บทที่ 4

การวิเคราะห์พารามิเตอร์ต่าง ๆ ที่ใช้ในฮิวริสติกอัลกอริทึม

4.1 กล่าวนำ

ในบทนี้จะวิเคราะห์พารามิเตอร์ต่าง ๆ ที่ใช้ในฮิวริสติกอัลกอริทึมเพื่อการปรับปรุงการออกแบบให้ดีขึ้น พารามิเตอร์ที่จะศึกษาเป็นพารามิเตอร์ที่ต้องกำหนดโดยผู้ออกแบบโครงข่ายและส่งผลต่อกระบวนการปรับปรุงการออกแบบ อีกนัยหนึ่ง คือ ส่งผลต่อต้นทุนของโครงข่ายที่ได้ออกแบบนั่นเอง ดังนั้นในบทนี้ การวิเคราะห์ส่วนแรกจึงมุ่งเน้นไปที่การวิเคราะห์พารามิเตอร์เฉพาะในฮิวริสติกอัลกอริทึมแบบ Simulated Annealing และแบบ Tabu Search ทั้ง 2 แบบ

การวิเคราะห์ในส่วนถัดมา เป็นการเปรียบเทียบต้นทุนที่ได้จากการออกแบบโครงข่ายระหว่างฮิวริสติกอัลกอริทึมทั้งแบบ Local Search, Simulated Annealing, Tabu Search และ Tabu Search ที่ประยุกต์ร่วมกับแบบ Local Search เพื่อพิจารณาหาฮิวริสติกอัลกอริทึมที่เหมาะสมกับการออกแบบโครงข่ายแบบหนึ่ง ๆ มากที่สุด

4.2 การวิเคราะห์ค่าพารามิเตอร์ที่ใช้ใน Simulated Annealing

ในอัลกอริทึมการปรับปรุงการออกแบบโดยใช้ Simulated Annealing ตัวแปรที่สำคัญในการตัดสินใจเพื่อยอมรับคำตอบที่ด้อย ได้แก่ ค่าความน่าจะเป็นที่จะยอมรับคำตอบที่ด้อย (P) ซึ่งจะถูกคำนวณในแต่ละรอบจากต้นทุนที่เปลี่ยนไปและค่าอุณหภูมิ (t_i) ดังสมการที่ 3.1

$$P = exp\left(- \frac{\left| Cost_{i+1} - Cost_i \right|}{t_i} \right) \qquad (3.1)$$

ค่าต้นทุนที่เปลี่ยนไปจะขึ้นกับกระบวนการค้นหาคำตอบ ส่วนค่าอุณหภูมินี้จะถูกลดค่าลงทุก ๆ r ครั้งที่มีการยอมรับคำตอบ ด้วยอัตราส่วนค่า *factor*

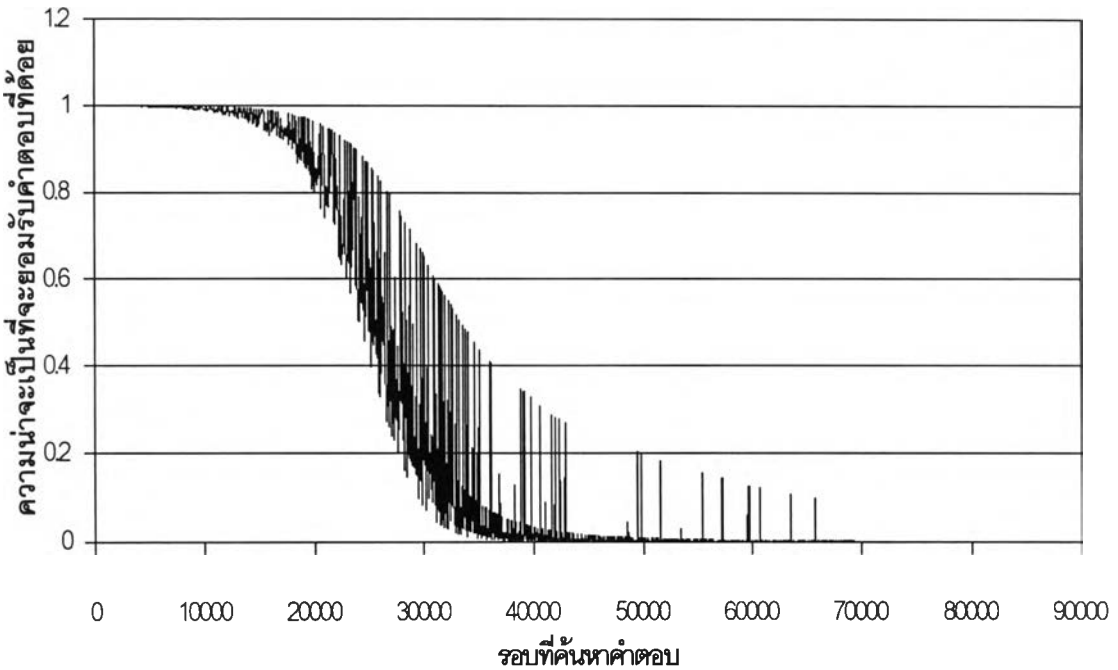
ดังนั้นค่าพารามิเตอร์ที่สำคัญในการกำหนดค่าความน่าจะเป็นที่จะยอมรับคำตอบที่ด้อยจึงมี 3 ค่า ได้แก่ ค่าอุณหภูมิเริ่มต้น (t_0), ค่า *factor* และค่า r การกำหนดค่าพารามิเตอร์ทั้ง 3 ค่านี้ต้องให้อัลกอริทึมสามารถค้นพบคำตอบที่มีค่าต้นทุนต่ำภายในเวลาที่เหมาะสม แต่เนื่องจากไม่มี

ทฤษฎีใด ๆ บ่งชี้ว่าควรใช้ค่าพารามิเตอร์อย่างไรจึงจะดีสำหรับโครงข่ายหนึ่ง ๆ ดังนั้นจึงต้องทดสอบโดยใช้ค่าพารามิเตอร์หลาย ๆ แบบ เพื่อหาค่าที่เหมาะสมกับโครงข่ายหนึ่ง ๆ มากที่สุด

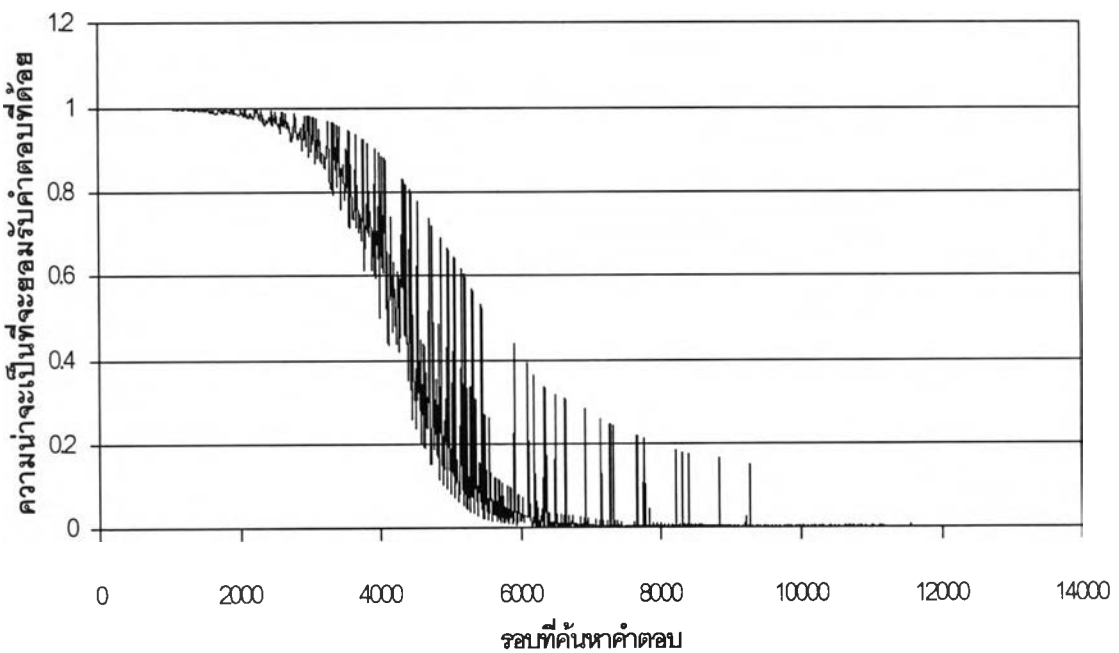
4.2.1 การวิเคราะห์ค่า *factor* และค่า *r*

ในหัวข้อนี้จะวิเคราะห์ค่า *factor* และค่า *r* ซึ่งเป็นพารามิเตอร์ที่ส่งผลต่อการลดลงของค่าอุณหภูมิโดยตรง จากสมการที่ (3.1) และ (3.2) ในบทที่ 3 จะเห็นว่าค่า *factor* และค่า *r* ที่น้อยจะทำให้อุณหภูมิลดลงอย่างรวดเร็ว จึงส่งผลให้ค่าความน่าจะเป็นที่จะยอมรับคำตอบที่ด้อยลดลงรวดเร็วกว่าค่า *factor* ที่มากและค่า *r* ที่มาก รูปที่ 4.1 (ก) - (ง) แสดงให้เห็นลักษณะการลดลงของค่าความน่าจะเป็นที่จะยอมรับคำตอบที่ด้อย ตามค่า *r* ในการปรับปรุงการออกแบบ เมื่อใช้ค่า *factor* และค่า *r* ที่จะลดอุณหภูมิต่าง ๆ กัน คือ (ค่า *factor* , ค่า *r*) เท่ากับ (0.99,30), (0.99,5), (0.5,30) และ (0.5,5) ดังแสดงในรูปที่ 4.1 (ก), (ข), (ค) และ (ง) ตามลำดับ การทดสอบทำบนโครงข่าย EUROCore โดยใช้ทราฟฟิกแบบ Uniform ที่มีขนาดเป็น 1, M=1 และกำหนดให้ค่าอุณหภูมิเริ่มต้น (t_0) เป็น 10000

รูปที่ 4.1 (ก) แสดงความน่าจะเป็นที่จะยอมรับคำตอบที่ด้อย เมื่อใช้ค่า *factor* เท่ากับ 0.99 และค่า *r* เป็น 30 ในช่วง 90,000 รอบของการค้นหาคำตอบ ในช่วงรอบแรก ๆ ของการค้นหาคำตอบ ความน่าจะเป็นมีการลดลงไม่มากนักและมีค่าใกล้เคียง 1 ในช่วงถัดมา คือ รอบที่ 20,000-40,000 ของการค้นหาคำตอบ ค่าความน่าจะเป็นจะมีการแกว่งตัวมากและมีแนวโน้มลดลงอย่างรวดเร็วเข้าหาค่า 0 แต่เมื่อเข้าสู่รอบที่ 40,000 เป็นต้นไป ความน่าจะเป็นจะค่อย ๆ ลดลงอย่างช้า ๆ เข้าหาค่า 0 และการแกว่งตัวของค่าจะลดลงจนประมาณรอบที่ 70,000 เป็นต้นไป ความน่าเป็นจะมีค่าน้อยมากจนถือได้ว่าเป็น 0

