ใบรับรองวิทยานิพนธ์

บัณฑิตวิทยาลัย มหาวิทยาลัยเกษตรศาสตร์

วิศวกรรมศาสตรมหาบัณฑิต (วิศวกรรมไฟฟ้า)

ปริญญา

วิศวกรรมไฟฟ้า	วิศวกรรมไฟฟ้า
สาขา	**ภาควิชา**

เรื่อง การจ่ายโหลดอย่างประหยัดด้วยวิธีการค้นหาแบบนกกาเหว่า

Economic Dispatch using Cuckoo Search Algorithm

นามผู้วิจัย นางสาวฐิติพร พงศ์กิดาการ

ได้พิจารณาเห็นชอบโดย

อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก _____

(ผู้ช่วยศาสตราจารย์ดุลย์พิเชษฐ์ ฤกษ์ปรีดาพงศ์, Ph.D.)

หัวหน้าภาควิชา _____

(รองศาสตราจารย์วิชัย สุระพัฒน์, วศ.ม.)

บัณฑิตวิทยาลัย มหาวิทยาลัยเกษตรศาสตร์รับรองแล้ว

_____

(รองศาสตราจารย์กัญจนา ธีระกุล, D.Agr.)

คณบดีบัณฑิตวิทยาลัย

วันที่ _____ **เดือน** _____ **พ.ศ.** _____

วิทยานิพนธ์

เรื่อง

การจ่ายโหลดอย่างประหยัดด้วยวิธีการค้นหาแบบนกกาเหว่า

Economic Dispatch using Cuckoo Search Algorithm

โดย

นางสาวฐิติพร พงศ์กิดาการ

เสนอ

บัณฑิตวิทยาลัย มหาวิทยาลัยเกษตรศาสตร์

เพื่อความสมบูรณ์แห่งปริญญาวิศวกรรมศาสตรมหาบัณฑิต (วิศวกรรมไฟฟ้า)

พ.ศ. 2556

ฐิติพร พงศ์กิดาการ 2556: การจ่ายโหลดอย่างประหยัดด้วยวิธีการค้นหาแบบนกกาเหว่า ปริญญาวิศวกรรมศาสตรมหาบัณฑิต (วิศวกรรมไฟฟ้า) สาขาวิศวกรรมไฟฟ้า ภาควิชาวิศวกรรมไฟฟ้า อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก: ผู้ช่วยศาสตราจารย์ดุลย์พิเชษฐ์ ฤกษ์ปรีดาพงศ์, Ph.D. 105 หน้า

วิทยานิพนธ์ฉบับนี้มีวัตถุประสงค์เพื่อนำเสนอการค้นหาแบบนกกาเหว่า (Cuckoo Search, CS) ซึ่งเป็นวิธีการค้นหาค่าเหมาะสมที่สุด มาใช้ในการแก้ปัญหาการจ่ายโหลดอย่างประหยัด (Economic Dispatch) โดยการค้นหาแบบนกกาเหว่าเป็นวิธีการที่เลียนแบบพฤติกรรมการวางไข่ของนกกาเหว่า และลักษณะการเคลื่อนที่ของ Lévy Flights ซึ่งการค้นหาแบบนกกาเหว่านั้นจะมีพารามิเตอร์ที่จะปรับ และจำนวนรอบหรือเวลาในการคำนวณน้อยกว่าในวิธีอื่น ๆ นอกจากนี้ในวิทยานิพนธ์นี้ยังทำการทดสอบเปรียบเทียบผลลัพธ์กับวิธีหาค่าเหมาะสมที่สุดแบบกลุ่มอนุภาค (Particle Swarm Optimization, PSO) โดยพิจารณากรณีศึกษา 2 กรณี คือ กรณีที่ 1 ฟังก์ชันราคาที่ราบเรียบ (Smooth Cost Function) และไม่มีค่าความสูญเสียของระบบ และกรณีที่ 2 ฟังก์ชันราคาที่ไม่ราบเรียบ (Non-Smooth Cost Function) อันเนื่องมาจากผลที่เกิดจากจุดวาล์ว (Valve Point Effect) และพิจารณาค่าความสูญเสียของระบบ และจากการทดสอบพบว่าวิธีการค้นหาแบบนกกาเหว่านั้นจะได้ค่าเชื้อเพลิงรวมน้อยกว่าวิธีหาค่าเหมาะสมที่สุดแบบกลุ่มอนุภาค อีกทั้งยังใช้จำนวนรอบในการคำนวณน้อยกว่าวิธีหาค่าเหมาะสมที่สุดแบบกลุ่มอนุภาค

_____ _____ ___/___/___

ลายมือชื่อนิสิต ลายมือชื่ออาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก

Thitiporn Phongkidakarn 2013: Economic Dispatch using Cuckoo Search Algorithm. Master of Engineering (Electrical Engineering), Major Field: Electrical Engineering, Department of Electrical Engineering. Thesis Avisor: Assistant Professor Dulpichet Rerkpreedapong, Ph.D. 105 pages.

This thesis presents the Cuckoo Search (CS), one of the newest optimization techniques, to solve economic dispatch problems. This search algorithm is based on the obligate brood parasitic behaviour of some cuckoo species and the characteristics of Lévy Flights. Cuckoo Search has fewer parameters to be adjusted and the number of cycles or time to compute than the other methods. Also in this thesis, the results of two case studies are compared with those obtained from the particle swarm optimization (PSO). The first case study uses a smooth cost function with a lossless power system. The second case study uses a non-smooth cost function due to valve point effects with taking into account losses of the system. Finally the results show that the Cuckoo Search has less total fuel cost and the number of cycles or time of computation than the particle swarm optimization.

_____ _____ ___/___/___

Student's signature Thesis Advisor's signature

กิตติกรรมประกาศ

ขอขอบพระคุณ ผศ.ดร.ดุลย์พิเชษฐ์ ฤกษ์ปรีดาพงศ์ อาจารย์ที่ปรึกษาวิทยานิพนธ์ ที่ได้สละเวลาอันมีค่าในการให้คำปรึกษาและคำแนะนำในการค้นคว้าวิจัย ตลอดจนติดตามความก้าวหน้าและตรวจแก้ไขวิทยานิพนธ์จนสำเร็จลุล่วงไปได้ด้วยดี

ขอขอบพระคุณคณาจารย์ภาควิชาวิศวกรรมไฟฟ้าทุกท่าน ที่ได้อบรมสั่งสอนให้ความรู้อันเป็นประโยชน์แก่ผู้วิจัยเป็นอย่างยิ่ง

ขอขอบคุณเพื่อน ๆ นิสิตภาควิชาวิศวกรรมไฟฟ้าทุกท่าน ที่คอยให้กำลังใจ สนับสนุน และช่วยเหลือกันมาโดยตลอด

และสุดท้ายนี้ ขอขอบพระคุณบิดา มารดา และครอบครัวของข้าพเจ้าเป็นอย่างสูง ที่คอยอบรมสั่งสอน เลี้ยงดู เอาใจใส่ ให้กำลังใจ และให้การสนับสนุนผู้วิจัยในทุก ๆ ด้านอย่างดีที่สุดเสมอมา

ฐิติพร พงศ์กิดาการ

เมษายน 2556

สารบัญ

หน้า

สารบัญตาราง

สารบัญภาพ

การจ่ายโหลดอย่างประหยัดด้วยวิธีการค้นหาแบบนกกาเหว่า

Economic Dispatch using Cuckoo Search Algorithm

คำนำ

ในปัจจุบัน พลังงานไฟฟ้านั้นมีความสำคัญมาก เพราะพลังงานไฟฟ้าเป็นสิ่งที่จำเป็นในการดำรงชีวิตของมนุษย์ รวมทั้งในด้านอุตสาหกรรมด้วย แต่การที่จะผลิตพลังงานไฟฟ้านั้นจะต้องมีต้นทุน และต้องใช้ทรัพยากรธรรมชาติมาเป็นเชื้อเพลิง ซึ่งในปัจจุบันทรัพยากรธรรมชาติเริ่มขาดแคลน หายาก และมีราคาที่สูงขึ้นมาก เพราะฉะนั้นจึงต้องพยายามผลิตพลังงานไฟฟ้าให้คุ้มค่ามากที่สุด และวิธีการหนึ่งซึ่งสามารถช่วยลดต้นทุนในการผลิตพลังงานไฟฟ้า ก็คือ การจ่ายโหลดอย่างประหยัด (Economic Dispatch) ปัญหาการจ่ายโหลดอย่างประหยัด เป็นปัญหาที่สำคัญปัญหาหนึ่งในระบบไฟฟ้ากำลัง ซึ่งปัญหาดังกล่าวเป็นปัญหาที่ต้องการหาค่าใช้จ่ายในการผลิตกำลังไฟฟ้าที่ต่ำที่สุดโดยกำลังการผลิตไฟฟ้าที่ได้ต้องเพียงพอต่อความต้องการของระบบ

การแก้ปัญหาการจ่ายโหลดอย่างประหยัดนั้นมีการคิดค้นและพัฒนาวิธีการต่าง ๆ มากมายและเมื่อเร็ว ๆ นี้ได้มีการคิดค้นวิธีการค้นหาค่าที่เหมาะสมที่สุด โดยใช้หลักการของนกกาเหว่าเรียกว่า การค้นหาแบบนกกาเหว่า (Cuckoo Search) ซึ่งการค้นหาแบบนกกาเหว่าเป็นวิธีการที่เลียนแบบพฤติกรรมการวางไข่ของนกกาเหว่า และลักษณะการเคลื่อนที่ของ Lévy Flights

วัตถุประสงค์

1. ศึกษาวิธีการค้นหาแบบนกกาเหว่า ซึ่งเป็นวิธีการที่เลียนแบบพฤติกรรมการวางไข่ของนกกาเหว่า และลักษณะการเคลื่อนที่ของ Lévy Flights

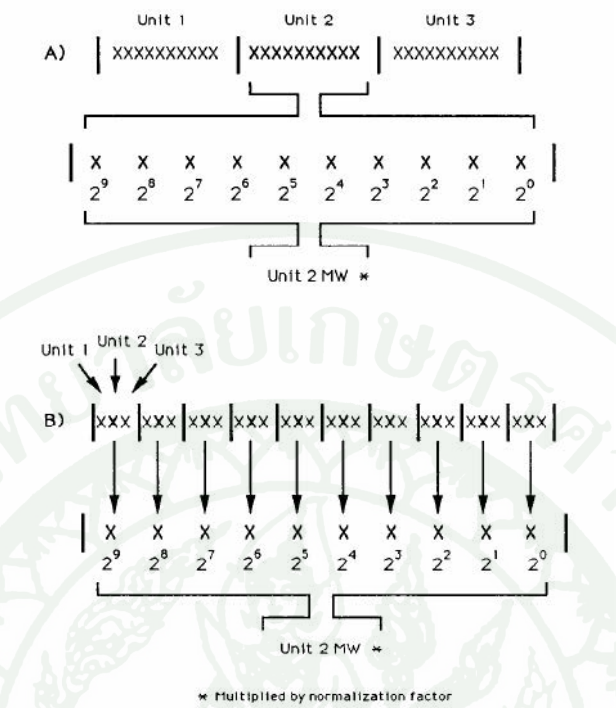
2. นำวิธีการวิธีการค้นหาแบบนกกาเหว่ามาประยุกต์ใช้ในการคำนวณการแก้ปัญหาการจ่ายโหลดอย่างประหยัด

3. ทำการทดสอบเปรียบเทียบผลลัพธ์การจ่ายโหลดอย่างประหยัดด้วยวิธีการค้นหาแบบนกกาเหว่ากับวิธีหาค่าเหมาะสมที่สุดแบบกลุ่มอนุภาค (Particle Swarm Optimization, PSO) โดยพิจารณากรณีศึกษา 2 กรณี คือ กรณีที่ 1 ฟังก์ชันราคาที่ราบเรียบ (Smooth Cost Function) และไม่มีค่าความสูญเสียของระบบ และกรณีที่ 2 ฟังก์ชันราคาที่ไม่ราบเรียบ (Non-Smooth Cost Function) อันเนื่องมาจากผลที่เกิดจากจุดวาล์ว (Valve Point Effect) และพิจารณาค่าความสูญเสียของระบบ

การตรวจเอกสาร

ในปี ค.ศ.1993 Walters และ Sheble ได้นำเอาขั้นตอนวิธีเชิงพันธุกรรม (Genetic Algorithm, GA) มาใช้ในการแก้ปัญหาการจ่ายโหลดอย่างประหยัดที่ฟังก์ชันราคาที่ไม่ราบเรียบ (Non-Smooth Cost Function) อันเนื่องมาจากผลที่เกิดจากจุดวาล์ว (Valve Point Effect) ซึ่งขั้นตอนวิธีเชิงพันธุกรรม นั้นเป็นการหาคำตอบโดยประมาณ โดยอาศัยหลักการจากทฤษฎีวิวัฒนาการจากชีววิทยา และ การคัดเลือกตามธรรมชาติ (natural selection) นั่นคือ สิ่งมีชีวิตที่เหมาะสมที่สุดจึงจะอยู่รอด กระบวนการคัดเลือกได้เปลี่ยนแปลงสิ่งมีชีวิตให้เหมาะสมยิ่งขึ้น (Walters and Sheble, 1993)

ขั้นตอนวิธีเชิงพันธุกรรมเป็นการจำลองทางคอมพิวเตอร์ โดยการแทนคำตอบที่มีอยู่ให้อยู่ในลักษณะ โครโมโซม (chromosomes) แล้วปรับปรุงคำตอบแต่ละชุด (individual) ด้วยวิธีการต่าง ๆ ซึ่งเกี่ยวข้องกับการวิวัฒนาการ (evolutionary operation) การเปลี่ยนแปลงยีนแบบสุ่ม ด้วยตัวปฏิบัติการทางพันธุกรรม (evolutionary operator) เพื่อให้ได้คำตอบที่ดีขึ้น โดยทั่วไปจะแทนคำตอบด้วยเลขฐานสอง (สายอักขระของเลข 0 และ 1) การวิวัฒน์ (evolution) เพื่อหาคำตอบที่เหมาะสมที่สุด (the fitness solution) จะเริ่มจากประชากรที่ได้จากการสุ่มทั้งหมดและจะทำเป็นรุ่น ๆ ในแต่ละรุ่นคำตอบหลายชุดจะถูกสุ่มเลือกขึ้นมาเปลี่ยนแปลง ซึ่งอาจจะทำให้เกิดการกลายพันธุ์หรือสับเปลี่ยนยีนระหว่างกัน จนได้ประชากรรุ่นใหม่ ที่มีค่าความเหมาะสม (fitness) มากขึ้น การวิวัฒน์นี้จะทำไปเรื่อย ๆ จนกระทั่งพบคำตอบที่มีค่าความเหมาะสมตามต้องการ



ภาพที่ 1 การเข้ารหัส (encoding)

ที่มา: Walters and Sheble (1993)

ในปี ค.ศ.1995 Eberhart และ Kennedy ได้เสนอเทคนิคการคำนวณที่มีวิวัฒนาการ (evolutionary computation technique) แบบใหม่ที่มีชื่อเรียกว่า วิธีหาค่าเหมาะสมที่สุดแบบกลุ่มอนุภาค (Particle Swarm Optimization, PSO) ซึ่งมีแนวคิดจากการหาอาหารของนกหรือหาอาหารของปลา และ ทฤษฎีการเคลื่อนที่ (Velocity Theory) ซึ่งต้องอาศัยการเคลื่อนที่กันเป็นกลุ่มและการเว้นระยะห่างระหว่างนกแต่ละตัว ซึ่งในที่นี้ เรียกว่า Particle (Eberhart and Kennedy, 1995)

วิธีหาค่าเหมาะสมที่สุดแบบกลุ่มอนุภาค มีความคล้ายคลึงกับเทคนิคการคำนวณที่มีวิวัฒนาการอื่น ๆ โดยวิธีหาค่าเหมาะสมที่สุดแบบกลุ่มอนุภาคจะใช้ทฤษฎีของการสุ่มสุ่มประชากรเริ่มต้นและใช้การประเมินค่าและการเปลี่ยนแปลงเหล่าประชากรเพื่อค้นหาคำตอบที่เหมาะสมที่สุด (optimal solution) ซึ่งความได้เปรียบของวิธีหาค่าเหมาะสมที่สุดแบบกลุ่มอนุภาค คือ เป็นวิธีที่ใช้งานได้ง่ายและมีพารามิเตอร์ที่ใช้ในการปรับปรุงที่น้อย

ในปี ค.ศ.2002 Attaviriyanupap และคณะ ได้เสนอวิธีไฮบริดวิธีใหม่ในการแก้ปัญหาการจ่ายโหลดอย่างประหยัดแบบพลวัต (dynamic economic dispatch, DED) ที่มีฟังก์ชันราคาที่ไม่ราบเรียบอันเนื่องมาจากผลของจุดวาล์ว โดยได้นำเอาวิธี Evolutionary Programming (EP) แบบง่ายมาใช้ในการค้นหาในระดับเบื้องต้น ซึ่งจะได้ทิศทางที่ดีที่นำไปสู่บริเวณที่เหมาะสมที่สุดโดยรวม และใช้วิธี Sequential Quadratic Programming (SQP) ในการค้นหาเฉพาะที่ ซึ่งใช้ในการปรับคำตอบที่เหมาะสมที่สุดที่คำนวณได้ให้ดีขึ้นในขั้นตอนสุดท้าย (Attaviriyanupap *et al.*, 2002)

ในปี ค.ศ.2004 Sinha และ Purkayastha ได้เสนอวิธีไฮบริดที่เกิดจากนำเอาคุณลักษณะหลักของวิธี Particle Swarm Optimization (PSO) มารวมเข้าด้วยกันกับวิธี Evolutionary Programming (EP) แบบเก่าที่มีการกลายพันธุ์แบบเกาส์เซียน (Classical Evolutionary Programming with Gaussian mutation, CEP) (PSO_CEP) เพื่อใช้ในการแก้ปัญหาการจ่ายโหลดอย่างประหยัดที่มีฟังก์ชันราคาที่ไม่ราบเรียบอันเนื่องมาจากผลที่เกิดจากจุดวาล์ว (Sinha and Purkayastha, 2004)

ในปี ค.ศ. 2004 Victoire และ Jeyakumar ได้เสนอวิธีที่ใช้ในการแก้ปัญหาการจ่ายโหลดอย่างประหยัดที่มีฟังก์ชันราคาที่ไม่ราบเรียบอันเนื่องมาจากผลที่เกิดจากจุดวาล์ววิธีใหม่ที่นำเอาวิธี Particle Swarm Optimization (PSO) มารวมเข้าด้วยกันกับวิธี Sequential Quadratic Programming (SQP) โดยใช้วิธี PSO เป็นวิธีที่ใช้ในหาผลลัพธ์ที่ดีที่สุดหลัก (main optimizer) และใช้วิธี SQP ในการปรับคำตอบที่ได้จากวิธี PSO ในแต่ละรอบการคำนวณให้ดีขึ้น วิธีนี้มีความสามารถในการค้นหาคำตอบที่มีคุณภาพสูง และมีคุณลักษณะการลู่เข้าหาคำตอบที่เร็ว (Victoire and Jeyakumar, 2004)

ในปี ค.ศ.2009 Yang และ Deb ได้เสนอวิธีการค้นหาแบบนกกาเหว่า (Cuckoo Search, CS) มาใช้หาค่าเหมาะสมที่สุด โดยการค้นหาแบบนกกาเหว่าเป็นวิธีการที่เลียนแบบพฤติกรรมการวางไข่ของนกกาเหว่า และลักษณะการเคลื่อนที่ของ Lévy Flights ซึ่งการค้นหาแบบนกกาเหว่านั้นจะมีพารามิเตอร์ที่จะปรับ และจำนวนรอบหรือเวลาในการคำนวณน้อยกว่าในวิธีอื่น ๆ (Yang and Deb, 2009)

ในปี ค.ศ.2010 Yang และ Deb ได้นำวิธีการค้นหาแบบนกกาเหว่า (Cuckoo Search, CS) มาใช้คำนวณหาค่าในงานวิศวกรรม และเขียนโปรแกรมช่วยวิเคราะห์ข้อมูลเชิงวิศวกรรมคณิตศาสตร์ MATLAB กับฟังก์ชันอย่างง่าย (Yang and Deb, 2010)

อุปกรณ์และวิธีการ

อุปกรณ์

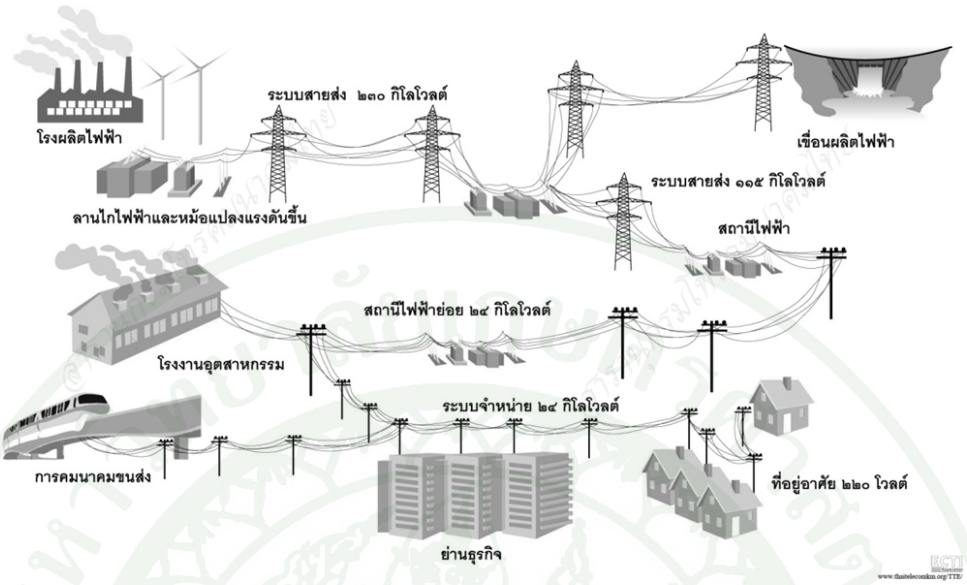
อุปกรณ์ที่ใช้ในการวิจัย

1. เครื่องคอมพิวเตอร์ส่วนบุคคล
2. ระบบปฏิบัติการ Windows 7
3. โปรแกรม Microsoft Office
4. โปรแกรมช่วยวิเคราะห์ข้อมูลเชิงวิศวกรรม คณิตศาสตร์ MATLAB เวอร์ชัน 7.6

วิธีการ

1. ระบบไฟฟ้ากำลัง

ระบบไฟฟ้ากำลัง (Power System) หมายถึง ระบบไฟฟ้าทั้งหมดที่เกี่ยวกับการผลิตไฟฟ้า (Generation System), การส่งจ่ายไฟฟ้า (Transmission System) และการจำหน่ายไฟฟ้า (Distribution System) ระบบไฟฟ้ากำลังเป็นระบบที่น่าศึกษาและเป็นสาขาหนึ่งทางด้านวิศวกรรมไฟฟ้า การศึกษาการทำงานของระบบไฟฟ้ากำลังสามารถแยกออกได้หลายลักษณะ ตั้งแต่การศึกษาการทำงานของระบบกำลัง อุปกรณ์ในระบบกำลัง การวิเคราะห์ระบบกำลัง และการควบคุมระบบกำลัง (ชำนาญ, 2548)



ภาพที่ 2 โครงสร้างระบบไฟฟ้ากำลัง

ที่มา: สมาคมวิชาการไฟฟ้า อิเล็กทรอนิกส์ คอมพิวเตอร์ โทรคมนาคมและสารสนเทศ (2552)

2. ระบบการผลิตไฟฟ้า

ระบบการผลิตไฟฟ้า หมายถึง ระบบที่มีการเปลี่ยนรูปพลังงานในรูปอื่น ๆ มาเป็นพลังงานไฟฟ้า เช่น การเปลี่ยนพลังงานศักย์เป็นพลังงานไฟฟ้าของน้ำที่ถูกกักเก็บในเขื่อน เรียกว่า โรงไฟฟ้าพลังน้ำ, การเปลี่ยนพลังงานความร้อนเป็นพลังงานไฟฟ้าที่ได้จากการเผาไหม้ ทำให้น้ำกลายเป็นไอน้ำ เรียกว่า โรงไฟฟ้าพลังไอน้ำ, ปฏิกิริยานิวเคลียร์ จากไอน้ำที่ได้ถูกปล่อยให้ไปหมุนเทอร์ไบน์และเครื่องกำเนิดไฟฟ้า เรียกว่า โรงไฟฟ้าพลังนิวเคลียร์ และการเปลี่ยนพลังงานความร้อนของแสงอาทิตย์ พลังงานลม พลังงานจากคลื่นในทะเล และอื่น ๆ ให้เปลี่ยนเป็นพลังงานไฟฟ้า (ชำนาญ, 2548)

ระบบการผลิตไฟฟ้าได้ถูกพัฒนาขึ้นมาโดยการเปลี่ยนพลังงานที่มีอยู่ในธรรมชาติให้มาเป็นพลังงานไฟฟ้าเพื่อใช้งานงาน และเมื่อเชื้อเพลิงที่ใช้ในการผลิตไฟฟ้าเริ่มถูกใช้หมดไปเรื่อย ๆ การค้นคว้าและวิจัย เพื่อผลิตพลังงานไฟฟ้าให้คุ้มค่ามากที่สุด และการหาแหล่งพลังงานอื่นมาทดแทนจึงเกิดขึ้น

3. โรงไฟฟ้าพลังความร้อน

โรงไฟฟ้าพลังความร้อน (Thermal Power Plant) คือ โรงไฟฟ้าที่ใช้พลังความร้อนจากเชื้อเพลิงชนิดต่าง ๆ เช่น ถ่านหิน, น้ำมันเตา, ก๊าซธรรมชาติ และนิวเคลียร์ มาต้มน้ำให้เป็นไอความดันสูงไปขับกังหันให้หมุนเป็นพลังงานกล และต่อเพลาเข้ากับเครื่องกำเนิดไฟฟ้า ทำให้เกิดกระแสไฟฟ้าซึ่งมีแรงดันและความถี่ตามที่กำหนดไว้ หรือใช้ความร้อนจากการสันดาปภายในของน้ำมันดีเซลของเครื่องยนต์ดีเซลไปฉุดเครื่องกำเนิดไฟฟ้าให้เกิดกระแสไฟฟ้าได้เช่นกัน

โรงไฟฟ้าพลังความร้อนอาจแยกออกได้เป็นแบบต่าง ๆ ได้ ดังนี้

1) โรงไฟฟ้าพลังความร้อนกังหันไอน้ำ คือ โรงไฟฟ้าที่ใช้ความร้อนจากเชื้อเพลิงชนิดต่าง ๆ ทำให้น้ำกลายเป็นไอน้ำไปขับเครื่องกังหันให้หมุน และให้พลังงานกลออกมาหมุนเครื่องกำเนิดไฟฟ้าเป็นพลังงานไฟฟ้า

2) โรงไฟฟ้ากังหันก๊าซ คือ โรงไฟฟ้าที่ใช้เครื่องกังหันก๊าซเป็นเครื่องยนต์สันดาปภายในโดยอัดอากาศให้มีความดันสูง 8 - 10 เท่า ส่งเข้าห้องเผาไหม้ที่มีก๊าซเป็นเชื้อเพลิง ทำให้เกิดการขยายตัวมีความดันและอุณหภูมิสูงไปดันกังหันให้หมุน โดยกังหันจะอยู่บนแกนเดี่ยวกับเครื่องกำเนิดไฟฟ้า

3) โรงไฟฟ้าพลังความร้อนร่วม คือ โรงไฟฟ้าที่ประกอบด้วยโรงไฟฟ้า 2 ระบบร่วมกัน คือ โรงไฟฟ้ากังหันก๊าซ และโรงไฟฟ้ากังหันไอน้ำโดยนำความร้อนจากไอเสียที่ออกจากเครื่องกังหันก๊าซซึ่งมีอุณหภูมิสูงถึง 550 องศา มาใช้แทนเชื้อเพลิงในการต้มน้ำ โดยต้มน้ำให้เป็นไอไปดันกังหันไอน้ำให้หมุนและต่ออยู่กับแกนเดียวกันของเครื่องกำเนิดไฟฟ้า

4) โรงไฟฟ้าดีเซล คือ โรงไฟฟ้าพลังความร้อนอีกประเภทหนึ่ง ซึ่งใช้น้ำมันดีเซลเป็นเชื้อเพลิง โดยสันดาปภายในร่วมกับการอัดของอากาศเกิดความร้อนและจุดระเบิดต่อเนื่องกันทำ ให้เครื่องยนต์หมุนไปขับเครื่องกำเนิดไฟฟ้า

5) โรงไฟฟ้าพลังนิวเคลียร์ ใช้ความร้อนจากปฏิกิริยานิวเคลียร์เป็นเชื้อเพลิงให้ความร้อนแก่น้ำ เกิดเป็นไอน้ำความดันสูง ไปขับกังหันไอน้ำให้หมุน และไปขับเครื่องกำเนิดไฟฟ้า

3.1 โรงไฟฟ้าพลังความร้อนกังหันไอน้ำ

3.1.1 ส่วนประกอบของโรงไฟฟ้าพลังความร้อนกังหันไอน้ำ อุปกรณ์หลักในโรงไฟฟ้า ได้แก่

1) หม้อกำเนิดไอน้ำ

2) เครื่องกังหันไอน้ำ

3) เครื่องกำเนิดไฟฟ้า

3.1.2 การทำงานของโรงไฟฟ้าพลังความร้อนกังหันไอน้ำ มีขั้นตอนต่าง ๆ ดังนี้คือ

1) เผาไหม้เชื้อเพลิง

2) นำความร้อนไปต้มน้ำให้กลายเป็นไอมีอุณหภูมิและความดันสูง

3) เปลี่ยนพลังงานกลจากเครื่องกังหันไอน้ำไปหมุนเครื่องกำเนิดไฟฟ้า เปลี่ยนเป็นพลังงานไฟฟ้าโดยมีอุปกรณ์ที่ประกอบการทำงานดังนี้ คือ

3.1) หม้อน้ำ (Boiler) หมายถึง ภาชนะที่บรรจุน้ำภายใต้ความดันสูง และน้ำนี้จะถูกทำให้กลายเป็นไอน้ำเมื่อได้รับความร้อนจากการสันดาปของเชื้อเพลิง และไอน้ำจะถูกส่งออกจากหม้อน้ำไปยังท่อไอน้ำเพื่อหมุนกังหันไอน้ำต่อไป เหตุที่ทำให้น้ำที่บรรจุอยู่ในหม้อน้ำมีความดันสูง เพราะว่าจะให้จุดเดือดของไอน้ำสูงขึ้น เพื่อจะให้ได้พลังงานจากไอน้ำมากขึ้น ขณะเดียวกัน น้ำซึ่งมีสภาพเป็นของเหลวจะมีปริมาตรน้อยกว่าน้ำที่มีสภาพเป็นไอจึงจะต้องการเนื้อที่ของหม้อน้ำน้อยกว่า

