การจัดเรียงเครือข่ายเพื่อเพิ่มประสิทธิภาพให้กับจีเนติกอัลกอริทึมในการหาเส้นทางแบบหลายปลายทาง

GENETIC ALGORITHM WITH NETWORK REORDERATION FOR MULTICAST NETWORK

ศิรเศรษฐ์ ศิรอดุลพัฒน์

SERASATE SERAJATUPAT

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาเทคโนโลยีสารสนเทศ
บัณฑิตวิทยาลัย
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ. 2545

ISBN 974-648-899-6

การจัดเรียงเครือข่ายเพื่อเพิ่มประสิทธิภาพให้กับจีเนติกอัลกอริทึมในการหาเส้นทางแบบหลายปลายทาง

GENETIC ALGORITHM WITH NETWORK REORDERATION FOR MULTICAST NETWORK

ศิรเศรษฐ์ ศิรจตุพัฒน์
SERASATE SERAJATUPAT

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาเทคโนโลยีสารสนเทศ
บัณฑิตวิทยาลัย
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ. 2545

ISBN 974-648-899-6

GENETIC ALGORITHM WITH NETWORK REORDERATION FOR MULTICAST NETWORK

SERASATE SERAJATUPAT

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENT FOR THE DEGREE OF
MASTER OF SCIENCE IN INFORMATION TECHNOLOGY
SCHOOL OF GRADUATE STUDIES
KING MONGKUT'S INSTITUE OF TECHNOLOGY LADKRABANG
2002

ISBN 974-648-899-6

หัวข้อวิทยานิพนธ์	การจัดเรียงเครือข่ายเพื่อเพิ่มประสิทธิภาพให้กับจีเนติกอัลกอริ-ทึมในการหาเส้นทางแบบหลายปลายทาง
ชื่อนักศึกษา	นายศิรเศรษฐ์ ศิรจตุพัฒน์
รหัสประจำตัว	42067023
ปริญญา	วิทยาศาสตร์มหาบัณฑิต
สาขาวิชา	เทคโนโลยีสารสนเทศ
พ.ศ.	2545
อาจารย์ผู้ควบคุมวิทยานิพนธ์	ผศ.ดร. โชติพัชร์ ภรณวลัย

บทคัดย่อ

วิทยานิพนธ์นี้นำเสนอวิธีการใหม่เพื่อเพิ่มประสิทธิภาพให้กับจีเนติกอัลกอริทึมในการหาเส้นทางแบบหลายปลายทาง แต่เดิม, จีเนติกอัลกอริทึมที่ใช้แก้ปัญหานี้ไม่ได้มีการจัดเรียงโครโมโซมให้มีประสิทธิภาพอย่างมีหลักการ และส่งผลให้กระบวนการของจีเนติกอัลกอริทึมใช้เวลามากกว่าที่ควรเป็น การจัดเรียงเครือข่ายขึ้นมาใหม่ให้เหมาะสมกับรูปแบบของโครโมโซมก่อนที่จะส่งให้จีเนติกอัลกอริทึมประมวณผลเพื่อหาคำตอบสามารถช่วยแก้ปัญหาดังกล่าวได้ ซึ่งส่งผลให้กระบวนการของจีเนติกอัลกอริทึมใช้เวลาในการหาผลลัพธ์น้อยลง (สามารถพบคำตอบได้โดยใช้จำนวนรุ่นของการเกิดของโครโมโซมน้อยลง และเวลาที่ใช้ในการประมวณผลน้อยลงด้วย) เนื่องจากการจัดเรียงเครือข่ายขึ้นมาใหม่นั้น ส่งผลให้โครโมโซมของจีเนติกอัลกอริทึมมีประสิทธิภาพมากขึ้น

Thesis Title	Genetic Algorithm with Network Reorderation for Multicast Network
Student	Mr. Serasate Serajatupat
Student ID.	42067023
Degree	Master of Science
Programme	Information Technology
Year	2002
Thesis Advisor	Dr. Chotipat Pornavalai

ABSTRACT

This thesis presents a new scheme for improve performance of multicast routing with genetic algorithm to find the multicast tree. In traditional genetic algorithm, which is used to solve this problem does not consider the order of chromosome. So it causes genetic algorithm spend more time than it should be. By the method of reorderation of network node before sending to genetic algorithm can help the genetic algorithm to find the solution with less generation and computation time.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงได้อย่างดี ด้วยคำแนะนำและคำปรึกษาเกี่ยวกับระบบเครือข่ายคอมพิวเตอร์แบบต่างๆ ระบบการทำงานของจีเนติกอัลกอริทึม รวมทั้งได้ทดสอบและตรวจวัดผลจาก ผศ.ดร. โชติพัชร์ ภรณวลัย ซึ่งเป็นอาจารย์ผู้ควบคุมวิทยานิพนธ์ ผู้วิจัยรู้สึกซาบซึ้งในความอนุเคราะห์จากท่านและขอกราบขอบพระคุณเป็นอย่างสูง

ขอขอบคุณ เพื่อน ๆ นักศึกษาทุกท่านที่ให้ความช่วยเหลือให้คำแนะนำต่าง ๆ และยังให้กำลังใจต่อผู้วิจัยอย่างใกล้ชิดตลอดมา

สุดท้ายขอขอบคุณบัณฑิตวิทยาลัย ที่ได้ให้ทุนสนับสนุนงานวิจัยเพื่อการทำวิทยานิพนธ์ครั้งนี้ให้สำเร็จได้ด้วยดี

คุณค่าและประโยชน์อันพึงมีของวิทยานิพนธ์ฉบับนี้ ผู้วิจัยขอมอบแด่ผู้มีพระคุณทุกท่าน

ศิรเศรษฐ์ ศิรจตุพัฒน์

สารบัญ

สารบัญ(ต่อ)

หน้า

สารบัญตาราง

สารบัญรูป

สารบัญรูป(ต่อ)

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

เนื่องจากระบบเครือข่ายคอมพิวเตอร์ในปัจจุบัน มีการถ่ายโอนข้อมูลเป็นจำนวนมากอยู่ตลอดเวลาโดยเฉพาะระบบเครือข่ายที่รองรับการทำงานแบบหลายสื่อ (Multimedia communication) ที่ต้องการรับ และ ส่งข้อมูลให้ทันภายในเวลาที่กำหนด เช่น การประชุมผ่านระบบเครือข่าย (Video Conference) การฟังรายการวิทยุผ่านระบบอินเตอร์เน็ต หรือการเรียนการสอนทางไกลผ่านระบบอินเตอร์เน็ต (Teleteaching) ซึ่งในลักษณะดังกล่าวจะเป็นการส่งข้อมูลแบบหลายปลายทาง (Multicast) เช่น ผู้ที่ฟังวิทยุจากสถานีเดียวกันในระบบอินเตอร์เน็ตก็จะได้รับข้อมูลชุดเดียวกัน ซึ่งการหาเส้นทางเพื่อกระจายข้อมูลในลักษณะนี้ จะต้องใช้เวลามากในการหาคำตอบ เพื่อที่จะได้คำตอบที่ดีที่สุด หรือ ใกล้เคียงกับคำตอบที่ดีที่สุด เพราะปัญหาในลักษณะดังกล่าวเป็นปัญหาแบบ NP-Complete

1.2 วัตถุประสงค์ของงานวิจัย

การรับส่งข้อมูลแบบกระจายข้อมูลเดียวกันจากผู้ส่งไปยังผู้รับหลาย ๆ รายนั้น เรียกว่า มัลติแคสติง (Multicasting) โดยที่เส้นทางในการกระจายข้อมูลจะเรียกว่า มัลติแคสตทรี (Multicast tree) ซึ่งที่ผ่านมานั้น ได้มีผู้ทำการวิจัยและค้นคว้าเกี่ยวกับเรื่องนี้มามากพอสมควร และวิธีการที่นักวิจัยเหล่านั้นได้นำเสนอก็จะมีแนวคิดหลายวิธี เช่น Neural Network, CRA Heuristic[2], CKMB Heuristic[3] และ จีเนติกอัลกอริทึม (Genetic Algorithm)[4][5][6] เป็นต้น หากวิธีการต่าง ๆ เหล่านี้สามารถนำมาเพิ่มประสิทธิภาพให้สูงขึ้นได้ ไม่ว่าจะโดยวิธีการปรับปรุงอัลกอริทึมให้ดีขึ้น ลดข้อด้อยบางอย่างภายในอัลกอริทึมลง หรือ เพิ่มแนวคิดใหม่เข้าไปเพื่อเพิ่มประสิทธิภาพก็ตาม ก็จะทำให้การใช้งานระบบเครือข่ายที่รองรับการรับส่งข้อมูลแบบ multicast เกิดประสิทธิภาพสูงสุด

1.2 แนวคิดที่ใช้ในงานวิจัย

จีเนติก อัลกอริทึม เป็นทฤษฎีที่จำลองกระบวนการวิวัฒนาการทางธรรมชาติโดยอาศัยพื้นฐานความคิดทางพันธุกรรมในการถ่ายทอดลักษณะต่างๆ ไปยังลูกหลาน ซึ่งสามารถนำมาประยุกต์เพื่อใช้หาคำตอบที่ใกล้เคียงหรือดีที่สุดของปัญหาได้ จีเนติก อัลกอริทึมเป็นวิธีการหาคำตอบโดยพิจารณา และดำเนินการจากกลุ่มคำตอบของปัญหาที่ถูกสร้างขึ้น โดยการเข้ารหัส (Coding) แปลงปัญหาค่าตัวแปร หรือพารามิเตอร์ต่างๆ ให้อยู่ในรูปโครงสร้างของโครโมโซมตามที่

กำหนดเพื่อคัดเลือกโครโมโซมคำตอบที่เหมาะสม สำหรับสร้างวิวัฒนาการของคำตอบให้ดีขึ้นตามกระบวนการทางพันธุศาสตร์ โดยแลกเปลี่ยนค่าพารามิเตอร์ต่างๆระหว่างโครโมโซมที่ถูกคัดเลือกเพื่อสร้างโครโมโซมใหม่ที่อาจจะนำไปสู่คำตอบของปัญหาที่ดีขึ้นเรื่อยๆ ในทำนองเดียวกันการใช้จีเนติกอัลกอริทึม เพื่อใช้แก้ปัญหาการหาเส้นทางแบบหลายปลายทางก็สามารถทำได้เช่นกัน โดยการสร้างกลุ่มของโหนดภายในเครือข่ายให้อยู่ในรูปโครงสร้างโครโมโซมของเครือข่ายที่เหมาะสมเพื่อคัดเลือกโดยวิเคราะห์ค่าความเหมาะสมจากการตรวจสอบตามเงื่อนไขที่กำหนดไว้ และปรับปรุงรูปแบบของเส้นทางให้ดีขึ้นโดยการแลกเปลี่ยนหรือสลับค่าพารามิเตอร์ต่างๆของโครโมโซมเครือข่ายที่ถูกคัดเลือก ซึ่งจะทำให้เกิดการวิวัฒนาการของคำตอบ เพื่อสร้างเส้นทางที่ต้องการ

1.4 ขอบเขตของการวิจัย และวิธีที่ใช้ในการวิจัย

วิทยานิพนธ์ฉบับนี้มีวัตถุประสงค์ที่จะศึกษาวิจัย เพื่อเสนอแนวทางในการเพิ่มประสิทธิภาพในการหาเส้นทางแบบหลายปลายทาง โดยมุ่งเน้นที่การนำเอา จีเนติกอัลกอริทึมมาใช้ให้เกิดประโยชน์สูงสุด โดยจำลองการแก้ปัญหาต่าง ๆ บนเครื่อง Personal Computer ที่ใช้ Pentium IV ความเร็ว 1600 MHz เป็นหน่วยประมวนผลกลาง และใช้ Borland C++ Builder 5.0 เป็น tool ในการสร้างโปรแกรมจำลอง รายละเอียดของวิทยานิพนธ์ฉบับนี้จึงแยกออกเป็นส่วน ๆ คือ จะกล่าวถึงความรู้พื้นฐานของจีเนติก อัลกอริทึม ลำดับขั้นตอนในกระบวนการต่าง ๆ ตัวอย่างการใช้งานจีเนติก อัลกอริทึม ทฤษฎีอื่น ๆ ที่เกี่ยวข้องในวิทยานิพนธ์ฉบับนี้ และแนวคิดที่จะใช้เพิ่มประสิทธิภาพให้กับจีเนติก อัลกอริทึม รวมถึงการสรุปผล

บทที่ 2
ความรู้พื้นฐานเกี่ยวกับจีเนติก อัลกอลิทึม

ในปัจจุบันนี้ปัญหาที่ต้องการคำตอบที่ดีที่สุด (Optimal Solution) ทางวิทยาศาสตร์, วิศวกรรมศาสตร์ คอมพิวเตอร์ หรือในการทำงานต่างๆ ที่เกิดขึ้นมากมายนั้น สามารถหาคำตอบได้หลายๆ วิธี ซึ่งแตกต่างกันไปตามชนิดของปัญหา ความคิด เทคนิค วิธีการวิเคราะห์ปัญหานั้นๆ และความแพร่หลายในการพัฒนาศักยภาพของคอมพิวเตอร์ให้รู้จักเรียนรู้เพื่อช่วยหาคำตอบหรือช่วยตัดสินคำตอบในขั้นต้นมีมากขึ้น โดยปัจจุบันนี้นักวิทยาศาสตร์ได้เริ่มนำความรู้เกี่ยวกับทฤษฎีหรือกฎเกณฑ์ทางธรรมชาติมาช่วยในการศึกษาวิจัย เช่น นิวรอลเน็ตเวิร์ค (Neural Network) ฟัซซี่ลอจิก (Fuzzy Logic) เป็นต้น จีเนติกอัลกอริทึมเป็นอีกวิธีการหนึ่งที่จำลองรูปแบบวิธีการทางชีววิทยา ในการให้กำเนิดประชากรรุ่นใหม่หรือขยายเผ่าพันธุ์ในรุ่นลูก รุ่นหลานต่อไป ซึ่งอาศัยพื้นฐานความคิดของวิวัฒนาการทางธรรมชาติถ่ายทอดลักษณะต่างๆ ทางพันธุกรรม โดยปฏิบัติตามกระบวนการทางพันธุศาสตร์ เพื่อจะใช้ในการหาคำตอบที่ดีที่สุดหรือใกล้เคียงคำตอบที่ดีที่สุดของปัญหาโดยคอมพิวเตอร์

2.1 พันธุศาสตร์ทางชีววิทยา

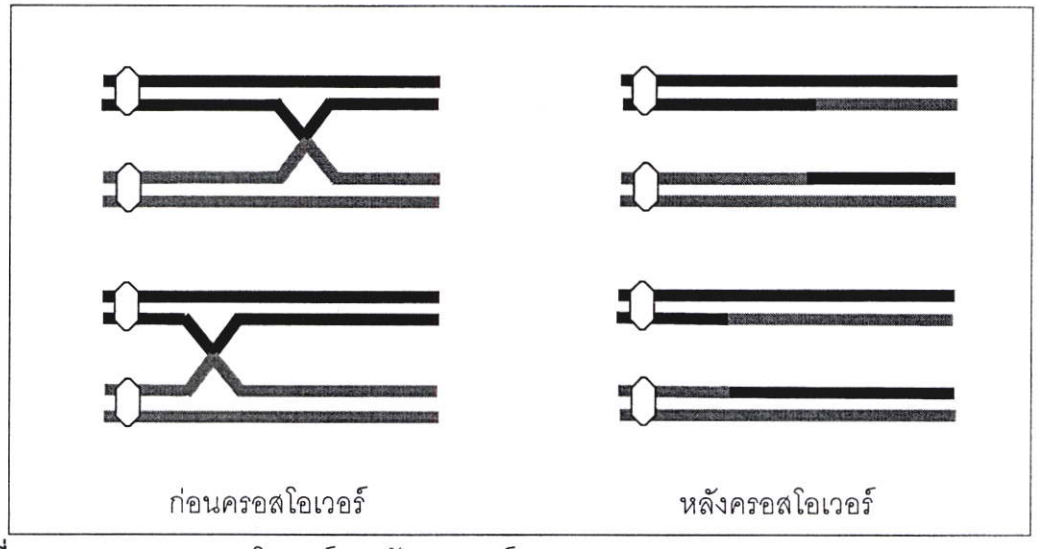
เมนเดล (Mendel) บิดาแห่งวิชาพันธุศาสตร์ ค้นพบว่า ยีนส์ (Genes) หน่วยเก็บลักษณะทางกรรมพันธุ์เป็นตัวกำหนดลักษณะภายนอก ซึ่งยีนส์หลายๆ ยีนส์จะเรียงตัวกันอยู่บนเส้นโครโมโซม (Chromosome) อีกทีหนึ่งในเซลของสิ่งมีชีวิตโดยจะอยู่กันเป็นคู่ๆ แต่จะแตกต่างกันที่ค่าลักษณะต่างๆ ในแต่ละยีนส์เรียกว่าแอลลี (Allele) ซึ่งแบบต่างๆ ของยีนส์ที่มีแอลลีต่างกันในแต่ละตำแหน่งยีนส์เดียวกันเรียกว่า ยีโนไทป์ (Genotype) สำหรับลักษณะภายนอกที่ปรากฏออกมาให้เห็น เรียกว่า ฟีโนไทป์ (Phenotype) เช่น ในคนจะมีโครโมโซม 23 คู่ 46 โครโมโซม ซึ่งแต่ละโครโมโซมจะประกอบด้วยยีนส์ต่างๆกันราว 1250 ยีนส์

การถ่ายทอดลักษณะทางพันธุกรรมเป็นการถ่ายทอดลักษณะต่างๆ ของสิ่งมีชีวิตที่เกิดขึ้นเมื่อมีการแบ่งตัวของเซลสิ่งมีชีวิต ซึ่งมี 2 แบบ คือ

1. การเพิ่มจำนวนเซล เป็นการแบ่งตัวแบบไมโตสิส (Mitosis) โดยโครโมโซมแต่ละตัวจะขยายตัวเพิ่มจำนวนตัวเองขึ้นเป็นสอง และเยื่อหุ้มนิวเคลียส (Nucleous) จะสลายลงเพื่อดึงแยกโครโมโซมที่เพิ่มจำนวนขึ้นออกจากโครโมโซมเดิมเป็นสองด้าน แล้วเยื่อหุ้มนิวเคลียสจะถูกสร้างขึ้นใหม่กลายเป็นเซลใหม่ 2 เซล ที่มีโครโมโซมเหมือนเดิม
2. การแบ่งตัวของเซลสืบพันธุ์ เป็นการแบ่งตัวแบบไมโอสิส (Meiosis) โดยโครโมโซมจากเซลพ่อ 1 โครโมโซมและโครโมโซมจากเซลแม่ 1 โครโมโซม จะเริ่มจับคู่กันที่โครโมโซม

ชนิดเดียวกัน ในขณะเดียวกันโครโมโซมแต่ละตัวทั้งที่มาจากพ่อและมาจากแม่ ต่างก็จำลองแบบของตนเพิ่มขึ้นอีกแต่ละโครโมโซม ทำให้ได้จำนวนโครโมโซมทั้งหมดเพิ่มขึ้นเป็นสองเท่า และดำเนินการทางพันธุกรรมจนถึงระยะแบ่งตัว โครโมโซมพ่อและแม่พร้อมกับแบบจำลองที่ได้จะแยกคู่ไปรวมกันเป็น 2 นิวเคลียส กลายเป็นเซลใหม่ 2 เซล ซึ่งจะแบ่งตัวต่อทันที โดยแต่ละโครโมโซมพ่อและแม่จะแยกตัวออกจากแบบจำลองมารวมกันเป็นเซลใหม่ 4 เซล

วิธีการทางพันธุศาสตร์ในระหว่างที่เกิดการแบ่งตัวของไมโอสิสนั้น โครโมโซมจะมีโอกาสแลกเปลี่ยนส่วนบางส่วนซึ่งกันและกัน อันเป็นสาเหตุที่ทำให้เกิดปรากฏการณ์ที่เรียกว่า ครอสโอเวอร์ (Crossover) ของลักษณะต่างๆขึ้น ซึ่งการครอสโอเวอร์นั้นเกิดขึ้นขณะที่มีการจำลองแบบเพิ่มขึ้น และเกิดขึ้นระหว่างโครโมโซมพ่อกับโครโมโซมแม่ ไม่ใช่เกิดขึ้นระหว่างโครโมโซมพ่อกับแบบจำลอง ซึ่งการครอสโอเวอร์จะทำให้เกิดการแลกเปลี่ยนลักษณะของยีนส์ต่างๆ ของคู่โครโมโซมพ่อกับแม่ โดยเนื่องจากยีนส์แต่ละยีนส์เรียงตัวกันบนเส้นโครโมโซมนั้นไม่ได้อยู่กันอย่างหนาแน่น แต่มีระยะห่างกันอย่างไม่สม่ำเสมอ เพราะคุณสมบัติของยีนส์นั้นเป็นโมเลกุลของสารโปรตีนประกอบตัวกันทางเคมี ช่องว่างระหว่างยีนส์นี่เองจะเป็นตำแหน่งที่แตกออกมาได้เวลาจะครอสโอเวอร์ และแลกเปลี่ยนยีนส์ของโครโมโซมโดยส่วนที่อยู่หลังรอยแตก โดยจะถูกย้ายไปอยู่อีกโครโมโซมหนึ่งทั้งหมด นอกจากนี้ยังจะสามารถแตกอีกกี่แห่งก็ได้ ซึ่งผลนั้นขึ้นอยู่กับความสามารถที่จะเชื่อมกันมากน้อยเพียงไรของช่องว่างระหว่างยีนส์ดังรูป 2.1



รูปที่ 2.1 แสดงการครอสโอเวอร์ทางพันธุศาสตร์

ประโยชน์ที่เกิดจากครอสโอเวอร์ที่เราจะเห็นได้คือทำให้มีโอกาสที่จะได้ลักษณะต่างๆกันมาอยู่รวมกันได้หลายแบบมากขึ้น ทำให้สิ่งมีชีวิตรุ่นลูกที่เกิดขึ้นมามีความหลากหลายมากขึ้น และอาจทำให้มีโอกาสเกิดสิ่งมีชีวิตที่มีลักษณะต่างๆ ที่ดีพอเหมาะรวมอยู่ด้วยกันได้อย่างพอดีเหมาะสมกับสิ่งแวดล้อม ถ้าการเกิดเซลใหม่โดยถ่ายทอดโครโมโซมไม่มีการครอสโอเวอร์แล้วโครโมโซมใดเคยมียีนส์ลักษณะใดก็จะมีลักษณะนั้นอยู่เรื่อยๆ รุ่นลูกก็จะมีลักษณะเช่นเดียวกัน โอกาสที่สิ่งมีชีวิตนั้นจะเจริญ หรือปรับตัวให้ดีขึ้นย่อมมีได้ยากกว่าการเปลี่ยนลักษณะยีนส์ใหม่หลายๆแบบมากขึ้น

ลักษณะต่างๆ ของสิ่งมีชีวิตจะสามารถอยู่รอดได้โดยการคัดเลือกทางธรรมชาติคือ คัดเลือกโครโมโซมที่มีลักษณะที่ทำให้สิ่งมีชีวิตแข็งแรงเพียงพอ หรือเหมาะสมต่อสภาพแวดล้อม ซึ่งจะสามารถอยู่รอดและถ่ายทอดไปยังลูกหลานได้ ดังนั้นการคัดเลือกของธรรมชาติเพื่อถ่ายทอดลักษณะทางพันธุกรรม เป็นเพียงส่วนประกอบของการเปลี่ยนแปลงของสิ่งมีชีวิตเท่านั้น มิวเตชัน (Mutation) หรือการผ่าเหล่า คือการเปลี่ยนแปลงลักษณะของยีนส์ไปจากเดิมที่ควรเป็นไปตามการถ่ายทอด ซึ่งเป็นต้นเหตุของการเกิดลักษณะที่แปลกๆ ออกไปมากมายสำหรับสิ่งมีชีวิตหนึ่งๆ ซึ่งเท่ากับเป็นการให้โอกาสแก่ธรรมชาติในการที่จะเลือกลักษณะแปลกๆมากขึ้น เนื่องจากขบวนการวิวัฒนาการโดยธรรมชาติเองนั้นช้ามาก เพราะกว่าที่ธรรมชาติจะปรับสภาพแวดล้อมให้สิ่งมีชีวิตค่อยๆปรับตัวเองให้เหมาะสมนั้นมีโอกาสน้อยมาก การผ่าเหล่านั้นทุกลักษณะในแต่ละยีนส์ย่อมมีโอกาสที่จะเกิดความเปลี่ยนแปลงไปจากเดิมได้พอๆกัน และถ้าเหมาะสมกับสภาพแวดล้อมขณะนั้นก็จะคงอยู่ต่อไป แต่ถ้าการเปลี่ยนแปลงเป็นลักษณะที่เกิดผิดจังหวะคือไม่เหมาะสมกับสภาพขณะนั้นๆ ก็จะไม่ถูกคัดเลือกและหายไป ซึ่งข้อสรุปนี้ได้จากการทดลองโดยการนำยีนส์ของแบคทีเรียมาผสมกัน และจัดสภาพแวดล้อมที่ไม่อำนวยต่อการผสมกันแล้ว ยีนส์จะปรับตัวเองเพื่อเร่งขบวนการผสมพันธ์จนได้ผลดีในรุ่นหลังๆ ตัวอย่างของการผ่าเหล่าในอดีตคือการกำเนิดของปลาทองนั้น มีต้นกำเนิดมาจากการกลายพันธุ์หรือผ่าเหล่าของปลาซิว และในปัจจุบันลักษณะใหม่ที่เกิดจากการผ่าเหล่าก็ยังคงมีให้เห็นอยู่ เช่น ความสามารถของเชื้อโรคแบคทีเรียในการต้านทานต่อยาฆ่าเชื้อ หรือเซลผิดปกติอันเกิดจากกัมมันตภาพรังสีต่างๆ ซึ่งมีผลต่อสารภายในเซลทำให้ลักษณะของยีนส์ในเซลเปลี่ยนไป

2.2 จีเนติก อัลกอริทึมเบื้องต้น

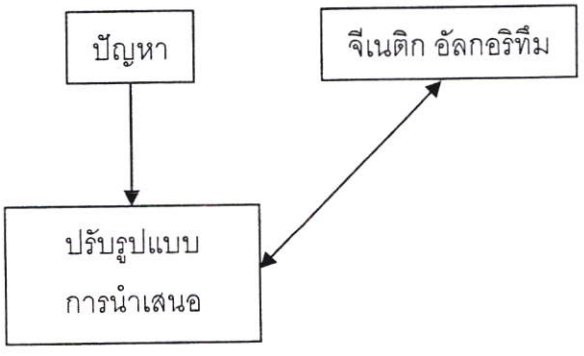
ปี ค.ศ. 1975 John Holland เริ่มสนใจศึกษาในทฤษฎีวิวัฒนาการทางธรรมชาติ (Natural Evolution) ในการกำเนิดประชากร (Population) สิ่งมีชีวิตในรุ่นต่อๆไป โดยกระบวนการธรรมชาติทางชีววิทยาประกอบด้วยการคัดเลือกทางธรรมชาติ (Natural Selection) คือสิ่งมีชีวิตใดแข็งแรงกว่าย่อมมีโอกาสอยู่รอดได้มากกว่านั่นหมายถึงการมีโครโมโซมซึ่งประกอบด้วยยีนส์ต่างๆ ที่มี

ลักษณะที่ดี นั้นจะมีโอกาสอยู่รอดได้มากกว่า โครโมโซมที่สามารถอยู่รอดได้ก็จะถูกถ่ายทอดยีนส์ที่มีลักษณะที่ดีเหล่านั้นไปยังลูกหลานได้มากกว่าเช่นกัน และกระบวนการทางพันธุศาสตร์ (Genetic Operation) คือการกำเนิดโครโมโซมใหม่โดยการผสมพันธ์เพื่อถ่ายทอดยีนส์จากการครอสโอเวอร์ หรือกลายพันธุ์จากมิวเตชัน

จากความเชื่อในวิวัฒนาการทางธรรมชาติ ที่แสดงถึงคุณลักษณะที่เป็นอยู่ของสิ่งมีชีวิตโดยการถ่ายทอดลักณะต่างๆบนโครโมโซมนั้นมีคุณสมบัติทั่วไปที่ยอมรับกันคือ

1. วิวัฒนาการเป็นผลที่เกิดขึ้นเนื่องจากความเปลี่ยนแปลงบนโครโมโซม ที่เป็นอยู่ซึ่งแสดงลักษณะของสิ่งมีชีวิตนั้นๆ
2. ธรรมชาติทางการคัดเลือกมีความสัมพันธ์กับโครโมโซมที่แสดงถึงประสิทธิภาพของโครงสร้างที่ดี ที่จะคัดเลือกเพื่อถ่ายทอดส่วนของโครงสร้างที่ดี
3. การถ่ายทอดในขณะที่เกิดวิวัฒนาการนั้น โครโมโซมพ่อและแม่มีการแลกเปลี่ยนส่วนโครงสร้างกันเพื่อสร้างโครโมโซมลูก และเหตุผลที่ทำให้เกิดโครโมโซมลูกที่แตกต่างออกไปคือขบวนการผ่าเหล่า
4. วิวัฒนาการทางธรรมชาติมิได้เป็นสิ่งที่เกิดจากความจดจำ แต่เป็นกระบวนการที่เกิดจากโครงสร้างต่างๆในโครโมโซมที่เหมาะสมกับสภาพแวดล้อมที่เกิดขึ้นในขณะนั้น

Holland คิดว่าแนวความคิดจากคุณสมบัติเหล่านี้ น่าจะนำมาปรับใช้กับคอมพิวเตอร์ให้ช่วยแก้ปัญหาที่ยุ่งยากต่างๆในการหาคำตอบที่ดีที่สุด เขาจึงได้ทำการวิจัยโดยจำลองแบบเพื่อทดลองกับปัญหาแบบต่างๆ โดยมีจุดมุ่งหมายที่จะศึกษาระบบปรับปรุงการประมวลผลเอง (self adaptive process) และเพื่อสร้างโปรแกรมระบบผู้เชี่ยวชาญ (artificial system software) เพื่อแก้ปัญหา โดยอาศัยแนวความคิดของระบบทางธรรมชาติ และค้นพบวิธีการใหม่ซึ่งเรียกว่า จีเนติดอัลกอริทึม (Genetic Algorithms: GA)



รูปที่ 2.2 แสดงหลักการเบื้องต้นของ GA

รูปที่ 2.2 แสดงหลักการเบื้องต้นในการใช้ GA แก้ปัญหา โดยจะต้องมีการปรับปรุงรูปแบบปัญหาในการนำเสนอ GA ในลักษณะที่เหมาะสม เพราะ GA เป็นวิธีการค้นหาคำตอบโดยอาศัยวิธีการเลือกแบบการคัดเลือกทางธรรมชาติ และธรรมชาติทางพันธุกรรมโดยการรวมกันหรือสลับเปลี่ยนตัวแปรต่างๆอันเป็นองค์ประกอบโครงสร้างของปัญหาที่ให้คำตอบที่ต้องการ ซึ่งอาศัยหลักการสุ่ม เพื่อปรับปรุงความสามารถในการค้นหาคำตอบที่ดีขึ้น การค้นหาคำตอบจากรุ่นหนึ่งไปรุ่นถัดไปตามวิวัฒนาการทางธรรมชาตินั้น คำตอบในรุ่นใหม่เกิดขึ้นจากการสร้างความสัมพันธ์ของโครงสร้างต่างๆ ที่ประกอบด้วยค่าตัวแปรที่เหมาะสมดีในรุ่นก่อน ดังนั้นจึงทำให้ได้คำตอบที่ดีขึ้น จะเห็นได้ว่าวิธีการพื้นฐานของ GA เป็นการสุ่ม แต่มีหลักการและประสิทธิภาพจากการคาดเดาคำตอบใหม่จากสถิติคำตอบเดิมที่ดี ซึ่งแตกต่างจากวิธีการทั่วไปคือ

1. GA ค้นหาคำตอบภายใต้โครงสร้างของปัญหาอันเกิดจากการกำหนดรหัส (coding) รูปแบบโครงสร้างจากกลุ่มตัวแปรต่างๆของปัญหานั้น ไม่ใช่ค้นหาคำตอบจากค่าของกลุ่มตัวแปรนั้น
2. GA ค้นหาคำตอบโดยพิจารณาจากประชากรคำตอบ หรือกลุ่มคำตอบ ไม่ใช่พิจารณาจากคำตอบใดคำตอบหนึ่ง
3. GA ค้นหาคำตอบจากผลลัพธ์ของกลุ่มค่าตัวแปรที่เป็นฟังก์ชันเป้าหมายของปัญหา
4. GA ค้นหาคำตอบโดยอาศัยการถ่วงน้ำหนักความเหมาะสมของแต่ละคำตอบจากกลุ่มคำตอบนั้นๆ