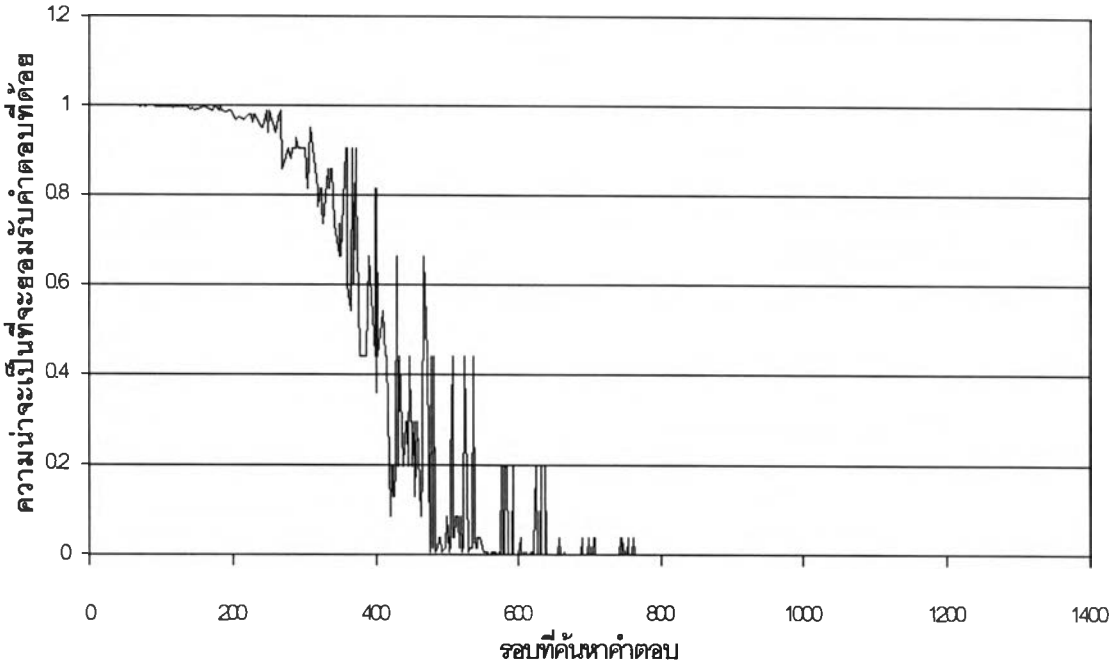


(ก) เมื่อใช้ค่า *factor* เป็น 0.99 และค่า *r* เป็น 30

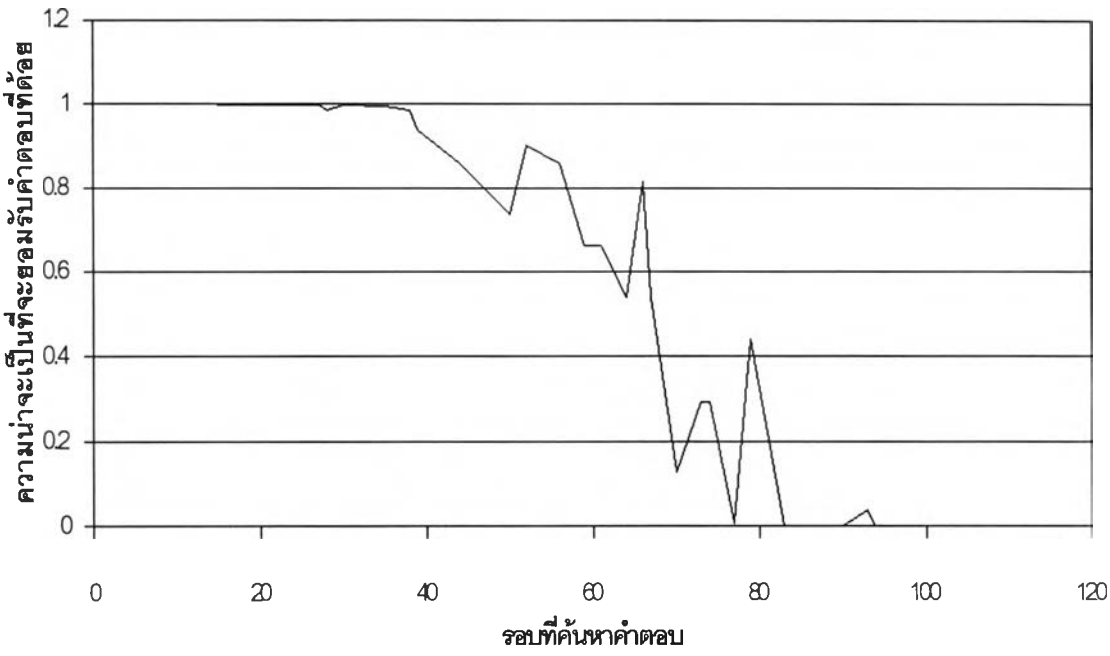


(ข) เมื่อใช้ค่า *factor* เป็น 0.99 และค่า *r* เป็น 5

รูปที่ 4.1 ลักษณะการลดลงของความน่าจะเป็นที่ยอมรับคำตอบที่ด้อย



(ค) เมื่อใช้ค่า *factor* เป็น 0.5 และค่า *r* เป็น 30



(ง) เมื่อใช้ค่า *factor* เป็น 0.5 และค่า *r* เป็น 5

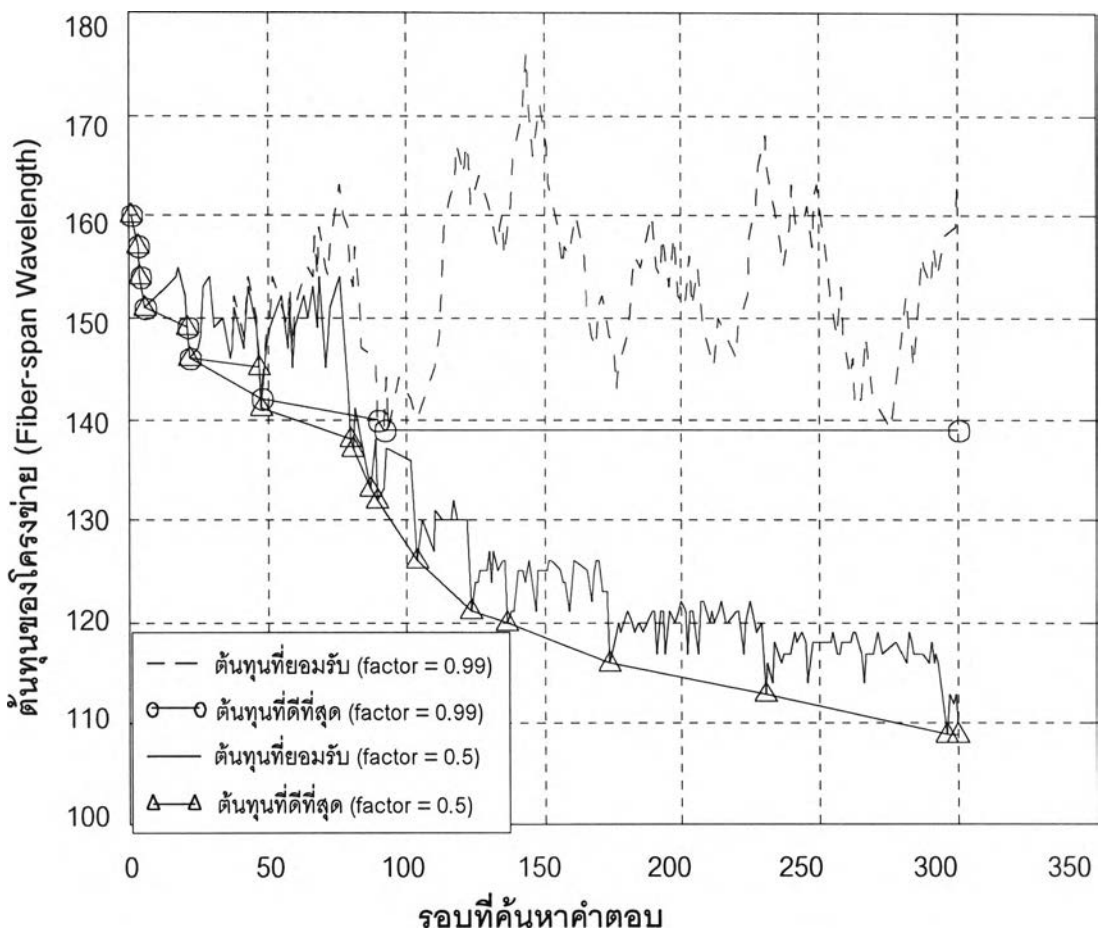
รูปที่ 4.1 (ต่อ) ลักษณะการลดลงของความน่าจะเป็นที่ยอมรับคำตอบที่ด้อย

รูปที่ 4.1 (ข) - (ง) แสดงลักษณะการลดลงของความน่าจะเป็นที่ยอมรับคำตอบที่ด้อย เมื่อใช้ค่า *factor* และค่า *r* เป็น (0.99,5), (0.5,30) และ (0.5,5) ตามลำดับ จะพบว่า ความน่าจะเป็นที่จะยอมรับคำตอบที่ด้อยมีแนวโน้มการลดลงคล้ายกรณีที่ค่า *factor* เป็น 0.99 และค่า *r* เป็น 30 คือ ในช่วงแรกจะมีค่าใกล้เคียง 1 และจะลดลงช้ามาก จากนั้นจะลู่เข้าหาค่า 0 อย่างรวดเร็วในช่วงถัดมา จนเมื่อมีค่าเข้าใกล้ 0 มากขึ้น การลดลงจะช้าลงและค่อย ๆ ลดเข้าหาค่า 0 แต่จะแตกต่างจากรูป 4.1 (ก) ตรงที่การเปลี่ยนแปลงเกิดขึ้นรวดเร็วกว่า คือ ช่วงเวลาตั้งแต่ความน่าจะเป็นมีค่าเป็น 1 จนค่าลดลงเข้าใกล้ 0 จะสั้นกว่ามาก โดยจะเกิดขึ้นภายในรอบที่ 10000, 800 และ 120 เมื่อใช้ค่า *factor* และค่า *r* เป็น (0.99,5), (0.5,30) และ (0.5,5) ตามลำดับ ความแตกต่างของระยะเวลานี้จะส่งผลให้กระบวนการค้นหาคำตอบกินเวลาสั้นหรือยาวแตกต่างกันไป