3.2) เครื่องแยกไอน้ำ (Boiler Drum) เป็นอุปกรณ์ที่แยกไอน้ำออกจากน้ำ ลักษณะเป็นเครื่องแยกไอน้ำรูปแคปซูลที่สามารถทนความดันและอุณหภูมิสูง ภายในเครื่องแยกไอน้ำนี้จะมีอุณหภูมิต่าง ๆ ที่ทำหน้าที่แยกไอน้ำออกจากน้ำโดยอาศัยหลักการแรงหนีศูนย์กลางและการเปลี่ยนทิศการไหล

3.3) แผงท่อรับความร้อน (Economizer) คือ แผงท่อน้ำซึ่งทำให้น้ำที่เข้าไปในหม้อน้ำมีอุณหภูมิสูงขึ้นชั้นหนึ่งก่อน แผงท่อรับความร้อนนี้จะติดตั้งอยู่ช่องสุดท้ายก่อนที่ก๊าซร้อนที่เกิดจากการเผาไหม้จะออกจากตัวหม้อน้ำเพื่อรับความร้อนจากก๊าซร้อน และถ่ายเทให้กับน้ำที่เข้าหม้อน้ำเป็นการเพิ่มประสิทธิภาพกับตัวหม้อน้ำ

3.4) เครื่องอุ่นอากาศ (Air Heater) เป็นอุปกรณ์ที่เพิ่มอุณหภูมิให้กับอากาศก่อนที่อากาศจะเข้าไปช่วยในการเผาไหม้เชื้อเพลิง เครื่องอุ่นอากาศนี้ทำงานโดยการรับความร้อนของก๊าซร้อนที่ออกจากหม้อน้ำและถ่ายความร้อนดังกล่าวให้กับอากาศ ซึ่งจะทำให้อากาศมีอุณหภูมิสูงขึ้น และเป็นการเพิ่มประสิทธิภาพของหม้อน้ำอีกด้วย

การทำงานของระบบหม้อน้ำในโรงไฟฟ้าพลังความร้อน ขั้นแรก น้ำบริสุทธิ์ที่ปราศจากแร่ธาตุ (Deminerize Water) จะถูกสูบเข้าไปสู่หม้อน้ำโดยรักษาระดับน้ำในหม้อน้ำให้เหมาะสม จากนั้นจะจุดเชื้อเพลิงภายในเตา ความร้อนที่เกิดจากการเผาไหม้เชื้อเพลิงจะส่งผ่านไปยังน้ำที่อยู่ในท่อผนังเตา น้ำในท่อผนังเตาจะมีอุณหภูมิสูงขึ้น และเกิดการไหลเวียนพร้อมกับการถ่ายเทความร้อนของน้ำ น้ำจะมีอุณหภูมิสูงขึ้นเรื่อย ๆ จนกลายเป็นไอน้ำทำให้มีความร้อนสูงขึ้นด้วย น้ำที่มีอุณหภูมิสูงจนกลายเป็นไอน้ำจะไหลเข้าสู่แผงท่อไอน้ำเมื่อรับความร้อน จากก๊าซร้อนที่เกิดจากการเผาไหม้เชื้อเพลิงอีกครั้งหนึ่งจนไอน้ำมีอุณหภูมิ สูงขึ้นพอเหมาะ จากนั้นไอน้ำที่มีอุณหภูมิและความดันสูงนี้จะไหลออกจากหม้อน้ำผ่านไปยังท่อน้ำเพื่อไปหมุนกังหันไอน้ำต่อไป

ความร้อนที่เกิดจากการเผาไหม้เชื้อเพลิงภายในเตาจะส่งผ่านไปยังน้ำในท่อผนังเตาด้วยการแผ่รังสีการนำและการพา ส่วนก๊าซร้อนที่เกิดจากการเผาไหม้จะไหลผ่านแผงท่อน้ำไอน้ำและแผงท่อรับความร้อน ซึ่งในช่วงนี้ก็จะถ่ายเทความร้อนให้กับไอน้ำและน้ำในแผงท่อรับความร้อน ทำให้อุณหภูมิของก๊าซร้อนที่ออกจากตัวหม้อน้ำต่ำลง ในขั้นสุดท้ายก่อนที่ก๊าซดังกล่าวจะไหลออกสู่ปล่องวันนั้น ก๊าซจะผ่านไปยังเครื่องอุ่นอากาศ เพื่อถ่ายเทความร้อนให้กับอากาศที่จะมาใช้ในการเผาไหม้ ทำให้อากาศมีอุณหภูมิสูงขึ้น แต่ก๊าซร้อนที่ออกจากเครื่องอุ่นอากาศจะมีอุณหภูมิต่ำลง ทั้งนี้เพื่อเป็นการเพิ่มประสิทธิภาพในการเผาไหม้ และประสิทธิภาพของหม้อน้ำ

3.2 เครื่องกังหันไอน้ำ (Steam Turbine)

เครื่องกังหันไอน้ำเป็นอุปกรณ์ที่ใช้เปลี่ยนพลังงานความร้อนของไอน้ำให้เป็นพลังงานกลเครื่องกังหันไอน้ำประกอบด้วย 4 ส่วนใหญ่ ๆ คือ

1) ระบบควบคุม (Governor System)
2) เพลาหมุนและใบพัด (Rotor & Moving Blade)
3) ตัวถังและใบพัด (Casing & Stationary Blade)
4) เครื่องควบแน่น (Condenser)

การทำงานของเครื่องกังหันไอน้ำ ไอน้ำที่มีอุณหภูมิและความดันสูงจากท่อนำไอน้ำจะไหลเข้าสู่เครื่องกังหันไอน้ำผ่านทางวาล์วของระบบควบคุม เพื่อควบคุมการไหลของไอน้ำที่จะไปหมุนกังหันไอน้ำให้เหมาะสมกับความเร็วรอบหรือภาวะที่ต้องการ จากนั้นไอน้ำก็จะไหลเข้าสู่ตัวกังหันไอน้ำ ซึ่งประกอบด้วยตัวถัง โดยมีเพลาหมุนและใบพัดติดตั้งอยู่ภายในตัวถัง เพลานี้จะถูกรองรับด้วยแบริ่ง (Bearing) เมื่อไอน้ำไหลเข้ามาในตัวกังหันไอน้ำ ความดันของไอน้ำจะลดลงและ

เกิดการขยายตัวของไอน้ำขึ้น การขยายตัวนี้จะทำให้ปริมาตรของไอน้ำเพิ่มขึ้น มีผลให้ความเร็วการไหลของไอน้ำในตัวกังหันสูงขึ้น ไอน้ำที่ความเร็วสูงนี้จะไปปะทะกับใบพัด (Moving Blade) ที่ติดอยู่กับเพลา ทำให้เกิดแรงผลักดันให้เพลาของกังหันหมุน แต่เนื่องจากใบพัดในตัวกังหันไอน้ำได้ถูกออกแบบไว้เป็นชุด ๆ จำนวนหลายชุดติดตั้งอยู่บนเพลาหมุนเดียวกัน ดังนั้นไอน้ำที่ไหลผ่านจากใบพัดชุดแรก จะไหลผ่านใบพัดที่ติดตั้งอยู่กับตัวถัง (Stationary Blade) และไปปะทะกับใบพัดชุดหลัง ๆ ไปเรื่อย ๆ ทำให้เราได้พลังงานในรูปพลังงานกลจากลักษณะการหมุนของเพลากังหันนั่นเอง เมื่อไอน้ำผ่านชุดของใบพัดจนครบ ความดันและอุณหภูมิของไอน้ำจะลดลง ไอน้ำก็จะไหลออกจากกังหันเข้าสู่เครื่องควบแน่น ซึ่งมีลักษณะเป็นห้องสี่เหลี่ยมขนาดใหญ่ สร้างด้วยเหล็กที่มีความแข็งแรงพอที่จะรับการกระแทกของไอน้ำได้ ภายในท่อจะมีท่อโลหะ เช่น ทองเหลืองสอดขวางอยู่เป็นจำนวนมากภายในท่อนี้จะมีน้ำที่ใช้ระบายความร้อนไหลอยู่ ดังนั้น เมื่อไอน้ำไหลเข้าสู่เครื่องควบแน่น ไอน้ำจะถ่ายเทความร้อนผ่านท่อทองเหลืองให้น้ำในท่อและตัวไอน้ำเองก็จะควบแน่น และเปลี่ยนสถานะเป็นน้ำบริสุทธิ์อีกครั้งหนึ่ง ในขณะไอน้ำเปลี่ยนสถานะเป็นน้ำนั้นปริมาตรของไอน้ำจะลดลงอย่างมากทำให้ความดันในเครื่องควบแน่นต่ำกว่าบรรยากาศ ซึ่งเป็นผลในด้านการไหลของไอน้ำและประสิทธิภาพของเครื่องกังหันไอน้ำจะสูงขึ้นด้วย ส่วนน้ำบริสุทธิ์ (Deminerize Water) ที่ได้รับจากการควบแน่นของไอน้ำจะถูกสูบวนกลับเข้าหม้อน้ำอีก การทำงานของระบบกังหันที่กล่าวมานี้เป็นเพียงคร่าว ๆ เท่านั้น ในการทำงานจริงจะมีระบบอื่นๆ อีก เช่น เครื่องอุ่นน้ำ ปั๊มน้ำมันความดันสูง เครื่องดูดอากาศ เป็นต้น ทั้งน้ำเพื่อเพิ่มประสิทธิภาพและความสะดวกในการเดินระบบเครื่องกังหันไอน้ำ

3.3 เครื่องกำเนิดไฟฟ้า

เครื่องกำเนิดไฟฟ้าจะติดตั้งอยู่ในแนวและระดับเดียวกับเครื่องกังหันไอน้ำ โดยเพลาของเครื่องกำเนิดไฟฟ้าต่อโดยตรงเข้ากับเพลาของเครื่องกังหันไอน้ำ ดังนั้น เมื่อเครื่องกังหันไอน้ำหมุนก็จะทำให้เพลาของเครื่องกำเนิดไฟฟ้าหมุนไปด้วย ที่เพลาของเครื่องกำเนิดไฟฟ้ามีตัวนำพันอยู่กับแกนเหล็กไฟฟ้ากระแสตรงจะถูกจ่ายให้กับตัวนำนี้ ดังนั้นจะเกิดสนามแม่เหล็กขึ้นที่เพลาของเครื่องกำเนิดไฟฟ้า เมื่อเพลาของเครื่องกำเนิดไฟฟ้าหมุน สนามแม่เหล็กก็หมุนไปด้วย สนามแม่เหล็กนี้จะหมุนไปตัดกับตัวนำอีกชุดหนึ่งซึ่งพันอยู่กับแกนเหล็กติดอยู่รอบ ๆ ตัวถังของเครื่องกำเนิดไฟฟ้า เมื่อตัวนำนี้ถูกสนามแม่เหล็กจากเพลาหมุนตัด จะเกิดการเหนี่ยวนำ และเกิดกระแสไฟฟ้าไหลในตัวนำที่ติดอยู่กับตัวเครื่องกำเนิดไฟฟ้า กระแสไฟฟ้าที่เกิดขึ้นจะส่งเข้าไปยังหม้อแปลงไฟฟ้าแรงดันสูงเพื่อจ่ายให้กับสายส่งแรงสูงต่อไป

3.4 หม้อแปลงไฟฟ้า

หม้อแปลงไฟฟ้าเป็นอุปกรณ์แรงดันไฟฟ้าที่ส่งมาจากเครื่องกำเนิดไฟฟ้า เป็นแรงดันสูงขนาดต่าง ๆ เช่น 69, 115, 230 kV หรือแรงดันสูงพิเศษ 500 kV การส่งพลังงานไฟฟ้าด้วยแรงดันสูง ๆ จะช่วยลดการสูญเสียพลังงานไฟฟ้าและลดขนาดสายส่งที่ใช้ให้มีขนาดเล็กลง

3.5 เครื่องกำจัดก๊าซซัลเฟอร์ไดออกไซด์

เครื่องกำจัดก๊าซซัลเฟอร์ไดออกไซด์ (Flue Gas Desulfurization, FGD) เป็นเครื่องกำจัดก๊าซหลังจากก๊าซร้อนผ่านเครื่องดักจับ (Precipitation) แล้วจะถูกเป่าผ่านอุปกรณ์ถ่ายเทความร้อนเพื่อลดอุณหภูมิก่อนที่จะฮีทเข้าไปในอุปกรณ์จับก๊าซ (Absorber) ซึ่งเป็นที่ให้น้ำหินปูน (Limestone Slurry) ทำปฏิกิริยาเคมีกับก๊าซซัลเฟอร์ไดออกไซด์กลายเป็นผลึกยิปซัม ก๊าซที่ถูกขจัดซัลเฟอร์ไดออกไซด์แล้วจะมีอุณหภูมิลดลงเหลือ 60 องศา จะถูกดูดออกโดยพัดลมดูดอากาศ (Booster Fan) และถูกปล่อยออกจากปล่อง (Stack) ของโรงไฟฟ้า

3.6 เชื้อเพลิง

เชื้อเพลิงที่ใช้เป็นหลักในเตาหม้อน้ำเพื่อต้มให้เป็นไอน้ำไปขับดันกังหันสำหรับโรงไฟฟ้ามีอยู่ทั้ง 3 สถานะ คือ ของแข็ง, ของเหลว และก๊าซ คือ ถ่านหิน, น้ำมันเตา และก๊าซธรรมชาติ

3.6.1 ถ่านหิน

ถ่านหิน เป็นเชื้อเพลิงที่เกิดจากการทับถมของซากพืชและสัตว์เป็นเวลานานหลายล้านปี มีส่วนประกอบของคาร์บอน, ไฮโดรเจน, ออกซิเจน, กำมะถัน และแร่ธาตุบางชนิดสะสมอยู่

การแบ่งชนิดของถ่านหิน ปกติจะแบ่งตามอายุ วัดจากการเปลี่ยนแปลงคุณภาพของถ่านหินที่เกิดจากการสะสมตัวเป็นเวลานานทำให้ปริมาณธาตุคาร์บอนของถ่านหินเพิ่มขึ้น ปริมาณธาตุไฮโดรเจน ออกซิเจนและความชื้นลดลง แบ่งเป็นดังนี้

1) แอนทราไซด์ เป็นถ่านหินที่มีคุณสมบัติที่จะเป็นเชื้อเพลิงดีที่สุด
2) บิทูมินัส เป็นถ่านหินชั้นรองมาจากแอนทราไซด์
3) ลิกไนต์ เป็นถ่านหินชนิดเลวที่สุดและมีอยู่เป็นจำนวนมาก

3.6.2 น้ำมันเตา

น้ำมันเตาเป็นเชื้อเพลิงเหลวที่เป็นผลผลิตมาจากการกลั่นน้ำมันปิโตเลียมให้ความร้อนและอุณหภูมิสูงเหมาะสำหรับใช้กับตาหม้อน้ำ

3.6.3 ก๊าซธรรมชาติ

ธรรมชาติเป็นก๊าซที่เกิดขึ้นเองตามธรรมชาติ สะสมกันอยู่ใต้ผิวพื้นดินที่เป็นช่องว่างอาจอยู่ใกล้กับสถานที่พบน้ำมันดิบ ก๊าซธรรมชาติประกอบด้วยมีเทน (CH_4) 70% โดยปริมาตรขึ้นไป และมีก๊าซอีเทน (C_2H_6) และก๊าซต่าง ๆ อีก เช่น คาร์บอนไดออกไซด์, ไนโตรเจน และออกซิเจน ซึ่งส่วนย่อยดังกล่าวนี้จะไม่แน่นอนเสมอไป

ก๊าซธรรมชาติจะแบ่งออกเป็น 2 สถานะ คือ การเป็นของเหลวและเป็นก๊าซ

1) ก๊าซธรรมชาติที่เป็นของเหลว ประกอบด้วยสารที่มีน้ำหนักโมเลกุลมากกว่าโพรเทน เช่น บิวเทน และสารไฮโดรคาร์บอนอื่น ๆ ที่มีน้ำหนักโมเลกุลมากกว่า ก๊าซธรรมชาติชนิดนี้ภายหลังที่แยก เอาสารที่เบากว่าไพรเทนออกแล้วจะได้น้ำมันก๊าซโซลีนชนิดหนึ่งแต่ยังไม่เหมาะที่จะใช้เป็นเชื้อเพลิงสำหรับเครื่องยนต์ก๊าซโซลีนเพราะจะเกิดการน็อกในเครื่องยนต์ จึงต้องมีการเติมสารชนิดอื่นเพื่อปรับปรุงคุณภาพเสียก่อน

2) ก๊าซธรรมชาติที่มีสถานะเป็นก๊าซ ประกอบด้วยมีเทนและอีเทน ใช้ในบ้านเรือน ตามโรงงานอุตสาหกรรมและการเกษตรได้โดยตรงพร้อมที่จะเป็นเชื้อเพลิงได้แล้ว เพียงแต่นำมากำจัดพวกสารต่าง ๆ เช่น สารประกอบของกำมะถันออกเสียก่อน

4. การทำ Optimization

4.1 สาเหตุการทำ Optimization

นักวิทยาศาสตร์ที่ต้องการแก้โจทย์ปัญหาทางคณิตศาสตร์ได้คิดค้นการทำ optimization เพื่อให้ได้คำตอบที่เหมาะสมที่สุด ซึ่งปัญหาส่วนใหญ่นั้น เป็นสมการทางด้านคณิตศาสตร์และฟิสิกส์ (Onwubiko, 1999)

ก่อนปี ค.ศ. 1940 วิธีที่ใช้ในการทำ optimization ของฟังก์ชันที่มีหลายตัวแปรยังมีไม่มากนัก เช่น การทำ least square ถูกนำมาประยุกต์ใช้กับปัญหาทางด้านฟิสิกส์ และการทำ newton method ถูกนำมาใช้กับปัญหาทางด้านเคมี

ในช่วงปี ค.ศ. 1940 - 1950 มีการแนะนำสาขาใหม่ทางด้าน optimization ที่เรียกกันว่า กำหนดการเชิงเส้น (linear programming) จากนั้น ได้มีการศึกษาและพัฒนากันมาอย่างต่อเนื่อง จนถึงปัจจุบันนี้ได้มีการนำวิธีการทำ optimization มาใช้อย่างกว้างขว้าง ไม่ว่าจะเป็นทางด้านวิทยาศาสตร์, วิศวกรรมศาสตร์, คณิตศาสตร์ และเศรษฐศาสตร์ เป็นต้น

ในปัจจุบันนี้ การทำ optimization ไม่ได้จำกัดอยู่แค่สาขาวิชาทางด้านวิศวกรรม แต่กลายมาเป็นสิ่งสำคัญในทุก ๆ สาขาวิชาโดยเฉพาะทางด้านเศรษฐศาสตร์ การทำ optimization ถือว่ามีความสำคัญอย่างมาก เนื่องจากมีการนำหลักการของการทำ optimization มาช่วยในเรื่องของเงินทุน และผลกำไร

และเมื่อไม่กี่ปีที่ผ่านมาได้มีการพัฒนาเทคนิคในการทำ optimization ให้มีประสิทธิภาพ และเป็นที่นิยม เรียกว่า modern optimization method ซึ่งเป็นวิธีการที่ไม่มีแบบแผน (nontraditional) และได้มีการนำวิธีนี้เข้าไปร่วมกับสาขาวิชาทางด้านพันธุศาสตร์ เช่น ขั้นตอนวิธีเชิงพันธุกรรม (genetic algorithm), การจำลองการอบเหนียว (simulated annealing), ขั้นตอนวิธีหาค่าเหมาะสมที่สุดแบบกลุ่มอนุภาค (particle swarm optimization) และขั้นตอนวิธีหาค่าเหมาะสมที่สุดด้วยระบบอาณาจักรมด (ant colony optimization)

4.2 หลักการพื้นฐานของการทำ Optimization

การทำ optimization เป็นกระบวนการทางคณิตศาสตร์ ซึ่งให้ผลลัพธ์เชิงปริมาณ (quantity) เนื่องจากผลลัพธ์ที่ได้จะเป็นจำนวนหรือค่าของตัวเลข ดังนั้น ปัญหาที่นำมาใช้ในการทำ optimization จะอยู่ในรูปของแบบจำลองทางคณิตศาสตร์ (mathematical model) ซึ่งโดยทั่วไปแล้วจุดประสงค์ของการทำ optimization คือ เพื่อต้องการหาค่าสูงสุด (maximum) หรือค่าต่ำสุด

(minimum) ของฟังก์ชันวัตถุประสงค์ (objective function) และการหาค่าของฟังก์ชันวัตถุประสงค์บางครั้งอาจจะมีการกำหนดเงื่อนไขต่าง ๆ ที่เรียกว่า ข้อจำกัด (constraints) ดังนั้น สิ่งที่สำคัญสำหรับการทำ optimization คือ การกำหนด objective function และการกำหนด constraints เพื่อใช้ในการหาค่าต่ำสุดหรือสูงสุด

4.2.1 ตัวแปรออกแบบ

ตัวแปรออกแบบ (design variable) หมายถึง ตัวแปรที่เป็นคำตอบของการแก้ปัญหาที่เหมาะสมที่สุด ซึ่งถูกกำหนดเพื่อใช้อธิบายลักษณะของระบบ โดยการกำหนดตัวแปรออกแบบนั้นต้องเลือกตัวแปรที่ใช้กำหนดในระบบให้ถูกต้องกับปัญหาแต่ละประเภท

4.2.2 ฟังก์ชันวัตถุประสงค์

ฟังก์ชันวัตถุประสงค์ (objective function) คือ ฟังก์ชันที่ต้องการหาค่าต่ำสุดหรือสูงสุด โดยจะต้องกำหนดฟังก์ชันวัตถุประสงค์ให้อยู่ในรูปของสมการทางคณิตศาสตร์ที่ติดอยู่ในรูปของตัวแปรออกแบบ เพื่อที่จะทำการหาค่าของตัวแปรที่เป็นจุด maximum หรือ minimum ของฟังก์ชันวัตถุประสงค์นั้น โดยทั่วไปสามารถเขียน Objective function ในรูปของสมการทางคณิตศาสตร์ได้เป็น

$$J = f(x) \tag{1}$$

โดย $x = [x_1, x_2, \dots, x_m]^T$ หรือตัวแปรที่มีจำนวน dimension เป็น m โดยทั่วไปแล้วปัญหาของการทำ optimization ส่วนมากจะเป็นปัญหาในรูปของ minimization problem

โดยทั่วไปสำหรับปัญหาที่นำมาทำ optimization จะมี objective function เพียงฟังก์ชันเดียว แต่ในความเป็นจริงปัญหาที่นำมาทำ optimization อาจมีการกำหนด objective function ได้หลายฟังก์ชัน เรียกว่า multicriteria หรือ multigoal optimization

4.2.3 ข้อจำกัด

ข้อจำกัด (constraints) เป็นเงื่อนไขหรือข้อจำกัดของ objective function ซึ่งโดยทั่วไปแล้ว constraints จะแบ่งเป็น 2 ประเภทด้วยกัน คือ external constraints และ internal constraints โดย external constraints เป็นข้อจำกัดของระบบที่อยู่เหนือการควบคุมของผู้ออกแบบ และ internal constraints เป็นข้อจำกัดที่กำหนดขึ้นโดยผู้ออกแบบระบบ รูปแบบทั่วไปของ constraints เป็นไปดังสมการที่ (2)

$$U_{min} \leq u(t) \leq U_{max} \tag{2}$$

constraints จะมีความสัมพันธ์ต่อตัวแปรที่เลือกไว้ใน objective function ด้วย หมายความว่า ถ้าฟังก์ชันของ constraints มีการเปลี่ยนแปลงค่า คำตอบที่ได้จาก objective function ก็จะมีการเปลี่ยนแปลงค่าด้วยเช่นเดียวกัน ดังนั้น ในการทำ optimization สิ่งสำคัญคือค่าที่ได้จาก objective function จะต้องสอดคล้องกับ constraints ที่กำหนดไว้ การแบ่งประเภทของ constraints สามารถแบ่งได้ตามเครื่องหมายทั้ง หมด 2 ประเภท ได้แก่ equality constraints เป็นเงื่อนไขที่กำหนดด้วยเครื่องหมาย (=) และ inequality constraints เป็นเงื่อนไขที่กำหนดด้วยเครื่องหมาย ($\leq$, $\geq$) สำหรับบางปัญหาที่ไม่ได้มีการกำหนด constraints เอาไว้ จะเรียกปัญหานี้ว่า unconstrained problem

4.2.4 ฟังก์ชันกำลังสอง

ในการทำ optimization ส่วนมากจะพบกับปัญหาที่เป็นแบบ non-linear หรือเป็นปัญหาที่มี objective function ที่อยู่ในรูปสมการที่มีเลขชี้กำลังมากกว่าสอง ซึ่งสามารถทำการประมาณสมการดังกล่าวให้มีเลขชี้กำลังต่ำ ๆ เพื่อง่ายแก่การหาคำตอบได้ โดยถ้าประมาณให้เป็นสมการที่มีเลขชี้กำลังเป็นหนึ่ง สมการดังกล่าวจะเรียกว่า ฟังก์ชันเชิงเส้น (linear function) แต่ถ้าประมาณให้เป็นสมการที่มีเลขชี้กำลังเท่ากับสองแล้ว ก็จะเรียกสมการนี้ว่าเป็น ฟังก์ชันกำลังสอง (quadratic function) โดยแบบฟอร์มทั่วไปของ quadratic function สามารถเขียนได้ดังสมการที่ (3)

$$f(x) = x^T Ax \tag{3}$$

โดยที่ A เป็นเมทริกซ์สมมาตรจำนวนจริง (real symmetric matrix) สำหรับ x ทุกตัวที่เป็นจำนวนจริงที่ไม่ใช่ศูนย์ ในกรณีของจำนวนเชิงซ้อน (complex number) จะเรียกรูปแบบสมการที่ (3) นี้ว่า Hermitian form สามารถเขียนได้ดังสมการที่ (4)

$$f(x) = x^{*}Ax \tag{4}$$

โดยที่ A เป็น Hermitian matrix สำหรับ x ทุกตัวที่เป็นจำนวนเชิงซ้อน และเครื่องหมาย * แสดงถึงการทำ conjugate transpose โดยที่ความแตกต่างของเมทริกซ์ A ใน quadratic form และ hermitian form จะเป็นรูปแบบ ดังนี้

Quadratic Form (Real)

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{12} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ a_{1n} & \cdots & \cdots & a_{nn} \end{bmatrix}$$

Hermitian Form (Complex)

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{12}^{*} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ a_{1n}^{*} & \cdots & \cdots & a_{nn} \end{bmatrix}$$

ภาพที่ 3 เปรียบเทียบความแตกต่างระหว่าง quadratic form และ hermitian form

ที่มา: Onwubiko (1999)

โดยที่ a_{ij}^{*} เป็น complex conjugate เทอมของ a_{ij} เมื่อเขียนฟังก์ชันใด ๆ ให้อยู่ในรูปของ quadratic form แล้ว ลำดับต่อมาจะพิจารณาว่าเมทริกซ์ A เป็น positive definite matrix หรือไม่ โดยนิยามของ positive definite matrix กล่าวไว้ว่า เมทริกซ์สมมาตร A ใด ๆ จะเป็น positive definite ก็ต่อเมื่อ $x^{T}Ax > 0$ สำหรับเวกเตอร์ x ทุกตัวที่ไม่ใช่ศูนย์

คุณสมบัติของ positive definite matrix มีอยู่ด้วยกัน 3 ข้อดังนี้

1) ถ้า K_1 และ K_2 เป็น positive definite matrix แล้ว $K_1 + K_2$ ก็จะเป็น positive definite matrix ด้วย

2) ผลคูณของ positive definite matrix $K = A^TA$ จะเป็น positive definite matrix หรือ positive semidefinite matrix

3) ค่า pivot และ eigenvalue ทุกตัวของ positive definite matrix จะมีค่าเป็นบวกเสมอ

ประเภทของเมทริกซ์ สามารถแบ่งออกได้โดยจัดให้อยู่ในรูป quadratic form หรือดูจากค่า eigenvalue โดย

ถ้า $x^TAx > 0$ สำหรับเวกเตอร์ x ทุกตัวที่ไม่ใช่ศูนย์ เรียก A ว่า Positive definite matrix (eigenvalue ของ A มีค่ามากกว่า 0)

ถ้า $x^TAx \geq 0$ สำหรับเวกเตอร์ x ทุกตัวที่ไม่ใช่ศูนย์ เรียก A ว่า Positive semi-definite matrix (eigenvalue ของ A มีค่ามากกว่าหรือเท่ากับ 0)

ถ้า $x^TAx < 0$ สำหรับเวกเตอร์ x ทุกตัวที่ไม่ใช่ศูนย์ เรียก A ว่า Negative definite matrix (eigenvalue ของ A มีค่าน้อยกว่า 0)

ถ้า $x^TAx \leq 0$ สำหรับเวกเตอร์ x ทุกตัวที่ไม่ใช่ศูนย์ เรียก A ว่า Negative semi-definite matrix (eigenvalue ของ A มีค่าน้อยกว่าหรือเท่ากับ 0)

ถ้า $x^TAx < 0$ หรือ $x^TAx > 0$ สำหรับ x บางตัว เรียก A ว่า Indefinite matrix

สมการโดยทั่วไปของ quadratic function สามารถเขียนให้อยู่ในรูปดังสมการที่ (5) โดยเมื่อทำการหาค่า อนุพันธ์อันดับหนึ่ง (Gradient) ของ quadratic from ก็จะได้สมการที่ (6) และมีอนุพันธ์อันดับสอง (Hessian matrix) ดังสมการที่ (7)

$$f(x) = \frac{1}{2}x^TAx + b^Tx + c \tag{5}$$

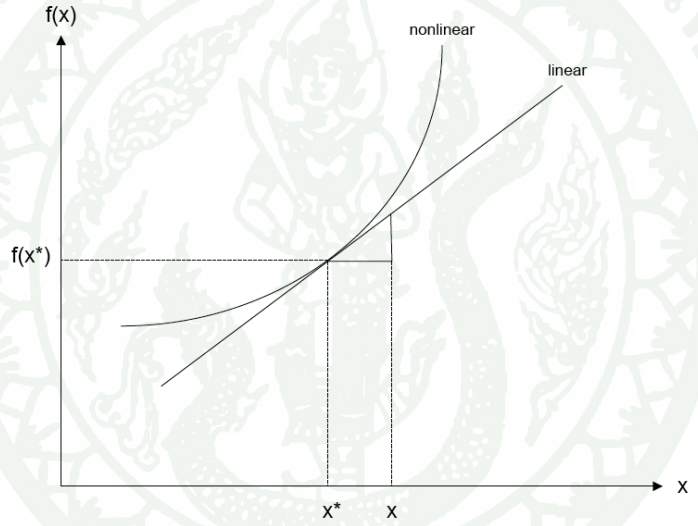
$$\nabla f(x) = Ax + b \tag{6}$$

$$H(x) = \nabla^2 f(x) = \begin{bmatrix} \frac{\partial^2 f(x)}{\partial x_1^2} & \cdots & \frac{\partial^2 f(x)}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f(x)}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f(x)}{\partial x_n^2} \end{bmatrix} = A \tag{7}$$

ในฟังก์ชันใด ๆ ที่เป็น nonlinear สามารถใช้หลักการของ taylor series expansion ประมาณค่าฟังก์ชันที่จุด x^* ได้ดังสมการที่ (8)

$$f(x) \approx f(x^*) + \nabla f(x^*)(x - x^*) + \frac{1}{2!}(x - x^*)^T H(x^*)(x - x^*) \quad (8)$$

