การวางแผนเส้นทางบินที่ดีที่สุดสำหรับอากาศยานไร้นักบิน

OPTIMUM PATH PLANNING FOR UNMANNED AERIAL VEHICLES

เกียรติศักดิ์ ควขุนทด
KEATTISAK QUEKHUNTHOD

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์
บัณฑิตวิทยาลัย
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ. 2550

สำนักหอสมุดกลาง พระจอมเกล้าลาดกระบัง

การวางแผนเส้นทางบินที่ดีที่สุดสำหรับอากาศยานไร้นักบิน

OPTIMUM PATH PLANNING FOR UNMANNED AERIAL VEHICLES

เกียรติศักดิ์ คิวขุนทด

KEATTISAK QUEKHUNTHOD

เลขหมู่.....
เลขทะเบียน..... 74851
วัน,เดือน,ปี..... 11 ต.ค. 2550

.b.....
.i.....

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ.2550

OPTIMUM PATH PLANNING FOR UNMANNED AERIAL VEHICLES

KEATTISAK QUEKHUNTHOD

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

MASTER OF ENGINEERING IN COMPUTER ENGINEERING

SCHOOL OF GRADUATE STUDIES

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2007

COPYRIGHT 2007

SCHOOL OF GRADUATE STUDIES

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองวิทยานิพนธ์

หัวข้อวิทยานิพนธ์ การวางแผนเส้นทางบินที่ดีที่สุดสำหรับอากาศยานไร้นักบิน

Optimum Path Planning for Unmanned Aerial Vehicles

นักศึกษา นายเกียรติศักดิ์ คิวขุนทด

รหัสประจำตัว 46061019

ปริญญา วิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชา วิศวกรรมคอมพิวเตอร์

อาจารย์ที่ปรึกษาวิทยานิพนธ์ ผศ.ดร.วิศิษฏ์ หิรัญกิตติ

คณะกรรมการสอบวิทยานิพนธ์		ลายมือชื่อ
รศ.ดร.เอื้อน	ปิ่นเงิน	
รศ.ดร.บุญธีร์	เครือตราชู	
ผศ.ดร.เกรียงไกร	ปอแก้ว	
รศ.สมศักดิ์	มิตะถา	
ผศ.ดร.วิศิษฏ์	หิรัญกิตติ	

วัน / เดือน/ ปี ที่สอบ 29 พฤษภาคม 2550 เวลา 14.00-16.00 น.

สถานที่สอบ ณ อาคาร 12 ชั้น ชั้น 4 (ห้อง E12-402)

บัณฑิตวิทยาลัยรับรองแล้ว

(รศ.ดร.จารุวัตร เจริญสุข)

คณบดีบัณฑิตวิทยาลัย

วันที่.....14.....**เดือน**...สิงหาคม.....**พ.ศ.**.2550....

หัวข้อวิทยานิพนธ์	การวางแผนเส้นทางบินที่ดีที่สุดสำหรับอากาศยานไร้นักบิน
นักศึกษา	เกียรติศักดิ์ คิวขุนทด
รหัสนักศึกษา	46061019
ปริญญา	วิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
พ.ศ.	2550
อาจารย์ที่ปรึกษาวิทยานิพนธ์	ผศ. ดร. วิศิษฏ์ หิรัญกิตติ

บทคัดย่อ

วิทยานิพนธ์ฉบับนี้นำเสนอวิธีใหม่ในการวางแผนเส้นทางบินที่ดีที่สุดโดยสามารถวางแผนเส้นทางโดยเลี่ยงสิ่งกีดขวางสำหรับอากาศยานไร้นักบิน (UAV: Unmanned Air Vehicle) โดยการประยุกต์ใช้วิธีการค้นหาเส้นทางแบบ A* ในกรณีที่จุดหมายมีจำนวนมากมายจนถึงไม่จำกัด ซึ่งวิธี A* เดิมมีข้อดีคือสามารถใช้ค้นหาเส้นทางที่ดีที่สุด แต่มีข้อเสียคือไม่สามารถใช้งานได้เมื่อนำไปใช้ในกรณีที่จุดหมายที่เป็นไปได้ทั้งหมดมีจำนวนเป็นอนันต์ งานวิจัยในวิทยานิพนธ์ได้แบ่งออกเป็น 2 ส่วน ส่วนแรกเป็นการแก้ปัญหาการวางแผนเส้นทางบินเพื่อเลี่ยงสิ่งกีดขวางที่มีจำนวนจุดหมายเพียงหนึ่งเดียว โดยเราได้ดัดแปลง A* ให้เหมาะสมกับการวางแผนเส้นทางบินของ UAV ที่มีลักษณะ Real time ด้วยการลดจำนวนโนดในกราฟที่ A* ต้องค้นหาลง โดยอาศัยข้อมูลต่อไปนี้ประกอบในการพิจารณา คือ ข้อสมมุติฐานที่ว่าเราสามารถทราบตำแหน่งที่แน่ชัดของสิ่งกีดขวางก่อนการวางแผน, ประการที่สองเป็นการวิเคราะห์ให้ทราบถึงเส้นทางที่สามารถเลี่ยงสิ่งกีดขวางได้เร็วที่สุด และสุดท้ายเป็นการอาศัยข้อมูลลักษณะการบินของ UAV