ในการเปรียบเทียบผลกระทบของความเร็วในการลดลงของความน่าจะเป็นที่มีต่อกระบวนการค้นหาคำตอบ จะแสดงเป็นค่าต้นทุนที่ได้รับการยอมรับในแต่ละรอบและต้นทุนที่ดีที่สุดที่ค้นพบในช่วง 300 รอบแรกของกระบวนการค้นหาคำตอบ ดังรูปที่ 4.2 ในการเปรียบเทียบจะทดสอบกับค่า *factor* 2 ค่าที่มีค่าต่างกันมาก ๆ คือ 0.5 และ 0.99 ใช้ค่า *r* เป็น 5 เท่ากัน จากรูป ต้นทุนที่ได้รับการยอมรับทั้ง 2 กรณีจะมีการลดลงเหมือนกัน ในช่วงตั้งแต่รอบแรกถึงประมาณรอบที่ 50 จากนั้นจึงมีค่าแตกต่างกันไป โดยต้นทุนที่ได้รับการยอมรับเมื่อใช้ ค่า *factor* เป็น 0.5 จะลดลงอย่างรวดเร็วจนกระทั่งในรอบที่ 300 ค่าต้นทุนที่ดีที่สุดที่ค้นพบเหลือเพียง 109 Fiber-span Wavelength เท่านั้น ในขณะที่หากใช้ค่า *factor* เป็น 0.99 ต้นทุนที่ได้รับการยอมรับจะมีการแกว่งตัวมาก และต้นทุนที่ดีที่สุดที่ค้นพบจะลดลงเล็กน้อย

ทั้งนี้ ในรอบแรก ๆ ของการค้นหาคำตอบ ความน่าจะเป็นที่จะยอมรับคำตอบที่ด้อยของทั้ง 2 กรณียังมีค่าใกล้เคียง 1 และไม่ลดลงมากนัก การยอมรับคำตอบในช่วงแรก ๆ ไม่ว่าจะใช้ค่า *factor* มากหรือน้อยจึงมีลักษณะเหมือนกัน กระบวนการค้นหาคำตอบที่เหมือนกันส่งผลให้ต้นทุนที่ได้รับการยอมรับในแต่ละรอบเท่ากัน อย่างไรก็ตามจะเกิดขึ้นเฉพาะในช่วงแรกและเป็นช่วงสั้น ๆ ของการค้นหาคำตอบเท่านั้น ในช่วงถัดมา ตั้งแต่รอบที่ 50 เป็นต้นไป เมื่อความน่าจะเป็นที่จะยอมรับคำตอบที่ด้อยของทั้ง 2 กรณีเริ่มมีความแตกต่างกันมากขึ้น หากใช้ค่า *factor* ต่ำ ๆ ความน่าจะเป็นที่จะยอมรับคำตอบที่ด้อยจะลดลงเข้าหาค่า 0 อย่างรวดเร็ว คำตอบที่ด้อยจึงได้รับการยอมรับยากขึ้น แต่จะมีการยอมรับคำตอบที่ดีขึ้นอย่างมาก ต้นทุนที่ได้รับการยอมรับและต้นทุนที่ดีที่สุดที่ค้นพบจึงมีการลดลงอย่างต่อเนื่อง ต่างจากกรณีที่ใช้ค่า

factor สูง ๆ ค่าความน่าจะเป็นที่จะยอมรับคำตอบที่ด้อยจะลดลงช้ากว่า คำตอบที่ด้อยกว่ามีโอกาสได้รับการยอมรับเป็นระยะเวลานาน ทำให้ค่าต้นทุนที่ได้รับการยอมรับในแต่ละรอบมีค่าแกว่งอยู่ในช่วงค่าสูง ๆ ค่าที่ดีที่สุดจึงยังไม่ลดลงมากนัก เมื่อเปรียบเทียบกันแล้วจึงสรุปได้ว่าการที่ใช้ค่า *factor* ต่ำจะสามารถหาคำตอบที่ดีในรอบแรก ๆ ได้รวดเร็วกว่าการใช้ค่า *factor* สูง ทั้งนี้ เป็นผลมาจากค่า *factor* ที่ต่างกันจะส่งผลต่าง ๆ กันต่อความน่าจะเป็นที่จะยอมรับคำตอบที่ด้อยนั่นเอง อย่างไรก็ตาม เรายังไม่อาจสรุปได้ว่าควรเลือกค่า *factor* ต่ำ ๆ ในการออกแบบโครงข่าย เนื่องจากกระบวนการค้นหาคำตอบจะต้องดำเนินต่อไปนานกว่า 300 รอบ ดังนั้นในส่วนต่อไปจะวิเคราะห์กระบวนการค้นหาคำตอบที่มีระยะเวลานานขึ้น

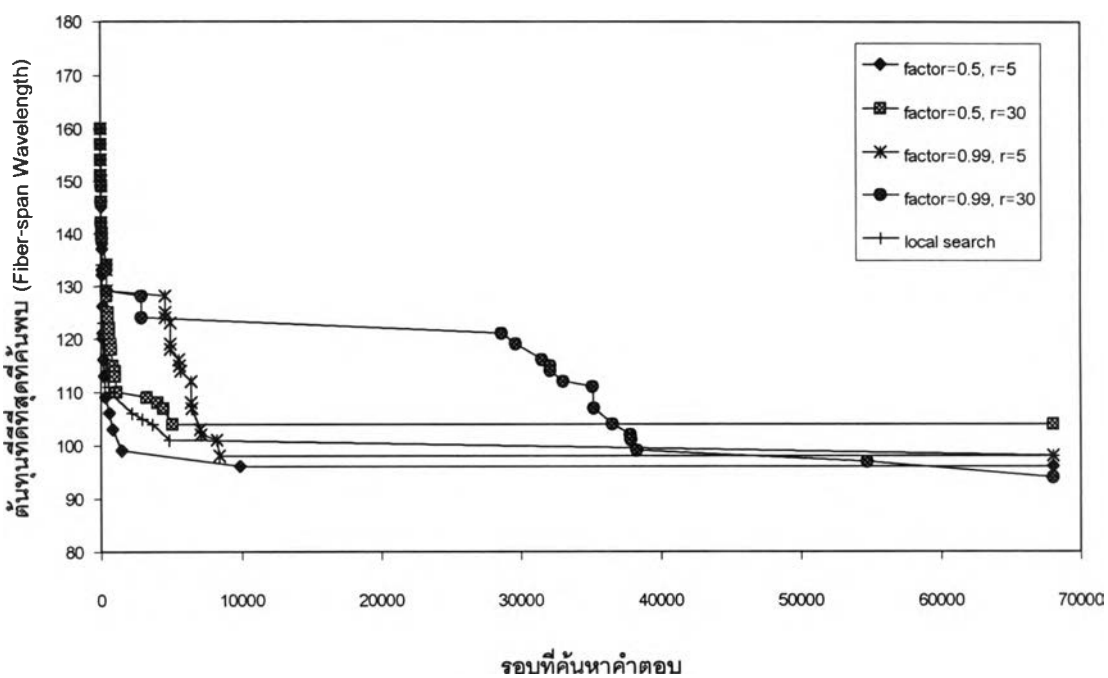


รูปที่ 4.2 ต้นทุนที่ได้รับการยอมรับและต้นทุนที่ดีที่สุด 300 รอบแรก ของโครงข่าย EUROCore เมื่อใช้ค่า *factor* เป็น 0.5 และ 0.99 ที่ค่า r เป็น 5

รูปที่ 4.3 (ก), (ข) และ (ค) แสดงต้นทุนที่ดีที่สุดที่ค้นพบ, จำนวนวงแหวนที่เลือกของต้นทุนที่ดีที่สุดที่ค้นพบและต้นทุนของคำตอบที่ค้นพบตามลำดับ เมื่อทดสอบบนโครงข่าย

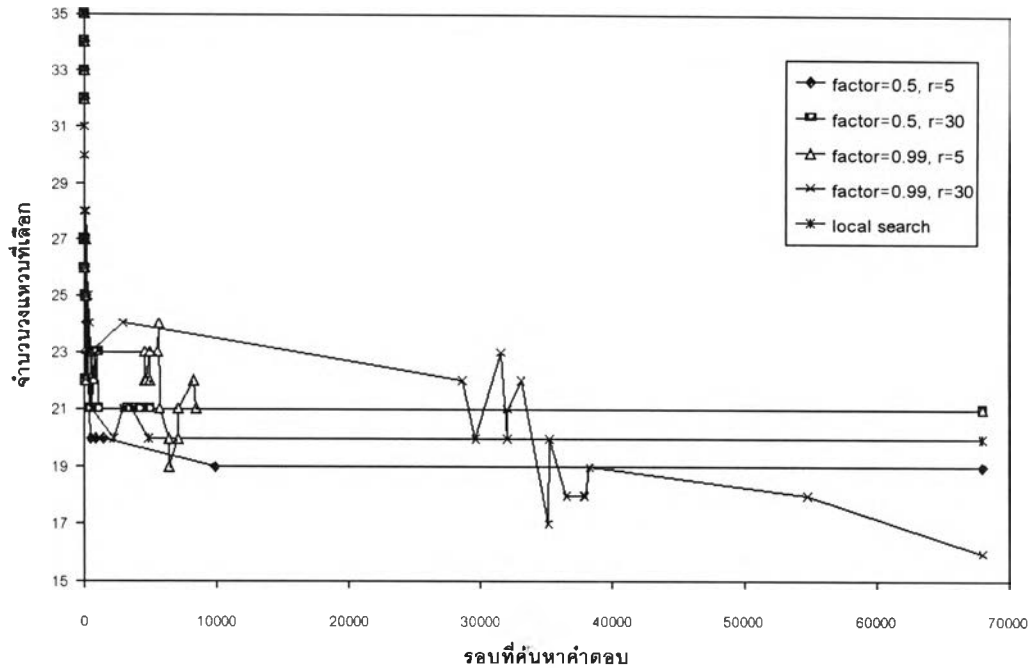
EUROCore ใช้ทราฟฟิกแบบ Uniform ที่มีขนาดเป็น 1 และกำหนดให้กระบวนการค้นหาคำตอบสิ้นสุดเมื่อไม่พบการพัฒนาของคำตอบภายใน 20000 รอบ โดยในการทดสอบจะใช้ค่าพารามิเตอร์ต่าง ๆ กัน 4 แบบ คือ (ค่า *factor* , ค่า *r*) เท่ากับ (0.99,30), (0.99,5), (0.5,30) และ (0.5,5) โดยกำหนดให้ค่าอุณหภูมิเริ่มต้นเป็น 10000 เท่ากัน

จากรูปจะเห็นว่าในรอบแรก ๆ ของการค้นหาคำตอบ ลักษณะการลดลงของต้นทุนที่ดีที่สุดมีการลดลงอย่างมากและไม่แตกต่างกันมากนักแม้จะใช้พารามิเตอร์แตกต่างกัน ทั้งนี้เนื่องจากในช่วงแรก ๆ ของการปรับปรุงการออกแบบ กระบวนการค้นหาคำตอบใหม่ไม่ขึ้นกับค่าพารามิเตอร์ แต่โอกาสค้นพบคำตอบที่ดีขึ้นได้ง่ายจากการย้ายทราฟฟิกมารวมกันในกระบวนการค้นหาคำตอบใหม่ สังเกตได้จากจำนวนวงแหวนที่มีการลดลงอย่างต่อเนื่อง จาก 35 วงเป็น 25 วง ดังในรูปที่ 4.3 (ข) อย่างไรก็ตาม เมื่อการค้นหาคำตอบดำเนินไประยะหนึ่ง คำตอบที่ให้ต้นทุนต่ำลงจะถูกค้นพบยากขึ้นนับตั้งแต่จุดนี้ ค่าพารามิเตอร์จะมีความสำคัญในการกำหนดกระบวนการยอมรับคำตอบ และกำหนดลักษณะของคำตอบที่ค้นพบ ดังจะเห็นได้ว่าเมื่อใช้ค่า *factor* และค่า *r* ต่าง ๆ กัน ลักษณะการลดลงของค่าต้นทุนที่ดีที่สุดที่ค้นพบจะแตกต่างกันไป