2.3 ฟังก์ชันเป้าหมายกับฟังก์ชันความเหมาะสม

การหาคำตอบที่ดีที่สุดของปัญหาของ GA มีพื้นฐานอยู่บนผลลัพธ์จากการหาคำตอบที่ผ่านมา วิธีการของ GA จะไม่พิจารณาจากขั้นตอนของการแก้ปัญหา แต่จะพิจารณาโดยตัดสินว่าคำตอบใหม่ที่ได้รับดีขึ้นหรือไม่ หรือเป็นคำตอบที่ใกล้เคียงคำตอบที่ต้องการหรือไม่ จากฟังก์ชันเป้าหมาย (Object Function: f) เนื่องจากแต่ละปัญหาจะสามารถกำหนดฟังก์ชันเป้าหมายซึ่งเป็นฟังก์ชันที่แสดงความสัมพันธ์ของแต่ละตัวแปร พารามิเตอร์ เงื่อนไข หรือข้อกำหนดต่างๆ ของปัญหานั้นๆ ที่ระบุคำตอบใดคำตอบหนึ่งที่สามารถเป็นไปได้ ณ.ค่าพารามิเตอร์ เงื่อนไข หรือข้อกำหนดชุดดังกล่าว สำหรับฟังก์ชันความเหมาะสม (Fitness Function: F) เป็นฟังก์ชันที่กำหนดค่าความเหมาะสม (fitness) ของแต่ละโครโมโซมเปรียบเสมือนค่าความสามารถในการอยู่รอดของแต่ละโครโมโซมและเป็นฟังก์ชันที่กำหนดโอกาส หรือสัดส่วนที่แต่ละโครโมโซมที่เหมาะสมจะถูกคัดเลือกมากน้อยเพียงใด นั่นคือฟังก์ชันความเหมาะสมจะเป็นฟังก์ชันที่แสดงถึงค่าของคำตอบที่เกิดขึ้นจากชุดตัวแปรของปัญหาของโครโมโซมนั้นดีเพียงใด โดยทั่วไปแล้วเรามักใช้

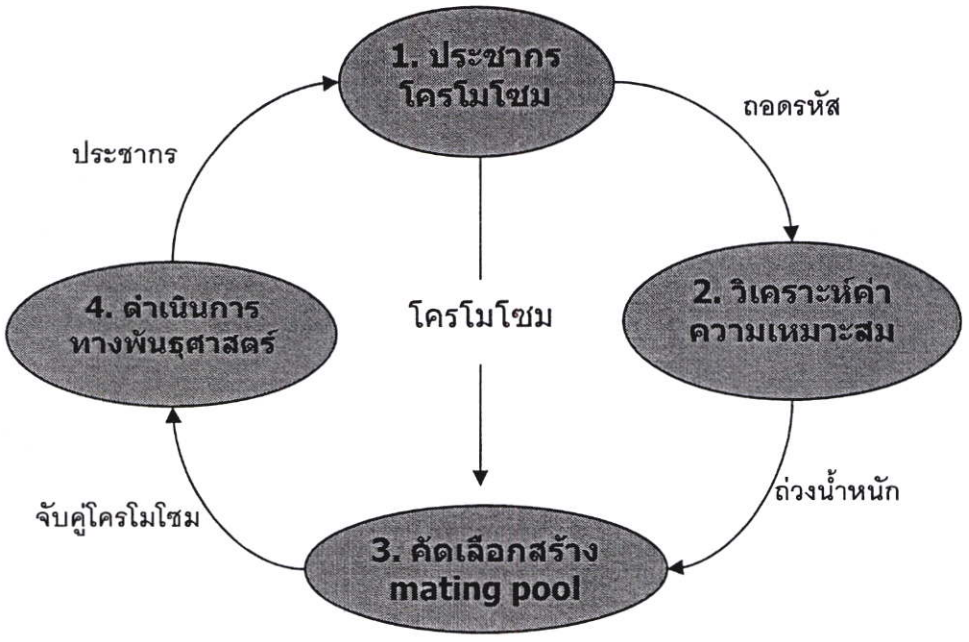
ฟังก์ชันเป้าหมายเป็นฟังก์ชันความเหมาะสม หรืออาจใช้ฟังก์ชันเป้าหมายที่ถูกปรับให้เหมาะสมกับการนำเสนอ GA เป็นฟังก์ชันความเหมาะสมก็ได้

2.4 รูปแบบโครโมโซม

เราทราบกันแล้วว่าวิวัฒนาการทางธรรมชาติชีววิทยานั้นเป็นความเปลี่ยนแปลงของสิ่งมีชีวิตที่เกิดขึ้นในโครโมโซม ดังนั้นจุดเริ่มต้นของการจำลองแบบทางธรรมชาติของ GA เพื่อใช้แก้ปัญหาจึงเริ่มจากการมองปัญหาเทียบเท่ากับโครโมโซม โดยโครโมโซมจะประกอบด้วยยีนส์ลักษณะต่างๆ ซึ่งหมายถึงลำดับข้อมูลต่างๆ ที่จะแปลความหมายแล้วให้คำคำตอบของปัญหาค่าหนึ่ง การมองภาพยีนส์ของ GA ให้เปรียบเสมือนยีนส์ทางพันธุกรรมที่แสดงความหมายหรือเป็นตัวแทนคำตอบใดคำตอบหนึ่ง หรือลักษณะใดลักษณะหนึ่งทางกรรมพันธุ์ ในทางพันธุศาสตร์นั้นยีนส์เป็นตัวแสดงลักษณะที่อยู่รอดในสภาพแวดล้อมขณะนั้น สำหรับ GA นั้นยีนส์เป็นตัวแสดงค่าคำตอบของปัญหาที่แปรผันไปตามการประยุกต์ใช้งาน ซึ่งโดยทั่วไปยีนส์หมายถึงตัวแปร พารามิเตอร์ เงื่อนไข หรือข้อกำหนดต่างๆที่เป็นองค์ประกอบของปัญหา ดังนั้นการกำหนดรูปแบบโครโมโซมของแต่ละปัญหาโดยการแปลงตัวแปร พารามิเตอร์ เงื่อนไข หรือข้อกำหนดต่างๆให้อยู่ในรูปลำดับของยีนส์บนโครโมโซม หรือเรียกว่าสตริง (string) อันประกอบด้วยบิต (bit) หรือเรียกว่าอักขระ (character) ซึ่งลักษณะต่างๆที่เป็นได้ของแต่ละยีนส์คือค่าของบิต (bit Value) หรือค่าตัวแปร พารามิเตอร์ต่างๆ ที่เป็นไปได้ และรูปแบบของค่าบิตที่จัดเรียงบนโครโมโซมคือ ยีโนไทป์ (genotype) ที่จะแสดงถึงค่าของตัวแปรพารามิเตอร์ต่างๆ ที่เป็นไปได้ชุดหนึ่งหรือฟีโนไทป์ (phenotype) นั่นเอง การกำหนดรูปแบบโครโมโซมของปัญหาให้เป็นตามแบบธรรมชาติ โดยกำหนดรหัสในรูปแบบตัวอักษรในช่วงที่จำกัดตามค่าตัวแปรหรือพารามิเตอร์ และประกอบรวมกันในจำนวนยีนส์หรือความยาวของโครโมโซมที่คงที่ เช่น หากต้องการหาค่าสูงสุดของฟังก์ชันทางคณิตศาสตร์ $y = x^2$ ที่ x เป็นจำนวนเต็มอยู่ในช่วง I[0,31] แล้ว วิธีการของ GA ในการแก้ปัญหาโดยกำหนดรูปแบบโครโมโซมจากการกำหนดรหัสตัวแปร x เป็นตัวเลขไบนารี 0 หรือ 1 จำนวน 5 ตำแหน่ง ซึ่ง x จะมีค่าตั้งแต่ 00000 ถึง 11111 เป็นค่า 0 ถึง 31 ตามต้องการ เป็นต้น

2.5 วัฏจักรจีเนติก อัลกอริทึม

เมื่อกำหนดรูปแบบโครโมโซมและฟังก์ชันความเหมาะสมของปัญหาแล้ว GA จะสามารถประมวลผลหาคำตอบของปัญหาได้ โดยสร้างวิวัฒนาการกลุ่มคำตอบในรุ่นต่อไปตามวัฏจักรการทำงานของ GA (Genetic Algorithm Cycle) ดังรูปที่ 2.3 ซึ่งมี 4 ขั้นตอน คือ



รูปที่ 2.3 แสดงวัฏจักรจีเนติก อัลกอริทึม

1. สร้างประชากรโครโมโซมรุ่นเก่าตามรูปแบบโครโมโซมที่กำหนดไว้ โดยประชากรต้นกำเนิด (Initial Population) เกิดจากการสร้างชุดโครโมโซมต้นกำเนิด จากการสุ่มสร้างค่าแต่ละบิตของแต่ละโครโมโซม
2. วิเคราะห์ค่าความเหมาะสมของแต่ละโครโมโซม โดยถอดรหัสค่าตัวแปรและพารามิเตอร์ต่างๆของแต่ละบิตในโครโมโซม และคำนวณค่าความเหมาะสมจากฟังก์ชันความเหมาะสมที่กำหนดไว้
3. สร้าง mating pool โดยชุดโครโมโซมต้นแบบหรือชุดโครโมโซมพ่อ-แม่ที่สามารถอยู่รอดเป็นต้นแบบ ซึ่งอาศัยการจำลองการคัดเลือกทางธรรมชาติ โดยพิจารณาถ่วงน้ำหนักจากค่าความเหมาะสมของแต่ละโครโมโซม หากโครโมโซมใดมีค่าความเหมาะสมมากก็จะมีโอกาสถูกคัดเลือกเป็นต้นแบบมาก
4. ดำเนินการทางพันธุศาสตร์โดยสุ่มจับคู่โครโมโซมต้นแบบใน mating pool เพื่อสร้างประชากรโครโมโซมรุ่นใหม่ ซึ่งตัวดำเนินการทางพันธุศาสตร์ประกอบด้วย ครอสโอเวอร์ โดยการแลกเปลี่ยนค่าบิตบางส่วนของโครโมโซมซึ่งกันและกัน หรือมิวเตชันโดยสุ่มเปลี่ยนค่าบิตบางบิตของแต่ละโครโมโซม เป็นต้น

การค้นหาคำตอบของ GA จะประมวลผลซ้ำตามวัฏจักร GA จนกว่าจะได้รับคำตอบที่พอใจตามกฎเกณฑ์ที่ตั้งไว้ หรือในระยะเวลาตามจำนวนรุ่นที่ดำเนินการที่ต้องการ ซึ่งแสดงอัลกอริทึมการทำงานของ GA ดังนี้

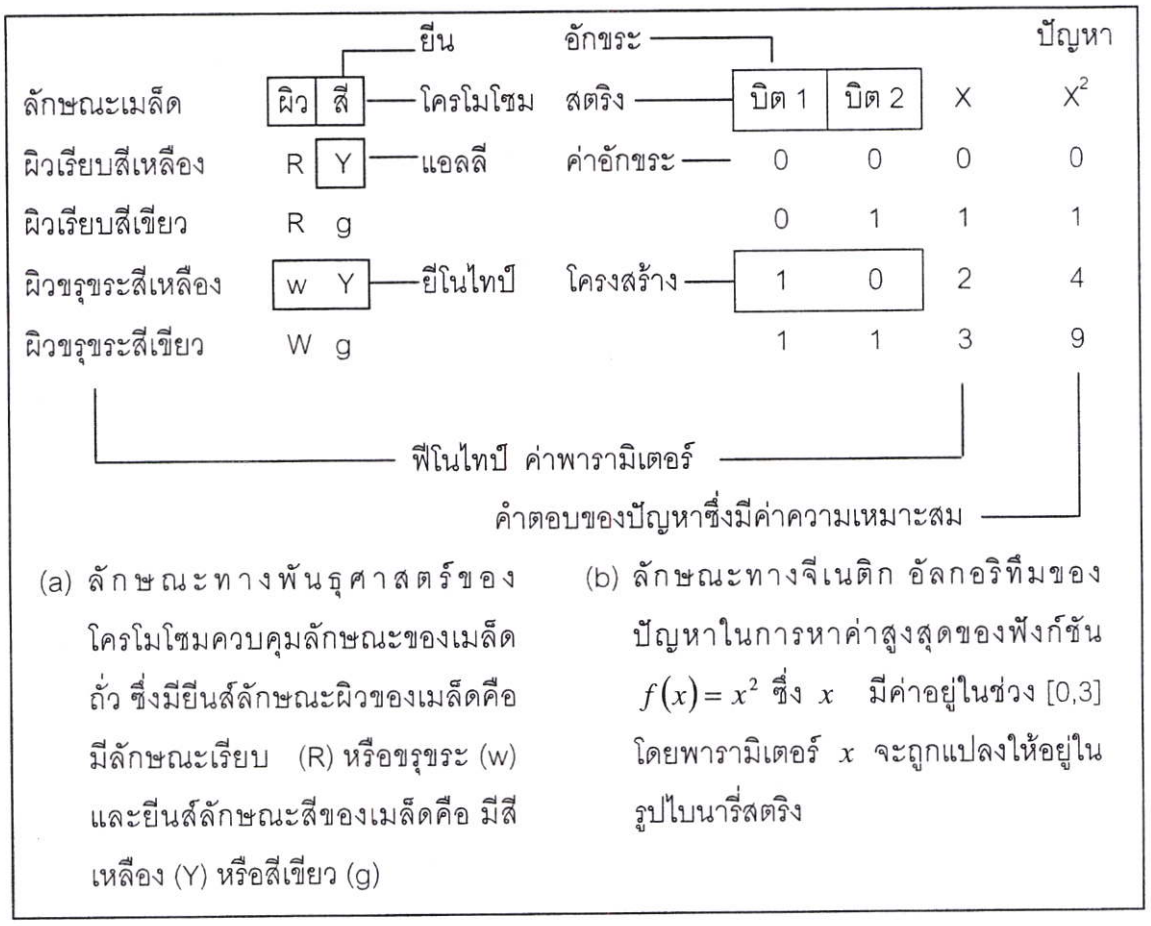
อัลกอริทึม GA

```
BEGIN
   F :=0;
   // สร้างประชากรโครโมโซมต้นกำเนิดโดยการสุ่ม
   Initpopulation P(t);
   // วิเคราะห์ค่าความเหมาะสมของแต่ละโครโมโซมในประชากรต้นกำเนิด
   Evaluate P(t);
   // ตรวจสอบเงื่อนไขความพอใจ (เช่น เวลา, ค่าความเหมาะสม เป็นต้น)
   while not terminate
   begin
         t := t+1;
         // คัดเลือกโครโมโซมต้นแบบจากประชากรรุ่นก่อน
         P'(t) := Selectparents P(t-1);
         // แลกเปลี่ยนบางส่วนของยีนส์ภายในโครโมโซมต้นแบบ
         Recombine P'(t);
         // มิวเตชันโครโมโซมต้นแบบ
         Mutate P'(t);
         // วิเคราะห์ค่าความเหมาะสมของประชากรรุ่นใหม่
         Evaluate P'(t);
         // ประชากรรุ่นใหม่กลายเป็นประชากรรุ่นเก่าต่อไป
         P(t) :=P'(t);
   end;
END.
```

2.6 พันธุศาสตร์ทางชีววิทยา กับจีเนติก อัลกอริทึม

เพื่อเปรียบเทียบลักษณะโครงสร้างทางพันธุศาสตร์ กับจีเนติก อัลกอริทึม แล้วเรากล่าวโดยสรุปได้คือ ในทางพันธุศาสตร์ แต่ละโครโมโซม ประกอบด้วยหน่วยเก็บลักษณะ หรือยีนส์ ซึ่งเก็บ

ค่าแสดงลักษณะ หรือแอลลี และแต่ละแบบของยีนส์เรียกว่ายีโนไทป์ ซึ่งแสดงลักษณะภายนอกที่ปรากฏเรียกว่า ฟีโนไทป์ ดังรูปที่ 2.4(a)



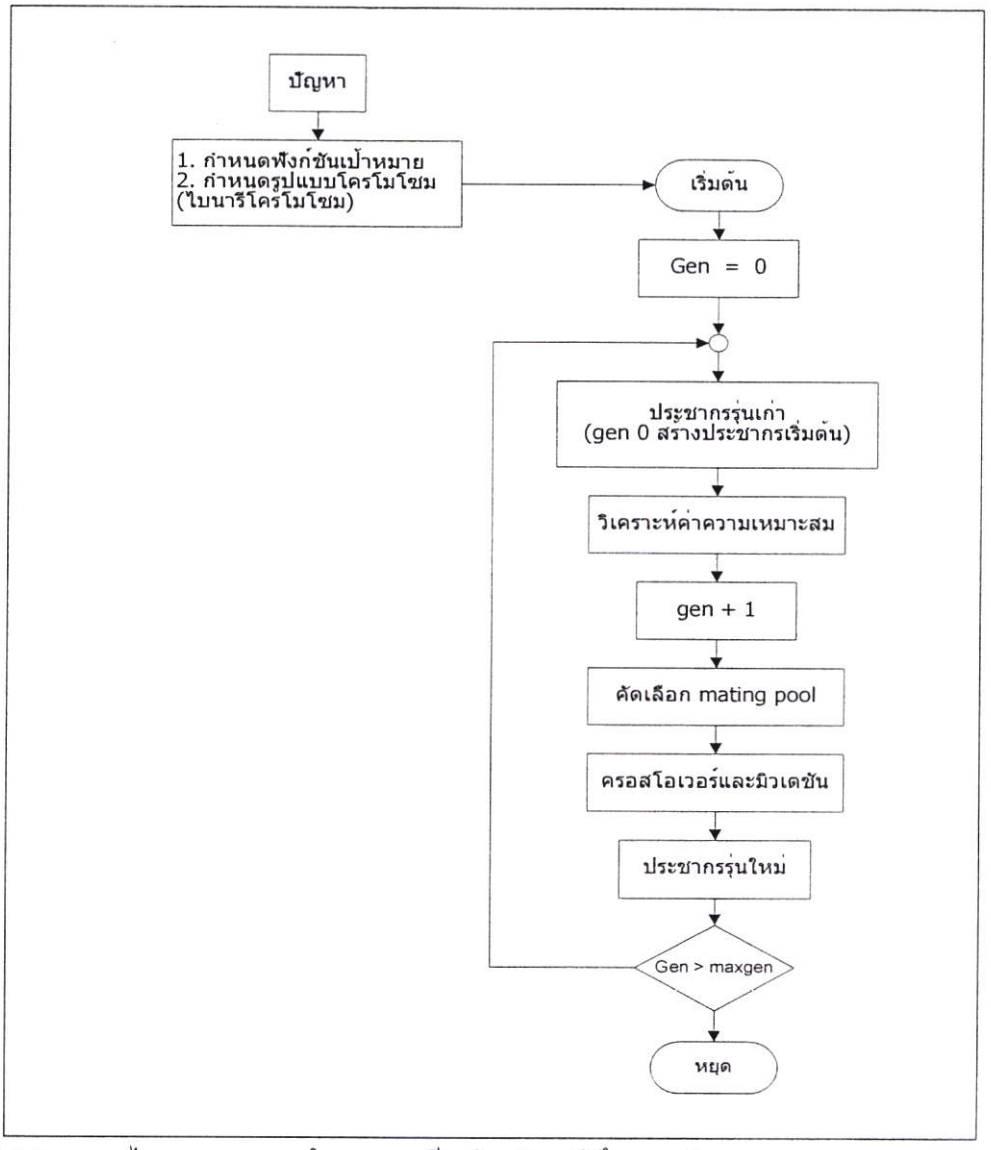
รูปที่ 2.4 แสดงรายละเอียดทางพันธุศาสตร์ กับจีเนติก อัลกอริทึม

สำหรับในทางจีเนติก อัลกอริทึม ตัวแปรหรือพารามิเตอร์ของปัญหาจะถูกแปลงให้อยู่ในรูปของสตริง ซึ่งมักเรียกกันว่าโครโมโซม ประกอบด้วยอักขระ หรือบิต แต่ละตำแหน่งของโครโมโซมจะเก็บค่าอักขระ หรือ ค่าของบิตที่แสดถึงโครงสร้างของแต่ละโครโมโซมที่มีค่าตัวแปรหรือพารามิเตอร์ของปัญหาแตกต่างกัน และเป็นตัวกำหนดค่าความเหมาะสมตามฟังก์ชันความเหมาะสมของแต่ละปัญหา ดังรูปที่ 2.4(b) ซึ่งสรุปความหมายเปรียบเทียบคำศัพท์ ที่ใช้ในทางพันธุศาสตร์กับทางจีเนติก อัลกอริทึม ดังตารางที่ 2.1

ตารางที่ 2.1 คำศัพท์ทางพันธุศาสตร์ กับคำศัพท์ทางจีเนติก อัลกอริทึม

Natural Genetic	Genetic Algorithm
chromosome	string
gene	character, bit
allele	character value, bit value
locus	string position
genotype	structure
phenotype	a decode structure

2.7 จีเนติก อัลกอริทึมแบบง่าย



รูปที่ 2.5 แสดงไดอะแกรมการทำงานของจีเนติก อัลกอริทึมแบบง่าย

GA ในยุคเริ่มแรกของ Holland นั้นเป็นจีเนติกอันกอริทึมแบบง่าย (Simple Genetic Algorithm: SGA) ซึ่งมีขั้นตอนพื้นฐานที่มีกระบวนการไม่มากนักและง่ายในการศึกษาความเข้าใจแต่ละขั้นตอนการทำงานของ GA เพื่อแก้ปัญหาในการหาคำตอบ แสดงดังไดอะแกรมในรูปที่ 2.5 แบ่งออกเป็น 2 ส่วนคือ ขั้นเตรียมการและขั้นตอนการทำงาน

สำหรับในส่วนของขั้นเตรียมการนั้นเป็นส่วนของการปรับรูปแบบของปัญหาให้เหมาะสมสำหรับการนำเสนอ GA เพื่อใช้ในการแก้ปัญหานั้นๆ ประกอบด้วย

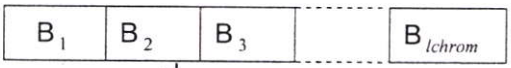
1. **กำหนดฟังก์ชันความเหมาะสม** เพื่อความสะดวกและง่ายต่อความเข้าใจในขั้นตอนการทำงานต่างๆ จะกำหนดตัวอย่างปัญหาสำหรับอธิบายรายละเอียดการหาคำตอบของ SGA คือ ปัญหาการหาค่าสูงสุดของฟังก์ชัน $y = x^2$ ที่ x มีค่าระหว่างจำนวนเต็ม I[0,31] ดังนั้น

ฟังก์ชันเป้าหมาย คือ $f = x^2$

และกำหนดให้ฟังก์ชันความเหมาะสมคือ $F = x^2$

ซึ่งคำตอบที่ดีที่สุด คือ ค่า x ที่มีค่าความเหมาะสมสูงสุด $(\max(f))$

2. **กำหนดรูปแบบโครโมโซม** รูปแบบโครโมโซมของ SGA นั้นเป็นแบบไบนารี โดยค่าตัวแปรหรือพารามิเตอร์ของปัญหาจะถูกแปลงให้อยู่ในรูปของไบนารีโครโมโซม คือประกอบด้วยบิตที่มีค่าเป็น 0 หรือ 1 ซึ่งเป็นค่าในเลขฐานสอง และมีความยาว (Chromosome Length: lchrom) ตามแต่จะกำหนด ซึ่งแสดงด้วยสัญลักษณ์ได้ดังนี้



ซึ่ง $B_i \in I\,[0\ ,\ 1]$

วิธีการเข้ารหัสแบบไบนารีโดยแปลงค่าพารามิเตอร์ x ให้อยู่ในรูปไบนารีบิต 5 บิต (lchrom = 5) ดังนั้นโครโมโซมของปัญหาจะมีค่าอยู่ในช่วง 00000 ถึง 11111 ซึ่งเมื่อถอดรหัสแล้วจะทำให้ x มีค่าอยู่ในช่วง 0 ถึง 31 ตามที่ต้องการ

ในส่วนของรายละเอียดขั้นตอนการทำงานของ SGA จะเป็นขั้นตอนพื้นฐานเบื้องต้นแบบง่ายประกอบด้วย

1. **ประชาการรุ่นเก่า** (Old Population) เป็นชุดโครโมโซมที่จะถูกคัดเลือกไปเป็นต้นแบบสำหรับสร้างประชาการรุ่นใหม่ (New Population) ในวิวัฒนาการรุ่น (generation: gen) ต่อไป

โดยประชากรเริ่มต้นที่ gen = 0 จะถูกสร้างขึ้นโดยการสุ่มตามจำนวนโครโมโซมในแต่ละรุ่น (Population Size: popsize) ที่กำหนด

ตัวอย่าง :

ลำดับ	โครโมโซม
1	01110
2	11001
3	01000
4	10011

ชุดโครโมโซมในรุ่นเริ่มต้นนี้เป็นชุดโครโมโซมที่กำหนดให้ในแต่ละรุ่นประกอบด้วย 4 โครโมโซม ซึ่งแต่ละโครโมโซมเกิดจากการสุ่มค่าไบนารี 0 หรือ 1 จำนวน 5 ครั้ง

2. **วิเคราะห์ค่าความเหมาะสม** เป็นขั้นตอนของการถอดรหัสจากรูปแบบโครโมโซมที่กำหนดไว้ เพื่อคำนวณค่าความเหมาะสมตามฟังก์ชันความเหมาะสมของปัญหาของแต่ละโครโมโซม ในที่นี้ฟังก์ชันเป้าหมายหรือฟังก์ชันความเหมาะสม คือ $F = x^2$ ดังนั้นการวิเคราะห์ค่าความเหมาะสมของ SGA โดยถอดรหัสเลขฐาน 2 ของแต่ละโครโมโซมเป็นค่าของตัวแปร x และคำนวณค่าความเหมาะสมคือคือ x^2

ตัวอย่าง :

ลำดับ	โครโมโซม	X	ค่าความเหมาะสม
1	01110	14	196
2	11001	25	625
3	01000	8	64
4	10011	19	361

ชุดโครโมโซมในรุ่นเริ่มต้นมีค่าความเหมาะสมเป็น 196 , 625 , 64 และ 361 ตามลำดับ

3. **การคัดเลือก** เป็นขั้นตอนที่จำลองแบบการคัดเลือกทางธรรมชาติเพื่อสร้าง mating pool โดยคัดเลือกชุดโครโมโซมรุ่นเก่าให้เป็นโครโมโซมต้นแบบหรือโครโมโซมพ่อ-แม่ เพื่อใช้สร้างโครโมโซมลูกเป็นรุ่นต่อไป สำหรับการคัดเลือกของ SGA เป็นแบบอ้างอิงค่าความเหมาะสม (Fitness-based Selection) โดยใช้ค่าความเหมาะสมเป็นตัวตัดสินว่า โครโมโซมใดในรุ่นเก่ามีโอกาสจะถูกเลือกเป็นโครโมโซมพ่อ-แม่มากน้อยเพียงใด โครโมโซมที่มีค่าความเหมาะสมที่ดีจะถูกกำหนดน้ำหนักค่าความน่าจะเป็นที่จะถูกเลือกแต่ละครั้งสูง การกำหนดค่าความน่าจะเป็นที่จะถูก

เลือกต่อการสุ่มเลือกแต่ละครั้ง (Probability of Selected Valued : pselect) ของแต่ละโครโมโซม โดยกำหนดจากค่าความเหมาะสม เทียบกับผลรวมของค่าความเหมาะสมทั้งหมด ดังสมการที่ 2.1

$$pselect_i = \frac{F_i}{\sum F} \qquad (2.1)$$

ซึ่งสามารถคำนวณค่าความคาดหวังว่าจะสุ่มได้ (Expected Value : E) ของแต่ละโครโมโซม ในแต่ละรุ่นดังสมการที่ 2.2

$$E_i = pselect_i * popsize = \frac{F_i}{F} \qquad (2.2)$$

สำหรับวิธีการสุ่มโครโมโซมต้นแบบของ SGA เป็นแบบจำลองการหมุนวงล้อถ่วงน้ำหนัก (Roullet Wheel : RW) ซึ่งกำหนดขนาดแต่ละช่องของวงล้อนั้นตามค่าความน่าจะเป็นที่จะสุ่มได้ ในแต่ละครั้งของแต่ละโครโมโซม ซึ่งมีวิธีการดังนี้

(1) หาค่าความเหมาะสมของแต่ละโครโมโซม

(2) หาค่าความน่าจะเป็นที่จะสุ่มได้ในแต่ละครั้งของแต่ละโครโมโซม

(3) หาความถี่สะสม (q) ของค่าความน่าจะเป็นของแต่ละโครโมโซมดังสมการที่ 2.3

$$q_i = \sum_{j=1}^{i} pselect_j \qquad (2.3)$$

(4) สร้างเลขสุ่มจำนวนจริง (r) มีค่าอยู่ในช่วง [0.0 ,1.0]

(5) เลือกโครโมโซมลำดับที่ r ซึ่ง r มีค่าอยู่ระหว่าง q_{i-1} และ q_i

ตัวอย่าง :

ลำดับ	โครโมโซม	X	ค่าความเหมาะสม (F)	ค่าความน่าจะเป็น ($pselect_i$)	จำนวนที่คาดหวัง (E_i)	จำนวนที่สุ่มได้จาก RW
1	01110	14	196	0.157	0.628	1
2	11001	25	625	0.502	2.008	2
3	01000	8	64	0.051	0.204	0
4	10011	19	361	0.290	1.160	1
	รวม		1246	1.000	4.000	
	ค่าเฉลี่ย		312	0.250	1.000	
	ค่าสูงสุด		625	0.502	2.008	

ตัวอย่างการกำหนดค่าความน่าจะเป็น โดยกำหนดจากค่าความเหมาะสมเทียบกับผลรวมของค่าความเหมาะสมทั้งหมด จะเห็นได้ว่าในการคัดเลือกโครโมโซมต้นแบบจาก 4 โครโมโซมนี้

โอกาสที่จะสุ่มได้โครโมโซมลำดับที่ 1 ต่อการสุ่มแต่ละครั้งเท่ากับ 0.157 และโอกาสที่จะสุ่มได้โครโมโซมลำดับที่ 2, 3, 4 ต่อการสุ่มแต่ละครั้งเท่ากับ 0.502, 0.051 และ 0.290 ตามลำดับ และจำนวนโครโมโซมต้นแบบที่สุ่มได้โดยจำลองการหมุนวงล้อดังนี้

ลำดับโครโมโซม	1	2	3	4
ค่าความเหมาะสม (F)	196	625	64	361
ค่าความน่าจะเป็นที่สุ่มได้แต่ละครั้ง ($pselect_i$)	0.157	0.502	0.051	0.290
ความถี่สะสมค่าความน่าจะเป็น (q_i)	0.157	0.659	0.710	1.000
สร้างเลขสุ่มในการหมุนวงล้อแต่ละครั้ง (r)	0.333	0.844	0.456	0.128
ลำดับโครโมโซมที่ถูกเลือก($q_{i-1} \leq r \leq q_i$)	2	4	2	1

ซึ่งโครโมโซมที่สุ่มได้ถูกสร้างเป็นโครโมโซมต้นแบบใน mating pool โดยที่ โครโมโซมที่ 1 สุ่มได้ 1 ครั้ง โครโมโซมที่ 2 สุ่มได้ 2 ครั้ง โครโมโซมที่ 3 สุ่มได้ 0 ครั้ง (ไม่ถูกเลือก) โครโมโซมที่ 4 สุ่มได้ 1 ครั้ง จะเห็นได้ว่าโครโมโซมลำดับที่ 2 มีค่าความเหมาะสมสูงที่สุด ซึ่งหมายความว่าจะมีโอกาสถูกคัดเลือกมากที่สุด ส่วนโครโมโซมลำดับที่ 3 มีค่าความเหมาะสมต่ำมากจึงมีโอกาสที่จะไม่ถูกคัดเลือกเลย

4. **ดำเนินการทางพันธุศาสตร์** เป็นขั้นตอนที่จำลองแบบธรรมชาติทางพันธุกรรม ซึ่งตัวดำเนินการทางพันธุศาสตร์ของ SGA คือ ครอสโอเวอร์และมิวเตชัน โดยมีรายละเอียดดังนี้

4.1 **ครอสโอเวอร์** เป็นตัวดำเนินการในการแลกเปลี่ยนส่วนของโครโมโซมพ่อ-แม่ตามการกำหนดอัตราความน่าจะเป็นของการครอสโอเวอร์ (Probability of Crossover: P_C) เพื่อสร้างชุดโครโมโซมรุ่นใหม่หรือโครโมโซมลูก มีขั้นตอนการทำงานคือ

