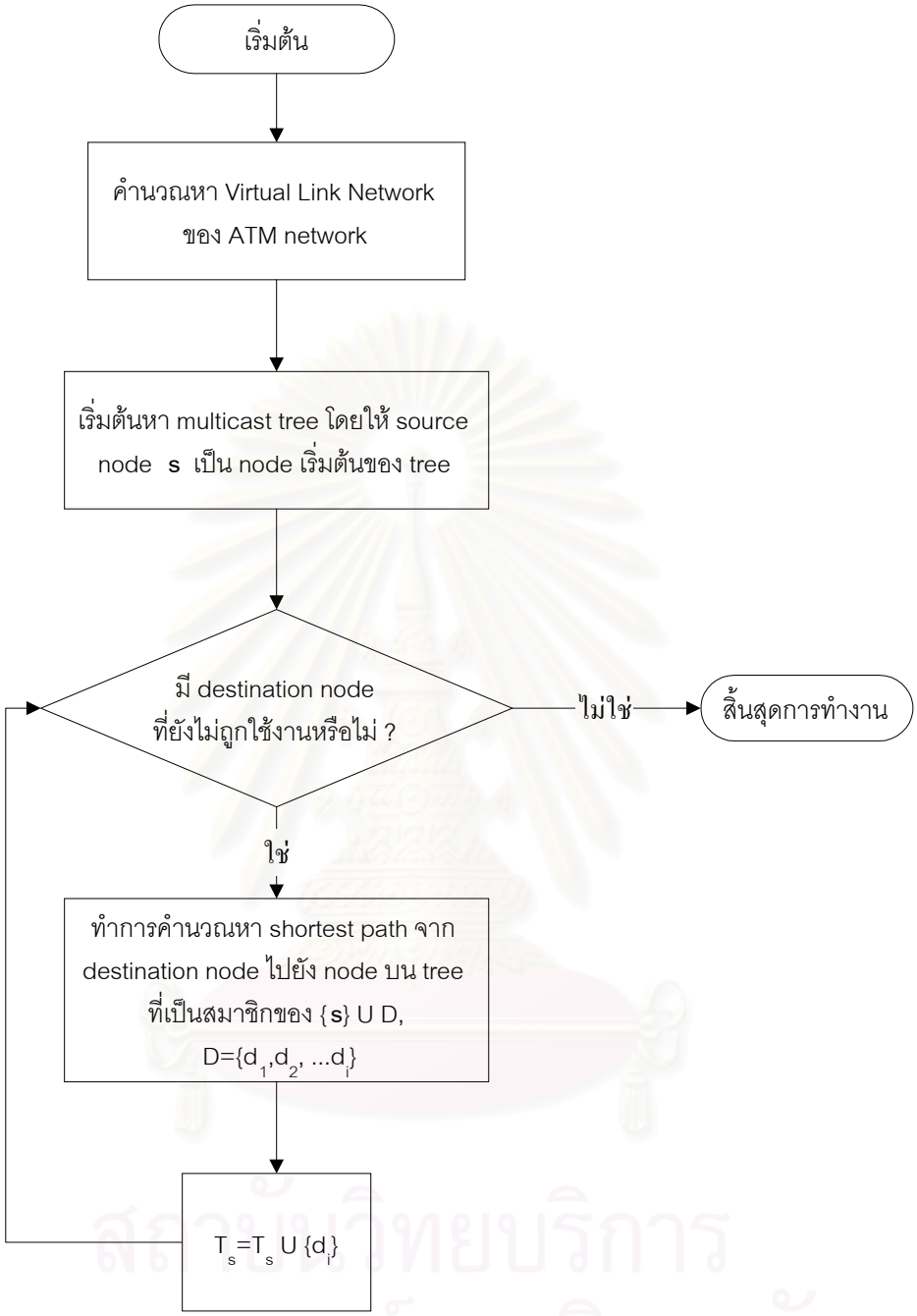
$$C_S(\text{VC switching}) = |D| + nS + 1 \quad (2.11)$$

$$C_S(\text{VP switching}) = \sum_{(u,v)\in R} (w(u,v)-1) \quad (2.12)$$

เมื่อ nS คือจำนวน Steiner node บน multicast tree

2.4.4 **อัลกอริทึมที่เสนอวิธีที่** 2 (p2)

เป็นอัลกอริทึมที่อาศัยหลักการ partial tree แต่พิจารณาจุดต่อถึงกันบน Tree เฉพาะโนดต้นปลายและกลุ่มของโนดปลายทาง ({s} U D) จึงไม่มีผลของ Steiner point เข้ามาเกี่ยวข้อง สมการคำนวณต้นทุนจึงเหมือนกับของ Jia ขั้นตอนการทำงานของอัลกอริทึม p2 แสดงดังรูปที่ 2.13



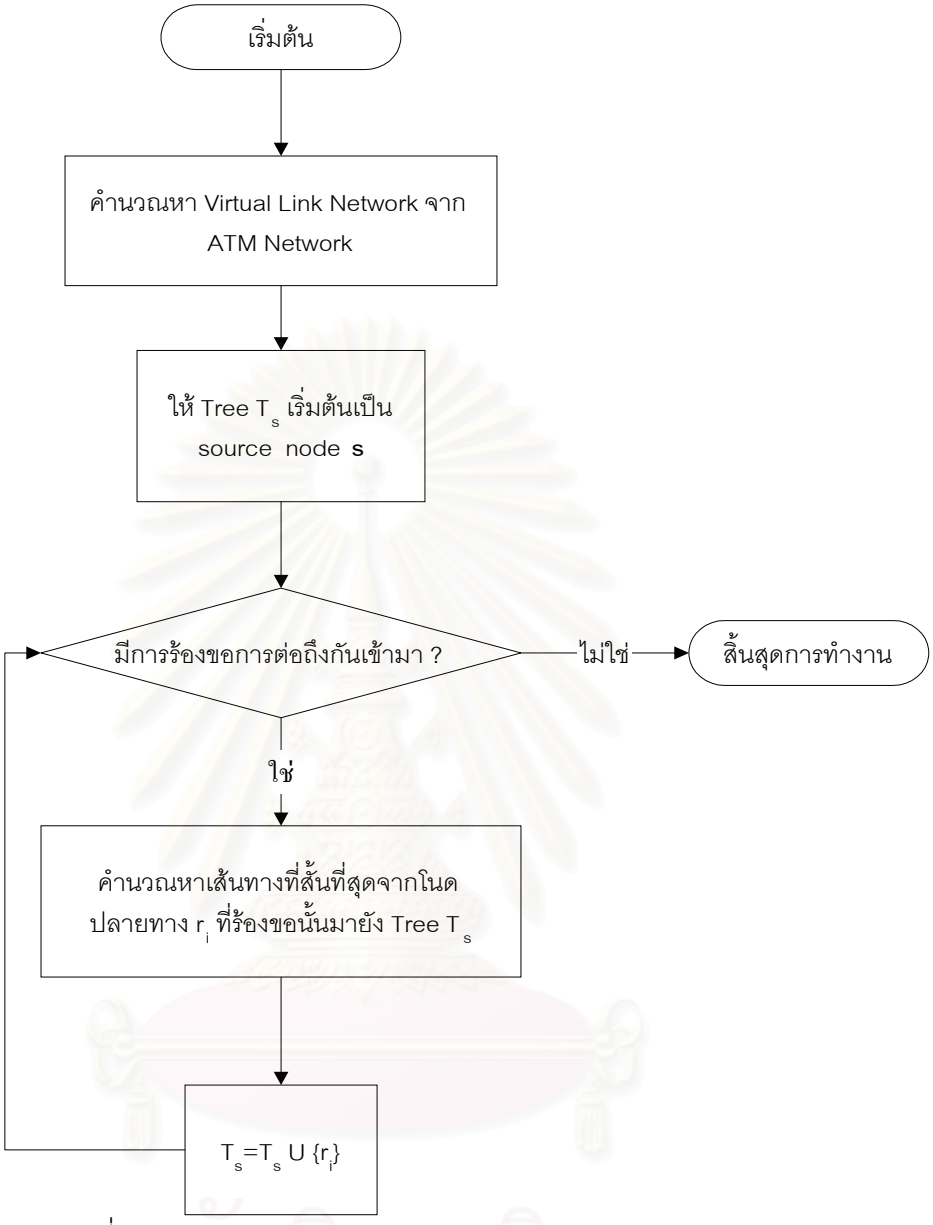
รูปที่ 2.13 p2 algorithm

2.4.5 **อัลกอริทึมที่ดัดแปลงจาก p1 สำหรับกรณี** dynamic multicast (p1 partial)

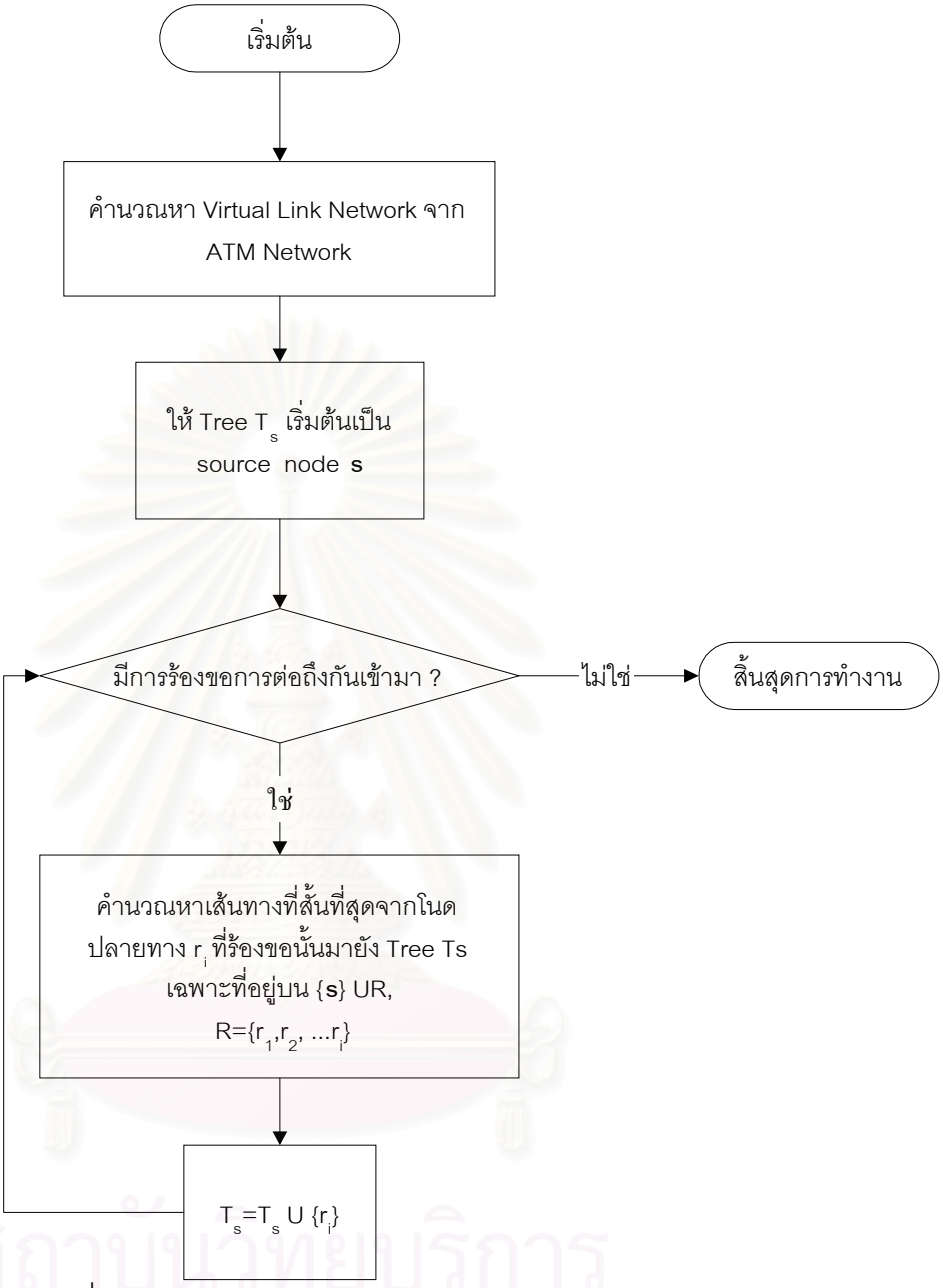
เป็นอัลกอริทึมที่ดัดแปลงจาก p1 เนื่องจากในภาวะ dynamic multicast ไม่สามารถเปรียบเทียบว่า destination node ใด ดีที่สุดที่จะต่อถึงกันก่อน ส่วนสมการต้นทุนการจัดเส้นทางเหมือนกับ p1 มีขั้นตอนการทำงานแสดงในรูปที่ 2.14

2.4.6 **อัลกอริทึมที่ดัดแปลงจาก p2 สำหรับกรณี** dynamic multicast (p2 partial)

เป็นอัลกอริทึมที่ดัดแปลงจาก p2 ด้วยเหตุผลเดียวกับ p1 partial สมการต้นทุนการจัดเส้นทางเหมือนของ p2 มีขั้นตอนการทำงานแสดงในรูปที่ 2.15



รูปที่ 2.14 p1 partial algorithm



รูปที่ 2.15 p2 partial algorithm

บทที่ 3
แบบจำลองและวิธีการจำลองแบบ

3.1 แบบจำลอง

แบบจำลองโครงข่ายที่ใช้ในการจำลองแบบการจัดเส้นทางแบบมัลติคาสต์เป็นแบบจำลองโครงข่ายแบบสุ่ม (random graph networks) ที่เสนอโดย Waxman [4] โดยที่แต่ละโนดแทน ATM switch และ edge แทน physical link โนดจะกระจายอยู่บนพิกัดคาร์ทีเซียนจำนวนเต็ม (rectangular coordinates) edge ระหว่างโนด u และ v จะถูกพิจารณาให้ต่อระหว่างโนดทั้งสองด้วยค่าความน่าจะเป็น

$$P(u,v) = \lambda \exp \frac{-d(u,v)}{\rho L} \qquad (3.1)$$

L คือระยะห่างที่มากที่สุดระหว่างคู่โนดใด ๆ

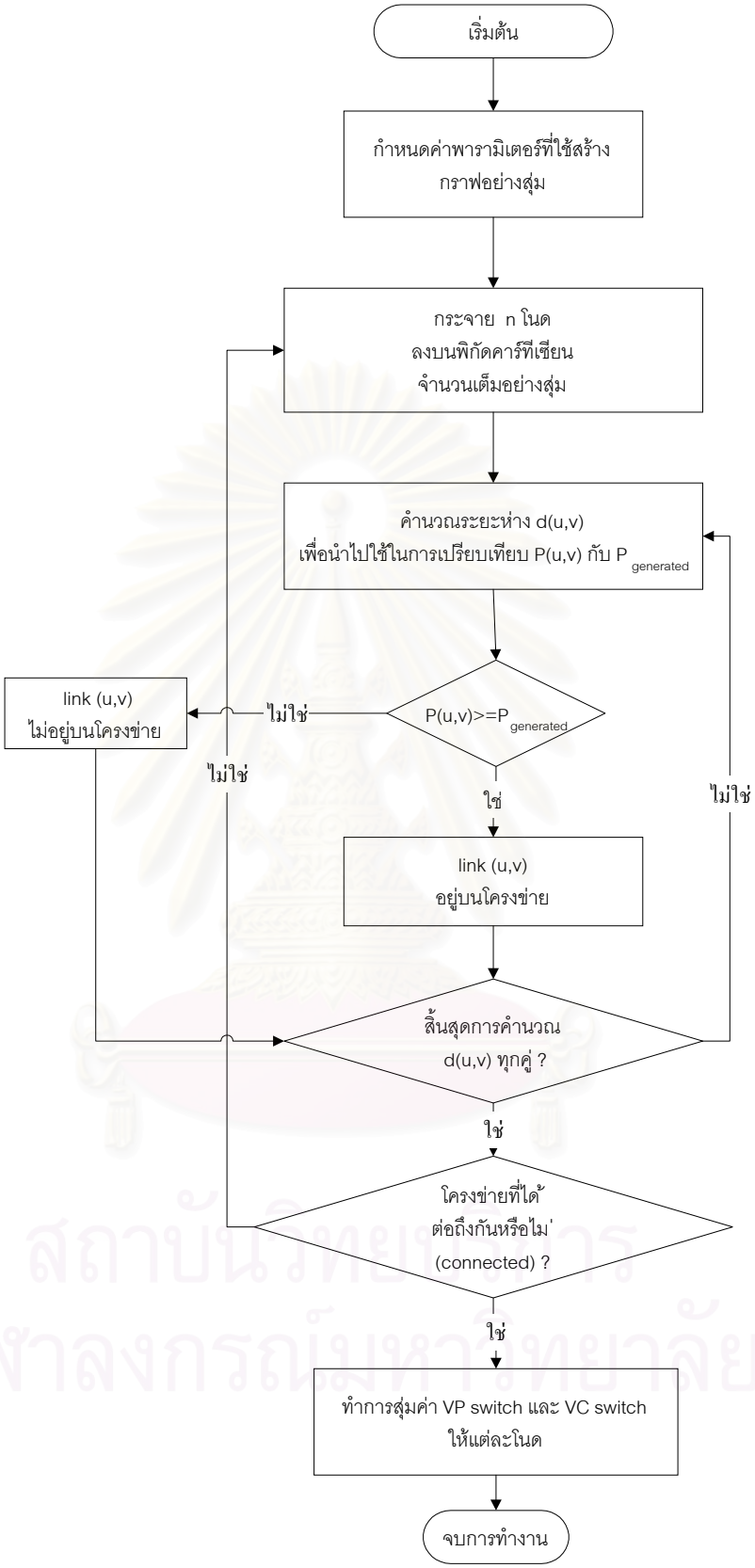
λ เป็นพารามิเตอร์ที่ถ้ายิ่งมีค่าเพิ่มขึ้นจะมีผลทำให้มี link หนาแน่นมากขึ้น

ρ เป็นพารามิเตอร์ที่ถ้ายิ่งมีค่าลดลงจะเพิ่มให้มี link ที่มีความยาวสั้นจำนวนมากขึ้น

d(u,v) คือระยะห่างระหว่างโนด u และ v บนระนาบคาร์ทีเซียน

และ $0 < \lambda \leq 1$, $0 < \rho \leq 1$

ทั้งนี้ในการจำลองแบบจะแบ่งชุดของแบบจำลองโครงข่ายออกเป็นสองชุดตามวิธีการจำลองแบบอัลกอริทึม static multicast และอัลกอริทึมที่เสนอเป็นแนวทางในการทำ dynamic multicast โดยมีขั้นตอนการสร้างแบบจำลองดังแสดงในรูปที่ 3.1

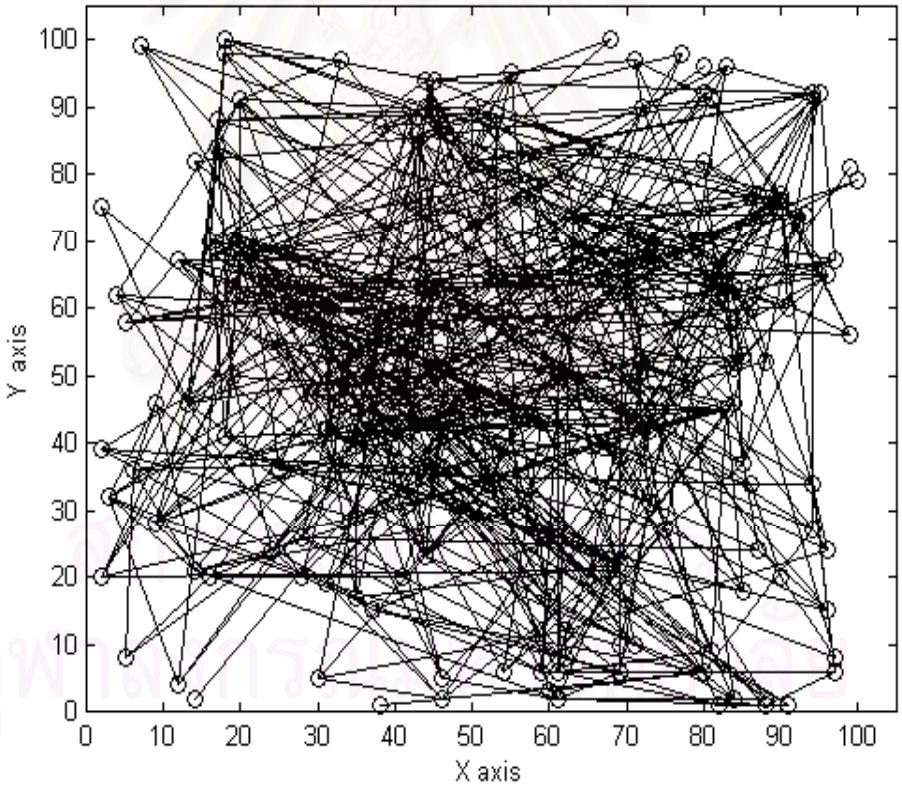


รูปที่ 3.1 ขั้นตอนการสร้างแบบจำลองโครงข่าย ATM เพื่อทดสอบอัลกอริทึมการจัดเส้นทางแบบมัลติคาสต์