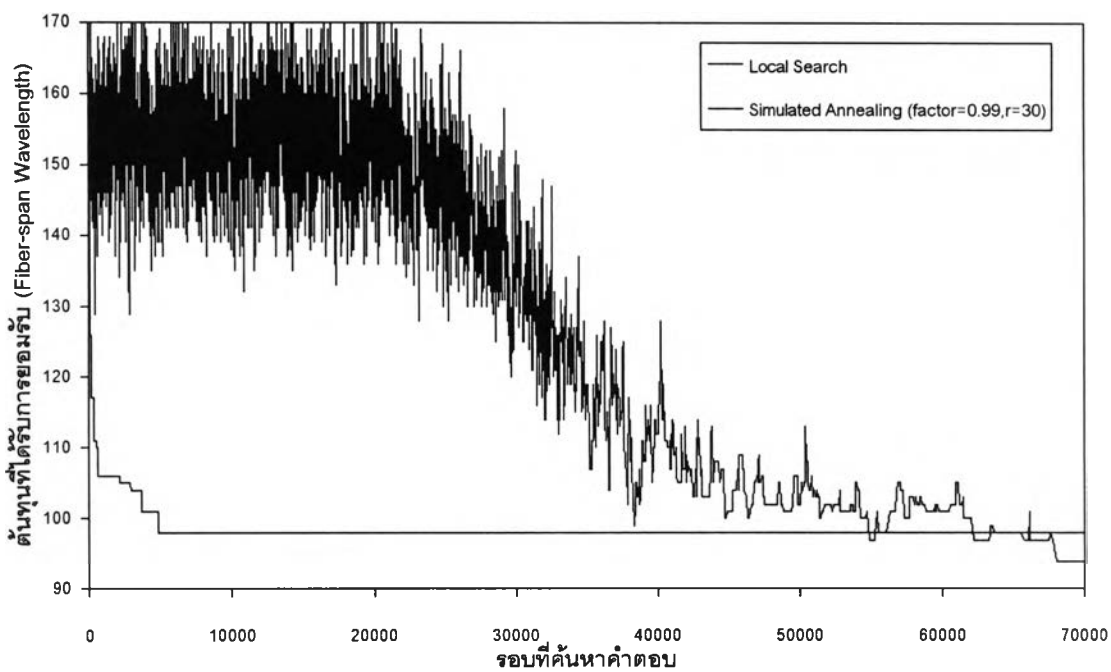


(ก) ต้นทุนที่ดีที่สุดที่ค้นพบ

รูปที่ 4.3 ผลการทดสอบโครงข่าย EUROCore ที่ทราฟฟิก Uniform ขนาดเป็น 1 ในช่วง 70000 รอบของการค้นหาคำตอบ



(ข) จำนวนวงแหวนของคำตอบที่ดีที่สุดที่ค้นพบ



(ค) ต้นทุนของคำตอบที่ค้นพบ

รูปที่ 4.3 (ต่อ) ผลการทดสอบโครงข่าย EUROCore ที่ทราฟฟิก Uniform ขนาดเป็น 1 ในช่วง 70000 รอบของการค้นหาคำตอบ

แม้ว่าการใช้ค่า *factor* ที่ต่ำ ๆ เช่น 0.5 จะมีการลดของต้นทุนอย่างมากในช่วงแรก แต่หลังจากนั้นต้นทุนจะมีค่าคงตัวก่อน *factor* ค่าอื่น ๆ การใช้ค่า *factor* สูง ๆ เช่น 0.99 แม้ว่าคำตอบจะไม่ลดลงมากนักในช่วงแรก แต่ในช่วงรอบหลัง ๆ ของกระบวนการ ต้นทุนของคำตอบจะลดลงอย่างมากจนกระทั่งเข้าสู่ค่าสุดท้ายที่ต่ำกว่าที่ได้จากค่า *factor* เป็น 0.5

ทั้งนี้เมื่อย้อนกลับไปพิจารณาลักษณะการลดลงของความน่าจะเป็นที่จะยอมรับคำตอบที่ด้อยในรูปที่ 4.1 (ก) ประกอบกับรูปที่ 4.3 ซึ่งเป็นค่าต้นทุนของคำตอบที่ค้นพบ จะอธิบายได้ว่า ในช่วง 30000 รอบแรกแม้ว่าอัลกอริทึมจะพยายามรวบรวมทราฟฟิกของหลาย ๆ คู่โนดเข้าด้วยกันเพื่อให้วงแหวนมีจำนวนน้อยลง แต่ความน่าจะเป็นที่จะยอมรับคำตอบที่ด้อยมีค่าสูง คำตอบที่ให้ต้นทุนสูงจึงได้รับการยอมรับได้ง่ายพอ ๆ กับคำตอบที่ให้ต้นทุนต่ำลง ต้นทุนของคำตอบที่ได้รับการยอมรับจึงมีลักษณะแกว่งอยู่ในช่วง 130-170 Fiber-span Wavelength คำตอบที่มีต้นทุนต่ำจึงยังไม่ถูกค้นพบมากนัก ต้นทุนที่ดีที่สุดจึงมีการลดลงอย่างช้า ๆ แต่เมื่อความน่าจะเป็นดังกล่าวลดลงอย่างมากในช่วงรอบที่ 30000-40000 คำตอบที่ด้อยจะได้รับการยอมรับน้อยลง ต้นทุนในช่วงนี้จึงลดลงอย่างมาก จนเมื่อพ้นรอบที่ 40000 ไปแล้วความน่าจะเป็นที่จะยอมรับคำตอบที่ด้อยจะมีค่าน้อยมากและใกล้ค่า 0 ทำให้มีการยอมรับคำตอบที่ดีขึ้นเท่านั้น ต้นทุนจึงลดลงอีกเล็กน้อย จนเมื่ออัลกอริทึมไม่สามารถหาคำตอบที่ให้ต้นทุนดีกว่าเดิมได้อีก จึงหยุดการค้นหาคำตอบในประมาณรอบที่ 70000

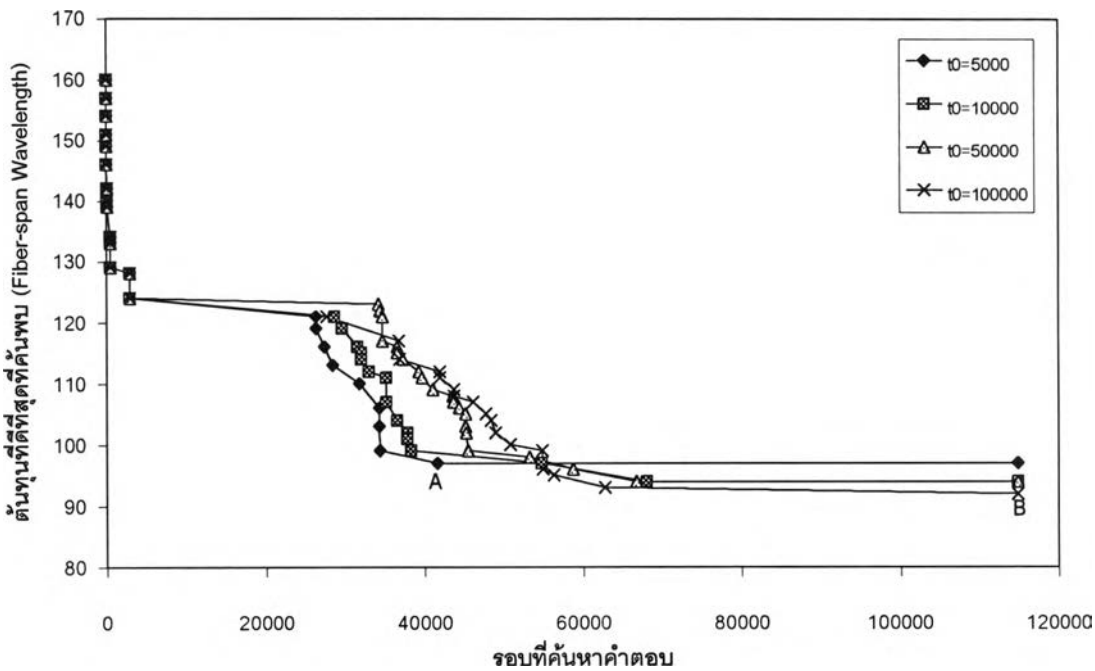
เมื่อเปรียบเทียบกับกรณีที่ใช้ค่า *factor* และค่า *r* ที่น้อยจะส่งผลให้ความน่าจะเป็นที่จะยอมรับคำตอบที่ด้อยลดลงเข้าหาค่า 0 อย่างรวดเร็ว ทำให้ต้นทุนที่ดีที่สุดที่ค้นพบลดลงเร็วมากในช่วงแรก คำตอบที่ดีจึงมักจะได้มาในช่วงนี้ หลังจากนั้นเมื่อความน่าจะเป็นดังกล่าวลดลงจนเกือบเป็น 0 จะไม่สามารถหาคำตอบที่ดีได้อีก สังเกตว่าต้นทุนจะคงตัวไปตลอดต้นทุนสุดท้ายที่ได้จึงมีค่าสูงกว่าต้นทุนในกรณีที่ใช้ค่า *factor* และค่า *r* สูง ๆ กระบวนการค้นหาคำตอบจึงมีลักษณะคล้ายกับเมื่อใช้ Local Search มาก

จากที่กล่าวมาทั้งหมด สามารถสรุปได้ว่ากระบวนการยอมรับคำตอบมีความสำคัญต่อกระบวนการค้นหาคำตอบอย่างมาก และเราสามารถปรับเปลี่ยนกระบวนการค้นหาคำตอบได้โดยใช้การปรับค่า *factor* และค่า *r* ค่า *factor* และค่า *r* ที่ต่ำจะทำให้เกิดการยอมรับคำตอบที่ดีขึ้นอย่างมากตั้งแต่ในรอบแรก ๆ การค้นหาคำตอบจึงพบปัญหาชุดของคำตอบติดอยู่ใน Local Optimum ได้ง่ายคล้ายใน Local Search แต่หากปรับค่า *factor* และค่า *r* ให้สูง จะสามารถแก้ปัญหานี้ได้ คำตอบที่ได้จะมีต้นทุนต่ำกว่ากรณีแรก แต่กระบวนการค้นหาคำตอบจะกินระยะเวลานานมาก ดังนั้นควรปรับค่า *factor* และค่า *r* ให้ต่ำลงเล็กน้อยเพื่อให้ได้คำตอบ