โดยการประมาณนั้น หาได้จากการหาค่าของฟังก์ชันที่จุด x^* บวกด้วย first order term ที่เป็นเทอมของสมการเส้นตรง (linear) และบวกด้วย second order term ซึ่งเป็นเทอมของ quadratic โดยที่ตัด high order term หรือเทอมที่มีเลขชี้กำลังมากกว่าสองทิ้งไป ซึ่งสามารถอธิบายหลักการด้วยภาพที่ 4 ดังแสดง



ภาพที่ 4 การทำ linearization โดยใช้ taylor series expansion

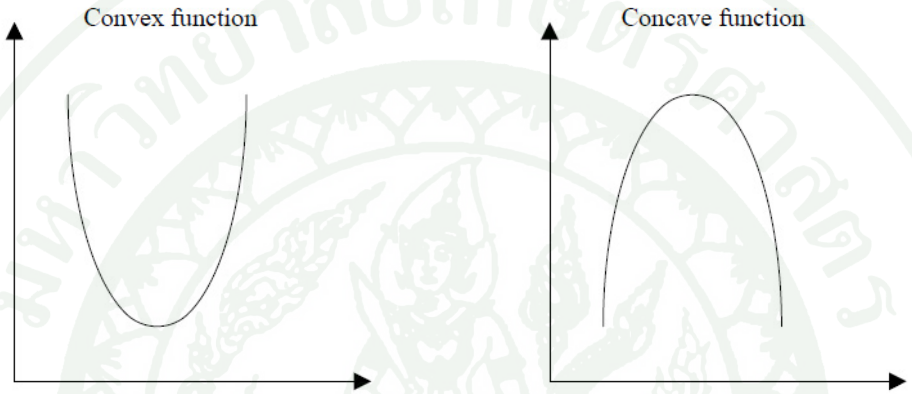
ที่มา: Onwubiko (1999)

4.3 Concave Functions และ Convex Functions

นิยามของ convex function คือฟังก์ชันที่สอดคล้องกับสมการ

$$f(\theta x_1 + (1 - \theta)x_2) \leq \theta f(x_1) + (1 - \theta)f(x_2) \quad (9)$$

โดย x_1 และ x_2 เป็นจุดใด ๆ ที่เป็นสมาชิกของจำนวนจริง $x_1, x_2 \in R$ และ θ มีค่าอยู่ในช่วง $0 \leq \theta \leq 1$ อาจจะพิจารณา convex function โดยดูจาก hessian matrix ได้โดย ฟังก์ชัน f (x) จะเป็น convex function ได้ ต่อเมื่อ hessian matrix ของ f(x) เป็น positive definite หรือ positive semi-definite แต่ถ้า hessian matrix ของ f(x) เป็น negative definite หรือ negative semi-definite จะเรียกฟังก์ชันนั้นว่า concave function



ภาพที่ 5 Convex function และ Concave function

ที่มา: Onwubiko (1999)

คุณสมบัติของ Convex function

1) ทำให้เกิดจุดต่ำสุดในฟังก์ชัน
2) ค่า eigenvalue ของ hessian matrix $\nabla^2 \leq 0$
3) Hessian matrix เป็น positive-definite/semidefinite สำหรับทุก ๆ ค่าของ x

คุณสมบัติของ Concave function

1) ทำให้เกิดจุดสูงสุดในฟังก์ชัน
2) ค่า eigenvalue ของ hessian matrix $\nabla^2 \geq 0$
3) Hessian matrix เป็น negative-definite/semidefinite สำหรับทุก ๆ ค่าของ x

4.4 คุณสมบัติควบคู่

คุณสมบัติควบคู่ (duality) เป็นคุณสมบัติอย่างหนึ่งที่สำคัญของเรื่องกำหนดการเชิงเส้น โดยหลักการพื้นฐานของ duality กล่าวไว้ว่าปัญหาเชิงเส้นใด ๆ สามารถเขียนให้อยู่ในรูปของการหา Maximization หรือ Minimization ได้ โดยในการวิเคราะห์ปัญหานั้นสามารถที่จะเขียนรูปแบบของ duality ขึ้นมาก่อน แล้วจึงใช้วิธี Linear Programming เพื่อหาจุดต่ำสุดหรือสูงสุดต่อไป รูปแบบของ duality โดยทั่วไปเป็นดังนี้

ปัญหาของการหาจุดสูงสุด Maximize $c^T x$

ภายใต้ข้อจำกัด $Ax \leq b$ และ $x \geq 0$

จะเป็น dual กับปัญหาของการหาจุดต่ำสุด Minimize $y^T b$

ภายใต้ข้อจำกัด $y^T A \geq c^T$ and $y \geq 0$

	x_1	x_2	$\cdots$	x_n	
y_1	a_{11}	a_{12}	$\cdots$	a_{1n}	$\leq b_1$
y_2	a_{21}	a_{22}	$\cdots$	a_{2n}	$\leq b_2$
$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$
y_m	a_{m1}	a_{m2}	$\cdots$	a_{mn}	$\leq b_m$
	$\geq c_1$	$\geq c_2$	$\cdots$	$\geq c_n$	

ภาพที่ 6 รูปแบบของ duality

ที่มา: Onwubiko (1999)

จากภาพด้านบน ตัวแปร $x_1, x_2, \ldots, x_n$ จะถูกเรียกว่า primal variable ของปัญหา primal problem ในขณะที่ตัวแปร $y_1, y_2, \ldots, y_m$ ถูกเรียกว่า dual variable ของ dual problem เมื่อเปรียบเทียบ primal problem กับ dual problem จะได้ความสัมพันธ์ ดังนี้

1) ค่าสัมประสิทธิ์ objective function ของ primal problem จะกลายเป็นค่าคงที่ของข้อจำกัดของ dual problem ในขณะเดียวกัน ค่าคงที่ของข้อจำกัดของ primal problem จะกลายเป็นค่าสัมประสิทธิ์ objective function ของ dual problem

2) เครื่องหมายของข้อจำกัดจะเปลี่ยนจาก ใน primal problem ไปเป็น ใน dual problem หรือเปลี่ยนจาก ใน primal problem ไปเป็น ใน dual problem

3) วัตถุประสงค์ของ objective function จะเปลี่ยนจาก maximization ใน primal problem เป็น minimization ใน dual problem

4) จำนวน constraints ใน dual problem จะเท่ากับจำนวนของตัวแปรใน primal problem

5) จำนวนของตัวแปรใน dual problem จะเท่ากับจำนวน constraint ใน primal problem

6) dual ของ dual problem คือ primal problem

4.5 คุณสมบัติความเหมาะสม

โดยทั่วไปปัญหาในการทำ optimization ส่วนใหญ่นั้นจะเป็นการหาค่าต่ำสุด ทำให้จะเน้นอธิบายเพียงหลักการหาค่าจุดต่ำสุดเท่านั้น เนื่องจากเทคนิคต่าง ๆ ที่จะใช้ในการหาค่า Maximization และ Minimization นั้น จะคล้ายกัน และสามารถเปลี่ยนปัญหาที่เป็นแบบ maximization problem ให้เป็นปัญหา minimization problem ได้โดยการใส่เครื่องหมายลบ (-) เข้าไปใน objective function แล้วทำการหาค่า minimum หลังจากนั้น จึงใส่เครื่องหมายลบ (-) ของคำตอบที่ได้จากการทำ minimization อีกที ซึ่งเรียกหลักการนี้ว่า คุณสมบัติความเหมาะสม (optimality) ซึ่งเขียนเป็นสมการได้ดังนี้

$$maximize\ f\ (x) = -minimize[-f(x)] \tag{10}$$

4.5.1 Local Maxima/ Local Minima และ Global Maxima/ Global Minima

คำจำกัดความของ Local maxima และ Global minima นั้น สามารถอธิบายได้จากกราฟภาพที่ 7 จุดที่อยู่สูงกว่าจุดข้างเคียง จะเรียกว่า local maximum และจุดที่อยู่สูงที่สุดของ local maximum ทั้งหมด จะเรียกว่า global maximum ในทางกลับกัน global minimum เป็นจุดที่อยู่ต่ำที่สุดของ local minimum ทั้งหมด ในที่นี้ สามารถเขียนอธิบายในเชิงคณิตศาสตร์ได้เป็น

1) ฟังก์ชัน f(x) จะมี local maximum ที่จุด x^* ก็ต่อเมื่อ

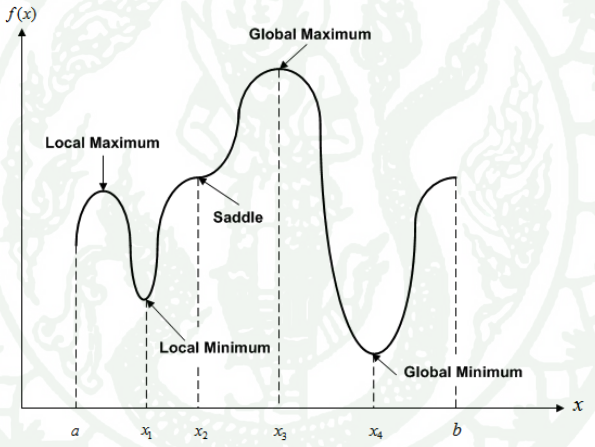
$$f(x) \leq f(x^*)$$

โดยที่จุด x ทุกตัว สอดคล้องกับเงื่อนไข $|x - x^*| < \varepsilon, \varepsilon > 0$

2) ฟังก์ชัน f(x) จะมี local minimum ที่จุด x^* ก็ต่อเมื่อ

$$f(x) \geq f(x^*)$$

โดยที่จุด x ทุกตัว สอดคล้องกับเงื่อนไข $|x - x^*| < \varepsilon, \varepsilon > 0$



ภาพที่ 7 ความสัมพันธ์ของ minima และ maxima

ที่มา: Onwubiko (1999)

4.5.2 Necessary Conditions และ Sufficient Conditions ของ local minima/ local maxima

ฟังก์ชัน f(x) ที่ถูกกำหนดภายใต้ช่วง $a \leq x \leq b$ จะมี relative minimum ที่จุด x^* โดยที่ $a < x^* < b$ ก็ต่อเมื่อมีค่า $\varepsilon > 0$ ใด ๆ ที่ทำให้ $f(x) - f(x^*) \geq 0$ โดย $|x - x^*| \leq \varepsilon$ ทั้งนี้กำหนดให้ว่าสามารถหาค่าอนุพันธ์ของ f (x) ที่จุด x^* ได้

ในที่นี้สามารถพิจารณา Taylor Series Expansion ของเทอม f(x) - f(x^*) ≥ 0

$$f(x) \approx f(x^*) + \nabla f(x^*)(x - x^*) + \frac{1}{2!}(x - x^*)^T H(x^*)(x - x^*)$$

$$f(x) = f(x^*) + \nabla f(x^*)(x - x^*) + \frac{1}{2!}(x - x^*)^T H(x^*)(x - x^*)$$

ถ้า f(x) - f(x^*) ≥ 0 นั่นหมายความว่า $\nabla f(x^*)(x - x^*) + \frac{1}{2!}(x - x^*)^T H(x^*)(x - x^*) \geq 0$ ซึ่งจะเกิดขึ้นได้ 2 กรณี คือ

1) $\nabla f(x^*) = 0$ หรือการเกิด weak minimum ที่จุด x^* (Necessary condition) สามารถกล่าวได้ว่าถ้าจุด x^* จะเป็นจุด local minimum แล้ว $\nabla f(x^*) = 0$ แต่ในทางกลับกัน ถ้า $\nabla f(x^*) = 0$ แล้ว ไม่สามารถกล่าวได้ว่าจะมีจุด local minimum อยู่ที่จุด x^* เพราะ x^* อาจทำให้เกิดจุด maximum หรือ saddle point ก็ได้

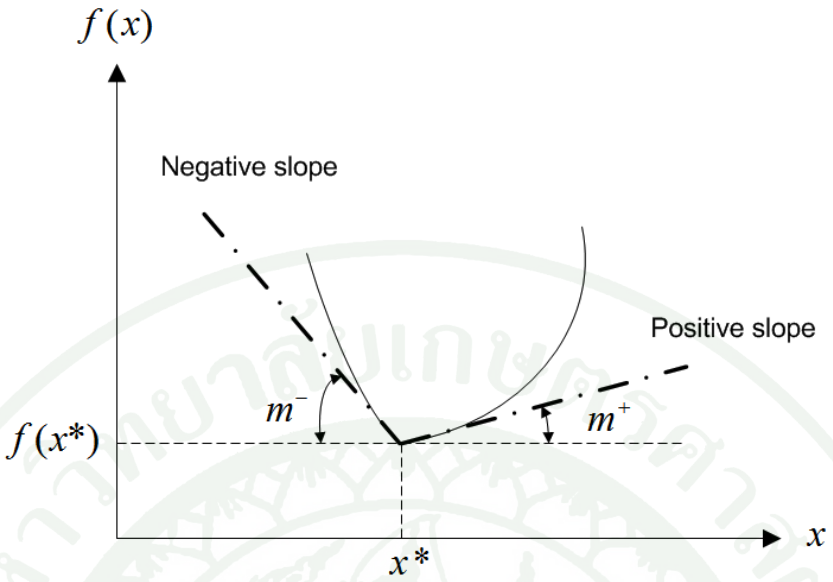
2) $\nabla^2 f(x^*) > 0$ หรือการเกิด strong minimum ที่จุด x^* (Sufficient condition) สามารถกล่าวได้ว่าถ้าจุด x^* จะเป็นจุด local minimum แล้ว $\nabla^2 f(x^*) > 0$ และในทางกลับกัน ถ้า $\nabla^2 f(x^*) > 0$ แล้ว สามารถสรุปได้ว่าจะมีจุด local minimum อยู่ที่จุด x^*

หมายเหตุ

1. ทฤษฎีนี้สามารถนำไปใช้ได้กับจุด x^* ที่เป็น relative maximum
2. ถ้าค่า minimum หรือ maximum เกิดขึ้น ที่จุด x^* โดยที่ไม่สามารถหาค่าอนุพันธ์ของฟังก์ชันได้ดังภาพ จะได้ว่า

$$\lim_{h \to 0} \frac{f(x^* + h) - f(x^*)}{h} = m^+(positive) or\ m^-(negative) \qquad (11)$$

ขึ้นอยู่กับว่าค่า h เข้าใกล้ศูนย์ทางด้านบวกหรือทางด้านลบ เว้นแต่ว่าค่า m^+ และ m^+ มีค่าเท่ากัน จะไม่มีค่าอนุพันธ์ของฟังก์ชัน และถ้าไม่มีค่าอนุพันธ์ของฟังก์ชัน ทฤษฎีนี้ไม่สามารถใช้ได้

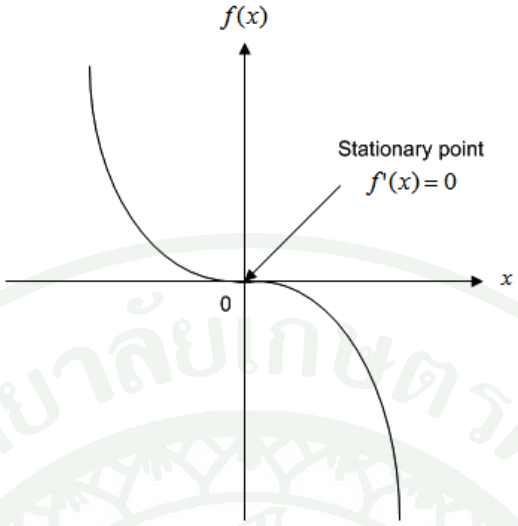


ภาพที่ 8 ฟังก์ชันที่ไม่สามารถหาค่าของอนุพันธ์ได้

ที่มา: Onwubiko (1999)

3. ถ้าค่า minimum หรือ maximum เกิดขึ้นที่จุดสุดท้าย (endpoint) ของช่วงการทำงานของฟังก์ชัน จะได้ของฟังก์ชันที่ค่า h เป็นบวกหรือเป็นลบเท่านั้นดังนั้นจะไม่สามารถหาค่าอนุพันธ์ของฟังก์ชันที่จุดสุดท้ายได้

4. สำหรับฟังก์ชันที่มีค่า minimum หรือ maximum ที่ทุก ๆ จุด โดยที่ค่าอนุพันธ์ของฟังก์ชันเป็นศูนย์ ดังภาพจะเห็นได้ว่า ค่าอนุพันธ์ของฟังก์ชันเป็นศูนย์ที่จุด x = 0 ซึ่งจุดนี้ไม่ได้เป็นทั้ง minimum หรือ maximum โดยทั่วไปแล้วจะเรียก จุดนี้ว่า stationary point



ภาพที่ 9 ฟังก์ชันที่มีค่าของอนุพันธ์เป็นศูนย์ที่จุด x = 0

ที่มา: Onwubiko (1999)

5. การหาค่าเหมาะสมที่สุดของฟังก์ชันหลายตัวแปรแบบไม่มีข้อจำกัด

การหาค่าเหมาะสมที่สุดของฟังก์ชันหลายตัวแปรแบบไม่มีข้อจำกัด (Unconstrained Multivariable Optimization) ในหัวข้อนี้จะขอนำเสนอวิธี Gradient method เพียงวิธีเดียว เนื่องจาก Gradient method เป็นพื้นฐานในการทำ optimization สำหรับฟังก์ชันที่เป็น multivariable อีกทั้งสามารถหาคำตอบของฟังก์ชันได้อย่างรวดเร็ว โดยในที่นี้จะแบ่ง Gradient method ออกเป็น 2 วิธี ได้แก่ Steepest Descent Method และ Newton-Rhapson Method (Onwubiko, 1999)

ในความเป็นจริงแล้ว ปัญหาส่วนใหญ่จะเป็นแบบ constrained problem แต่สาเหตุที่ต้องมีการเรียนรู้ถึงการแก้ปัญหาที่เป็นแบบ unconstrained problem เนื่องจาก

1) ในบางปัญหา ไม่จำเป็นต้องกำหนดเงื่อนไข (constraints)

2) วิธีที่ใช้แก้ปัญหาที่เป็น constrained problem ได้อย่างมีประสิทธิภาพ บางปัญหาจำเป็นต้องใช้วิธีที่ใช้แก้ปัญหาที่เป็น unconstrained problem เข้ามาร่วมด้วย

3) การเรียนรู้เทคนิคของ unconstrained problem จะทำให้เข้าใจหลักการของวิธีที่ใช้ใน constrained problem ได้ง่ายขึ้น

4) วิธีที่ใช้แก้ปัญหาที่เป็น unconstrained problem สามารถใช้แก้ปัญหาทางด้าน engineering ที่มีความซับซ้อนได้

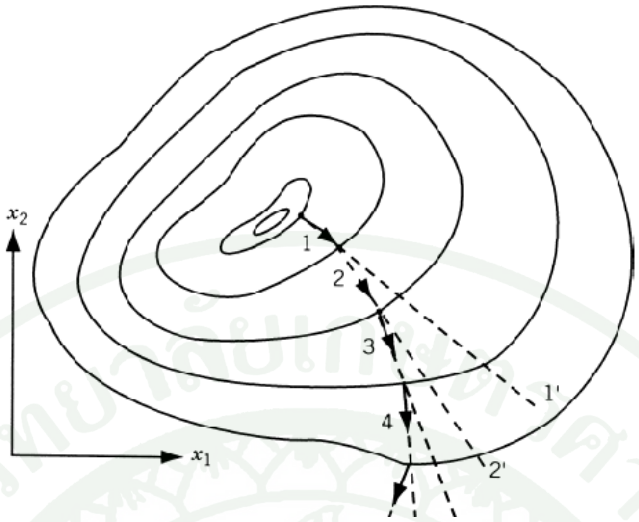
สำหรับปัญหาที่เป็น unconstrained problem จะมีวิธีแก้ปัญหาอยู่หลายวิธีด้วยกัน แต่จะแบ่งได้เป็น 2 วิธีหลัก ๆ คือ direct search method และ descent method สำหรับวิธี direct search method เป็นวิธีที่ใช้หาคำตอบของ objective function โดยไม่ได้มีการหาค่า partial derivative ของฟังก์ชันในกระบวนการหาคำตอบ ซึ่งในบางครั้ง อาจเรียกวิธีนี้ว่า nongradient method สำหรับวิธีที่เป็น descent method เป็นวิธีที่หาคำตอบของฟังก์ชันโดยการนำค่า first derivative ของฟังก์ชันมาช่วยในการหาคำตอบ และในบางกระบวนการอาจมีการใช้ค่าของ second derivative ของฟังก์ชันเข้ามาคิดด้วย โดยทั่วไปแล้ววิธี descent method จะมีประสิทธิภาพมากกว่าวิธี direct search method ส่วนใหญ่จะเรียกวิธี descent methods นี้ว่า gradient method

5.1 การหาค่าเหมาะสมที่สุดด้วยวิธี Gradient method

Gradient ของฟังก์ชันที่มีตัวแปรหลายตัว $\{x_1, x_2,\ldots,x_n\}$ สามารถเขียนให้อยู่ในรูปคอลัมน์เวกเตอร์ได้

$$\nabla f = \begin{bmatrix} \partial f/\partial x_1 \\ \partial f/\partial x_2 \\ \vdots \\ \partial f/\partial x_n \end{bmatrix}_{n\times 1} \tag{12}$$

Gradient เป็นคุณสมบัติที่สำคัญมากในฟังก์ชัน เนื่องจากถ้าทำการเคลื่อนที่ไปยังจุดต่าง ๆ ของฟังก์ชันโดยเคลื่อนที่ไปตามทิศทางของ Gradient จะพบว่าค่าของฟังก์ชันของจุดที่เคลื่อนที่ไปจะมีค่าเพิ่มขึ้น ทิศทางของ gradient direction นี้ถูกเรียกว่า steepest ascent โดยทั่วไปทิศทางของ steepest ascent จะมีการเปลี่ยนแปลงเมื่อมีการเคลื่อนที่จากจุดหนึ่งไปยังอีกจุดหนึ่ง ซึ่งถ้ากำหนดให้มีการเคลื่อนที่ในทิศทางของ steepest ascent ไปเรื่อย ๆ อย่างไม่มีขีดจำกัด จะได้เส้นทางการเคลื่อนที่เป็นเส้นโค้ง ดังภาพ



ภาพที่ 10 ทิศทางของ steepest ascent

ที่มา: Onwubiko (1999)

เนื่องจากเวกเตอร์ gradient จะแสดงถึงทิศทางของ steepest ascent และค่าลบของเวกเตอร์ gradient จะแสดงถึงทิศทางของ steepest descent หรือเมื่อเคลื่อนที่ไปในทิศทางของลบ gradient จะได้ค่าฟังก์ชันที่ลดลงเรื่อย ๆ ดังนั้น การหาค่าจุดต่ำสุดของฟังก์ชันจะสามารถทำได้โดยการเคลื่อนที่ไปในทิศทางตรงกันข้ามกับ gradient จะเห็นได้ว่าไม่ว่า จะหาค่าต่ำสุดหรือสูงสุดของฟังก์ชัน จำเป็นต้องกำหนดทิศทางของ search direction ให้สัมพันธ์กับ gradient ของฟังก์ชันด้วย

Gradient method เป็นกระบวนการทำซ้ำ (Iterative process) โดยทำการกระโดดจากจุดหนึ่ง x^k ไปยังอีกจุดหนึ่ง x^{k+1} ในทิศทางที่ทำให้ค่าของฟังก์ชันมีค่าเพิ่มมากขึ้นหรือลดลงในปัญหาของการหาค่าสูงสุดหรือค่าต่ำสุดตามลำดับ จนกระทั่งได้ค่าเหมาะสมออกมา โดยมีความสัมพันธ์ของการกระโดดในแต่ละครั้งมีสมการเป็น

$$x^{k+1} = x^k + \lambda^k d^k \tag{13}$$

โดยที่ d^k = ทิศทางการเคลื่อนที่ครั้งที่ k (Search direction)

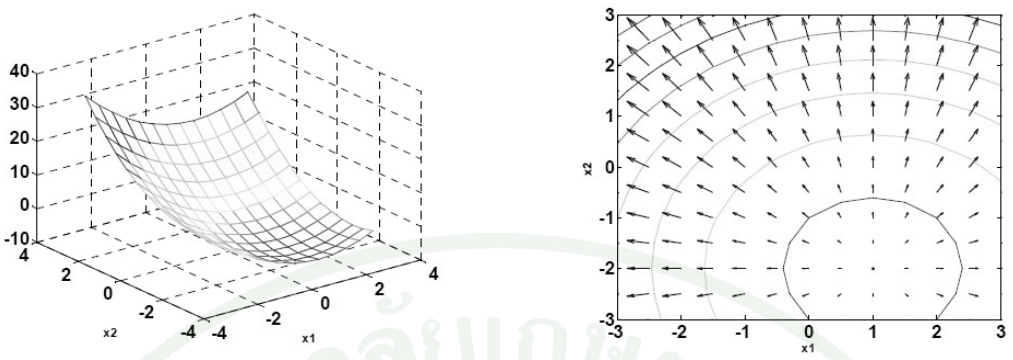
λ^k = ระยะทางที่เคลื่อนที่ครั้งที่ k (Step size)

Gradient method จะแบ่งเป็น 2 กลุ่ม คือ first order method และ second order method ขึ้นอยู่กับฟังก์ชันที่กำหนดให้ว่าสามารถหาค่า first partial derivative หรือ second partial derivative ได้หรือไม่ ในหัวข้อนี้จะนำเสนอ Steepest descent method ที่ต้องใช้ first derivative ของฟังก์ชันมากำหนดทิศทางการเคลื่อนที่ของจุดบนฟังก์ชัน และ Newton-Raphson method ที่ต้องใช้ second derivative ของฟังก์ชันมากำหนดทิศทางการเคลื่อนที่ของจุดบนฟังก์ชัน

ปัญหาที่สำคัญในการทำ gradient method คือ จะกำหนด search direction d และ step size λ ในแต่ละ iteration อย่างไร ถ้า ที่เลือกมีขนาดใหญ่เกินจะทำให้เกิดการลู่ออก (diverge) และไม่สามารถลู่เข้าหาค่าเหมาะสมออกมาได้ แต่ถ้า ที่เลือกมีขนาดเล็กเกินไปจะทำให้ลู่เข้าสู่ค่าเหมาะสมได้ช้า วิธีที่ใช้เลือกค่า step size λ มีอยู่หลายวิธี แต่วิธีที่นิยมใช้กันมากที่สุด คือ การหาค่าของ step size λ ที่ทำให้ objective function มีค่า optimal ที่สุด

5.1.1 Steepest Descent Method

การใช้ค่าลบของเวกเตอร์ gradient เป็นทิศทางในการหาค่า minimum ของฟังก์ชัน ถูกคิดขึ้นครั้งแรกในปี ค.ศ.1847 โดย Cauchy การใช้ steepest descent method วิธีนี้ ก่อนอื่นต้องกำหนดจุดเริ่มต้นขึ้นมาก่อน แล้วเคลื่อนที่ตามทิศทางของ steepest descent สำหรับปัญหาของการหาค่าต่ำสุด จากนั้นทำการ update ค่าไปเรื่อย ๆ จนกระทั่งได้ค่า optimal ของฟังก์ชันออกมา สำหรับปัญหาของการหาค่าสูงสุดนั้น สามารถแปลงให้เป็นปัญหาของการหาค่าต่ำสุดได้ โดยถ้าต้องการหาค่ามากที่สุดของ objective function f(x) สามารถหาจุดที่ทำให้เกิดค่าต่ำที่สุดของ objective function f(x) ดังนั้นในหัวข้อนี้นี้จะอธิบายวิธีการหาค่าต่ำสุดของฟังก์ชัน แล้วใช้วิธีที่กล่าวมาข้างต้นในการหาคำตอบของโจทย์ที่ต้องการหาค่าสูงสุดต่อไป



ภาพที่ 11 ค่าของฟังก์ชันที่เกี่ยวข้องกับ Steepest Descent Method

ที่มา: Onwubiko (1999)

จากกราฟทั้ง 2 ภาพจะเห็นได้ว่ากราฟทางด้านซ้ายมือแสดงถึงค่าของฟังก์ชันเมื่อตัวแปร x_1 และ x_2 มีการเปลี่ยนแปลงค่าไป และต้องการหาค่า x_1 และ x_2 ที่ทำให้เกิดค่าฟังก์ชันต่ำสุด ส่วนกราฟทางด้านขวามือแสดงถึง contour และทิศทางของ Gradient ซึ่งจะสามารถสังเกตได้ว่าทิศทางของ gradient มีทิศทางไปยังตำแหน่งที่ค่าของฟังก์ชันมีค่าสูงขึ้น ดังนั้น ถ้าต้องการหาค่าต่ำสุดของฟังก์ชัน ต้องเคลื่อนที่ไปในทิศทางที่สวนกับทิศทางของ gradient โดยสมการการเคลื่อนที่จากจุดเริ่มต้นไปจุด optimum สามารถแสดงได้เป็น

$$x^{k+1} = x^k - \lambda^k \nabla f(x^k) \tag{14}$$

ขั้นตอนของการค่าต่ำสุดของ objective function สามารถอธิบายได้ดังนี้

1) กำหนดจุดเริ่มต้น x^0 และค่า tolerance ε โดยกำหนดให้ k = 0

2) หาค่า Gradient ของ objective function แล้วแทนค่า จุด x^k ลงไป

3) ตรวจสอบว่า $\nabla f(x^k) \leq \varepsilon$ หรือไม่ ถ้าสอดคล้องกับข้อกำหนดจึงตอบคำตอบของฟังก์ชันที่จุด x^k แต่ถ้าไม่สอดคล้องกับข้อกำหนดให้ทำข้อถัดไป

4) ทำการ update ค่า x^{k+1} ตามสมการที่ (14) โดยติดค่าของ λ^k เอาไว้

5) หาค่า λ^k ที่ทำให้ first derivative ของ $f(x^{k+1})$ มีค่าต่ำที่สุด แล้วแทนค่า λ^k ลงในสมการที่ (14) อีกครั้งหนึ่งซึ่งจะได้ค่า x^{k+1} ที่เป็นตัวเลขออกมา

6) เพิ่มค่า k = k + 1 และเริ่มทำข้อ 2 ใหม่

5.1.2 Newton-Raphson method

ข้อเสียของ steepest descent method คือ สามารถหาจุด optimum ได้ช้า เนื่องจากจุดที่ได้จากการใช้ gradient ของฟังก์ชันเป็นการประมาณเชิงเส้นของ objective function เท่านั้น ทำให้ไม่มีประสิทธิภาพในการเข้าใกล้ค่า optimum ได้ จากเหตุผลนี้จึงมีการใช้ second partial derivative ของ objective function ในการประมาณ โดยจะมีสมการคล้ายกับสมการที่ (13) เมื่อพิจารณา Taylor series expansion ของ objective function $f(x)$ ที่จุด $x + \Delta x$ จะได้

$$f(x + \Delta x) \approx f(x) + \nabla^T f(x)\Delta x + 0.5(\Delta x)^T \nabla^2 f(x)\Delta x \tag{15}$$