3.2 ข้อกำหนดของการจำลองแบบ

แบบจำลองที่ใช้ทดสอบอัลกอริทึม static multicast [4] เป็นโครงข่ายขนาด 200 โนดในรูปที่ 3.2 ที่สร้างตามสมการที่ (3.1) มีข้อกำหนดของการจำลองแบบดังนี้

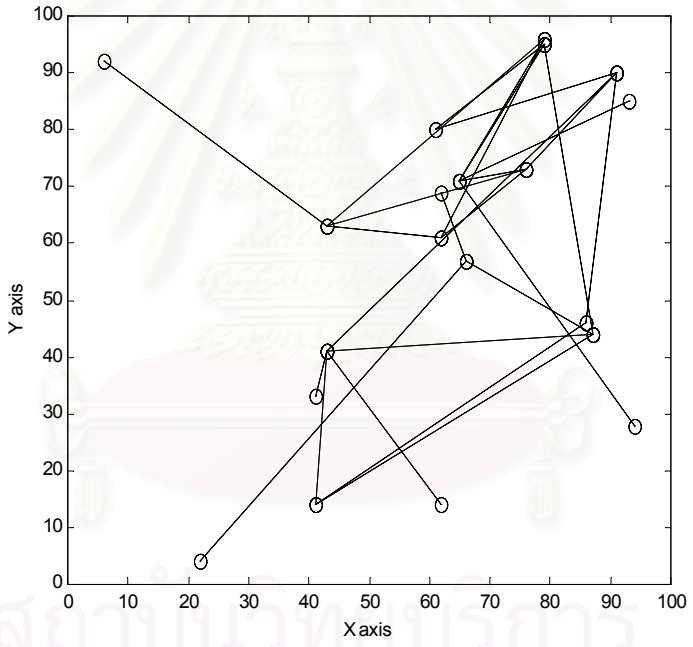
1. ประกอบด้วย VC switch จำนวน 70 % และ VP switch 30 % ของจำนวนสวิตช์ทั้งหมด
2. โนดปลายทางในกลุ่มมัลติคาสต์ D ถูกสุ่มเลือกขึ้นมาจากกลุ่มของ VC switch
3. ขนาดของโนดปลายทางในกลุ่มมัลติคาสต์ $|D|$ เริ่มจาก 5 โนด เพิ่มทีละ 5 โนดไปจนถึง 20 โนด หลังจากนั้นเพิ่มทีละ 10 โนด ไปจนถึง 130 โนด
4. แต่ละ link บนโครงข่ายมี weight เป็น 1
5. พารามิเตอร์ λ = 0.25 และ ρ = 0.2



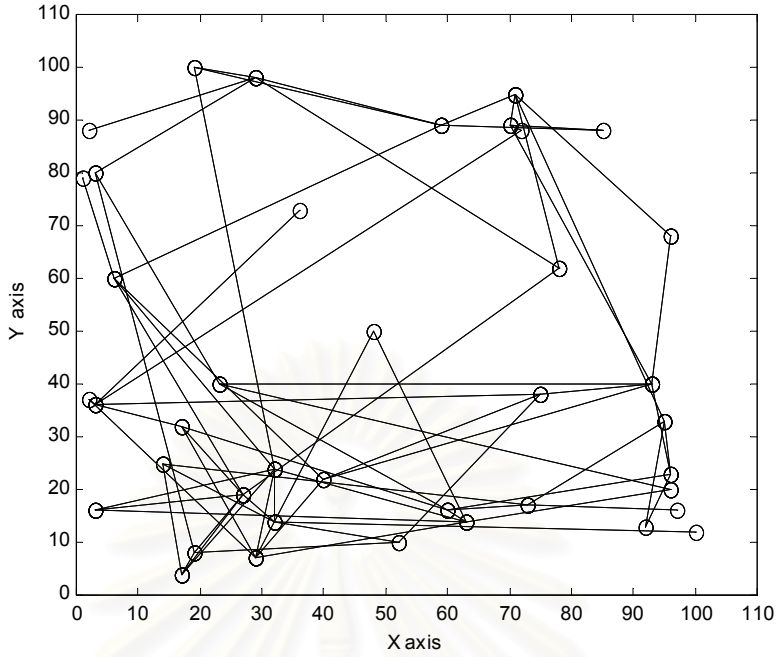
รูปที่ 3.2 โครงข่าย ATM ที่ใช้ในการทดสอบการจัดเส้นทาง static multicast

แบบจำลองที่ใช้ทดสอบอัลกอริทึม dynamic multicast [4] ประกอบด้วยโครงข่ายขนาด 20, 40 และ 60 โนด ในรูปที่ 3.3, 3.4 และ 3.5 มีข้อกำหนดของการจำลองแบบดังนี้

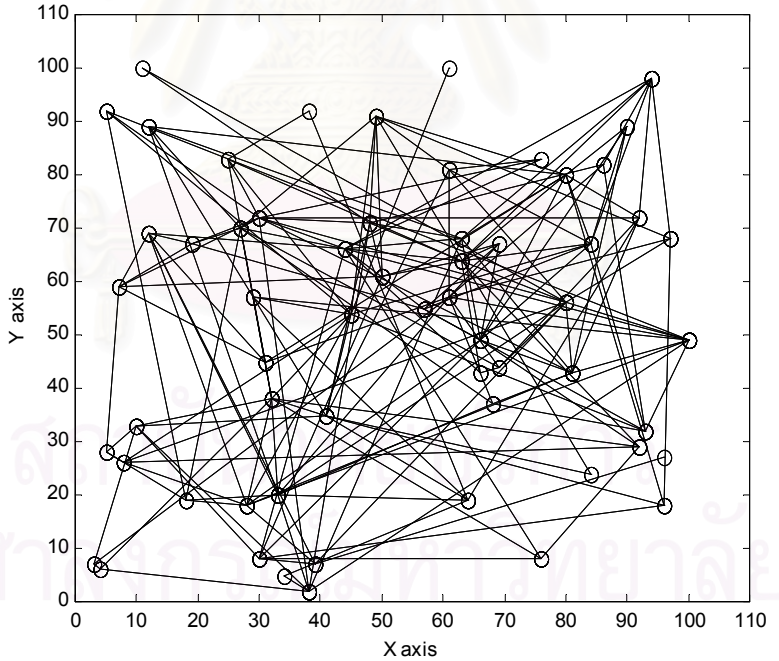
1. มี VC switch จำนวน 70 % และ VP switch จำนวน 30 %
2. โนดปลายทางในกลุ่มมัลติคาสต์ D ถูกสุ่มเลือกขึ้นมาจากกลุ่มของ VC switch
3. ขนาดของโนดปลายทางของกลุ่มมัลติคาสต์สำหรับโครงข่ายขนาด 20 โนด โนดปลายทางเริ่มจาก 2 โนดเพิ่มขึ้นทีละ 1 โนดไปจนถึง 9 โนด โครงข่าย 40 โนด เริ่มจาก 2 โนดเพิ่มขึ้นทีละ 1 โนดไปจนถึง 25 โนด และ โครงข่ายขนาด 60 โนด เริ่มจาก 2 โนดเพิ่มจำนวนขึ้นทีละ 1 โนดไปจนถึง 36 โนด
4. แต่ละ link บนโครงข่ายมี weight เป็น 1
5. พารามิเตอร์ λ = 0.4 และ ρ = 0.3



รูปที่ 3.3 โครงข่าย ATM ขนาด 20 โนดที่ใช้ทดสอบอัลกอริทึมในกรณี dynamic multicast



รูปที่ 3.4 โครงข่าย ATM ขนาด 40 โนดที่ใช้ทดสอบอัลกอริทึมในกรณี dynamic multicast



รูปที่ 3.5 โครงข่าย ATM ขนาด 60 โนดที่ใช้ทดสอบอัลกอริทึมในกรณี dynamic multicast

3.3 วิธีการจำลองแบบ

ในการจัดเส้นทางแบบ static multicast จะสมมติว่ามีการส่งแพ็กเกตข้อมูลในลักษณะมัลติคาสต์ไปยังโนดปลายทางทุกโนดพร้อมกันโดยที่โนดทุกโนดที่จะต่อเข้ากลุ่มมัลติคาสต์จะกำหนดค่าแอดเดรสไว้ล่วงหน้า กล่าวคือไม่มีลำดับก่อนหลังในการส่งข้อมูลแบบมัลติคาสต์ ผลของต้นทุนที่ใช้ในการสร้างเส้นทาง (C_E, C_B และ C_S) ของแต่ละจำนวนโนดปลายทาง $|D|$ จะทำการคำนวณซ้ำ 10 ครั้งตามจำนวนการเลือกเซต D อย่างสุ่มตามเอกสารอ้างอิงของ Jia, et al. [1] แล้วนำมาหาค่าเฉลี่ย

สำหรับการจัดเส้นทางในลักษณะของ dynamic multicast จะเป็นการทดสอบอัลกอริทึม p1 partial และ p2 partial ที่ดัดแปลงมาจากอัลกอริทึม p1 และ p2 ที่เสนอไว้ใน static multicast เปรียบเทียบกับอัลกอริทึมที่ทำการจัดเส้นทางใหม่ทั้งหมดอย่าง Jia restruct , p1 restruct และ p2 restruct เพื่อเป็นแนวทางสำหรับการออกแบบอัลกอริทึมในแบบ dynamic multicast โดยที่ Jia restruct , p1 restruct และ p2 restruct คืออัลกอริทึม Jia, p1 และ p2 ที่ดัดแปลงให้รับทุกการร้องขอ (request) การต่อเข้ามาเก็บไว้ทั้งหมด แล้วจึงจัดเส้นทางสำหรับทุกโนดที่ร้องขอการต่อเข้ากับกลุ่มมัลติคาสต์เหล่านั้นพร้อมกัน ดังนั้นการร้องขอที่เข้ามาใหม่ของโนดใด ๆ จะถูกนำไปใช้เป็นตัวเลือกในการจัดเส้นทางร่วมกับการร้องขอจากโนดก่อนหน้า เป็นการทำ static multicast โดยนำการร้องขอการต่อสำหรับสภาวะ dynamic multicast มาเก็บรวมไว้แล้วจัดเส้นทางพร้อมกัน ซึ่งต่างจากในกรณีของอัลกอริทึม p1 partial และ p2 partial ซึ่งจะจัดเส้นทางสำหรับแต่ละการร้องขอการต่อ ในสภาวะการใช้งานแบบ dynamic multicast การส่งแพ็กเกตข้อมูลไปยังโนดปลายทางที่กำหนดหรือที่มีการร้องขอการต่อเข้ากับกลุ่มมัลติคาสต์ของแต่ละโนดปลายทางจะเกิดขึ้นไม่พร้อมกัน ดังนั้นการจำลองแบบสมมติให้มีการร้องขอการต่อเข้ามาหากลุ่มมัลติคาสต์ทีละ 1 การร้องขอ อย่างไรก็ตามไม่พิจารณาผลการร้องขอที่จะออกจากกลุ่มมัลติคาสต์ โดยได้กำหนดให้มีการสุ่มการร้องขอการต่อกับกลุ่มมัลติคาสต์ในรูปของลำดับ $\{r_1, r_2, r_3 \ldots, r_n\}$ และอัลกอริทึมจะทำการจัดเส้นทางให้กับแต่ละการร้องขอ r_1, r_2 ไปจนถึง r_n ผลที่ได้จากการจัดเส้นทางประกอบด้วยต้นทุนจากการสร้างการต่อ C_E, ต้นทุนของแบนด์วิดท์ C_B, ต้นทุนการสวิตช์วงจรเสมือนและวิถีเสมือน C_S และเวลาที่ใช้ในการประมวลผลเส้นทางของอัลกอริทึมสำหรับแต่ละการร้องขอที่เพิ่มเข้ามา (running time) โดยค่าของต้นทุนและเวลาจะทำการเฉลี่ยตามจำนวนรอบการสุ่ม VP switch และ

VC switch ในที่นี้ทำการสุ่ม 100 รอบ สำหรับเวลาที่ใช้ในการจัดเส้นทางของอัลกอริทึมของอัลกอริทึม p1 partial และ p2 partial จะวัดสำหรับทุกการจัดเส้นทางสำหรับแต่ละการร้องขอการต่อที่เข้ามา แต่สำหรับ Jia restruct, p1 restruct และ p2 restruct จะวัดเวลาสำหรับทุก ๆ การจัดเส้นทางของกลุ่มการร้องขอที่เก็บเอาไว้ เหมือนกับที่วัดใน static multicast

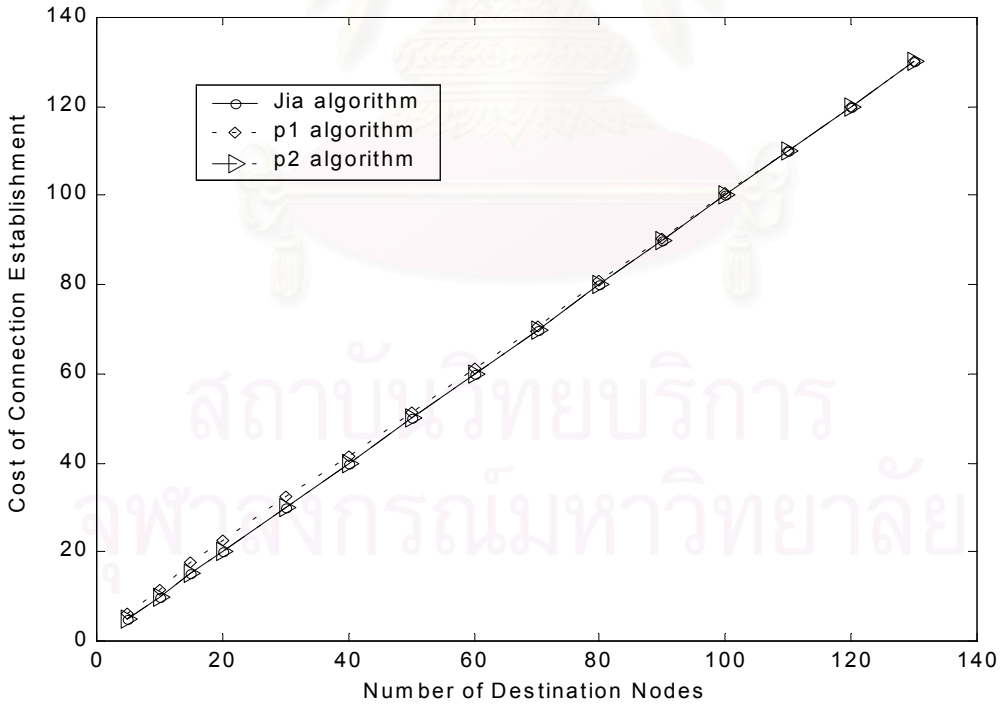
สำหรับเวลาในการคำนวณเส้นทางของอัลกอริทึมการจัดเส้นทางแบบมัลติคาสต์ที่ใช้ในการจำลองแบบทั้งในกรณี static multicast และในกรณี dynamic multicast อัลกอริทึมทุกชุดทำการประมวลผลบนเครื่องคอมพิวเตอร์ที่ใช้ CPU Pentium III 667 MHz และมีขนาดของหน่วยความจำ 128 MB วัดค่าเวลาเป็นวินาที

บทที่ 4
ผลการจำลองแบบและวิเคราะห์ผลการจำลองแบบ

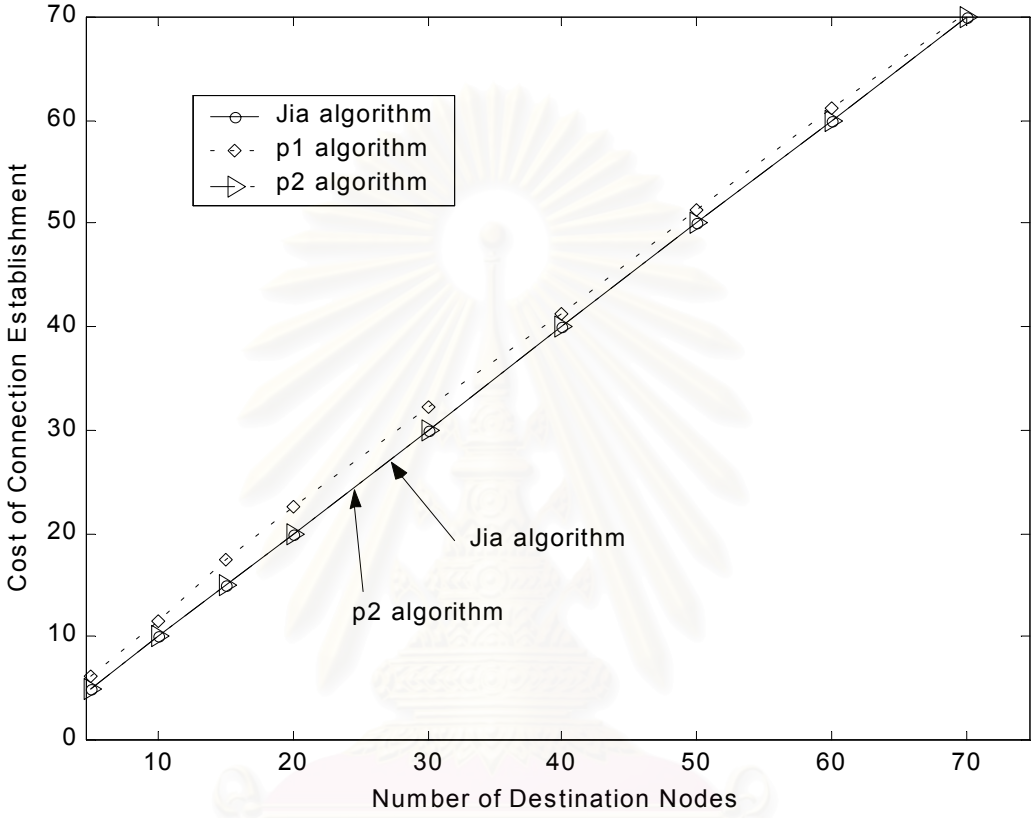
ผลการจำลองแบบการจัดเส้นทางแบบมัลติคาสต์บนโครงข่าย ATM แบ่งออกเป็นการจัดเส้นทางในกรณีของ static multicast และกรณี dynamic multicast

4.1 ผลการจำลองแบบการจัดเส้นทางแบบมัลติคาสต์ในโครงข่าย ATM กรณี static multicast

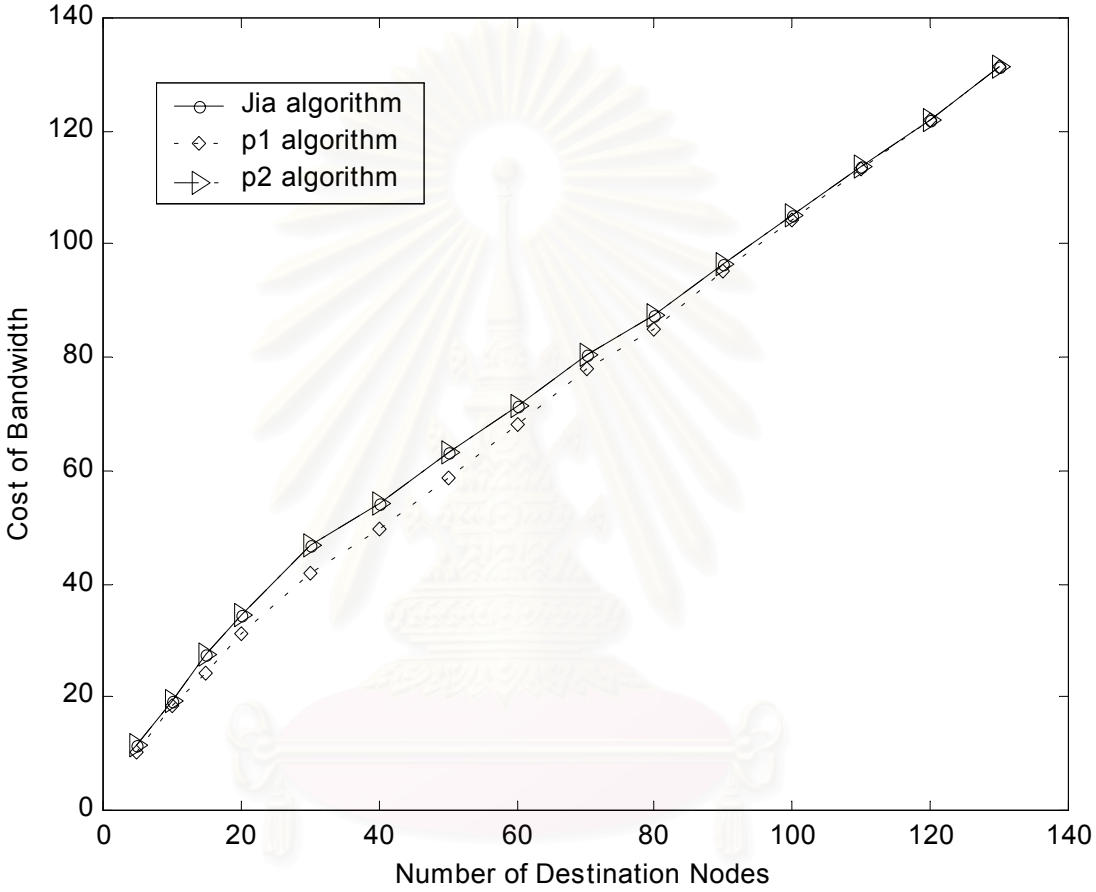
ผลการจำลองแบบการจัดเส้นทางในกรณี static multicast ที่สมมติให้ทราบแอดเดรสของโนดปลายทางที่จะทำการส่งแพ็กเกตข้อมูลทั้งหมด จะทำการเปรียบเทียบระหว่างอัลกอริทึมที่เสนอคือ p1 และ p2 กับอัลกอริทึมที่เสนอโดย Jia, et al. [1] ประกอบด้วยผลกราฟต้นทุนการสร้างการต่อ C_E, กราฟต้นทุนแบนด์วิดท์ C_B, กราฟต้นทุนของการสวิตช์ VP, กราฟต้นทุนของการสวิตช์ VC และกราฟของเวลาที่ใช้ในการคำนวณเส้นทางของแต่ละอัลกอริทึม, กราฟของจำนวน Steiner nodes, กราฟแสดงความยาววิถีสูงสุด และกราฟแสดงความยาววิถีเฉลี่ย ดังแสดงในรูปที่ 4.1 - 4.11 ตามลำดับ