ภายในเวลาที่กำหนด การวิเคราะห์หาค่า *factor* และค่า r ที่เหมาะสมสำหรับโครงข่ายแบบต่าง ๆ จะกล่าวถึงอีกครั้งในหัวข้อ 4.2.3 หลังจากได้วิเคราะห์ค่าพารามิเตอร์อีกตัวหนึ่งที่มีความสำคัญสำหรับกระบวนการค้นหาคำตอบแบบ Simulated Annealing คือ ค่าอุณหภูมิเริ่มต้น ดังจะได้กล่าวถึงในหัวข้อถัดไป

4.2.2 การวิเคราะห์ค่าอุณหภูมิเริ่มต้น (t_0)

ค่าอุณหภูมิเป็นพารามิเตอร์ที่สำคัญอีกตัวหนึ่งซึ่งมีผลต่อกระบวนการค้นหาคำตอบ ในหัวข้อนี้จะศึกษาการกำหนดค่าอุณหภูมิเริ่มต้นให้เหมาะสมกับปัญหา ในฮิวริสติกอัลกอริทึมแบบ Simulated Annealing จะกำหนดให้ค่าอุณหภูมิมีค่าสูงมากในรอบแรกก่อน คือ $t_0 >> |Cost_1 - Cost_0|$ เพื่อให้ความน่าจะเป็นที่จะยอมรับคำตอบที่ด้อยมีค่าสูงและใกล้เคียง 1 มากที่สุด ในการทดสอบจะใช้โครงข่าย EUROCore ทราฟฟิกที่ใช้เป็นแบบ Uniform ขนาดเท่ากับ 1 และ M มีค่าเป็น 1 โดยทดลองกับอุณหภูมิเริ่มต้นค่าต่าง ๆ กัน คือ 5000, 10000, 50000 และ 100000 และกำหนดให้ค่า *factor* คงตัวเป็น 0.99, ค่า *r* เป็น 30 การลดลงของต้นทุนที่ดีที่สุดที่ค้นพบ ในช่วง 120000 รอบของการค้นหาคำตอบ แสดงดังรูปที่ 4.4



รูปที่ 4.4 ลักษณะการลดลงของค่าต้นทุนที่ดีที่สุดที่ค้นพบ
เมื่อใช้อุณหภูมิเริ่มต้นค่าต่าง ๆ

จากรูปพบว่าการใช้ค่าอุณหภูมิเริ่มต้นต่าง ๆ กันมีผลต่อการลดลงของต้นทุนเห็นได้ชัดเฉพาะช่วงที่มีการลดลงอย่างมากของต้นทุน คือ ตั้งแต่รอบที่ 30000-50000 โดยต้นทุนจะเริ่มลดลงเรียงจากก่อนไปหลังตามลำดับ คือ เมื่อใช้อุณหภูมิเริ่มต้นเป็น 5000, 10000, 50000 และ 100000 แสดงว่าหากใช้อุณหภูมิเริ่มต้นต่ำ เช่น 5000 หรือ 10000 ช่วงที่ต้นทุนจะลดลงอย่างมากจะเกิดขึ้นรวดเร็วกว่าอุณหภูมิเริ่มต้นค่าสูง ๆ เช่น 50000 หรือ 100000 ทั้งนี้เนื่องจากการใช้อุณหภูมิเริ่มต้นที่มีค่าต่ำจะทำให้ความน่าจะเป็นที่จะยอมรับคำตอบที่ด้อยลดลงหาค่า 0 ในเวลาอันรวดเร็ว กระบวนการค้นหาคำตอบจะสิ้นสุดก่อน แต่การใช้อุณหภูมิเริ่มต้นที่มีค่าสูง ความน่าจะเป็นจะลดลงหาค่า 0 โดยใช้เวลานานมาก กระบวนการค้นหาคำตอบจะกินเวลานานกว่ากรณีแรก เช่น เมื่อใช้ค่าอุณหภูมิเริ่มต้นเป็น 5000 ค่าต้นทุนที่ดีที่สุดจะได้มาในรอบที่ 41597 และมีค่าเท่ากับ 97 Fiber-span Wavelength (จุด A) ส่วนการใช้อุณหภูมิเริ่มต้นเป็น 100000 ค่าต้นทุนที่ดีที่สุดจะได้มาในรอบที่ 114795 และมีค่าเท่ากับ 92 Fiber-span Wavelength (จุด B) จะเห็นว่ากรณีหลังนี้ ต้นทุนสุดท้ายจะมีค่าต่ำกว่ากรณีแรก

แม้ว่าการใช้อุณหภูมิเริ่มต้นเป็น 100000 จะให้ต้นทุนสุดท้ายต่ำ แต่กระบวนการค้นหาคำตอบที่กินเวลานานกว่ามาก เมื่อเปรียบเทียบกับอุณหภูมิเริ่มต้นเป็น 5000 ซึ่งเป็นกรณีที่กระบวนการค้นหาคำตอบกินเวลาน้อยที่สุด ค่าต้นทุนสุดท้ายของกรณีอุณหภูมิเริ่มต้นเป็น 100000 จะต่ำกว่ากรณีอุณหภูมิเริ่มต้นเป็น 5000 เพียง 5 Fiber-span Wavelength ในขณะที่คำตอบสุดท้ายจะถูกค้นพบที่จำนวนรอบสูงกว่าถึง 73198 รอบ ดังนั้นหากคำนึงถึงเวลาในการคำนวณ จึงไม่ควรเลือกค่าอุณหภูมิเริ่มต้นที่สูงที่สุดแต่ควรเลือกค่าที่สามารถให้ค่าต้นทุนที่ดีได้ภายในช่วงเวลาที่ต้องการ

4.2.3 ค่า *factor* และ ค่า *r* ที่มีความเหมาะสมกับการออกแบบโครงข่ายแต่ละแบบ

ในส่วนนี้จะวิเคราะห์หาค่าพารามิเตอร์ที่เหมาะสมในการใช้กับฮิวริสติกอัลกอริทึมแบบ Simulated Annealing (SA) เมื่อต้องการหยุดค้นหาคำตอบเมื่อไม่พบการพัฒนาของคำตอบภายใน 10000 รอบ ในการหาค่าพารามิเตอร์ที่เหมาะสมที่จะทำให้ต้นทุนที่ดีที่สุด จะทดสอบกับค่า *factor* ต่าง ๆ กัน โดยจะแปรค่าไปทีละ 0.05 ตั้งแต่ 0.50 ถึง 0.99 ซึ่งเป็นค่าช่วงที่มีการใช้กันอย่างแพร่หลาย โดยสามารถให้คำตอบที่ดีได้ [15] โดยจะแปรค่า *r* ไปทีละ 5 รอบ ตั้งแต่ 5 รอบถึง 30 รอบ ค่าอุณหภูมิเริ่มต้นที่จะกำหนดให้มีค่าเท่ากับ 10000 ในทุก ๆ กรณี การทดสอบจะทำบนโครงข่าย EUROCore ผลที่ได้จะแสดงเป็นต้นทุนของโครงข่ายเมื่อใช้ฮิวริสติก

อัลกอริทึมแบบ SA ดังแสดงในตารางที่ 4.1 อนึ่ง ทราฟฟิกที่ใช้เป็นทราฟฟิกแบบ Uniform ที่มีขนาดเป็น 1 และ M มีค่าเป็น 1

ตารางที่ 4.1 ต้นทุนจากการปรับปรุงคำตอบด้วย Simulated Annealing บนโครงข่าย EUROCore เมื่อใช้ทราฟฟิก Uniform ขนาดเท่ากับ 1 ที่ M=1 และจะหยุดค้นหาคำตอบเมื่อไม่พบการพัฒนาของคำตอบภายใน 10000 รอบ (ต้นทุนที่ได้จาก Local Search = 98)

ค่า *r*	ค่า *factor*										
	0.99	0.95	0.90	0.85	0.80	0.75	0.70	0.65	0.60	0.55	0.50
5	97	98	96	98	102	98	98	101	101	101	96
10	94	99	96	95	96	98	95	100	101	99	103
15	96	97	98	99	99	98	95	100	98	96	101
20	96	98	98	101	99	96	98	95	96	98	99
25	128	98	98	100	96	97	96	95	103	101	102
30	131	98	99	97	96	98	96	105	98	94	95

ตารางที่ 4.2 ต้นทุนจากการปรับปรุงคำตอบด้วย Simulated Annealing บนโครงข่าย THAINet เมื่อใช้ทราฟฟิก Uniform ขนาดเท่ากับ 1 ที่ M=1 และจะหยุดค้นหาคำตอบเมื่อไม่พบการพัฒนาของคำตอบภายใน 10000 รอบ (ต้นทุนที่ได้จาก Local Search = 276)

ค่า *r*	ค่า *factor*										
	0.99	0.95	0.90	0.85	0.80	0.75	0.70	0.65	0.60	0.55	0.50
5	268	271	263	268	271	266	269	266	270	264	269
10	260	276	266	271	267	275	267	269	267	260	272
15	313	262	267	268	268	267	268	268	264	266	263
20	310	265	273	274	263	263	265	269	270	265	262
25	310	276	274	272	268	265	270	260	260	270	273
30	313	263	270	267	258	258	265	261	277	265	274

ตารางที่ 4.3 ต้นทุนจากการปรับปรุงคำตอบด้วย Simulated Annealing บนโครงข่าย NSFNet เมื่อใช้ทราฟฟิก Uniform ขนาดเท่ากับ 1 ที่ M=1 และจะหยุดค้นหาคำตอบเมื่อไม่พบการพัฒนาของคำตอบภายใน 10000 รอบ (ต้นทุนที่ได้จาก Local Search = 219)

ค่า *r*	ค่า *factor*										
	0.99	0.95	0.90	0.85	0.80	0.75	0.70	0.65	0.60	0.55	0.50
5	211	216	217	212	215	209	213	217	218	217	219
10	212	216	217	218	213	214	210	214	218	216	215
15	212	210	212	213	217	215	217	219	215	217	214
20	213	215	214	217	215	216	212	215	217	219	211
25	216	216	218	216	218	216	218	216	214	218	217
30	269	219	219	214	210	220	217	210	218	218	211

ตารางที่ 4.4 ต้นทุนจากการปรับปรุงคำตอบด้วย Simulated Annealing บนโครงข่าย EON เมื่อใช้ทราฟฟิก Uniform ขนาดเท่ากับ 1 ที่ M=1 และจะหยุดค้นหาคำตอบเมื่อไม่พบการพัฒนาของคำตอบภายใน 10000 รอบ (ต้นทุนที่ได้จาก Local Search = 397)

ค่า *r*	ค่า *factor*										
	0.99	0.95	0.90	0.85	0.80	0.75	0.70	0.65	0.60	0.55	0.50
5	385	396	385	393	398	396	392	389	389	390	397
10	390	390	395	384	390	385	397	386	396	386	392
15	534	394	400	392	388	395	392	394	380	402	392
20	547	387	398	395	393	395	389	387	395	491	399
25	547	384	392	393	390	392	394	396	386	394	389
30	548	391	387	384	393	393	397	384	389	385	391