หาค่า optimum ของฟังก์ชัน จากการ take derivative สมการที่ (15) เทียบกับ Δx และให้เท่ากับ 0 จะได้

$$g + H\Delta x = 0 \tag{16}$$

โดยที่ $g = \nabla f(x)$
$H = \nabla^2 f(x)$

ถ้า H เป็น nonsingular matrix หรือนั่นคือ สามารถหาค่า inverse ของ H ได้ ดังนั้น สามารถเขียนสมการที่ (16) ได้เป็น

$$\Delta x = -H^{-1} g \tag{17}$$

ดังนั้น จึงใช้สมการที่ (17) เป็น search direction ที่ต้องเคลื่อนที่ไป

$$x^{k+1} = x^k - \lambda^k (H^{-1})^k g^k \tag{18}$$

$$x^{k+1} = x^k - \lambda^k (H^{-1})^k \nabla f(x^k) \tag{19}$$

ขั้นตอนในการทำ Newton-Raphson method มีดังนี้

1) กำหนดจุดเริ่มต้น x^0 และค่า tolerance ε และกำหนด k = 0

2) หาค่า gradient g^k ที่จุด x^k ถ้า $\|g^k\| \leq \varepsilon$ แล้ว จะได้คำตอบของฟังก์ชันที่จุด x^k

3) ถ้า $\|g^k\| > \varepsilon$ หาค่า Hessian matrix (H^k) และ $(H^{-1})^k$ (Hessian matrix จะเป็น positive definite matrix เมื่อต้องการหาจุดต่ำสุด)

4) หา search direction จากสมการ

$$d^k = -(H^{-1})^k g^k \tag{20}$$

5) กำหนดให้ $x^{k+1} = x^k + \lambda^k d^k$ โดยแทนค่า d^k จากสมการที่ (20) จะได้ $x^{k+1} = x^k - \lambda^k (H^{-1})^k g^k$ และหาค่า λ^k ที่ทำให้ objective function มีค่าต่ำสุด จากนั้นกำหนดให้ k = k + 1 และเริ่มทำข้อ 2 ใหม่

6. การหาค่าเหมาะสมที่สุดของฟังก์ชันหลายตัวแปรแบบมีข้อจำกัด

การหาค่าเหมาะสมที่สุดของฟังก์ชันหลายตัวแปรแบบมีข้อจำกัด (Constrained Multivariable Optimization) ในหัวข้อนี้จะกล่าวถึงการทำ optimization ของปัญหาที่เป็น nonlinear constraint ซึ่งเป็นปัญหาที่พบบ่อยในด้านวิศวกรรม เนื่องจากโดยปกติแล้วปัญหาส่วนใหญ่จะมีเงื่อนไขที่มีองค์ประกอบหลายตัว ในการทำ optimization สำหรับปัญหาที่เป็น constrained จะมีการเพิ่มเติมในส่วน algorithm ของปัญหาที่เป็น unconstrained ด้วย และในหัวข้อนี้จะพิจารณา constrained เป็น 2 แบบด้วยกัน คือ equality constraints หรือเงื่อนไขที่มีเครื่องหมายเท่ากับ และ inequality constraints หรือเงื่อนไขที่มีเครื่องหมายน้อยกว่าหรือเท่ากับ หรือมากกว่าหรือเท่ากับ สำหรับปัญหาที่เป็น equality constraints จะใช้เทคนิคที่เรียกว่า Jacobain method และ Lagrange's method ในการแก้ปัญหา และสำหรับปัญหาที่เป็น inequality constraints จะใช้เทคนิคที่เรียกว่า Karush Kuhn Tucker ในการแก้ปัญหา (Onwubiko, 1999)

6.1 ฟังก์ชันแบบมีข้อจำกัด (Constrained Problem)

ในหัวข้อนี้จะอธิบายถึงวิธีการหรือเทคนิคที่ใช้ในการหาคำตอบของฟังก์ชั่นที่มี constrained โดยรูปแบบของปัญหาที่เป็น conhstrained problem คือ การหาค่า x ที่ทำให้คำตอบของฟังก์ชัน f(x) มีค่าน้อยที่สุด โดยกำหนดเงื่อนไขดังนี้

$$g_{\mathrm{i}}(\mathrm{x}) \leq 0, \;\; i = 1, 2, \ldots, m$$

$$h_k(x) = 0, \;\; k = 1, 2, \ldots, n$$

6.2 คุณสมบัติของฟังก์ชันแบบมีข้อจำกัด

คุณสมบัติของฟังก์ชันที่มี constrained มีทั้งหมด 4 ข้อด้วยกัน ดังนี้

1) เงื่อนไขที่กำหนดอาจไม่ส่งผลต่อการเปลี่ยนแปลงคำตอบที่เหมาะสม (optimum point) ของฟังก์ชัน หมายความว่า ค่า minimum ของฟังก์ชันที่เป็น constrained problem และ unconstrained problem อาจเป็นค่าเดียวกัน ซึ่งในกรณีนี้ค่า minimum point สามารถหาได้โดยใช้ necessary conditions และ sufficient conditions

$$\nabla f|_{x^*} = 0$$

$$J_{x^*} = \left[\frac{\partial^2 f}{\partial x_i \partial x_j}\right]_{x^*} = \text{positive definite}$$

2) ในกรณีที่คำตอบที่เหมาะสมของฟังก์ชันอยู่บนขอบเขตของเงื่อนไขที่กำหนด พิจารณาจาก Khun – Tucker necessary conditions จะเห็นว่าค่าลบของ gradient ของฟังก์ชันจะเป็น positive linear combination ของ gradient ของ constraints

3) ถ้า objective function ที่กำหนดมี local minima ที่เป็น unconstrained มากกว่า 2 จุด เมื่อพิจารณาปัญหาที่เป็น constrained problem อาจจะได้ค่า minimum หลายค่า

4) ในบางกรณีแม้ว่า objective function ที่กำหนดมีค่า minimum สำหรับ unconstrained

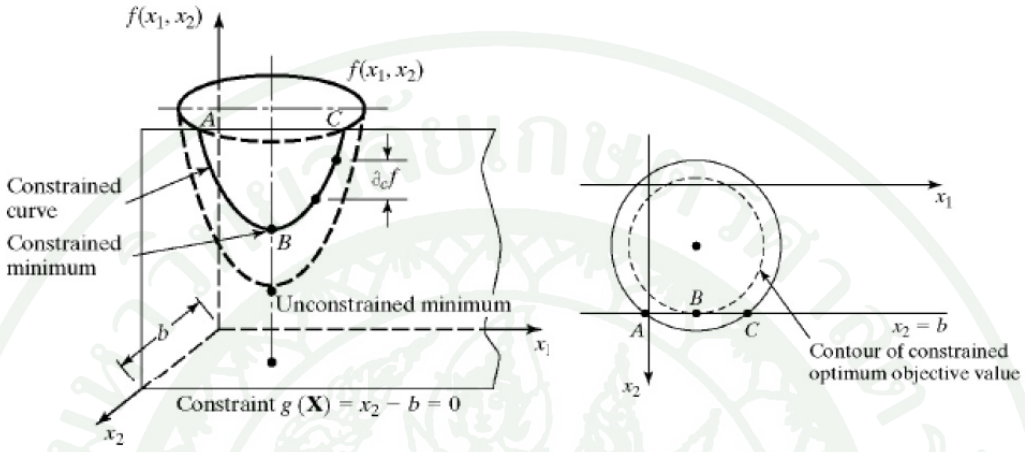
เพียงค่าเดียว แต่สำหรับ constrained อาจจะมีค่า local minima ได้หลายค่า

6.3 การหาค่าเหมาะสมที่สุดด้วยวิธี Jacobian Method

หลักการของ Jacobian method จะใช้อนุพันธ์ฟังก์ชันของ constrained ในการหาอนุพันธ์ของฟังก์ชัน f(x) ครั้งที่ 1 ที่สอดคล้องกับ constrained g(x) = 0 โดยมีรูปแบบของฟังก์ชัน เช่น หาค่า minimum ของฟังก์ชัน z = f(x) และกำหนดให้ constrained g(x) = 0 โดยที่

$$x = (x_1, x_2, \dots, x_n) \tag{21}$$

$$g = (g_1, g_2, \dots, g_m)^T \tag{22}$$



ภาพที่ 12 ขอบเขตของปัญหาที่มีข้อจำกัดและไม่มีข้อจำกัด

ที่มา: Onwubiko (1999)

จากภาพนี้จะเห็นได้ว่า ถ้าฟังก์ชันที่ไม่มีการกำหนดเงื่อนไข คำตอบที่ได้จะเป็นเส้นประ แต่เมื่อมีการกำหนดเงื่อนไขให้กับฟังก์ชันแล้ว ช่วงของค่าที่ได้จะเป็นเส้นทึบ ซึ่งมีช่วงแคบลง เมื่อทำการพิจารณา gradient ของฟังก์ชัน ด้วย Taylor's theorem จะได้สมการของฟังก์ชันและconstrained เป็น

$$f(x + \Delta x) - f(x) = \nabla f(x)\Delta x + o(\Delta x_j^2) \tag{23}$$

$$g(x + \Delta x) - g(x) = \nabla g(x)\Delta x + o(\Delta x_j^2) \tag{24}$$

โดยที่ $o(\Delta x_j^2)$ เป็นเทอมของ high order ซึ่งจะสมมติให้ $o(\Delta x_j^2) \to 0$ ดังนั้นจะสามารถลดรูปของสมการที่ (23) และ (24) ได้เป็น

$$\partial f(x) = \nabla f(x)\partial x \tag{25}$$

$$\partial g(x) = \nabla g(x)\partial x \tag{26}$$

กำหนดให้สมการของ constraints $g(x) = 0$ และ $\partial g(x) = 0$ ดังนั้น จากสมการที่ (25) และ (26) จะได้สมการใหม่เป็น

$$\partial f(x) - \nabla f(x)\partial x = 0 \tag{27}$$

$$\nabla g(x)\partial x = 0 \tag{28}$$

ถ้ากำหนดให้ x = (y, z) โดยที่ y = (y_1, y_2,…,y_m) เป็น dependence variable หรือตัวแปรที่ขึ้นอยู่กับตัวแปรอื่นและ z = (z_1, z_2,…,z_n) เป็น independence variable หรือตัวแปรที่ไม่ขึ้น อยู่กับตัวแปรอื่น สามารถเขียน gradient vector ของฟังก์ชั่นและ constrained ได้เป็น

$$\nabla f(y,z) = (\nabla_{\mathrm{y}}\mathrm{f}, \nabla_{\mathrm{z}}\mathrm{f}) \tag{29}$$

$$\nabla g(y,z) = (\nabla_{\mathrm{y}}\mathrm{g}, \nabla_{\mathrm{z}}\mathrm{g}) \tag{30}$$

จะได้ Jacobian matrix (J) และ control matrix (C) เป็น

$$J = \nabla_y g = \begin{pmatrix} \nabla_y g_1 \\ \nabla_y g_2 \\ \vdots \\ \nabla_y g_m \end{pmatrix} \tag{31}$$

$$J = \nabla_{\mathrm{z}} g = \begin{pmatrix} \nabla_z g_1 \\ \nabla_z g_2 \\ \vdots \\ \nabla_z g_m \end{pmatrix} \tag{32}$$

จากสมการที่ (27) และ (28) สามารถเขียนในรูปสมการใหม่ได้เป็น

$$\partial f(y,z) = \nabla_y f\partial y + \nabla_z f\partial z \tag{33}$$

$$J\partial y = -C\partial z \tag{34}$$

ถ้า Jacobian matrix เป็น nonsingular matrix จากสมการที่ (34) จะได้

$$\partial y = -J^{-1}C\partial z \tag{35}$$

$$\partial f(y,z) = (\nabla_z f - \nabla_y f J^{-1}C)\partial z \tag{36}$$

จากสมการที่ (36) สามารถนิยาม constrained gradient ของฟังก์ชัน f ที่สอดคล้องกับ z ได้เป็น

$$\nabla_c f = \frac{\partial_c f(y,z)}{\partial_c z} = \nabla_z f - \nabla_y f J^{-1}C \tag{37}$$

ในการศึกษาเรื่องของ sensitivity จะทำการศึกษาผลของการเปลี่ยนแปลงค่าทางด้านขวามือของสมการ constraints ที่มีต่อค่า optimal value กล่าวคือ ถ้าสามารถหาค่า optimal value จากสมการ constraints $g(x) = 0$ ออกมา และเรียกค่าดังกล่าวว่าค่า f แล้ว จะพิจารณาว่าถ้าสมการ constraints เปลี่ยนแปลงค่าไปเป็น $g_i(x) = \partial g_i$ แล้ว ค่า f จะมีการเปลี่ยนแปลงค่าไปเป็นอย่างไร ซึ่งในการวิเคราะห์ sensitivity analysis ในโจทย์ที่เป็น nonlinear programming นี้ สามารถทำได้เฉพาะการเปลี่ยนแปลงค่าน้อย ๆ ใกล้ ๆ กับจุด optimal value เท่านั้น

จากสมการดังกล่าวดังข้างต้น ถ้า $g_i(x) \neq 0$ สามารถเขียนใหม่ได้เป็น

$$\partial f(y,z) = \nabla_y f\partial y + \nabla_z f\partial z$$

$$\partial g(x) = J\partial y + C\partial z$$

กำหนดให้ $\partial g(x) \neq 0$ ดังนั้น

$$\partial y = J^{-1}\partial g - J^{-1}C\partial z$$

แทนค่าที่ได้ลงใน $\partial f(y,z)$ จะได้

$$\partial f(y,z) = \nabla_y f J^{-1}\partial g + \nabla_c f\partial z$$

เมื่อ

$$\nabla_c \mathrm{f} = \nabla_z \mathrm{f} - \nabla_y \mathrm{f} J^{-1} C$$

ที่จุด optimal point $x_0 = (y_0, z_0)$ซึ่งจะพิจารณาว่าระบบเป็นแบบ stationary ที่จุดนี้ ทำให้ค่า constrained gradient $\nabla_c \mathrm{f}$ มีค่าเป็นศูนย์ ดังนั้น จะได้ว่า

$$\nabla \mathrm{f}(y_0, z_0) = \nabla_{y_0} \mathrm{f} J^{-1} \partial g(y_0, z_0) \tag{38}$$

หรือ

$$\frac{df}{dg} = \nabla_{y_0} \mathrm{f} J^{-1} \tag{39}$$

สมการที่ (39) จะถูกเรียกว่า sensitivity coefficients ซึ่งคือปริมาณที่จะบอกว่าการเปลี่ยนแปลงค่าทางด้านขวามือของสมการ constraint g จะส่งผลต่อการเปลี่ยนแปลงของ f ไปมากน้อยขนาดไหน

6.4 การหาค่าเหมาะสมที่สุดด้วยวิธี Lagrange Method

Lagrange method เป็นอีกวิธีการหนึ่งที่ใช้ในการแก้ปัญหาสำหรับ constrained problem โดยใช้อนุพันธ์ของฟังก์ชันครั้งที่หนึ่ง (first derivative) และความสำคัญของตัวแปร ถ้ากำหนดให้ objective function ที่ต้องการ optimize นั้น เป็น $f(x_1, x_2)$ และกำหนดเงื่อนไขที่เป็น equality constraint เป็น

$$g(x_1, x_2) = 0 \tag{40}$$

โดยที่ x_1 เป็นฟังก์ชันของ x_2 ซึ่งมีความสัมพันธ์ระหว่าง x_1 และ x_2 เป็น

$$x_1 = \emptyset(x_2) \tag{41}$$

ดังนั้น จากสมการที่ (41) สามารถเขียนฟังก์ชันของ x_2 ได้เป็น

$$F(x_2) = f(\emptyset(x_2), x_2) \tag{42}$$

และเงื่อนไขของ x_2 คือ

$$h(x_2) = g(\emptyset(x_2), x_2) = 0 \tag{43}$$

จากฟังก์ชัน $F = f(x_1, x_2)$ สามารถหาอนุพันธ์ของฟังก์ชันได้ดังนี้

$$dF = \frac{\partial f}{\partial x_1} dx_1 + \frac{\partial f}{\partial x_2} dx_2 \tag{44}$$

จากสมการที่ (44) จะได้อนุพันธ์ของฟังก์ชันเทียบกับ x_1 และ x_2

$$\frac{dF}{dx_1} = \frac{\partial f}{\partial x_1} + \frac{\partial f}{\partial x_2}\frac{dx_2}{dx_1} \tag{45}$$

$$\frac{dF}{dx_2} = \frac{\partial f}{\partial x_1}\frac{dx_1}{dx_2} + \frac{\partial f}{\partial x_2} \tag{46}$$

จากสมการที่ (42) และ (43) สามารถหาคำตอบที่เป็น minimum ของ objective function โดยกำหนดให้อนุพันธ์ของฟังก์ชัน $F(x_2)$ และอนุพันธ์ของเงื่อนไข $h(x_2)$ เท่ากับศูนย์ จะได้

$$\frac{dF(x_2)}{dx_2} = \frac{\partial f}{\partial x_2} + \frac{\partial f}{\partial x_1}\frac{d\emptyset(x_2)}{dx_2} = 0 \tag{47}$$

$$\frac{dh(x_2)}{dx_2} = \frac{\partial g}{\partial x_2} + \frac{\partial g}{\partial x_1}\frac{d\emptyset(x_2)}{dx_2} = 0 \tag{48}$$

จากกฎลูกโซ่ สามารถกำหนดได้

$$\frac{df}{dx_1} - \left(\frac{\frac{df}{dx_1}}{\frac{dg}{dx_1}}\right)\frac{dg}{dx_1} = 0 \tag{49}$$

$$\nabla f - \lambda \nabla g = 0 \tag{50}$$

$$\lambda = \frac{df}{dg} \tag{51}$$

โดยที่ λ เป็น Lagrange multiplier ดังนั้น จะได้ Lagrange function (Lagrangean) เป็น

$$L(x,\lambda) = f(x) - \lambda g(x) \tag{52}$$

และจะได้ necessary condition จาก gradient ของ Lagrangean เท่ากับศูนย์

$$L(x,\lambda) = 0 \tag{53}$$

ผลคูณลากรองจ์ (Lagrange Multiplier)

Lagrange multiplier เป็นตัววัดการเปลี่ยนแปลงของ objective function ที่สอดคล้องกับการเปลี่ยนแปลงของ constraint โดยมีรูปแบบสมการของ Lagrange multiplier คือ

$$\lambda_{\mathrm{i}} = \frac{df}{dg_i} \tag{54}$$

6.5 การหาค่าเหมาะสมที่สุดด้วยวิธี Karush Kuhn Tucker

Karush Kuhn Tucker เป็นวิธีการที่ใช้ในการหาคำตอบของฟังก์ชันที่มี constraints เป็น inequality constraints โดยรูปแบบของ Karush Kuhn Tucker เมื่อกำหนดให้หาค่า minimum ของฟังก์ชัน $f(x)$ โดยมีเงื่อนไขเป็น

$$h(x) = 0$$

$$g(x) \leq 0$$

จากนั้นทำการเพิ่มค่า slack ไปใน constraint $g(x) \leq 0$ ดังสมการ

$$s^2 = (s_1^2, s_2^2, \dots, s_m^2) \tag{55}$$

ดังนั้น จะได้รูปแบบ Lagrangean เป็น

$$L(x, s, \mu) = f(x) - \lambda^T h - \mu^T \nabla g(x) = 0 \quad (56)$$

โดยที่ λ^T คือ Lagrange multiplier

และ μ^T คือ KKT multiplier

จากสมการที่ (56) ทำการหาค่าอนุพันธ์ของฟังก์ชันครั้งที่ 1 เทียบกับ $x,\ s,\ \mu$ และ λ จะได้

$$\frac{\partial L}{\partial x} = \nabla f(x) - \lambda^T \nabla h - \mu^T \nabla g(x) = 0 \quad (57)$$

$$\frac{\partial L}{\partial x} = -2\mu_i s_i = 0, \qquad i = 1, 2, \ldots, m \quad (58)$$

$$\frac{\partial L}{\partial \mu} = -(g(x) + s^2) = 0 \quad (59)$$

$$\frac{\partial L}{\partial \lambda} = h(x) = 0 \quad (60)$$

จากสมการที่ (58) และ (59) สามารถลดรูปได้เป็น

$$\mu_i g_i(x) = 0 \quad (61)$$

เงื่อนไขที่กำหนดใน KKT มีดังนี้

1) หาค่า gradient ของ Lagrange function (Lagrangean) เท่ากับศูนย์
2) Constraints ที่กำหนด

$h(x) = 0$ (equality constraint)

$g(x) \leq 0$ (inequality constaint)

3) Complementary slackness ของ slack variable $\mu s = 0$
4) Feasibility ของ inequality constraint $s_i^2 = 0$
5) Sign condition ของ inequality multiplier $\mu \geq 0$

สำหรับการหาค่า maximize ของฟังก์ชันจะกำหนดรูปแบบของฟังก์ชันและ KKT condition ดังนี้

Maximize $f(x)$

subject to $h(x) = 0$

$g(x) \leq 0$

$\nabla f(x) - \lambda^T \nabla h - \mu^T \nabla g(x) = 0$

KKT condition

$\mu \geq 0, \quad i = 1, 2, \ldots, m$

$\nabla f(x) - \lambda^T \nabla h - \mu^T \nabla g(x) = 0$

$\mu^T g(x) = 0$

$h(x) = 0$

$g(x) \leq 0$

สำหรับการหาค่า minimize ของฟังก์ชันจะกำหนดรูปแบบของฟังก์ชันและ KKT condition ดังนี้

Minimize $f(x)$

subject to $h(x) = 0$

$g(x) \leq 0$

$\nabla f(x) - \lambda^T \nabla h - \mu^T \nabla g(x) = 0$

KKT condition

$\mu \leq 0, \quad i = 1, 2, \ldots, m$

$\nabla f(x) - \lambda^T \nabla h - \mu^T \nabla g(x) = 0$

$\mu^T g(x) = 0$

$h(x) = 0$

$g(x) \leq 0$

7. การจ่ายโหลดอย่างประหยัด (Economic Dispatch)

ปัญหาการจ่ายโหลดอย่างประหยัด คือ ปัญหาที่ต้องการหาค่ากำลังการผลิตไฟฟ้าจากเครื่องกำเนิดไฟฟ้าแต่ละเครื่องของระบบ โดยมีค่าเชื้อเพลิงที่ใช้ในการผลิตต่ำที่สุด ซึ่งกำลังการผลิตไฟฟ้าที่ได้นั้นจะต้องเพียงพอต่อความต้องการกำลังไฟฟ้าของระบบ ดังนั้นปัญหาการจ่ายโหลดอย่างประหยัดจึงเป็นปัญหาการหาค่าที่เหมาะสม (Optimization Problem) โดยมีฟังก์ชันของราคาค่าเชื้อเพลิงเป็นฟังก์ชันเป้าหมาย (Objective Function) ซึ่งเป็นไปตามสมการที่ (62) (Wood and Wollenberg, 1996)

$$F_T = \sum_{i=1}^{N} F_i(P_i) \tag{62}$$

เมื่อ P_i คือ กำลังไฟฟ้าที่ผลิตได้จากเครื่องกำเนิดไฟฟ้าที่ *i*

F_T คือ ฟังก์ชันของราคาค่าเชื้อเพลิงรวม มีหน่วยเป็น บาทต่อชั่วโมง

$F_i(P_i)$ คือ ฟังก์ชันของราคาค่าเชื้อเพลิงของเครื่องกำเนิดไฟฟ้าที่ *i* มีหน่วยเป็น บาทต่อชั่วโมง

โดยจะต้องเป็นไปตามเงื่อนไขของการผลิตกำลังไฟฟ้าให้ตรงกับความต้องการและข้อจำกัดของการผลิตกำลังไฟฟ้า ดังสมการที่ (63) และ (64)

$$\sum_{i=1}^{N} P_i = P_D + P_{loss} \tag{63}$$

$$P_{i,min} \leq P_i \leq P_{i,max} \tag{64}$$

เมื่อ P_D คือ ความต้องการกำลังไฟฟ้าของระบบ

P_{loss} คือ ค่าความสูญเสียของระบบ

N คือ จำนวนของเครื่องกำเนิดไฟฟ้า

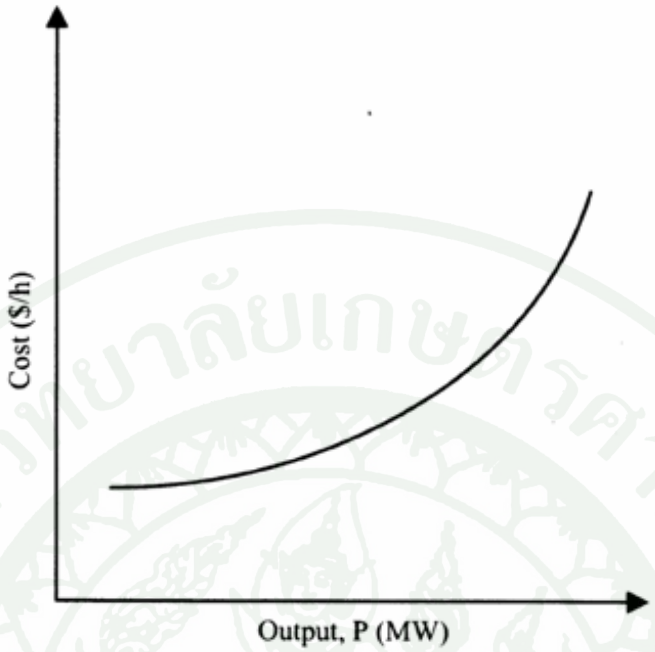
$P_{i,min}$ คือ กำลังการผลิตไฟฟ้าต่ำสุดของเครื่องกำเนิดไฟฟ้าที่ i

$P_{i,max}$ คือ กำลังการผลิตไฟฟ้าสูงสุดของเครื่องกำเนิดไฟฟ้าที่ i

โดยปกติแล้วฟังก์ชันของราคาเชื้อเพลิงที่ใช้ในปัญหาการจ่ายโหลดอย่างประหยัดซึ่งเป็นฟังก์ชันราคาที่ราบเรียบ (Smooth Cost Function) จะอยู่ในรูปของสมการโพลิโนเมียล (Polynomial) ดังแสดงในภาพที่ 13 และสมการที่ (65)

$$F_i(P_i) = a_i P_i^2 + b_i P_i + c_i \tag{65}$$

เมื่อ a_i, b_i และ c_i คือสัมประสิทธิ์ของฟังก์ชันราคาค่าเชื้อเพลิงของเครื่องกำเนิดไฟฟ้าที่ *i*



ภาพที่ 13 ฟังก์ชันราคาค่าเชื้อเพลิงที่ราบเรียบของเครื่องกำเนิดไฟฟ้า

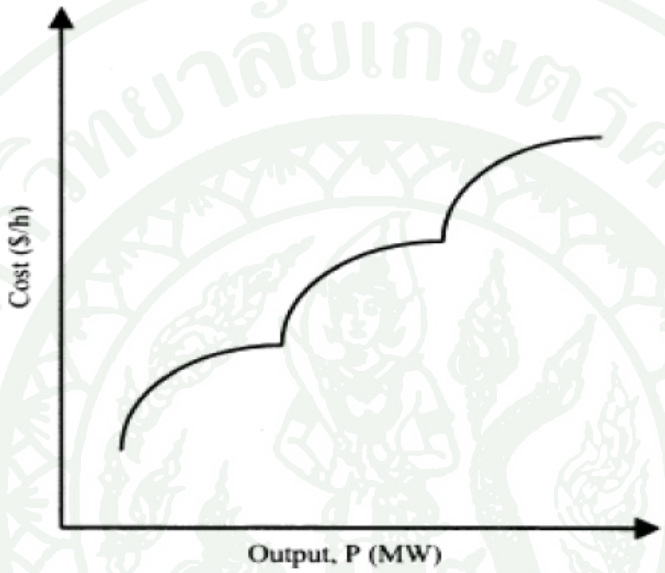
ที่มา: Perez-Guerrero and Cedeno-Maldonado (2005)

ในปัญหาการจ่ายโหลดอย่างประหยัดที่มีผลจากจุดวาล์ว (Valve Point Effect) จะทำให้เกิดฟังก์ชันราคาที่ไม่ราบเรียบ (Non-Smooth Cost Function) ขึ้น ซึ่งจะทำให้ปัญหาการจ่ายโหลดอย่างประหยัดมีความซับซ้อนและยากต่อการคำนวณมากยิ่งขึ้น

ผลที่เกิดจากจุดวาล์ว (Valve Point Effect) คือ ผลที่เกิดขึ้นเนื่องจากเครื่องกำเนิดไฟฟ้ากังหันไอน้ำขนาดใหญ่ที่มีจำนวนของวาล์วที่ใช้ในการปล่อยไอน้ำเป็นจำนวนมาก ทำการเปิดวาล์วเป็นลำดับเพื่อให้ได้กำลังไฟฟ้าในการผลิตเพิ่มมากขึ้น และเมื่อระบบมีความต้องการใช้ไฟฟ้าที่มากขึ้น ดังนั้นจึงต้องเพิ่มไอน้ำให้มากขึ้นโดยทำการเปิดวาล์วตัวที่สอง ซึ่งระหว่างที่มีการเปิดวาล์วตัวที่สองจะทำให้ Incremental Heat Rate ลดลง และเมื่อมีการเปิดวาล์วตัวแรกจะทำให้เกิด Throttling Losses เพิ่มขึ้นอย่างรวดเร็วและจะทำให้ Incremental Heat Rate เพิ่มขึ้นแบบทันทีทันใด ซึ่งในการเพิ่มขึ้นนี้ทำให้เกิดความไม่ต่อเนื่องของ Incremental Heat Rate และทำให้เกิดการกระเพื่อม (Ripple) บนกราฟเส้นโค้งของฟังก์ชันราคาค่าเชื้อเพลิงดังแสดงในภาพที่ 14 ซึ่งจะทำให้ฟังก์ชันของราคาค่าเชื้อเพลิงมีลักษณะสมการที่ (66) (วิลาสินี, 2551)

$$F_i(P_i) = a_i P_i^2 + b_i P_i + c_i + \left| e_i \times \sin\left(f_i \times \left(P_{i,min} - P_i\right)\right)\right| \quad (66)$$

เมื่อ e_i และ f_i คือ สัมประสิทธิ์ของฟังก์ชันราคาค่าเชื้อเพลิง อันเนื่องมาจากผลที่เกิดจากจุดวาล์ว ของเครื่องกำเนิดไฟฟ้าที่ i



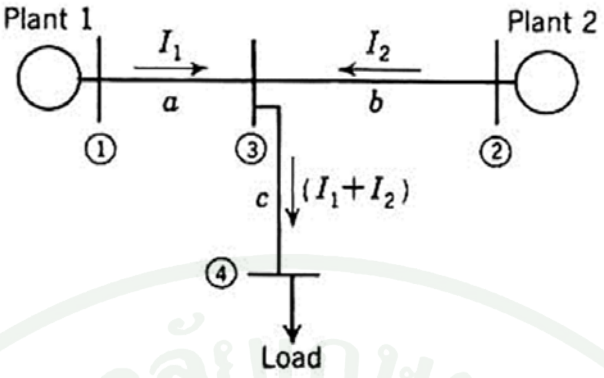
ภาพที่ 14 ฟังก์ชันราคาค่าเชื้อเพลิงที่มีวาล์วที่ใช้ในการปล่อยไอน้ำจำนวน 3 ตัวของเครื่องกำเนิดไฟฟ้า

ที่มา: Perez-Guerrero and Cedeno-Maldonado (2005)