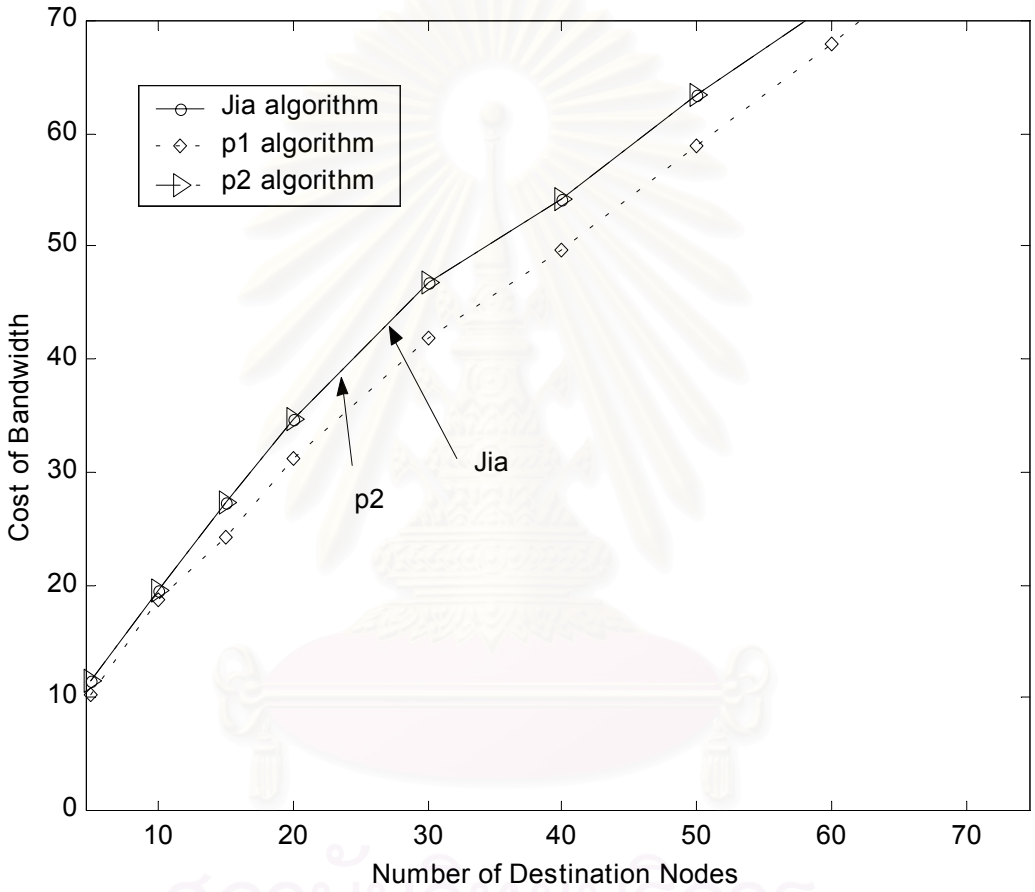
รูปที่ 4.1 กราฟแสดงผลของต้นทุนการสร้างการต่อถึงกัน (C_E) ของการจัดเส้นทางแบบ static multicast เปรียบเทียบระหว่างอัลกอริทึม p1, p2 และ Jia



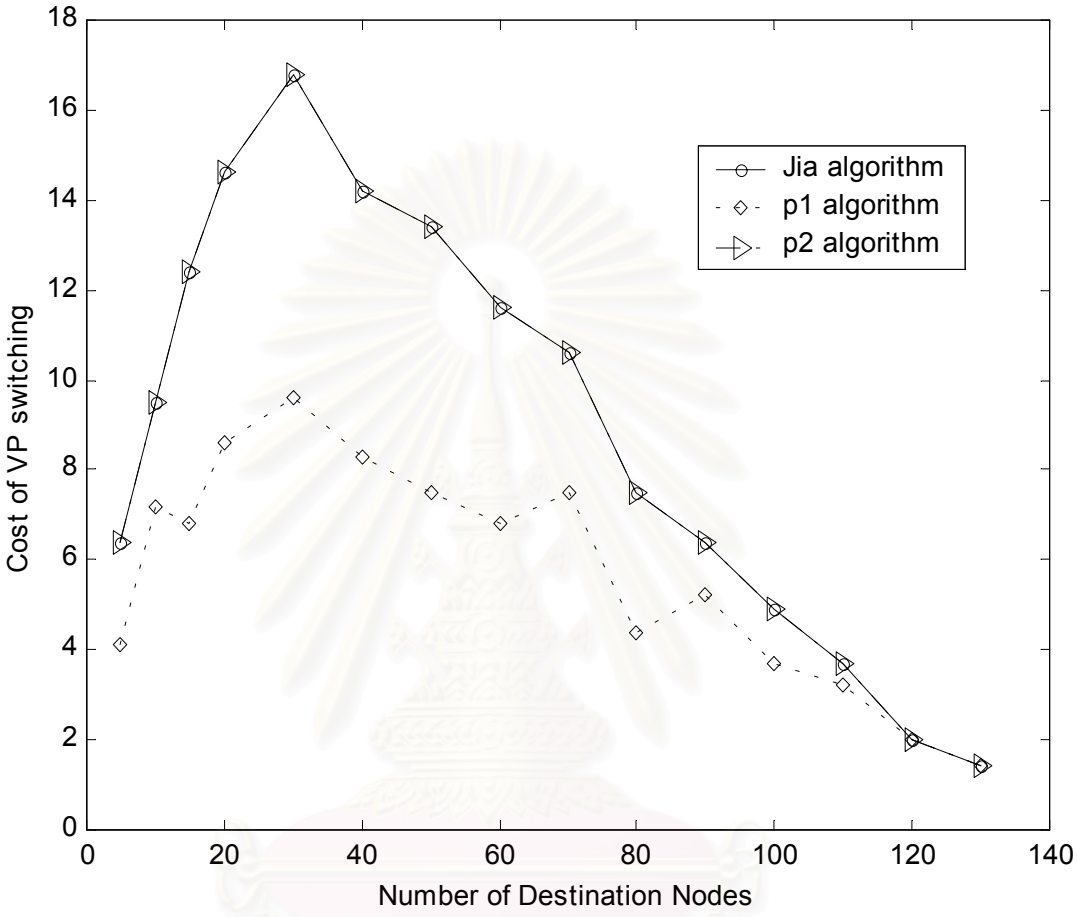
รูปที่ 4.2 กราฟแสดงผลของต้นทุนการสร้างการต่อถึงกัน (C_E) ของการจัดเส้นทางแบบ static multicast เปรียบเทียบระหว่างอัลกอริทึม p1, p2 และ Jia ในช่วงโนดปลายทาง 5 โนดถึง 70 โนด



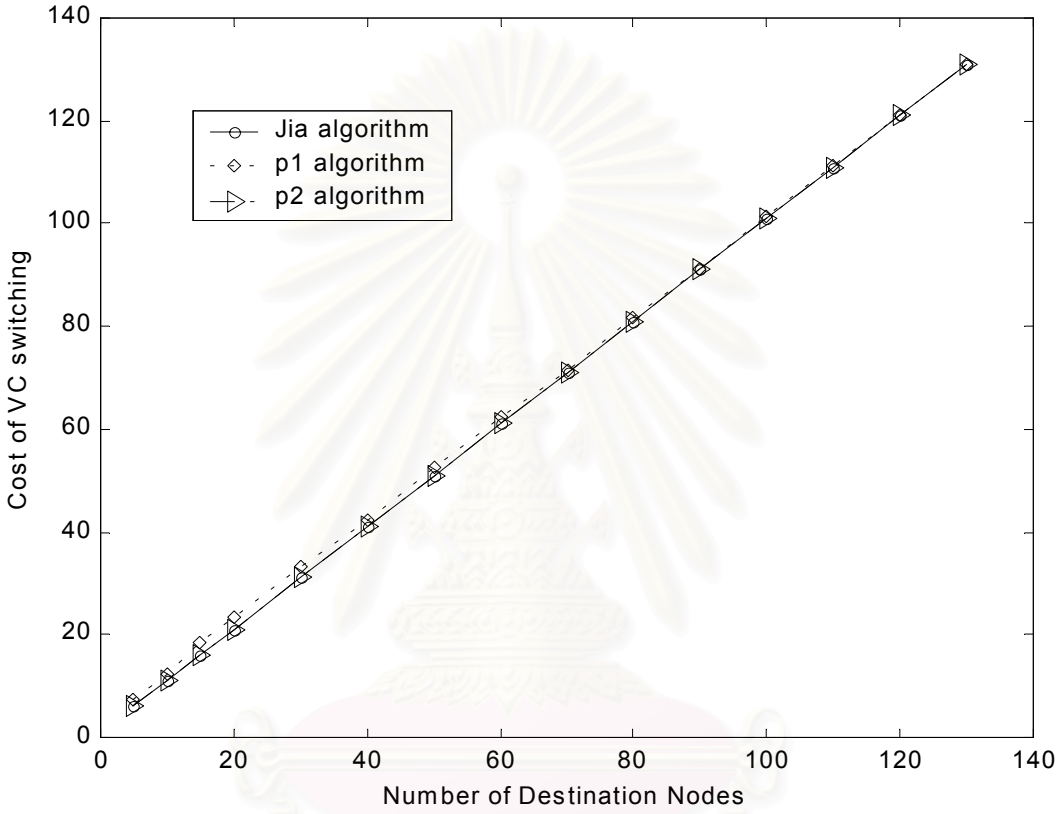
รูปที่ 4.3 กราฟแสดงผลของต้นทุนของแบนด์วิดท์ (C_B) ของการจัดเส้นทางแบบ static multicast เปรียบเทียบระหว่างอัลกอริทึม p1, p2 และ Jia



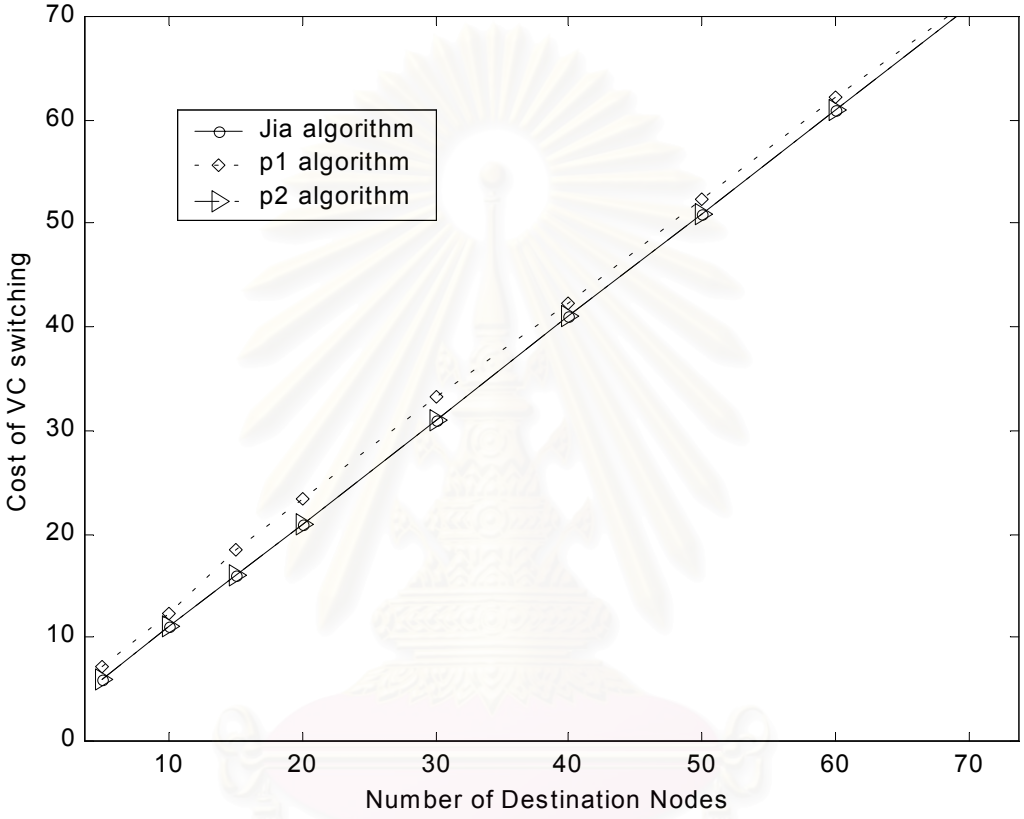
รูปที่ 4.4 กราฟแสดงผลของต้นทุนของแบนด์วิดท์ (C_B) ของการจัดเส้นทางแบบ static multicast เปรียบเทียบระหว่างอัลกอริทึม p1, p2 และ Jia ในช่วงโนดปลายทางตั้งแต่ 5 โนดถึง 70 โนด



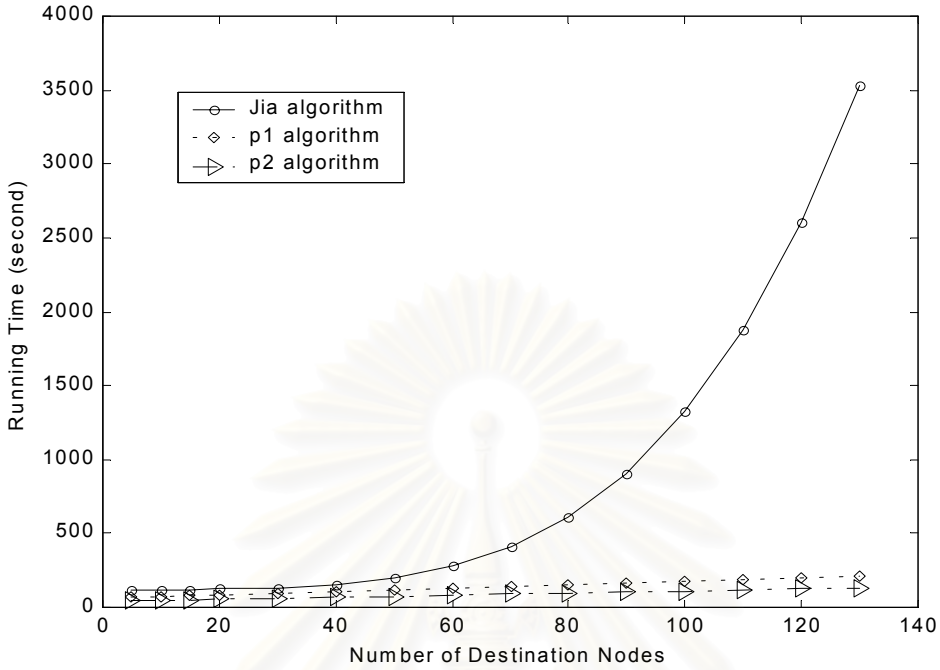
รูปที่ 4.5 กราฟแสดงผลของต้นทุนการสวิตช์ VP ของการจัดเส้นทางแบบ static multicast เปรียบเทียบระหว่างอัลกอริทึม p1, p2 และ Jia



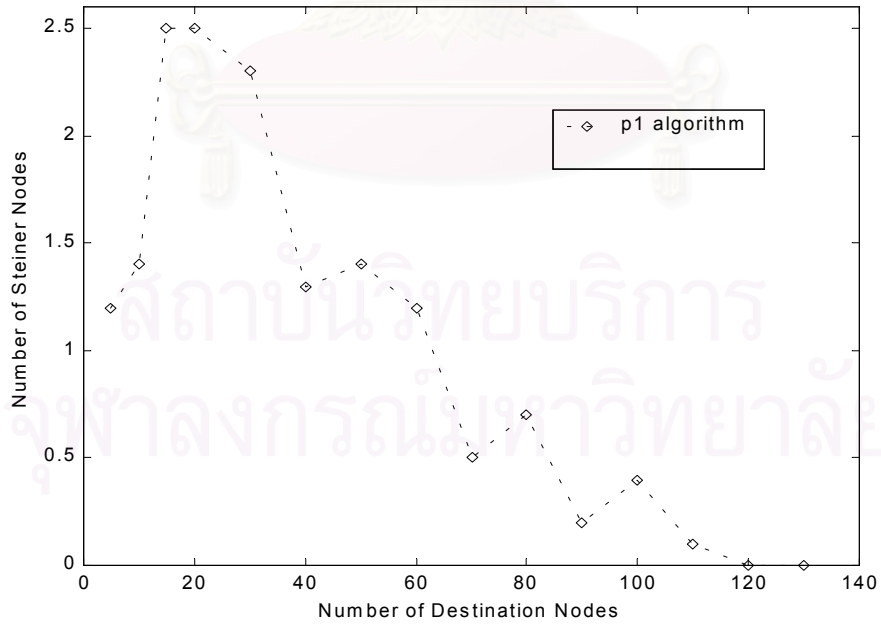
รูปที่ 4.6 กราฟแสดงผลของต้นทุนการสวิตช์ VC ของการจัดเส้นทางแบบ static multicast เปรียบเทียบระหว่างอัลกอริทึม p1, p2 และ Jia



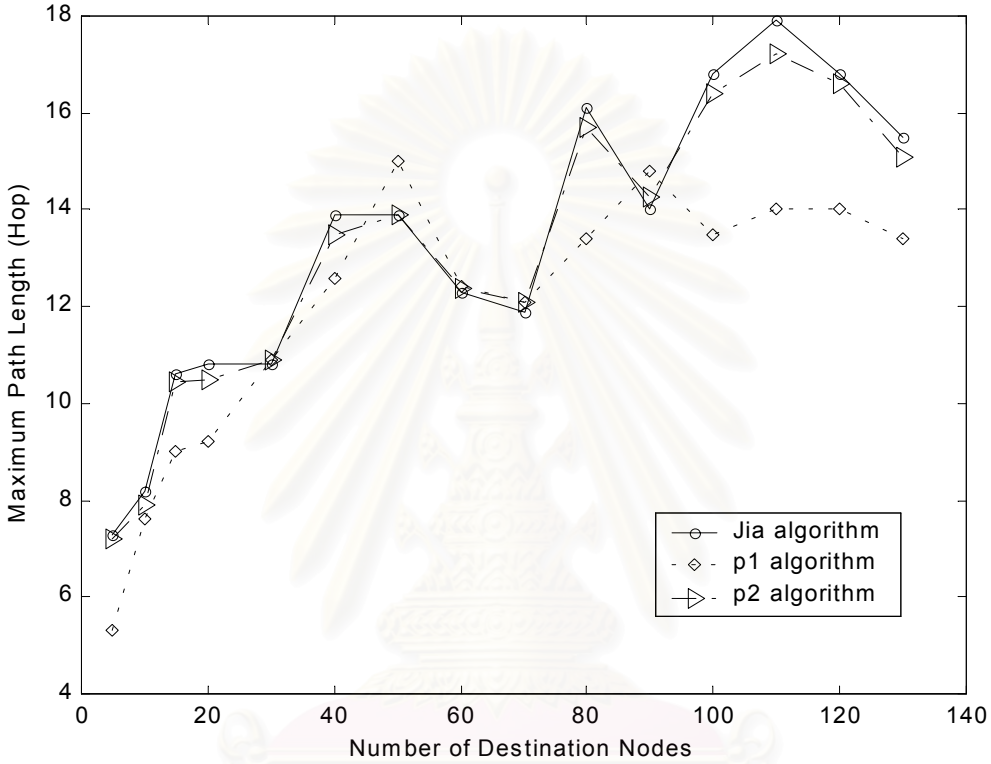
รูปที่ 4.7 กราฟแสดงผลของต้นทุนการสวิตช์ VC ของการจัดเส้นทางแบบ static multicast เปรียบเทียบระหว่างอัลกอริทึม p1, p2 และ Jia ในช่วงโนดปลายทางตั้งแต่ 5 โนดถึง 60 โนด