ตารางที่ 4.5 ต้นทุนจากการปรับปรุงคำตอบด้วย Simulated Annealing บนโครงข่าย ARPANet เมื่อใช้ทราฟฟิก Uniform ขนาดเท่ากับ 1 ที่ M=1 และจะหยุดค้นหาคำตอบเมื่อไม่พบการพัฒนาของคำตอบภายใน 10000 รอบ (ต้นทุนที่ได้จาก Local Search = 608)

ค่า *r*	ค่า *factor*										
	0.99	0.95	0.90	0.85	0.80	0.75	0.70	0.65	0.60	0.55	0.50
5	607	601	607	606	594	603	601	602	610	591	602
10	610	603	597	598	609	595	600	613	599	593	601
15	736	599	594	603	596	590	617	589	606	605	597
20	732	590	594	604	612	609	601	596	599	595	604
25	736	606	609	613	600	617	612	588	595	598	608
30	736	601	622	600	603	597	594	606	603	605	595

ตารางที่ 4.6 ต้นทุนจากการปรับปรุงคำตอบด้วย Simulated Annealing บนโครงข่าย UKNet เมื่อใช้ทราฟฟิก Uniform ขนาดเท่ากับ 1 ที่ M=1 และจะหยุดค้นหาคำตอบเมื่อไม่พบการพัฒนาของคำตอบภายใน 10000 รอบ (ต้นทุนที่ได้จาก Local Search = 613)

ค่า *r*	ค่า *factor*										
	0.99	0.95	0.90	0.85	0.80	0.75	0.70	0.65	0.60	0.55	0.50
5	599	608	597	607	596	596	603	589	590	595	602
10	607	595	594	611	594	594	610	595	607	599	589
15	611	593	591	595	595	595	595	596	604	607	607
20	609	600	619	594	603	603	604	592	605	617	607
25	604	596	589	605	600	600	596	604	616	598	601
30	901	606	620	589	583	583	595	612	600	591	602

เมื่อเปรียบเทียบต้นทุนที่ได้จากอัลกอริทึมแบบ SA กับต้นทุนที่ได้จากอัลกอริทึมแบบ Local Search (LS) จะเห็นว่าต้นทุนของอัลกอริทึมแบบ SA ในบางกรณีมีค่าต่ำกว่าต้นทุนของอัลกอริทึมแบบ LS แต่ในบางกรณีมีค่าสูงกว่าหรือเท่ากัน แสดงให้เห็นว่าอัลกอริทึมแบบ SA ไม่จำเป็นต้องให้ต้นทุนที่ดีกว่าอัลกอริทึมแบบ LS เสมอไปในทุกค่า *factor* และค่า *r* ที่ใช้

อัลกอริทึมแบบ SA เพียงแต่เพิ่มโอกาสในการค้นพบคำตอบที่ดีกว่าคำตอบที่พบใน LS เท่านั้น และหากต้องการหยุดค้นหาคำตอบเมื่อไม่พบการพัฒนาของคำตอบภายใน 10000 รอบ ไม่ควรเลือกใช้ค่า *factor* และค่า *r* ที่สูงเกินไป เช่น *factor* เป็น 0.99 และค่า *r* เป็น 25 หรือ 30 รอบ เนื่องจากยังไม่ถึงจุดที่ต้นทุนจะลดลงสู่ค่าที่ต่ำ ต้นทุนจึงยังคงสูงมากดังที่ได้อธิบายไปแล้ว

เมื่อพิจารณาผลการทดสอบในแต่ละโครงข่ายที่เหลือในตารางที่ 4.2 – 4.6 จะพบว่าประสิทธิภาพในการค้นหาคำตอบของอัลกอริทึมแบบ SA เมื่อเทียบกับ LS นอกจากจะแตกต่างกันไปในแต่ละค่า *factor* และค่า *r* แต่ยังแตกต่างกันไปในแต่ละโครงข่าย ตารางที่ 4.7 แสดงจำนวนต้นทุนที่อัลกอริทึมแบบ SA สามารถค้นพบได้ค่าดีกว่าและด้อยกว่าต้นทุนที่ค้นพบโดยอัลกอริทึมแบบ LS ในโครงข่ายต่าง ๆ ที่ทดสอบ โดยจะเรียงจากโครงข่ายที่มี Connectivity มากที่สุด คือ โครงข่าย EUROCore ไปโครงข่ายที่มี Connectivity น้อยที่สุด คือ โครงข่าย THAINet

ตารางที่ 4.7 จำนวนต้นทุนที่ค้นพบในอัลกอริทึมแบบ SA ที่ดีกว่าและด้อยกว่า LS เมื่อหยุดการค้นหาคำตอบเมื่อไม่มีการพัฒนาของคำตอบภายใน 10000 รอบ

โครงข่าย	ค่า Connectivity ของโครงข่าย	จำนวนคำตอบที่ SA ดีกว่า LS	จำนวนคำตอบที่ SA ด้อยกว่าหรือเท่ากับ LS
EUROCore	4.55	25	41
EON	3.89	51	15
UKNet	3.71	60	6
APPANet	3.10	48	18
NSFNet	3.00	58	8
THAINet	2.86	58	8

จากตารางจะเห็นว่า ภายใต้ข้อกำหนดที่ให้หยุดค้นหาคำตอบภายใน 10000 รอบที่ไม่พบการพัฒนาของคำตอบ อัลกอริทึมแบบ SA จะสามารถค้นพบคำตอบได้ดีกว่าอัลกอริทึมแบบ LS เป็นส่วนใหญ่ ยกเว้นในโครงข่าย EUROCore ซึ่งมี Connectivity สูงที่สุด แสดงว่าอัลกอริทึมแบบ SA สามารถช่วยแก้ปัญหาการออกแบบโครงข่ายได้ดีกว่าอัลกอริทึมแบบ LS ในโครงข่ายที่มี Connectivity ต่ำเป็นส่วนใหญ่ แต่ในโครงข่ายที่มี Connectivity สูง เช่น EUROCore อัลกอริทึมแบบ LS สามารถค้นพบคำตอบที่ดีซึ่งให้ต้นทุนต่ำได้อยู่แล้ว อัลกอริทึม

แบบ SA เมื่อใช้ค่า factor และค่า r บางค่าจึงไม่สามารถออกแบบให้ได้ต้นทุนต่ำกว่าต้นทุนที่ได้จากอัลกอริทึมแบบ LS แต่เมื่อเพิ่มระยะเวลาในการค้นหาคำตอบบนโครงข่าย EUROCore ให้นานขึ้น คือ จะหยุดค้นหาคำตอบเมื่อไม่พบการพัฒนาของคำตอบภายใน 50000 รอบ ผลการทดสอบดังในตารางที่ 4.8 แสดงให้เห็นว่า อัลกอริทึมแบบ SA สามารถแก้ปัญหาการออกแบบโครงข่าย EUROCore ได้ดีขึ้น คือ จำนวนคำตอบที่อัลกอริทึมแบบ SA สามารถค้นพบได้ต้นทุนต่ำกว่าอัลกอริทึมแบบ LS จะเพิ่มขึ้นเป็นจาก 25 ค่าเป็น 40 ค่า

ตารางที่ 4.8 ต้นทุนจากการปรับปรุงคำตอบด้วย Simulated Annealing บนโครงข่าย EUROCore เมื่อใช้ทราฟฟิก Uniform ขนาดเท่ากับ 1 ที่ M=1 และจะหยุดค้นหาคำตอบเมื่อไม่พบการพัฒนาของคำตอบภายใน 50000 รอบ (ต้นทุนที่ได้จาก Local Search = 98)

ค่า *r*	ค่า *factor*										
	0.99	0.95	0.90	0.85	0.80	0.75	0.70	0.65	0.60	0.55	0.50
5	97	98	96	98	102	95	98	95	101	101	96
10	94	98	96	95	96	96	95	100	95	99	103
15	93	96	98	97	98	98	95	96	98	94	98
20	95	97	95	98	99	96	98	95	95	98	95
25	93	95	98	95	96	97	96	95	100	98	98
30	96	98	98	96	96	96	96	96	98	94	95

ดังนั้นสามารถสรุปได้ว่า อัลกอริทึมแบบ SA สามารถแก้ปัญหาการออกแบบโครงข่ายได้ดีกว่าอัลกอริทึมแบบ LS เป็นส่วนใหญ่ ยกเว้นในโครงข่ายที่มี Connectivity สูงมาก อัลกอริทึมแบบ LS สามารถออกแบบแล้วได้คำตอบที่ให้ต้นทุนต่ำอยู่แล้ว การใช้อัลกอริทึมแบบ SA ในการแก้ปัญหาเพื่อให้ได้มาซึ่งคำตอบที่ต้นทุนต่ำกว่าอัลกอริทึมแบบ LS จึงอาจต้องขยายเวลาในการค้นหาคำตอบออกไปให้นานขึ้น

4.3 การวิเคราะห์พารามิเตอร์ที่ใช้ในฮิวริสติกอัลกอริทึมแบบ Tabu Search

ในการวิเคราะห์พารามิเตอร์ใน Tabu Search ส่วนแรกเป็นการพิจารณาค่าพารามิเตอร์ที่ส่งผลต่อประสิทธิภาพของฮิวริสติกอัลกอริทึมแบบ Tabu Search (TS) พารามิเตอร์ที่กล่าวถึงในที่นี้ คือ ความยาวของ Tabu List โดยจะพิจารณาในแง่ประโยชน์ของ Tabu List ต่อต้นทุนของโครงข่าย ในส่วนถัดไปจึงพิจารณาประสิทธิภาพในฮิวริสติกอัลกอริทึมอีกแบบหนึ่ง คือ ฮิวริสติกอัลกอริทึมแบบ Tabu Search ประยุกต์ร่วมกับแบบ Local Search (TabuLS)

4.3.1 การวิเคราะห์พารามิเตอร์ในฮิวริสติกอัลกอริทึมแบบ Tabu Search

ในส่วนนี้ จะพิจารณาผลของความยาวของ Tabu List ตารางที่ 4.9 – 4.11 แสดงผลของความยาวของ Tabu List ต่อค่าต้นทุนของโครงข่ายและจำนวนรอบที่ได้คำตอบที่ดีที่สุด เมื่อทดสอบบนโครงข่าย 3 โครงข่าย ได้แก่ NSFNet ขนาด 14 โนด, EUROCore ขนาด 11 โนด และ EON ขนาด 18 โนด ทราฟฟิกที่ใช้เป็นแบบ Uniform ที่มีขนาดเป็น 1, M เป็น 1 และจะหยุดการค้นหาคำตอบเมื่อไม่มีการพัฒนาของคำตอบให้เห็นภายใน 10000 รอบ