8. ค่าความสูญเสียของระบบ

การแบ่งโหลดไปยังโรงไฟฟ้าต่างๆ ตามหลักเศรษฐศาสตร์ จำเป็นต้องคิดกำลังสูญเสียในระบบส่ง ถึงแม้โรงไฟฟ้าจะมีอัตราค่าใช้จ่ายต่ำสุด แต่ถ้าอยู่ห่างไกลจากโหลดมาก การสูญเสียในระบบสายส่งจะมีค่าสูง ทำให้ไม่เหมาะสมในทางเศรษฐศาสตร์ได้

โดยจะพิจารณาระบบดังภาพ ประกอบด้วยโรงไฟฟ้า 2 โรง จ่ายโหลด



ภาพที่ 15 ตัวอย่างโรงไฟฟ้า 2 โรงที่จ่ายโหลด

ที่มา: คณะวิศวกรรมศาสตร์ มหาวิทยาลัยราชธานี

กำหนดให้ R_a, R_b และ R_c เป็นความต้านทานสายส่งช่วง a ,b และ c

กำลังไฟฟ้าสูญเสียในสายส่ง (P_L) เท่ากับ

$$P_L = 3|I_1|^2R_a + 3|I_2|^2R_b + 3|I_1 + I_2|^2R_c \quad (67)$$

สมมติ ให้ I_1 แล I_2 มีมุมเฟสเท่ากัน จะได้ $|I_1+I_2| = |I_1| + |I_2|$

กำลังไฟฟ้าสูญเสียในสายส่ง จะมีค่าเป็น

$$P_L = 3|I_1|^2R_a + 3|I_2|^2R_b + 3|I_1 + I_2|^2R_c$$

$$= 3|I_1|^2R_a + 3|I_2|^2R_b + 3(|I_1| + |I_2|)^2R_c$$

$$= 3|I_1|^2R_a + 3|I_2|^2R_b + 3(|I_1|^2 + |I_1||I_2| + |I_2|^2)^2R_c$$

$$= 3|I_1|^2(R_a + R_c) + 3 \times 2|I_1||I_2|R_c + 3|I_2|^2(R_b + R_c) \quad (68)$$

ถ้ากำหนดให้ P_1 และ P_2 คือ กำลังผลิต 3 เฟสของโรงไฟฟ้าทั้งสอง ซึ่งมีตัวประกอบกำลังเป็น $\cos\theta_1$และ $\cos\theta_2$แรงดันที่บัส เป็น V_1, V_2

กระแสที่ไหลออกจากโรงไฟฟ้าแต่ละโรง มีค่าเป็น

$$|I_1| = \frac{P_1}{\sqrt{3}|V_1|\cos\theta_1} \tag{69}$$

และ

$$|I_2| = \frac{P_2}{\sqrt{3}|V_2|\cos\theta_2} \tag{70}$$

แทนค่า $|I_1|$, $|I_2|$ ในสมการ

$$P_L = 3|I_1|^2(R_a + R_c) + 3 \times 2|I_1||I_2|R_c + 3|I_2|^2(R_b + R_c)$$

จะได้สมการกำลังสูญเสียในระบบส่ง เป็นฟังชั่นของกำลังผลิตโรงไฟฟ้า เป็น

$$P_L = P_1^2 \frac{R_a+R_c}{|V_1|^2(\cos\theta_1)^2} + 2P_1P_2 \frac{R_c}{|V_1||V_2|(\cos\theta_1)(\cos\theta_2)} + P_2^2 \frac{R_b+R_c}{|V_2|^2(\cos\theta_2)^2} \tag{71}$$

โดยที่

$$B_{11} = \frac{R_a+R_c}{|V_1|^2(\cos\theta_1)^2} \tag{72}$$

$$B_{12} = \frac{R_c}{|V_1||V_2|(\cos\theta_1)(\cos\theta_2)} \tag{73}$$

$$B_{22} = \frac{R_b+R_c}{|V_2|^2(\cos\theta_2)^2} \tag{74}$$

B_{11}, B_{12} และ B_{22} เรียกว่า ค่าสัมประสิทธิ์การสูญเสีย (Loss Coefficients) หรือเรียกอีกอย่างได้ว่า ค่าสัมประสิทธิ์ B (B Coefficients) มีหน่วยเป็น 1/MW

โดย ค่าสัมประสิทธิ์ B หาได้จากแต่ละพื้นที่การผลิต โดยการกำหนดให้แต่ละพื้นที่นั้นยังคงมีโหลด และสมการในการหาค่าความสูญเสีย ค่าความสูญเสียกำลังไฟฟ้าของแต่ละพื้นที่การผลิตที่หาได้นั้นจะอยู่ในรูปของ square matrix มี order เท่ากับจำนวนโครงจักและจำนวน ties line รวมกัน (อำนาจ, 2524)

9. การค้นหาแบบนกกาเหว่า (Cuckoo Search)

การค้นหาแบบนกกาเหว่าเป็นวิธีการที่เลียนแบบพฤติกรรมการวางไข่ของนกกาเหว่า โดยนกกาเหว่าจะวางไข่ในรังของนกสายพันธุ์อื่นซึ่งเราจะเรียกว่านกโฮสต์ (Host Birds) ถ้านกโฮสต์รู้ว่าไข่ที่อยู่ในรังไม่ใช่ไข่ของมัน นกโฮสต์อาจจะโยนไข่ของนกกาเหว่าทิ้งหรือทิ้งรังแล้วไปสร้างรังใหม่ที่อื่นได้ (Yang and Deb, 2009)

โดยทั่วไปไข่ของนกกาเหว่าจะฟักเร็วกว่าไข่ของนกโฮสต์เล็กน้อย เมื่อลูกนกกาเหว่าฟักออกมา มันจะมีสัญชาตญาณที่จะขับไล่ไข่ของนกโฮสต์ออกจากรังเมื่อนกโฮสต์ไม่อยู่ เพื่อเป็นการเพิ่มส่วนแบ่งของอาหาร

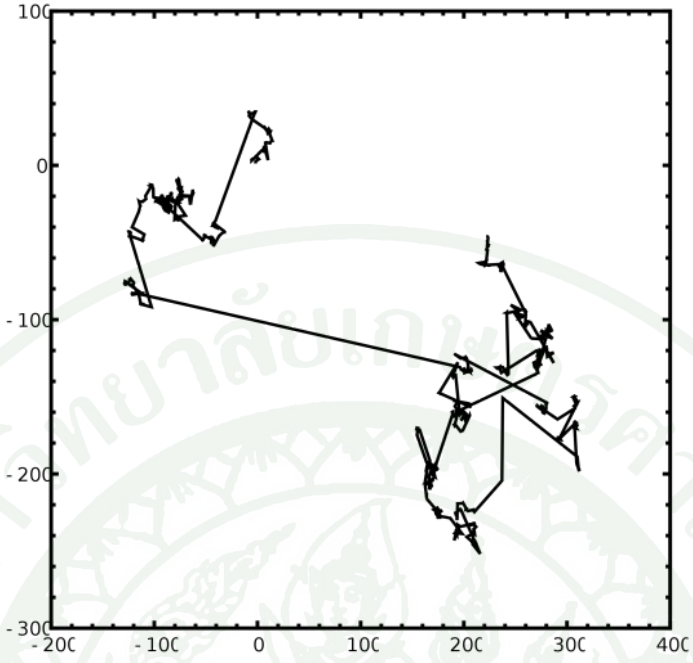
เพื่อให้เข้าใจง่ายในการอธิบายการค้นหาแบบนกกาเหว่า จะใช้กฎอุดมคติ 3 ข้อ คือ

1) นกกาเหว่าจะวางไข่ 1 ฟองต่อครั้ง และจะทิ้งไข่ของมันไว้ในรังที่เลือกสุ่ม

2) รังที่ดีและไข่ที่มีคุณภาพสูง จะดำเนินไปยังรุ่นถัดไปได้

3) จะกำหนดจำนวนรังของนกโฮสต์ที่ใช้ได้ (n) และไข่ที่วางโดยนกกาเหว่าถูกค้นพบโดยนกโฮสต์มีค่าความน่าจะเป็น (p_a) อยู่ระหว่าง 0 ถึง 1

จากการศึกษาต่าง ๆ ได้พบว่าพฤติกรรมการบินของสัตว์หลายชนิด ได้แสดงให้เห็นถึงลักษณะการเคลื่อนที่ของ Lévy Flights



ภาพที่ 16 ภาพแสดงการเคลื่อนที่ของ Lévy flights

ที่มา: กิตติพงษ์ และคณะ (2555)

เมื่อมีการสร้างวิธีแก้ปัญหาใหม่

$$x_i^{(t+1)} = x_i^t + \alpha \oplus \text{Lévy}(\lambda) \tag{75}$$

ที่ $\alpha > 0$ คือขนาดขั้นตอนที่เกี่ยวข้องกับมาตรส่วนของปัญหาที่สนใจ สมการข้างต้นเป็นสมการสำหรับการสุ่มทางเดิน โดยทั่วไปการสุ่มทางเดินเป็น Markov Chain ที่มีสถานะหรือตำแหน่งถัดไปขึ้นอยู่กับตำแหน่งในปัจจุบันและความน่าจะเป็นของการเปลี่ยนแปลง เครื่องหมาย $\oplus$ คือ Entrywise Multiplications ซึ่งมีความคล้ายคลึงกับการปฏิบัติการที่ใช้ในขั้นตอนวิธีหาค่าเหมาะสมที่สุดแบบกลุ่มอนุภาค (Particle Swarm Optimization, PSO) แต่การสุ่มทางเดินของ Lévy Flights นั้น จะมีประสิทธิภาพมากขึ้นในการสำรวจพื้นที่การค้นหา

$$\text{Lévy}(\lambda) \sim u = t^{-\lambda}, \qquad (1 < \lambda \leq 3) \tag{76}$$

การสุ่มหาการเคลื่อนที่ของ Lévy Flights

จะใช้ Mantegna's algorithm มาช่วยในการสร้างตัวเลขสุ่ม โดย Mantegna's algorithm จะสร้างตัวเลขสุ่มตามการกระจายที่มีเสถียรภาพ ตามสมการที่ (77) และ (78) อัลกอริทึมจะมีกระจายโดยที่ α อยู่ในช่วง 0.3 ถึง 1.99

$$\sigma_x = \left[\frac{\Gamma(1+\alpha)\sin\left(\frac{\pi\alpha}{2}\right)}{\Gamma\left(\frac{1+\alpha}{2}\right)\alpha 2^{\frac{(\alpha-1)}{2}}}\right]^{1/\alpha} \tag{77}$$

$$\sigma_y = 1 \tag{78}$$

ในการหาทางเดินถัดไป

$$step = \frac{u}{|v|^{1/\alpha}} \tag{79}$$

โดยที่

$$u = randn[s] * \sigma_x \tag{80}$$

$$v = randn[s] * \sigma_y \tag{81}$$

เมื่อ randn[s] คือ ฟังก์ชันสุ่มหาค่าจำนวนเต็มจาก 1 ถึง s

จะได้

$$stepsize = 0.01 \cdot step \cdot (s - xbest) \tag{82}$$

เมื่อ xbest คือ ตำแหน่งก่อนหน้าที่ดีที่สุด

จากกฎอุดมคติ 3 ข้อ ขั้นตอนพื้นฐานของการค้นหาแบบนกกาเหว่า (Cuckoo Search) สามารถสรุปขั้นตอนได้ดังนี้

1) กำหนดฟังก์ชันวัตถุประสงค์ f(x), $x = (x_1,\ldots,x_d)^T$

2) สร้างประชากรเริ่มต้นของ n โดยเป็นจำนวนรังเริ่มต้น x_i (i = 1,2,…,n)

3) ตรวจสอบว่าครบจำนวนรอบที่กำหนด หรือถึงเกณฑ์ที่ต้องหยุดหรือไม่ ถ้าใช่ให้ทำขั้นตอนที่ 9 แต่ถ้าไม่ก็ให้ทำขั้นตอนที่ 4

4) ประเมินคุณภาพของรังจากฟังก์ชันวัตถุประสงค์ F_i หลังจากนั้นเลือกรังใหม่มาประเมินคุณภาพของรังด้วยฟังก์ชันวัตถุประสงค์ที่ F_j

5) นำค่าที่ได้มาเปรียบเทียบ ถ้าค่า F_i มีค่ามากกว่า F_j ให้ทำการแทนที่รัง j เป็นคำตอบใหม่

6) โดยการค้นพบรังใหม่นั้นจะต้องมีค่าความน่าจะเป็นมากกว่าค่า p_a ที่กำหนดไว้ สำหรับรังที่ไม่ผ่านค่า p_a ให้ทิ้งค่ารังแล้วสุ่มเลือกค่ารังใหม่

7) เก็บคำตอบที่ดีที่สุด หรือรังที่มีคุณภาพ

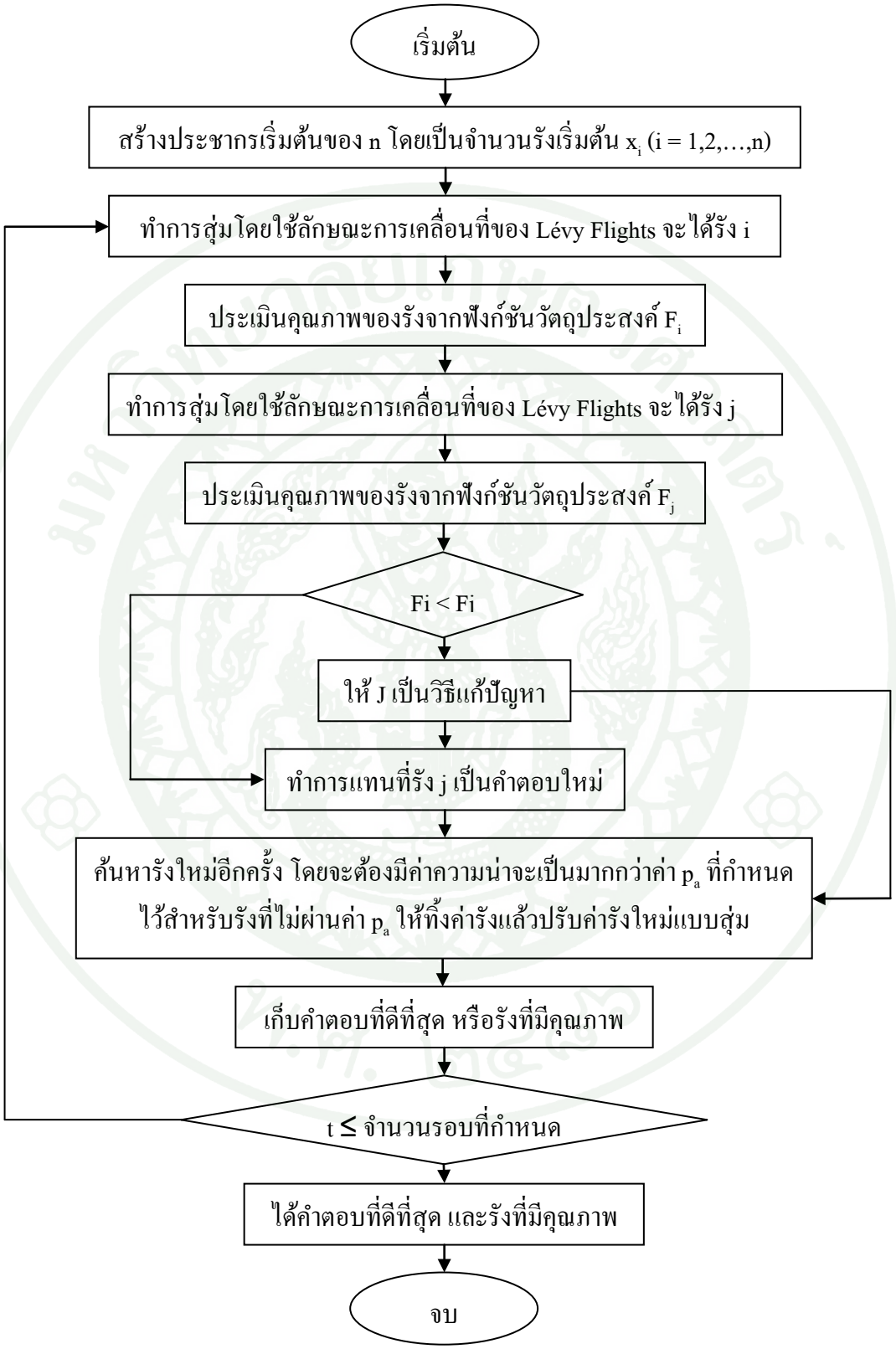
8) เรียงลำดับผลลัพธ์ที่ได้จากนั้นกลับไปทำขั้นตอนที่ 3

9) ผลลัพธ์สุดท้าย จะได้คำตอบที่ดีที่สุดและรังที่มีคุณภาพ

```
begin
  Objective function f(x),  x = (x_1, ..., x_d)^T
  Generate initial population of
     n host nests x_i  (i = 1, 2, ..., n)
  while (t <MaxGeneration) or (stop criterion)
    Get a cuckoo randomly by Lévy flights
      evaluate its quality/fitness F_i
    Choose a nest among n (say, j) randomly
    if (F_i > F_j),
       replace j by the new solution;
    end
    A fraction (p_a) of worse nests
       are abandoned and new ones are built;
    Keep the best solutions
       (or nests with quality solutions);
    Rank the solutions and find the current best
  end while
  Postprocess results and visualization
end
```

ภาพที่ 17 Pseudo code ของการค้นหาแบบนกกาเหว่า

ที่มา: Yang and Deb (2009)

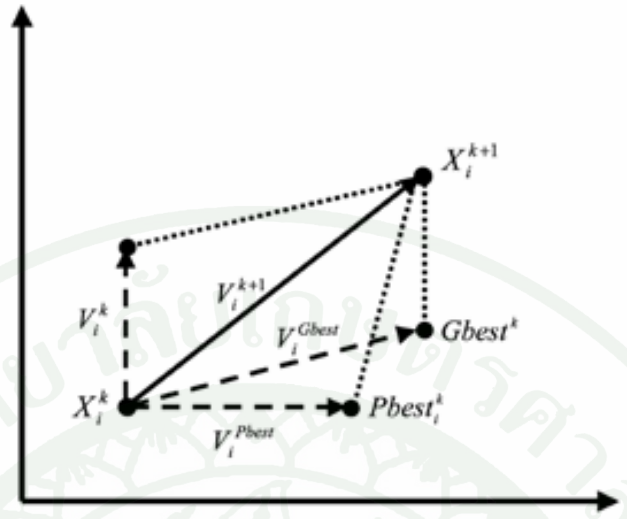


ภาพที่ 18 แผนผังขั้นตอนพื้นฐานของการค้นหาแบบนกกาเหว่า

10. วิธีการทำให้เหมาะสมแบบกลุ่มอนุภาค (Particle Swarm Optimization, PSO)

ในปี 1995 Kennedy และ Eberhart ได้ริเริ่มเทคนิคการคำนวณที่มีวิวัฒนาการ (evolutionary computation technique) แบบใหม่ที่มีชื่อเรียกว่า การทำให้เหมาะสมแบบกลุ่มอนุภาค (Particle Swarm Optimization, PSO) (Kennedy and Eberhart, 1995) ซึ่งมีความคล้ายคลึงกับเทคนิคการคำนวณที่มีวิวัฒนาการอื่นๆ โดย PSO จะใช้ทฤษฎีของการสุ่มสุ่มประชากรเริ่มต้นและใช้การประเมินค่าและการเปลี่ยนแปลงเหล่าประชากรเพื่อค้นหาคำตอบที่เหมาะสมที่สุด (optimal solution) ซึ่ง PSO จะมีความคล้ายคลึงกับวิธีเจนเนติก (Genetic Algorithm, GA) ตรงที่ประชากรเริ่มต้นได้จากการสุ่มเหมือนกัน แต่ต่างกันตรงที่ PSO ไม่ได้ใช้ตัวดำเนินการต่างๆ ในขั้นตอนการเปลี่ยนแปลง เช่น การกลายพันธุ์และการสลับไขว้ เหมือน GA แต่ PSO ใช้การปรับปรุงตัวมันเองในขั้นตอนการเปลี่ยนแปลง และเมื่อนำ PSO มาเปรียบเทียบกับเทคนิคการคำนวณที่มีวิวัฒนาการอื่นๆ ซึ่งความได้เปรียบของ PSO คือเป็นวิธีที่ใช้งานได้ง่ายและมีพารามิเตอร์ที่ใช้ในการปรับปรุงที่น้อย โดยแบบจำลองพื้นฐานทางคณิตศาสตร์ของ PSO สามารถอธิบายได้ดังนี้ (Eberhart and Kennedy, 1995)

ให้ฝูง (swarm) หนึ่งฝูงมีอนุภาค (particle) n ตัวอยู่ในขนาดของช่วงที่ใช้ในการค้นหา d ($d$dimensional search space) ที่รอบการทำงานที่ k ให้ $x_i^k = (x_{i1}^k, x_{i2}^k, \ldots, x_{id}^k)$ ใช้อธิบายตำแหน่ง (position) ของอนุภาคตัวที่ i, $Pbest_i^k = (Pbest_{i1}^k, Pbest_{i2}^k, \ldots, Pbest_{id}^k)$ แสดงตำแหน่งก่อนหน้าที่ดีที่สุดของอนุภาคตัวที่ i, ตำแหน่งที่ดีที่สุดของอนุภาคโดยรวมคือ $Gbest_d^k = (Gbest_1^k, Gbest_2^k, \ldots, Gbest_d^k)$, ความเร็ว (velocity) ของอนุภาคตัวที่ i คือ $v_i^k = (v_{i1}^k, v_{i2}^k, \ldots, v_{id}^k)$ และประชากรแต่ละตัวเรียกว่าอนุภาค (particle) หรือตัวกระทำ (agent) สามารถนำไปปรับปรุงหรือเปลี่ยนแปลงไปยังตำแหน่งใหม่ที่สอดคล้องกับความเร็วปัจจุบัน ซึ่งความแตกต่างระหว่างตำแหน่งปัจจุบันและตำแหน่งที่ดีที่สุดของอนุภาค (*Pbest*) และตำแหน่งที่ดีที่สุดของกลุ่มอนุภาค (*Gbest*) ใช้ในการปรับปรุงความเร็วของอนุภาคตัวที่ i สามารถแสดงได้ดังภาพที่ 19 ซึ่งวิธีในการปรับปรุงความเร็วของอนุภาคมีหลายวิธีดังนี้



ภาพที่ 19 การปรับปรุงความเร็วของอนุภาค

ที่มา: Park *et al*. (2005)

10.1 วิธีดั้งเดิมของ PSO (Original PSO algorithm, OPSO) การปรับปรุงความเร็วของอนุภาคสามารถคำนวณได้จากสมการที่ (83)

$$v_{id}^{k+1} = v_{id}^{k} + c_1 \times rand_1 \times \left(Pbest_{id} - x_{id}^{k}\right) + c_2 \times rand_2 \times \left(Gbest_d - x_{id}^{k}\right) \quad (83)$$

การปรับปรุงตำแหน่งของแต่ละอนุภาคสามารถคำนวณได้จากสมการที่ (84)

$$x_{id}^{k+1} = x_{id}^{k} + v_{id}^{k+1} \quad (84)$$

เมื่อ v_{id}^{k} คือ ความเร็วของอนุภาคตัวที่ i ที่รอบการทำงานที่ k ในขนาดของช่วงที่ใช้ในการค้นหา d; $V_{d,min} \leq v_{id}^{k} \leq V_{d,max}$; $i = 1,2,...,n$, $d = 1,2,...,m$

x_{id}^{k} คือ ตำแหน่งปัจจุบันของอนุภาคตัวที่ i ที่รอบการทำงานที่ k

k คือ จำนวนของรอบการทำงาน

n คือ จำนวนของอนุภาคในกลุ่ม

m คือ จำนวนของสมาชิกในอนุภาคหนึ่งตัว

$c1$, $c2$ คือ ค่าคงที่ความเร่ง (acceleration constant)

$rand_1$, $rand_2$ คือ การกระจายตัวอย่างสม่ำเสมอ (uniformly distributed) โดยการสุ่มจำนวนในช่วง [0,1]

10.2 วิธีการพื้นฐานของ PSO (Basic PSO algorithm, BPSO) เนื่องจาก OPSO ไม่ได้ทำการปรับขนาดก้าว (step size) ของความเร็วซึ่งทำให้ความสามารถในการค้นหามีคุณภาพต่ำ ดังนั้น BPSO จึงใช้ค่าน้ำหนักความเฉื่อย (w) เพื่อให้การค้นหาเฉพาะที่ (local) และโดยรวม (global) สมดุล ซึ่งการปรับปรุงความเร็วของอนุภาคใน BPSO สามารถคำนวณได้จากสมการที่ (85)

$$\mathrm{v}_{\mathrm{id}}^{\mathrm{k+1}} = \mathrm{w} \cdot \mathrm{v}_{\mathrm{id}}^{\mathrm{k}} + c_1 \times rand_1 \times \left(Pbest_{id} - x_{id}^{k}\right) + c_2 \times rand_2 \times \left(Gbest_{d} - x_{id}^{k}\right) \quad (85)$$

เมื่อ w คือ ค่าน้ำหนักความเฉื่อย

ผลและวิจารณ์

ในงานวิจัยนี้จะทำการทดสอบวิธีการค้นหาแบบนกกาเหว่า (Cuckoo Search) กับวิธีหาค่าเหมาะสมที่สุดแบบกลุ่มอนุภาค (PSO) เนื่องจากวิธีหาค่าเหมาะสมที่สุดแบบกลุ่มอนุภาคมีแนวคิดจากการเลียนแบบพฤติกรรมของธรรมชาติเหมือนกับวิธีการค้นหาแบบนกกาเหว่า โดยที่วิธีการค้นหาแบบนกกาเหว่าเลียนแบบพฤติกรรมการวางไข่ของนกกาเหว่า ส่วนวิธีหาค่าเหมาะสมที่สุดแบบกลุ่มอนุภาคเลียนแบบพฤติกรรมการหาอาหารของนกหรือปลา และนอกจากนี้วิธีหาค่าเหมาะสมที่สุดแบบกลุ่มอนุภาคยังเป็นที่นิยมกันอย่างแพร่หลาย โดยในงานวิจัยนี้จะพิจารณากรณีศึกษา 2 กรณี คือ กรณีที่ 1 ฟังก์ชันราคาที่ราบเรียบ (Smooth Cost Function) และไม่มีค่าความสูญเสียของระบบ และกรณีที่ 2 ฟังก์ชันราคาที่ไม่ราบเรียบ (Non-Smooth Cost Function) อันเนื่องมาจากผลที่เกิดจากจุดวาล์ว (Valve Point Effect) และพิจารณาค่าความสูญเสียของระบบ ซึ่งจะใช้โปรแกรมช่วยวิเคราะห์ข้อมูลเชิงวิศวกรรม คณิตศาสตร์ MATLAB ในการคำนวณ

ผล

1. กรณีที่ 1

ปัญหาการจ่ายโหลดอย่างประหยัดที่มีฟังก์ชันราคาค่าเชื้อเพลิงที่ราบเรียบ และไม่มีค่าความสูญเสียของระบบ ($P_{loss} = 0$) สำหรับระบบที่มีเครื่องกำเนิดไฟฟ้าจำนวน 6 เครื่อง

ตารางที่ 1 ข้อมูลของระบบที่มีเครื่องกำเนิดไฟฟ้าจำนวน 6 เครื่อง ซึ่งมีฟังก์ชันราคาค่าเชื้อเพลิงที่ราบเรียบ

Unit	a_i ($/(MW)^{2}h)	b_i ($/MWh)	c_i ($/h)	$P_{i,min}$ (MW)	$P_{i,max}$ (MW)
1	0.1525	38.540	756.800	10	125
2	0.1060	46.160	451.325	10	150
3	0.0280	40.400	1050.000	35	225
4	0.0355	38.310	1243.530	35	210

ตารางที่ 1 (ต่อ)

Unit	a_i ($/(MW)^{2}h)	b_i ($/MWh)	c_i ($/h)	$P_{i,min}$ (MW)	$P_{i,max}$ (MW)
5	0.0211	36.328	1658.570	130	325
6	0.0180	38.270	1356.660	125	315

ที่มา: Güvenç (2010)

กำหนด ความต้องการกำลังไฟฟ้าของระบบ (P_D) เท่ากับ 1100 MW

ตารางที่ 2 ผลการคำนวณของวิธีการค้นหาแบบนกกาเหว่าและวิธีหาค่าเหมาะสมที่สุดแบบกลุ่มอนุภาค จากข้อมูลของระบบที่มีเครื่องกำเนิดไฟฟ้าจำนวน 6 เครื่อง ซึ่งมีฟังก์ชันราคาค่าเชื้อเพลิงที่ราบเรียบ และไม่มีค่าความสูญเสียของระบบ

Unit	Cuckoo Search			PSO		
	P_i (MW)	F_i ($/h)	จำนวนรอบ	P_i (MW)	F_i ($/h)	จำนวนรอบ
1	43.18	2705.07		45.06	2803.05	
2	26.17	1732.06		31.64	2017.94	
3	201.94	10350.16		179.68	9213.05	
4	188.71	9737.35		203.62	10516.08	
5	325.00	15693.86		325.00	15693.86	
6	315.00	15197.76		315.00	15197.76	
Total	1100.00	55416.27	50	1100.00	55441.74	200

ตารางที่ 3 ผลการคำนวณของวิธีหาค่าเหมาะสมที่สุดแบบกลุ่มอนุภาคโดยใช้จำนวนรอบในการคำนวณที่ต่างกัน จากข้อมูลของระบบที่มีเครื่องกำเนิดไฟฟ้าจำนวน 6 เครื่องซึ่งมีฟังก์ชันราคาค่าเชื้อเพลิงที่ราบเรียบและไม่มีค่าความสูญเสียของระบบ

จำนวนรอบ	F_i($/h)
50	56832.62
100	56566.42
150	55990.18
200	55441.74

2. กรณีที่ 2

ปัญหาการจ่ายโหลดอย่างประหยัดที่มีฟังก์ชันราคาที่ไม่ราบเรียบ อันเนื่องมาจากผลที่เกิดจากจุดวาล์ว และพิจารณาค่าความสูญเสียของระบบ สำหรับระบบที่มีเครื่องกำเนิดไฟฟ้าจำนวน 10 เครื่อง

ตารางที่ 4 ข้อมูลของระบบที่มีเครื่องกำเนิดไฟฟ้าจำนวน 10 เครื่อง ซึ่งมีฟังก์ชันราคาที่ไม่ราบเรียบ อันเนื่องมาจากผลที่เกิดจากจุดวาล์ว

Unit	a_i ($/(MW)^{2}h)	b_i ($/MWh)	c_i ($/h)	d_i ($/h)	e_i (rad/MW)	$P_{i,min}$ (MW)	$P_{i,max}$ (MW)
1	0.1524	38.5379	786.7988	450	0.041	150	470
2	0.1058	46.1591	451.3251	600	0.036	135	470
3	0.0280	40.3965	1049.9977	320	0.028	73	340
4	0.0354	38.3055	1243.5311	260	0.052	60	300
5	0.0211	36.3278	1658.5696	280	0.063	73	243
6	0.0179	38.2704	1356.6592	310	0.048	57	160
7	0.0121	36.5104	1450.7045	300	0.086	20	130
8	0.0124	36.5104	1450.7045	340	0.082	47	120

ตารางที่ 4 (ต่อ)