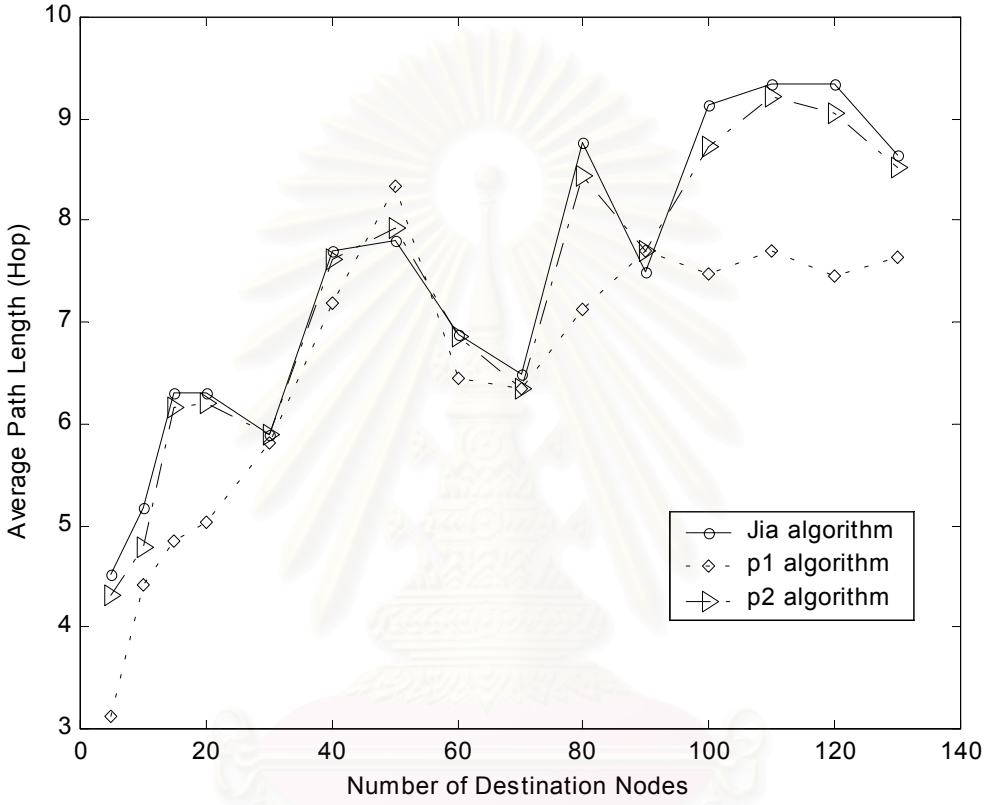
รูปที่ 4.8 กราฟแสดงเวลาที่ใช้ในการคำนวณเส้นทางของการจัดเส้นทาง
แบบ static multicast เปรียบเทียบระหว่างอัลกอริทึม p1, p2 และ jia



รูปที่ 4.9 กราฟแสดงจำนวน Steiner Nodes ที่เกิดขึ้นบน multicast tree
ของอัลกอริทึม p1



รูปที่ 4.10 กราฟแสดงความยาววิถีสูงสุด (Maximum Path Length) ของโนดต้นทางไปยังโนดปลายทางบน multicast tree เปรียบเทียบระหว่างอัลกอริทึม p1, p2 และ Jia



รูปที่ 4.11 กราฟแสดงความยาววิถีเฉลี่ย (Average Path Length) ของ multicast tree ซึ่งวัดจากโนดต้นทางไปยังโนดปลายทางทุก ๆ โนด มาหาค่าเฉลี่ย เปรียบเทียบระหว่างอัลกอริทึม p1, p2 และ Jia

4.2 วิเคราะห์ผลการจำลองแบบการจัดเส้นทางมัลติคาสต์บนโครงข่าย ATM ในกรณี static multicast

การเปรียบเทียบต้นทุนของอัลกอริทึมการจัดเส้นทางระหว่าง p1, p2 และ Jia นั้น ในที่นี้จะพิจารณาเป็นค่าสัมพัทธ์ซึ่งไม่มีหน่วยที่สืบเนื่องมาจากการคำนวณหา Virtual Network ในบทที่ 3 โดยจะพิจารณาเปรียบเทียบต้นทุนแยกเป็น C_E, C_B และต้นทุนในการสวิตช์ VC และ VP

4.2.1 ต้นทุนการสร้างการต่อถึงกัน (C_E)

ตารางที่ 4.1 ค่า C_E ที่ได้จากการหาเส้นทางโดยใช้อัลกอริทึม Jia, p1 และ p2

จำนวนโนดปลายทาง	Jia algorithm	p1 algorithm	p2 algorithm	$\frac{C_{E,p1}-C_{E,Jia}}{C_{E,Jia}}\times 100$ (%)	$\frac{C_{E,p2}-C_{E,Jia}}{C_{E,Jia}}\times 100$ (%)
5	5.0	6.2	5.0	24	0
10	10.0	11.4	10.0	14	0
15	15.0	17.5	15.0	16	0
20	20.0	22.5	20.0	12.5	0
30	30.0	32.3	30.0	7.67	0
40	40.0	41.3	40.0	3.25	0
50	50.0	51.4	50.0	2.8	0
60	60.0	61.2	60.0	2.0	0
70	70.0	70.5	70.0	0.71	0
80	80.0	80.7	80.0	0.88	0
90	90.0	90.2	90.0	0.22	0
100	100.0	100.4	100.0	0.40	0
110	110.0	110.1	110.0	0.09	0
120	120.0	120.0	120.0	0	0
130	130.0	130.0	130.0	0	0

พิจารณาต้นทุนการสร้างการต่อถึงกัน (C_E) ในรูปที่ 4.1,4.2 และตารางที่ 4.1 อัลกอริทึม p1 มีต้นทุนการสร้างการต่อถึงกันของ VPC สูงกว่า ของ Jia และ p2 ในช่วงจำนวนโนดปลายทางขนาดเล็ก ตั้งแต่ 5 โนดไปจนถึง 30 โนด โดยมีค่า $\frac{C_{E,p1} - C_{E,Jia}}{C_{E,Jia}} \times 100$ ลดลงจาก 24 % ไปจนถึง 7.67 % และค่า C_E เริ่มลู่เข้าหากันเมื่อจำนวนโนดปลายทางเพิ่มมากขึ้น ทั้งนี้เนื่องจากอัลกอริทึม p1 มีการนำโนดที่อยู่นอกกลุ่มมัลติคาสต์มาพิจารณาใช้เป็น branch points เพื่อใช้ในการสร้าง multicast tree ซึ่งโนดเหล่านี้ถูกเรียกว่า Steiner node จึงทำให้เกิดการเริ่มต้นและสิ้นสุดของ VPC ที่โนดดังกล่าว ทำให้จำนวนการต่อถึงกันของ VPC ผ่าน VC switch มีเพิ่มมากขึ้น อย่างไรก็ตามเมื่อจำนวนโนดปลายทางมีปริมาณเพิ่มขึ้น จำนวนโนดที่จะมีโอกาสเป็น Steiner node จะลดน้อยลงดังรูปที่ 4.9 ทำให้ต้นทุนการต่อถึงกันเท่ากันที่จำนวนโนดปลายทางตั้งแต่ 120 โนดเป็นต้นไป ส่วนอัลกอริทึม p2 นั้น เนื่องจากไม่ใช้ Steiner node มาช่วยในการสร้าง ทำให้มีต้นทุนการต่อถึงกันเท่ากับของ Jia (ค่า $\frac{C_{E,p2} - C_{E,Jia}}{C_{E,Jia}} \times 100$ เท่ากับศูนย์) ในส่วนลักษณะกราฟ กราฟทั้ง 3 เส้นมีลักษณะเพิ่มขึ้นโดยตลอด เนื่องจากเมื่อมีการสร้าง multicast tree สำหรับจำนวนโนดที่มากขึ้น จะมีการสร้างการต่อถึงกันแบบจุดถึงจุด (point-to-point) เพื่อประกอบกันเป็น multicast tree มากขึ้น ทำให้จำนวน VPC เพิ่มมากตามไปด้วย และจำนวนโนดที่ทำการสวิตช์ VC ก็จะเพิ่มขึ้นด้วยดังกราฟรูปที่ 4.6 และตารางที่ 4.4

4.2.2 **ต้นทุนของแบนด์วิดท์** (C_B)

ตารางที่ 4.2 ต้นทุนของแบนด์วิดท์ที่ได้จากการจัดเส้นทางโดยใช้อัลกอริทึม Jia, p1 และ p2

จำนวนโนดปลายทาง	Jia algorithm	p1 algorithm	p2 algorithm	$\frac{C_{B,p1}-C_{B,Jia}}{C_{B,Jia}}\times 100$ (%)	$\frac{C_{B,p2}-C_{B,Jia}}{C_{B,Jia}}\times 100$ (%)
5	11.4	10.3	11.4	- 9.65	0
10	19.5	18.6	19.5	- 4.62	0
15	27.4	24.3	27.4	- 11.31	0
20	34.6	31.1	32.6	- 11.25	0
30	46.8	41.9	46.8	- 10.47	0
40	54.2	49.6	54.2	- 8.49	0
50	63.4	58.9	63.4	- 7.10	0
60	71.6	68.0	71.6	- 5.03	0
70	80.6	78.0	80.6	- 3.23	0
80	87.5	85.1	87.5	- 2.74	0
90	96.4	95.4	96.4	- 1.04	0
100	104.9	104.1	104.9	- 0.76	0
110	113.7	113.3	113.7	- 0.35	0
120	122.0	122.0	122.0	0	0
130	131.4	131.4	131.4	0	0

พิจารณาต้นทุนของแบนด์วิดท์ (C_B) ในรูปที่ 4.3, 4.4 และตารางที่ 4.2 พบว่า อัลกอริทึม p1 มีต้นทุนต่ำกว่า Jia โดยในช่วงจำนวนโนดปลายทางตั้งแต่ 5 โนดไปจนถึง 60 โนดมีค่า C_B ต่ำกว่า Jia อยู่ในช่วงตั้งแต่ 11.31 % จนถึง 4.62 % เนื่องมาจากอัลกอริทึม p1 อาศัย Steiner node มาช่วยในการสร้างการต่อถึงกัน อย่างไรก็ตามการเลือก Steiner node เป็นแบบสุ่มทำให้ได้จำนวน Steiner node ไม่มากนัก แสดงดังกราฟรูปที่ 4.9 ในส่วนอัลกอริทึม p2 มีต้นทุนแบนด์วิดท์เท่ากันกับอัลกอริทึมของ Jia ทั้งนี้เนื่องจากอัลกอริทึม p2 ไม่มีการใช้ Steiner node มาช่วยสร้างเส้นทาง และเส้นกราฟทั้ง

3 เส้นมีลักษณะเพิ่มขึ้นโดยตลอดเพราะยิ่งจำนวนโนดปลายทางเพิ่มมากขึ้น จะสิ้นเปลืองแบนด์วิดท์ของการสร้างเส้นทางมากขึ้น และกราฟของ C_B ทั้ง 3 เส้นจะลู่เข้าหากันเมื่อจำนวนโนดปลายทาง 120 โนดเป็นต้นไป (จำนวน VC switches ทั้งหมดมี 140 โนด)

4.2.3 **ต้นทุนของการสวิตช์** VP

ตารางที่ 4.3 ต้นทุนของการสวิตช์ VP ในการจัดเส้นทางของอัลกอริทึม Jia, p1 และ p2

จำนวนโนดปลายทาง	Jia algorithm	p1 algorithm	p2 algorithm	$\frac{C_{VP,p1}-C_{VP,Jia}}{C_{VP,Jia}}\times 100$ (%)	$\frac{C_{VP,p2}-C_{VP,Jia}}{C_{VP,Jia}}\times 100$ (%)
5	6.4	4.1	6.4	- 35.94	0
10	9.5	7.2	9.5	- 24.21	0
15	12.4	6.8	12.4	- 45.16	0
20	14.6	8.6	14.6	- 41.78	0
30	16.8	9.6	16.8	- 42.86	0
40	14.2	8.3	14.2	- 41.55	0
50	13.4	7.5	13.4	- 44.03	0
60	11.6	6.8	11.6	- 41.38	0
70	10.6	7.5	10.6	- 29.25	0
80	7.5	4.4	7.5	- 41.33	0
90	6.4	5.2	6.4	- 18.75	0
100	4.9	3.7	4.9	- 24.49	0
110	3.7	3.2	3.7	- 13.51	0
120	2.0	2.0	2.0	0	0
130	1.4	1.4	1.4	0	0

ในส่วนของต้นทุนของการสวิตช์ VP ในรูปที่ 4.5 และตารางที่ 4.3 พบว่าอัลกอริทึม p1 มีต้นทุนการสวิตช์ต่ำกว่าอัลกอริทึม p2 และ Jia ค่อนข้างมากเนื่องจากใช้ Steiner node มาช่วยในการสร้าง multicast tree อัลกอริทึม p1 ให้ผลต่ำกว่า Jia ในช่วงโนดปลายทางจำนวน 5 โนดถึง 100 โนด โดยมีค่าต่ำกว่า Jia อยู่ในช่วงตั้งแต่ 24.21 % ไปจนถึง 45.16 % และอัลกอริทึม p2 ให้ผลเท่ากับ Jia ลักษณะของกราฟต้นทุนการสวิตช์ VP เพิ่มขึ้นจนถึงค่าโนดปลายทางเท่ากับ 30 โนด จึงเริ่มลดค่าลง เนื่องจากจำนวนโนดปลายทางที่เพิ่มขึ้นทำให้ VC switch ที่จะทำหน้าที่เป็น VP switch เริ่มมีจำนวนลดลง เพราะ VC switch เหล่านั้นจะทำการ VC switching ที่โนดต้นทาง โนดปลายทาง และโนดที่เป็น branch points

4.2.4 **ต้นทุนของการสวิตช์** VC

ตารางที่ 4.4 ต้นทุนของการสวิตช์ VC ในการจัดเส้นทางของอัลกอริทึม Jia, p1 และ p2

จำนวนโนดปลายทาง	Jia algorithm	p1 algorithm	p2 algorithm	$\frac{C_{VC,p1}-C_{VC,Jia}}{C_{VC,Jia}}\times 100$ (%)	$\frac{C_{VC,p2}-C_{VC,Jia}}{C_{VC,Jia}}\times 100$ (%)
5	6.0	7.2	6.0	20.00	0
10	11.0	12.4	11.0	12.72	0
15	16.0	18.5	16.0	15.63	0
20	21.0	23.5	21.0	10.64	0
30	31.0	33.3	31.0	7.42	0
40	41.0	42.3	41.0	3.17	0
50	51.0	52.4	51.0	2.75	0
60	61.0	62.2	61.0	1.97	0
70	71.0	71.5	71.0	0.70	0
80	81.0	81.7	81.0	0.86	0
90	91.0	91.2	91.0	0.22	0
100	101.0	101.4	101.0	0.40	0

จำนวนโนดปลายทาง	Jia algorithm	p1 algorithm	p2 algorithm	$\frac{C_{VC,p1}-C_{VC,Jia}}{C_{VC,Jia}}\times 100$ (%)	$\frac{C_{VC,p2}-C_{VC,Jia}}{C_{VC,Jia}}\times 100$ (%)
110	111.0	111.1	111.0	0.09	0
120	121.0	121.0	121.0	0	0
130	131.0	131.0	131.0	0	0

ต้นทุนการสวิตช์ VC ในรูปที่ 4.6 และตารางที่ 4.4 มีแนวโน้มเพิ่มขึ้นเช่นเดียวกับกราฟของ C_E และ C_B และต้นทุนการสวิตช์ VC ของอัลกอริทึม p1 สูงกว่าของ Jia ในช่วงจำนวนโนดปลายทางมีจำนวนน้อยเมื่อจำนวนโนดปลายทางเพิ่มขึ้นจะมีต้นทุนการสวิตช์สูงกว่า Jia เพียงเล็กน้อยเช่นตั้งแต่ 20 โนดเป็นต้นไป ค่าความแตกต่างของต้นทุนการสวิตช์ VC จะลดลงจาก 10.61 % ไปจนถึง 0.4 % ที่ 100 โนด ขณะที่กราฟของ p2 ทับกันกับของ Jia เพราะไม่มีการใช้โนดนอกเหนือกลุ่มมัลติคาสต์มาช่วยสร้างเส้นทาง จึงมีจำนวน VC switching เท่ากัน

4.2.5 เวลาที่ใช้ในการจัดเส้นทาง

ตารางที่ 4.5 ค่า worse - case running time ที่แสดงในรูปแบบ complexity ของอัลกอริทึม Jia, p1 และ p2

อัลกอริทึม	Complexity
Jia	$O(n^3 + m\log m)$
p1	$O(n^3)$
p2	$O(n^3)$

หมายเหตุ m และ n คือ จำนวนข่ายเชื่อมโยงและจำนวนโนดบนโครงข่าย และค่า $O(.)$ คือ ค่าขีดจำกัดบน (upper bound) ของเวลาที่ใช้ในการคำนวณเมื่อเกิดกรณีที่แย่ที่สุด

เมื่อพิจารณาเวลาที่ใช้ในการคำนวณเส้นทางของอัลกอริทึมแต่ละชุดดังรูปที่ 4.8 พบว่าอัลกอริทึมของ Jia จะใช้เวลาเพิ่มมากขึ้นอย่างรวดเร็วเมื่อจำนวนโนดปลายทางสูงกว่า 70 โนดเป็นต้นไปเมื่อเทียบกับอัลกอริทึม p1 และ p2 ส่วนอัลกอริทึม p1 ใช้เวลามากกว่าอัลกอริทึม p2 เนื่องจากต้องพิจารณาหา Steiner Node ขณะที่ p2 ไม่ต้องคำนวณในส่วนนี้ เมื่อพิจารณาค่าเวลาที่แย่ที่สุดที่ใช้ในการประมวลผลหรือค่าความซับซ้อน (complexity) ของอัลกอริทึมดังตารางที่ 4.5 กล่าวได้ว่า

อัลกอริทึม p1 และ p2 จะทำงานได้เร็วกว่าอัลกอริทึม Jia โดยที่อัลกอริทึมของ Jia มีความซับซ้อนที่นอกจากจะขึ้นกับจำนวนโนดแล้วยังขึ้นกับจำนวนข่ายเชื่อมโยงอีกด้วย ดังค่า $O(n^3 + m\log m)$ ในกรณีที่มีโครงข่ายมีจำนวนข่ายเชื่อมโยงหนาแน่นค่า $m\log m$ อาจมีค่ามากกว่า n^3 และอัลกอริทึม Jia จะขึ้นกับจำนวนข่ายเชื่อมโยงมากกว่าจำนวนโนดของโครงข่าย (ค่า $O(.)$ ของแต่ละอัลกอริทึมแสดงการคำนวณในภาคผนวก ค.)