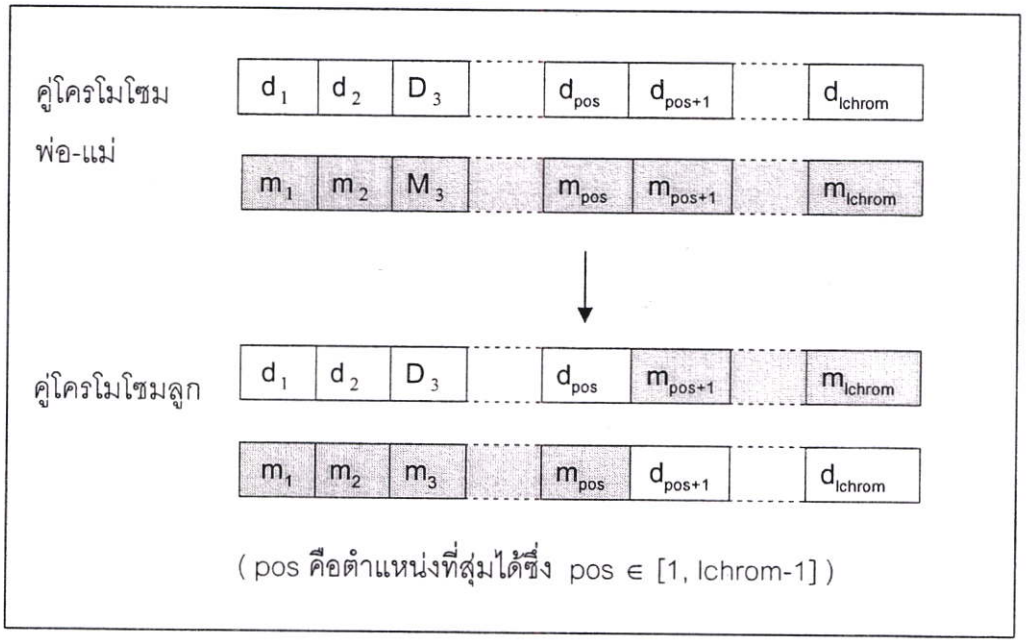
ขั้นตอนแรก : สุ่มจับคู่โครโมโซมพ่อ-แม่ใน mating pool ที่สร้างไว้จากการคัดเลือก

ขั้นตอนที่สอง : สร้างเลขสุ่มจำนวนจริง (r) มีค่าอยู่ในช่วง [0.0, 1.0] โดยถ้า $r \leq P_c$ แล้วโครโมโซมพ่อ-แม่นั้นจึงมีการครอสโอเวอร์

ขั้นตอนที่สาม : ครอสโอเวอร์โดยแลกเปลี่ยนส่วนของคู่โครโมโซมพ่อ-แม่นั้น ซึ่งการครอสโอเวอร์ของ SGA เป็นการครอสโอเวอร์แบบ 1 จุด (One-point Crossover) แสดงดังรูปที่ 2.6 ดังนี้

- สุ่มเลือกตำแหน่ง pos เป็นตำแหน่งที่จะครอสโอเวอร์ ซึ่ง pos มีค่าอยู่ในช่วง [1, lchrom-1]

- แลกเปลี่ยนค่าในแต่ละบิตของคู่โครโมโซมพ่อ-แม่ตั้งแต่ตำแหน่งที่ pos+1 ถึง lchrom ซึ่งจะทำให้เกิดโครโมโซมลูกใหม่ 2 โครโมโซม



รูปที่ 2.6 ครอสโอเวอร์แบบ 1 จุด

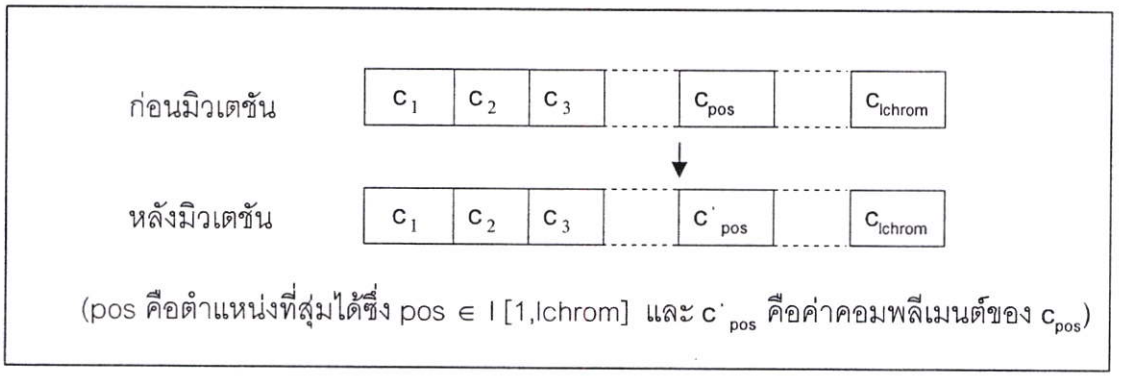
จำนวนการครอสโอเวอร์ในแต่ละรุ่นดำเนินการขึ้นอยู่กับการกำหนดค่า P_c ซึ่งแตกต่างกันในแต่ละปัญหา เช่น ถ้าจำนวนประชากรแต่ละรุ่น popsize เท่ากับ 30 โครโมโซม และกำหนดให้ Pc = 0.6 และจำนวนการครอสโอเวอร์ในแต่ละรุ่นเท่ากับ P_c*(popsize / 2) = 0.6*(30 / 2) = 9 ครั้ง (การครอสโอเวอร์ 1 ครั้ง เกิดจากโครโมโซม 2 โครโมโซม)

ตัวอย่าง : กำหนด P_c = 0.5 โครโมโซมพ่อ-แม่ใน mating pool ถูกคัดเลือกเพื่อครอสโอเวอร์ดังนี้

ลำดับที่คัดเลือก	Mating Pool	สุ่มจับคู่พ่อ-แม่	เลขสุ่ม (r)	ก่อนครอสโอเวอร์	สุ่มตำแหน่ง (pos)	หลังครอสโอเวอร์	X	ค่าความเหมาะสม (F)	ลำดับโครโมโซมลูก
2	11001	1,2	0.321	0 1 1 1 0	2	0 1 0 0 1	9	81	1
4	10011		≤0.5	1 1 0 0 1		1 1 1 1 0	30	*900*	2
2	11001	2,4	0.654	ไม่ครอสโอเวอร์		1 1 0 0 1	25	625	3
1	01110		> 0.5			1 0 0 1 1	19	361	4
						รวม		1967	
						ค่าเฉลี่ย		492	
						ค่าสูงสุด		900	

จากการสุ่มจับคู่โครโมโซมพ่อ-แม่ใน Mating pool ได้โครโมโซมลำดับที่ 1 คู่ลำดับที่ 2 และลำดับที่ 2 คู่กับลำดับที่ 4 แต่เฉพาะโครโมโซมคู่แรกจะเกิดครอสโอเวอร์ เนื่องจากเลขสุ่ม $r \leq 0.5$ ตามอัตราการครอสโอเวอร์ที่กำหนด โดยตำแหน่งในการครอสโอเวอร์ที่สุ่มได้คือ pos = 2 จะเห็นได้ว่าโครโมโซมลูกลำดับที่ 2 ที่เกิดขึ้นหลังจากครอสโอเวอร์มีค่าความเหมาะสมดีขึ้นกว่าโครโมโซมพ่อ-แม่ทั้งหมดในรุ่นก่อนเป็น 900 ซึ่งแสดงให้เห็นถึงการจำลองแบบกระบวนการครอสโอเวอร์รตามธรรมชาติทางพันธุศาสตร์ของ SGA ที่ช่วยสร้างคำตอบที่ดีขึ้น

4.2 **มิวเตชัน** เป็นตัวดำเนินการผ่าเหล่าตัวหนึ่งที่อาจช่วยให้โครโมโซมมีค่าเหมาะสมดีขึ้นหลังจากครอสโอเวอร์ โดยกลับค่าของบิตเป็นค่าใหม่ในตำแหน่งบิตที่สุ่มได้ ตามอัตราความน่าจะเป็นของการมิวเตชันในแต่ละบิต (Probability of Mutation: Pm) ที่กำหนด สำหรับการมิวเตชันของ SGA นั้นเป็นแบบไบนารีมิวเตชัน (Binary Mutation) โดยกลับค่าบิตเป็นค่าคอมพลีเมนต์ คือจาก 0 เป็น 1 หรือจาก 1 เป็น 0 ดังรูปที่ 2.7



รูปที่ 2.7 ไบนารีมิวเตชัน

จำนวนการมิวเตชันในแต่ละรุ่นขึ้นอยู่กับการกำหนดค่า P_m ซึ่งแตกต่างกันในแต่ละปัญหา เช่น ถ้าจำนวนประชากรแต่ละรุ่น popsize เท่ากับ 30 โครโมโซม ซึ่งแต่ละโครโมโซมประกอบด้วย 5 บิต และกำหนดให้ $P_m = 0.02$ แล้ว จำนวนการมิวเตชันในแต่ละรุ่นจะเท่ากับ P_m * popsize * lchrom = 0.02 * 30 * 5 = 3 บิต

ตัวอย่าง : กำหนด $P_m = 0.1$ ดำเนินการมิวเตชันโครโมโซมลูกที่ได้จากการครอสโอเวอร์ดังนี้

ลำดับ	ก่อน มิวเตชัน	เลขสุ่ม (r)	หลัง มิวเตชัน	X	ค่าความเหมาะสม(F)
1	0 1 0 0 1	0.581 0.346 **0.062** 0.785 0.401	0 1 1 0 1	13	169
2	1 1 1 1 0	0.829 0.534 0.947 0.308 0.277	1 1 1 1 0	30	900
3	1 1 0 0 1	0.398 0.646 0.494 0.765 **0.029**	1 1 0 0 1	24	576
4	1 0 0 1 1	0.175 0.335 0.837 0.577 0.308	1 0 0 1 1	19	361
			รวม		2006
			ค่าเฉลี่ย		*502*
			ค่าสูงสุด		900

จากการสุ่มตำแหน่งที่จะมิวเตชันโดยสร้างเลขสุ่ม r ของแต่ละตำแหน่งบิตในแต่ละโครโมโซมแล้ว ตำแหน่งบิตที่ 3 ของโครโมโซมลำดับที่ 1 และตำแหน่งบิตที่ 4 ของโครโมโซมลำดับที่ 3 เป็นตำแหน่งที่ $r \leq 0.1$ ตามอัตราที่กำหนด จึงเกิดการมิวเตชันขึ้นที่สองตำแหน่งดังกล่าว ทำให้โครโมโซมมีค่าความเหมาะสมจากเดิม 81 และ 625 เป็น 169 และ 576 ตามลำดับ จะเห็นได้ว่ามิวเตชันเป็นตัวดำเนินการที่อาจทำให้โครโมโซมมีค่าความเหมาะสมสูงขึ้นหรือลดลงได้ แต่อย่างไรก็ตามค่าเฉลี่ยของค่าความเหมาะสมดีขึ้นจาก 492 เป็น 502 แสดงถึงการหาคำตอบของ SGA โดยส่วนมากดีขึ้น และความสำคัญของการหาคำตอบของ GA นั้น เป็นการหาคำตอบโดยพิจารณาจากคำตอบที่ดีขึ้น โดยการถ่ายทอดส่วนที่ดีให้กับประชากรรุ่นต่อไป

5. **ประชากรรุ่นใหม่** เป็นชุดโครโมโซมลูกที่เกิดจากขั้นตอนของวิวัฒนาการต่างๆ ทั้งหมด ซึ่งประชากรรุ่นใหม่ทั้งหมดที่เกิดขึ้น จะถูกถ่ายทอดกลายเป็นประชากรรุ่นเก่าสำหรับวิวัฒนาการในรุ่นถัดไป ซึ่งเรียกวิวัฒนาการแบบนี้ว่า การถ่ายทอดแบบทั่วไปหรือรีโพรดักชันแบบทั่วไป (General Reproduction) กระบวนการต่างๆจะถูกปฏิบัติซ้ำๆ จนกระทั่งถึงรุ่นที่มากที่สุด (max generation : maxgen) ที่ต้องการ

การทดลองเพื่อทดสอบการทำงานของ SGA ต่อไปนี้จะทดสอบโดยการหาค่าสูงสุดของฟังก์ชัน $y = x^n$ ที่ x มีค่าระหว่าง $I[1,2^l - 1]$ ซึ่ง l คือ จำนวนบิตของโครโมโซม ด้วยวิธีการของ SGA กำหนดให้ฟังก์ชันเป้าหมาย คือ $f = x^n$ เพื่อความสะดวกในการวิเคราะห์การทำงานของ SGA จึงปรับให้ค่าความเหมาะสมอยู่ในช่วง [0, 1] ซึ่งคำตอบที่ดีที่สุดจะมีค่าความเหมาะสมเป็น 1 ดังนั้น

ฟังก์ชันความเหมาะสม คือ $F=\left(\frac{x}{2^l-1}\right)^n$

```
-----------------------------------------------------------------
                        SGA Parameter
-----------------------------------------------------------------
Chromosome length                 = 30
Maximum # of Generation           = 80
Crossover probability (Pc)        = 0.8
Mutation probability (Pm)         = 0.033
-----------------------------------------------------------------

-----------------------------------------------------------------
                        Generation 0
-----------------------------------------------------------------

  #            Chromosome            Decoded Value  Fitness Value
  1) 100101000110111111001001011100    622588508       0.00430
  2) 010010000010111000100001111001    302745721       0.00000
  3) 011010111011101001001011110110    451842806       0.00017
  4) 011001110100110110000000110101    433283125       0.00011
  5) 100010100111001011010010001011    580695179       0.00214
  6) 110110000000000001101001011101    905976413       0.18288
  7) 100001010101101000111110100110    559320998       0.00147
  8) 000111111011000010001010111010    132915898       0.00000
  9) 110001000100011000110000111011    823233595       0.07018
 10) 010110100111100010010011101100    379462892       0.00003
 11) 000110011110011101011010010001    108648081       0.00000
 12) 110111001110110011111100011001    926629657       0.22912
 13) 000101110000010001000001111111    96538751        0.00000
 14) 001001011010110101010110001100    158029196       0.00000
 15) 010010000101011011000111100110    303411686       0.00000
 16) 111001011000101100100100101000    962775336       0.33593
 17) 110111000100011011010000110000    923907120       0.22248
 18) 110011110110110010010001000001    869999681       0.12195
 19) 111001010000000110011100100010    960522018       0.32815
 20) 011001111100011000011011010011    435259091       0.00012
 21) 011000111001110111100000100001    417822753       0.00008
 22) 100101011111000100000100111111    628900159       0.00475
 23) 000001100101001011111010001110    26525326        0.00000
 24) 010100101100111100100111001100    347326924       0.00001
 25) 100101001011011001001100011100    623743772       0.00438
 26) 110110000110111001011111001101    907777997       0.18655
 27) 011010110100001101110101111001    449895801       0.00017
 28) 011110001011110100010011110011    506414323       0.00054
 29) 001111111011100011011100011110    267269918       0.00000
 30) 111001000100010011101111000001    957430721       0.31774
-----------------------------------------------------------------
Note:    maxF   = 0.33593
   minF   = 0.00000
   avgF   = 0.06711
   sumF   = 2.01327
   nmuta  = 0
   ncross = 0
-----------------------------------------------------------------
```

a) วิวัฒนาการรุ่นที่ 0

รูปที่ 2.8 ผลการทำงานของ SGA ในการหาค่าสูงสุดของฟังก์ชั่น $y = x^n$ ที่ n = 10

Generation 1

#	Parents	Pos1	Pos2	Chromosome	Decoded Value	Fitness Value
1)	[16,30]	11	30	11100101100001001101111000001	962673601	0.33558
2)	[16,30]	11	30	11100100010011010010010010100	957565224	0.31819
3)	[30,12]	3	30	11111100111011001111100011001	1060847385	0.88620
4)	[30,12]	3	30	11000100010000100111011110000001	823212993	0.07017
5)	[9,19]	13	30	11001100010001100011000011011	856788027	0.10465
6)	[9,19]	13	30	11100101000000011001110010010	960522018	0.32815
7)	[9,30]	5	30	11000100010001111001100001110 11	823249979	0.07020
8)	[9,30]	5	30	11100101010001001111011110000001	961625025	0.33194
9)	[30,16]	10	30	11110100010010110010010010100 0	1024641320	0.62621
10)	[30,16]	10	30	10100101100001000110111000001	694229953	0.01277
11)	[26,9]	28	30	11011000011011100101111100 1101	907777997	0.18655
12)	[26,9]	28	30	11000101010001100011000011011	827427899	0.07384
13)	[19,9]	22	30	11100101000000011001010011011	960521531	0.32815
14)	[19,9]	22	30	11001100010001100111000010 0010	856792098	0.10465
15)	[12,30]	6	30	11011110010011001110111000001	932395969	0.24378
16)	[12,30]	6	30	11100100111011001111100011001	960184089	0.32700
17)	[16,16]	26	30	11100101100010110010010010100 0	962775336	0.33593
18)	[16,16]	26	30	11100101100010110010010010100 0	962775336	0.33593
19)	[6,17]	6	30	11011000010001101101000010 0000	907129888	0.18522
20)	[6,17]	6	30	11010100000000000110100101 1101	889199197	0.15170
21)	[16,19]	21	30	11100101100010110010010010 0010	962775330	0.33593
22)	[16,19]	21	30	11100101000000011001110010 1000	960522024	0.32815
23)	[9,30]	11	30	11000100010001001110101100 0001	823212737	0.07017
24)	[9,30]	11	30	11100100010001100011000010 1011	957451307	0.31781
25)	[17,12]	8	30	11011100111011001111100011001	926629657	0.22912
26)	[17,12]	8	30	11011100010001101101000011 0000	923907120	0.22248
27)	[16,16]	23	30	11100101100010110010010010100 0	962775336	0.33593
28)	[16,16]	23	30	11100101100010110010010010100 0	962776352	0.33594
29)	[9,9]	4	30	11000100010001100001000011101 1	823231547	0.07018
30)	[9,9]	4	30	11000100010001100111000011101 1	823237691	0.07019

Note: maxF=0.88620, minF=0.01277, avgF=0.25576, sumF=7.67270, nmuta=22, ncross=12

Generation 2

#	Parents	Pos1	Pos2	Chromosome	Decoded Value	Fitness Value
1)	[9,27]	11	30	11110100010010110010010010100 0	1024641320	0.62621
2)	[9,27]	11	30	11100101100011010010010010100 0	962808104	0.33605
3)	[28,4]	3	30	11100100010001001110111000001	957430721	0.31774
4)	[28,4]	3	30	11000101100010110011010010000 0	828558624	0.07486
5)	[3,21]	13	30	11110100111011001111100011001	1027292953	0.64261
6)	[3,21]	13	30	11100101100010110010010010 0010	962775330	0.33593
7)	[3,28]	5	30	11111100111011011111100011001	1060863769	0.88633
8)	[3,28]	5	30	11100100100010110011010010 0000	958582048	0.32158
9)	[27,9]	10	30	11110101100010110010010010100 0	1029884200	0.65900
10)	[27,9]	10	30	10110100010010111010010010100 0	756214056	0.03002
11)	[22,3]	28	30	11100101000000011001110010100 0	960522024	0.32815
12)	[22,3]	28	30	11111101111011001111100011001	1065041689	0.92187
13)	[21,3]	22	30	11100101100010110010110011001	962775833	0.33593
14)	[21,3]	22	30	11110100111011001011100100010	1027288866	0.64258
15)	[3,28]	6	30	11111111100000110011010010 0000	1071697184	0.98112
16)	[3,28]	6	30	11100100111011001111100011001	960184089	0.32700
17)	[9,6]	26	30	11110100010010110010010010 0010	1024641314	0.62621
18)	[9,6]	26	30	11100101000000011001110010100 0	960522024	0.32815
19)	[2,13]	6	30	11100101000000011001010010 1011	960521515	0.32815
20)	[2,13]	6	30	11101100010011010010010010100 0	991119656	0.44902
21)	[9,21]	21	30	11110100010010110010010010 0010	1024641314	0.62621
22)	[9,21]	21	30	11100101100010110010010010100 0	962775336	0.33593
23)	[3,28]	11	30	11111100111010110011000010 0000	1060817952	0.88595
24)	[3,28]	11	30	11100101100011001111100001001	962805513	0.33604
25)	[15,3]	8	30	11011110111011001111100011001	935018265	0.25073
26)	[15,3]	8	30	11111100010011001110111000001	1058225089	0.86453
27)	[9,9]	23	30	11110100010010110010010010100 0	1024641320	0.62621
28)	[9,9]	23	30	11110100010010110011010010 0000	1024642336	0.62622
29)	[3,3]	4	30	11111100111011001101110011001	1060845337	0.88618
30)	[3,3]	4	30	11111100111011001011100011001	1060843289	0.88616

Note: maxF=0.98112, minF=0.03002, avgF=0.52742, sumF=15.82267, nmuta=22, ncross=12

b) วิวัฒนาการรุ่นที่ 1 และ 2

รูปที่ 2.8 (ต่อ)

Generation 16

#	Parents	Pos1	Pos2	Chromosome	Decoded Value	Fitness Value
1)	[15,15]	26	30	111111100110110011110110000000	1067138432	0.94018
2)	[15,15]	26	30	111111101110110011110110000000	1069235584	0.95882
3)	[2,28]	2	30	111111101000110010110000010001	1067658257	0.94477
4)	[2,28]	2	30	101111101000110011111100000001	799227649	0.05220
5)	[16,15]	18	30	111111111110100100110110000000	1073368448	0.99653
6)	[16,15]	18	30	111111101110010011101100100001	1069103905	0.95764
7)	[5,9]	8	30	111111001100100100100110000000	1060260224	0.88130
8)	[5,9]	8	30	111111111110000100000111001001	1073234377	0.99528
9)	[20,18]	4	30	111111101110010011111101111011	1069105019	0.95765
10)	[20,18]	4	30	111101101110100100101000001001	1035618825	0.69663
11)	[16,15]	18	30	111111111100110100110110000000	1072909696	0.99228
12)	[16,15]	18	30	111111101110110011101100100111	1069234983	0.95881
13)	[11,18]	5	30	111111100110100110100110000001	1067084161	0.93970
14)	[11,18]	5	30	111111101110010011111101111001	1069105017	0.95765
15)	[16,5]	27	30	111111111100000100101100100001	1072712481	0.99045
16)	[16,5]	27	30	111111101110100100000100001001	1069170953	0.95824
17)	[16,9]	8	30	111110111100100100100110000000	1056065920	0.84705
18)	[16,9]	8	30	111111111100100100101100100001	1072843553	0.99167
19)	[5,1]	9	30	111111101110110011111100000001	1069235969	0.95882
20)	[5,1]	9	30	111111101110100100000110011001	1069171097	0.95824
21)	[10,15]	26	30	111011111110100100101101000000	1006259008	0.52252
22)	[10,15]	26	30	111011101110110011100110000001	1002125697	0.50144
23)	[15,11]	25	30	111111101110110011010110000000	1069233536	0.95880
24)	[15,11]	25	30	111111101110100110100110000001	1069181313	0.95833
25)	[5,1]	9	30	111111101110110011111100010001	1069235985	0.95882
26)	[5,1]	9	30	111111101110100100000110001000	1069171080	0.95824
27)	[16,18]	5	30	111111101110010011111101111001	1069105017	0.95765
28)	[16,18]	5	30	110111111100100100101101100001	938625889	0.26057
29)	[5,16]	15	30	111111101110100100101100100101	1069173541	0.95826
30)	[5,16]	15	30	111111111100100000010110001001	1072825737	0.99150

Note: maxF=0.99653, minF=0.05220, avgF=0.86533, sumF=25.96001, nmuta=31, ncross=13

Generation 17

#	Parents	Pos1	Pos2	Chromosome	Decoded Value	Fitness Value
1)	[14,14]	26	30	111111100110010011111101111001	1067007865	0.93903
2)	[14,14]	26	30	111111101110010011111101111001	1069105017	0.95765
3)	[2,27]	2	30	111111101110010010110001111001	1069100153	0.95760
4)	[2,27]	2	30	101111101110110011110110000000	800800128	0.05324
5)	[15,14]	18	30	111111111110000100111101111001	1073237881	0.99532
6)	[15,14]	18	30	111111101110110011101100100001	1069234977	0.95881
7)	[5,8]	8	30	111111011110000100000111001001	1064845769	0.92017
8)	[5,8]	8	30	111111111110000100110111000000	1073237440	0.99531
9)	[19,17]	4	30	111110111100100100100110000010	1056065922	0.84705
10)	[19,17]	4	30	111101101110110011111100000001	1035681537	0.69705
11)	[15,14]	18	30	111111111100010100111101111001	1072779129	0.99107
12)	[15,14]	18	30	111111101110010011101100100111	1069103911	0.95764
13)	[9,17]	5	30	111111100110010011111101111011	1067007867	0.93903
14)	[9,17]	5	30	111110111100100100100110000000	1056065920	0.84705
15)	[15,5]	27	30	111111111100100100101100100000	1072843552	0.99167
16)	[15,5]	27	30	111111111110100100110100000001	1073368321	0.99653
17)	[15,8]	8	30	111110111110000100000111001001	1056457161	0.85020
18)	[15,8]	8	30	111111111100000100101100100001	1072712481	0.99045
19)	[6,1]	9	30	111111101110110011110110010000	1069235600	0.95882
20)	[6,1]	9	30	111111100110010011101100110001	1067006769	0.93902
21)	[9,14]	26	30	111111111110010011111101111001	1073299321	0.99589
22)	[9,14]	26	30	111011101110010011101101111011	1001995131	0.50079
23)	[14,9]	25	30	111111101110010011011101111001	1069102969	0.95763
24)	[14,9]	25	30	111111101110010011111101111011	1069105019	0.95765
25)	[5,1]	9	30	111111111110110011110110000000	1073429888	0.99710
26)	[5,1]	9	30	111111100110100100110110000001	1067076993	0.93963
27)	[15,17]	5	30	111110111100100100100110000000	1056065920	0.84705
28)	[15,17]	5	30	110111111100000100101101100001	938494817	0.26021
29)	[6,15]	15	30	111111101110010100101100100101	1069108005	0.95767
30)	[6,15]	15	30	111111111100000111111100100001	1072725793	0.99058

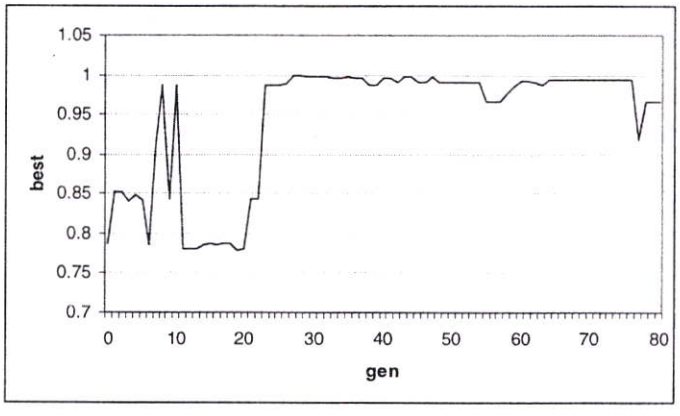
Note: maxF=0.99710, minF=0.05324, avgF=0.87290, sumF=26.18689, nmuta=31, ncross=13

c) วิวัฒนาการรุ่นที่ 16 และ 17

รูปที่ 2.8 (ต่อ)

ผลการทดสอบการทำงานของ SGA ในการหาคำตอบ เมื่อกำหนดให้ n = 10 และกำหนดค่า SGA พารามิเตอร์คือ l = 30, popsize = 30, maxgen = 30, P_c = 0.8 และ P_m = 0.033 จะเห็นว่าในรุ่นเริ่มต้น (gen = 0) ซึ่งประชากรโครโมโซมที่เกิดจากการสุ่มนั้นคำตอบที่ดีที่สุดคือ 0.33593 ดังรูปที่ 2.8(a) และผลการหาคำตอบของ SGA ในรุ่นที่ 1 ดังแสดงในรูปที่ 2.8(b) ซึ่งแสดงรายละเอียดการจับคู่โครโมโซมพ่อ-แม่ในคู่ลำดับของ parents และตำแหน่งในการครอสโอเวอร์เพื่อแลกเปลี่ยนค่าบิตตั้งแต่ตำแหน่ง pos1 ที่สุ่มได้ถึง pos2 = 30 จะเห็นว่าการครอสโอเวอร์ทำให้โครโมโซมมีความเปลี่ยนแปลงรูปแบบของบิตให้คล้ายกันมากขึ้น ในแบบของโครโมโซมที่ดีซึ่งมีค่าความเหมาะสมสูง และแสดงผลจำนวนการครอสโอเวอร์และจำนวนบิตที่เกิดมิวเตชันทั้งหมดคือ ncross และ nmuta ตามลำดับ รวมทั้งสรุปผลค่าความเหมาะสมสูงสุด ต่ำสุด และค่าเฉลี่ย คือ max, min และ avg ตามลำดับ ซึ่งในรุ่นที่ 1 นี้ SGA สามารถหาคำตอบที่ดีที่สุดคือ max = 0.88620 รูปที่ n(c) แสดงผลวิวัฒนาการของ SGA ในรุ่นที่ 16-17 จะเห็นได้ว่า รูปแบบโครโมโซมที่ดีเกิดมากยิ่งขึ้น และ SGA สามารถหาคำตอบได้ดีขึ้นเป็น 0.99710

2.8 การประยุกต์จีเนติก อัลกอริทึมแบบง่าย



รูปที่ 2.9 แสดงกราฟค่าความเหมาะสมสูงสุดของ SGA ในการหาค่าสูงสุดของฟังก์ชัน $y = x^n$ ที่ n = 10

เมื่อวิเคราะห์การทำงานของ SGA ซึ่งเป็น GA ในยุคแรกๆนั้น จะเห็นว่า วิธีการของ GA เป็นการหาคำตอบแบบสุ่ม ซึ่งเป็นวิธีการที่ไม่มีการบันทึกหรือจดจำคำตอบที่ดีที่สุดของรุ่นก่อนไว้ จึงทำให้การหาคำตอบของ SGA ได้คำตอบที่ดีมากขึ้นหรือน้อยลงได้ ดังรูปที่ 2.9 ซึ่งแสดงกราฟคำตอบของค่าความเหมาะสมสูงสุดแต่ละรุ่นภายใน 30 รุ่นในการหาค่าสูงสุดของฟังก์ชัน $y = x^n$ ที่ n = 10 จะเห็นได้ว่าคำตอบค่าความเหมาะสมสูงสุดของ SGA มีค่าสูงขึ้นและลดลงด้วยวิธีการทำงานที่อาศัยการสุ่ม ดังนั้นหากสามารถพัฒนาวิธีการค้นหาคำตอบของ GA ให้ดีขึ้น

ก็จะเป็นการปรับปรุงสมรรถนะของ GA ให้ดียิ่งขึ้นด้วย สำหรับหาค่าสูงสุดของฟังก์ชัน $y = x^n$ เราปรับปรุงการค้นหาคำตอบของ GA โดย

1. **รีโพรดักชันแบบรักษาค่าความเหมาะสมที่ดี** เนื่องจากในการค้นหาคำตอบของ SGA นั้นมีโอกาสที่จะสูญเสียโครโมโซมในรุ่นเก่า ที่มีค่าความเหมาะสมที่ดีไปได้ ซึ่งจะทำให้คำตอบในรุ่นถัดไปนั้นดีมากขึ้นหรือน้อยลง ดังนั้นหากปรับปรุง SGA ให้ควบคุมการค้นหาคำตอบ โดยรักษาโครโมโซมที่ดีไว้แล้ว จะช่วยให้วิวัฒนาการคำตอบในรุ่นถัดไปดีขึ้นเรื่อยๆ โดยมีวิธีการดังนี้

- กำหนดจำนวนโครโมโซมที่ดีที่สุด (#best) ของรุ่นเก่าที่ต้องการรักษา เป็น 1, 2, 4, ...
- ถ้าจำนวนโครโมโซมที่กำหนดเป็น 1 ให้สร้างชุดโครโมโซมรุ่นใหม่ทั้งหมด แล้วจึงคัดลอก (copy) โครโมโซมที่ดีที่สุดของรุ่นเก่า มาแทนที่โครโมโซมรุ่นใหม่ที่มีค่าความเหมาะสมน้อยที่สุด
- ถ้าจำนวนโครโมโซมที่กำหนดเป็น 2, 4, ... ให้คัดลอกโครโมโซมที่ดีที่สุดในรุ่นเก่า ตามจำนวนที่กำหนดมาเป็นโครโมโซมรุ่นใหม่ แล้วจึงสร้างโครโมโซมรุ่นใหม่ส่วนที่เหลือต่อไป

ดังรูปที่ 2.10(a) แสดงวิธีการรีโพรดักชันแบบรักษาค่าความเหมาะสมที่ดี 1 โครโมโซม และรูปที่ 2.10(b) แสดงวิธีการรีโพรดักชันแบบรักษาค่าความเหมาะสมที่ดี 2 โครโมโซม โดยใช้โครโมโซมในรูปที่ 2.10

	ลำดับ	โครโมโซมรุ่นเก่า	ค่าความเหมาะสม	จำนวนที่สุ่มได้
โครโมโซมรุ่นเก่าที่จะใช้ในการสร้างโครโมโซมรุ่นใหม่	1	0 1 1 1 0	196	1
	2	1 1 0 0 1	625	2
	3	0 1 0 0 0	64	0
	4	1 0 0 1 1	361	1