Unit	a_i ($/(MW)^{2}h)	b_i ($/MWh)	c_i ($/h)	d_i ($/h)	e_i (rad/MW)	$P_{i,min}$ (MW)	$P_{i,max}$ (MW)
9	0.1090	39.5804	1455.6056	270	0.098	20	80
10	0.1295	40.5407	1469.4026	380	0.094	10	455

ที่มา: Elshahed *et al*. (2011)

ค่าสัมประสิทธิ์การสูญเสีย (B) = 10^{-6}

$$\begin{bmatrix} 49 & 14 & 15 & 15 & 16 & 17 & 17 & 18 & 19 & 20 \\ 14 & 45 & 16 & 16 & 17 & 15 & 15 & 16 & 18 & 18 \\ 15 & 16 & 39 & 10 & 12 & 12 & 14 & 14 & 16 & 16 \\ 15 & 16 & 10 & 40 & 14 & 10 & 11 & 12 & 14 & 15 \\ 16 & 17 & 12 & 14 & 35 & 11 & 13 & 13 & 15 & 16 \\ 17 & 15 & 12 & 10 & 11 & 36 & 12 & 12 & 14 & 15 \\ 17 & 15 & 14 & 11 & 13 & 12 & 38 & 16 & 16 & 18 \\ 18 & 16 & 14 & 12 & 13 & 12 & 16 & 40 & 15 & 16 \\ 19 & 18 & 16 & 14 & 15 & 14 & 16 & 15 & 42 & 19 \\ 20 & 18 & 16 & 15 & 16 & 15 & 18 & 16 & 19 & 44 \end{bmatrix}$$

กำหนด ความต้องการกำลังไฟฟ้าของระบบ (P_D) เท่ากับ 2000 MW

ตารางที่ 5 ผลการคำนวณของวิธีการค้นหาแบบนกกาเหว่าและวิธีหาค่าเหมาะสมที่สุดแบบกลุ่มอนุภาค จากข้อมูลของระบบที่มีเครื่องกำเนิดไฟฟ้าจำนวน 10 เครื่อง ซึ่งมีฟังก์ชันราคาที่ไม่ราบเรียบ อันเนื่องมาจากผลที่เกิดจากจุดวาล์ว และมีค่าความสูญเสียของระบบ

	Cuckoo Search			PSO		
Unit	P_i (MW)	F_i ($/h)	จำนวนรอบ	P_i (MW)	F_i ($/h)	จำนวนรอบ
1	206.78	15599.00		206.16	15543.87	
2	261.44	20343.20		263.88	20597.34	
3	340.00	18319.02		340.00	18319.02	
4	300.00	15943.61		300.00	15943.61	

ตารางที่ 5 (ต่อ)

Unit	Cuckoo Search			PSO		
	P_i (MW)	F_i ($/h)	จำนวนรอบ	P_i (MW)	F_i ($/h)	จำนวนรอบ
5	243.00	12000.82		243.00	12000.82	
6	160.00	8239.89		160.00	8239.89	
7	130.00	6412.11		130.00	6412.11	
8	120.00	6110.07		120.00	6110.07	
9	80.00	5425.57		80.00	5425.57	
10	235.10	18439.03		233.32	18297.68	
Total	2076.32	126832.32	50	2076.36	126889.98	300

ตารางที่ 6 ผลการคำนวณของวิธีหาค่าเหมาะสมที่สุดแบบกลุ่มอนุภาคโดยใช้จำนวนรอบในการคำนวณที่ต่างกัน จากข้อมูลของระบบที่มีเครื่องกำเนิดไฟฟ้าจำนวน 10 เครื่อง ซึ่งมีฟังก์ชันราคาที่ไม่ราบเรียบ อันเนื่องมาจากผลที่เกิดจากจุดวาล์ว และมีค่าความสูญเสียของระบบ

จำนวนรอบ	F_i($/h)
50	130743.76
100	130256.08
150	129482.64
200	128646.49
250	127943.62
300	126889.98

วิจารณ์

1. กรณีที่ 1

จากการคำนวณผลการทดลองโดยใช้โปรแกรม MATLAB โดยใช้ข้อมูลในตารางที่ 1 จะได้ผลการทดลองในตารางที่ 2 และ 3 ซึ่งพบว่าวิธีการค้นหาแบบนกกาเหว่าให้ค่าเชื้อเพลิงของเครื่องกำเนิดไฟฟ้าเครื่องที่ 5 และ 6 เท่ากับกับวิธีหาค่าเหมาะสมที่สุดแบบกลุ่มอนุภาค และให้ค่าเชื้อเพลิงรวมน้อยกว่าวิธีหาค่าเหมาะสมที่สุดแบบกลุ่มอนุภาค อีกทั้งยังใช้จำนวนรอบในการคำนวณน้อยกว่าวิธีหาค่าเหมาะสมที่สุดแบบกลุ่มอนุภาคถึง 4 เท่า

2. กรณีที่ 2

จากการคำนวณผลการทดลองโดยใช้โปรแกรม MATLAB โดยใช้ข้อมูลในตารางที่ 4 และค่าสัมประสิทธิ์การสูญเสียการส่งผ่าน จะได้ผลการทดลองในตารางที่ 5 และ 6 ซึ่งพบว่าวิธีการค้นหาแบบนกกาเหว่าให้ค่าเชื้อเพลิงของเครื่องกำเนิดไฟฟ้าเครื่องที่ 3, 4, 5, 6, 7, 8 และ 9 เท่ากับกับวิธีหาค่าเหมาะสมที่สุดแบบกลุ่มอนุภาค และให้ค่าเชื้อเพลิงรวมน้อยกว่าวิธีหาค่าเหมาะสมที่สุดแบบกลุ่มอนุภาค อีกทั้งยังใช้จำนวนรอบในการคำนวณน้อยกว่าวิธีหาค่าเหมาะสมที่สุดแบบกลุ่มอนุภาคถึง 6 เท่า

สรุปและข้อเสนอแนะ

สรุป

จากการที่ได้ศึกษาการค้นหาแบบนกกาเหว่า พบว่าการค้นหาแบบนกกาเหว่านั้นมีพารามิเตอร์ที่จำเป็นต้องปรับ และจำนวนรอบหรือเวลาในการคำนวณน้อยกว่าในวิธีอื่น ๆ จึงเหมาะที่จะนำเอาการค้นหาแบบนกกาเหว่ามาประยุกต์ใช้ในการคำนวณการแก้ปัญหาการจ่ายโหลดอย่างประหยัด จากการทดสอบ โดยทดสอบกรณีศึกษา 2 กรณี คือ กรณีที่ 1 ฟังก์ชันราคาที่ราบเรียบ และไม่มีค่าความสูญเสียของระบบ สำหรับระบบที่มีเครื่องกำเนิดไฟฟ้าจำนวน 6 เครื่อง และกรณีที่ 2 ฟังก์ชันราคาที่ไม่ราบเรียบ อันเนื่องมาจากผลที่เกิดจากจุดวาล์ว และพิจารณาค่าความสูญเสียของระบบ สำหรับระบบที่มีเครื่องกำเนิดไฟฟ้าจำนวน 10 เครื่อง โดยใช้โปรแกรม MATLAB มาช่วยในการคำนวณ แล้วนำผลลัพธ์มาเปรียบเทียบกับวิธีหาค่าเหมาะสมที่สุดแบบกลุ่มอนุภาค พบว่าวิธีการค้นหาแบบนกกาเหว่านั้นจะได้ค่าเชื้อเพลิงรวมน้อยกว่าวิธีหาค่าเหมาะสมที่สุดแบบกลุ่มอนุภาค อีกทั้งยังใช้จำนวนรอบในการคำนวณน้อยกว่าวิธีหาค่าเหมาะสมที่สุดแบบกลุ่มอนุภาค และนอกจากนี้วิธีการค้นหาแบบนกกาเหว่ายังมีพารามิเตอร์ที่ต้องปรับไม่มาก ทำให้สามารถปรับค่าและคำนวณได้ง่ายและรวดเร็วยิ่งขึ้น

ข้อเสนอแนะ

การนำการแก้ปัญหาการจ่ายโหลดอย่างประหยัดด้วยวิธีการค้นหาแบบนกกาเหว่าไปปรับค่าพารามิเตอร์ต่าง ๆ เช่น จำนวนรังของนกโฮสต์ที่ใช้ได้ (n), ค่าความน่าจะเป็น (p_a) ให้เหมาะสม เพื่อที่จะได้คำตอบที่ดีและรังที่มีคุณภาพมากยิ่งขึ้น และในอนาคตอาจนำวิธีการค้นหาแบบนกกาเหว่ามาแก้ปัญหาต่าง ๆ ทางด้านระบบไฟฟ้าได้อีกด้วย

เอกสารและสิ่งอ้างอิง

กิตติพงษ์ จรัญศิริไพศาล, สิรภัทร เชี่ยวชาญวัฒนา และ คำรณ สุนัติ. 2555. เพิ่มความถูกต้องของตัวแบบซัพพอร์ตเวกเตอร์แมทชีนแบบค่ากำลังสองน้อยที่สุดด้วยขั้นตอนวิธีการค้นหาแบบนกดุเหว่า, น.316-323. *ใน* **การประชุมวิชาการนำเสนอผลงานวิจัยระดับบัณฑิตศึกษาครั้งที่ 13**. 17 กุมภาพันธ์ 2555, มหาวิทยาลัยขอนแก่น, ขอนแก่น.

ชำนาญ ห่อเกียรติ. 2548. **ระบบไฟฟ้ากำลัง.** พิมพ์ครั้งที่ 3. จรัลสนิทวงศ์การพิมพ์, กรุงเทพฯ.

วิลาสินี ศึกษาการ. 2551. **การจ่ายโหลดอย่างประหยัดที่มีฟังก์ชันราคาที่ไม่ราบเรียบโดยใช้วิธีทำให้เหมาะสมแบบกลุ่มอนุภาคร่วมกับวิธีโปรแกรมกำลังสองแบบลำดับ.** วิทยานิพนธ์ปริญญาโท, มหาวิทยาลัยเกษตรศาสตร์.

อำนาจ ตันชวลิต. 2524. **การส่งจ่ายพลังงานไฟฟ้าในระบบไฟฟ้ากำลังให้ประหยัดค่าใช้จ่าย.** วิทยานิพนธ์ปริญญาโท, มหาวิทยาลัยเกษตรศาสตร์.

Attaviriyanupap, P., H. Kita, E. Tanaka and J. Hasegawa. 2002. A Hybrid EP and SQP for Dynamic Economic Dispatch with Nonsmooth Fuel Cost Function, pp. 411-416. **IEEE Trans. on Power Systems Vol.17(2).**

Elshahed, M.A., M.M. Elmarsfawy and H.M.Z. Eldain. 2011. A New Economic Dispatch Constrained by Correlated Weibull Probability Distribution Model for Wind Power, pp. 1-6. *In* **Innovative Smart Grid Technologies - Middle East.** 17-20 December 2011, Saudi Arabia.

Eberhart, R. and J. Kennedy. 1995. A new optimizer using particle swarm theory, pp. 39-43. *In* **Micro Machine and Human Science, 1995. MHS '95., Proceedings of the Sixth International Symposium on.** 4-6 October 1995, Indianapolis, IN, USA.

Park, J.B., K.S. Lee, J.R. Shin and K.Y. Lee. 2005. A Particle Swarm Optimization for Economic Dispatch with Nonsmooth Cost Functions, pp. 34-42. **IEEE Trans. On Power Systems Vol.20(1).**

Perez-Guerrero, R.E. and J.R. Cedenio-Maldonado. 2005. Economic Power Dispatch with Non-Smooth Cost Functions Using Differential Evolution, pp. 183-190. *In* **Power Symposium, 2005. Proceedings of the 37th Annual North American.** 23-25 October 2005, Ames, Iowa, USA.

Rani, K.N.A. and F. Malek. 2011. Preliminary Study on Cuckoo Search Parameters for Symmetric Linear Array Geometry Synthesis, pp. 568-572. *In* **TENCON 2011 - 2011 IEEE Region 10 Conference.** 21-24 November 2011, Bali, Indonesia.

Rani, K.N.A. and F. Malek. 2011. Symmetric Linear Antenna Array Geometry Synthesis using Cuckoo Search Metaheuristic Algorithm, pp. 374-379. *In* **2011 17th Asia-Pacific Conference on Communications.** 2-5 October 2011, Kota Kinabalu, Sabah, Malaysia.

Sinha, N. and B. Purkayastha. 2004. PSO Embedded Evolutionary Programming Technique for Non-convex Economic Load Dispatch, pp. 66-71. *In* **Power Systems Conference and Exposition, 2004. IEEE PES.** 10-13 October 2004, New York, USA.

Soneji, H. and R.C. Sanghvi. 2012. Towards the Improvement of Cuckoo Search Algorithm, pp. 878-883. *In* **2012 World Congress on Information and Communication Technologies (WICT).** 30 October-2 November 2012, Trivandrum, India.

Uğur Güvenç. 2010. Combined economic emission dispatch solution using genetic algorithm based on similarity crossover, pp. 2451-2456. **Scientific Research & Essays Vol. 5(17).**

Victoire, T.A.A. and A.E. Jeyakumar. 2004. Hybrid PSO–SQP for Economic Dispatch with Valve-Point Effect, pp. 51-59. **Electric Power System Research Vol. 71(1)**.

Wood, A.J. and B.F. Wollenberg. 1996. **Power Generation, Operation, and Control**. second edition. John Wiley and Sons, Inc., New York.

Walters, D.C. and G.B. Sheble. 1993. Genetic Algorithm Solution of Economic Dispatch with Valve Point Loading, pp. 1325-1332. **Power Systems, IEEE Transactions on Vol.8(3).**

Yang, X.S. and S. Deb. 2009. Cuckoo Search via Lévy flights, pp. 210 - 214. *In* **2009 World Congress on Nature & Biologically Inspired Computing (NaBIC 2009).** 9-11 December 2009, Coimbatore, India.

Yang, X.S. and S. Deb. 2010. Engineering Optimisation by Cuckoo Search, pp. 330 - 343. **International Journal of Mathematical Modelling and Numerical Optimisation, Vol.1(4).**

Yang, X.S. and S. Deb. 2012. Cuckoo Search for Business Optimization Applications, pp. 1-5. *In* **2012 National Conference on Computing and Communication Systems (NCCCS).** 21-22 November 2012, Durgapur, West Bengal, India.

ภาคผนวก

ภาคผนวก ก

ตัวอย่าง Code MATLAB การแก้ปัญหาการจ่ายโหลดอย่างประหยัด
ด้วยวิธีการค้นหาแบบนกกาเหว่า

ตัวอย่าง Code MATLAB การแก้ปัญหาการจ่ายโหลดอย่างประหยัดด้วยวิธีการค้นหาแบบนกกาเหว่าที่มีฟังก์ชันราคาค่าเชื้อเพลิงที่ราบเรียบ จากข้อมูลของระบบที่มีเครื่องกำเนิดไฟฟ้าจำนวน 6 เครื่อง และไม่มีค่าความสูญเสียของระบบ

```
function [bestnest,fmin]=cuckoo_search(n)

if nargin<1,
% Number of nests (or different solutions)
n=25;
end

% Discovery rate of alien eggs/solutions
pa=0.25;

%% Change this if you want to get better results
% Tolerance
Tol=1.0e-5;

%% Simple bounds of the search domain
nd=15;

% Lower bounds
Lb=-5*ones(1,nd);

% Upper bounds
Ub=5*ones(1,nd);

% Random initial solutions
for i=1:n,
nest(i,:)=Lb+(Ub-Lb).*rand(size(Lb));
end
```

```
 % Get the current best
fitness=10^10*ones(n,1);
[fmin,bestnest,nest,fitness]=get_best_nest(nest,nest,fitness);

N_iter=0;

%% Starting iterations
while (fmin>Tol),
        % Generate new solutions (but keep the current best)
        new_nest=get_cuckoos(nest,bestnest,Lb,Ub);
        [fnew,best,nest,fitness]=get_best_nest(nest,new_nest,fitness);

% Update the counter
         N_iter=N_iter+n;

% Discovery and randomization
        new_nest=empty_nests(nest,Lb,Ub,pa) ;

        % Evaluate this set of solutions
         [fnew,best,nest,fitness]=get_best_nest(nest,new_nest,fitness);

% Update the counter again
         N_iter=N_iter+n;

        % Find the best objective so far
        if fnew<fmin,
                 fmin=fnew;
                bestnest=best;
        end
end %% End of iterations
 %% Post-optimization processing
```

```
%% Display all the nests
disp(strcat('Total number of iterations=',num2str(N_iter)));
fmin
bestnest

%% ---------- All subfunctions are list below ----------
%% Get cuckoos by ramdom walk
function nest=get_cuckoos(nest,best,Lb,Ub)

% Levy flights
n=size(nest,1);

% Levy exponent and coefficient
% For details, see equation (2.21), Page 16 (chapter 2) of the book
% X. S. Yang, Nature-Inspired Metaheuristic Algorithms, 2nd Edition, Luniver Press, (2010).
beta=3/2;
sigma=(gamma(1+beta)*sin(pi*beta/2)/(gamma((1+beta)/2)*beta*2^((beta-1)/2)))^(1/beta);

for j=1:n,
        s=nest(j,:);
        % This is a simple way of implementing Levy flights
         % For standard random walks, use step=1;
         %% Levy flights by Mantegna's algorithm
        u=randn(size(s))*sigma;
        v=randn(size(s));
        step=u./abs(v).^(1/beta);

% In the next equation, the difference factor (s-best) means that
% when the solution is the best solution, it remains unchanged.
stepsize=0.01*step.*(s-best);
% Here the factor 0.01 comes from the fact that L/100 should the typical
```

```
% step size of walks/flights where L is the typical lenghtscale;
% otherwise, Levy flights may become too aggresive/efficient,
% which makes new solutions (even) jump out side of the design domain
% (and thus wasting evaluations).
% Now the actual random walks or flights
s=s+stepsize.*randn(size(s));

% Apply simple bounds/limits
nest(j,:)=simplebounds(s,Lb,Ub);
end

%% Find the current best nest
function [fmin,best,nest,fitness]=get_best_nest(nest,newnest,fitness)

% Evaluating all new solutions
for j=1:size(nest,1),
        fnew=fobj(newnest(j,:));
        if fnew<=fitness(j),
                 fitness(j)=fnew;
                nest(j,:)=newnest(j,:);
        end
end

% Find the current best
[fmin,K]=min(fitness) ;
best=nest(K,:);

%% Replace some nests by constructing new solutions/nests
function new_nest=empty_nests(nest,Lb,Ub,pa)

% A fraction of worse nests are discovered with a probability pa
```

```
n=size(nest,1);

% Discovered or not -- a status vector
K=rand(size(nest))>pa;

% In the real world, if a cuckoo's egg is very similar to a host's eggs, then
% this cuckoo's egg is less likely to be discovered, thus the fitness should
% be related to the difference in solutions.  Therefore, it is a good idea
% to do a random walk in a biased way with some random step sizes.
%% New solution by biased/selective random walks
stepsize=rand*(nest(randperm(n),:)-nest(randperm(n),:));
new_nest=nest+stepsize.*K;

% Application of simple constraints
function s=simplebounds(s,Lb,Ub)

% Apply the lower bound
ns_tmp=s;
I=ns_tmp<Lb;
ns_tmp(I)=Lb(I);

% Apply the upper bounds
J=ns_tmp>Ub;
ns_tmp(J)=Ub(J);

% Update this new move
s=ns_tmp;

%% You can replace the following by your own functions
% A d-dimensional objective function
```

```
function Fcost=fobj(h)

% 1.a ($/MW^2) 2. b $/MW 3. c ($) 4.lower lomit(MW) 5.Upper limit(MW)
data=[0.1525 38.540 756.800 10 125
0.1060 46.160 451.325 10 150
0.0280 40.400 1050.000 35 225
0.0355 38.310 1243.530 35 210
0.0211 36.328 1658.570 130 325
0.0180 38.270 1356.660 125 315];

% Demand (MW)
Pd=1100;

% Loss coefficients it should be squarematrix of size nXn where n is the no
% of plants
B=[0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0];

n=length(data(:,1));
Aeq=ones(1,n);
a=data(:,1);
b=data(:,2);
c=data(:,3);
l=data(:,4);
u=data(:,5);
P=l;
```

```
for i=1:10
Pl=P'*B*P;
Pd1=Pd+Pl;
ll=diag(1-2*B*P);
A1=inv(ll)*a;
B1=inv(ll)*b;
H=2*diag(A1);
P=quadprog(H,B1,[],[],Aeq,Pd1,l,u);
Fcost1=(a.*P.*P+b.*P+c);
end

Pt=P
Ploss=Pl
Fcost=sum(a.*P.*P+b.*P+c)
```

ตัวอย่าง Code MATLAB การแก้ปัญหาการจ่ายโหลดอย่างประหยัดด้วยวิธีการค้นหาแบบนกกาเหว่าที่มีฟังก์ชันราคาที่ไม่ราบเรียบ อันเนื่องมาจากผลที่เกิดจากจุดวาล์ว จากข้อมูลของระบบที่มีเครื่องกำเนิดไฟฟ้าจำนวน 10 เครื่อง และมีค่าความสูญเสียของระบบ

```
function [bestnest,fmin]=cuckoo_search(n)

if nargin<1,
% Number of nests (or different solutions)
n=25;
end

% Discovery rate of alien eggs/solutions
pa=0.25;

%% Change this if you want to get better results
% Tolerance
Tol=1.0e-5;

%% Simple bounds of the search domain
nd=15;

% Lower bounds
Lb=-5*ones(1,nd);

% Upper bounds
Ub=5*ones(1,nd);

% Random initial solutions
for i=1:n,
nest(i,:)=Lb+(Ub-Lb).*rand(size(Lb));
end
```

```
 % Get the current best
fitness=10^10*ones(n,1);
[fmin,bestnest,nest,fitness]=get_best_nest(nest,nest,fitness);

N_iter=0;

%% Starting iterations
while (fmin>Tol),
        % Generate new solutions (but keep the current best)
        new_nest=get_cuckoos(nest,bestnest,Lb,Ub);
        [fnew,best,nest,fitness]=get_best_nest(nest,new_nest,fitness);

% Update the counter
         N_iter=N_iter+n;

% Discovery and randomization
        new_nest=empty_nests(nest,Lb,Ub,pa) ;

        % Evaluate this set of solutions
         [fnew,best,nest,fitness]=get_best_nest(nest,new_nest,fitness);

% Update the counter again
         N_iter=N_iter+n;

        % Find the best objective so far
        if fnew<fmin,
                 fmin=fnew;
                bestnest=best;
        end
end %% End of iterations
%% Post-optimization processing
```

```
%% Display all the nests
disp(strcat('Total number of iterations=',num2str(N_iter)));
fmin
bestnest

%% ---------- All subfunctions are list below ----------
%% Get cuckoos by ramdom walk
function nest=get_cuckoos(nest,best,Lb,Ub)

% Levy flights
n=size(nest,1);

% Levy exponent and coefficient
% For details, see equation (2.21), Page 16 (chapter 2) of the book
% X. S. Yang, Nature-Inspired Metaheuristic Algorithms, 2nd Edition, Luniver Press, (2010).
beta=3/2;
sigma=(gamma(1+beta)*sin(pi*beta/2)/(gamma((1+beta)/2)*beta*2^((beta-1)/2)))^(1/beta);

for j=1:n,
        s=nest(j,:);
        % This is a simple way of implementing Levy flights
         % For standard random walks, use step=1;
         %% Levy flights by Mantegna's algorithm
        u=randn(size(s))*sigma;
        v=randn(size(s));
        step=u./abs(v).^(1/beta);

% In the next equation, the difference factor (s-best) means that
% when the solution is the best solution, it remains unchanged.
stepsize=0.01*step.*(s-best);
% Here the factor 0.01 comes from the fact that L/100 should the typical
```

```
% step size of walks/flights where L is the typical lenghtscale;
% otherwise, Levy flights may become too aggresive/efficient,
% which makes new solutions (even) jump out side of the design domain
% (and thus wasting evaluations).
% Now the actual random walks or flights
s=s+stepsize.*randn(size(s));

% Apply simple bounds/limits
nest(j,:)=simplebounds(s,Lb,Ub);
end

%% Find the current best nest
function [fmin,best,nest,fitness]=get_best_nest(nest,newnest,fitness)

% Evaluating all new solutions
for j=1:size(nest,1),
        fnew=fobj(newnest(j,:));
        if fnew<=fitness(j),
                 fitness(j)=fnew;
                nest(j,:)=newnest(j,:);
        end
end

% Find the current best
[fmin,K]=min(fitness) ;
best=nest(K,:);

%% Replace some nests by constructing new solutions/nests
function new_nest=empty_nests(nest,Lb,Ub,pa)

% A fraction of worse nests are discovered with a probability pa
```

```
n=size(nest,1);

% Discovered or not -- a status vector
K=rand(size(nest))>pa;

% In the real world, if a cuckoo's egg is very similar to a host's eggs, then
% this cuckoo's egg is less likely to be discovered, thus the fitness should
% be related to the difference in solutions.  Therefore, it is a good idea
% to do a random walk in a biased way with some random step sizes.
%% New solution by biased/selective random walks
stepsize=rand*(nest(randperm(n),:)-nest(randperm(n),:));
new_nest=nest+stepsize.*K;

% Application of simple constraints
function s=simplebounds(s,Lb,Ub)

% Apply the lower bound
ns_tmp=s;
I=ns_tmp<Lb;
ns_tmp(I)=Lb(I);

% Apply the upper bounds
J=ns_tmp>Ub;
ns_tmp(J)=Ub(J);

% Update this new move
s=ns_tmp;

%% You can replace the following by your own functions
% A d-dimensional objective function
```

```
function Fcost=fobj(h)

% 1.a ($/MW^2) 2. b $/MW 3. c ($) 4.lower lomit(MW) 5.Upper limit(MW)
data=[0.1524 38.5379 786.7988 450 0.041 150 470
0.1058 46.1591 451.3251 600 0.036 135 470
0.0280 40.3965 1049.9977 320 0.028 73 340
0.0354 38.3055 1243.5311 260 0.052 60 300
0.0211 36.3278 1658.5696 280 0.063 73 243
0.0179 38.2704 1356.6592 310 0.048 57 160
0.0121 36.5104 1450.7045 300 0.086 20 130
0.0124 36.5104 1450.7045 340 0.082 47 120
0.1090 39.5804 1455.6056 270 0.098 20 80
0.1295 40.5407 1469.4026 380 0.094 10 455];

% Demand (MW)
Pd=2000;

% Loss coefficients it should be squarematrix of size nXn where n is the no
% of plants
B=[0.000049 0.000014 0.000015 0.000015 0.000016 0.000017 0.000017 0.000018 0.000019
0.000020
0.000014 0.000045 0.000016 0.000016 0.000017 0.000015 0.000015 0.000016 0.000018
0.000018
0.000015 0.000016 0.000039 0.000010 0.000012 0.000012 0.000014 0.000014 0.000016
0.000016
0.000015 0.000016 0.000010 0.000040 0.000014 0.000010 0.000011 0.000012 0.000014
0.000015
0.000016 0.000017 0.000012 0.000014 0.000035 0.000011 0.000013 0.000013 0.000015
0.000016
0.000017 0.000015 0.000012 0.000010 0.000011 0.000036 0.000012 0.000012 0.000014
0.000015
```

```
0.000017 0.000015 0.000014 0.000011 0.000013 0.000012 0.000038 0.000016 0.000016
0.000018
0.000018 0.000016 0.000014 0.000012 0.000013 0.000012 0.000016 0.000040 0.000015
0.000016
0.000019 0.000018 0.000016 0.000014 0.000015 0.000014 0.000016 0.000015 0.000042
0.000019
0.000020 0.000018 0.000016 0.000015 0.000016 0.000015 0.000018 0.000016 0.000019
0.000044];

n=length(data(:,1));
Aeq=ones(1,n);
a=data(:,1);
b=data(:,2);
c=data(:,3);
d=data(:,4);
e=data(:,5);
l=data(:,6);
u=data(:,7);
P=l;

for i=1:10
Pl=P'*B*P;
Pd1=Pd+Pl;
ll=diag(1-2*B*P);
A1=inv(ll)*a;
B1=inv(ll)*b;
H=2*diag(A1);
P=quadprog(H,B1,[],[],Aeq,Pd1,l,u);
Fcost1=(a.*P.*P+b.*P+c+abs(d.*sin(e.*(l-P))));
end
```

Pt=P

Ploss=Pl

Fcost=sum(a.*P.*P+b.*P+c+abs(e.*sin(f.*(l-P))))

ภาคผนวก ข

บทความที่เกี่ยวข้องกับงานวิจัย

Cuckoo Search via Lévy Flights

Xin-She Yang
Department of Engineering,
University of Cambridge
Trumpinton Street
Cambridge CB2 1PZ, UK
Email: xy227@cam.ac.uk

Suash Deb
Department of Computer Science & Engineering
C. V. Raman College of Engineering
Bidyanagar, Mahura, Janla
Bhubaneswar 752054, INDIA
Email: suashdeb@gmail.com

***Abstract*—In this paper, we intend to formulate a new metaheuristic algorithm, called Cuckoo Search (CS), for solving optimization problems. This algorithm is based on the obligate brood parasitic behaviour of some cuckoo species in combination with the Lévy flight behaviour of some birds and fruit flies. We validate the proposed algorithm against test functions and then compare its performance with those of genetic algorithms and particle swarm optimization. Finally, we discuss the implication of the results and suggestion for further research.**

***Index Terms*—algorithm; cuckoo search; Lévy flight; metaheuristics; nature-inspired strategy; optimization;**

I. Introduction

More and more modern metaheuristic algorithms inspired by nature are emerging and they become increasingly popular. For example, particles swarm optimization (PSO) was inspired by fish and bird swarm intelligence, while the Firefly Algorithm was inspired by the flashing pattern of tropical fireflies [2], [3], [6], [21], [22]. These nature-inspired metaheuristic algorithms have been used in a wide range of optimization problems, including NP-hard problems such as the travelling salesman problem [2], [3], [6], [8], [10], [21].

The power of almost all modern metaheuristics comes from the fact that they imitate the best feature in nature, especially biological systems evolved from natural selection over millions of years. Two important characteristics are selection of the fittest and adaptation to the environment. Numerically speaking, these can be translated into two crucial characteristics of the modern metaheuristics: intensification and diversification [3]. Intensification intends to search around the current best solutions and select the best candidates or solutions, while diversification makes sure the algorithm can explore the search space efficiently.

This paper aims to formulate a new algorithm, called Cuckoo Search (CS), based on the interesting breeding bebaviour such as brood parasitism of certain species of cuckoos. We will first introduce the breeding bebaviour of cuckoos and the characteristics of Lévy flights of some birds and fruit flies, and then formulate the new CS, followed by its implementation. Finally, we will compare the proposed search strategy with other popular optimization algorithms and discuss our findings and their implications for various optimization problems.