4.2.6 **ความยาวของวิถีของการจัดเส้นทางแบบมัลติคาสต์**

ตารางที่ 4.6 ความยาววิถีที่ยาวที่สุด (Maximum Path Length) ที่ได้จาก
การจัดเส้นทางแบบมัลติคาสต์ โดยใช้อัลกอริทึม Jia, p1 และ p2

จำนวนโนดปลายทาง	Jia algorithm	p1 algorithm	p2 algorithm	$\frac{L_{MAX,p1} - L_{MAX,Jia}}{L_{MAX,Jia}} \times 100$ (%)	$\frac{L_{MAX,p2} - L_{MAX,Jia}}{L_{MAX,Jia}} \times 100$ (%)
5	7.30	5.30	7.20	- 27.40	- 1.36
10	8.20	7.60	7.90	- 7.32	- 3.66
15	10.60	9.00	10.46	- 15.09	- 1.32
20	10.80	9.20	10.50	- 14.81	- 2.78
30	10.80	10.90	10.90	0.93	0.93
40	13.90	12.60	13.50	- 9.35	- 2.88
50	13.90	15.00	13.90	7.91	0
60	12.30	12.40	12.38	0.81	0.65
70	11.90	12.10	12.10	1.68	1.68
80	16.10	13.40	15.7	- 16.77	- 2.48
90	14.00	14.80	14.27	5.71	1.93
100	16.80	13.50	16.4	- 19.64	- 2.38
110	17.90	14.00	17.2	- 21.79	- 3.91
120	18.60	14.00	16.6	- 24.73	- 10.75
130	15.50	13.40	15.10	- 13.55	- 2.58

ตารางที่ 4.7 ความยาววิถีเฉลี่ย (Average Path Length) ที่ได้จากการจัดเส้นทางแบบมัลติคาสต์ โดยใช้อัลกอริทึม Jia, p1 และ p2

จำนวนโนดปลายทาง	Jia algorithm	p1 algorithm	p2 algorithm	$\frac{L_{AV,p1}-L_{AV,Jia}}{L_{AV,Jia}}\times 100$ (%)	$\frac{L_{AV,p2}-L_{AV,Jia}}{L_{AV,Jia}}\times 100$ (%)
5	4.52	3.13	4.31	- 30.75	- 4.73
10	5.17	4.41	4.78	-14.70	- 7.54
15	6.31	4.84	6.16	- 23.30	- 2.38
20	6.30	5.04	6.20	- 20.90	- 1.59
30	5.89	5.82	5.90	- 1.19	0.17
40	7.71	7.19	7.62	- 6.74	- 1.17
50	7.80	8.35	7.93	7.05	1.67
60	6.88	6.45	6.86	- 6.25	- 0.29
70	6.50	6.35	6.35	- 2.31	- 2.31
80	8.77	7.12	8.43	-18.81	- 3.88
90	7.49	7.70	7.70	2.80	2.80
100	9.13	7.48	8.72	-18.07	- 4.48
110	9.35	7.69	9.21	- 17.75	- 1.50
120	9.34	7.45	9.06	- 20.24	- 3.0
130	8.65	7.63	8.52	- 11.80	- 1.50

พิจารณาค่า Maximum Path Length จากรูปที่ 4.10 และตารางที่ 4.6 พบว่าอัลกอริทึม p1 ให้ความยาวของวิถีที่ยาวที่สุดสั้นกว่าวิถีที่ได้จากอัลกอริทึม Jia ในหลายช่วงของจำนวนโนดปลายทาง สำหรับอัลกอริทึม p1 ให้ความยาววิถีที่ยาวที่สุดสั้นกว่าอัลกอริทึม Jia 7.32 % ถึง 27.40 % ในช่วงระหว่างจำนวนโนด 5 โนด ถึง 130 โนด ส่วนอัลกอริทึม p2 ให้ความยาววิถีที่ยาวที่สุดใกล้เคียงกับอัลกอริทึม Jia ในช่วงจำนวนโนดเดียวกัน ซึ่งในการจัดเส้นทางแบบมัลติคาสต์ความยาวของวิถีที่ยาวที่สุดสามารถใช้กำหนดขอบเขตของเวลาประวิง (delay) สำหรับการจัดเส้นทางสำหรับส่งข้อมูลมัลติคาสต์แบบเวลาจริง (real time) เนื่องจากในกรณีที่ค่า propagation delay มีค่ามากกว่าค่า delay

อื่น ๆ เช่น transmission delay และ queueing delay ค่า propagation delay จะแปรผันตามระยะทางและความเร็วข้อมูลที่ส่ง ทั้งนี้ถ้าเรากำหนดให้ระยะทางระหว่างโนดต่าง ๆ เท่ากันจะได้ว่า propagation delay จะแปรผันตามจำนวน hop [2] ของ link ที่ส่งผ่านข้อมูล ดังนั้นกล่าวได้ว่าอัลกอริทึม p1 โดยมากให้ค่าขอบเขตของเวลาประวิงน้อยกว่าที่ได้จากอัลกอริทึม Jia

พิจารณาค่าความยาววิถีเฉลี่ยในรูปที่ 4.11 และตารางที่ 4.7 อัลกอริทึม p1 ให้ค่าความยาววิถีเฉลี่ยน้อยกว่าอัลกอริทึม Jia 2.31 % ถึง 30.75 % ในช่วงโนดปลายทาง 5 โนดถึง 130 โนด อัลกอริทึม p2 ให้ค่าใกล้เคียงกับอัลกอริทึม Jia ในช่วงโนดปลายทางเดียวกัน แสดงว่าอัลกอริทึม p1 ให้ค่าเวลาประวิงเวลาเฉลี่ยของเส้นทางจากโนดผู้ส่งถึงโนดปลายทางน้อยกว่าอัลกอริทึม Jia

ค่า C_E, C_B และต้นทุนการสวิตช์ VP และ VC เป็นค่าสัมพัทธ์ไม่มีหน่วยเนื่องจากการละเลยการแทนค่าคงที่ K และ H ในสมการที่ (2.2) เมื่อใช้ virtual network ถ้าลองพิจาณา C_E, C_B และต้นทุนการสวิตช์ VP และ VC ของอัลกอริทึม p1, p2 และ Jia โดยกำหนดให้ต้นทุนที่ได้จากอัลกอริทึม Jia เท่ากับ

$$C = \alpha C_E + \beta C_B + \gamma(C_{vp}T_{vp} + C_{vc}T_{vc})$$

เมื่อ T_{vp} และ T_{vc} คือเวลาที่ใช้ในการสวิตช์ VP และ VC ของสวิตช์ตามลำดับ

ถ้าให้ T_{vp} และ T_{vc} เท่ากับ 1 จะได้ว่าต้นทุนของอัลกอริทึม p1 เท่ากับ

$$C_{p1} = \alpha\ (C_E + \%\delta_{CE}C_E\ ) + \beta\ (C_B + \%\delta_{CB}C_B) + \gamma\ (C_{vp} + \%\delta_{vp}C_{vp} + C_{vc} + \%\delta_{vc}C_{vc})$$

เมื่อ $\%\delta_{CE}$, $\%\delta_{CB}$, $\%\delta_{vp}$ และ $\%\delta_{vc}$ คือค่าความแตกต่างของต้นทุนแต่ละชนิดของ p1, p2 เทียบกับ Jia

$$C_{p1} - C = \alpha\ (\%\delta_{CE}C_E\ ) + \beta\ (\%\delta_{CB}C_B) + \gamma\ (\%\delta_{vp}C_{vp} + \%\delta_{vc}C_{vc})$$

พิจารณาที่ขนาดโนดปลายทางเท่ากับ 30 โนด ค่าในตารางที่ 4.1 - 4.4

C_{p1} - C = α (0.0767 x 30) + β (- 0.1047 x 46.8) + γ (- 0.4286 x 16.8 + 0.0742 x 31)

= 2.301α - 4.89996β - 4.90028γ

ถ้ากำหนดให้ α = 0.5, β = 0.5 และ γ = 0.1 ตามเอกสารอ้างอิง [10]

จะได้ว่า

C_{p1} - C = - 1.789508

เมื่อพิจารณาที่ขนาดโนดปลายทางเท่ากับ 40 โนด, 50 โนด และ 60 โนดจะได้ค่าดังตารางที่ 4.8

ตารางที่ 4.8 ต้นทุนของอัลกอริทึม p1 เทียบกับ Jia

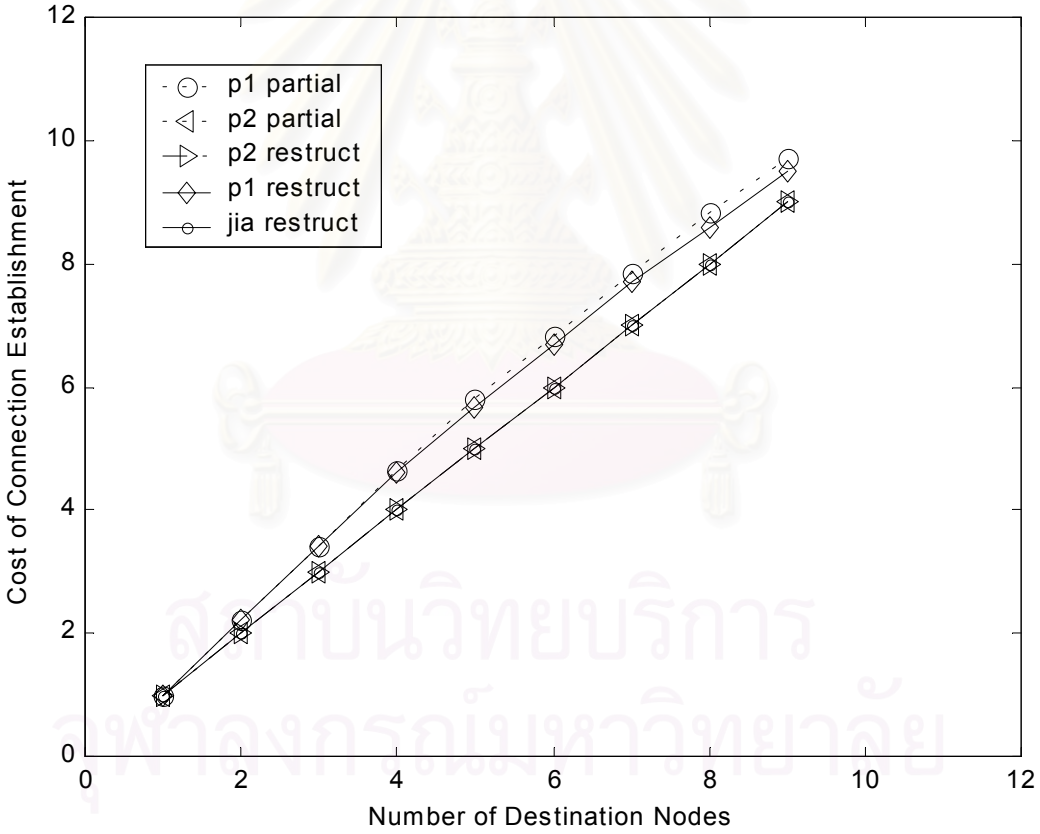
เมื่อกำหนดให้ α = 0.5, β = 0.5 และ γ = 0.1

ขนาดของจำนวนโนดปลายทาง	ค่าความแตกต่างของต้นทุน
40	- 2.110838
50	- 2.000452
60	- 1.560578

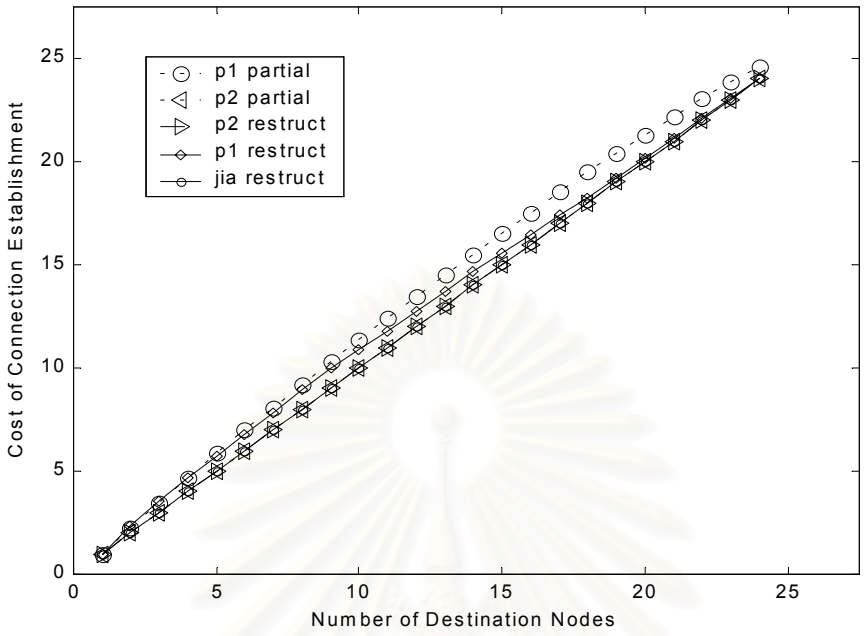
ซึ่งเป็นตัวอย่างที่แสดงให้เห็นว่าอัลกอริทึม p1 ให้ค่าต้นทุนการจัดเส้นทางแบบมัลติคาสต์น้อยกว่าอัลกอริทึม Jia หรือในกรณีที่แย่ที่สุดน่าจะได้ต้นทุนใกล้เคียงกัน

4.3 ผลการจำลองแบบการจัดเส้นทางแบบมัลติคาสต์ในโครงข่าย ATM กรณี dynamic multicast

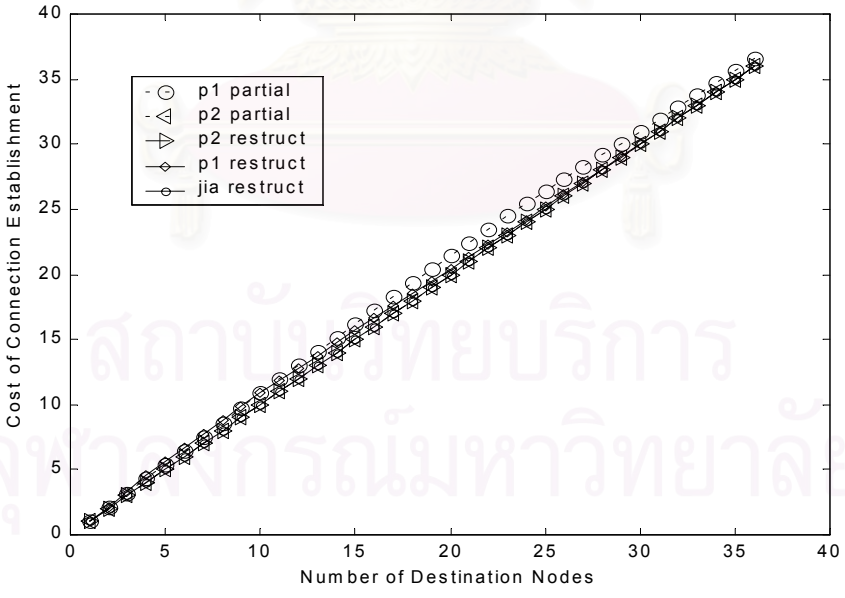
ในส่วนของการจัดเส้นทางแบบ dynamic multicast นั้น สมมติให้มีการร้องขอการต่อถึงกันกับกลุ่มมัลติคาสต์เพิ่มเข้ามาทีละ 1 การร้องขอ โดยจะเปรียบเทียบระหว่างอัลกอริทึม p1 partial, p2 partial, p1 restruct, p2 restruct, และ Jia restruct (อัลกอริทึม 3 ชุดสุดท้ายสมมติให้รู้ลำดับการร้องขอทั้งหมดและเป็นชุดเปรียบเทียบ) โดยพิจารณาค่าต้นทุนต่าง ๆ ดังนี้ C_E, C_B, ต้นทุนในการสวิตช์ VC และ VP และเวลาที่ใช้ในการคำนวณ โครงข่ายที่ใช้จำลองแบบมีขนาด 20, 40 และ 60 โนด



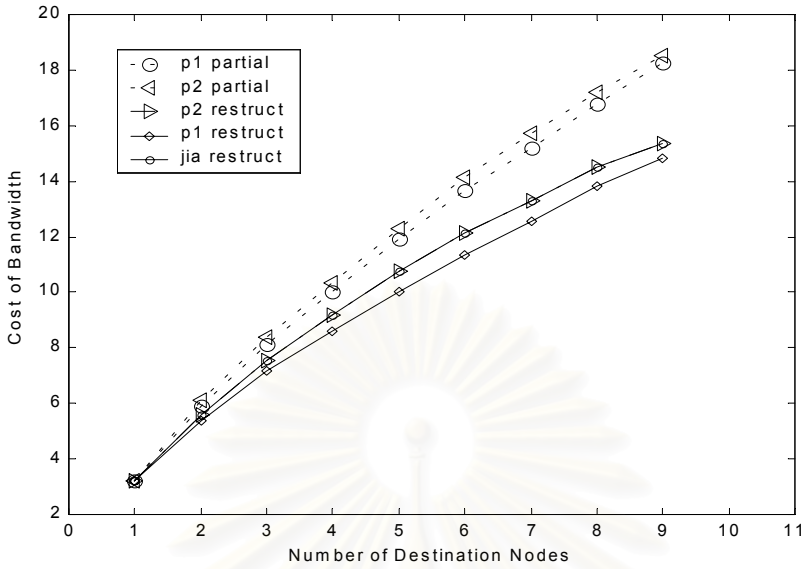
รูปที่ 4.12 กราฟแสดงผลของต้นทุนการสร้างการต่อถึงกัน (C_E) ของการจัดเส้นทางแบบ dynamic multicast บนโครงข่ายขนาด 20 โนด เปรียบเทียบระหว่างอัลกอริทึม p1 partial, p2 partial, p1 restruct, p2 restruct และ Jia restruct



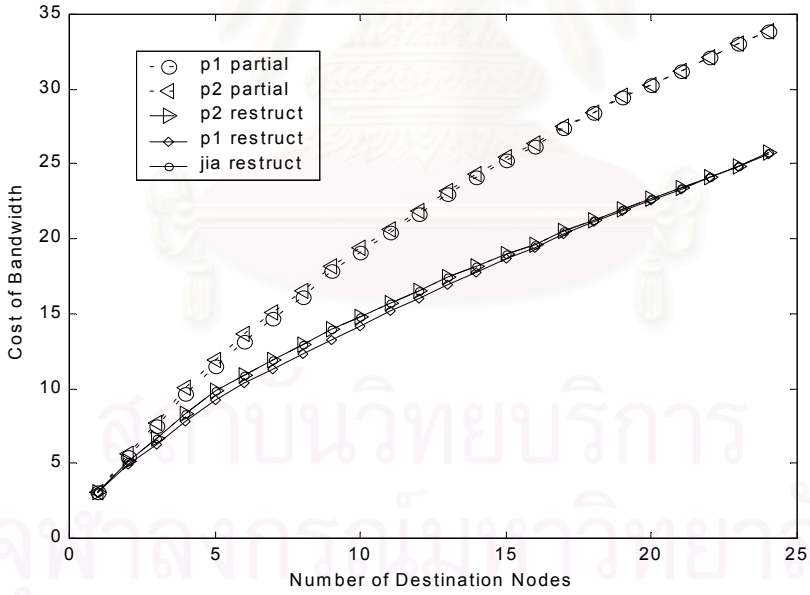
รูปที่ 4.13 กราฟแสดงผลของต้นทุนการสร้างการต่อถึงกัน (C_E) ของการจัดเส้นทางแบบ dynamic multicast บนโครงข่ายขนาด 40 โนดเปรียบเทียบระหว่างอัลกอริทึม p1 partial, p2 partial, p1 restruct, p2 restruct และ Jia restruct



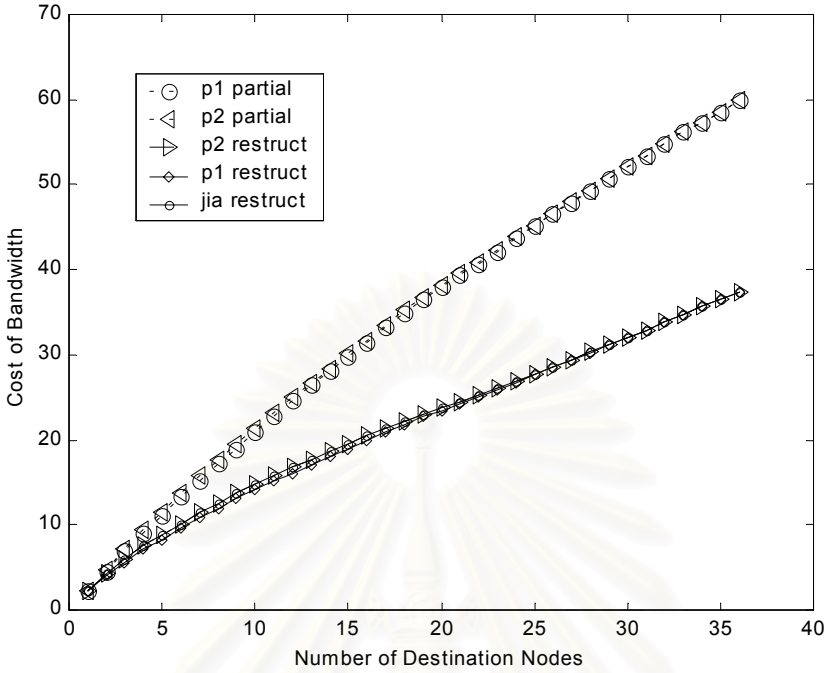
รูปที่ 4.14 กราฟแสดงผลของต้นทุนการสร้างการต่อถึงกัน (C_E) ของการจัดเส้นทางแบบ dynamic multicast บนโครงข่ายขนาด 60 โนด เปรียบเทียบระหว่างอัลกอริทึม p1 partial, p2 partial, p1 restruct, p2 restruct และ Jia restruct



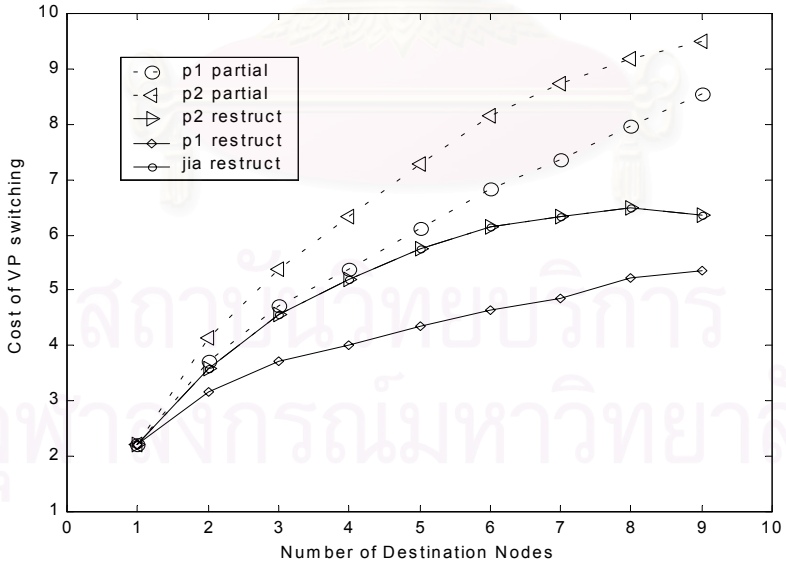
รูปที่ 4.15 กราฟแสดงผลของต้นทุนของแบนด์วิดท์ (C_B) ของการจัดเส้นทางแบบ dynamic multicast บนโครงข่ายขนาด 20 โนด เปรียบเทียบระหว่างอัลกอริทึม p1 partial, p2 partial, p1 restruct, p2 restruct และ Jia restruct