รูปที่ 2.10 รีโพรดักชันแบบรักษาค่าความเหมาะสมที่ดี

ลำดับ		กระบวนการทางพันธุศาสตร์						โครโมโซมรุ่นใหม่	ค่าความเหมาะสม
		ครอสโอเวอร์			มิวเตชั่น		ค่าความเหมาะสม		
ปัจจุบัน	อ้างอิง		pos		pos				
1	1	01110	2	01001		01001	81	← ค่าความเหมาะสมต่ำ	
2	2	11001		11110	2, 4	10100	400	10100	400
3	4	10011	1	11001	2	10001	289	10001	289
4	2	11001		10011		10011	361	10011	361
		คัดลอกโครมโซมรุ่นเก่าลำดับที่ 2 แทนที่โครโมโซมรุ่นใหม่ที่มีค่าความเหมาะสมต่ำที่สุด →						11001	625

a) รักษาค่าความเหมาะสมที่ดี 1 โครโมโซม

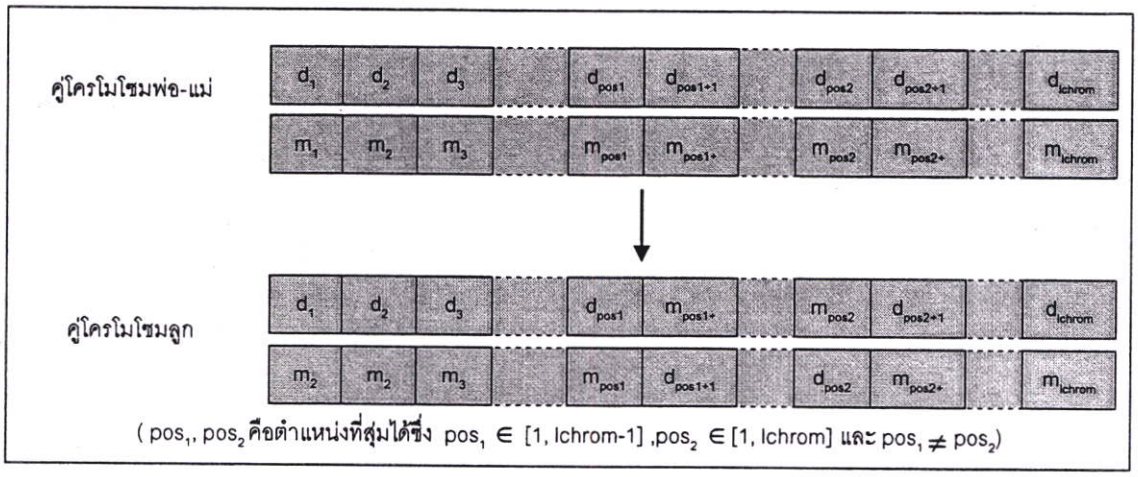
ลำดับ		กระบวนการทางพันธุศาสตร์						โครโมโซมรุ่นใหม่	ค่าความเหมาะสม
		ครอสโอเวอร์			มิวเตชั่น		ค่าความเหมาะสม		
ปัจจุบัน	อ้างอิง		pos		pos				
1	2							11001	625
2	4				คัดลอกโครมโซมรุ่นเก่าลำดับที่ 2 และ 4 →			10011	361
3	1	01110	2	01001		01001		01001	81
4	2	11001		11110	2, 4	10100		10100	400

b) รักษาค่าความเหมาะสมที่ดี 2 โครโมโซม

รูปที่ 2.10 (ต่อ)

2. **ครอสโอเวอร์แบบ 2 จุด** การแลกเปลี่ยนส่วนของโครโมโซมพ่อ-แม่นั้น บางครั้งหากแลกเปลี่ยนค่าบิตเพียงบางช่วงของโครโมโซมแล้วจะสร้างโครโมโซมที่ดีกว่า เช่น การหาค่าสูงสุดของฟังก์ชัน $y = x^2$ ของคู่โครโมโซมพ่อ-แม่ 01110 และ 11001 ซึ่งมีค่าความเหมาะสมเป็น 196 และ 625 หากแลกเปลี่ยนค่าบิตตำแหน่งที่ 3 และ 4 เท่านั้น จะทำให้เกิดโครโมโซมลูกคือ 01*00* และ 11*11* มีค่าความเหมาะสมเป็น 64 และ 961 ซึ่งโครโมโซม 11111 เป็นโครโมโซมที่ให้คำตอบที่ดีที่สุดที่ต้องการ ดังนั้นการประยุกต์ SGA โดยพัฒนาตัวดำเนินการครอสโอเวอร์ให้เป็นแบบ 2 จุด (Two-point Crossover) จะทำให้ SGA ค้นหาคำตอบที่ดีขึ้นได้ ดังรูปที่ 2.14 ครอสโอเวอร์แบบ 2 จุด มีวิธีการดังนี้

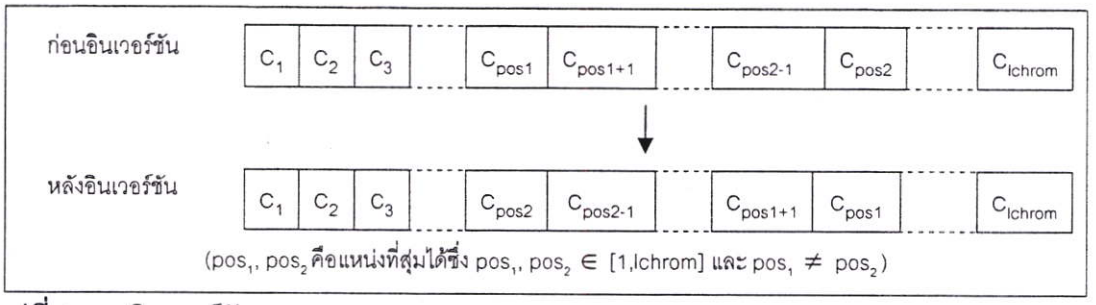
- สุ่มตำแหน่ง pos_1, pos_2 ซึ่งเป็นตำแหน่งเริ่มต้นและตำแหน่งสุดท้ายที่จะครอสโอเวอร์ตามลำดับ โดยที่ pos_1 มีค่าอยู่ในช่วง [1, lchrom-1] และ pos_2 มีค่าอยู่ในช่วง [1, lchrom] โดยที่ pos_1 มีค่าน้อยกว่า pos_2



- สุ่มตำแหน่ง pos_1 , pos_2 คือตำแหน่งเริ่มต้นและตำแหน่งสุดท้ายที่จะครอสโอเวอร์ตามลำดับ ซึ่ง pos_1 มีค่าอยู่ในช่วง [1, lchrom-1] และ pos_2 มีค่าอยู่ในช่วง [1, lchrom] โดยที่ pos_1 มีค่าน้อยกว่า pos_2
- แลกเปลี่ยนค่าในแต่ละบิตของคู่โครโมโซมพ่อ-แม่ตั้งแต่ตำแหน่งที่ pos_1+1 ถึง pos_2

3. **ไบนารีมิวเตชันแบบกำหนดค่าบิต** เนื่องจากการหาคำตอบของ SGA กระบวนการไบนารีมิวเตชันอาจทำให้โครโมโซมที่เปลี่ยนแปลงไปให้คำตอบที่ดีลดลง และทำให้สูญเสียโครโมโซมที่ดีไป เช่น โครโมโซม 11110 มีค่าความเหมาะสมเป็น 900 หากสุ่มได้บิตตำแหน่งที่ 1 เกิดมิวเตชันแล้ว โครโมโซมที่เกิดขึ้นจากการมิวเตชันคือ 01110 ทำให้มีค่าความเหมาะสมลดลงเป็น 196 แต่ในบางครั้งข้อดีหรือจุดเด่นของปัญหาจะสามารถนำมาปรับให้เข้ากับการค้นหาคำตอบที่ดีขึ้นได้ สำหรับในการหาค่าสูงสุดของฟังก์ชัน $y = x^n$ นี้ค่าบิตของโครโมโซมที่เป็น 1 จะทำให้ค่าความเหมาะสมสูงขึ้นเสมอ ดังนั้นหากปรับปรุงไบนารีมิวเตชันให้เป็นแบบกำหนดค่าแน่นอนให้กับบิตที่เกิดมิวเตชัน โดยกำหนดให้บิตที่เกิดมิวเตชันมีค่าบิตเป็น 1 เสมอจะช่วยปรับแนวทางการค้นหาคำตอบของ SGA ดีขึ้น เช่น หากประยุกต์ไบนารีมิวเตชันที่กำหนดค่าบิตให้เป็น 1 เสมอกับโครโมโซม 11110 ในตำแหน่งที่ 1 แล้วโครโมโซมที่เกิดขึ้นจะเหมือนเดิม และยังเป็นการรักษาโครโมโซมที่มีค่าความเหมาะสมที่ดีไว้ด้วย

4. **อินเวอร์ชัน** (Inversion) เป็นตัวดำเนินการที่ประยุกต์เพิ่มเติมใน SGA โดยจำลองแบบลักษณะของการอินเวอร์ชันในทางพันธุศาสตร์ที่เป็นลักษณะของการกลับหัวกลับหางส่วนของยีนส์ภายในโครโมโซม ที่อาจช่วยให้เกิดโครโมโซมที่ดีขึ้นได้ โดยการกลับส่วนค่าบิตภายในช่วงตำแหน่งของโครโมโซมที่สุ่มได้ตามอัตราค่าความน่าจะเป็นของการอินเวอร์ชันแต่ละโครโมโซม (Probability of Inversion: Pi) ที่กำหนด ดังรูปที่ 2.11 มีขั้นตอนดังนี้



รูปที่ 2.11 อินเวอร์ชัน

- สุ่มเลือกตำแหน่ง pos_1, pos_2 ให้เป็นตำแหน่งเริ่มต้นและตำแหน่งสุดท้ายที่จะอินเวอร์ชันตามลำดับ ซึ่ง pos_1, pos_2 มีค่าอยู่ในช่วง [1,lchrom] โดยที่ pos_1 มีค่าน้อยกว่า pos_2
- กลับค่าบิตในช่วงของตำแหน่ง pos_1 ถึง pos_2 ของโครโมโซม โดยสลับค่าบิต pos_1 กับ pos_2, pos_1+1 กับ pos_2-1, pos_1+2 กับ pos_2-2, ...

ตัวอย่างเช่น สุ่มโครโมโซมที่จะอินเวอร์ชัน คือ 01010 มีค่าความเหมาะสมเป็น 100 โดยสุ่มตำแหน่ง pos_1 = 1 และ pos_2 = 4 แล้ว จะเห็นว่าการอินเวอร์ชันทำให้เกิดโครโมโซม *1010*0 ซึ่งมีค่าความเหมาะสมดีขึ้นเป็น 400 เป็นต้น สำหรับจำนวนการอินเวอร์ชันในแต่ละรุ่นขึ้นอยู่กับการกำหนดค่า Pi ซึ่งแตกต่างกันในแต่ละปัญหา เช่น ถ้าจำนวนประชากรแต่ละรุ่นมี popsize เท่ากับ 30 โครโมโซม และกำหนดให้ Pi = 0.1 แล้ว จำนวนการอินเวอร์ชันในแต่ละรุ่นเท่ากับ Pi * popsize = 0.1 * 30 = 3 ครั้ง

ผลการทำงานโดยสรุปของการประยุกต์ SGA ในการหาค่าสูงสุดของฟังก์ชัน $y = x^n$ ที่ n = 10 ภายใน 30 รุ่น ซึ่งกำหนดค่า SGA พารามิเตอร์ต่างๆเปรียบเทียบดังตารางที่ 2.2 แสดงดังรูปที่ 2.16(a) ถึงรูปที่ 2.16(g) โดยเปรียบเทียบการหาคำตอบที่มีการกำหนดให้มีการมิวเตชันในรูปที่ 2.16(b) นั้นดีกว่าการหาคำตอบเพียงครอสโอเวอร์เท่านั้นในรูปที่ 2.16(a) และจะเห็นได้ว่า SGA สามารถหาคำตอบได้ดีขึ้นเมื่อมีการปรับปรุงวิธีการต่าง ๆ คือ หากใช้รีโพรดักชันแบบรักษาค่าที่ดีจำนวน 4 โครโมโซม และครอสโอเวอร์แบบ 2 ตำแหน่งในรูปที่ 2.16(f) แล้ว ผลการทำงานของ SGA สามารถหาคำตอบที่ดีที่สุดเท่ากับ 1.0 ได้ ซึ่งดีกว่าในรูปที่ 2.16(d) เมื่อรีโพรดักชันแบบรักษาค่าที่ดีจำนวน 1 โครโมโซม ที่สามารถหาคำตอบได้เป็น 0.99992 และดีกว่าการรีโพรดักชันแบบทั่วไป ในรูปที่ 2.16(a), 2.16(b), 2.16(c) และ 2.16(d) ที่ยังไม่สามารถหาคำตอบที่ดีที่สุดได้ภายในการดำเนินการ 30 รุ่น สำหรับในรูปที่ 2.16-f นั้นเมื่อเพิ่มตัวดำเนินการอินเวอร์ชันแล้ว ผลการดำเนินงานนั้นสามารถหาคำตอบที่ดีที่สุดเท่ากับ 1.0 ได้เร็วขึ้นในรุ่นที่ 26 และเมื่อเพิ่มเทคนิค

การกำหนดค่าบิตของการมิวเตชันให้เป็น 1 เสมอแล้ว ก็ยิ่งทำให้ SGA ทำงานได้ดียิ่งขึ้น โดยสามารถหาคำตอบที่ดีที่สุดได้เมื่อดำเนินงานผ่านไปเพียง 12 รุ่นเท่านั้น ดังในรูปที่ 2.16(g)

ตารางที่ 2.2 การกำหนด SGA พารามิเตอร์เปรียบเทียบผลการทำงานในการหาค่าสูงสุดของฟังก์ชัน $y = x^n$ ที่ n = 10

รูปที่	P_c	P_m	P_i	จำนวนการรักษาโครโมโซมรุ่นเก่า (# best)	จำนวนตำแหน่งในการครอสโอเวอร์ (pt. Cross)	การกำหนดค่าบิตมิวเตชัน (fix value)
2.16-a	0.8000	0.0000	0.0000	0	1	0->1 และ 1->0
2.16-b	0.8000	0.0333	0.0000	0	1	0->1 และ 1->0
2.16-c	0.8000	0.0333	0.0000	0	2	0->1 และ 1->0
2.16-d	0.8000	0.0333	0.0000	1	2	0->1 และ 1->0
2.16-e	0.8000	0.0333	0.0000	4	2	0->1 และ 1->0
2.16-f	0.8000	0.0333	0.3000	4	2	0->1 และ 1->0
2.16-g	0.8000	0.0333	0.3000	4	2	0 , 1 -> 1

popsize = 30 Pc = 0.8 #best = 0
chro_len = 30 Pm = 0.0 pt_crs = 1
max_gen = 30 Pi = 0.0 fix_val = 2

generation	max	average	min
0	0.70749	0.09471	0
1	0.70749	0.42783	0.11111
2	0.71468	0.55239	0.23296
3	0.83815	0.64638	0.36243
4	0.71468	0.62662	0.36234
5	0.71472	0.66739	0.36243
6	0.71468	0.70856	0.7066
7	0.7149	0.70927	0.70731
8	0.7149	0.70904	0.70731
9	0.71468	0.70849	0.70731
10	0.72193	0.70821	0.70037
11	0.72193	0.70826	0.70727
12	0.71468	0.70773	0.70727
13	0.70776	0.70753	0.70731
14	0.70794	0.70754	0.70731
15	0.70794	0.70757	0.70731
16	0.70794	0.70757	0.70731
17	0.70794	0.70758	0.70731
18	0.70794	0.70756	0.70731
19	0.70776	0.70753	0.70731
20	0.70776	0.70756	0.70731
21	0.70776	0.7076	0.70731
22	0.70776	0.70761	0.70731
23	0.70776	0.70761	0.70749
24	0.70776	0.7076	0.70749
25	0.70776	0.70759	0.70749
26	0.70776	0.70754	0.70749
27	0.70776	0.70752	0.70749
28	0.70776	0.70752	0.70749
29	0.70776	0.70752	0.70749
30	0.70776	0.70751	0.70749

(a)

popsize = 30 Pc = 0.8 #best = 0
chro_len = 30 Pm = 0.033 pt_crs = 1
max_gen = 30 Pi = 0.0 fix_val = 2

generation	max	average	min
0	0.43298	0.04906	0
1	0.43308	0.18852	0.00008
2	0.81967	0.29894	0.00004
3	0.87917	0.39725	0.00017
4	0.87917	0.53188	0.00017
5	0.97869	0.63206	0.00065
6	0.97867	0.67935	0.00008
7	0.97867	0.72485	0.00007
8	0.97867	0.61631	0.00065
9	0.97869	0.62996	0.00033
10	0.99679	0.67665	0.00065
11	0.99679	0.76617	0.0009
12	0.9974	0.78496	0.0009
13	0.99862	0.7671	0.00005
14	0.99862	0.79154	0.00016
15	0.99862	0.84542	0.00097
16	0.99862	0.8435	0.00077
17	0.99862	0.83287	0.00097
18	0.95925	0.79465	0.00074
19	0.97823	0.88375	0.0009
20	0.99648	0.87305	0.0009
21	0.96113	0.73022	0.00077
22	0.96113	0.73475	0.0009
23	0.97959	0.78115	0.00077
24	0.9582	0.79107	0.00077
25	0.95866	0.69329	0.00056
26	0.95866	0.6988	0.00039
27	0.95893	0.71553	0.00077
28	0.95853	0.74432	0.00074
29	0.99557	0.71986	0.00073
30	0.99557	0.67322	0.00013

(b)

popsize = 30 Pc = 0.8 #best = 0
chro_len = 30 Pm = 0.033 pt_crs = 2
max_gen = 30 Pi = 0.0 fix_val = 2

generation	max	average	min
0	0.97919	0.12466	0
1	0.97925	0.49422	0.00094
2	0.97921	0.56689	0.00093
3	0.98007	0.78508	0.00041
4	0.97928	0.76774	0.00092
5	0.98056	0.77113	0.0316
6	0.9804	0.71125	0.00049
7	0.97536	0.66249	0.0004
8	0.97904	0.71912	0.03645
9	0.99478	0.74335	0.0544
10	0.98791	0.76997	0.05061
11	0.99822	0.81854	0.04908
12	0.99823	0.87496	0.0543
13	0.99822	0.82192	0.05367
14	0.99761	0.81145	0.05377
15	0.99828	0.74479	0.04897
16	0.98858	0.60484	0.00055
17	0.98858	0.58893	0.00017
18	0.99827	0.73856	0.00013
19	0.98888	0.6916	0.00058
20	0.98858	0.69158	0.00018
21	0.98861	0.65461	0.00017
22	0.99827	0.68135	0.00069
23	0.99859	0.865	0.04904
24	0.99827	0.82567	0.00091
25	0.99961	0.83325	0.00091
26	0.99012	0.92368	0.50535
27	0.99342	0.81871	0.00016
28	0.99342	0.82807	0.00069
29	0.99342	0.89397	0.05517
30	0.99339	0.87325	0.05374

(c)

popsize = 30 Pc = 0.8 #best = 1
chro_len = 30 Pm = 0.033 pt_crs = 2
max_gen = 30 Pi = 0.0 fix_val = 2

generation	max	average	min
0	0.63974	0.07857	0
1	0.64312	0.30245	0.00039
2	0.94399	0.48589	0.02169
3	0.9555	0.61641	0.00076
4	0.9555	0.6941	0.15766
5	0.95948	0.71667	0.32288
6	0.95948	0.67251	0.03402
7	0.95948	0.67203	0.15584
8	0.95976	0.73793	0.15584
9	0.96066	0.76729	0.03473
10	0.99809	0.63443	0
11	0.99809	0.69008	0.00065
12	0.99893	0.75211	0.05262
13	0.99893	0.74185	0.00003
14	0.99893	0.72603	0.00005
15	0.99893	0.73361	0.0009
16	0.99908	0.6821	0.04692
17	0.99948	0.70679	0.0009
18	0.99948	0.79324	0.17931
19	0.99992	0.82336	0.36962
20	0.99992	0.72442	0.00083
21	0.99992	0.82453	0.48038
22	0.99992	0.90255	0.52156
23	0.99992	0.84737	0.00097
24	0.99992	0.6834	0.00097
25	0.99992	0.77486	0.03421
26	0.99992	0.62282	0.02279
27	0.99992	0.83256	0.00093
28	0.99992	0.90506	0.21408
29	0.99992	0.93018	0.25433
30	0.99992	0.73846	0.00051

(d)

รูปที่ 2.12 ผลการทำงานโดยสรุปของ GA ในการหาค่าสูงสุดของฟังก์ชัน $y = x^n$ ที่ n = 10

popsize = 30 Pc = 0.8 #best = 4
chro_len = 30 Pm = 0.033 pt_crs = 2
max_gen = 30 Pi = 0.0 fix_val = 2

generation	max	average	min
0	0.81716	0.1011	0
1	0.81716	0.41909	0.00029
2	0.9572	0.60992	0.00053
3	0.9572	0.68155	0.00084
4	0.9572	0.68188	0.00031
5	0.9572	0.66324	0.00084
6	0.95748	0.69604	0.00031
7	0.98598	0.7519	0.00039
8	0.98599	0.77122	0.05314
9	0.98599	0.77277	0.05311
10	0.99933	0.77363	0.00018
11	0.99933	0.77138	0.05617
12	0.99994	0.79869	0.0478
13	0.99998	0.76835	0.00089
14	0.99998	0.79505	0.00096
15	0.99998	0.82649	0.00098
16	0.99998	0.92317	0.05489
17	0.99998	0.91168	0.05242
18	0.99998	0.82499	0.00094
19	0.99998	0.83259	0.00084
20	0.99998	0.90017	0.00088
21	0.99998	0.81041	0.00064
22	0.99998	0.8187	0.00085
23	*0.99999*	*0.92446*	*0.51901*
24	0.99999	0.93983	0.52102
25	0.99999	0.92311	0.00026
26	0.99999	0.89821	0.0559
27	0.99999	0.9293	0.05513
28	0.99999	0.83865	0.00097
29	0.99999	0.8573	0.0008
30	0.99999	0.73417	0.00071

(e)

popsize = 30 Pc = 0.8 #best = 4
chro_len = 30 Pm = 0.033 pt_crs = 2
max_gen = 30 Pi = 0.2 fix_val = 2

generation	max	average	min
0	0.74758	0.09576	0
1	0.90826	0.52266	0.0014
2	0.97845	0.62949	0.00001
3	0.97845	0.66078	0.0004
4	0.97845	0.67828	0.00001
5	0.97845	0.70916	0.00001
6	0.97846	0.71433	0
7	0.97846	0.73986	0
8	0.98017	0.84169	0.00093
9	0.99839	0.81042	0.05478
10	0.99839	0.89854	0.05481
11	0.99839	0.87857	0.05548
12	0.99839	0.78099	0.00096
13	0.99839	0.81873	0.00096
14	0.99967	0.89835	0.00094
15	0.99967	0.93273	0.00093
16	0.99972	0.7861	0.00093
17	0.99972	0.91525	0.25521
18	0.99972	0.90106	0.23679
19	0.99972	0.90851	0.00094
20	0.99972	0.90279	0.00092
21	0.99982	0.95773	0.7244
22	0.99982	0.90139	0.04934
23	0.99982	0.87324	0.00097
24	0.99982	0.83537	0.00097
25	0.99984	0.8874	0.00077
26	*1*	*0.89518*	*0.00098*
27	1	0.83699	0.00024
28	1	0.93298	0.49712
29	1	0.94088	0.52436
30	1	0.89647	0.00098

(f)

popsize = 30 Pc = 0.8 #best = 4
chro_len = 30 Pm = 0.033 pt_crs = 2
max_gen = 30 Pi = 0.2 fix_val = 1

generation	max	average	min
0	0.98489	0.14726	0
1	0.99117	0.80128	0.21662
2	0.99117	0.85199	0.45021
3	0.9996	0.85631	0.00716
4	0.9996	0.91723	0.59326
5	0.9996	0.9075	0.71696
6	0.9996	0.88948	0.3635
7	0.99969	0.90864	0.37115
8	0.99969	0.90595	0.81984
9	0.99969	0.95015	0.82022
10	0.99969	0.95688	0.3139
11	0.99969	0.9885	0.96604
12	*1*	*0.99228*	*0.96604*
13	1	0.97308	0.52446
14	1	0.97066	0.48745
15	1	0.93128	0
16	1	0.96652	0.34267
17	1	0.99573	0.92457
18	1	0.96488	0.00018
19	1	0.99975	0.99756
20	1	0.9999	0.99969
21	1	0.99708	0.94293
22	1	0.99473	0.9383
23	1	0.99295	0.94293
24	1	0.9564	0.05486
25	1	0.99613	0.98064
26	1	0.99742	0.98064
27	1	0.99749	0.92457
28	1	0.99246	0.92456
29	1	1	1
30	1	1	1

(g)

รูปที่ 2.12 (ต่อ)

บทที่ 3
งานวิจัยที่เกี่ยวข้อง

3.1 ความหมายของ Multicasting

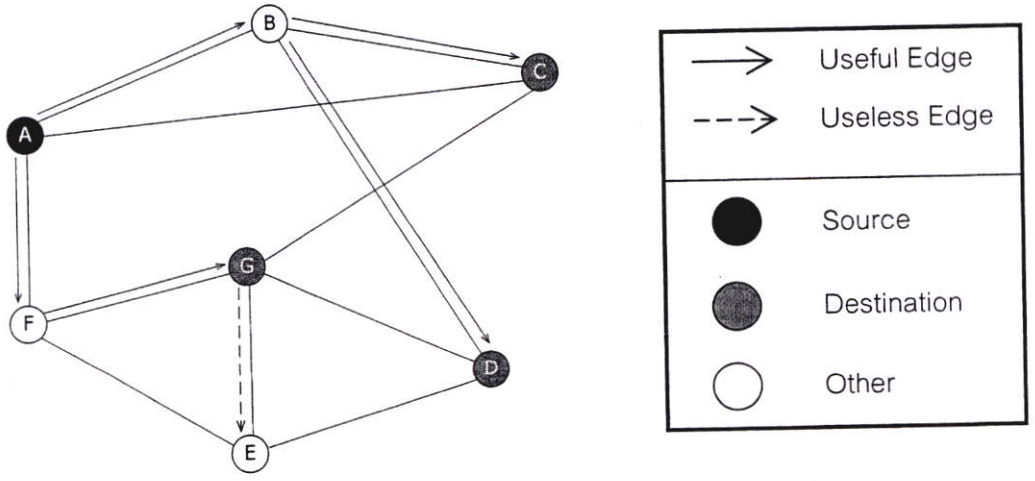
Multicast[1] คือ การส่งข้อมูลจากต้นทางหนึ่งแห่งกระจายไปยังหลาย ๆ ปลายทาง ปัญหาในลักษณะนี้ได้มีการศึกษาวิจัยมาตั้งแต่ปี พ.ศ. 2513 แล้ว แต่เป็นการศึกษาวิจัยในมุมมองที่แตกต่างไปจากปัจจุบัน เหตุผลบางอย่างที่ทำให้ดูเหมือนว่าการทำ Multicasting จะช่วยให้เราทำงานได้ง่าย และสะดวกขึ้น คือ

- การกระจายปัญหาไปยังเครื่อง sever
- การส่งผลลัพธ์ของการคำนวณไปยังอุปการณ์ต่าง ๆ ระบบ parallel processor
- กระจายข้อมูลไปยังเครื่องคอมพิวเตอร์ที่อยู่ในระบบ distributed computation

ในอดีตการติดต่อสื่อสารจะมีอยู่ 2 วิธี คือ Unicasting และ Broadcasting ซึ่งทั้ง 2 วิธีนี้ได้ถูกนำมาปรับปรุง เพื่อให้มีประสิทธิภาพมากขึ้น และเป็นที่มาของ Multicasting

- Unicast : จะส่งข้อมูลจากหนึ่งต้นทางไปยังหนึ่งปลายทาง
- Broadcast : จะส่งข้อมูลจากหนึ่งต้นทางไปยังทุก ๆ ปลายทางที่มีอยู่

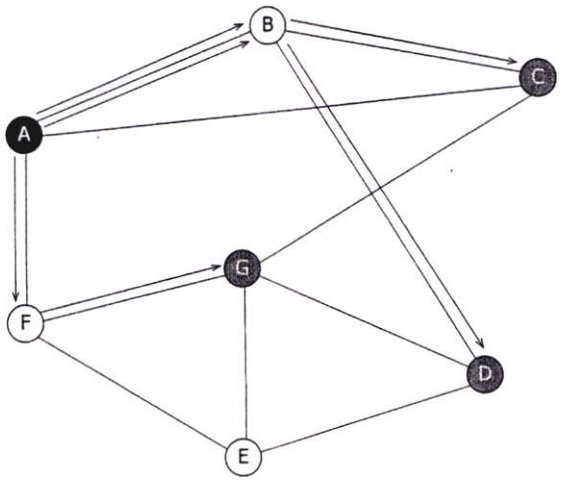
การสร้างเส้นทางแบบ Multicast นั้นสามารถสร้างได้โดยใช้ทั้ง Unicast และ Broadcast แต่ผลลัพธ์ที่ได้ส่วนใหญ่จะยังใช้ทรัพยากรสิ้นเปลืองอยู่ ซึ่งยังยอมรับไม่ได้ ซึ่งจะแสดงให้เห็น ดังรูปที่ 3.1



รูปที่ 3.1 การนำ Broadcast มาใช้เพื่อสร้าง Multicast

เมื่อต้นทาง A ต้องการส่งข้อมูลบางอย่างไปยังปลายทาง C, D, และ G โดยใช้การส่งแบบ Broadcast จะเห็นว่า ข้อมูลจาก A จะถูกส่งไปยังทุก ๆ ปลายทาง โดยที่ E ไม่จำเป็นที่จะต้องรับข้อมูลดังกล่าวก็ได้ ส่วน B แล F ทำหน้าที่ส่งต่อข้อมูลเพื่อให้ถึงปลายทาง

มีอยู่ 2 เหตุผลที่ไม่เหมาะสมในการส่งข้อมูลจากต้นทาง กระจายไปยังหลาย ๆ ปลายทาง โดยใช้ Unicast คือ ประการแรก ข้อมูลที่ส่งไปยังปลายทางเป็นข้อมูลที่ต่อเนื่อง ซึ่งอาจส่งผลให้เกิดการหน่วงเวลาได้ ถึงแม้ว่าทรัพยากรในระบบจะสามารถใช้งานได้พร้อมๆ กันก็ตาม แต่การทำงานแบบ Unicast นั้นจะต้องส่งข้อมูลไปยังปลายทางให้เสร็จทีละปลายทาง ดังแสดงในรูปที่ 3.2



รูปที่ 3.2 การนำ unicasts มาใช้เพื่อสร้าง Multicast

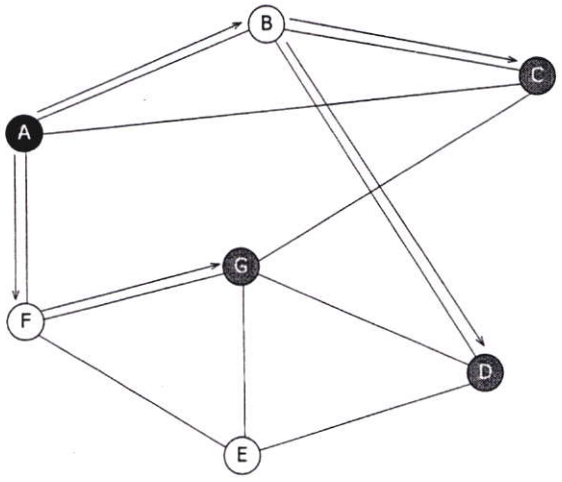
จากรูปจะต้องส่งข้อมูลจาก A ไป G ก่อน จึงจะสามารถส่งข้อมูลจาก A ไป C ได้ ในขณะที่เส้นทางจาก A ไป G และ A ไป C นั้น เป็นเส้นทางที่ไม่เกี่ยวข้องกันเลย ปัจจุบันสามารถนำเทคโนโลยีที่เรียกว่า Switch เข้ามาช่วยแก้ปัญหาดังกล่าวได้ โดย Switch จะทำสำเนาข้อมูลก่อนแล้วจึงส่งข้อมูลออกไปพร้อม ๆ กัน ในช่องทางที่ต่างกัน ซึ่งสามารถสนับสนุนการทำงานแบบ multicast ได้ดี

เหตุผลที่สองที่ไม่เหมาะสมที่จะใช้ Unicast ในการทำ Multicast และเป็นเหตุผลที่สำคัญกว่าเหตุผลแรกก็คือ มีความเป็นไปได้ที่การส่งข้อมูลกระจายไปยังหลาย ๆ ปลายทางนั้น จะสามารถใช้บางส่วนของเส้นทางที่เหมือนกัน (Share link) ตัวอย่าง เช่น เส้นทางจาก A ไป C และ เส้นทางจาก A ไป D สามารถใช้เส้นทาง AB ร่วมกันได้ ซึ่งหากใช้ Unicast ส่งข้อมูลจะทำให้เส้นทาง AB ถูกส่งข้อมูลเดิมซ้ำถึงสองครั้ง