II. Cuckoo Behaviour and Lévy Flights

A. Cuckoo Breeding Behaviour

Cuckoo are fascinating birds, not only because of the beautiful sounds they can make, but also because of their aggressive reproduction strategy. Some species such as the *ani* and *Guira* cuckoos lay their eggs in communal nests, though they may remove others' eggs to increase the hatching probability of their own eggs [12]. Quite a number of species engage the obligate brood parasitism by laying their eggs in the nests of other host birds (often other species). There are three basic types of brood parasitism: intraspecific brood parasitism, cooperative breeding, and nest takeover. Some host birds can engage direct conflict with the intruding cuckoos. If a host bird discovers the eggs are not their owns, they will either throw these alien eggs away or simply abandon its nest and build a new nest elsewhere. Some cuckoo species such as the New World brood-parasitic *Tapera* have evolved in such a way that female parasitic cuckoos are often very specialized in the mimicry in colour and pattern of the eggs of a few chosen host species [12]. This reduces the probability of their eggs being abandoned and thus increases their reproductivity.

In addition, the timing of egg-laying of some species is also amazing. Parasitic cuckoos often choose a nest where the host bird just laid its own eggs. In general, the cuckoo eggs hatch slightly earlier than their host eggs. Once the first cuckoo chick is hatched, the first instinct action it will take is to evict the host eggs by blindly propelling the eggs out of the nest, which increases the cuckoo chick's share of food provided by its host bird. Studies also show that a cuckoo chick can also mimic the call of host chicks to gain access to more feeding opportunity.

B. Lévy Flights

On the other hand, various studies have shown that flight behaviour of many animals and insects has demonstrated the typical characteristics of Lévy flights [4], [15], [13], [14]. A recent study by Reynolds and Frye shows that fruit flies or *Drosophila melanogaster*, explore their landscape using a series of straight flight paths punctuated by a sudden 90° turn, leading to a Lévy-flight-style intermittent scale free search pattern. Studies on human behaviour such as the Ju/'hoansi hunter-gatherer foraging patterns also show the typical feature of Lévy flights. Even light can be related to Lévy flights [1].

978-1-4244-5612-3/09/$26.00 ©2009 IEEE

Cuckoo Search via Lévy Flights

```
begin
  Objective function f(x),  x = (x_1, ..., x_d)^T
  Generate initial population of
      n host nests x_i (i = 1, 2, ..., n)
  while (t <MaxGeneration) or (stop criterion)
    Get a cuckoo randomly by Lévy flights
      evaluate its quality/fitness F_i
    Choose a nest among n (say, j) randomly
    if (F_i > F_j),
        replace j by the new solution;
    end
    A fraction (p_a) of worse nests
        are abandoned and new ones are built;
    Keep the best solutions
        (or nests with quality solutions);
    Rank the solutions and find the current best
  end while
  Postprocess results and visualization
end
```

Fig. 1. Pseudo code of the Cuckoo Search (CS).

Subsequently, such behaviour has been applied to optimization and optimal search, and preliminary results show its promising capability [13], [15], [19], [20].

III. Cuckoo Search

For simplicity in describing our new Cuckoo Search, we now use the following three idealized rules: 1) Each cuckoo lays one egg at a time, and dump its egg in randomly chosen nest; 2) The best nests with high quality of eggs will carry over to the next generations; 3) The number of available host nests is fixed, and the egg laid by a cuckoo is discovered by the host bird with a probability $p_a \in [0, 1]$. In this case, the host bird can either throw the egg away or abandon the nest, and build a completely new nest. For simplicity, this last assumption can be approximated by the fraction p_a of the n nests are replaced by new nests (with new random solutions).

For a maximization problem, the quality or fitness of a solution can simply be proportional to the value of the objective function. Other forms of fitness can be defined in a similar way to the fitness function in genetic algorithms. For simplicity, we can use the following simple representations that each egg in a nest represents a solution, and a cuckoo egg represent a new solution, the aim is to use the new and potentially better solutions (cuckoos) to replace a not-so-good solution in the nests. Of course, this algorithm can be extended to the more complicated case where each nest has multiple eggs representing a set of solutions. For this present work, we will use the simplest approach where each nest has only a single egg.

Based on these three rules, the basic steps of the Cuckoo Search (CS) can be summarized as the pseudo code shown in Fig. 1.

When generating new solutions $\mathbf{x}^{(t+1)}$ for, say, a cuckoo i, a Lévy flight is performed

$$\mathbf{x}_i^{(t+1)} = \mathbf{x}_i^{(t)} + \alpha \oplus \text{Lévy}(\lambda), \qquad (1)$$

where $\alpha > 0$ is the step size which should be related to the scales of the problem of interests. In most cases, we can use $\alpha = 1$. The above equation is essentially the stochastic equation for random walk. In general, a random walk is a Markov chain whose next status/location only depends on the current location (the first term in the above equation) and the transition probability (the second term). The product $\oplus$ means entrywise multiplications. This entrywise product is similar to those used in PSO, but here the random walk via Lévy flight is more efficient in exploring the search space as its step length is much longer in the long run.

The Lévy flight essentially provides a random walk while the random step length is drawn from a Lévy distribution

$$\text{Lévy} \sim u = t^{-\lambda}, \qquad (1 < \lambda \leq 3), \qquad (2)$$

which has an infinite variance with an infinite mean. Here the steps essentially form a random walk process with a power-law step-length distribution with a heavy tail. Some of the new solutions should be generated by Lévy walk around the best solution obtained so far, this will speed up the local search. However, a substantial fraction of the new solutions should be generated by far field randomization and whose locations should be far enough from the current best solution, this will make sure the system will not be trapped in a local optimum.

From a quick look, it seems that there is some similarity between CS and hill-climbing in combination with some large scale randomization. But there are some significant differences. Firstly, CS is a population-based algorithm, in a way similar to GA and PSO, but it uses some sort of elitism and/or selection similar to that used in harmony search. Secondly, the randomization is more efficient as the step length is heavy-tailed, and any large step is possible. Thirdly, the number of parameters to be tuned is less than GA and PSo, and thus it is potentially more generic to adapt to a wider class of optimization problems. In addition, each nest can represent a set of solutions, CS can thus be extended to the type of meta-population algorithm.

IV. Implementation and Numerical Experiments

A. Validation and Parameter Studies

After implementation, we have to validate the algorithm using test functions with analytical or known solutions. For example, one of the many test functions we have used is the bivariate Michaelwicz function

$$f(x, y) = -\sin(x)\sin^{2m}(\frac{x^2}{\pi}) - \sin(y)\sin^{2m}(\frac{2y^2}{\pi}), \qquad (3)$$

where $m = 10$ and $(x, y) \in [0, 5] \times [0, 5]$. This function has a global minimum $f_* \approx -1.8013$ at $(2.20319, 1.57049)$. The landscape of this funciton is shown in Fig. 2. This global optimum can easily be found using Cuckoo Search, and the results are shown in Fig. 3 where the final locations of the nests are also marked with $\diamond$ in the figure. Here we have used $n = 15$ nests, $\alpha = 1$ and $p_a = 0.25$. In most of our simulations, we have used $n = 15$ to 50.

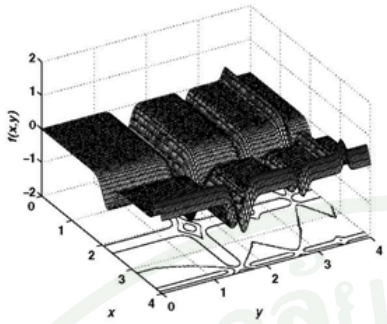


Fig. 2. The landscaped of Michaelwicz's function.

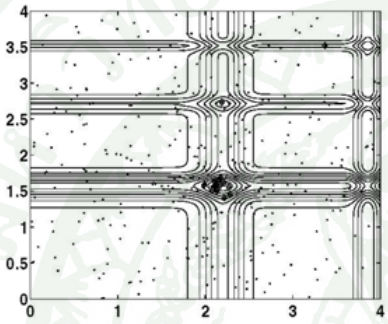


Fig. 3. Search paths of nests using Cuckoo Search. The final locations of the nests are marked with ◇ in the figure.

From the figure, we can see that, as the optimum is approaching, most nests aggregate towards the global optimum. We also notice that the nests are also distributed at different (local) optima in the case of multimodal functions. This means that CS can find all the optima simultaneously if the number of nests are much higher than the number of local optima. This advantage may become more significant when dealing with multimodal and multiobjective optimization problems.

We have also tried to vary the number of host nests (or the population size n) and the probability p_a. We have used $n = 5, 10, 15, 20, 50, 100, 150, 250, 500$ and $p_a = 0, 0.01, 0.05, 0.1, 0.15, 0.2, 0.25, 0.4, 0.5$. From our simulations, we found that $n = 15$ and $p_a = 0.25$ are sufficient for most optimization problems. Results and analysis also imply that the convergence rate, to some extent, is not sensitive to the parameters used. This means that the fine adjustment is not needed for any given problems. Therefore, we will use fixed $n = 15$ and $p_a = 0.25$ in the rest of the simulations, especially for the comparison studies presented in the next section.

B. Test Functions

There are many benchmark test functions in literature [5], [17], [16], and they are designed to test the performance of optimization algorithms. Any new optimization algorithm should also be validated and tested against these benchmark functions. In our simulations, we have used the following test functions.

De Jong's first function is essentially a sphere function

$$f(\mathbf{x}) = \sum_{i=1}^{d} x_i^2, \qquad x_i \in [-5.12, 5.12], \tag{4}$$

whose global minimum $f_* = 0$ occurs at $\mathbf{x}_* = (0, 0, ..., 0)$. Here d is the dimension.

Easom's test function is unimodal

$$f(x, y) = -\cos(x)\cos(y)\exp[-(x-\pi)^2 - (y-\pi)^2], \tag{5}$$

where

$$(x, y) \in [-100, 100] \times [-100, 100]. \tag{6}$$

It has a global minimum of $f_* = -1$ at (π, π) in a very small region.

Shubert's bivariate function

$$f(x, y) = -\sum_{i=1}^{5} i\cos[(i+1)x + 1] \sum_{i=1}^{5} \cos[(i+1)y + 1)], \tag{7}$$

has 18 global minima in the region $(x, y) \in [-10, 10] \times [-10, 10]$. The value of its global minima is $f_* = -186.7309$.

Griewangk's test function has many local minima

$$f(\mathbf{x}) = \frac{1}{4000}\sum_{i=1}^{d} x_i^2 - \prod_{i=1}^{d} \cos(\frac{x_i}{\sqrt{i}}) + 1, \tag{8}$$

but a single global mimimum $f_* = 0$ at $(0, 0, ..., 0)$ for all $-600 \le x_i \le 600$ where $i = 1, 2, ..., d$.

Ackley's function is multimodal

$$f(\mathbf{x}) = -20\exp\left[-0.2\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right] - \exp[\frac{1}{d}\sum_{i=1}^{d}\cos(2\pi x_i)] + (20 + e), \tag{9}$$

with a global minimum $f_* = 0$ at $\mathbf{x}_* = (0, 0, ..., 0)$ in the range of $-32.768 \le x_i \le 32.768$ where $i = 1, 2, ..., d$.

The generalized Rosenbrock's function is given by

$$f(\mathbf{x}) = \sum_{i=1}^{d-1}\left[(1 - x_i)^2 + 100(x_{i+1} - x_i^2)^2\right], \tag{10}$$

which has a minimum $f(\mathbf{x}_*) = 0$ at $\mathbf{x}_* = (1, 1, ..., 1)$.

Schwefel's test function is also multimodal

$$f(\mathbf{x}) = \sum_{i=1}^{d}\left[-x_i \sin(\sqrt{|x_i|}\,)\right], \quad -500 \le x_i \le 500, \tag{11}$$

with a global minimum of $f_* = -418.9829d$ at $x_i^* = 420.9687 (i = 1, 2, ..., d)$.

Rastrigin's test function

$$f(\mathbf{x}) = 10d + \sum_{i=1}^{d}[x_i^2 - 10\cos(2\pi x_i)], \qquad (12)$$

has a global minimum $f_* = 0$ at $(0, 0, ..., 0)$ in a hypercube $-5.12 \le x_i \le 5.12$ where $i = 1, 2, ..., d$.

Michalewicz's test function has $d!$ local optima

$$f(\mathbf{x}) = -\sum_{i=1}^{d} \sin(x_i)\Big[\sin(\frac{ix_i^2}{\pi})\Big]^{2m}, \quad (m = 10), \qquad (13)$$

where $0 \le x_i \le \pi$ and $i = 1, 2, ..., d$. The global mimimum is $f_* \approx -1.801$ for $d = 2$, while $f_* \approx -4.6877$ for $d = 5$.

C. Comparison of CS with PSO and GA

Recent studies indicate that PSO algorithms can outperform genetic algorithms (GA) [8] and other conventional algorithms for many optimization problems. This can partly be attributed to the broadcasting ability of the current best estimates which potentially gives better and quicker convergence towards the optimality. A general framework for evaluating statistical performance of evolutionary algorithms has been discussed in detail by Shilane et al. [18].

Now we will compare the Cuckoo Search with PSO and genetic algorithms for various standard test functions. After implementing these algorithms using Matlab, we have carried out extensive simulations and each algorithm has been run at least 100 times so as to carry out meaningful statistical analysis. The algorithms stop when the variations of function values are less than a given tolerance $\epsilon \le 10^{-5}$. The results are summarized in the following tables (see Tables 1 and 2) where the global optima are reached. The numbers are in the format: average number of evaluations (success rate), so $927 \pm 105(100\%)$ means that the average number (mean) of function evaluations is 927 with a standard deviation of 105. The success rate of finding the global optima for this algorithm is 100%.

TABLE I
COMPARISON OF CS WITH GENETIC ALGORITHMS

Functions/Algorithms	GA	CS
Multiple peaks	$52124 \pm 3277(98\%)$	$927 \pm 105(100\%)$
Michalewicz's (d=16)	$89325 \pm 7914(95\%)$	$3221 \pm 519(100\%)$
Rosenbrock's (d=16)	$55723 \pm 8901(90\%)$	$5923 \pm 1937(100\%)$
De Jong's (d=256)	$25412 \pm 1237(100\%)$	$4971 \pm 754(100\%)$
Schwefel's (d=128)	$227329 \pm 7572(95\%)$	$8829 \pm 625(100\%)$
Ackley's (d=128)	$32720 \pm 3327(90\%)$	$4936 \pm 903(100\%)$
Rastrigin's	$110523 \pm 5199(77\%)$	$10354 \pm 3755(100\%)$
Easom's	$19239 \pm 3307(92\%)$	$6751 \pm 1902(100\%)$
Griewank's	$70925 \pm 7652(90\%)$	$10912 \pm 4050(100\%)$
Shubert's (18 minima)	$54077 \pm 4997(89\%)$	$9770 \pm 3592(100\%)$

We can see that the CS is much more efficient in finding the global optima with higher success rates. Each function evaluation is virtually instantaneous on modern personal computer. For example, the computing time for 10,000 evaluations on a 3GHz desktop is about 5 seconds.

For all the test functions, CS has outperformed both GA and PSO. The primary reasons are two folds: A fine balance of randomization and intensification, and less number of control parameters. As for any metaheuristic algorithm, a good balance of intensive local search strategy and an efficient exploration of the whole search space will usually lead to a more efficient algorithm. On the other hand, there are only two parameters in this algorithm, the population size n, and p_a. Once n is fixed, p_a essentially controls the elitism and the balance of the randomization and local search. Few parameters make an algorithm less complex and thus potentially more generic. Such observations deserve more systematic research and further elaboration in the future work.

TABLE II
COMPARISON OF CS WITH PARTICLE SWARM OPTIMISATION

Functions/Algorithms	PSO	CS
Multiple peaks	$3719 \pm 205(97\%)$	$927 \pm 105(100\%)$
Michalewicz's (d=16)	$6922 \pm 537(98\%)$	$3221 \pm 519(100\%)$
Rosenbrock's (d=16)	$32756 \pm 5325(98\%)$	$5923 \pm 1937(100\%)$
De Jong's (d=256)	$17040 \pm 1123(100\%)$	$4971 \pm 754(100\%)$
Schwefel's (d=128)	$14522 \pm 1275(97\%)$	$8829 \pm 625(100\%)$
Ackley's (d=128)	$23407 \pm 4325(92\%)$	$4936 \pm 903(100\%)$
Rastrigin's	$79491 \pm 3715(90\%)$	$10354 \pm 3755(100\%)$
Easom's	$17273 \pm 2929(90\%)$	$6751 \pm 1902(100\%)$
Griewank's	$55970 \pm 4223(92\%)$	$10912 \pm 4050(100\%)$
Shubert's (18 minima)	$23992 \pm 3755(92\%)$	$9770 \pm 3592(100\%)$

V. CONCLUSIONS

In this paper, we have formulated a new metaheuristic Cuckoo Search in combination with Lévy flights, based on the breeding strategy of some cuckoo species. The proposed algorithm has been validated and compared with other algorithms such as genetic algorithms and particle swarm optimization. Simulations and comparison show that CS is superior to these existing algorithms for multimodal objective functions. This is partly due to the fact that there are fewer parameters to be fine-tuned in CS than in PSO and genetic algorithms. In fact, apart from the population size n, there is essentially one parameter p_a. Furthermore, our simulations also indicate that the convergence rate is insensitive to the parameter p_a. This also means that we do not have to fine tune these parameters for a specific problem. Subsequently, CS is more generic and robust for many optimization problems, comparing with other metaheuristic algorithms.

This potentially powerful optimization strategy can easily be extended to study multiobjecitve optimization applications with various constraints, even to NP-hard problems. Further studies can focus on the sensitivity and parameter studies and their possible relationships with the convergence rate of the algorithm. Hybridization with other popular algorithms such as PSO will also be potentially fruitful.

REFERENCES

[1] Barthelemy P., Bertolotti J., Wiersma D. S., A Lévy flight for light, *Nature*, **453**, 495-498 (2008).
[2] Bonabeau E., Dorigo M., Theraulaz G., *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, (1999)
[3] Blum C. and Roli A., Metaheuristics in combinatorial optimization: Overview and conceptural comparision, *ACM Comput. Surv.*, **35**, 268-308 (2003).

[4] Brown C., Liebovitch L. S., Glendon R., Lévy flights in Dobe Ju/'hoansi foraging patterns, *Human Ecol.*, **35**, 129-138 (2007).
[5] Chattopadhyay R., A study of test functions for optimization algorithms, *J. Opt. Theory Appl.*, **8**, 231-236 (1971).
[6] Deb. K., *Optimisation for Engineering Design*, Prentice-Hall, New Delhi, (1995).
[7] Gazi K., and Passino K. M., Stability analysis of social foraging swarms, *IEEE Trans. Sys. Man. Cyber. Part B - Cybernetics*, **34**, 539-557 (2004).
[8] Goldberg D. E., *Genetic Algorithms in Search, Optimisation and Machine Learning*, Reading, Mass.: Addison Wesley (1989).
[9] Kennedy J. and Eberhart R. C.: Particle swarm optimization. *Proc. of IEEE International Conference on Neural Networks*, Piscataway, NJ. pp. 1942-1948 (1995).
[10] Kennedy J., Eberhart R., Shi Y.: *Swarm intelligence*, Academic Press, (2001).
[11] Passino K. M., *Biomimicrt of Bacterial Foraging for Distributed Optimization*, University Press, Princeton, New Jersey (2001).
[12] Payne R. B., Sorenson M. D., and Klitz K., *The Cuckoos*, Oxford University Press, (2005).
[13] Pavlyukevich I., Lévy flights, non-local search and simulated annealing, *J. Computational Physics*, **226**, 1830-1844 (2007).
[14] Pavlyukevich I., Cooling down Lévy flights, *J. Phys. A:Math. Theor.*, **40**, 12299-12313 (2007).
[15] Reynolds A. M. and Frye M. A., Free-flight odor tracking in Drosophila is consistent with an optimal intermittent scale-free search, *PLoS One*, **2**, e354 (2007).
[16] Schoen F., A wide class of test functions for global optimization, *J. Global Optimization*, **3**, 133-137, (1993).
[17] Shang Y. W., Qiu Y. H., A note on the extended rosenrbock function, *Evolutionary Computation*, **14**, 119-126 (2006).
[18] Shilane D., Martikainen J., Dudoit S., Ovaska S. J., A general framework for statistical performance comparison of evolutionary computation algorithms, *Information Sciences: an Int. Journal*, **178**, 2870-2879 (2008).
[19] Shlesinger M. F., Zaslavsky G. M. and Frisch U. (Eds), *Lévy Flights and Related Topics in Phyics*, Springer, (1995).
[20] Shlesinger M. F., Search research, *Nature*, **443**, 281-282 (2006).
[21] Yang X. S., *Nature-Inspired Metaheuristic Algorithms*, Luniver Press, (2008).
[22] Yang X. S., Biology-derived algorithms in engineering optimizaton (Chapter 32), in *Handbook of Bioinspired Algorithms and Applications* (eds Olarius & Zomaya), Chapman & Hall / CRC (2005).

Engineering Optimisation by Cuckoo Search

Xin-She Yang
Department of Engineering
University of Cambridge
Trumpington Street
Cambridge CB2 1PZ, UK

Suash Deb
Dept of Computer Science & Engineering
C. V. Raman College of Engineering
Bidyanagar, Mahura, Janla
Bhubaneswar 752054, INDIA

Abstract

A new metaheuristic optimisation algorithm, called Cuckoo Search (CS), was developed recently by Yang and Deb (2009). This paper presents a more extensive comparison study using some standard test functions and newly designed stochastic test functions. We then apply the CS algorithm to solve engineering design optimisation problems, including the design of springs and welded beam structures. The optimal solutions obtained by CS are far better than the best solutions obtained by an efficient particle swarm optimiser. We will discuss the unique search features used in CS and the implications for further research.

Reference to this paper should be made as follows:
Yang, X.-S., and Deb, S. (2010), "Engineering Optimisation by Cuckoo Search", *Int. J. Mathematical Modelling and Numerical Optimisation*, Vol. **1**, No. 4, 330–343 (2010).

1 Introduction

Most design optimisation problems in engineering are often highly nonlinear, involving many different design variables under complex constraints. These constraints can be written either as simple bounds such as the ranges of material properties, or as nonlinear relationships including maximum stress, maximum deflection, minimum load capacity, and geometrical configuration. Such nonlinearity often results in multimodal response landscape. Subsequently, local search algorithms such as hill-climbing and Nelder-Mead downhill simplex methods are not suitable, only global algorithms should be used so as to obtain optimal solutions (Deb 1995, Arora 1989, Yang 2005, Yang 2008).

Modern metaheuristic algorithms have been developed with an aim to carry out global search, typical examples are genetic algorithms (Glodberg 1989), particle swarm optimisation (PSO) (Kennedy and Eberhart 1995, Kennedy et al 2001). The efficiency of metaheuristic algorithms can be attributed to the fact that they imitate the best features in nature, especially the selection of the fittest in biological systems which have evolved by natural selection over millions of years. Two important characteristics of metaheuristics are: intensification and diversification (Blum and Roli 2003, Gazi and Passino 2004, Yang 2009). Intensification intends to search

around the current best solutions and select the best candidates or solutions, while diversification makes sure that the algorithm can explore the search space more efficiently, often by randomization.

Recently, a new metaheuristic search algorithm, called Cuckoo Search (CS), has been developed by Yang and Deb (2009). Preliminary studies show that it is very promising and could outperform existing algorithms such as PSO. In this paper, we will further study CS and validate it against test functions including stochastic test functions. Then, we will apply it to solve design optimisation problems in engineering. Finally, we will discuss the unique features of Cuckoo Search and propose topics for further studies.

2 Cuckoo Search

In order to describe the Cuckoo Search more clearly, let us briefly review the interesting breed behaviour of certain cuckoo species. Then, we will outline the basic ideas and steps of the proposed algorithm.

2.1 Cuckoo Breeding Behaviour

Cuckoo are fascinating birds, not only because of the beautiful sounds they can make, but also because of their aggressive reproduction strategy. Some species such as the *ani* and *Guira* cuckoos lay their eggs in communal nests, though they may remove others' eggs to increase the hatching probability of their own eggs (Payne et al 2005). Quite a number of species engage the obligate brood parasitism by laying their eggs in the nests of other host birds (often other species). There are three basic types of brood parasitism: intraspecific brood parasitism, cooperative breeding, and nest takeover. Some host birds can engage direct conflict with the intruding cuckoos. If a host bird discovers the eggs are not its owns, it will either throw these alien eggs away or simply abandons its nest and builds a new nest elsewhere. Some cuckoo species such as the New World brood-parasitic *Tapera* have evolved in such a way that female parasitic cuckoos are often very specialized in the mimicry in colour and pattern of the eggs of a few chosen host species (Payne et al 2005). This reduces the probability of their eggs being abandoned and thus increases their reproductivity.

Furthermore, the timing of egg-laying of some species is also amazing. Parasitic cuckoos often choose a nest where the host bird just laid its own eggs. In general, the cuckoo eggs hatch slightly earlier than their host eggs. Once the first cuckoo chick is hatched, the first instinct action it will take is to evict the host eggs by blindly propelling the eggs out of the nest, which increases the cuckoo chick's share of food provided by its host bird (Payne et al 2005). Studies also show that a cuckoo chick can also mimic the call of host chicks to gain access to more feeding opportunity.

2.2 Lévy Flights

In nature, animals search for food in a random or quasi-random manner. In general, the foraging path of an animal is effectively a random walk because the next move is based on the current location/state and the transition probability to the next location. Which direction it chooses depends implicitly on a probability which can be modelled mathematically. For example, various studies have shown that the flight

behaviour of many animals and insects has demonstrated the typical characteristics of Lévy flights (Brown et al 2007, Reynods and Frye 2007, Pavlyukevich 2007).

A recent study by Reynolds and Frye (2007) shows that fruit flies or *Drosophila melanogaster*, explore their landscape using a series of straight flight paths punctuated by a sudden 90^o turn, leading to a Lévy-flight-style intermittent scale-free search pattern. Studies on human behaviour such as the Ju/'hoansi hunter-gatherer foraging patterns also show the typical feature of Lévy flights. Even light can be related to Lévy flights (Barthelemy et al 2008). Subsequently, such behaviour has been applied to optimization and optimal search, and preliminary results show its promising capability (Shlesinger 2006, Pavlyukevich 2007).

2.3 Cuckoo Search

For simplicity in describing our new Cuckoo Search (Yang and Deb 2009), we now use the following three idealized rules:

- Each cuckoo lays one egg at a time, and dumps it in a randomly chosen nest;
- The best nests with high quality of eggs (solutions) will carry over to the next generations;
- The number of available host nests is fixed, and a host can discover an alien egg with a probability $p_a \in [0, 1]$. In this case, the host bird can either throw the egg away or abandon the nest so as to build a completely new nest in a new location.

For simplicity, this last assumption can be approximated by a fraction p_a of the n nests being replaced by new nests (with new random solutions at new locations). For a maximization problem, the quality or fitness of a solution can simply be proportional to the objective function. Other forms of fitness can be defined in a similar way to the fitness function in genetic algorithms.

Based on these three rules, the basic steps of the Cuckoo Search (CS) can be summarised as the pseudo code shown in Fig. 1.

When generating new solutions $\boldsymbol{x}^{(t+1)}$ for, say cuckoo i, a Lévy flight is performed

$$\boldsymbol{x}_i^{(t+1)} = \boldsymbol{x}_i^{(t)} + \alpha \oplus \text{Lévy}(\lambda), \tag{1}$$

where $\alpha > 0$ is the step size which should be related to the scales of the problem of interest. In most cases, we can use $\alpha = O(1)$. The product $\oplus$ means entry-wise multiplications. Lévy flights essentially provide a random walk while their random steps are drawn from a Lévy distribution for large steps

$$\text{Lévy} \sim u = t^{-\lambda}, \quad (1 < \lambda \leq 3), \tag{2}$$

which has an infinite variance with an infinite mean. Here the consecutive jumps/steps of a cuckoo essentially form a random walk process which obeys a power-law step-length distribution with a heavy tail.

It is worth pointing out that, in the real world, if a cuckoo's egg is very similar to a host's eggs, then this cuckoo's egg is less likely to be discovered, thus the fitness should be related to the difference in solutions. Therefore, it is a good idea to do a random walk in a biased way with some random step sizes. A demo version is attached in the Appendix (this demo is not published in the actual paper, but as a supplement to help readers to implement the cuckoo search correctly).

Objective function $f(\boldsymbol{x})$, $\boldsymbol{x} = (x_1, ..., x_d)^T$;
Initial a population of n host nests $\boldsymbol{x}_i$ $(i = 1, 2, ..., n)$;
while (t <MaxGeneration) or (stop criterion);
  Get a cuckoo (say i) randomly by Lévy flights;
  Evaluate its quality/fitness F_i;
  Choose a nest among n (say j) randomly;
  if $(F_i > F_j)$,
    Replace j by the new solution;
  end
  Abandon a fraction (p_a) of worse nests
    [and build new ones at new locations via Lévy flights];
  Keep the best solutions (or nests with quality solutions);
  Rank the solutions and find the current best;
end while
Postprocess results and visualisation;

Figure 1: Cuckoo Search (CS).

3 Implementation and Validation

3.1 Validation and Parameter Studies

It is relatively easy to implement the algorithm, and then we have to benchmark it using test functions with analytical or known solutions. There are many benchmark test functions and there is no standard list or collection, though extensive descriptions of various functions do exist in literature (Floudas et al 1999, Hedar 2005, Molga and Smutnicki 2005). For example, Michalewicz's test function has many local optima

$$f(\boldsymbol{x}) = -\sum_{i=1}^{d} \sin(x_i)\Big[\sin(\frac{ix_i^2}{\pi})\Big]^{2m}, \quad (m = 10), \tag{3}$$

in the domain $0 \le x_i \le \pi$ for $i = 1, 2, ..., d$ where d is the number of dimensions. The global mimimum $f_* \approx -1.801$ occurs at $(2.20319, 1.57049)$ for $d = 2$, while $f_* \approx -4.6877$ for $d = 5$. In the 2D case, its 3D landscape is shown Fig. 2.

The global optimum in 2D can easily be found using Cuckoo Search, and the results are shown in Fig. 3 where the final locations of the nests are marked with $\diamond$. Here we have used $n = 20$ nests, $\alpha = 1$ and $p_a = 0.25$. From the figure, we can see that, as the optimum is approaching, most nests aggregate towards the global optimum. In various simulations, we also notice that nests are also distributed at different (local) optima in the case of multimodal functions. This means that CS can find all the optima simultaneously if the number of nests are much higher than the number of local optima. This advantage may become more significant when dealing with multimodal and multiobjective optimization problems.

We have also tried to vary the number of host nests (or the population size n) and the probability p_a. We have used $n = 5, 10, 15, 20, 50, 100, 150, 250, 500$ and $p_a = 0, 0.01, 0.05, 0.1, 0.15, 0.2, 0.25,\ 0.4, 0.5$. From our simulations, we found that $n = 15$ to 25 and $p_a = 0.15$ to 0.30 are sufficient for most optimization problems.

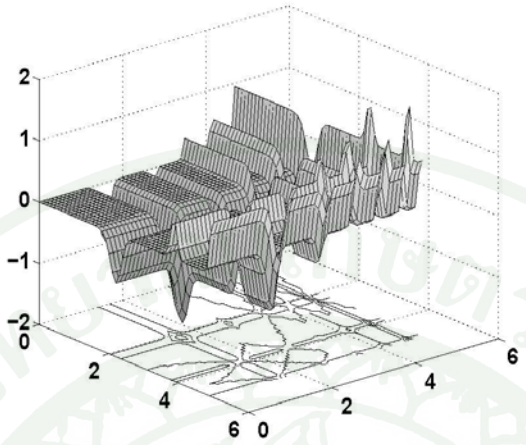


Figure 2: The landscape of Michaelwicz's 2D function.