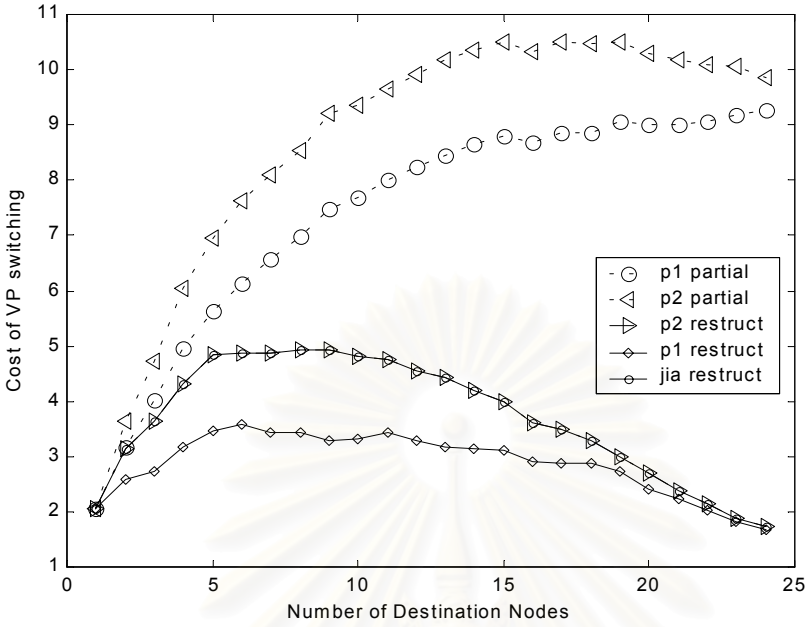
รูปที่ 4.16 กราฟแสดงผลของต้นทุนของแบนด์วิดท์ (C_B) ของการจัดเส้นทางแบบ dynamic multicast บนโครงข่ายขนาด 40 โนด เปรียบเทียบระหว่างอัลกอริทึม p1 partial, p2 partial, p1 restruct, p2 restruct และ Jia restruct



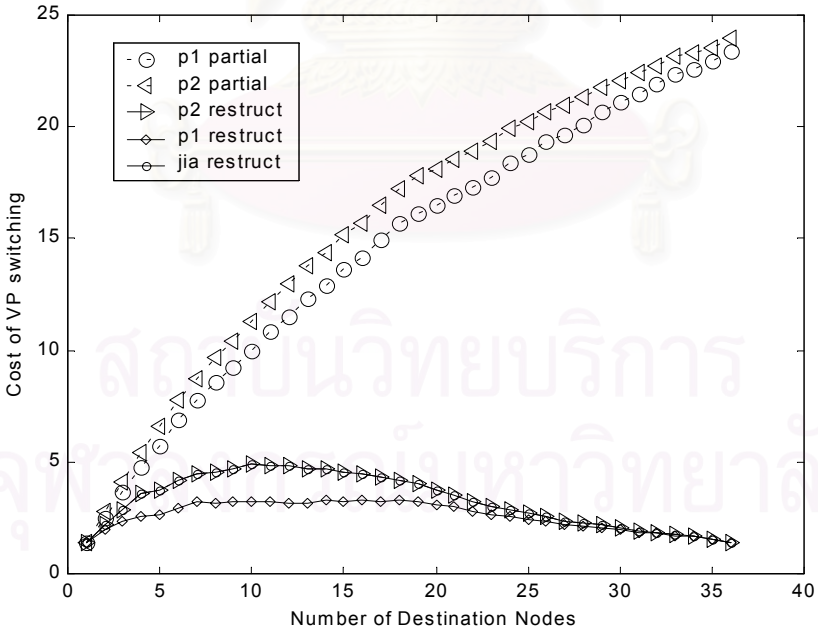
รูปที่ 4.17 กราฟแสดงผลของต้นทุนของแบนด์วิดท์ (C_B) ของการจัดเส้นทางแบบ dynamic multicast บนโครงข่ายขนาด 60 โนด เปรียบเทียบระหว่างอัลกอริทึม p1 partial, p2 partial, p1 restruct, p2 restruct และ Jia restruct



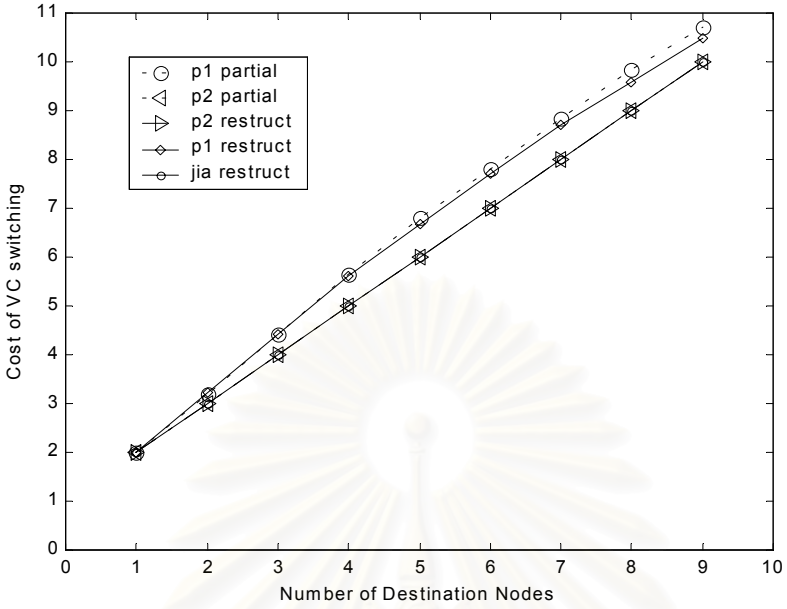
รูปที่ 4.18 กราฟแสดงผลของต้นทุนของการสวิตช์ VP ของการจัดเส้นทางแบบ dynamic multicast บนโครงข่ายขนาด 20 โนด เปรียบเทียบระหว่างอัลกอริทึม p1 partial, p2 partial, p1 restruct, p2 restruct และ Jia restruct



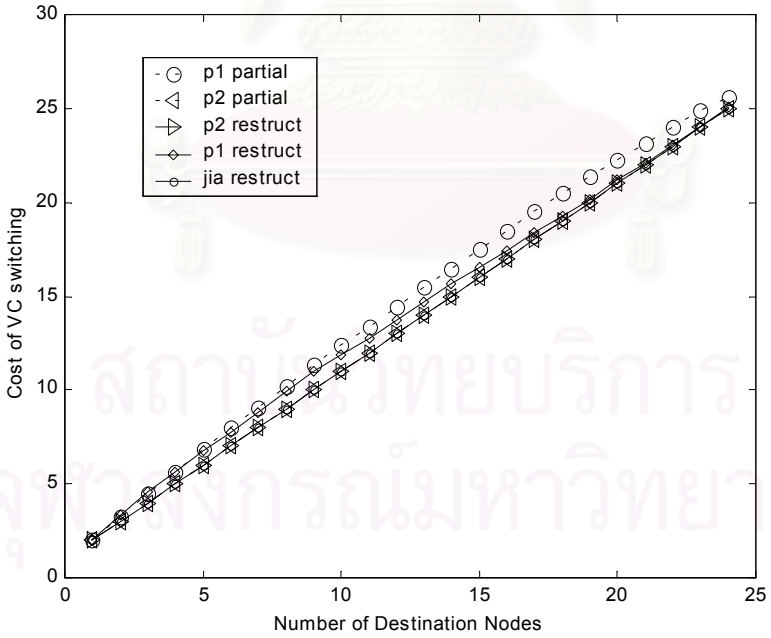
รูปที่ 4.19 กราฟแสดงผลของต้นทุนของการสวิตช์ VP ของการจัดเส้นทางแบบ dynamic multicast บนโครงข่ายขนาด 40 โนด เปรียบเทียบระหว่างอัลกอริทึม p1 partial, p2 partial, p1 restruct, p2 restruct และ Jia restruct



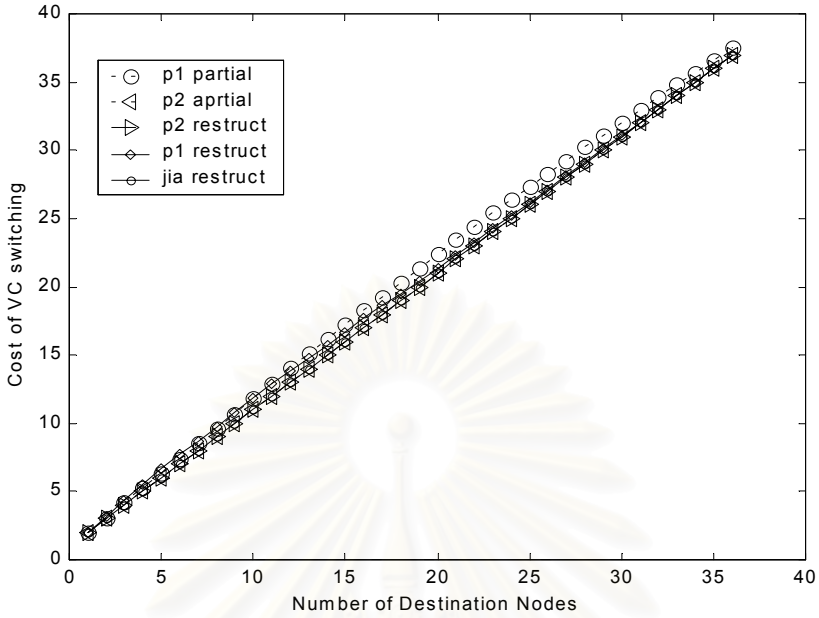
รูปที่ 4.20 กราฟแสดงผลของต้นทุนของการสวิตช์ VP ของการจัดเส้นทางแบบ dynamic multicast บนโครงข่ายขนาด 60 โนด เปรียบเทียบระหว่างอัลกอริทึม p1 partial, p2 partial, p1 restruct, p2 restruct และ Jia restruct



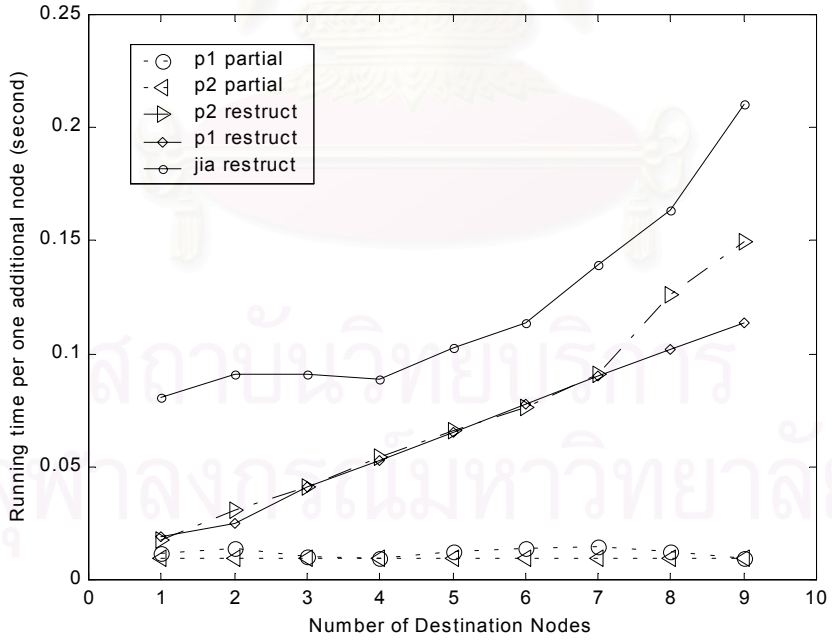
รูปที่ 4.21 กราฟแสดงผลของต้นทุนของการสวิตช์ VC ของการจัดเส้นทางแบบ dynamic multicast บนโครงข่ายขนาด 20 โนด เปรียบเทียบระหว่างอัลกอริทึม p1 partial, p2 partial, p1 restruct, p2 restruct และ Jia restruct



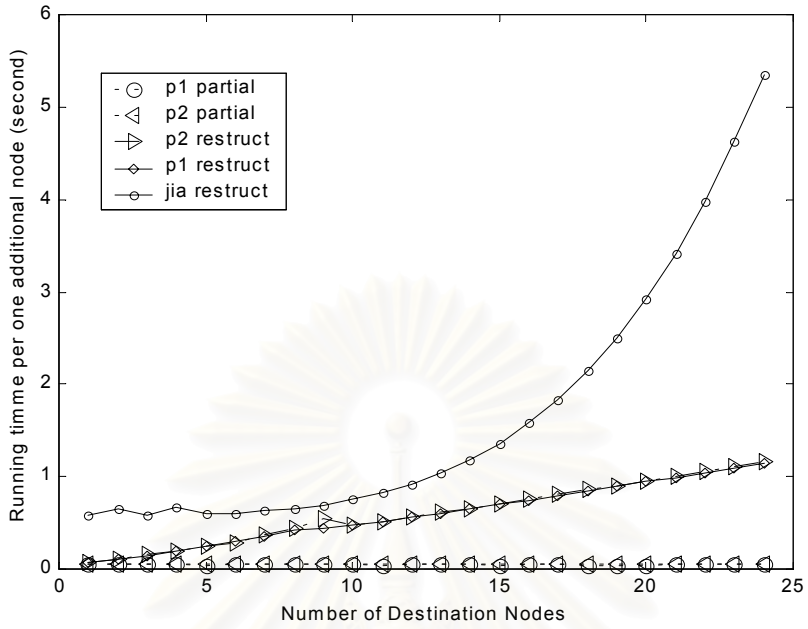
รูปที่ 4.22 กราฟแสดงผลของต้นทุนของการสวิตช์ VC ของการจัดเส้นทางแบบ dynamic multicast บนโครงข่ายขนาด 40 โนด เปรียบเทียบระหว่างอัลกอริทึม p1 partial, p2 partial, p1 restruct, p2 restruct และ Jia restruct



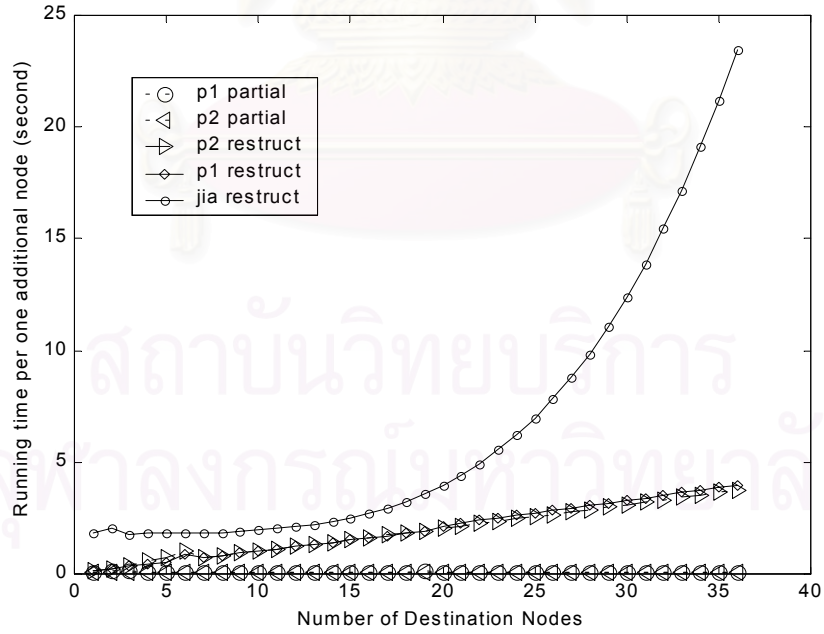
รูปที่ 4.23 กราฟแสดงผลของต้นทุนของการสวิตช์ VC ของการจัดเส้นทางแบบ dynamic multicast บนโครงข่ายขนาด 60 โนดเปรียบเทียบระหว่างอัลกอริทึม p1 partial, p2 partial, p1 restruct, p2 restruct และ Jia restruct



รูปที่ 4.24 กราฟแสดงเวลาในการคำนวณของอัลกอริทึมการจัดเส้นทางแบบ dynamic multicast บนโครงข่ายขนาด 20 โนดเปรียบเทียบระหว่างอัลกอริทึม p1 partial, p2 partial, p1 restruct, p2 restruct และ Jia restruct



รูปที่ 4.25 กราฟแสดงเวลาในการคำนวณของอัลกอริทึมการจัดเส้นทางแบบ dynamic multicast บนโครงข่ายขนาด 40 โนด เปรียบเทียบระหว่างอัลกอริทึม p1 partial, p2 partial, p1 restruct, p2 restruct และ Jia restruct



รูปที่ 4.26 กราฟแสดงเวลาในการคำนวณของอัลกอริทึมการจัดเส้นทางแบบ dynamic multicast บนโครงข่ายขนาด 60 โนดเปรียบเทียบระหว่างอัลกอริทึม p1 partial, p2 partial, p1 restruct, p2 restruct และ Jia restruct

4.4 วิเคราะห์ผลการจำลองแบบการจัดเส้นทางมัลติคาสต์บนโครงข่าย ATM ในกรณี dynamic multicast

4.4.1 ต้นทุนการต่อถึงกัน (C_E)

พิจารณารูปที่ 4.12 ถึง 4.14 ต้นทุนการต่อถึงกัน (C_E) ของอัลกอริทึม p1 partial มากกว่า p2 partial, p1 restruct, p2 restruct และ Jia restruct ทั้งนี้เนื่องจากต้องทำการจัดเส้นทางทันทีที่มีการร้องขอการต่อเข้ามา ไม่สามารถเลือกได้ว่าควรจะให้โนดปลายทางตัวใดต่อเข้ากับ multicast tree ก่อนกัน จึงทำให้ไม่สามารถใช้ connection ร่วมกันได้มากพอ นอกจากนี้ยังมีผลของ Steiner node ที่ใช้ ส่งผลให้ค่า C_E สูงกว่าอัลกอริทึมที่ใช้เปรียบเทียบทั้ง 4 ชุด

4.4.2 ต้นทุนของแบนด์วิดท์ (C_B)

พิจารณาต้นทุนของแบนด์วิดท์บนโครงข่ายขนาด 20 โนด 40 โนด และ 60 โนด ดังรูปที่ 4.15 ถึง 4.17 ต้นทุนแบนด์วิดท์จาก p1 partial และ p2 partial มีแนวโน้มเพิ่มมากขึ้นเมื่อจำนวนโนดปลายทางเพิ่มขึ้น สาเหตุที่อัลกอริทึมทั้ง 2 ให้ต้นทุนแบนวิดท์มากกว่าชุดเปรียบเทียบทั้งสามเป็นเหตุผลเดียวกันกับต้นทุนการต่อถึงกัน เพราะต้องมีการต่อของ VPC มากขึ้นและมีวิถี (path) ของ VPC ที่ใช้ร่วมกันได้น้อยลง ทำให้สิ้นเปลืองแบนด์วิดท์มากขึ้น โดยเฉพาะเมื่อขนาดของโครงข่ายเพิ่มเป็น 40 โนดและ 60 โนด p1 partial และ p2 partial ให้ค่าผิดไปจาก p1 restruct มาก โดยมากกว่า p1 restruct ถึง 62.2 % ที่จำนวนโนดปลายทาง 36 โนด บนโครงข่ายขนาด 60 โนด และให้ค่ามากกว่า 31.89 % ที่จำนวนโนดปลายทาง 24 โนดบนโครงข่ายขนาด 40 โนดซึ่งที่จุดเดียวกันอัลกอริทึม Jia restruct, p1 restruct และ p2 restruct ให้ค่าต่างกันเป็นศูนย์ สำหรับโครงข่ายขนาด 20 โนดที่จำนวนโนดปลายทางจำนวน 9 โนด อัลกอริทึม p1 partial ให้ค่าต่างจากอัลกอริทึม Jia restruct และ p2 restruct เพียง 18.18 %

4.4.3 **ต้นทุนการสวิตช์** VC

พิจารณาต้นทุนการสวิตช์ VC ในรูปที่ 4.21 ถึง 4.23 ต้นทุนการสวิตช์ของ p1 partial สูงกว่า p1 restruct เพราะมีการสร้าง connection ของ VPC ที่มากขึ้นดังเหตุผลข้างต้น ทำให้เกิด VC switching ที่ปลายของ VPC เพิ่มมากขึ้นตามไปด้วย อย่างไรก็ตามต้นทุนการสวิตช์ VC ของอัลกอริทึม p1 partial แตกต่างจาก p1 restruct น้อย อย่างไรก็ตามขนาดของโครงข่ายที่เหมาะสมสำหรับอัลกอริทึม p1 partial และ p2 partial ยังต้องพิจารณาต้นทุนส่วนอื่น ๆ เป็นองค์ประกอบเพิ่มเติม