จากตารางที่ 4.9 – 4.11 จะเห็นว่าความยาว Tabu List มีผลต่อทั้งต้นทุนและจำนวนรอบที่ค้นพบคำตอบต่างกันไปในแต่ละโครงข่าย โดยในโครงข่ายแต่ละแบบ ต้นทุนที่ต่ำที่สุดจากการทดสอบกำหนด Tabu List ให้มีความยาวต่าง ๆ กัน จะพบเมื่อใช้ Tabu List ยาว 2, 6 และ 3 ในโครงข่าย EUROCore, NSFNet และ EON ตามลำดับ จากการพิจารณาต้นทุนที่ได้จาก Tabu List ความยาวต่าง ๆ ในโครงข่ายแต่ละแบบ กลับไม่พบความสัมพันธ์ใด ๆ ระหว่างค่าต้นทุนของโครงข่ายกับความยาวของ Tabu List การกำหนดความยาว Tabu List ให้เหมาะสมสำหรับโครงข่ายแบบหนึ่ง ๆ จึงต้องอาศัยการทดสอบกับทุกกรณี อีกทั้งต้นทุนเมื่อกำหนดความยาว Tabu List ความยาวต่าง ๆ กันมีค่าไม่แตกต่างกันมากกับกรณีที่ไม่มี Tabu List (กรณีที่ความยาวเป็น 0) ดังนั้นสรุปได้ว่าการใช้ Tabu List ไม่ได้เอื้อประโยชน์ต่อการแก้ปัญหาเท่าใดนัก ในบางกรณีอาจให้คำตอบที่ต้นทุนด้อยลงอีกด้วย

ตารางที่ 4.9 ต้นทุนที่ดีที่สุดที่ค้นพบของโครงข่าย EUROCore และจำนวนรอบที่ค้นพบคำตอบเมื่อใช้ความยาวของ Tabu List ต่าง ๆ กัน

ความยาว Tabu List	0	1	2	3	4	5	6	7	8
ต้นทุน	128	127	121	129	129	129	129	129	129
รอบที่ค้นพบ	7225	10806	2731	1569	1569	1569	1569	1569	1569

ตารางที่ 4.10 ต้นทุนที่ดีที่สุดที่ค้นพบของโครงข่าย NSFNet และจำนวนรอบที่ค้นพบคำตอบเมื่อใช้ความยาวของ Tabu List ต่าง ๆ กัน

ความยาว Tabu List	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
ต้นทุน	271	269	267	268	272	269	258	263	269
รอบที่ค้นพบ	9200	15053	13515	1957	2369	12019	9724	9724	3388
ความยาว Tabu List	**9**	**10**							
ต้นทุน	269	269							
รอบที่ค้นพบ	3388	3388							

ตารางที่ 4.11 ต้นทุนที่ดีที่สุดที่ค้นพบของโครงข่าย EON และจำนวนรอบที่ค้นพบคำตอบเมื่อใช้ความยาวของ Tabu List ต่าง ๆ กัน (× หมายถึง ขั้นตอนการปรับปรุงการออกแบบไม่สามารถค้นพบคำตอบที่ให้ต้นทุนต่ำกว่าคำตอบเบื้องต้นจากอัลกอริทึม SRSR ได้)

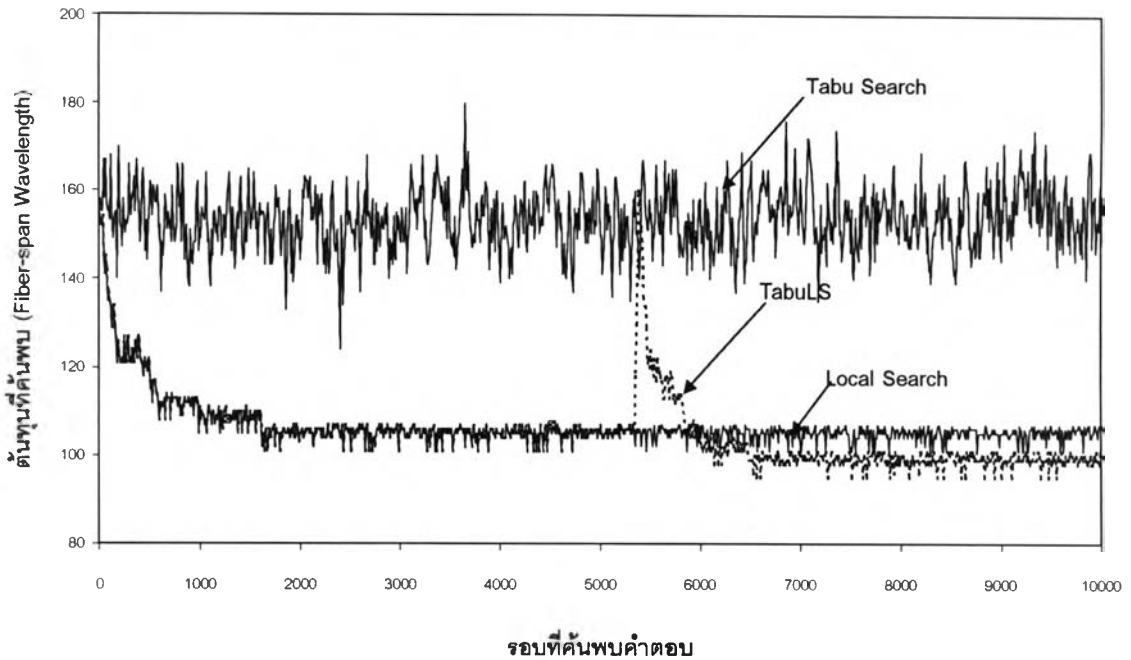
ความยาว Tabu List	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
ต้นทุน	555	555	547	545	559	552	553	559	553
รอบที่ค้นพบ	17476	509	8462	7788	×	5370	5762	×	4031
ความยาว Tabu List	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
ต้นทุน	556	558	559	559	559	559	556	559	559
รอบที่ค้นพบ	8806	7028	×	×	×	×	5365	×	×
ความยาว Tabu List	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	
ต้นทุน	559	559	556	555	555	559	559	559	
รอบที่ค้นพบ	×	×	5365	5363	5363	×	×	×	

4.3.2 การวิเคราะห์ประสิทธิภาพของฮิวริสติกอัลกอริทึมแบบ Tabu Search ประยุกต์ร่วมกับแบบ Local Search

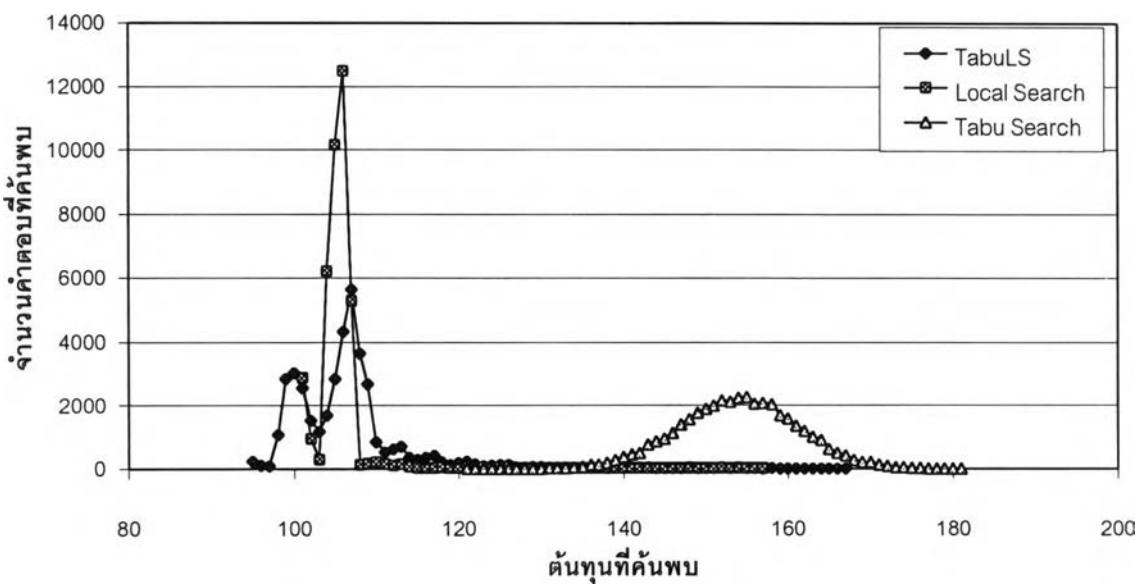
ในส่วนนี้จะเป็นการเปรียบเทียบประสิทธิภาพของการปรับปรุงคำตอบเมื่อใช้ฮิวริสติกอัลกอริทึม 3 แบบ คือ แบบ Tabu Search ประยุกต์ร่วมกับแบบ Local Search (TabuLS), แบบ Tabu Search และแบบ Local Search รูปที่ 4.5 แสดงต้นทุนของโครงข่าย EUROCore แสดงตามจำนวนรอบที่ค้นหาคำตอบของฮิวริสติกอัลกอริทึมแต่ละแบบ เมื่อกำหนดให้ความยาว Tabu List เป็น 2, ทราฟฟิกที่ใช้ทดสอบเป็นทราฟฟิกแบบ Uniform ที่มีขนาดเป็น 1, M=1 และจำนวนรอบที่อนุญาตให้คำตอบอยู่ใน Local Optimum ในฮิวริสติกอัลกอริทึมแบบ TabuLS เป็น 6000 รอบ โดยจะพิจารณาไปพร้อม ๆ กับรูปที่ 4.6 ซึ่งแสดงการกระจายของคำตอบที่ให้ต้นทุนแต่ละค่า

จากรูปที่ 4.5 ต้นทุนที่ค้นพบส่วนใหญ่ในฮิวริสติกอัลกอริทึมแบบ Tabu Search จะมีค่าอยู่ในช่วงสูงกว่า Local Search และ TabuLS และช่วงการค้นหาคำตอบจะกว้างมาก คือ มีคำตอบที่ให้ต้นทุนหลากหลายตั้งแต่ 121 ถึง 181 Fiber-span Wavelength ทั้งนี้เนื่องจากใน Tabu Search คำตอบทุกกรณีจะได้รับการยอมรับ แม้ว่าการยอมรับคำตอบแบบนี้จะส่งผลดี คือ ทำให้คำตอบที่ได้มีความหลากหลาย แต่กลับไม่ทำให้เกิดการรวมกันของทราฟฟิกเพื่อลดจำนวนวงแหวนลง จึงค้นพบแต่คำตอบที่มีต้นทุนสูงจำนวนมาก จนไม่สามารถค้นพบคำตอบที่ดีได้ เมื่อเปรียบเทียบกับฮิวริสติกอัลกอริทึมอีก 2 แบบแล้ว สรุปได้ว่า Tabu Search ไม่เหมาะสมที่จะนำมาใช้ในการปรับปรุงการออกแบบโครงข่าย

สำหรับฮิวริสติกอัลกอริทึมแบบ Local Search และแบบ TabuLS จะพบว่าค่าต้นทุนที่ค้นพบมีค่าอยู่ในช่วงที่แคบลง และมีต้นทุนต่ำกว่าในแบบ Tabu Search แสดงให้ว่าหากมีการจำกัดการยอมรับคำตอบให้ยอมรับเฉพาะคำตอบที่ดีขึ้นตลอดการค้นหาคำตอบทั้งหมดหรือแม้แต่ในบางช่วง อัลกอริทึมจะสามารถให้คำตอบที่มีต้นทุนต่ำได้ ข้อกำหนดในการยอมรับคำตอบจึงมีผลต่อคำตอบที่ค้นพบอย่างมาก