Results and analysis also imply that the convergence rate, to some extent, is not sensitive to the parameters used. This means that the fine adjustment of algorithm-dependent parameters is not needed for any given problems. Therefore, we will use $n = 20$ and $p_a = 0.25$ in the rest of the simulations, especially for the comparison studies presented later.

3.2 Standard Test Functions

Various test functions in literature are designed to test the performance of optimization algorithms (Chattopadhyay 1971, Schoen 1993, Shang and Qiu 2006). Any new optimization algorithm should also be validated and tested against these benchmark functions. In our simulations, we have used the following test functions.

De Jong's first function is essentially a sphere function

$$f(\boldsymbol{x}) = \sum_{i=1}^{d} x_i^2, \quad x_i \in [-5.12, 5.12], \tag{4}$$

whose global minimum $f(\boldsymbol{x}_*) = 0$ occurs at $\boldsymbol{x}_* = (0, 0, ..., 0)$. Here d is the dimension.

The generalized Rosenbrock's function is given by

$$f(\boldsymbol{x}) = \sum_{i=1}^{d-1} \left[(1 - x_i)^2 + 100(x_{i+1} - x_i^2)^2\right], \tag{5}$$

which has a unique global minimum $f_* = 0$ at $\boldsymbol{x}_* = (1, 1, ..., 1)$.

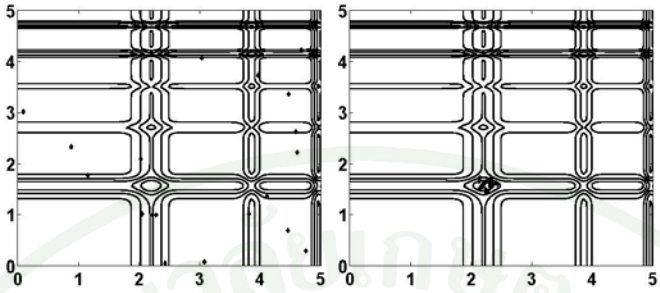


Figure 3: Initial locations of 20 nests in Cuckoo Search, and their final locations are marked with $\diamond$.

Schwefel's test function is multimodal

$$f(\boldsymbol{x}) = \sum_{i=1}^{d} \Big[- x_i \sin(\sqrt{|x_i|}\,) \Big], \quad -500 \le x_i \le 500, \tag{6}$$

whose global minimum $f_* = -418.9829d$ is at $x_i^* = 420.9687 (i = 1, 2, ..., d)$.

Ackley's function is also multimodal

$$f(\boldsymbol{x}) = -20 \exp\left[- 0.2 \sqrt{\frac{1}{d} \sum_{i=1}^{d} x_i^2} \right] - \exp[\frac{1}{d} \sum_{i=1}^{d} \cos(2\pi x_i)] + (20 + e), \tag{7}$$

with the global minimum $f_* = 0$ at $\boldsymbol{x}_* = (0, 0, ..., 0)$ in the range of $-32.768 \le x_i \le 32.768$ where $i = 1, 2, ..., d$.

Rastrigin's test function

$$f(\boldsymbol{x}) = 10d + \sum_{i=1}^{d} [x_i^2 - 10 \cos(2\pi x_i)], \tag{8}$$

has a unique global minimum $f_* = 0$ at $(0, 0, ..., 0)$ in a hypercube $-5.12 \le x_i \le 5.12$ where $i = 1, 2, ..., d$.

Easom's test function has a sharp tip

$$f(x, y) = -\cos(x) \cos(y) \exp[-(x - \pi)^2 - (y - \pi)^2], \tag{9}$$

in the domain $(x, y) \in [-100, 100] \times [-100, 100]$. It has a global minimum of $f_* = -1$ at (π, π) in a very small region.

Griewangk's test function has many local minima

$$f(\boldsymbol{x}) = \frac{1}{4000} \sum_{i=1}^{d} x_i^2 - \prod_{i=1}^{d} \cos(\frac{x_i}{\sqrt{i}}) + 1, \tag{10}$$

but a unique global mimimum $f_* = 0$ at $(0, 0, ..., 0)$ for all $-600 \le x_i \le 600$ where $i = 1, 2, ..., d$.

3.3 Stochastic Test Functions

Almost all the test functions in literature are deterministic. It is usually more difficult for algorithms to deal with stochastic functions. We have designed some stochastic test functions for such a purpose.

The first test function designed by Yang (2010) looks like a standing-wave function with a region of defects

$$f(\boldsymbol{x}) = \left[e^{-\sum_{i=1}^{d}(x_i/\beta)^{2m}} - 2e^{-\sum_{i=1}^{d}\epsilon_i(x_i-\pi)^2}\right] \cdot \prod_{i=1}^{d}\cos^2 x_i, \quad m = 5, \tag{11}$$

which has many local minima and the unique global minimum $f_* = -1$ at $\boldsymbol{x}_* = (\pi, \pi, ..., \pi)$ for β=15 within the domain $-20 \le x_i \le 20$ for $i = 1, 2, ..., d$. Here the random variables ϵ_i $(i = 1, 2, ..., d)$ are uniformly distributed in $(0, 1)$. For example, if all ϵ_i are relatively small (say order of 0.05), a snapshot of the landscape in 2D is shown in Fig. 4, while for higher values such as 0.5 the landscape is different, also shown in Fig. 4.

Yang's second test function is also multimodal but it has a singularity

$$f(\boldsymbol{x}) = \Big(\sum_{i=1}^{d}\epsilon_i|x_i|\Big)\exp\Big[-\sum_{i=1}^{d}\sin(x_i^2)\Big], \tag{12}$$

which has a unique global minimum $f_* = 0$ at $\boldsymbol{x}_* = (0, 0, ..., 0)$ in the domain $-2\pi \le x_i \le 2\pi$ where $i = 1, 2, ..., d$ (Yang 2010). This function is singular at the optimum $(0, ..., 0)$. Similarly, ϵ_i should be drawn from a uniform distribution in $[0, 1]$ or Unif[0,1]. In fact, using the same methodology, we can turn many determistic functions into stochastic test functions. For example, we can extend Robsenbrock's function as the following stochastic function

$$f(\boldsymbol{x}) = \sum_{i=1}^{d-1}\Big[(1 - x_i)^2 + 100\epsilon_i(x_{i+1} - x_i^2)^2\Big], \tag{13}$$

where ϵ_i should be drawn from Unif[0,1]. Similarly, we can also extend De Jong's function into its corresponding stochastic form

$$f(\boldsymbol{x}) = \sum_{i=1}^{d}\epsilon_i x_i^2, \tag{14}$$

which still has the same global minimum $f_* = 0$ at $(0, 0, ..., 0)$, despite its stochastic nature due to the factor ϵ_i. For stochastic functions, most deterministic algorithms such as hill climbing and Nelder-Mead downhill simplex method would simply fail. However, we can see later that most metaheuristic algorithms such as PSO and CS are still robust.

3.4 Simulations and Comparison

Recent studies indicate that PSO can outperform genetic algorithms (GA) and other conventional algorithms (Goldberg 1989, Kennedy et al 2001, Yang 2008). This can be attributed partly to the broadcasting ability of the current best estimates, potentially leading to a better and quicker convergence rate towards the optimality. A

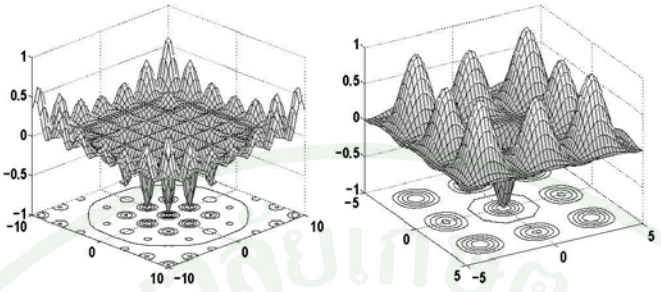


Figure 4: Landscape of stochastic function (11) for small ϵ (left) and large ϵ (right).

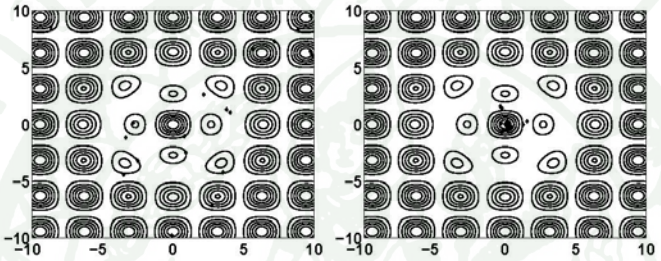


Figure 5: The initial location of 20 nests (left) for function (11) and their final locations after 15 iterations (right).

general framework for evaluating statistical performance of evolutionary algorithms has been discussed in detail by Shilane et al (2008).

Now we can compare the Cuckoo Search with PSO and genetic algorithms for various test functions. After implementing these algorithms using Matlab, we have carried out extensive simulations and each algorithm has been run at least 100 times so as to carry out meaningful statistical analysis. The algorithms stop when the variations of function values are less than a given tolerance $\epsilon \leq 10^{-5}$. The results are summarised in Table 1 where the numbers are in the format: average number of evaluations $\pm$ one standard deviation (success rate), so $3321 \pm 519(100\%)$ means that the average number (mean) of function evaluations is 3321 with a standard deviation of 519. The success rate of finding the global optima for this algorithm is 100%. The functions used in the Table are (1) Michaelwicz ($d = 16$), (2) Rosenrbrock ($d = 16$), (3) De Jong ($d = 32$), (4) Schwefel ($d = 32$), (5) Ackley ($d = 128$), (6) Rastrigin, (7) Easom, (8) Griewangk, (9) Yang's first stochastic function, (10) Yang's second stochastic function, (11) Generalised Robsenbrock's function with stochastic components, and (12) De Jong's stochastic function.

We can see that CS is much more efficient in finding the global optima with

Table 1: Comparison of CS with genetic algorithms and particle swarm optimisation

Functions	GA	PSO	CS
(1)	$89325 \pm 7914(95\%)$	$6922 \pm 537(98\%)$	$3221 \pm 519(100\%)$
(2)	$55723 \pm 8901(90\%)$	$32756 \pm 5325(98\%)$	$5923 \pm 1937(100\%)$
(3)	$15232 \pm 1270(100\%)$	$10079 \pm 970(100\%)$	$3015 \pm 540(100\%)$
(4)	$23790 \pm 6523(95\%)$	$92411 \pm 1163(97\%)$	$4710 \pm 592(100\%)$
(5)	$32720 \pm 3327(90\%)$	$23407 \pm 4325(92\%)$	$4936 \pm 903(100\%)$
(6)	$110523 \pm 5199(77\%)$	$79491 \pm 3715(90\%)$	$10354 \pm 3755(100\%)$
(7)	$19239 \pm 3307(92\%)$	$17273 \pm 2929(90\%)$	$6751 \pm 1902(100\%)$
(8)	$70925 \pm 7652(90\%)$	$55970 \pm 4223(92\%)$	$10912 \pm 4050(100\%)$
(9)	$79025 \pm 6312(49\%)$	$34056 \pm 4470(90\%)$	$11254 \pm 2733(99\%)$
(10)	$35072 \pm 3730(54\%)$	$22360 \pm 2649(92\%)$	$8669 \pm 3480(98\%)$
(11)	$63268 \pm 5091(40\%)$	$49152 \pm 6505(89\%)$	$10564 \pm 4297(99\%)$
(12)	$24164 \pm 4923(68\%)$	$11780 \pm 4912(94\%)$	$7723 \pm 2504(100\%)$

higher success rates. Each function evaluation is virtually instantaneous on a modern personal computer. For example, the computing time for 10,000 evaluations on a 3GHz desktop is about 5 seconds. In addition, for stochastic functions, genetic algorithms do not perform well, while PSO is better. However, CS is far more promising.

4 Engineering Design

Design optimisation is an integrated part of designing any new products in engineering and industry. Most design problems are complex and multiobjective, sometimes even the optimal solutions of interest do not exist. In order to see how the CS algorithm may perform, we now use two standard but well-known test problems.

4.1 Spring Design Optimisation

Tensional and/or compressional springs are used widely in engineering. A standard spring design problem has three design variables: the wire diameter w, the mean coil diameter d, and the length (or number of coils) L.

The objective is to minimise the weight of the spring, subject to various constraints such as maximum shear stress, minimum deflection, and geometrical limits. For detailed description, please refer to earlier studies (Belegundu 1982, Arora 1989, Cagnina et al 2008). This problem can be written compactly as

$$\text{Minimise } f(\boldsymbol{x}) = (L+2)w^2 d, \tag{15}$$

subject to

$$\begin{aligned} g_1(\boldsymbol{x}) &= 1 - \frac{d^3 L}{71785 w^4} \leq 0, \\ g_2(\boldsymbol{x}) &= 1 - \frac{140.45 w}{d^2 L} \leq 0, \\ g_3(\boldsymbol{x}) &= \frac{2(w+d)}{3} - 1 \leq 0, \\ g_4(\boldsymbol{x}) &= \frac{d(4d-w)}{w^3(12566d-w)} + \frac{1}{5108 w^2} - 1 \leq 0, \end{aligned} \tag{16}$$

with the following limits

$$0.05 \leq w \leq 2.0, \quad 0.25 \leq d \leq 1.3, \quad 2.0 \leq L \leq 15.0. \tag{17}$$

Using Cuckoo Search, we have obtained the same or slightly better solutions than the best solution obtained by Cagnina et al (2008)

$$f_* = 0.012665 \quad \text{at} \quad (0.051690, 0.356750, 11.287126), \tag{18}$$

but cuckoo search uses significantly fewer evaluations.

4.2 Welded Beam Design

The so-called welded beam design is another standard test problem for constrained design optimisation (Ragsdell and Phillips 1976, Cagnina et al 2008). The problem has four design variables: the width w and length L of the welded area, the depth h and thickness h of the main beam. The objective is to minimise the overall fabrication cost, under the appropriate constraints of shear stress τ, bending stress σ, buckling load P and maximum end deflection δ.

The problem can be written as

$$\text{minimise} \quad f(\boldsymbol{x}) = 1.10471 w^2 L + 0.04811 dh(14.0 + L), \tag{19}$$

subject to

$$\begin{aligned} & g_1(\boldsymbol{x}) = w - h \leq 0, \\ & g_2(\boldsymbol{x}) = \delta(\boldsymbol{x}) - 0.25 \leq 0, \\ & g_3(\boldsymbol{x}) = \tau(\boldsymbol{x}) - 13,600 \leq 0, \\ & g_4(\boldsymbol{x}) = \sigma(\boldsymbol{x}) - 30,000 \leq 0, \\ & g_5(\boldsymbol{x}) = 0.10471 w^2 + 0.04811 hd(14 + L) - 5.0 \leq 0, \\ & g_6(\boldsymbol{x}) = 0.125 - w \leq 0, \\ & g_7(\boldsymbol{x}) = 6000 - P(\boldsymbol{x}) \leq 0, \end{aligned} \tag{20}$$

where

$$\sigma(\boldsymbol{x}) = \frac{504{,}000}{hd^2}, \qquad Q = 6000(14 + \frac{L}{2}),$$

$$D = \frac{1}{2}\sqrt{L^2 + (w+d)^2}, \qquad J = \sqrt{2}\, wL[\frac{L^2}{6} + \frac{(w+d)^2}{2}],$$

$$\delta = \frac{65{,}856}{30{,}000hd^3}, \qquad \beta = \frac{QD}{J}, \tag{21}$$

$$\alpha = \frac{6000}{\sqrt{2}wL}, \qquad \tau(\boldsymbol{x}) = \sqrt{\alpha^2 + \frac{\alpha\beta L}{D} + \beta^2},$$

$$P = 0.61423 \times 10^6 \frac{dh^3}{6}(1 - \frac{d\sqrt{30/48}}{28}).$$

The simple limits or bounds are $0.1 \le L, d \le 10$ and $0.1 \le w, h \le 2.0$.

Using our Cuckoo Search, we have the following optimal solution

$$\boldsymbol{x}_* = (w, L, d, h)$$

$$= (0.205729639786079, 3.470488665627977, 9.036623910357633, 0.205729639786079), \tag{22}$$

with

$$f(\boldsymbol{x}*)_{\min} = 1.724852308597361. \tag{23}$$

This solution is exactly the same as the solution obtained by Cagnina et al (2008)

$$f_* = 1.724852 \quad \text{at} \quad (0.205730, 3.470489, 9.036624, 0.205729). \tag{24}$$

We have seen that, for both test problems, CS has found the optimal solutions which are either better than or the same as the solutions found so far in literature.

5 Discussions and Conclusions

From the comparison study of the performance of CS with GAs and PSO, we know that our new Cuckoo Search in combination with Lévy flights is very efficient and proves to be superior for almost all the test problems. This is partly due to the fact that there are fewer parameters to be fine-tuned in CS than in PSO and genetic algorithms. In fact, apart from the population size n, there is essentially one parameter p_a. If we look at the CS algorithm carefully, there are essentially three components: selection of the best, exploitation by local random walk, and exploration by randomization via Lévy flights globally.

The selection of the best by keeping the best nests or solutions is equivalent to some form of elitism commonly used in genetic algorithms, which ensures the best solution is passed onto the next iteration and there is no risk that the best solutions are cast out of the population. The exploitation around the best solutions is performed by using a local random walk

$$\boldsymbol{x}^{t+1} = \boldsymbol{x}^t + \alpha\varepsilon_t. \tag{25}$$

If ε_t obeys a Gaussian distribution, this becomes a standard random walk indeed. This is equivalent to the crucial step in pitch adjustment in Harmony Search (Geem

et al 2001, Yang 2009). If ε_t is drawn from a Lévy distribution, the step of move is larger, and could be potentially more efficient. However, if the step is too large, there is risk that the move is too far away. Fortunately, the elitism by keeping the best solutions makes sure that the exploitation moves are within the neighbourhood of the best solutions locally.

On the other hand, in order to sample the search space effectively so that new solutions to be generated are diverse enough, the exploration step is carried out in terms of Lévy flights. In contrast, most metaheuristic algorithms use either uniform distributions or Gaussian to generate new explorative moves (Geem et al 2001, Blum and Rilo 2003). If the search space is large, Lévy flights are usually more efficient. A good combination of the above three components can thus lead to an efficient algorithm such as Cuckoo Search.

Furthermore, our simulations also indicate that the convergence rate is insensitive to the algorithm-dependent parameters such as p_a. This also means that we do not have to fine tune these parameters for a specific problem. Subsequently, CS is more generic and robust for many optimisation problems, comparing with other metaheuristic algorithms.

This potentially powerful optimisation strategy can easily be extended to study multiobjecitve optimization applications with various constraints, including NP-hard problems. Further studies can focus on the sensitivity and parameter studies and their possible relationships with the convergence rate of the algorithm. In addition, hybridization with other popular algorithms such as PSO will also be potentially fruitful. More importantly, as for most metaheuristic algorithms, mathematical analysis of the algorithm structures is highly needed. At the moment, no such framework exists for analyzing metaheuristics in general. Any progress in this area will potentially provide new insight into the understanding of how and why metaheuristic algorithms work.

References

[1] Arora, J., 1989. *Introduction to Optimum Design*, McGraw-Hill.

[2] Belegundu, A., 1982. 'A study of mathematical programming methods for structural optimization', PhD thesis, Department of Civil Environmental Engineering, University of Iowa, USA.

[3] Barthelemy, P., Bertolotti, J., Wiersma, D. S., 2008. 'A Lévy flight for light', *Nature*, **453**, 495-498.

[4] Blum, C. and Roli, A., 2003. 'Metaheuristics in combinatorial optimization: Overview and conceptural comparision', *ACM Comput. Surv.*, **35**, 268-308.

[5] Brown, C., Liebovitch, L. S., Glendon, R., 2007. 'Lévy flights in Dobe Ju/'hoansi foraging patterns', *Human Ecol.*, **35**, 129-138.

[6] Cagnina, L. C., Esquivel, S. C., and Coello, C. A., 2008. 'Solving engineering optimization problems with the simple constrained particle swarm optimizer', *Informatica*, **32**, 319-326.

[7] Chattopadhyay, R., 1971. 'A study of test functions for optimization algorithms', *J. Opt. Theory Appl.*, **8**, 231-236.

[8] Deb, K., 1995. *Optimisation for Engineering Design*, Prentice-Hall, New Delhi.

[9] Floudas, C. A., Pardalos, P. M., Adjiman, C. S., Esposito, W. R., Gumus, Z. H., Harding, S. T., Klepeis, J. L., Meyer, C. A., Scheiger, C. A., 1999. *Handbook of Test Problems in Local and Global Optimization*, Springer, 1999.

[10] Gazi, K., and Passino, K. M., 2004. Stability analysis of social foraging swarms, *IEEE Trans. Sys. Man. Cyber. Part B - Cybernetics*, **34**, 539-557.

[11] Geem, Z. W., Kim, J. H., Loganathan, G. V., 2001. 'A new heuristic optimization algorithm: Harmony search', *Simulation*, **76**, 60-68.

[12] Goldberg, D. E., 1989. *Genetic Algorithms in Search, Optimisation and Machine Learning*, Reading, Mass., Addison Wesley.

[13] Hedar, A., 2005, 'Test function web pages', http://www-optima.amp.i.kyoto-u.ac.jp /member/student/hedar/Hedar_files/TestGO_files/Page364.htm

[14] Kennedy, J. and Eberhart, R. C., 1995. 'Particle swarm optimization'. *Proc. of IEEE International Conference on Neural Networks*, Piscataway, NJ. pp. 1942-1948.

[15] Kennedy, J., Eberhart, R. C., Shi, Y., 2001. *Swarm intelligence*, Academic Press.

[16] Molga, M., Smutnicki, C., 2005. "Test functions for optimization needs", http://www.zsd.ict.pwr.wroc.pl/files/docs/functions.pdf

[17] Passino, K. M., 2001. *Biomimicry of Bacterial Foraging for Distributed Optimization*, University Press, Princeton, New Jersey.

[18] Payne, R. B., Sorenson, M. D., and Klitz, K.,2005. *The Cuckoos*, Oxford University Press.

[19] Pavlyukevich, I., 2007. 'Lévy flights, non-local search and simulated annealing', *J. Computational Physics*, **226**, 1830-1844.

[20] Ragsdell, K. and Phillips, D.,1976. 'Optimal design of a class of welded structures using geometric programming', *J. Eng. Ind.*, **98**, 1021-1025.

[21] Reynolds, A. M. and Frye, M. A., 2007. 'Free-flight odor tracking in Drosophila is consistent with an optimal intermittent scale-free search', *PLoS One*, **2**, e354.

[22] Schoen, F., 1993. 'A wide class of test functions for global optimization', *J. Global Optimization*, **3**, 133-137.

[23] Shang, Y. W., Qiu Y. H., 2006. 'A note on the extended rosenrbock function', *Evolutionary Computation*, **14**, 119-126.

[24] Shilane D., Martikainen J., Dudoit S., Ovaska S. J., 2008. 'A general framework for statistical performance comparison of evolutionary computation algorithms', *Information Sciences*, **178**, 2870-2879.

[25] Shlesinger, M. F.,2006. 'Search research', *Nature*, **443**, 281-282.

[26] Yang, X. S., 2008. *Nature-Inspired Metaheuristic Algorithms*, Luniver Press, (2008).

[27] Yang, X. S., 2005. 'Biology-derived algorithms in engineering optimizaton' (chapter 32), in *Handbook of Bioinspired Algorithms and Applications* (eds Olarius & Zomaya), Chapman & Hall / CRC.

[28] Yang, X. S. and Deb, S., 2009. 'Cuckoo search via Lévy flights', *Proceeings of World Congress on Nature & Biologically Inspired Computing* (NaBIC 2009, India), IEEE Publications, USA, pp. 210-214.

[29] Yang, X. S., 2009. 'Harmony search as a metaheuristic algorithm', in: *Music-Inspired Harmony Search: Theory and Applications* (Eds Z. W. Geem), Springer, pp.1-14.

[30] Yang, X. S., 2010. *Engineering Optimisation: An Introduction with Metaheuristic Applications*, John Wiley and Sons.

Appendix: Demo Implementation

```
% ---------------------------------------------------------------
% Cuckoo Search (CS) algorithm by Xin-She Yang and Suash Deb      %
% Programmed by Xin-She Yang at Cambridge University              %
% Programming dates: Nov 2008 to June 2009                        %
% Last revised: Dec  2009   (simplified version for demo only)    %
% ---------------------------------------------------------------
% Papers -- Citation Details:
% 1) X.-S. Yang, S. Deb, Cuckoo search via Levy flights,
% in: Proc. of World Congress on Nature & Biologically Inspired
% Computing (NaBIC 2009), December 2009, India,
% IEEE Publications, USA,  pp. 210-214 (2009).
% http://arxiv.org/PS_cache/arxiv/pdf/1003/1003.1594v1.pdf
% 2) X.-S. Yang, S. Deb, Engineering optimization by cuckoo search,
% Int. J. Mathematical Modelling and Numerical Optimisation,
% Vol. 1, No. 4, 330-343 (2010).
% http://arxiv.org/PS_cache/arxiv/pdf/1005/1005.2908v2.pdf
% ---------------------------------------------------------------%
% This demo program only implements a standard version of         %
% Cuckoo Search (CS), as the Levy flights and generation of       %
% new solutions may use slightly different methods.               %
% The pseudo code was given sequentially (select a cuckoo etc),   %
% but the implementation here uses Matlab's vector capability,    %
% which results in neater/better codes and shorter running time.  %
% This implementation is different and more efficient than the    %
% the demo code provided in the book by                           %
%    "Yang X. S., Nature-Inspired Metaheuristic Algoirthms,       %
%     2nd Edition, Luniver Press, (2010).                 "       %
```

```
% ------------------------------------------------------------- %

% ============================================================= %
% Notes:                                                        %
% Different implementations may lead to slightly different      %
% behavour and/or results, but there is nothing wrong with it,  %
% as this is the nature of random walks and all metaheuristics. %
% ------------------------------------------------------------------

function [bestnest,fmin]=cuckoo_search(n)
if nargin<1,
% Number of nests (or different solutions)
n=25;
end

% Discovery rate of alien eggs/solutions
pa=0.25;

%% Change this if you want to get better results
% Tolerance
Tol=1.0e-5;
%% Simple bounds of the search domain
% Lower bounds
nd=15;
Lb=-5*ones(1,nd);
% Upper bounds
Ub=5*ones(1,nd);

% Random initial solutions
for i=1:n,
nest(i,:)=Lb+(Ub-Lb).*rand(size(Lb));
end

% Get the current best
fitness=10^10*ones(n,1);
[fmin,bestnest,nest,fitness]=get_best_nest(nest,nest,fitness);

N_iter=0;
%% Starting iterations
while (fmin>Tol),

    % Generate new solutions (but keep the current best)
     new_nest=get_cuckoos(nest,bestnest,Lb,Ub);
     [fnew,best,nest,fitness]=get_best_nest(nest,new_nest,fitness);
    % Update the counter
      N_iter=N_iter+n;
    % Discovery and randomization
      new_nest=empty_nests(nest,Lb,Ub,pa) ;

    % Evaluate this set of solutions
      [fnew,best,nest,fitness]=get_best_nest(nest,new_nest,fitness);
    % Update the counter again
      N_iter=N_iter+n;
    % Find the best objective so far
```

```
    if fnew<fmin,
        fmin=fnew;
        bestnest=best;
    end
end %% End of iterations

%% Post-optimization processing
%% Display all the nests
disp(strcat('Total number of iterations=',num2str(N_iter)));
fmin
bestnest

%% --------------- All subfunctions are list below ------------------
%% Get cuckoos by ramdom walk
function nest=get_cuckoos(nest,best,Lb,Ub)
% Levy flights
n=size(nest,1);
% Levy exponent and coefficient
% For details, see equation (2.21), Page 16 (chapter 2) of the book
% X. S. Yang, Nature-Inspired Metaheuristic Algorithms, 2nd Edition, Luniver Press, (2010).
beta=3/2;
sigma=(gamma(1+beta)*sin(pi*beta/2)/(gamma((1+beta)/2)*beta*2^((beta-1)/2)))^(1/beta);

for j=1:n,
    s=nest(j,:);
    % This is a simple way of implementing Levy flights
    % For standard random walks, use step=1;
    %% Levy flights by Mantegna's algorithm
    u=randn(size(s))*sigma;
    v=randn(size(s));
    step=u./abs(v).^(1/beta);

    % In the next equation, the difference factor (s-best) means that
    % when the solution is the best solution, it remains unchanged.
    stepsize=0.01*step.*(s-best);
    % Here the factor 0.01 comes from the fact that L/100 should the typical
    % step size of walks/flights where L is the typical lenghtscale;
    % otherwise, Levy flights may become too aggresive/efficient,
    % which makes new solutions (even) jump out side of the design domain
    % (and thus wasting evaluations).
    % Now the actual random walks or flights
    s=s+stepsize.*randn(size(s));
   % Apply simple bounds/limits
   nest(j,:)=simplebounds(s,Lb,Ub);
end

%% Find the current best nest
function [fmin,best,nest,fitness]=get_best_nest(nest,newnest,fitness)
% Evaluating all new solutions
for j=1:size(nest,1),
    fnew=fobj(newnest(j,:));
    if fnew<=fitness(j),
       fitness(j)=fnew;
       nest(j,:)=newnest(j,:);
```

```
    end
end
% Find the current best
[fmin,K]=min(fitness) ;
best=nest(K,:);

%% Replace some nests by constructing new solutions/nests
function new_nest=empty_nests(nest,Lb,Ub,pa)
% A fraction of worse nests are discovered with a probability pa
n=size(nest,1);
% Discovered or not -- a status vector
K=rand(size(nest))>pa;

% In the real world, if a cuckoo's egg is very similar to a host's eggs, then
% this cuckoo's egg is less likely to be discovered, thus the fitness should
% be related to the difference in solutions.  Therefore, it is a good idea
% to do a random walk in a biased way with some random step sizes.
%% New solution by biased/selective random walks
stepsize=rand*(nest(randperm(n),:)-nest(randperm(n),:));
new_nest=nest+stepsize.*K;

% Application of simple constraints
function s=simplebounds(s,Lb,Ub)
  % Apply the lower bound
  ns_tmp=s;
  I=ns_tmp<Lb;
  ns_tmp(I)=Lb(I);

  % Apply the upper bounds
  J=ns_tmp>Ub;
  ns_tmp(J)=Ub(J);
  % Update this new move
  s=ns_tmp;

%% You can replace the following by your own functions
% A d-dimensional objective function
function z=fobj(u)
%% d-dimensional sphere function sum_j=1^d (u_j-1)^2.
%  with a minimum at (1,1, ...., 1);
z=sum((u-1).^2);
```

ประวัติการศึกษา และการทำงาน

ชื่อ	นางสาวฐิติพร พงศ์กิดาการ
เกิดวันที่	22 พฤศจิกายน 2530
สถานที่เกิด	จังหวัดกรุงเทพมหานคร
ประวัติการศึกษา	วศ.บ. (วิศวกรรมไฟฟ้า) มหาวิทยาลัยเกษตรศาสตร์
ตำแหน่งปัจจุบัน	-
สถานที่ทำงานปัจจุบัน	-
ผลงานดีเด่นและ/รางวัลทางวิชาการ	-
ทุนการศึกษาที่ได้รับ	-