4.4.4 **ต้นทุนการสวิตช์** VP

พิจารณาต้นทุนการสวิตช์ VP ดังรูปที่ 4.18 ถึง 4.20 พบว่าค่าที่ได้จาก p1 partial มากกว่าค่าที่ได้จากอัลกอริทึมชุดเปรียบเทียบทั้งสาม (p1 restruct, p2 restruct และ Jia restruct) โดยกราฟที่ได้มีแนวโน้มเพิ่มขึ้นเรื่อย ๆ ไม่ลดลงเหมือนดังเช่นกรณีของ static multicast ทั้งนี้เพราะในกรณี static multicast เมื่อเพิ่มจำนวนโนดปลายทางถึงค่า ๆ หนึ่งจำนวน VC switch ที่เคยทำเฉพาะ VP switching จะกลายเป็นโนดปลายทางมากขึ้นทำให้ต้นทุนการสวิตช์ VC สูงขึ้นขณะเดียวกันต้นทุนการสวิตช์ VP จะลดลง แต่เนื่องจากกรณีที่เป็น dynamic multicast นี้ ต้องมีการสร้าง VPC จำนวนมากกว่าที่ได้จากวิธี static multicast เดิมดังรูปที่ 4.12 ถึง 4.14 และมีการใช้ VPC ร่วมกันน้อยลง ทำให้มี VC switch จำนวนมากทำ VP switching เท่านั้นในการจัดเส้นทางจากต้นทางไปยังปลายทาง จากรูปที่ 4.19 และ 4.20 ค่าต้นทุนการสวิตช์ VP ของอัลกอริทึม p1 partial และ p2 partial สำหรับโครงข่ายขนาด 40 โนด และ 60 โนดให้ค่าแตกต่างจากอัลกอริทึม Jia restruct, p1 restruct และ p2 restruct มาก แม้ว่าจำนวนโนดปลายทางจะมีค่าน้อยมากเช่น 5 โนด ดังนั้นจากผลของโครงข่ายที่นำมาจำลองแบบแสดงว่าอัลกอริทึม p1 partial และ p2 partial ไม่เหมาะสำหรับโครงข่ายขนาด 40 โนดขึ้นไป สำหรับโครงข่ายขนาด 20 โนด p1 partial มีค่าต้นทุนการสวิตช์ VP มากกว่า Jia restruct 36.3 % เมื่อพิจารณาร่วมกับค่า C_E ซึ่งมากกว่าอัลกอริทึมเปรียบเทียบทั้งสามเพียง 6.5 % และมีค่าต้นทุนการสวิตช์ VC มากกว่าทั้ง 3 อัลกอริทึม (p2 partial, p2 restruct และ Jia restruct) 7.5 % ที่จำนวนโนดปลายทาง 9 โนด ดังนั้นโครงข่ายขนาด 20 โนด น่าจะมีความเหมาะสมสำหรับอัลกอริทึม p1 partial และ p2 partial มากกว่า

4.4.5 เวลาที่ใช้ในการจัดเส้นทาง

ตารางที่ 4.9 ค่า complexity ของอัลกอริทึม p1 partial, p2 partial, Jia restruct, p1 restruct, p2 restruct

อัลกอริทึม	Complexity
p1 partial	$O(n^3)$
p2 partial	$O(n^3)$
Jia restruct	$O(n^3 + m\log m)$
p1 restruct	$O(n^3)$
p2 restruct	$O(n^3)$

ในส่วนของเวลาที่ใช้ในการจัดเส้นทางของอัลกอริทึมนั้นพบว่า อัลกอริทึม p1 partial และ p2 partial นั้นใช้เวลาน้อยมากและเกือบจะคงที่ เนื่องจากเป็นการสร้าง partial tree ที่ไม่ขึ้นกับจำนวนโนดปลายทางอื่น ๆ ที่เพิ่มเข้าไปยังกลุ่มมัลติคาสต์ทั้งที่เพิ่มเข้าไปก่อนและที่ยังไม่เพิ่มเข้ามา ขณะที่อัลกอริทึมชุดเปรียบเทียบทั้งสามที่ใช้ในกรณี static multicast มีเวลาเพิ่มขึ้นเมื่อเพิ่มขนาดของโนดปลายทางเพราะจะมีการจัดเส้นทางใหม่ทั้งหมด และอัลกอริทึมที่เสนอโดย Jia จะใช้เวลาค้นหาเส้นทางนานที่สุด รองลงมาคือ p1 และ p2 ตามลำดับ ซึ่งสอดคล้องกับค่าความซับซ้อนที่ได้จากตารางที่ 4.9

ทั้งนี้วัตถุประสงค์ของการนำอัลกอริทึม p1 และ p2 มาดัดแปลงใช้ในกรณีที่เป็น dynamic multicast นั้นเพื่อชี้ให้เห็นถึงกรณีที่หากนำอัลกอริทึมที่ไม่เหมาะสมที่ต้องจัดเส้นทางใหม่ทั้งหมดมาใช้ใน dynamic multicast จะทำให้สิ้นเปลืองเวลาคำนวณและรบกวนการต่อเดิม และใช้เป็นแนวทางในการออกแบบอัลกอริทึมสำหรับ dynamic multicast ต่อไป

บทที่ 5
สรุปผลการจำลองแบบและข้อเสนอแนะ

5.1 สรุปผลการจำลองแบบ

งานวิจัยนี้ได้นำเสนออัลกอริทึมการจัดเส้นทางแบบมัลติคาสต์บนโครงข่าย ATM ซึ่งมีพื้นฐานมาจากการจัดเส้นทางแบบจุดถึงจุด (point - to - point) ที่ใช้อัลกอริทึม shortest path เป็นหลัก โดยเปรียบเทียบกับอัลกอริทึมของ Jia ที่ใช้แนวคิด spanning tree จากการจำลองแบบสามารถสรุปผลการจำลองแบบของอัลกอริทึม p1 และ p2 เปรียบเทียบกับอัลกอริทึม Jia ในกรณี static multicast ดังตารางที่ 5.1 โดยเปรียบเทียบกับอัลกอริทึมของ Jia ในรูปของ $\frac{C_{X,algor} - C_{X,Jia}}{C_{X,jia}} \times 100$ เมื่อ $C_{X,algor}$ คือ ต้นทุนหรือค่าที่สนใจ X ของอัลกอริทึม algor เทียบกับอัลกอริทึม Jia

5.1.1 กรณี static multicast สรุปผลได้ดังตารางที่ 5.1

ตารางที่ 5.1 เปรียบเทียบต้นทุนชนิดต่าง ๆ และความยาววิถีที่ใช้ในการจัดเส้นทาง
ในกรณี static multicast ของอัลกอริทึม p1 และ p2 เทียบกับอัลกอริทึม Jia

$\frac{C_{X,algor} - C_{X,Jia}}{C_{X,Jia}} \times 100$ (%)	อัลกอริทึม p1	อัลกอริทึม p2
ต้นทุนของแบนด์วิดท์ (C_B) ในช่วง $\|D\|$ = 5 ถึง 110 โนด	- 0.35 % ถึง -11.31 %	0 %
ต้นทุนของการต่อถึงกัน (C_E) ในช่วง $\|D\|$ = 5 ถึง 110 โนด	0.09 % ถึง 24 %	0 %
ต้นทุนของการสวิตช์ VC ในช่วง $\|D\|$ = 5 ถึง 110 โนด	0.09 % ถึง 20 %	0%
ต้นทุนของการสวิตช์ VP ในช่วง $\|D\|$ = 5 ถึง 110 โนด	- 13.51 % ถึง - 45.16 %	0 %

$\frac{C_{X,algor}-C_{X,Jia}}{C_{X,Jia}}\times 100$ (%)	อัลกอริทึม p1	อัลกอริทึม p2
ความยาววิถีเฉลี่ย (L_{av}) ในช่วง $\|D\|$ = 5 ถึง 130 โนด	- 1.19 % ถึง - 30.75 %	- 0.29 % ถึง - 7.54 %
ความยาววิถีสูงสุด (L_{max}) ในช่วง $\|D\|$ = 5 ถึง 130 โนด	- 7.32 % ถึง -27.40 %	- 1.36 % ถึง - 10.75 %

หมายเหตุ เมื่อ $|D|$ คือจำนวนโนดปลายทางที่ใช้ในการจัดเส้นทาง และค่า $\frac{C_{X,algor}-C_{X,Jia}}{C_{X,Jia}}\times 100$ ที่ติดลบหมายความว่าอัลกอริทึม X มีค่าต้นทุนหรือปริมาณที่สนใจเปรียบเทียบน้อยกว่าอัลกอริทึม Jia

ตารางที่ 5.2 ขอบเขตของเวลาที่ใช้ในการคำนวณของอัลกอริทึม p1, p2 และ Jia

อัลกอริทึม	Complexity
Jia	$O(n^3 + m\log m)$
p1	$O(n^3)$
p2	$O(n^3)$

หมายเหตุ n คือจำนวนโนดทั้งหมดบนโครงข่าย และ m คือจำนวน links ทั้งหมดของโครงข่าย

สรุปว่าอัลกอริทึม p1 และ p2 ให้ต้นทุนการจัดเส้นทางที่ใกล้เคียงหรือน้อยกว่าอัลกอริทึม Jia แต่ใช้เวลาในการคำนวณเส้นทางมัลติคาสต์ต่ำกว่า ดังแสดงในตารางที่ 5.2 ดังนั้นจึงมีความเหมาะสมในการจัดเส้นทางมากกว่าในกรณี static multicast นอกจากนี้ในตารางที่ 5.1 อัลกอริทึม p1 ยังมีความยาวของวิถีสั้นกว่าเส้นทางที่ใช้อัลกอริทึม Jia ทำให้การส่งข้อมูลใช้เวลาประวิงน้อยกว่าในกรณีที่ส่งทราฟฟิกข้อมูลแบบ real time

5.1.2 กรณี dynamic multicast

ในการจัดเส้นทางแบบ dynamic multicast ที่ดัดแปลงจากอัลกอริทึม p1 และ p2 มาเป็นอัลกอริทึม p1 partial และ p2 partial จากการจำลองแบบสรุปได้ว่าขนาดของโครงข่ายที่เหมาะสมสำหรับการใช้งานอัลกอริทึม p1 partial และ p2 partial คือ โครงข่ายขนาด 20 โนด และอัลกอริทึม p1 partial ให้ค่าต้นทุน C_E, C_B และ C_S มากกว่าอัลกอริทึมที่ใช้เปรียบเทียบทั้งสามชุดคือ Jia restruct, p1 restruct และ p2 restruct ซึ่งทั้ง 3 ชุดดัดแปลงมาจากอัลกอริทึมปกติที่ใช้ใน static multicast เพื่อเหตุผลในการเปรียบเทียบ แต่ในสภาพใช้งานจริงของ dynamic multicast อัลกอริทึม p1 partial และ p2 partial จะมีความเหมาะสมต่อสภาพในการใช้งานมากกว่า เนื่องจากมีลักษณะที่รองรับการใช้งานในสถาวะพลวัตมากกว่า และมีเวลาในการหาเส้นทางน้อยที่สุดเมื่อเทียบกับวิธีที่ทำการจัดเส้นทางใหม่ทั้งหมด ดังรูปที่ 4.24 - 4.26

5.2 ลำดับความสำคัญของต้นทุนแต่ละชนิด (C_B, C_E และ C_S)

ในทางปฏิบัติ การจัดเส้นทางแบบมัลติคาสต์ต้องพิจารณาชนิดของข้อมูลที่จะทำการส่ง ถ้าข้อมูลที่ส่งเป็นข้อมูล non real time ต้นทุนที่มีความสำคัญมากที่สุดในการจัดเส้นทางคือต้นทุนของแบนด์วิดท์ [3] แต่ถ้าทำการส่งข้อมูลแบบ real time ต้นทุนที่สำคัญที่สุดจะเป็นเวลาประวิง (delay) [3] รองลงมาคือต้นทุนของแบนด์วิดท์ ดังนั้นการจัดเส้นทางแบบมัลติคาสต์บนโครงข่าย ATM ก็เช่นเดียวกัน แต่มีความแตกต่างที่ต้นทุนของการสวิตช์วิถีเสมือน (ซึ่งก่อให้เกิดเวลาประวิงเช่นเดียวกับการสวิตช์ช่องสัญญาณเสมือน) ไม่เป็นอิสระจากต้นทุนของแบนด์วิดท์ดังสมการที่ (2.6), (2.7), (2.10) และ (2.11) ขณะเดียวกันถ้าเราสามารถลดการสร้างการต่อถึงกันลงไปได้ก็น่าจะลดการใช้แบนด์วิดท์ลงไปได้ แต่ผลจากการใช้ Steiner nodes พบว่า การต่อถึงกันที่เพิ่มขึ้นเนื่องจาก Steiner nodes กลับทำให้มีการใช้งานแบนด์วิดท์ลดลง และให้ผลต้นทุนของการสวิตช์ช่องสัญญาณเสมือนเพิ่มขึ้นเล็กน้อย จึงสรุปได้ว่าลำดับความสำคัญของต้นทุนในการส่งข้อมูลแบบมัลติคาสต์สำหรับโครงข่าย ATM ทั้ง static multicast และ dynamic multicast ในกรณีที่ส่งข้อมูลที่มีคุณสมบัติ delay sensitive (ได้แก่ทราฟฟิกแบบ real time) ต้นทุนการสวิตช์วิถีเสมือนและช่องสัญญาณเสมือนมีความสำคัญอันดับแรกเนื่องจากกำหนดขนาดของ delay ที่เกิดขึ้นบนเส้นทางโดยตรง อันดับต่อมาคือต้นทุนของแบนด์วิดท์ แต่สำหรับข้อมูลที่ไม่มีคุณสมบัติ delay sensitive ต้นทุนของแบนด์วิดท์มีความสำคัญอันดับแรก และรองลงมาคือต้นทุนของการสวิตช์และต้นทุนของการสร้างการต่อถึงกัน ทั้งนี้ต้องพิจารณาถึงความสัมพันธ์ที่ไม่เป็นอิสระกัน

ของต้นทุนทั้ง 3 ชนิดหลังจากที่มีการใช้ Steiner node ประกอบ ในวิทยานิพนธ์นี้ให้ความสำคัญของต้นทุนของแบนด์วิดท์เป็นอันดับแรก เนื่องจากไม่สนใจขีดจำกัดของ delay ในการหาเส้นทาง รองลงมาให้ความสำคัญแก่ต้นทุนในการสวิตช์ และต้นทุนในการสร้างการต่อถึงกันมีความสำคัญเป็นอันดับสุดท้าย

5.3 ข้อดีและข้อเสียของอัลกอริทึมที่เสนอ

5.3.1 ข้อดี

1. อัลกอริทึม p1 และ p2 สามารถใช้งานร่วมกับการจัดเส้นทางแบบจุดถึงจุดของโครงข่าย ATM ได้เนื่องจากใช้อัลกอริทึมในการหา shortest path ชุดเดียวกัน
2. อัลกอริทึม p1 และ p2 ให้ผลต้นทุนในการจัดเส้นทางใกล้เคียงกับอัลกอริทึม Jia แต่มีความซับซ้อนของการทำงานน้อยกว่า
3. อัลกอริทึม p1 partial และ p2 partial สามารถดัดแปลงมาจากอัลกอริทึม p1 และ p2 ได้ง่ายเพื่อนำมาใช้ในการจัดเส้นทางแบบ dynamic multicast

5.3.2 ข้อเสีย

1. อัลกอริทึม p1 และ p2 ไม่เหมาะกับโครงข่ายแนวราบที่ มีขนาดของโครงข่ายใหญ่มาก ๆ เนื่องจากการหา shortest path จำเป็นต้องรู้พารามิเตอร์ที่ใช้ในการจัดเส้นทางจากทุก ๆ โนด
2. อัลกอริทึม p1 partial และ p2 partial มีความเหมาะสมในการใช้งานเฉพาะในโครงข่ายที่มีขนาดเล็ก หรือโครงข่ายที่เป็นลำดับชั้น (Hierarchical network) สำหรับโครงข่ายขนาดใหญ่จะให้ค่าด้อยกว่าอัลกอริทึมที่ใช้ในกรณี static multicast มาก

5.4 ข้อเสนอแนะ

1. อัลกอริทึมที่เสนอเป็นการแก้ปัญหาการจัดเส้นทางแบบมัลติคาสต์บนโครงข่าย ATM ที่ยังไม่ได้คำนึงถึงเงื่อนไขบังคับในเรื่องของคุณภาพของการให้บริการ (Quality of service, QOS) เช่น เวลาประวิงจากต้นทางถึงปลายทาง จึงเหมาะสมกับบริการการส่งข้อมูลที่ไม่เป็นเวลาจริง (non real time) ดังนั้นน่าจะมีการพิจารณาเงื่อนไขบังคับเพื่อให้ครอบคลุมการจัดเส้นทางแบบมัลติคาสต์สำหรับข้อมูลที่เป็นเวลาจริง

2. อัลกอริทึมที่เสนอชี้ให้เห็นว่าการนำ Steiner node เข้ามาช่วยจัดเส้นทางทำให้สิ้นเปลืองต้นทุนของแบนด์วิดท์น้อยลง ดังนั้นน่าจะมีการศึกษาว่าแบนด์วิดท์ที่ลดลงขึ้นกับจำนวนของ Steiner node ที่ใช้เพียงตัวแปรเดียวหรือไม่

3. การจัดเส้นทางในกรณีของ dynamic multicast พิจารณาเพียงกรณีที่มีการร้องขอการต่อถึงกันเข้ากับ multicast tree ที่มีอยู่ก่อนไม่ได้พิจารณาเมื่อมีการร้องขอออกจาก multicast tree ดังนั้นจึงควรมีการออกแบบอัลกอริทึมสำหรับรองรับการจัดเส้นทางเมื่อมีการร้องขอออกจากกลุ่มมัลติคาสต์

รายการอ้างอิง

1. X. Jia, C. H. Lee, J. M. Ng and E. Chan, " Routing Multicast Connections with Optimal Network Cost in ATM Networks, "**Computer Communication and Network Proceeding IEEE**, 1995: 66-71.
2. Z. Dziong. **ATM Network Resource Management.**, (n.p.): McGraw-Hill, 1997.
3. H. S. Laxman and B. Mukherjee, " Multicast Routing Algorithms and Protocols : A Tutorial, "**IEEE Network**, 2000: 90-102
4. B. M. Waxman, " Routing of Multipoint Connections, "**IEEE Journal on Selected Area in Communications** Vol. 6, No. 9 (December, 1998) :1617-1622.
5. P. Winter, " Steiner Problem in Networks : A Survey, "**Networks** Vol. 17 (1987) :129-167.
6. Q. Sun and H. Langendofer, " Routing Multipoint Connection in VP-based ATM Networks, "**Communication technology Proceedings** 1996, ICCT'96 :1-4.
7. H. Tode, Y. Sakai, M. Yamamoto and H. Okada, " Multicast Routing Schemes in ATM, " **International Journal of Communication Systems** Vol. 9, 1996 :185-196.
8. M. H. Ammar, S. Y. Cheung and C. M. Scoglio, " Routing Multipoint Connections Using Virtual Paths in an ATM Network, " **INFOCOMM'93 Proceeding**, IEEE, (1993) : 95-105.
9. T. H. Cormen and C. E. Leiserson. **Introduction to Algorithms** 23 ed. (n.p.): McGraw-Hill, 1998.
10. B. H. Ryu, M. Murata and H. Miyahara, " A Dynamic Application - Oriented Multicast Routing for Virtual - Path Based ATM Networks, "**IEICE Transaction on Communication** Vol. E80-B, No. 11, (November, 1998), : 1654-1663.
11. A. Kershenbaum. **Telecommunications Network Design Algorithms.**, (n.p.): McGraw-Hill, 1993.