3.2 ปัญหาการหาเส้นทางแบบ Multicast

คำถามหลักในการหาเส้นทางแบบ Multicast ก็คือจะใช้โครงสร้างแบบใดในการนำเสนอรูปแบบของเส้นทาง คำถามถัดมาก็คือ จะสร้างโครงสร้างดังกล่าวขึ้นมาได้อย่างไร คำตอบก็คือ มีโครงสร้างอยู่ชนิดหนึ่งที่เหมาะสมและมีประสิทธิภาพที่จะใช้ในการสร้างโครงสร้างแบบ Multicast ก็คือ โครงสร้างต้นไม้ (tree structure) ในโครงสร้างต้นไม้ ต้นทางจะเป็น root และกระจายเส้นทางไปยังปลายทางในลักษณะของกิ่งก้านของต้นไม้ ซึ่งโครงสร้างแบบต้นไม้จะไม่ก่อให้เกิดปัญหาที่กล่าวมาข้างต้น

รูปที่ 3.3 จะแสดงให้เห็นถึงโครงสร้างต้นไม้ที่สามารถใช้ส่งข้อมูลไปยัง C, D และ G (สมมุติว่า switch ที่ใช้มีความสามารถในการ copy ข้อมูล) ข้อมูลที่ส่งไปยัง B จะถูกกระทำเพียงครั้งเดียว และข้อมูลจาก B ก็จะถูกกระจายออกไปยัง C และ D พร้อมๆกัน



รูปที่ 3.3 โครงสร้างแบบต้นไม้เป็นโครงสร้างที่เหมาะสมสำหรับการทำ multicast

3.3 รูปแบบของระบบเครือข่าย และนิยามของปัญหา

เราจะใช้กราฟ (graph) ในการนำเสนอข้อมูลของระบบเครือข่าย โดยให้การเชื่อมต่อระหว่างสองโหนดใด ๆ มีเพียงเส้นทางเดียว เส้นทางที่ใช้เชื่อมต่อโหนด เรียกว่าลิงค์ (link) จะสามารถส่งข้อมูลแบบ 2 ทิศทางได้ (ไป / กลับ) แต่ละลิงค์จะมีค่า cost และ delay กำกับอยู่ ในวิทยานิพนธ์ฉบับนี้เราจะกำหนดให้ค่า cost ของ multicast tree คือ ผลรวมของค่า cost ของแต่ละลิงค์ที่อยู่ใน multicast tree

กราฟ $G=(V,E)$ จะใช้ในการนำเสนอข้อมูลของระบบเครือข่าย โดยที่ V จะใช้อธิบายเซ็ตของโหนดภายในเครือข่าย และ E จะใช้อธิบายเซ็ตของลิงค์ โดยที่ในแต่ละลิงค์ $e(e \in E)$ จะมีค่าจำนวนจริง 2 ค่าที่เกี่ยวข้องด้วย คือ $delay\,D(e)$ และ $cost\,C(e)$ โดยที่ $delay\,D(e)$ จะเป็นเวลา

ที่ข้อมูลใช้ในการเดินทางภายในลิงค์นั้นๆ และ $cost\,C(e)$ คือค่าที่คำนวณมาจากการใช้งานทรัพยากรที่เกี่ยวข้องกับลิงค์นั้นๆ ในแต่ละลิงค์อาจจะมีค่า cost และ delay ไม่เหมือนกัน (ไป/กลับ) หมายความว่าลิงค์ $e=(i,j)$ และลิงค์ $e'=(j,i)$ อาจไม่ใช้ค่าเดียวกัน

การแก้ปัญหาของการหาเส้นทางแบบหลายปลายทางนั้น เราจะกำหนดให้ $s(s\in V)$ คือ โหนดต้นทาง $S(S\subseteq V-s)$ คือ เซ็ตของโหนดปลายทาง และ Δ คือ เวลาสูงสุดที่ยอมรับได้ในการส่งข้อมูลจากต้นทางไปปลายทาง โดยที่ tree $T(T\subseteq G)$ จะเป็น multicast tree ที่มี root อยู่ที่ s แล้วกระจายเส้นทางออกไปยังทุกๆโหนดปลายทาง v ที่อยู่ในเซ็ตของโหนดปลายทาง S และเป็นไปตามสมการดังต่อไปนี้

$$\sum_{e\in P(s,v)} D(e) < \Delta \qquad (1)$$

เมื่อ $P(s,v)$ เป็นเส้นทางจาก s ไป v ที่ unique อยู่ใน T และจะต้องมีค่า tree cost เป็น $\sum_{e\in T} C(e)$ ที่มีค่า cost ต่ำที่สุด

เราจะกำหนดให้ $n=|V|$ และ $m=|S|$ โดยที่โหนดที่อยู่ใน S จะถูกเรียกว่า multicast group member และ m จะหมายถึงขนาดของ multicast group และ Δ จะเรียกว่า delay bound เมื่อ m มีค่าเป็น 1 นั่นหมายความว่าปัญหาของเราจะถูกลดขนาดลงจาก multicast เป็น unicast แต่ก็ยังคงเป็นปัญหาแบบ *NP*-complete อยู่เช่นเดิม

Tree ที่กระจายจากโหนดต้นทาง s ไปยังหลายๆโหนดปลายทาง S และเป็นไปตามสมการ (1) จะเรียกว่า constrained multicast tree วิธีการแบบ heuristic ที่ใช้แก้ปัญหาแบบ delay-constrained multicast routing นั้น จะต้องสามารถหาเส้นทางดังกล่าวได้ ถ้าภายในเครือข่ายนั้นมีเส้นทางนี้อยู่ และวิธีการแบบ heuristic ที่มีประสิทธิภาพจะต้องใช้เวลาน้อยในการหาเส้นทางแบบ multicast และจะต้องสามารถสร้าง tree ที่มีค่า cost ต่ำได้

3.4 อัลกอริทึมที่ใช้หาเส้นทางแบบ Multicast

อัลกอริทึมที่จะกล่าวถึงต่อไปนี้ จะเป็นอัลกอริทึมที่มีความสัมพันธ์กัน 3 อัลกอริทึม คือ Constrained Routing Algorithm (CRA)[2], CKMB Algorithm[3] และ Genetic Algorithm (GA)[4][5][6] โดยใน 2 อัลกอริทึมแรกจะใช้แนวคิดแบบ heuristic

3.4.1 CRA Algorithm[2]

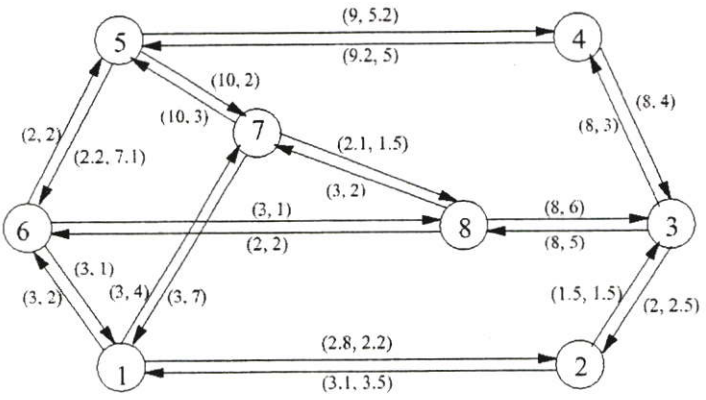
ก่อนที่จะกล่าวถึง CKMB Algorithm เราจะต้องทำความเข้าใจ CRA Algorithm ก่อน เนื่องจาก CRA เป็นอัลกอริทึมหลักที่จะใช้ใน CKMB โดยที่ CRA เป็นอัลกอริทึมจะพยายามหา

เส้นทางในการกระจายข้อมูลจากโหนดต้นทาง s ไปยังโหนดปลายทางบางโหนด (หรือทุกโหนดถ้าเป็นไปได้) ที่อยู่ใน S และเวลาที่ใช้จากโหนดต้นทางไปยังโหนดปลายทางจะต้องไม่เกินเวลาที่กำหนด โดยอัลกอริทึมของ CRA จะแสดงดังรูปที่ 3.4

```
Procedure CRA(s, S, Δ, C, D)
/* s is the source node, S is the set of destination nodes. Δ is the bounded delay.
The network is represented by its cost adjacency matrix C with C(i,j) being the cost
of the link (i,j) and by its delay adjacency matrix D with D(i,j) being the delay of
the link (i,j). Both C(i,j) and D(i,j) will be set to 1 in case the link (i,j) is not
in E. For i=j, C(i,j) and D(i,j) are set to 0. The nodes of the network G(V,E) are
numbered 1 through n. */

boolean S(1:n); real CST(1:n), DLY(1:n);

/* The set of nodes V is divided into two subsets, namely the set of already computed
nodes and the set of nodes to be computed. If S(i)=1, node I is then an already
computed node and the cost and delay of the computed path from the source s to i are
given in CST(i) and DLY(i) respectively. S(i)=0 means node i has not been computed
yet. */

integer u, i, w; real vc;
begin                                                                          (1)
/* initialize the set of already computed nodes to empty */
        for i ←1 to n do                                                       (2)
                S(i) ← 0;                                                      (3)
                if D(s,i) < Δ                                                  (4)
                        then CST(i) ←C(s,i) and DLY(i) ←D(s,i);                (5)
                else CST(i) ←1                                                 (6)
                end if                                                         (7)
        end for                                                                (8)
/*s is the first computed node */
S(s) ←1; CST(s) ← 0; DLY(s) ← 0;                                               (9)
/* compute paths */
while there are nodes in S which have not been computed yet do                 (10)
        choose u such that CST(u) =min{CST(w)} for all 1 ≤ w ≤ n and S(w)=0;    (11)
        if CST(u) is ∞ then break; end if                                      (12)
        /* u is a new computed node */
        S(u) ← 1;                                                              (13)
        /* update costs and delays */
        for all w with S(w)=0 do                                               (14)
                if DLY(u) +D(u, w) < Δ then                                    (15)
                        vc ← CST(u) +C(u, w);                                  (16)
                        if vc < CST(w) then                                    (17)
                                CST(w) ← vc;                                   (18)
                                DLY(w) ← DLY (u) +D(u, w);                     (19)
                        End if                                                 (20)
                end if                                                         (21)
        end for                                                                (22)
end while                                                                      (23)
prune those leaf nodes in the resulting tree that are not in S;                (24)
end                                                                            (25)
```

รูปที่ 3.4 CRA อัลกอริทึม

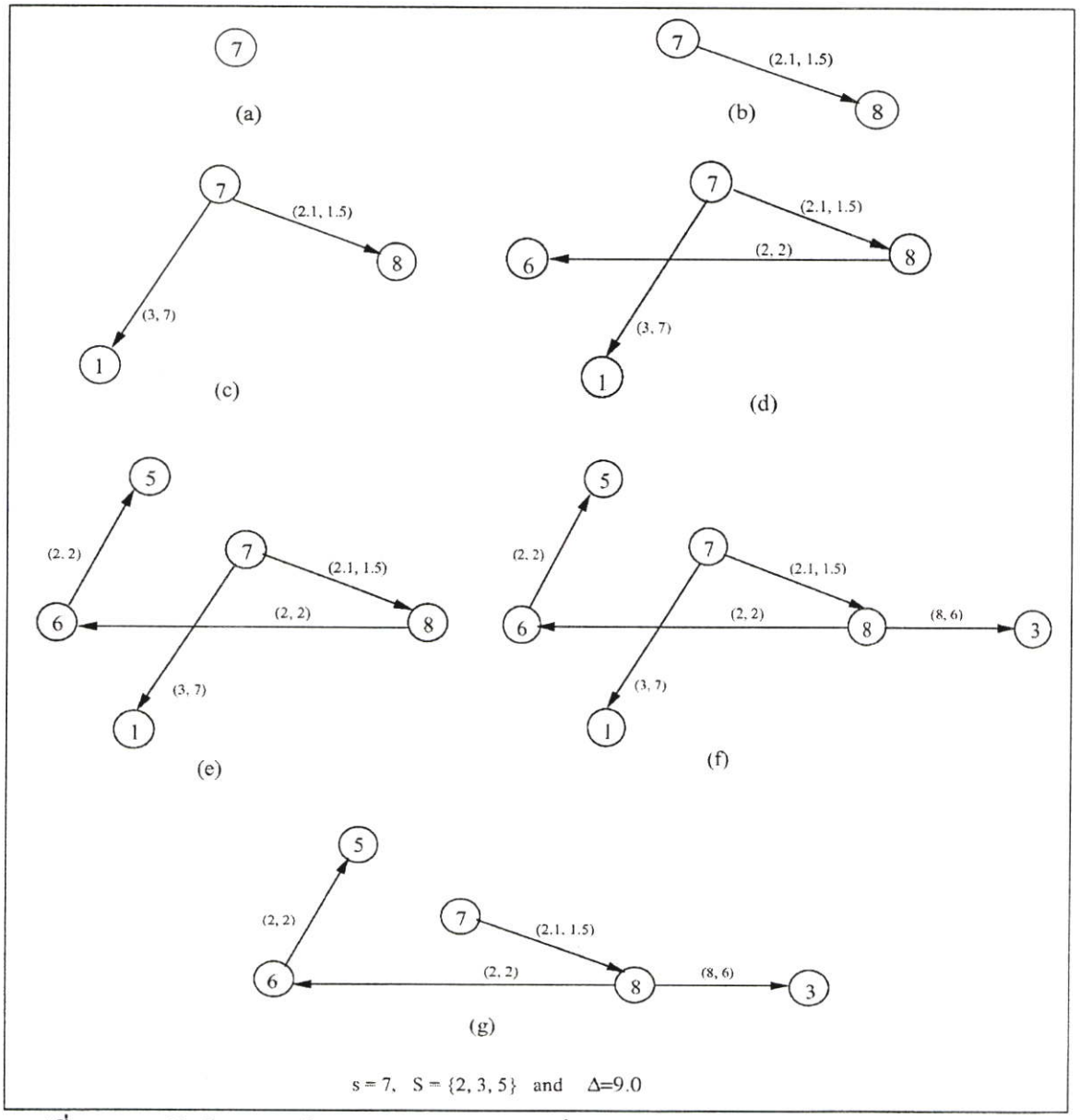
ถ้าเรามีความคุ้นเคยกับ Dijkstra's shortest path algorithm เราจะสามารถเข้าใจ CRA algorithm ในรูปที่ 3.4 ได้อย่างง่ายดาย ต่อไปนี้เราจะใช้รูปที่ 3.6 ในการอธิบายวิธีการสร้าง multicast tree จากเครือข่ายในรูปที่ 3.5 โดยใช้ CRA algorithm และกำหนดให้โหนดต้นทาง $s = 7$ โหนดปลายทาง $S = \{2,3,5\}$ และ $\Delta = 9.0$ โดยมีวิธีการดังนี้



รูปที่ 3.5 ตัวอย่างเครือข่ายขนาด 8 โหนด

1. โหนด 7 มีเส้นทางที่สามารถเชื่อมต่อไปยังโหนดอื่นได้ 3 เส้นทางที่ใช้เวลาไม่เกิน Δ คือ (7->1), (7->5) และ (7->8) ซึ่งมีค่า cost เป็น 3, 10 และ 2.1 ตามลำดับ เส้นทาง (7->8) ถูกเลือกเนื่องจากมีค่า cost ต่ำที่สุดคือ 2.1 ดังนั้น โหนด 7 และ 8 จะถือว่าเป็นโหนดที่ถูกพิจารณาแล้ว และจะได้ tree ดังแสดงในรูปที่ 3.6(a) และ 3.6(b)
2. โหนด 7 และ 8 มีเส้นทางที่สามารถเชื่อมต่อไปยังโหนดอื่นได้ 4 เส้นทางที่ใช้เวลาไม่เกิน Δ คือ (7->1), (7->8->3), (7->5) และ (7->8->6) ซึ่งมีค่า cost เป็น 3, 10.1, 10 และ 4.1 ตามลำดับ เส้นทาง (7->1) ถูกเลือกเนื่องจากมีค่า cost ต่ำที่สุดคือ 3 ดังนั้น โหนด 1 จะถือว่าเป็นโหนดที่ถูกพิจารณาแล้ว และจะได้ tree ดังแสดงในรูปที่ 3.6(c)
3. โหนด 1, 7 และ 8 มีเส้นทางที่สามารถเชื่อมต่อไปยังโหนดอื่นได้ 3 เส้นทางที่ใช้เวลาไม่เกิน Δ คือ (7->8->3), (7->5) และ (7->8->6) ซึ่งมีค่า cost เป็น 10.1, 10 และ 4.1 ตามลำดับ เส้นทาง (7->8->6) ถูกเลือกเนื่องจากมีค่า cost ต่ำที่สุดคือ 4.1 ดังนั้น โหนด 6 จะถือว่าเป็นโหนดที่ถูกพิจารณาแล้ว และจะได้ tree ดังแสดงในรูปที่ 3.6(d)
4. โหนด 1, 6, 7 และ 8 มีเส้นทางที่สามารถเชื่อมต่อไปยังโหนดอื่นได้ 3 เส้นทางที่ใช้เวลาไม่เกิน Δ คือ (7->8->3), (7->5) และ (7->8->6->5) ซึ่งมีค่า cost เป็น 10.1, 10 และ 6.1 ตามลำดับ เส้นทาง (7->8->6->5) ถูกเลือกเนื่องจากมีค่า cost ต่ำที่สุดคือ 6.1 ดังนั้น โหนด 5 จะถือว่าเป็นโหนดที่ถูกพิจารณาแล้ว และจะได้ tree ดังแสดงในรูปที่ 3.6(e)

5. โหนด 1, 5, 6, 7 และ 8 มีเส้นทางที่สามารถเชื่อมต่อไปยังโหนดอื่นได้เพียงเส้นทางเดียวที่ใช้เวลาไม่เกิน Δ คือ (7->8->3) ซึ่งมีค่า cost เป็น 10.1 ดังนั้น โหนด 3 จะถือว่าเป็นโหนดที่ถูกพิจารณาแล้ว และจะได้ tree ดังแสดงในรูปที่ 3.6(f)
6. หลังจากที่ tree ไม่สามารถสร้างต่อได้แล้ว โหนดปลายทุกโหนดที่ไม่ได้เป็นโหนดปลายทางจะถูกตัดออก จากรูปที่ 3.6(f) จะเห็นว่าโหนด 1 เป็นโหนดปลายที่ไม่ใช่โหนดปลายทาง ดังนั้น โหนด 1 จึงถูกตัดทิ้ง และจะได้ผลลัพธ์เป็น tree ที่แสดงดังรูปที่ 3.6(g)



รูปที่ 3.6 ผลลัพธ์จากการทำงานของ CRA อัลกอริทึม

จากผลลัพธ์ที่แสดงในรูปที่ 3.6 จะเห็นว่า tree ที่สร้างโดย CRA อาจจะไม่สามารถกระจายไปยังทุกโหนดปลายทางได้นั่นก็คือโหนด 2 ทั้งๆที่ยังมีเส้นทางที่สามารถไปยังโหนด 2 ได้โดยไม่เกินเวลาที่กำหนดอยู่ ซึ่งเหตุการณ์ดังกล่าวนี้จะไม่เกิดขึ้นใน CKMB algorithm

3.4.2 CKMB algorithm[3]

ขั้นตอนของ CKMB จะประกอบไปด้วย 5 ขั้นตอนด้วยกัน โดยขั้นตอนทั้ง 5 นี้ จะสามารถคำนวณหา tree ที่มี s เป็น root แล้วกระจายไปยังกลุ่มของโหนดปลายทาง S ได้ ดังแสดงในรูปที่ 3.7 ในขั้นตอนที่ 1 ของ CKMB จะทำการคำนวณหา tree T_1 ที่มีค่า cost ต่ำ และมี s เป็น root แล้วกระจายไปยังกลุ่มของโหนดปลายทาง $S_1 (S_1 \subseteq S)$ โดยใช้ CRA ถ้า T_1 ไม่สามารถกระจายไปยังทุกโหนดปลายทางใน S ได้ Tree T_2 ที่มีค่า delay ต่ำและกระจายไปยังโหนด $S - S_1$ จะถูกสร้างขึ้นในขั้นตอนที่ 2 โดยใช้ Dijkstra's shortest path algorithm ในขั้นตอนที่ 1 และ 2 จะสามารถคำนวณหาเส้นทางจากโหนด s ไปยังโหนดปลายทาง $i \in S$ ได้ โดยที่ค่า delay $d_s(i)$ ของเส้นทางจาก s ไป i จะมีค่าอยู่ในช่วงของ Δ

ในขั้นตอนที่ 3 จะทำการสร้างเครือข่าย G_1 จากเครือข่าย G โดยการลบโหนด s และลิงค์ที่เชื่อมต่อโหนด s ออกให้หมด แล้วทำการสร้างกลุ่มของ tree ที่มี root เป็น $v \in S$ แล้วกระจายไปยังโหนดปลายทาง $S - \{v\}$ ที่เหลือโดยใช้ CRA ทุกครั้งโหนดปลายทางจะถูกเลือกขึ้นมาด้วย

$$d_s(v) = min\{d_s(i)\} \text{ for } i \in S$$

แล้วสร้าง tree ที่มี root อยู่ที่โหนด v ด้วย CRA ในขั้นตอนที่ 3 นี้ จะพยายามคำนวณหาเส้นทางจากทุกโหนดปลายทางไปยังโหนดปลายทางอื่นๆ และมีจุดมุ่งหมายที่ delay ที่น้อยกว่า Δ โดยใช้ CRA

หลังจากขั้นตอนที่ 1, 2 และ 3 แล้ว กราฟ G' จะถูกสร้างขึ้นจากโหนด $\{s\} \cup S$ โดยลิงค์ของกราฟนี้จะมีเส้นทางที่ถูกสร้างขึ้นมาสัมพันธ์กันอยู่ ขั้นตอนที่ 4 จะเป็นการนำเอา G' มาสร้าง spanning tree โดยใช้ CRA' (เป็น CRA ที่ถูกปรับปรุงเพื่อให้เหมาะสมกับการนำมาใช้) ขั้นตอนที่ 5 จะทำการสร้าง tree จาก tree ในขั้นตอนที่ 4 โดยใช้เส้นทางที่ถูกสร้างขึ้นมาสัมพันธ์กันก่อนหน้านี้ ซึ่งจะแสดงให้เห็นในตัวอย่างต่อไปนี้

รูปที่ 3.8 จะแสดงถึงการทำงานของ CKMB โดยใช้เครือข่ายในรูปที่ 3.5 โดยกำหนดให้ โหนดต้นทาง $s = 7$ โหนดปลายทาง $S = \{2,3,5\}$ และ $\Delta = 9.0$ โดยมีขั้นตอนดังนี้

1. สร้าง tree T_1 โดยใช้ CRA ดังแสดงในรูปที่ 3.8(a)

2. เนื่องจากโหนด 2 ไม่ได้อยู่ใน tree T_1 ดังนั้น tree T_2 จึงถูกสร้างขึ้น เพื่อให้ได้เส้นทางจากโหนด 7 ไป 2 ดังแสดงในรูปที่ 3.8(b)

ในขั้นตอนที่ 1 และ 2 ได้สร้างเส้นทางจากโหนดต้นทาง 7 ไปยังโหนดปลายทาง 2, 3 และ 5 ดังนี้ คือ

จาก 7 ไป 5 ใช้เส้นทาง 7->8->6->5 และมี delay เป็น $d_7(5)=5.5$ (จากขั้นตอนที่ 1)

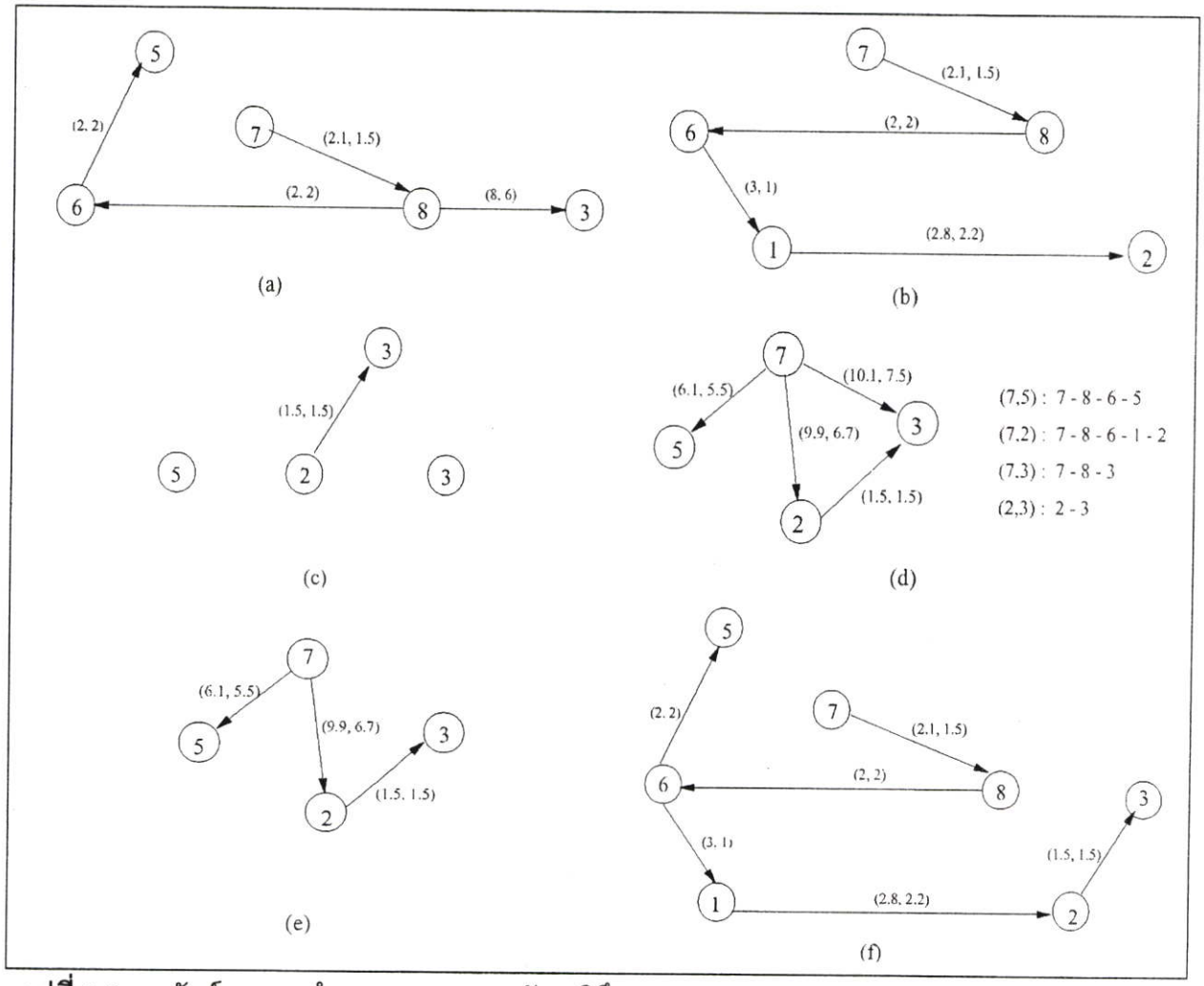
จาก 7 ไป 3 ใช้เส้นทาง 7->8->3 และมี delay เป็น $d_7(3)=7.5$ (จากขั้นตอนที่ 1)

จาก 7 ไป 2 ใช้เส้นทาง 7->8->6->1->2 และมี delay เป็น $d_7(2)=6.7$ (จากขั้นตอนที่ 2)

```
Steps taken by CKMB(s, S, Δ, C, D);

/*
s is the source node, S is the set of destination nodes. Δ is the bounded delay. C and D
are link costs and link delays of the network G respectively.
*/

Step 1: Call CRA(s, S, Δ, C, D) to compute a low cost tree T1 rooted at s. Let S1(S1 ⊆ S)
        denote the set of destination nodes spanned by T1 and ds(i) the delay of the path
        from s to the node i ∈ S1 in T1.

Step 2: If S1 ⊂ S, compute a least delay tree T2 rooted at s and spanning all the nodes
        in S-S1. Let ds (i) denote the delay of the path from s to the node i ∈ S - S1 in
        T2.

Step 3: Simplify G to G1 by deleting s and all links connecting s in G. Let C1 and D1
        denote link costs and link delays of G1 respectively.
        Find node v such that ds(v) = min{ds(i)} for i ∈ S;
        while ds(v) < Δ do
             call CRA(v, (S - {v}), (Δ - ds(v)), C1, D1) to compute a tree Tv(Vv, Ev);
             if Ev is not empty, then
                  let dv(i) denote the delay of the path in Tv from v to i ∈ S - {v};
                  for all i ∈ S - {v} with ds(i) < Δ
                       if ds(i) > ds(v) + dv(i) then ds(i) ← ds(v) + dv(i);
             set ds(v) ← Δ;
             find node v such that ds(v) = min{ds(i)} for I ∈ S;
        end while

/*
Step 1, 2 and 3 generate a directed graph G' with the nodes being {s} ∪ S. The link
(v1, v2) in G' represents the corresponding path from v1 to v2 generated from step 1 to
step 3. Note that G' may not be a complete graph.
*/

Step 4: Construct a spanning tree of G' satisfying the delay constraint Δ:
        Let C' and D' represent the link costs and link delays of G' respectively.
        Replace line (16) of the procedure CRA :
                vc ← CST(u) +C(u, w);
        with
                vc ← C(u, w);
        Let this changed version of CRA denoted by CRA'.
        Call CRA'(s, S, Δ, C', D') to compute a spanning tree of G' rooted at s.

Step 5: Expand the links of the constructed constrained spanning tree into the
        corresponding paths they represent (and remove any loops that may be caused by
        this expansion) to give the constrained multicast routing tree solution.
```

รูปที่ 3.7 CKMB อัลกอริทึม

3. เนื่องจาก $d_7(5)=\min\{ d_7(2), d_7(3), d_7(5)\}$ tree ที่มีโหนด 5 เป็น root แล้วกระจายไปยังโหนด 2 และ 3 จึงถูกสร้างขึ้นโดย CRA ที่ใช้ค่า delay เป็น $\Delta - d_7(5)=3.5$ แต่ไม่สามารถสร้าง tree ดังกล่าวได้ $d_7(5)$ จึงถูกกำหนดค่าใหม่ให้เป็น Δ ต่อมา $d_7(2)=\min\{ d_7(2), d_7(3), d_7(5)\}$ tree ที่มีโหนด 2 เป็น root แล้วกระจายไปยังโหนด 3 และ 5 จึงถูกสร้างขึ้นโดย CRA ที่ใช้ค่า delay เป็น $\Delta - d_7(2)=2.3$ tree ที่คำนวณมาได้มีเพียงลิงค์ (2,3) เท่านั้น และเนื่องจากว่า $d_7(2)+ d_2(3)=6.7+1.5 > d_7(3)$ จึงไม่จำเป็นที่จะต้องเปลี่ยนแปลงค่าของ $d_7(3)$. $d_7(2)$ จึงถูกกำหนดค่าใหม่ให้เป็น Δ ด้วยเช่นกัน สุดท้าย tree ที่มีโหนด 3 เป็น root แล้วกระจายไปยังโหนด 2 และ 5 จึงถูกสร้างขึ้นโดย CRA ที่ใช้ค่า delay เป็น $\Delta - d_7(3)=1.5$ แต่ไม่สามารถสร้าง tree ดังกล่าวได้ ซึ่งผลลัพธ์จะแสดงในรูปที่ 3.8(c)



รูปที่ 3.8 ผลลัพธ์จากการทำงานของ CKMB อัลกอริทึม

ในขั้นตอนที่ 1, 2 และ 3 ได้สร้างกราฟ G' ที่มีโหนด {7, 2, 3, 5} ดังแสดงในรูปที่ 3.8(d)

4. ใช้ CRA' สร้าง tree จาก G' จะได้ tree ดังแสดงในรูปที่ 3.8(e)
5. Multicast tree ที่มี cost เท่ากับ 13.4 ถูกสร้างขึ้นจากเส้นทางที่สัมพันธ์กับเส้นทางในรูปที่ 3.8(d) ดังแสดงในรูปที่ 3.8(f)

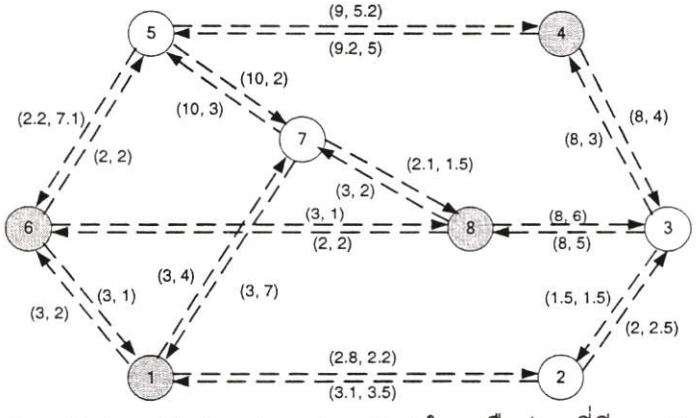
3.4.3 Genetic Algorithm (GA)[5]