รูปที่ 4.5 ต้นทุนที่ค้นพบของฮิวริสติกอัลกอริทึมการค้นหาคำตอบ
แบบ Local Seach, Tabu Search และ TabuLS



รูปที่ 4.6 จำนวนคำตอบที่ให้ต้นทุนแต่ละค่า

เมื่อเปรียบเทียบการกระจายของต้นทุนของคำตอบ ระหว่างฮิวริสติกอัลกอริทึมแต่ละแบบ ดังในรูปที่ 4.6 จะพบว่าแบบ Local Search สามารถค้นพบคำตอบที่ต้นทุนต่ำเป็นจำนวนมากกว่าแบบ TabuLS แต่พบว่าคำตอบที่ต่ำที่สุดที่ค้นพบได้มาจากฮิวริสติกอัลกอริทึมแบบ TabuLS ทั้งนี้เนื่องจากฮิวริสติกอัลกอริทึมแบบ TabuLS มีการยอมรับคำตอบที่ให้ต้นทุน

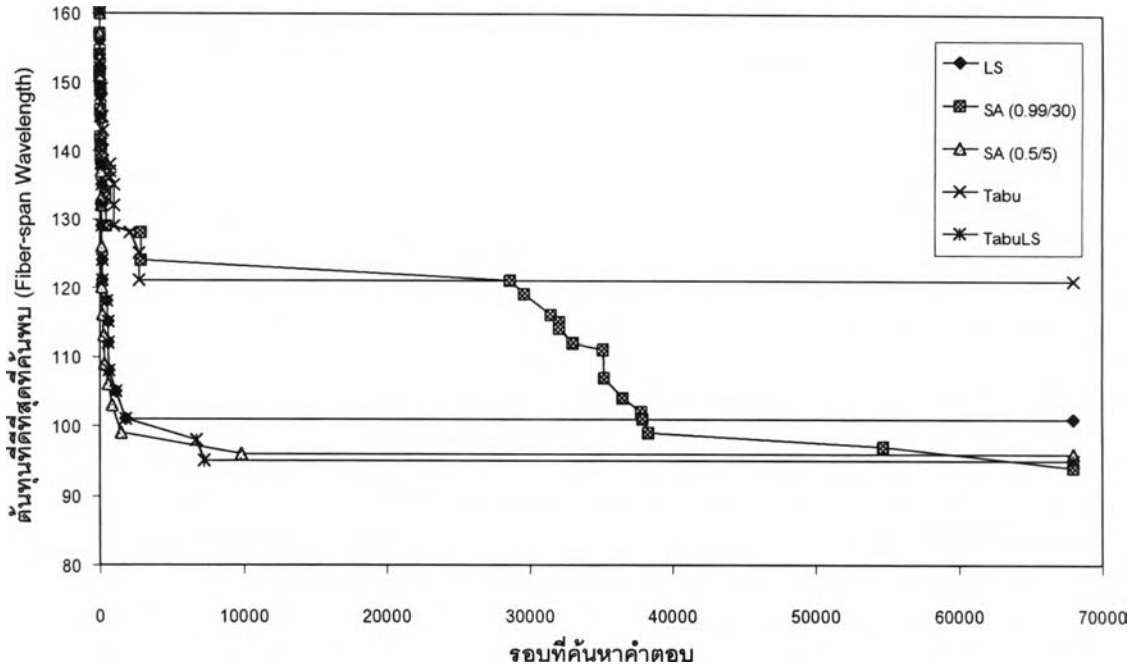
ด้อยลงในบางช่วง ทำให้คำตอบที่ได้จาก TabuLS ไม่ติดอยู่ใน Local Optimum เหมือนใน Local Search โดยเมื่อพิจารณาการกระจายของต้นทุน จะเห็นว่า แม้ Local Search จะให้คำตอบที่มีต้นทุนต่ำได้จำนวนมาก แต่คำตอบส่วนใหญ่ที่ค้นพบจะมีต้นทุนพอ ๆ กัน และมีค่าอยู่ในช่วงเดียว คือ ตั้งแต่ 100-105 Fiber-span Wavelength อัลกอริทึมไม่สามารถค้นพบคำตอบที่ต่ำกว่าช่วงนี้ได้มากนัก แต่หากเป็นแบบ TabuLS จะสามารถค้นพบคำตอบที่ให้ต้นทุนในช่วงที่ต่ำกว่า Local Search คือ ต้นทุนในช่วง 95-100 Fiber-span Wavelength ได้เป็นจำนวนมากขึ้น แสดงว่าฮิวริสติกอัลกอริทึมแบบ TabuLS สามารถแก้ปัญหาที่กระบวนการค้นหาคำตอบติดอยู่ใน Local Optimum ได้ดี โดยให้ผลที่ดีกว่าทั้งแบบ Tabu Search และ Local Search แม้ว่าอาจจะต้องใช้ระยะเวลาค้นหาคำตอบนานกว่า Local Search ก็ตาม

4.4 การเปรียบเทียบต้นทุนที่ได้ระหว่างฮิวริสติกอัลกอริทึมแบบต่าง ๆ

หลังจากการวิเคราะห์ค่าพารามิเตอร์ต่าง ๆ ที่ต้องกำหนดในอัลกอริทึมแล้ว ในส่วนต่อไปจะพิจารณาประสิทธิภาพของฮิวริสติกอัลกอริทึมการปรับปรุงการออกแบบทุกแบบที่นำเสนอในบทที่ 3 ฮิวริสติกอัลกอริทึมเหล่านี้ ได้แก่

- ฮิวริสติกอัลกอริทึมแบบ Local Search (LS)
- ฮิวริสติกอัลกอริทึมแบบ Simulated Annealing (SA)
- ฮิวริสติกอัลกอริทึมแบบ Tabu Search (TS)
- ฮิวริสติกอัลกอริทึมแบบ Tabu Search ที่ประยุกต์ร่วมกับแบบ Local Search (TabuLS)

ในการพิจารณาจะใช้วิธีเปรียบเทียบผลที่ได้จากอัลกอริทึมเหล่านี้ในแง่ต้นทุนและจำนวนรอบในการค้นหาคำตอบ เพื่อวิเคราะห์ประสิทธิภาพของแต่ละวิธี รูปที่ 4.7 แสดงต้นทุนที่ดีที่สุดที่ค้นพบตามจำนวนรอบในการปรับปรุงคำตอบบนโครงข่าย EUROCore ที่ใช้ทราฟฟิกเป็น Uniform ขนาดเป็น 1 และ M=1 และจะหยุดการปรับปรุงการออกแบบเมื่อไม่พบการพัฒนาของคำตอบภายใน 100,000 รอบ



รูปที่ 4.7 ต้นทุนที่ดีที่สุดที่ค้นพบตามจำนวนรอบในการปรับปรุงการออกแบบ

จากรูป ต้นทุนที่ดีที่สุดจากทุกแบบมีลักษณะการลดลงคล้ายกัน คือ ในช่วงแรกต้นทุนจะลดลงอย่างมาก โดยที่ต้นทุนในแบบ LS, $SA_{0.5/5}$ (กรณี factor เป็น 0.5 และ จำนวนรอบ r เป็น 5) และ TabuLS จะมีการลดลงมากกว่าแบบ $SA_{0.99/30}$ (กรณี factor เป็น 0.99, จำนวนรอบ r เป็น 30) และแบบ TS หลังจากนั้นต้นทุนของทุกแบบจะมีค่าค่อนข้างคงตัวหรือลดลงน้อยมากไปตลอดจนสิ้นสุดการค้นหาคำตอบ ยกเว้นแบบ $SA_{0.99/30}$ ที่ต้นทุนจะลดลงอย่างมากอีกครั้งหนึ่งตั้งแต่รอบที่ 30,000-40,000 แต่หลังจากนั้นจะมีค่าลดลงน้อยมากจนยุติกระบวนการ

เมื่อเปรียบเทียบต้นทุนสุดท้ายที่ได้จากทุกแบบ พบว่าต้นทุนจะมีการเรียงลำดับจากค่ามากที่สุดไปน้อยที่สุด คือ TS (ต้นทุนสุดท้ายเป็น 121), LS (98), $SA_{0.5/5}$ (96), TabuLS (95) และ $SA_{0.99/30}$ (94) อัลกอริทึมแบบที่สามารถให้ค่าต้นทุนได้ดีที่สุด คือ SA เมื่อใช้ factor เป็น 0.99 และจำนวนรอบ r เป็น 30 รอบ แต่คำตอบที่ได้มาจะได้มาในช่วงรอบที่สูงมาก คือ ประมาณรอบที่ 68,000 ดังนั้นการใช้อัลกอริทึม SA ที่กำหนดค่าพารามิเตอร์สูง ๆ จึงใช้ได้เฉพาะกรณีที่ต้องการออกแบบโครงข่ายให้มีต้นทุนต่ำโดยไม่คำนึงถึงระยะเวลาในการค้นหาคำตอบเท่านั้น ส่วนฮิวริสติกอัลกอริทึมอีก 3 แบบที่เหลือ คือ TabuLS, $SA_{0.5/5}$ และ LS ต่างก็มีต้นทุนสุดท้ายพอ ๆ กัน โดยฮิวริสติกอัลกอริทึมแบบ TabuLS สามารถให้ค่าต้นทุนต่ำพอ ๆ กับ $SA_{0.99/30}$ ในขณะที่จะได้คำตอบมารวดเร็วกว่ามาก คือ ประมาณรอบที่ 7000 แสดงให้เห็นว่าอัลกอริทึมแบบ TabuLS มีประสิทธิภาพดีสำหรับการออกแบบโครงข่ายในกรณีนี้ ประกอบกับจำนวนพารามิเตอร์น้อยกว่า

SA มาก ทำให้สรุปได้ว่า TabuLS เป็นวิธีที่เหมาะสมที่สุดหากต้องการคำตอบที่ดีภายในเวลาอันรวดเร็ว และ SA เป็นวิธีที่ดีที่สุดในกรณีที่ต้องการคำตอบในกรณีที่ไม่คำนึงถึงระยะเวลาในการค้นหาคำตอบ ส่วนฮิวริสติกอัลกอริทึมอีกแบบที่เหลือ คือ TS พบว่าวิธีนี้ให้ผลการออกแบบที่ไม่ดีนัก โดยต้นทุนจะไม่ลู่ลงสู่ค่าต่ำตลอดช่วงการค้นหา

และจากผลที่ได้จากฮิวริสติกอัลกอริทึมที่ได้พัฒนา เมื่อเปรียบเทียบกับการออกแบบด้วยวิธี Genetic Algorithm (GA) และวิธี Heuristic Algorithm (HA) จาก [13] ซึ่งให้ต้นทุนเป็น 129 และ 108 ตามลำดับ จะพบว่า GA ให้ต้นทุนด้อยกว่าต้นทุนที่ได้จากฮิวริสติกอัลกอริทึมส่วนใหญ่ที่เสนอในวิทยานิพนธ์นี้มาก (ยกเว้น TS) แต่สำหรับวิธี HA พบว่าจะให้ต้นทุนด้อยกว่าฮิวริสติกอัลกอริทึมที่เสนอในวิทยานิพนธ์นี้เล็กน้อย