ภาคผนวก

ภาคผนวก ก

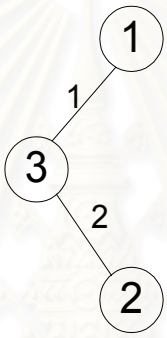
การคำนวณหาระยะห่างระหว่างโนด u และ v บนระนาบคาร์ทีเซียน

หลังจากการกระจายโนดต่าง ๆ ลงบนระนาบคาร์ทีเซียนโดยใช้การแจกแจงความน่าจะเป็นแบบ uniform distribution สมมติว่าโนด u มีพิกัดเป็น (x_1, y_1) และโนด v มีพิกัดเป็น (x_2,y_2) ระยะห่างระหว่างโนด u และ v คำนวณได้จาก

$$d(u,v) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \qquad \text{(ก.1)}$$

ภาคผนวก ข
การตรวจสอบการต่อถึงกันของกราฟ

หลังจากที่มีการพิจารณาระยะทางระหว่างโนดคู่ต่าง ๆ และมีการพิจารณาข่ายเชื่อมโยงตามค่าความน่าจะเป็นที่เสนอโดย Waxman [4] ในขั้นตอนการตรวจสอบว่าโครงข่ายที่ได้จากการสร้าง random graph นั้นเป็น connected graph หรือไม่ กล่าวคือตรวจสอบว่าเป็นโครงข่ายที่ต่อถึงกันหมดหรือไม่ สมมติให้โครงข่ายที่ได้แทนได้ด้วยเมตริกซ์ (adjacency matrix) และมีโครงข่ายแสดงดังรูปที่ ข.1



รูปที่ ข.1 โครงข่ายที่ทดสอบการต่อถึงกัน

โครงข่ายดังรูปที่ ข.1 แทนด้วยเมตริกซ์ $A = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 2 \\ 1 & 2 & 0 \end{bmatrix}$ ที่มีโนด 1 ต่ออยู่กับโนด 3 แต่ไม่ต่อกับโนด 2 และโนด 2 ต่ออยู่กับโนด 3 เมื่อนำเมตริกซ์ A มายกกำลังสองจะได้

$$A^2 = \begin{bmatrix} 1 & 2 & 0 \\ 2 & 4 & 0 \\ 0 & 0 & 5 \end{bmatrix}$$

จากเมตริกซ์ A^2 พิจารณาสมาชิก $A^2(1,2)$ ซึ่งแทนการเชื่อมโยงระหว่างโนด 1 และโนด 2 จากเดิมในเมตริกซ์ A โนด 1 และโนด 2 ไม่ต่อกัน แต่ต่อถึงกันผ่านโนด 3 จากโนด 1 ไปหาโนด 2 ต้องใช้ 2 hop คือ จากโนด 1 ไปโนด 3 และจากโนด 3 ไปโนด 2 นั่นคือสามารถพิจารณาได้ว่าเมตริกซ์ A คือการต่อถึงกันโดยผ่านข่ายเชื่อมโยงเพียง 1 hop ส่วนเมตริกซ์ A^2 แทนการต่อถึงกันโดยผ่านข่ายเชื่อมโยง 2 hop เช่นเดียวกัน A^k คือเมตริกซ์ที่แทนการต่อถึงกันผ่านข่ายเชื่อมโยง k hop เมื่อหาผลรวมของ A และ A^2 จะได้

$$sum = A + A^2 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 2 \\ 1 & 2 & 0 \end{bmatrix} + \begin{bmatrix} 1 & 2 & 0 \\ 2 & 4 & 0 \\ 0 & 0 & 5 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 5 \end{bmatrix}$$

จากผลรวมของเมตริกซ์ sum พบว่าทุก ๆ สมาชิกไม่เป็น 0 แสดงว่ากราฟของโครงข่ายนี้ต่อถึงกัน (connected) ดังนั้นขั้นตอนการตรวจสอบการต่อถึงกันของกราฟคือการนำ adjacency matrix ที่แทนโครงข่ายนั้นมายกกำลังแล้วหาผลรวมกับผลรวมของเมตริกซ์ที่มีดีกรีเลขชี้กำลังต่ำกว่าจากนั้นตรวจดูว่าเมตริกซ์ผลรวมมีสมาชิกใดเป็นศูนย์หรือไม่ ถ้าไม่มีสมาชิกเป็นศูนย์ โนดทั้งหมดบนโครงข่ายก็ต่อถึงกัน ถ้ามีสมาชิกมีค่าเป็นศูนย์ก็ให้ยกกำลังเพิ่มขึ้นทีละหนึ่งและรวมเข้ากับผลรวมของเมตริกซ์ก่อนหน้า ถ้าพบว่าผลรวมเมตริกซ์ที่ค่ายกกำลังสูง ๆ ทำให้ผลรวมมีสมาชิกศูนย์ ก็สรุปว่ากราฟไม่ต่อกัน (disconnected) แสดงตามขั้นตอนได้ดังนี้

1. จาก adjacency matrix A ที่แทนโครงข่าย คำนวณหา

$$sum = A + A^2$$

2. ตรวจสอบว่ามีสมาชิกในเมตริกซ์ *sum* ที่ไม่ใช่สมาชิกในแนวทแยงมุมเป็นศูนย์หรือไม่

3. ถ้าไม่พบสมาชิกอื่นนอกเหนือสมาชิกในแนวทแยงมุมเป็นศูนย์ก็สรุปว่ากราฟต่อกัน แต่ถ้ามีบางสมาชิกเป็นศูนย์ก็แสดงว่ากราฟยังไม่ต่อกัน ให้เพิ่มเลขชี้กำลังเพิ่มขึ้นอีกหนึ่ง แล้วไปรวมกับเมตริกซ์ *sum* เดิม ทำซ้ำไปเรื่อย ๆ ถ้ายังพบสมาชิกที่ไม่ใช่แนวทแยงมุมเป็นศูนย์จะได้ผลรวม

$$sum = A + A^2 + A^3 + A^4 \ldots + A^k \qquad \text{(ข.1)}$$

4. ค่าของ k สูงสุดที่เป็นไปได้คือ n - 1 เมื่อ n คือจำนวนโนดทั้งหมดของโครงข่าย เพราะกรณีที่แย่ที่สุดที่กราฟจะมีโอกาสต่อกันได้คือเป็นกราฟที่ต่อถึงกันเป็นเส้นเพียงเส้นเดียว

ภาคผนวก ค Complexity of Algorithm

Worse Case Running Time [9]

โดยทั่วไปการวัดเวลาที่ใช้ในการคำนวณ (running time) ของอัลกอริทึมมักจะสนใจเวลาที่มากที่สุดที่ใช้ในการคำนวณ (Worse Case Running Time) ซึ่งเป็น upper bound ของเวลาที่ใช้ในการคำนวณอัลกอริทึมหนึ่ง ๆ และเป็นค่าที่ใช้ในการเปรียบเทียบอัลกอริทึมที่มีความเหมาะสมสำหรับการแก้ปัญหานั้น ๆ นอกจากนี้ยังใช้เป็นค่าที่วัด complexity ของอัลกอริทึมอีกด้วย

Big O Notation [9]

การหาเวลาที่มากที่สุดที่ใช้ในการคำนวณของอัลกอริทึมหนึ่ง ๆ มักจะวัดออกมาในรูปของค่า Big O ซึ่งมีนิยามดังต่อไปนี้

$O(g(n)) = \{ f(n) :$ มีค่าคงที่บวก c และ n_o ซึ่ง $0 \leq f(n) \leq c.g(n)$ สำหรับทุก ๆ $n \geq n_o \}$

กล่าวคือ เซตของเวลาที่ใช้คำนวณ f(n) ทั้งหลายที่ถ้ามีค่าคงที่ c และ n_0 สำหรับทุก ๆ n ที่มากกว่า n_0 และทำให้กราฟของเวลาในการคำนวณอัลกอริทึมอยู่ระหว่างเส้นกราฟ c.g(n) และ f(n) = 0 เวลาที่ใช้คำนวณอัลกอริทึมนี้มี upper bound เป็น g(n)

เราจะใช้ค่า Big O ในการให้ค่า asymtotic upper bound ของฟังก์ชัน f(n) ใด ๆ ตัวอย่างเช่น $f(n) = an^2 + bn + c$ เมื่อ a, b, c เป็นค่าคงที่ จะสามารถเขียนได้ว่า $an^2 + bn + c = O(n^2)$ ในกรณีที่มีอัลกอริทึมในการแก้ปัญหาเดียวกัน 2 ชุดคือ A1 และ A2 และมี worse case running time ของทั้งสองเป็น $O(n^4)$ และ $O(n^3)$ ตามลำดับ อัลกอริทึมที่น่าจะเลือกใช้ในการแก้ปัญหามากกว่าคืออัลกอริทึม A2 เพราะเมื่อขนาดของปัญหามีขนาดใหญ่มากขึ้น (n มากขึ้น) อัลกอริทึม A2 มีแนวโน้มที่จะใช้เวลาในการคำนวณน้อยกว่าอัลกอริทึม A1

ตัวอย่างการคำนวณหาเวลาในการคำนวณของอัลกอริทึม

จะใช้ตัวอย่างของการทำ Insertion sort ทำการจัดเรียงข้อมูลจากค่าน้อยไปหามากของค่าในเวกเตอร์ A ซึ่งมีจำนวนทั้งหมด n ค่า

อัลกอริทึม	เวลาคำนวณต่อรอบ	จำนวนรอบการคำนวณ	worse case running time
INSERTION-SORT(A)			
for j ← 2 to length(A)	c_1	n	O(n)
do key ← A[j]	c_2	n - 1	O(n)
% Insert A[j] into the sorted sequence			
% A[1 .. j -1]			
k ← j -1	c_4	n - 1	O(n)
while k > 0 and A[k] > key	c_5	$\sum_{j=2}^{n} t_j$	$O(n^2)$
do A[k+1] ← A[k]	c_6	$\sum_{j=2}^{n} (t_j - 1)$	$O(n^2)$
k ← k - 1	c_7	$\sum_{j=2}^{n} (t_j - 1)$	$O(n^2)$
A[k + 1] ← key	c_8	n - 1	O(n)

เวลาที่ใช้คำนวณ T(n) ของอัลกอริทึม INSERTION-SORT คือผลรวมของเวลาที่ใช้ไปในแต่ละคำสั่ง คำสั่งที่ใช้การคำนวณไป c_i ขั้นตอนและประมวลผลไป n ครั้งจะใช้เวลาคำนวณของคำสั่งนั้นเท่ากับ $c_i n$ ดังนั้น

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\sum_{j=2}^{n} t_j + c_6\sum_{j=2}^{n}(t_j - 1) + c_7\sum_{j=2}^{n}(t_j - 1) + c_8(n-1)$$

ในกรณีที่ชุดตัวเลขทั้งหมดเรียงลำดับกันอยู่แล้ว (best case) $t_j = 1$

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1)$$

$$= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8)$$

ในกรณีที่ตัวเลขเรียงสลับลำดับจากมากไปน้อย (worst case) $t_j = j$

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n-1)}{2} - 1\right) + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1)$$

$$= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)n - (c_2 + c_4 + c_5 + c_8)$$

$$= O(n^2)$$

เราสามารถพิจารณาจากผลรวมของ upper bound ของเวลาที่ใช้ในการทำงานแต่ละคำสั่งได้เป็น

$$T(n) = O(n) + O(n) + O(n) + O(n^2) + O(n^2) + O(n^2) + O(n) = O(n^2)$$

จะพบว่า loop ของชุดคำสั่งมีผลอย่างมากต่อการทำงานของโปรแกรมและการคำนวณค่า worst-case running time ค่า running time ที่มากที่สุด (complexity) มีค่าเท่ากับจำนวนการวนรอบของ loop ที่ซ้อนกันมากที่สุดในโปรแกรม

การหาเวลาที่ใช้ในการคำนวณของอัลกอริทึม Jia, อัลกอริทึมที่เสนอ p1, p2, p1 partial และ p2 partial

อัลกอริทึมของ Jia ประกอบด้วยฟังก์ชันย่อยได้แก่ ฟังก์ชัน **Remove** ทำหน้าที่ลบ VP switches ออกจากโครงข่าย ATM ให้ผลลัพธ์เป็นโครงข่ายเสมือน (virtual network), ฟังก์ชัน **Induce** ทำหน้าที่ยุบโครงข่ายเสมือนให้เหลือเฉพาะโครงข่ายที่มีสวิตช์ที่เป็นโนดต้นทางและโนดปลายทาง และฟังก์ชัน **Kruskal** ทำหน้าที่คำนวณหา minimum spanning tree ตามวิธีที่เสนอโดย Kruskal [9] โดยอัลกอริทึมของ Jia แสดงได้ดังนี้

```
function  R = Jia( G(V, E), s, D )
1       Remove VP switches from ATM network G( V, E) , the result is G'
2       Induce  G' with {s} U D,  I(G')
3       Kruskal  find minimum spanning tree of I(G') by Kruskal's algorithms,  R
4       return R
end
```

โดยที่ฟังก์ชัน **Remove**, **Induce** และ **Kruskal** มีรายละเอียดดังต่อไปนี้

```
function  G' = Remove( G(V, E))
1       d ← G
2        n ← length(G)
3       %  find shortest path between VC switches
4       for r ← 1  to  n
5          if  r ≠ VC switches
6           for p ← 1 to n
7               for  q ← 1 to n
8                  if d(p, q) > d(p, r) + d(r, q)
9                     d(p, q) ← d(p, r) + d(r, q)
```

```
10                  end
11                end
12              end
13            end
14          end
15          %delete the links  VP switches - VC switches  and VP switches - VP switches
16          for u ←1 to  n
17             if u ∈ VP switches
18                for v ← 1 to n
19                  d(u,v) ← 0
20                end
21             end
22          end
23          return G'
end
```

พิจารณาฟังก์ชัน **Remove** ในบรรทัดที่ 4 ถึงบรรทัดที่ 14 มีค่า complexity เท่ากับ $O(n^3)$ ส่วนบรรทัดที่ 16 ถึง 23 มี complexity เท่ากับ $O(n^2)$ ดังนั้น complexity ทั้งหมดของฟังก์ชัน **Remove** จึงเท่ากับ $O(n^3) + O(n^2) = O(n^3)$

```
function  I(G') = induce(G'(V', E'), s, D)
1           d ← G'
2           n ← length(G')
3          for r ← 1 to n
4             for p ← 1 to n
5                for q ← 1 to n
```

```
6                   if d(p, q) > d(p, r) + d(r, q)
7                     d(p, q) ← d(p, r) + d(r, q)
8                   end
9                 end
10              end
11            end
12          for (p, q) ∈ E'
13              if I or q ∉ {s} U D
14                  d(p, q) ← 0
15              end
16          end
17          I(G') ← d
18          return I(G')
end
```

พิจารณาฟังก์ชัน **Induce** ในบรรทัดที่ 2 ถึง 10 มีค่า complexity เท่ากับ $O(n^3)$ ส่วนบรรทัดที่ 11 ถึง 15 มีค่า complexity เท่ากับ $O(n^2)$ ดังนั้น complexity ของฟังก์ชัน **Induce** เท่ากับ $O(n^3) + O(n^2) = O(n^3)$

```
function  R = Kruskal(I(G'(V', E')))
1       R ← ∅
2       for each node v in the set of nodes in I(G'(V', E'))
3           do make a set  {v}
4       end
5       sort links in I(G') by ascending weight w
6                   for (u, v) ∈ E'
```

```
7          do if u and v belong to different sets
8          R ← R U {(u, v)}
9          Union(u, v)
10      end
11      return R
end
```

พิจารณาฟังก์ชัน **Kruskal** บรรทัดที่ 2 ถึงบรรทัดที่ 4 มีค่า complexity เท่ากับ $O(n)$ ส่วนในบรรทัดที่ 5 การเรียงลำดับ links ตามขนาดของ weight จากน้อยไปมากมีค่า complexity เท่ากับ $O(m\log m)$ ตามเอกสารอ้างอิง [9] เมื่อ n คือจำนวนโนดและ m คือจำนวนข่ายเชื่อมโยง (links) ในโครงข่าย และบรรทัดที่ 6 ถึง 10 มีค่า complexity เท่ากับ $O(m\log m)$ ตามเอกสาร [9] ดังนั้น complexity ของฟังก์ชัน Kruskal เท่ากับ $O(n)+O(m\log m)+O(m\log m)=O(m\log m)$

พิจารณาอัลกอริทึมที่เสนอโดย Jia [1] ในบรรทัดที่ 1 ฟังก์ชัน **Remove** มี complexity เท่ากับ $O(n^3)$ บรรทัดที่ 2 ฟังก์ชัน **Induce** มี complexity เท่ากับ $O(n^3)$ และฟังก์ชัน **Kruskal** ในบรรทัดที่ 3 มี complexity เท่ากับ $O(m\log m)$ ดังนั้นอัลกอริทึม **Jia** มี complexity เท่ากับ

$$O(n^3)+O(n^3)+O(m\log m)=O(n^3+n^3+m\log m)=O(n^3+m\log m)$$

```
function  R = p1( G(V, E), s, D)
1       Remove VP switches from ATM network G( V, E) , the result is G'
2       T = {s}
3       while num(D) <> 0
4             if v not used
5                Dijkstra Find the closet path from any  destinations  v to node on tree
6                        by using Dijkstra's algorithm
7                if v is nearest to u of T
```

```
8                   T ← T U {v}
9                   Recover path from u to v
10                end
11              end
12              num(D) ← num(D) -1
13        end
14        return R
end

function [P,pred] = Dijkstra(G(V, E), root)
1     n ← length(G)
2     dist ← G
3     for k ← 1  to n
4          label(k) ← dist(root, k)
5          pred(k) ← root
6         scanned(k) ← false
7     end
8     scanned(root) ← true
9     numscanned ← 1
10    while numscanned < n
11        % Find next node to scan
12        bestL ← inf
13        for k ← 1 to n
14             if scanned(k) = false  and  label(k) < bestL
15                bestL ← label(k)
16                lscan ← k
```

```
17            end
18          end
19          % Scan node Iscan
20          for  k ← 1 to n
21              if  label(k) > label(Iscan) + dist(Iscan, k)
22                  label(k) ← label(Iscan)+ dist(Iscan, k)
23                  pred(k) ← Iscan
24              end
25          end
26              scanned(Iscan) ← true
27              numscanned ← numscanned + 1
28          end
29          P ← label
30          return P
31          return pred
end
```

พิจารณาอัลกอริทึม **Dijkstra** ตั้งแต่บรรทัดที่ 3 ถึงบรรทัดที่ 7 มีค่า complexity เท่ากับ $O(n)$ และจากบรรทัดที่ 10 ถึงบรรทัดที่ 28 มี complexity เท่ากับ $O(n^2)$ ดังนั้นค่า complexity ของอัลกอริทึม **Dijkstra** มีค่าเท่ากับ $O(n)+O(n^2)=O(n^2)$

พิจารณาอัลกอริทึม **p1** ฟังก์ชัน **Dijkstra** ในบรรทัดที่ 5 ให้ค่า complexity เท่ากับ $O(n^2)$ และฟังก์ชัน **Recover** ให้ค่า complexity เท่ากับ $O(n)$ ตามเอกสารอ้างอิง [11] แต่ทั้งสองฟังก์ชันอยู่ในลูป while ซึ่งในกรณีที่แย่ที่สุดจะทำการวนรอบจำนวน d รอบ เมื่อให้ d แทนจำนวนโนดที่เป็นสมาชิกกลุ่มมัลติคาสต์ ดังนั้น worse-case running timeหรือ complexity ของโปรแกรมตั้งแต่บรรทัดที่ 3 ถึงบรรทัดที่ 13 จึงเท่ากับ $O(d(O(n^2)+O(n)))=O(d(O(n^2)))=O(O(dn^2))=O(dn^2)$ หรือเมื่อ d มีจำนวนมากขึ้นอาจพิจารณาให้ complexity เท่ากับ $O(n^3)$ ได้ และ complexity ของ

ฟังก์ชัน Remove ในบรรทัดที่ 1 เท่ากับ $O(n^3)$ ดังนั้น complexity ของอัลกอริทึม **p1** เท่ากับ $O(n^3) + O(dn^2) = O(n^3)$

```
function  R = p2( G(V, E), s, D)
1       Remove VP switches from ATM network G( V, E) , the result is G'
2       T = {s}
3       while num(D) <> 0
4             if v not used
5               Dijkstra Find the closet path from any  destinations  v to node on tree
6                       by using Dijkstra's algorithm
7               if v is nearest to u of T and  u  ∉ V - D
8                 T ⟵ T U {v}
9                 Recover path from u to v
10              end
11            end
12            num(D) ⟵ num(D) -1
13      end
14      return R
end
```

ฟังก์ชัน **p2** แตกต่างจากฟังก์ชัน **p1** ในบรรทัดที่ 7 มีการพิจารณาโนดบน tree เฉพาะที่มีคุณสมบัติ u ∉ V - D อย่างไรก็ตาม complexity ของคำสั่งตั้งแต่บรรทัดที่ 3 ถึงบรรทัดที่ 13 ยังคงเป็น $O(dn^2)$ เช่นเดียวกับอัลกอริทึม **p1** ดังนั้น complexity ของอัลกอริทึม **p2** จึงเท่ากับ $O(n^3) + O(dn^2) = O(n^3)$

```
function  R = p1_partial( G(V, E), s, r_k)
1       Remove VP switches from ATM network G( V, E) , the result is G'
2       T = {s}
3       k ← node that send request r_k
4               Dijkstra Find the closet path from any  destinations  k to node on tree
5                       by using Dijkstra's algorithm
6               if k is nearest to u of T
7                 T ← T U {k}
8                 Recover path from u to k
9               end
10            end
11      end
12      return R
end
```

ฟังก์ชัน **p1_partial** ประกอบด้วย complexity $O(n^2)$ ในบรรทัดที่ 4 และ $O(n)$ ในบรรทัดที่ 8 ดังนั้น complexity ของอัลกอริทึม **p1_partial** จึงเท่ากับ $O(n^3) + O(n^2) + O(n) = O(n^3)$

```
function  R = p2_partial( G(V, E), s, r_k)
1       Remove VP switches from ATM network G( V, E) , the result is G'
2       T = {s}
3       k ← node that send request r_k
4               Dijkstra Find the closet path from any  destinations  k to node on tree
5                       by using Dijkstra's algorithm
6               if k is nearest to u of T  and u  ∉ V - D
7                 T ← T U {k}
8                 Recover path from u to k
9               end
10            end
11      end
12      return R
end
```

ฟังก์ชัน p2_partial แตกต่างจาก p1_partial ตรงที่การตรวจเงื่อนไขในบรรทัดที่ 6 ดังนั้น complexity ของ p2_partial จึงเท่ากับ p1_partial คือ $O(n^3)$

ประวัติผู้เขียน

นายสุชัย โรจนวิไลกุล เกิดวันที่ 1 มกราคม พ.ศ. 2518 ที่กรุงเทพมหานคร สำเร็จการศึกษาปริญญาตรีวิศวกรรมศาสตร์บัณฑิต ภาควิชาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2539 และเข้าศึกษาต่อในหลักสูตรวิศวกรรมศาสตร์มหาบัณฑิตที่จุฬาลงกรณ์มหาวิทยาลัยเมื่อ พ.ศ. 2540