เป็นอัลกอริทึมที่มีหลักการอยู่บนการค้นหาคำตอบจากกลุ่มคำตอบที่มีอยู่ เพื่อสร้างคำตอบใหม่จากกลุ่มคำตอบเดิม โดยการเลียนแบบกระบวนการแลกเปลี่ยนข้อมูลทางธรรมชาติของยีนส์ เช่น ครอสโอเวอร์ และ มิวเตชั่น โดยที่รูปแบบของปัญหานั้นจะต้องถูกแปลงให้อยู่ในรูปแบบของโครโมโซมที่เหมาะสมกับปัญหา แล้วใช้ฟังก์ชันความเหมาะสมเพื่อประเมินค่าของโครโมโซมแต่ละตัวในกลุ่มของคำตอบแต่ละรุ่น ซึ่งโครโมโซมที่มีค่าความเหมาะสมดี ก็จะมีสิทธิที่จะถูกเลือกให้เป็นโครโมโซมในรุ่นถัดไปได้สูงกว่าโครโมโซมตัวอื่น โดยมีวงจรการทำงานดังนี้

1. สร้างกลุ่มของโครโมโซมชุดแรกขึ้นมาจากการสุ่ม
2. ใช้ fitness function ประเมินค่าความเหมาะสมของโครโมโซมทุกตัว
3. สร้างกลุ่มของโครโมโซมชุดใหม่จากโครโมโซมชุดเดิม ด้วยกระบวนการครอสโอเวอร์ และ มิวเตชั่น โดยโครโมโซมที่เลือกเข้ามากระทำจะพิจารณาจากค่าความเหมาะสม
4. ใช้ fitness function ประเมินค่าความเหมาะสมของโครโมโซมที่สร้างขึ้นมาใหม่
5. ทำซ้ำ ข้อ 3) และ 4) จนกว่าจะได้คำตอบที่พอใจ

3.4.3.1 Steiner Node

คือ โหนดที่เหลือทั้งหมดในเครือข่าย ที่ไม่ใช่โหนดต้นทาง และโหนดปลายทาง เช่น จากเครือข่ายในรูปที่ 2 หากโหนดต้นทางเป็น 7 และโหนดปลายทางเป็น 2, 3, 5 ดังนั้น stiner node จะเป็น 1, 4, 6, 8 ดังแสดงในรูปที่ 3.9



รูปที่ 3.9 แสดง Steiner Nodes: S_t = 1, 4, 6, 8 ใน เครือข่าย ที่มี s = 7; S = 2, 3, 5

3.4.3.2 Genotype

Genotype จะเป็นตัวกำหนดชุดของ Steiner โหนดที่จะถูกเลือก ด้วยการใช้สายอักขระของตัวอักษรที่มีค่า 0 และ 1 มาเป็นตัวกำหนด ให้ Steiner โหนด $V-S-\{s\}$ มีหมายเลขกำกับเป็น 0, 1, 2, ..., (n-m-2) ชุดของ genotype จะเป็นดังนี้

$$\{g(0), g(1), ..., g(n-m-2)\}$$

เมื่อ $g(i) \in \{0,1\},\ i = 0,1,...,(n-m-2)$ เซ็ตของ Steiner โหนด $S_T \subseteq V-S-\{s\}$ ที่ถูกกำหนดโดย genotype คือ $S_T = \{v_i \in V \mid g(i) = 1, i = 0,1,...,n-m-2\}$

ตัวอย่าง: ในรูปที่ 3.9 ให้โหนดต้นทาง $s = 7$ โหนดปลายทาง $S = \{2,3,5\}$ ดังนั้น $S_T = V-S-\{s\} = \{1,4,6,8\}$ และ genotype เป็นเซ็ตของ $\{g(0), g(1), g(2), g(3)\}$ เพราะฉะนั้น genotype ที่มีค่าเป็น {0, 1, 1, 0} จะหมายถึง Steiner โหนด $S_T = \{4,6\}$

3.4.3.3 Decoder

Genotype จะต้องถูกถอดรหัสเพื่อที่จะสามารถนำไปใช้หาค่าความเหมาะสมต่อไป CKMB ถูกนำมาใช้ในการ decode ในครั้งนี้ ซึ่งมีขั้นตอนในการ decode อยู่ 2 ขั้นตอน คือ

1. เรียกใช้ CKMB($s, S \cup S_T, \Delta, C, D$) เพื่อคำนวณหา tree T ที่มี s เป็น root และกระจายไปยังโหนด $S \cup S_T$
2. ตัดโหนดปลายใน T ที่ไม่มีอยู่ใน S ออก

เพื่อที่จะลดเวลาในการ decode ลง วิวัฒนาการของ GA จึงถูกแบ่งออกเป็น 2 ส่วนคือ PREPARATION() และ DECODING() โดยที่ PREPARATION() จะสร้าง directed graph G_0' ด้วยโหนดที่อยู่ใน V โดยใช้ CKMB ขั้นตอนที่ 1-3 และกำหนดให้ $S = V - s$ และลิงค์ (v_i, v_j) ใน G_0' คือเส้นทางจาก v_i ไปยัง v_j ใน $G(V,E)$ การทำ DECODING() มีขั้นตอนดังนี้

1. สร้าง directed graph G_1' ด้วยโหนด $s \cup S \cup S_T$ ที่ได้จาก G_0' โดยการลบโหนด $V-s-S-S_T$ และลิงค์ที่เชื่อมต่อโหนดเหล่านี้ใน G_0'
2. สร้าง spanning tree T_1' จาก G_1'
3. แปลงลิงค์ใน T_1' ให้เป็นเส้นทางที่สอดคล้องกัน (ที่ถูกสร้างในขั้นตอนของการทำ PREPARATION()) แล้วทำการกำจัดลูปที่เกิดขึ้น
4. ตัดโหนดปลายใน T ที่ไม่มีอยู่ใน S ออก

3.4.3.4 Firness Measure

ค่าความเหมาะสมของประชากรในแต่ละรุ่นมีความจำเป็นเพื่อใช้คัดเลือกประชากรในรุ่นถัดไป โดยสามารถคำนวณได้จาก

$$F(i)=1-\frac{C_T(i)}{C_{TM}}$$

โดยที่ $C_T(i)$ หมายถึงค่า cost ของ tree ที่ได้จากการ decode โครโมโซมตัวที่ i และให้

$$C_{TM}=\max\{C_T(j)\} \quad for\ j=0,1,...,N-1 \quad N=Population\ size$$

ถ้าหาก $F(i)=(0\le F(i)\le 1)$ มีค่ามาก จะหมายถึงโครโมโซมตัวที่ i มีประสิทธิภาพที่ดี

3.4.3.5 **การใช้ GA เพื่อแก้ปัญหาแบบ** multicast

GA ในรูปที่ 3.10 จะทำ PREPARATION() เป็นอันดับแรก ประชากรเริ่มต้น (Φ_c) จะถูกสร้างขึ้นจากการสุ่ม ด้วยฟังก์ชัน Generate() และขนาดของประชากรจะมีขนาดคงที่เท่ากับ $N=|\Phi_c|$ ฟังก์ชัน Evaluate() จะ decode โครโมโซมทุกตัวโดยเรียกใช้ฟังก์ชัน DECODING()

```
PREPARATION();                                                          (1)
Φc ← Generate();                                                        (2)
Evaluate(Φc);                                                           (3)
Repeat                                                                  (4)
        Φc = 0;                                                         (5)
        for i ← 1 to N/2 do                                             (6)
                select φ1 ∈ Φc, φ2 ∈ Φc;                                (7)
                {φ1c, φ2c} = crossover(φ1, φ2, Pc);                     (8)
                ψ1 = mutation(φ1c, Pm);                                 (9)
                ψ2 = mutation(φ2c, Pm);                                 (10)
                Φn ← Φn ∪ {ψ1, ψ2};                                     (11)
        end for                                                         (12)
        Φc ← Φn;                                                        (13)
        Evaluate(Φc);                                                   (14)
Until StopCriterion;                                                    (15)
Output the individual with the largest fitness in Φc;                   (16)
```

รูปที่ 3.10 รายละเอียดการทำงานของ GA

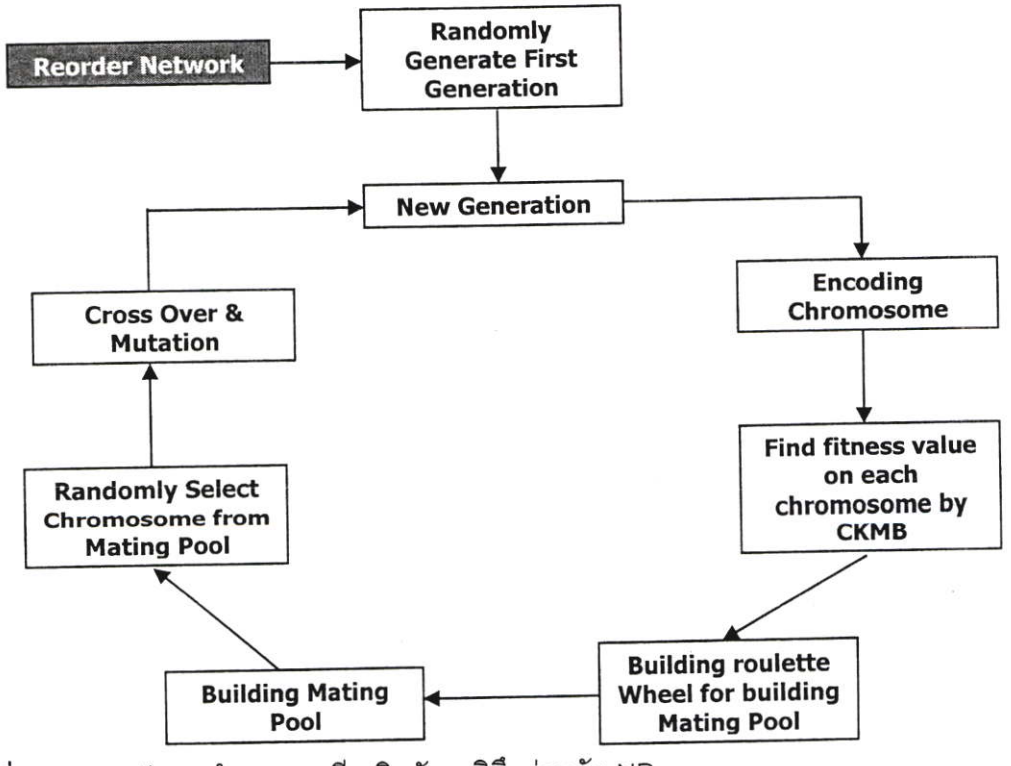
และคำนวณหาค่าความเหมาะสมที่สัมพันธ์กับโครโมโซมของประชากรนั้นๆ บรรทัดที่ 5-12 เป็นวิวัฒนาการโดยทั่วไปของ GA ที่สร้างประชากรรุ่นใหม่ Φ_n จากประชากรในรุ่นปัจจุบัน Φ_c โดย

ในบรรทัดที่ 7 จะเป็นการเลือกโครโมโซม ϕ_1 และ ϕ_2 จาก Φ_c โดยใช้ค่าความเหมาะสมที่คำนวณจากฟังก์ชัน Evaluate() เป็นตัวตัดสิน หลังจากนั้น โครโมโซมตัวอ่อน 2 โครโมโซม ϕ_1^c และ ϕ_2^c จะถูกสร้างขึ้นด้วยฟังก์ชัน crossover() และ mutation() ตามลำดับ เพื่อสร้างโครโมโซมคู่ใหม่ ψ_1 และ ψ_2 เพื่อใช้เป็นประชากร Φ_n รุ่นใหม่ต่อไป ปัจจัยที่กำหนดให้ GA หยุดดำเนินการคือ จะต้องมีโครโมโซมอย่างน้อย $N/2$ โครโมโซมที่มีค่าความเหมาะสมมากกว่า 99% ของค่าความเหมาะสมที่มากที่สุดในประชากรรุ่นนั้น และจะต้องมีวิวัฒนาการอย่างน้อย 5 รุ่นเป็นอย่างน้อย

บทที่ 4
แนวทางที่ใช้ในการวิจัย

เนื่องจาก GA มีประสิทธิภาพในการสร้าง multicast tree ที่มีค่า cost ใกล้เคียง optimum มากกว่า heuristic algorithm อื่นๆที่ได้กล่าวไปแล้ว แต่ประชากรเริ่มต้น(โครโมโซม)ของ GA นั้นเกิดขึ้นจากการสุ่ม ซึ่งหากเรานำโครงสร้างโครโมโซมของจีเนติกอัลกอริทึมแบบเดิมมาวิเคราะห์เพื่อปรับปรุงรูปแบบของโครโมโซมให้มีคุณภาพมากยิ่งขึ้น ผลก็คือ เมื่อเรานำโครโมโซมดังกล่าวมาจัดเรียงเรียงรูปแบบขึ้นมาใหม่ก่อนแล้วค่อยส่งต่อให้ GA ประมวลผล ส่งผลให้ผลลัพธ์ที่ได้จากการสร้าง multicast tree โดยใช้โครโมโซมแบบใหม่นี้ มีค่า cost โดยรวมต่ำลง และเวลาที่ใช้ในการประมวลผลเพื่อสร้าง multicast tree ก็น้อยลงด้วยเช่นกัน ซึ่งวิธีการที่เรานำมาใช้นั้นจะขอเรียกว่า การทำ Network Reorderation (NR)

ก่อนที่เราจะกล่าวถึงขั้นตอนในการทำ NR นั้น เราจะขอกล่าวถึงอัลกอริทึมที่ใช้หา minimum spanning tree ที่เป็นที่รู้จักกันดีก่อน นั่นก็คือ Prim's Algorithm เนื่องจากว่า NR จะใช้ Prim's Algorithm เป็นหัวใจหลักในการดำเนินการเพื่อจัดเรียงเครือข่าย จึงมีความจำเป็นอย่างยิ่งที่จะต้องเข้าใจอัลกอริทึมนี้ ภาพโดยรวมของการทำงานของจีเนติกอัลกอริทึมร่วมกับ NR จะแสดงดังรูปที่ 4.1



รูปที่ 4.1 แผนผังการทำงานของจีเนติกอัลกอริทึมร่วมกับ NR

4.1 Prim's Algorithm

อัลกอริทึมนี้เป็นอัลกอริทึมที่มีแนวคิดแบบ greedy heuristic ที่สามารถสร้าง spanning tree ที่มีค่า cost ต่ำ เราจึงนำอัลกอริทึมนี้มาใช้จัดเรียงเครือข่ายขึ้นมาใหม่ก่อนที่จะส่งให้ GA

```
Prim(G,w)
Q ← 0                                                          (1)
for each u ∈ V[G]                                              (2)
    do Initialize-Key[u] ← ∞                                   (3)
       Q ← Q ∪ {u}                                             (4)
r ← some vertex of G[V]                                        (5)
edge[r] ← NIL                                                  (6)
Decrease-Key(r,0)                                              (7)
T ← 0                                                          (8)
While Q ≠ 0                                                    (9)
    Do u ← Remove-Smallest-Key(Q)                              (10)
       T ← T ∪ {edge[u]}                                       (11)
       for each v ∈ Adj[u]                                     (12)
           do if v ∈ Q and w(u,v) < key[v]                     (13)
              then Decrease-Key(v, w(u,v))                     (14)
                   edge[u] ← (u,v)                             (15)
```

รูปที่ 4.2 Prim อัลกอริทึม

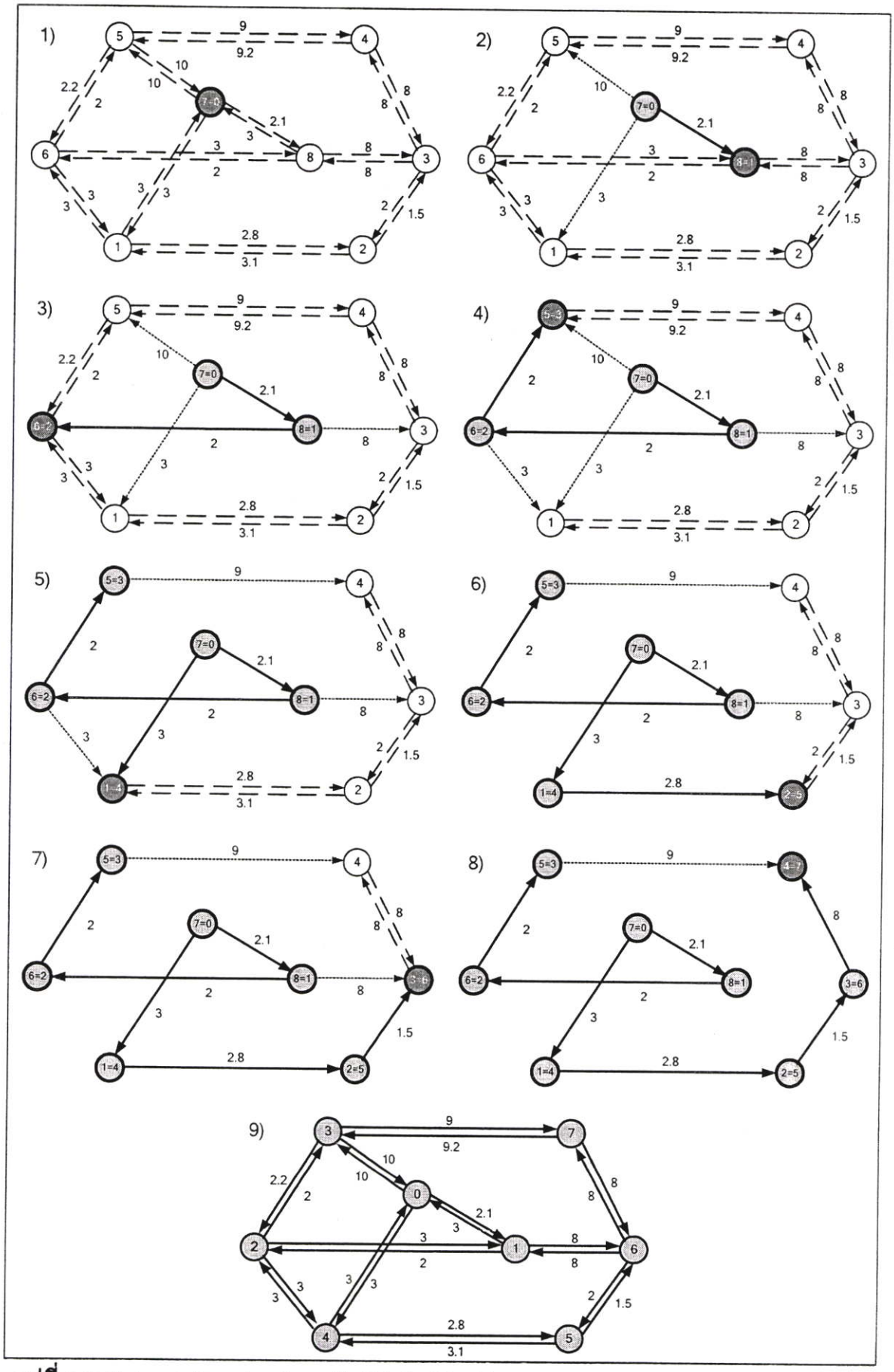
ทำงาน โดยเครือข่ายที่ถูกจัดเรียงขึ้นมาใหม่นั้นจะเป็นเครือข่ายที่มีหมายเลขโหนดใกล้กันจะมีระยะทางใกล้กัน ซึ่งจะส่งผลให้โครโมโซมที่จะใช้ระบุ Steiner node ขึ้นมาเพื่อใช้งานมีโครงสร้างในลักษณะดังกล่าวด้วยเช่นกันคือ โหนดที่อยู่ติดๆกันในโครโมโซมจะมีระยะทางใกล้กัน โดย Prim's Algorithm จะแสดงดังรูปที่ 4.2 ซึ่งวิธีการและขั้นตอนของการทำงานจะขอใช้รูปที่ 4.3 อธิบายเพื่อความง่ายในการทำความเข้าใจ

4.2 Network Reorderation

เป็นการจัดลำดับเครือข่ายขึ้นใหม่เพื่อให้โครโมโซมของจีเนติกอัลกอริทึมมีคุณภาพมากขึ้น โดยใช้ heuristic algorithm ที่มีชื่อว่า Prim ในการจัดเรียง

การจัดลำดับจะทำโดยให้โหนดที่มีระยะทางใกล้ๆกันจะต้องมีหมายเลขใกล้ๆกัน ส่วนโหนดที่มีระยะทางห่างออกไปก็ให้มีหมายเลขโหนดไกลกันออกไปเรื่อยๆ โดยอ้างอิงจากโหนดเริ่มต้น และมีขั้นตอนโดยใช้ Prim ดังนี้ (ดูรูปที่ 4.3 ประกอบ) คือ

1. กำหนดให้โหนดเริ่มต้นเป็นโหนดหมายเลข 7 ดังนั้นโหนด 7 จะถูกเปลี่ยนเป็นโหนดแรกในเครือข่ายแบบ reorder (ในเครือข่ายแบบ reorder โหนดแรกคือโหนด 0) เพราะฉะนั้น 7=0 ดังแสดงในรูปที่ 4.3(1)

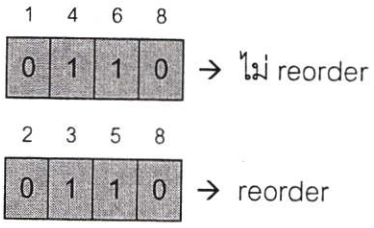


รูปที่ 4.3 Network Reorderation with Prim's Algorithm

2. ที่โหนดเริ่มต้น (7=0) มีลิงค์เชื่อมโยงทั้งหมด 3 ลิงค์ คือ (7->1), (7->5) และ (7->8) ซึ่งมีค่า cost เป็น 3, 10 และ 2.1 ตามลำดับ โดยลิงค์ที่มีค่า cost ต่ำที่สุดจะถูกเลือก นั่นคือ ลิงค์ (7->8) ซึ่งมีค่า cost เป็น 2.1 หมายเลขของโหนดที่เชื่อมโดยลิงค์ดังกล่าวจะถูกเปลี่ยนเป็นหมายเลขถัดไป (หมายเลขที่จัดเรียงใหม่) นั่นคือ โหนด 8 จะถูกเลี่ยนเป็นโหนด 1 (8=1) ดังแสดงในรูปที่ 4.3(2)
3. พิจารณาลิงค์ทั้งหมดที่เชื่อมระหว่างโหนด 7=0 และ 8=1 ที่ยังไม่ได้ถูกเลือก ซึ่งมีด้วยกัน 4 ลิงค์คือ (7->1), (7->5), (8->3) และ (8->6) และมีค่า cost เป็น 3, 10, 8 และ 2 ตามลำดับ ลิงค์ที่มีค่า cost ต่ำที่สุดจะถูกเลือก นั่นคือ ลิงค์ (8->6) ซึ่งมีค่า cost เป็น 2 หมายเลขของโหนดที่เชื่อมโดยลิงค์ดังกล่าวจะถูกเปลี่ยนเป็นหมายเลขถัดไป (หมายเลขที่จัดเรียงใหม่) นั่นคือ โหนด 6 จะถูกเลี่ยนเป็นโหนด 2 (6=2) ดังแสดงในรูปที่ 4.3(3)
4. พิจารณาลิงค์ทั้งหมดที่เชื่อมระหว่างโหนด 6=2, 7=0 และ 8=1 ที่ยังไม่ได้ถูกเลือก ซึ่งมีด้วยกัน 5 ลิงค์คือ (6->1), (6->5), (7->1), (7->5) และ (8->3) และมีค่า cost เป็น 3, 2, 3, 10 และ 8 ตามลำดับ ลิงค์ที่มีค่า cost ต่ำที่สุดจะถูกเลือก นั่นคือ ลิงค์ (6->5) ซึ่งมีค่า cost เป็น 2 หมายเลขของโหนดที่เชื่อมโดยลิงค์ดังกล่าวจะถูกเปลี่ยนเป็นหมายเลขถัดไป (หมายเลขที่จัดเรียงใหม่) นั่นคือ โหนด 5 จะถูกเลี่ยนเป็นโหนด 3 (5=3) ดังแสดงในรูปที่ 4.3(4)
5. พิจารณาลิงค์ทั้งหมดที่เชื่อมระหว่างโหนด 5=3, 6=2, 7=0 และ 8=1 ที่ยังไม่ได้ถูกเลือก ซึ่งมีด้วยกัน 4 ลิงค์คือ (5->4), (6->1), (7->1) และ (8->3) และมีค่า cost เป็น 9, 3, 3 และ 8 ตามลำดับ ลิงค์ที่มีค่า cost ต่ำที่สุดจะถูกเลือก เนื่องจากลิงค์ที่มีค่า cost ต่ำที่สุดมีด้วยกัน 2 ลิงค์ และเป็นลิงค์ไปยังโหนดเดียวกัน คือ (6->1) และ (7->1) ลิงค์ที่มีจำนวน hop ย้อนกลับไปยังโหนดต้นทางน้อยที่สุดจะถูกเลือก นั่นคือลิงค์ (7->1) ซึ่งมีค่า cost เป็น 3 หมายเลขของโหนดที่เชื่อมโดยลิงค์ดังกล่าวจะถูกเปลี่ยนเป็นหมายเลขถัดไป (หมายเลขที่จัดเรียงใหม่) นั่นคือ โหนด 1 จะถูกเลี่ยนเป็นโหนด 4 (1=4) ดังแสดงในรูปที่ 4.3(5)
6. พิจารณาลิงค์ทั้งหมดที่เชื่อมระหว่างโหนด 1=4, 5=3 และ 8=1 ที่ยังไม่ได้ถูกเลือก (โหนด (6=2) และ (7=0) ไม่ถูกนำมาพิจารณา เนื่องจากโหนดที่อยู่ใกล้เคียงถูกนำมาใช้หมดแล้ว) ซึ่งมีด้วยกัน 3 ลิงค์คือ (1->2), (5->4) และ (8->3) และมีค่า cost เป็น 2.8, 9 และ 8 ตามลำดับ ลิงค์ที่มีค่า cost ต่ำที่สุดจะถูกเลือก นั่นคือ ลิงค์ (1->2) ซึ่งมีค่า cost เป็น 2.8 หมายเลขของโหนดที่เชื่อมโดยลิงค์ดังกล่าวจะถูกเปลี่ยนเป็นหมายเลขถัดไป (หมายเลขที่จัดเรียงใหม่) นั่นคือ โหนด 2 จะถูกเลี่ยนเป็นโหนด 5 (2=5) ดังแสดงในรูปที่ 4.3(6)

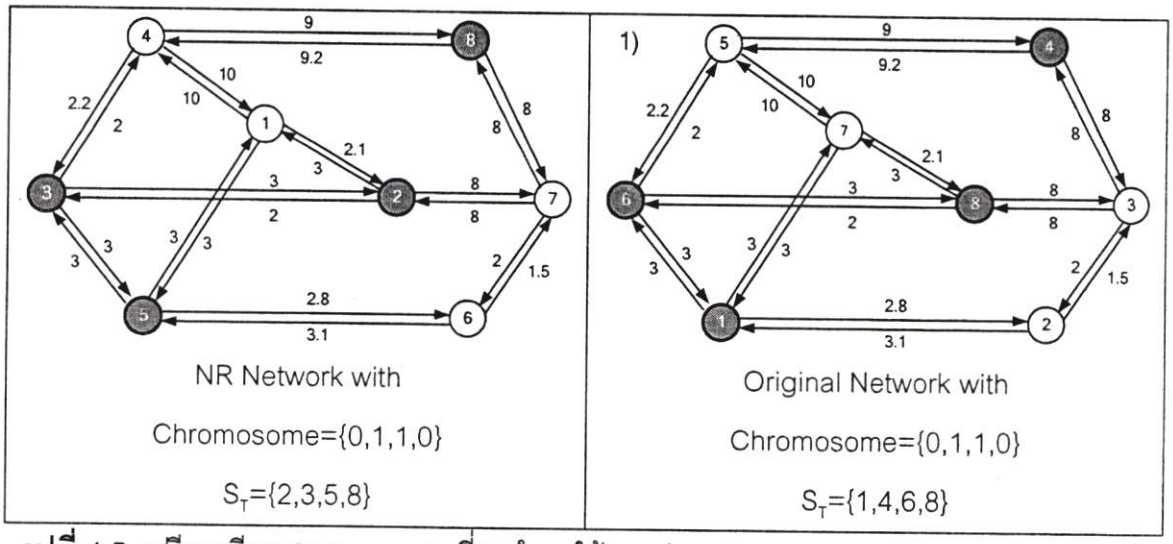
7. พิจารณาลิงค์ทั้งหมดที่เชื่อมระหว่างโหนด 2=5, 5=3 และ 8=1 ที่ยังไม่ได้ถูกเลือก (โหนด (1=4), (6=2) และ (7=0) ไม่ถูกนำมาพิจารณา เนื่องจากโหนดที่อยู่ใกล้เคียงถูกนำมาใช้หมดแล้ว) ซึ่งมีด้วยกัน 3 ลิงค์คือ (2->3), (5->4) และ (8->3) และมีค่า cost เป็น 1.5, 9 และ 8 ตามลำดับ ลิงค์ที่มีค่า cost ต่ำที่สุดจะถูกเลือก นั่นคือ ลิงค์ (2->3) ซึ่งมีค่า cost เป็น 1.5 หมายเลขของโหนดที่เชื่อมโดยลิงค์ดังกล่าวจะถูกเปลี่ยนเป็นหมายเลขถัดไป (หมายเลขที่จัดเรียงใหม่) นั่นคือ โหนด 3 จะถูกเลี่ยนเป็นโหนด 6 (3=6) ดังแสดงในรูปที่ 4.3(7)
8. พิจารณาลิงค์ทั้งหมดที่เชื่อมระหว่างโหนด 3=5 และ 5=3 ที่ยังไม่ได้ถูกเลือก (โหนด (1=4), (2=5), (6=2), (7=0) และ (8=1) ไม่ถูกนำมาพิจารณา เนื่องจากโหนดที่อยู่ใกล้เคียงถูกนำมาใช้หมดแล้ว) ซึ่งมีด้วยกัน 2 ลิงค์คือ (3->4) และ (5->4) และมีค่า cost เป็น 8 และ 9 ตามลำดับ ลิงค์ที่มีค่า cost ต่ำที่สุดจะถูกเลือก นั่นคือ ลิงค์ (3->4) ซึ่งมีค่า cost เป็น 8 หมายเลขของโหนดที่เชื่อมโดยลิงค์ดังกล่าวจะถูกเปลี่ยนเป็นหมายเลขถัดไป (หมายเลขที่จัดเรียงใหม่) นั่นคือ โหนด 4 จะถูกเลี่ยนเป็นโหนด 7 (4=7) ดังแสดงในรูปที่ 4.3(8)
9. รูปเครือข่ายที่ผ่านกระบวนการ NR แล้วจะแสดงดังรูปที่ 4.3(9)

จากผลลัพธ์ที่ได้ในขั้นตอนที่ 9 ในรูปที่ 4.3 จะส่งผลให้โครโมโซมที่มี Steiner node เรียงจากซ้ายไปขวา แสดงถึงระยะทางของ Steiner node จากใกล้ไปไกลตามลำดับ เช่น เมื่อกำหนดให้ s=7, S={2,3,5} ดังนั้น S_T={1,4,6,8} โครโมโซมของ Steiner node จะแสดงดังรูปที่ 4.4



รูปที่ 4.4 เปรียบเทียบโครโมโซมที่จัดเรียงใหม่กับโครโมโซมที่ไม่ได้จัดเรียง

จากรูปที่ 4.4 จะเห็นว่าโครโมโซมที่ไม่ได้ทำการจัดเรียงใหม่นั้นโหนดที่อยู่ติดกันในโครโมโซมไม่ได้แสดงว่าโหนดทั้งสองอยู่ใกล้กัน และโครโมโซมที่ได้จากเครือข่ายที่ถูกจัดเรียงขึ้นมาใหม่นั้นโหนดที่อยู่ติดกันในโครโมโซม จะหมายถึงโหนดทั้งสองอยู่ใกล้กัน ดังแสดงในรูปที่ 4.5



รูปที่ 4.5 เปรียบเทียบ Steiner node ที่ถูกนำมาใช้ ระหว่าง NR และ Original Network

บทที่ 5
ผลการทดลอง

5.1 เครือข่ายที่ใช้ในการทดลอง

การทดลองในครั้งนี้ได้นำข้อมูลเครือข่ายมาจาก Or-library[7] (สามารถดาวน์โหลดได้ที่ http://elib.zib.de/steinlib/steinlib.html) ซึ่งเป็นข้อมูลที่มีการทดสอบและหาค่า optimal tree cost ไว้แล้ว โดยข้อมูลที่นำมาทดสอบจะประกอบด้วย 4 ชุดข้อมูลด้วยกันคือ

1. ข้อมูลที่อยู่ใน Testset B ที่มีขนาด 50, 75 และ 100 โหนด รวม 18 เครือข่าย
2. ข้อมูลที่อยู่ใน Testset C ที่มีขนาด 500 โหนด จำนวน 20 เครือข่าย
3. ข้อมูลที่อยู่ใน Testset D ที่มีขนาด 1000 โหนด จำนวน 20 เครือข่าย

โดยรายละเอียดของข้อมูลเครือข่ายข้างต้นจะแสดงในตารางที่ 5.1, 5.2 และ 5.3 ตามลำดับ และภายในตารางจะประกอบด้วยรายละเอียดดังนี้

- |V| คือ จำนวนโหนดภายในเครือข่าย
- |E| คือ จำนวนลิงค์ภายในเครือข่าย
- |T| คือ จำนวนโหนดที่ใช้หามัลติคาสทรีภายในเครือข่าย

ตารางที่ 5.1 ข้อมูลเครือข่าย Testset B

Name	|V|	|E|	|T|	Opt
b01	50	63	9	**82**
b02	50	63	13	**83**
b03	50	63	25	**138**
b04	50	100	9	**59**
b05	50	100	13	**61**
b06	50	100	25	**122**
b07	75	94	13	**111**
b08	75	94	19	**104**
b09	75	94	38	**220**

Name	|V|	|E|	|T|	Opt
b10	75	150	13	**86**
b11	75	150	19	**88**
b12	75	150	38	**174**
b13	100	125	17	**165**
b14	100	125	25	**235**
b15	100	125	50	**318**
b16	100	200	17	**127**
b17	100	200	25	**131**
b18	100	200	50	**218**

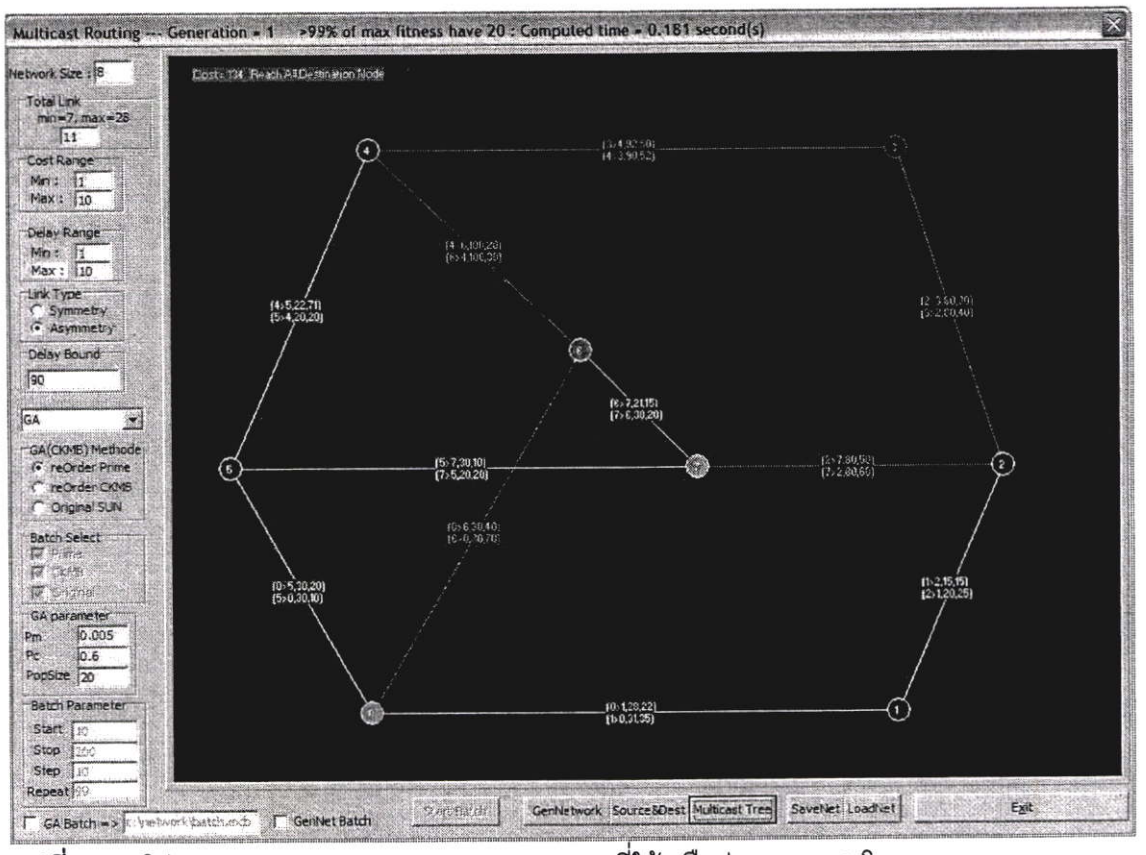
ตารางที่ 5.2 ข้อมูลเครือข่าย Testset C

Name	|V|	|E|	|T|	Opt
c01	500	625	5	**85**
c02	500	625	10	**144**
c03	500	625	83	**754**
c04	500	625	125	**1079**
c05	500	625	250	**1579**
c06	500	1000	5	**55**
c07	500	1000	10	**102**
c08	500	1000	83	**509**
c09	500	1000	125	**707**
c10	500	1000	250	**1093**

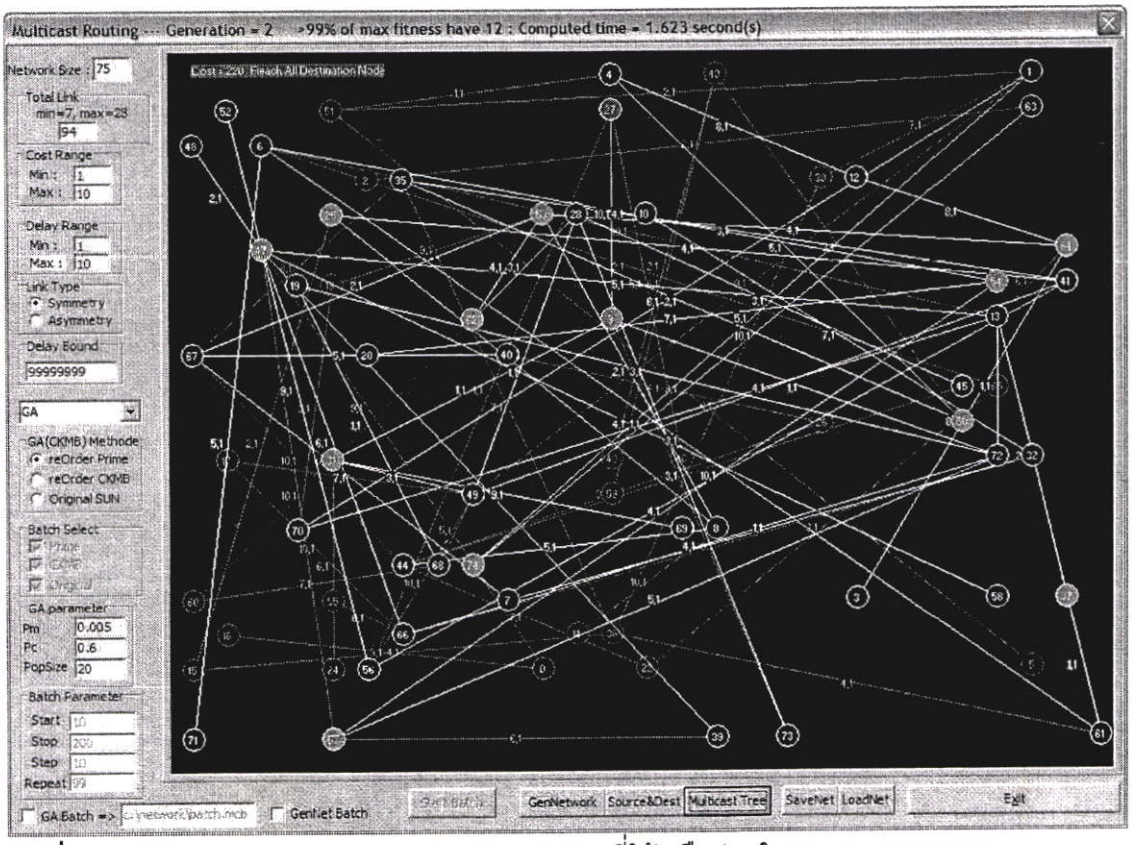
Name	|V|	|E|	|T|	Opt
c11	500	2500	5	**32**
c12	500	2500	10	**46**
c13	500	2500	83	**258**
c14	500	2500	125	**323**
c15	500	2500	250	**556**
c16	500	12500	5	**11**
c17	500	12500	10	**18**
c18	500	12500	83	**113**
c19	500	12500	125	**146**
c20	500	12500	250	**267**

5.2 อุปกรณ์ที่ใช้ในการทดลอง

ในการทดลองครั้งนี้ได้ใช้ Borland C++ builder 5.0 เป็นเครื่องมือในการสร้างโปรแกรม simulation เพื่อหาเส้นทางแบบหลายปลายทาง โดยใช้ CRA, CKMB, Genetic Algorithm และ Genetic Algorithm with NR เป็นอัลกอริทึมในการหาเส้นทาง โดยทดลองกับเครื่องไมโครคอมพิวเตอร์ที่มีหน่วยประมวลผลแบบ CPU เดียวเป็น Pentium IV 1600 MH_z หน่วยความจำ 256 MB ที่ความละเอียดจอภาพขนาด 1024x768 พิกเซล และใช้ระบบปฏิบัติการ การ Microsoft Windows 2000 Professional ในการรันโปรแกรม simulation นี้ ตัวอย่างของโปรแกรมที่ใช้ในการทดลองจะแสดงดังรูปที่ 5.1 โดยในรูปที่ 5.1 นั้น จะใช้เครือข่ายขนาด 8 โหนด และใช้พารามิเตอร์ของ GA ดังนี้ ความน่าจะเป็นในการครอสโอเวอร์ (Pc) เท่ากับ 60 เปอร์เซ็นต์ ความน่าจะเป็นในการมิวเตชั่น (Pm) เท่ากับ 0.5 เปอร์เซ็นต์ และจำนวนประชากร (popsize) เท่ากับ 20 ส่วนในรูปที่ 5.2 นั้น จะใช้เครือข่ายขนาด 75 โหนด และใช้พารามิเตอร์ของ GA ดังนี้ Pc=60 เปอร์เซ็นต์ Pm=0.5 เปอร์เซ็นต์ และ popsize=20 ซึ่งเป็นเครือข่ายที่ 9 (b09) ที่ได้จาก Testset B



รูปที่ 5.1 โปรแกรม Multicast Routing Simulation ที่ใช้เครือข่ายขนาด 8 โหนด



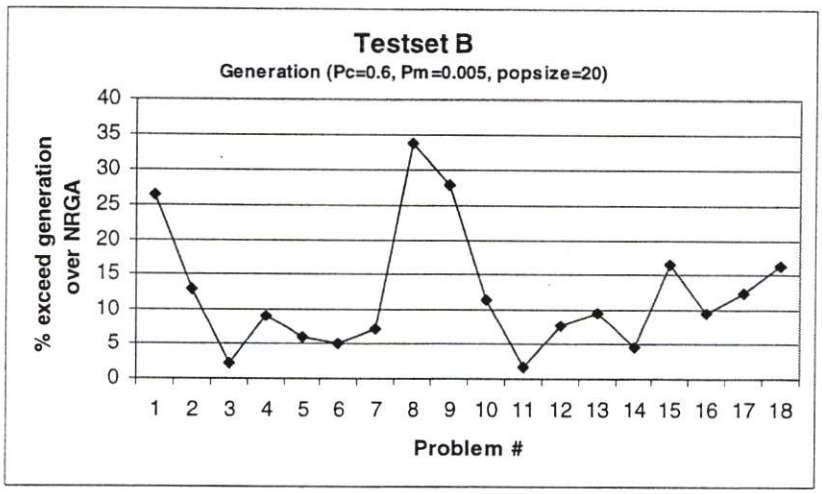
รูปที่ 5.2 โปรแกรม Multicast Routing Simulation ที่ใช้เครือข่ายใน Testset B (b09)

5.3 ผลการทดลอง

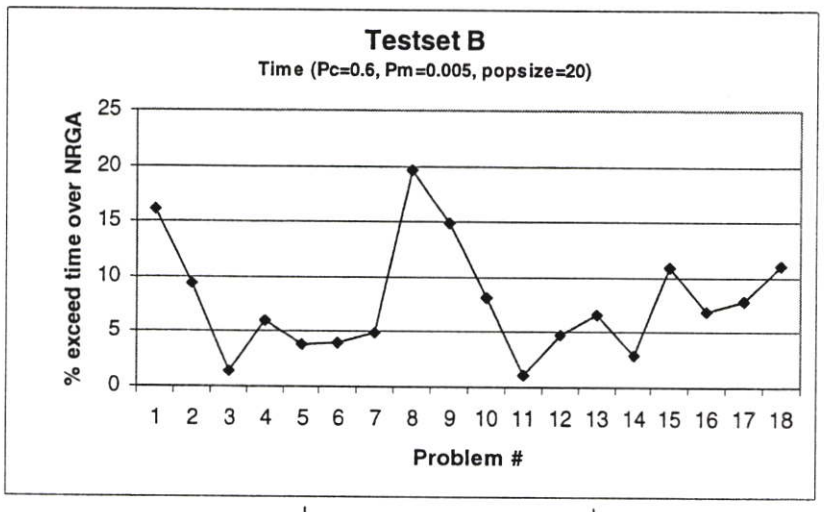
ในการทดลองครั้งนี้ได้แบ่งการทดลองออกเป็น 2 กลุ่มคือ กลุ่มที่ใช้ testset B (ตารางที่ 5.1) และ กลุ่มที่ใช้ testset C (ตารางที่ 5.2) โดยการทดลองกลุ่มแรกที่ใช้ testset B นั้นจะทดลองโดยเปลี่ยนพารามิเตอร์ต่างๆของ GA ส่วนกลุ่มที่ 2 นั้นจะใช้ค่าพารามิเตอร์ของ GA เพียงค่าเดียวในการทดลอง

5.3.1 การทดลองกับ Testset B

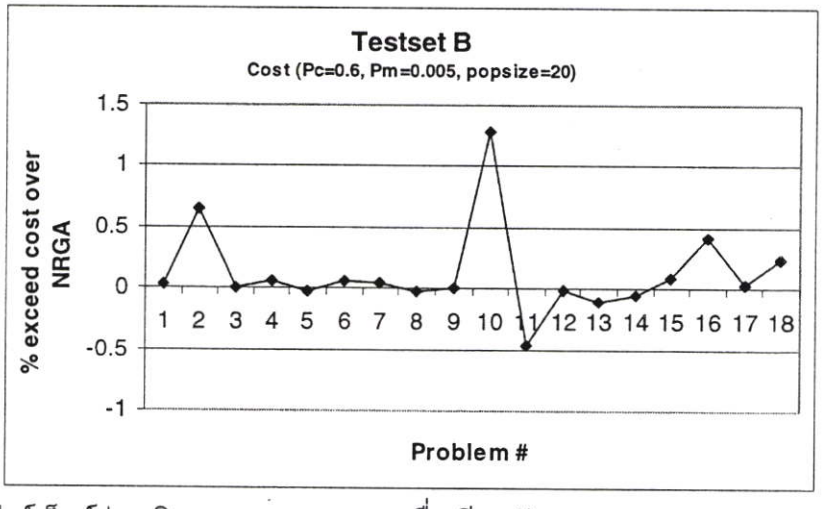
จากผลการทดลองแสดงให้เห็นว่า วิธีการที่เราได้นำเสนอไปนั้น สามารถเพิ่มประสิทธิภาพให้กับจีเนติกอัลกอริทึมได้ ทั้งในแง่ของจำนวนรุ่นของการเกิดที่น้อยลง และเวลาโดยเฉลี่ยที่น้อยลงด้วยเช่นกัน ดังแสดงในกราฟรูปที่ 5.3 และ 5.4 และกำหนดค่าพารามิเตอร์ของ GA เป็น Pc=0.6, Pm=0.005 และ popsize=20 โดยในรูปที่ 5.3 นั้นเป็นกราฟแสดงจำนวนรุ่นของการเกิดของจีเนติกอัลกอริทึมแบบที่ไม่ได้มีการจัดเรียงเครือข่ายใหม่ที่เกินมาจากแบบที่มีการจัดเรียงเครือข่ายใหม่ที่คิดเป็นเปอร์เซ็นต์ ส่วนในรูปที่ 5.4 เป็นกราฟเปรียบเทียบเวลาของจีเนติกอัลกอริทึมแบบที่ไม่ได้มีการจัดเรียงเครือข่ายใหม่ที่เกินมาจากแบบที่มีการจัดเรียงเครือข่ายใหม่ที่คิดเป็นเปอร์เซ็นต์ และกราฟรูปที่ 5.5 เป็นกราฟที่เปรียบเทียบ cost ของจีเนติกอัลกอริทึมแบบที่ไม่ได้มีการจัดเรียงเครือข่ายใหม่กับแบบที่มีการจัดเรียงเครือข่ายใหม่ที่คิดเป็นเปอร์เซ็นส่วนเกินเช่นกัน ซึ่งกราฟในรูปที่ 5.4 นั้นจะเห็นว่าจะมีบางเครือข่ายที่มีค่า cost ของ NRGA สูงกว่า GA แต่เมื่อคิดเป็นเปอร์เซ็นแล้วก็ถือว่าน้อยมากคือ ไม่ถึง 1% และค่า cost โดยรวมของ NRGA ก็ยังจัดได้ว่าดีกว่าแบบ GA จากกราฟทั้งสามสามารถสรุปได้ว่าการจัดเรียงเครือข่ายใหม่นั้นส่งผลให้การทำงานของจีเนติกอัลกอริทึมมีประสิทธิภาพมากขึ้นอย่างเห็นได้ชัด



รูปที่ 5.3 เปอร์เซ็นต์ส่วนเกินของ generation ของ GA เมื่อเทียบกับ NRGA

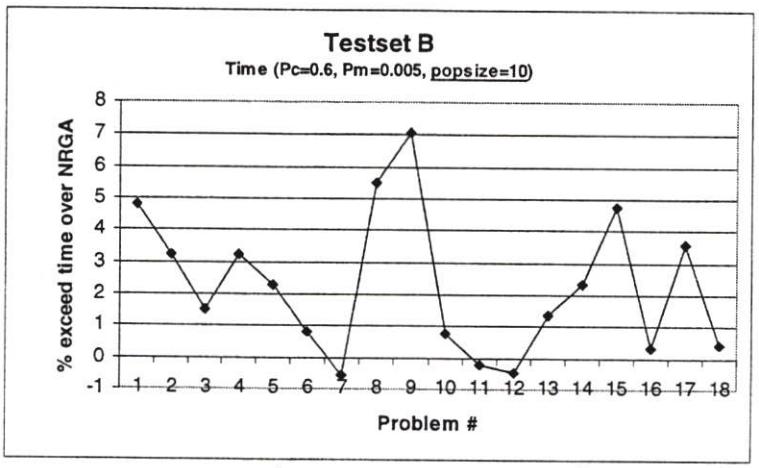


รูปที่ 5.4 เปอร์เซ็นต์ส่วนเกินของเวลาที่ใช้ประมวลผลของ GA เมื่อเทียบกับ NRGA

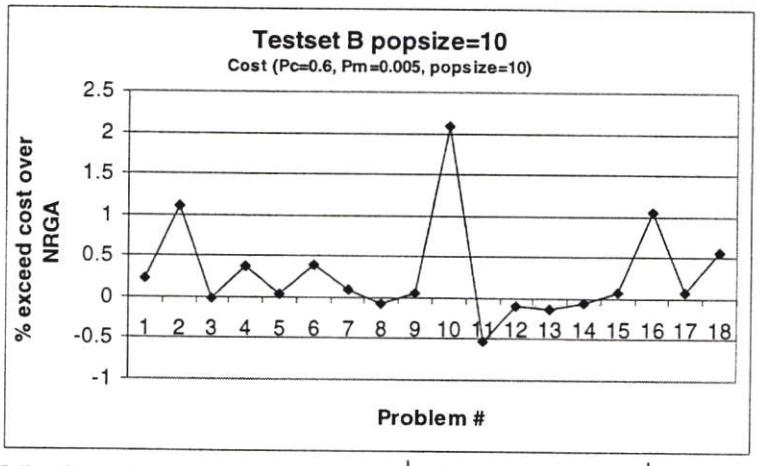


รูปที่ 5.5 เปอร์เซ็นต์ส่วนเกินของ cost ของ GA เมื่อเทียบกับ NRGA

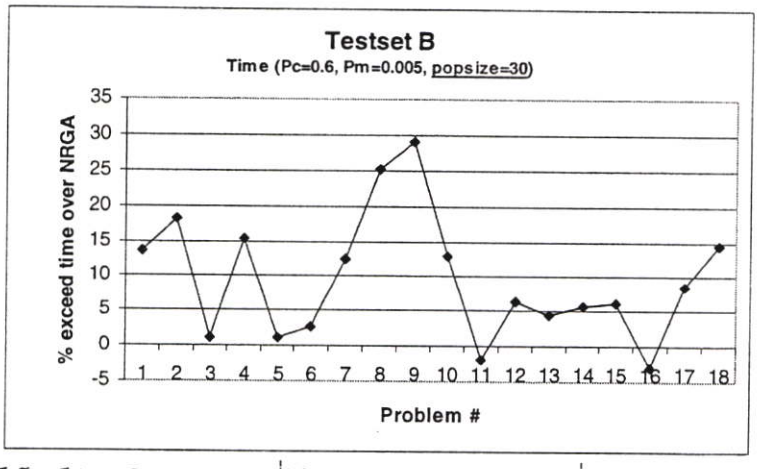
รูปที่ 5.6, 5.7, 5.8, 5.9, 5.10 และ 5.11 เป็นรูปที่แสดงเปอร์เซ็นต์ส่วนเกินของเวลาและ cost ที่ GA ใช้มากกว่า NRGA โดยใช้จำนวนของประชากรขนาด 10, 30 และ 50 ตามลำดับ ในรูปที่ 5.6 และ 5.7 เป็นกราฟของเวลาและ cost ซึ่งจะเห็นว่า เมื่อเราใช้จำนวนประชากรเท่ากับ 10 ผลลัพธ์ที่ได้โดยรวมแล้ว NRGA จะดีกว่า GA แบบเดิม คือ เวลาที่ใช้จะน้อยกว่าโดยมีค่าสูงสุดอยู่ที่ประมาณ 7% ส่วนค่า cost ที่ได้นั้น ถึงแม้ว่าจะมีบางเครือข่ายที่ NRGA มีค่า cost สูงกว่าก็ตามแต่ก็มีค่าอยู่ประมาณไม่เกิน 0.55% เมื่อเพิ่มจำนวนประชากรให้มีขนาดใหญ่ขึ้นเป็น 30 และ 50 ดังรูปที่ 5.8, 5.9, 5.10 และ 5.11 จะเห็นว่าเปอร์เซ็นต์ความต่างของเวลาจะมีมากขึ้น และเปอร์เซ็นต์ความต่างของ cost จะมีน้อยลง



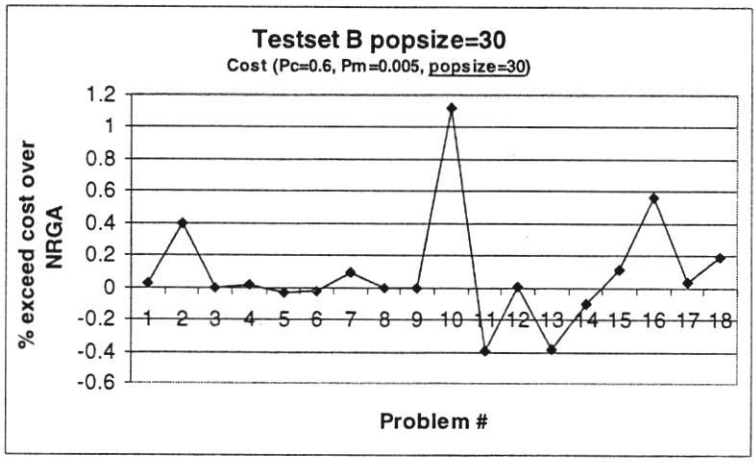
รูปที่ 5.6 เปอร์เซ็นต์ส่วนเกินของเวลาที่ใช้ประมวลผลของ GA เมื่อเทียบกับ NRGA เมื่อจำนวนประชากรขนาด 10 ประชากร



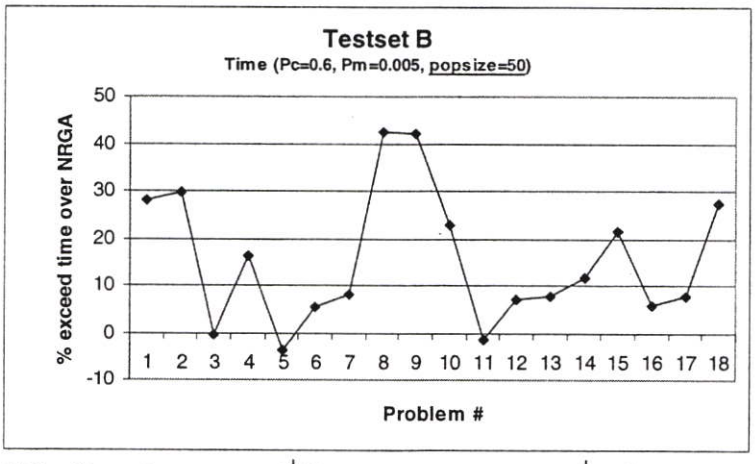
รูปที่ 5.7 เปอร์เซ็นต์ส่วนเกินของ cost ของ GA เมื่อเทียบกับ NRGA เมื่อจำนวนประชากรขนาด 10 ประชากร



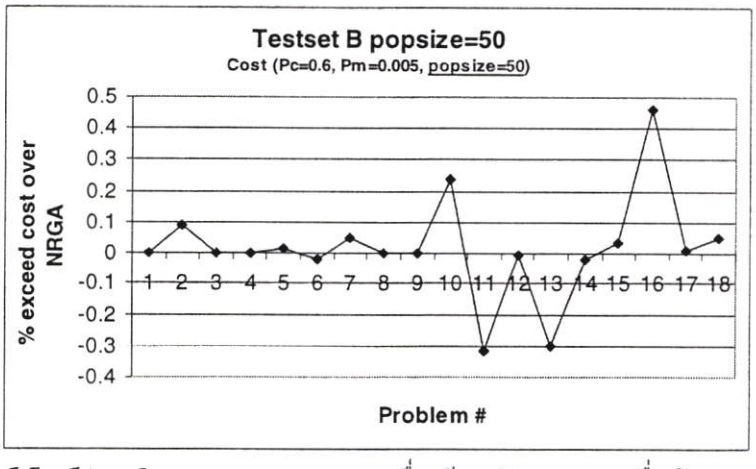
รูปที่ 5.8 เปอร์เซ็นต์ส่วนเกินของเวลาที่ใช้ประมวลผลของ GA เมื่อเทียบกับ NRGA เมื่อจำนวนประชากรขนาด 30 ประชากร



รูปที่ 5.9 เปอร์เซ็นต์ส่วนเกินของ cost ของ GA เมื่อเทียบกับ NRGA เมื่อจำนวนประชากรขนาด 30 ประชากร

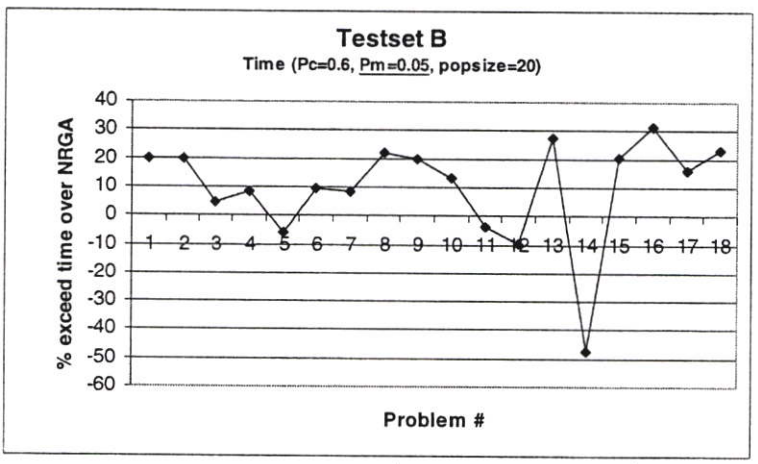


รูปที่ 5.10 เปอร์เซ็นต์ส่วนเกินของเวลาที่ใช้ประมวลผลของ GA เมื่อเทียบกับ NRGA เมื่อจำนวนประชากรขนาด 50 ประชากร

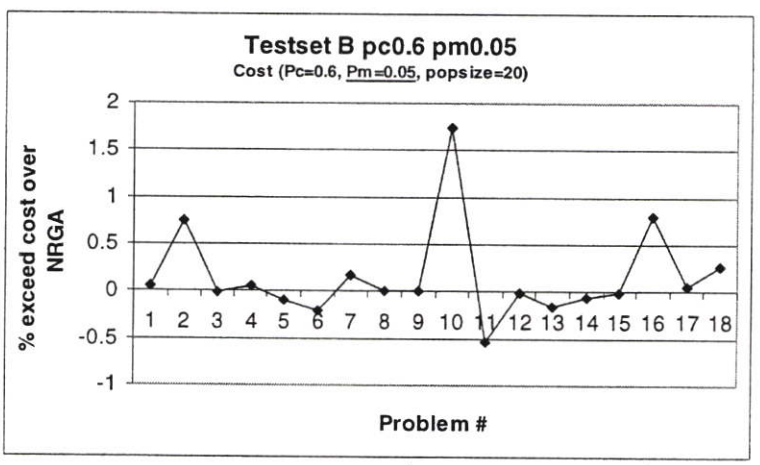


รูปที่ 5.11 เปอร์เซ็นต์ส่วนเกินของ cost ของ GA เมื่อเทียบกับ NRGA เมื่อจำนวนประชากรขนาด 50 ประชากร

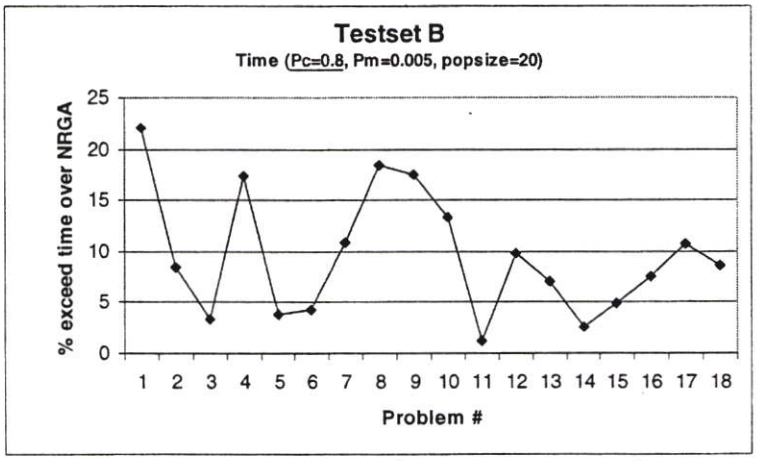
รูปที่ 5.12-5.17 เป็นกราฟแสดงการเปลี่ยนแปลงพารามิเตอร์ต่างๆของ GA ซึ่งจะเห็นว่าถึงแม้ว่าเราจะเปลี่ยนแปลงพารามิเตอร์ไม่ว่าจะเป็น Pc หรือ Pm ก็ตาม NRGA ก็ยังให้ผลลัพธ์ที่ดีกว่า GA เสมอ



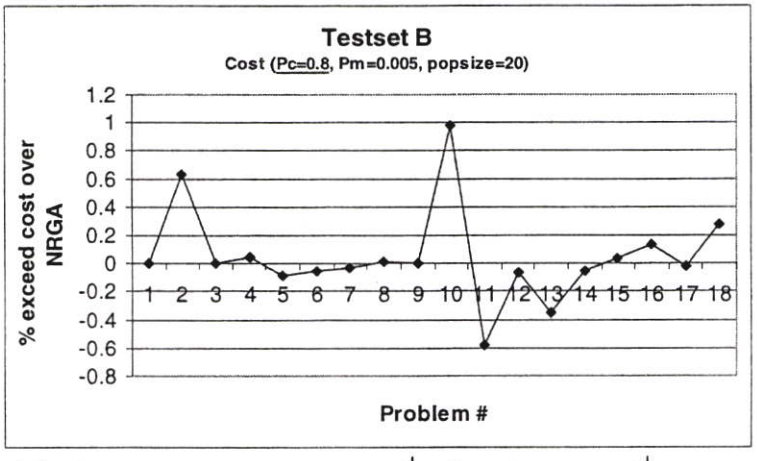
รูปที่ 5.12 เปอร์เซ็นต์ส่วนเกินของเวลาของ GA เมื่อเทียบกับ NRGA เมื่อ Pm=0.05



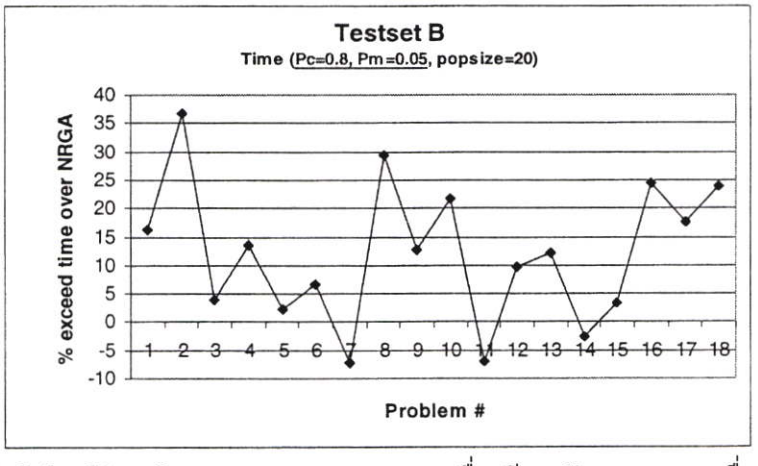
รูปที่ 5.13 เปอร์เซ็นต์ส่วนเกินของ cost ของ GA เมื่อเทียบกับ NRGA เมื่อ Pm=0.05



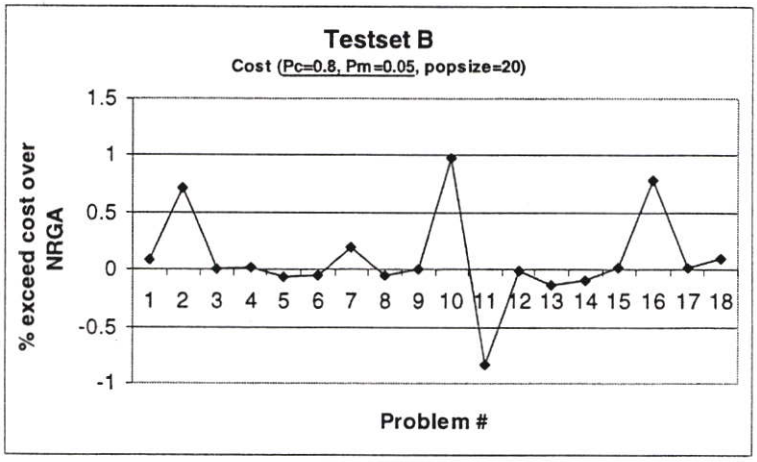
รูปที่ 5.14 เปอร์เซ็นต์ส่วนเกินของเวลาของ GA เมื่อเทียบกับ NRGA เมื่อ Pc=0.08



รูปที่ 5.15 เปอร์เซ็นต์ส่วนเกินของ cost ของ GA เมื่อเทียบกับ NRGA เมื่อ Pc=0.08



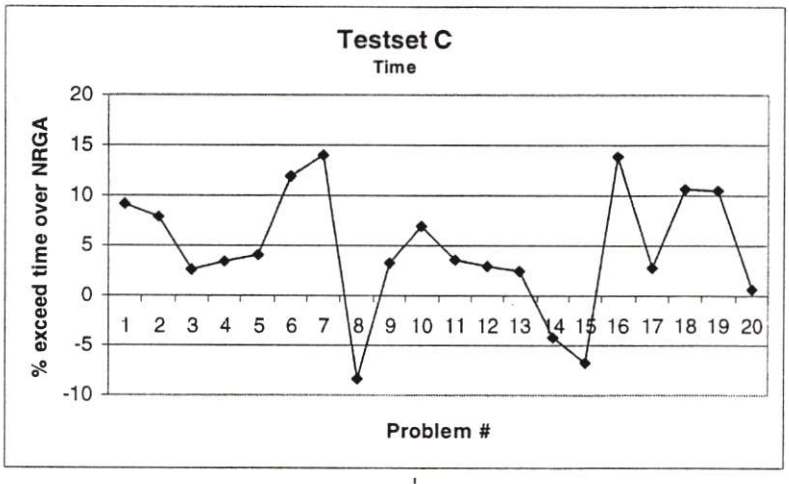
รูปที่ 5.16 เปอร์เซ็นต์ส่วนเกินของเวลาของ GA เมื่อเทียบกับ NRGA เมื่อ Pc=0.08 และ Pm=0.05



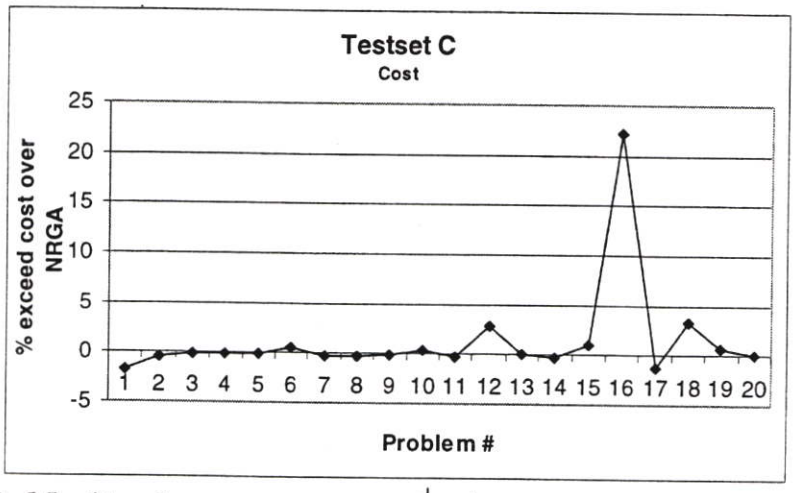
รูปที่ 5.17 เปอร์เซ็นต์ส่วนเกินของ cost ของ GA เมื่อเทียบกับ NRGA เมื่อ Pc=0.08

5.3.2 **การทดลองกับ** Testset C

การทดลองกลุ่มนี้จะใช้เครือข่ายด้วยกันทั้งหมด 20 เครือข่าย โดยใช้ Pc=0.6, Pm=0.005 และ popsize=20 ซึ่งได้ผลลัพธ์ดังรูปที่ 5.18 และ รูปที่ 5.19 ซึ่งจะเห็นว่า NRGA ก็ยังสร้างผลลัพธ์ได้ดีกว่า GA เช่นเดิม



รูปที่ 5.18 เปอร์เซ็นต์ส่วนเกินของเวลาของ GA เมื่อเทียบกับ NRGA



รูปที่ 5.19 เปอร์เซ็นต์ส่วนเกินของ cost ของ GA เมื่อเทียบกับ NRGA

เอกสารอ้างอิง

[1] Vachaspathi Peter Kompella, "Multicast Routing Algorithms for Multimedia Traffic", University of California, San Diego.

[2] Q. Sun and H. Langendorfer, "Efficient Multicast Routing for Delay-Sensitive Applications", in Proceedings of 2nd international Workshop on Protocols for Multimedia Systems (PROMS'95)

[3] Q. Sun and H. Langendorfer, "An Efficient Delay-Constrained Multicast Routing Algorithm", *Journal of High-Speed Networks*, vol. 7, no. 1, pp.43-55, 1998.

[4] David E. Goldberg, "Genetic Algorithms in Search, Optimization, and Machine Learning", *Addison-Wesley Publishing Company, Inc.* 1989

[5] Quan Sun. "A Genetic Algorithm for Delay-Constrained Minimum-Cost Multicasting", Technical Report, IBR, TU Braunschweig, Bueltenweg 74/75, 38106 Braunschweig, Germany, January 1999.

[6] กาญจนี วงศ์วิภาภร, "การจัดตารางสอนของโรงเรียนแบบอัตโนมัติโดยจีเนติกอัลกอริทึม", *สถาบันเทคโนโลยีพระจอมเล้าเจ้าคุณทหารลาดกระบัง ISBN 974-622-126-4*, พ.ศ. 2541

[7] J. E. Beasley, "OR-Library: Distributing Test Problems by Electronic Mail", *Journal of the Operational Research Society*, Vol.41, No.11, pp.1069-1072, 1990.

[8] Yee Leung, Guo Li, and Zong-Ben Xu, "A Genetic Algorithm for the Multiple Destination Routing Problems", *IEEE Transactions on Evolutionary Computation*, Vol. 2, No. 4, November 1998

[9] Bryant A. Julstrom, "A Genetic Algorithm for the Rectilinear Steiner Problem", Department of Computer Science, St. Cloud State University.

[10] M. Dror, M. Haouari, J. Chaouachi, "Generalized spanning trees", *European Journal of Operational Research*, Vol. 120, pp. 583-592, 2000.

[11] David K. Smith, Godfrey A. Walters, "An evolutionary approach for finding optimal trees in undirected networks", *European Journal of Operational Research*, Vol. 120, pp. 593-602, 2000.

[12] Serasate Serajatupat, Chotipat Pornavalai, Goutam Chakraborty, "Genetic Algorithm with Network Reordering for Multicast Networks", The 2001 International

Technical Conference on Circuits/Systems, Computers and Comunications, Volume II, pp.1067-1070, July 10-12, 2001.

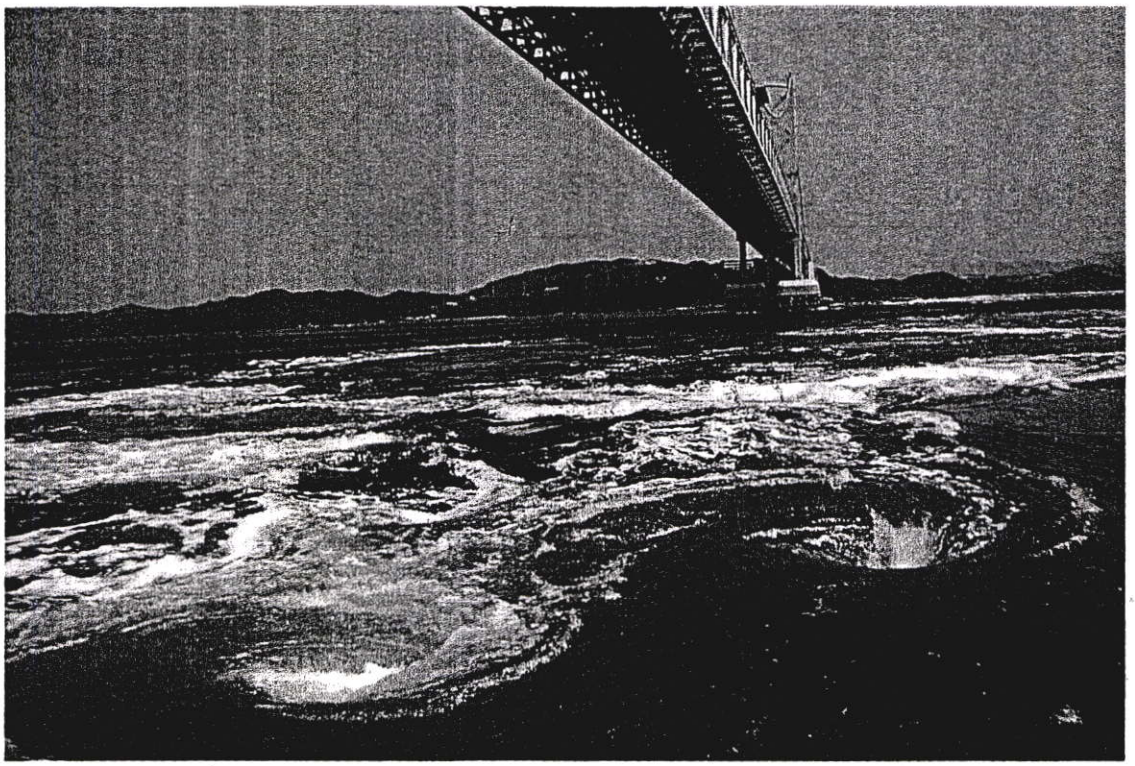
ภาคผนวก

ผลงานวิจัยที่ได้รับการตีพิมพ์

Proceedings

The 2001 International Technical Conference on Circuits/Systems, Computers and Communications

Volume II



July 10-12, 2001
Hotel Clement Tokushima, Tokushima, Japan

Sponsored by
The Institute of Electronics, Information and Communication Engineers,
Research Society of System and Signal Processing
The Institute of Electronics Engineering of Korea
In Cooperation with
The Institute of Electrical Engineers, Shikoku Branch
The Institute of Electronics, Information and Communication Engineers, Shikoku Branch
Corporate sponsors
The Telecommunication Advancement Foundation (TAF)
NTT DoCoMo Shikoku, Inc.
International Communications Foundation (ICF)
Support Center for Advanced Telecommunications Technology Research (SCAT)
Kayamori Foundation of Informational Science Advancement
The University of Tokushima

PROCEEDINGS

The 2001 International Technical Conference on Circuits/Systems, Computers and Communications

Volume II

July 10-12, 2001
Hotel Clement Tokushima, Tokushima, Japan

Sponsored by
The Institute of Electronics, Information and Communication Engineers,
Research Society of System and Signal Processing
The Institute of Electronics Engineering of Korea
In Cooperation with
The Institute of Electrical Engineers, Shikoku Branch
The Institute of Electronics, Information and Communication Engineers, Shikoku Branch
Corporate sponsors
The Telecommunication Advancement Foundation (TAF)
NTT DoCoMo Shikoku, Inc.
International Communications Foundation (ICF)
Support Center for Advanced Telecommunications Technology Research (SCAT)
Kayamori Foundation of Informational Science Advancement
The University of Tokushima

Genetic Algorithm with Network Reordering for Multicast Networks

Serasate Serajatupat* Chotipat Pornavalai* Goutam Chakraborty**

*Faculty of Information Technology,
King Mongkut's Institue of Technology,
Ladkrabang, Bangkok 10520 Thailand
Email: sirasate@mut.ac.th, chotipat@it.kmitl.ac.th

**Faculty of Software and Information Science,
Iwate Prefectural University
Takizawa, Iwate, 020-0193, Japan
Email: goutam@soft.iwate-pu.ac.jp

Abstract: With the increase demands in multimedia applications, the multicast support from the communication networks is required. Network has to find the multicast route (tree) that minimizes the resource usage by using the minimum cost tree. There are many existing algorithms such as heuristic, Neural Network, Genetic algorithm to solve this problem, which is known to be a NP-Complete. In this paper, we presents a new method for improve performance of multicast routing by reordering network before sending to genetic algorithm. The proposed genetic algorithm can find the solution with less generation and computation time without effecting the multicast tree cost. Our results show that the proposed algorithm can find very near optimum tree cost with acceptable computation time than any existing algorithm.

1. Introduction

Multicasting is simultaneous delivery of data distribution from a sender to several receivers in form of tree, which called multicast tree. The tree will be used as paths for data distribution to multiple destinations, for example persons who listen to the same radio station will be receive the same data. To find the optimal route of multicast tree causes so many computation time because this kind of problem is known to be a NP-Complete. At present, ones of the efficient heuristic algorithms are CRA and CKMB which are proposed in [4, 5]. Both algorithms can find multicast tree which use delay tolerance of data transmission from source to destinations but the result tree cost is still a bit higher than the optimal one. Recently, genetic algorithms are proposed for this problem such as [2, 6]. Their results show that genetic can find very near-optimal tree with acceptable computation time than any other existing solutions.

However in their genetic algorithm, the arrangements of the chromosome or the positions of the reference node in the genetic algorithm is chosen as random values. In this paper we propose to adjust order in the chromosome by rearranging the number of network nodes in chromosome using Prim Spanning Tree before starting the genetic algorithm. This reorder method is done to make the nodes that have near distances also have near node numbers, and nodes which are far away have far away node number.

By this way we can reduce computation time of genetic algorithm, at the same time, the cost of the multicast tree is still the same, and many cases are a little bit lower.

2. Theory and related algorithms

2.1. Constrained Routing (CRA) and CKMB Algorithms

CRA is an algorithm, which is used to find route of data distribution through multiple destinations nodes. It tries to compute a low cost tree rooted at source node and spanning some (or possibly all) of the destination nodes subject to the delay bound constraint if one exists. However in some cases, CRA cannot distribute the data to the destinations although there is route that can be reached and satisfy the delay bound. CKMB is developed from CRA. It can find this route. The performance of CKMB is known to be one of the best heuristic algorithms for this problem. The detail can be found in [3, 4].

2.2. Genetic Algorithm (GA)

Genetic algorithms are search algorithms [2] based on the mechanisms of genetic adaptation in biological systems, by imitating the nature of gene's of chromosome and exchange the data of chromosome, for example crossover and mutation. The quality of an individual is judged by its fitness measure.

2.2.1 Multicast chromosome structure

To form of multicast routing, Chromosome fixes which Steiner node will be used by the position of chromosome from left to right which mean the number of Steiner Node in order of distance. The Steiner nodes are the nodes in the multicast tree except the source and destinations. The genetic algorithm is used to find the Steiner nodes that will give the near-optimal multicast tree.

Referring the example network in Figure 1, the chromosome, as shown in Figure 2, in position 0,1,2 and 3 refers to node 1, 4, 6 and 8 respectively. The number inside chromosome indicates which node will be used. For example, if its value is 1, the node at that position will be selected as the Steiner node. By this way, it adds more choice to CKMB, because the main algorithm that find the route for GA still be the CKMB.

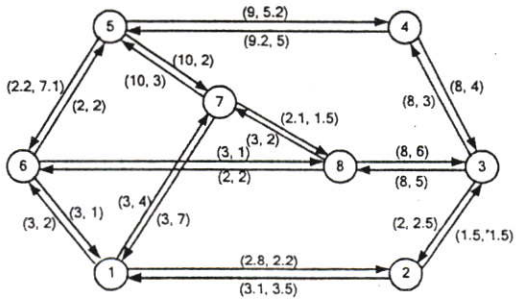
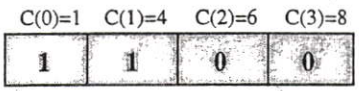


Figure 1. Example Network



Source = 7
Destinations = 2, 3, 5
Steiner nodes = 1, 4

Figure 2. Chromosome structure

2.2.2. Network reorderation

In this paper, we propose to number the respective number of network by letting nearly nodes have nearly number and letting far away nodes have far away number respectively. The algorithm to reorder this chromosome is similar to run Prim spanning tree to numbering the node numbers.

After we rearrange the number of network and send it to GA. The chromosome of GA will have position of Steiner node in chromosome order by distance or cost. The procedure is described by example as shown in Figure 3. After we initialize the chromosome, the similar GA algorithm used in [1, 5] can be used to find Steiner nodes. Each GA is then used CKMB to find the multicast tree.

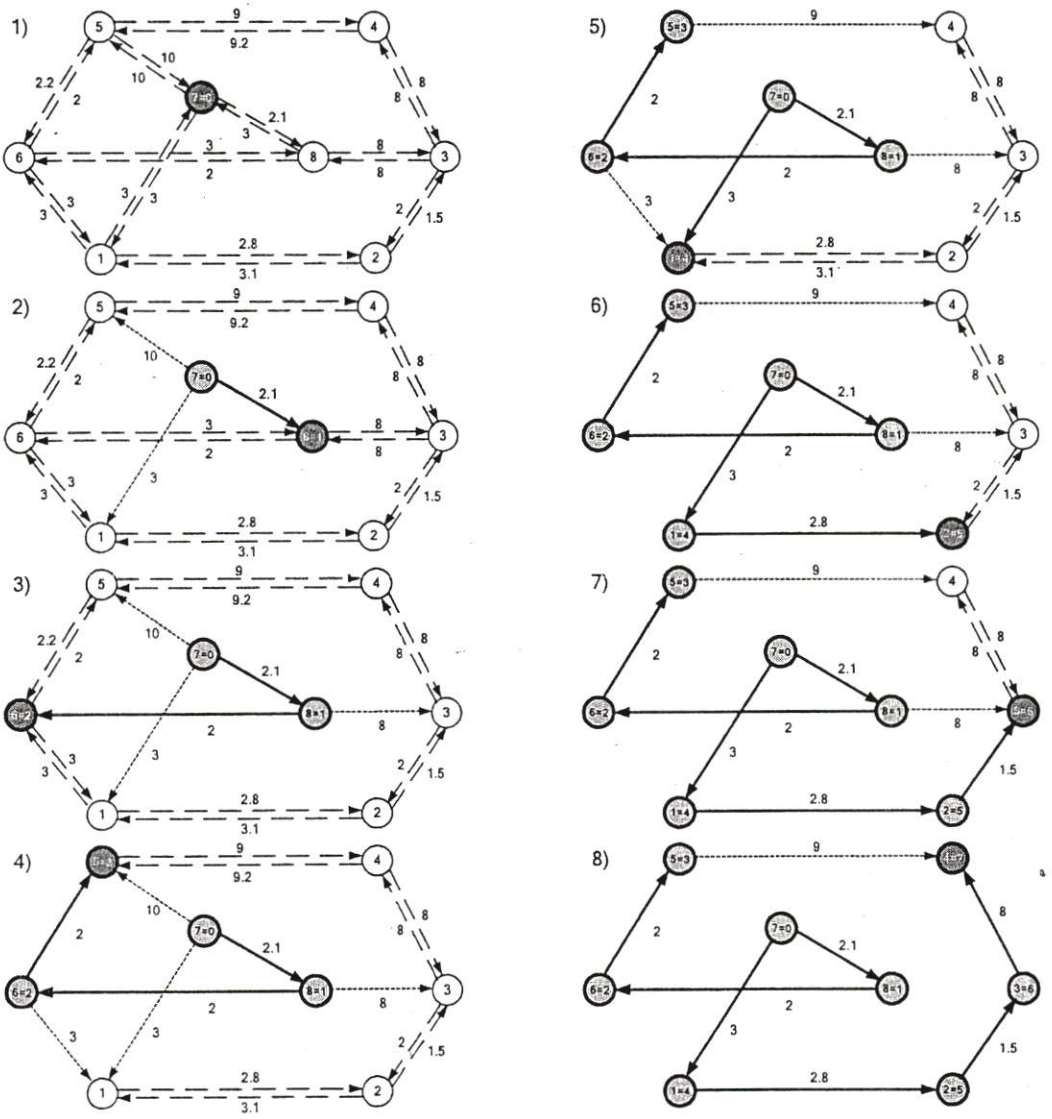


Figure 3. Network reorderation

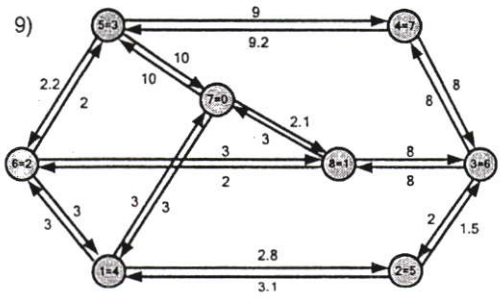


Figure 4. Reordered Network

In Figure 3 can be explain by the following step

Step 1: Let source node 7 be the first node in reorder network (in this implementation the first node is 0) so node 7 convert to node 0 (7=0)

Step 2: At the source node(7=0), it has 3 links to other nodes, link to node 8 with cost equal to 2.1, link to node 1 with cost equal to 3, And link to node 5 with cost equal to 10. We choose node that has link from source node with minimum cost. So node 8 was chosen and convert to node 1 (8=1). Node 1 is the next number of reorder network node.

Step 3: We consider total link of node 7=0 and 8=1 that have not been chosen yet. Those links are cost 3 to node 1, cost 8 to node 3, cost 10 to node 5 and cost 2 to node 6 respectively. The minimum link cost will be chosen. So node 6 was chosen and convert to node 2(6=2). Node 2 is the next number of reorder network node.

Step 4-8: Use the same method as in step 3 until all node have been used.

Figure 4 shows the network that has been reordered. If source and destination nodes in original network are 7 and 2, 3, 5 respectively then source and destination nodes in reorder network should be 0 and 5, 6, 3 respectively. The Steiner nodes of the original network are 1, 4, 6, 8 And the Steiner node of the reorder network are 1, 2, 4, 7. The chromosome of original and reorder network with c(1) and c(2) is add as extend destination nodes can see in figure 5.

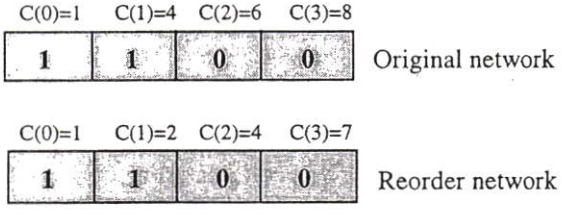


Figure 5. Original vs. reorder chromosome

3. The simulation results

We use the network test set B from Or-library [6] with the network size of 50, 75 and 100 nodes as shown in table 1 where $|V|$ is the number of network node, $|E|$ is the number of edge, and $|D|$ is the total multicast destinations.

The simulation results show that our proposed algorithm can improve performance of traditional GA with less computation time and less number of generations as shown in Figure 5 and 6. The cost of multicast tree is also almost the same as original one as in Figure 7 and 8.

4. Conclusions

In this paper, we presents a new method for improve performance of multicast routing by reordering network before sending to genetic algorithm. The proposed genetic algorithm can find the solution with less generation and computation time without effecting the multicast tree cost. Our results show that the proposed algorithm can find near-optimum tree cost with acceptable computation time than any other existing algorithms.

Table 1. Network test set B detail (Or-library)

Problem no.	$\|V\|$	$\|E\|$	$\|D\|$
1	50	63	9
2	50	63	13
3	50	63	25
4	50	100	9
5	50	100	13
6	50	100	25
7	75	94	13
8	75	94	19
9	75	94	38
10	75	150	13
11	75	150	19
12	75	150	38
13	100	125	17
14	100	125	25
15	100	125	50
16	100	200	17
17	100	200	25
18	100	200	50

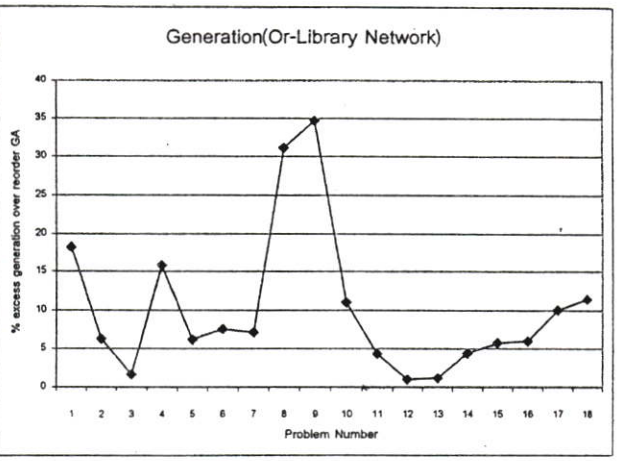


Figure 5: Percentage excess number of generations of traditional GA over the proposed reordering GA

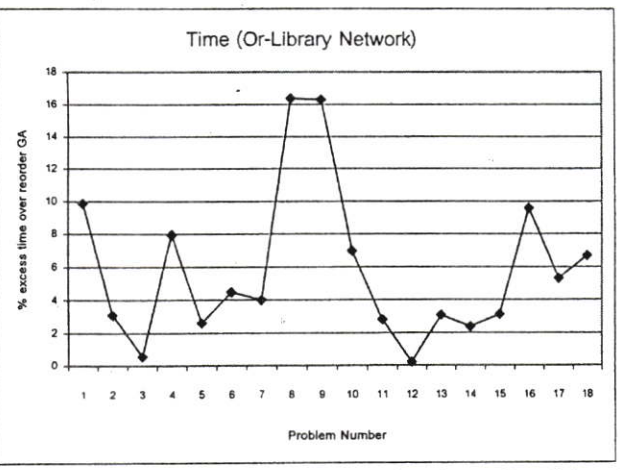


Figure 6: Percentage excess computation time of traditional GA over the proposed reordering GA

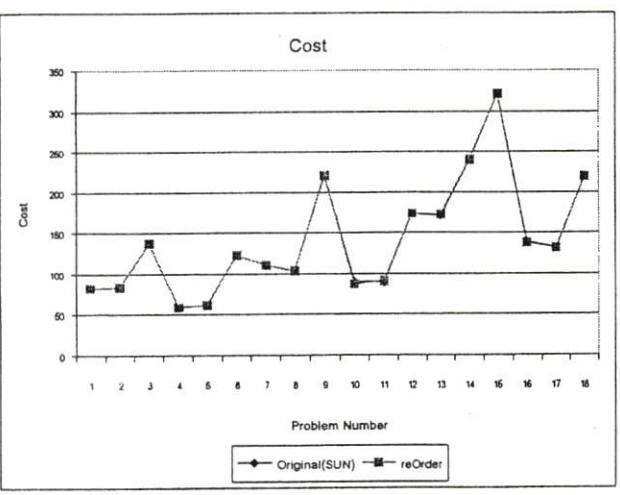


Figure 7: Cost comparison

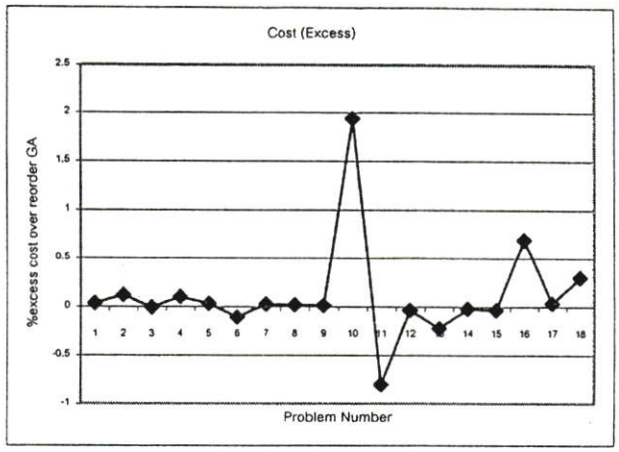


Figure 7. Percentage excess cost of traditional GA over the proposed reordering GA

6. Reference

[1]. Quan Sun. "A Genetic Algorithm for Delay-Constrained Minimum-Cost Multicasting", Technical Report, IBR, TU Braunschweig, Bueltenweg 74/75, 38106 Braunschweig, Germany, January 1999.

[2]. David E. Goldberg, "Genetic Algorithms in Search, Optimization, and Machine Learning", *Addison-Wesley Publishing Company, Inc.* 1989

[3]. Q. Sun and H. Langendorfer, "An Efficient Delay-Constrained Multicast Routing Algorithm", *Journal of High-Speed Networks*, vol. 7, no. 1, pp.43-55, 1998.

[4]. Q. Sun and H. Langendorfer, "Efficient Multicast Routing for Delay-Sensitive Applications", in Proceedings of 2nd international Workshop on Protocols for Multimedia Systems (PROMS'95)

[5]. Yee Leung, Guo Li, and Zong-Ben Xu, "A Genetic Algorithm for the Multiple Destination Routing Problems", *IEEE Transactions on Evolutionary Computation*, Vol. 2, No. 4, November 1998

[6]. J. E. Beasley, "OR-Library: Distributing Test Problems by Electronic Mail", *Journal of the Operational Research Society*, Vol.41, No.11, pp.1069-1072, 1990.

[7]. Bryant A. Julstrom, "A Genetic Algorithm for the Rectilinear Steiner Problem", Department of Computer Science, St. Cloud State University.

[8]. M. Dror, M. Haouari, J. Chaouachi, "Generalized spanning trees", *European Journal of Operational Research*, Vol. 120, pp. 583-592, 2000.

[9]. Vachaspathi Peter Kompella, "Multicast Routing Algorithms for Multimedia Traffic", University of California, San Diego.

[10]. David K. Smith, Godfrey A. Walters, "An evolutionary approach for finding optimal trees in undirected networks", *European Journal of Operational Research*, Vol. 120, pp. 593-602, 2000.

ประวัติผู้เขียน

ชื่อ นาย ศิรเศรษฐ์ ศิรจตุพัฒน์ เกิดวันที่ 20 กรกฎาคม พ.ศ. 2516 ที่จังหวัดสุรินทร์ การศึกษา วิศวกรรมศาสตรบัณฑิต (วิศวกรรมไฟฟ้า-คอมพิวเตอร์) มหาวิทยาลัยเทคโนโลยีมหานคร

ปี พ.ศ. 2541-2542 เข้าทำงานที่ มหาวิทยาลัยเทคโนโลยีมหานคร ตำแหน่งผู้ช่วยอาจารย์ คณะวิศวกรรมไฟฟ้า-คอมพิวเตอร์

ปี พ.ศ. 2543-2544 เข้าทำงานที่ มหาวิทยาลัยเทคโนโลยีมหานคร ตำแหน่งอาจารย์ประจำ คณะวิศวกรรมไฟฟ้า-คอมพิวเตอร์

ปี พ.ศ. 2545 เข้าทำงานที่ คณะเทคโนโลยีสารสนเทศ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ตำแหน่งเจ้าหน้าที่ฝ่ายสนับสนุนเทคโนโลยี