ส่วนที่ 2 เป็นการแก้ปัญหาการวางแผนเส้นทางบินเพื่อเลี่ยงสิ่งกีดขวางโดยพิจารณากรณีที่มีจำนวนจุดหมายจำนวนมากจนนับไม่ถ้วน การนำเอาวิธีที่เสนอในส่วนที่หนึ่งมาประยุกต์ใช้ค้นหาเส้นทางที่ดีที่สุดไปยังจุดหมายใดๆจากจุดหมายทั้งหมด จะใช้เวลาเพิ่มมากขึ้นตามจำนวนจุดหมายที่มีเนื่องจากในการเลือกโนดแต่ละครั้งในกราฟของเส้นทางที่เป็นไปได้ เพื่อนำมาสร้างเป็นเส้นทางที่ดีที่สุดจำเป็นต้องมีการประเมินค่าด้วยฮิวริสติกฟังก์ชั่น (Heuristic function) สำหรับทุกๆ จุดหมายที่มีในกราฟ ดังนั้นยิ่งมีจำนวนจุดหมายมากเท่าไร ก็จะเสียเวลาเพิ่มขึ้นมากเท่านั้น ดังนั้นเพื่อแก้ปัญหานี้ เราได้เสนออัลกอริธึมใหม่ที่มีพื้นฐานมาจาก A* ซึ่งสามารถค้นหาคำตอบได้อย่างมีประสิทธิภาพสูงกว่า

Thesis Title	Optimum Path Planning for Unmanned Aerial Vehicles
Student	Keattisak Quekhunthod
Student ID.	46061019
Degree	Master of Engineering
Program	Computer Engineering
Year	2007
Thesis Advisor	Asst. Prof. Dr. Visit Hirankitti

ABSTRACT

In this thesis we propose a new algorithm for optimum path planning among infinite number of goals to use by an unmanned aerial vehicle (UAV). This approach is adapted from the A* algorithm which has been used by many for finding a shortest path. However, A* may not be applicable directly to use for path planning by a UAV, since the number of possible paths for a UAV may be *infinite* whilst A* presumes a *finite* search space.

The result presented in this thesis is separated into two parts. In the first part, we investigate the path planning for a single goal, where we adapt A* for UAV path planning such that it can find an optimum path while avoiding obstacles in a real-time manner. According to our approach, during the searching process the number of candidate nodes, to be considered by A*, in the search space is reduced substantially by adopting the following information: (1) the assumption that the positions of the obstacles to be avoided by the UAV are certain and made known in advance of the planning (2) the results of analysis to identify shortest ways to avoid the obstacles (3) the knowledge of flying constraints for a certain type of UAVs.

In the second part, we have realized that to use the algorithm, proposed in the first part, to handle path planning with multiple goals or are infinite number of goals is too costly. This is because with multiple goals, for the algorithm to choose a node to construct an optimum path, a general A* based algorithm has to calculate the heuristic function for every goal. The more goals existing the more time will be needed to spend on these calculations. Therefore for path planning with a large number or even infinite number of goals, this algorithm will be impractical. So in the second part we propose the algorithm, Deepening A* algorithm, to overcome the problem.

กิตติกรรมประกาศ

คุณความดีอันใดที่บังเกิดจากวิทยานิพนธ์นี้ ขอมอบแต่บิดามารดาและพี่ๆ ผู้ที่เกื้อหนุน คอยห่วงใย เอาใจใส่ และเป็นกำลังใจ จนสามารถทำให้วิทยานิพนธ์เล่นนี้ให้เสร็จลุล่วงไปด้วยดี ผู้วิจัยรู้สึกซาบซึ้งในความกรุณาและขอกราบขอบพระคุณเป็นอย่างสูง

วิทยานิพนธ์นี้สามรถประสบความสำเร็จลุล่วงได้เป็นอย่างดีก็ด้วยความกรุณาจาก ผศ. ดร. วิศิษฏ์ หิรัญกิตติ ซึ่งเป็นอาจารย์ที่ปรึกษาวิทยานิพนธ์ที่ได้ให้ความเอาใจใส่ แนะนำความรู้และทฤษฎีต่างๆที่เป็นประโยชน์ ชี้แนะแนวทางในการแก้ปัญหา ให้คำปรึกษา และให้ความช่วยเหลือเสมอมา ผู้วิจัยรู้สึกซาบซึ้งในความอนุเคราะห์และขอกราบพระคุณเป็นอย่างสูง

ขอขอบคุณ สำนักงานกองทุนสนับสนุนการวิจัย (สกว.)และสำนักงานวิจัยและพัฒนาการทหารกลาโหม (สวพ.กห.) ที่ให้การสนับสนุนทุนเพื่อการวิจัยที่เป็นที่มาของวิทยานิพนธ์ฉบับนี้ภายใต้โครงการวิจัยและพัฒนาอากาศยานไร้นักบิน (THAI PUKSIN)

ขอขอบคุณเพื่อนทุกคนที่ให้กำลังใจ คำแนะนำ ประสบการณ์ที่ดี และยังคอยให้กำลังใจอย่างใกล้ชิดเสมอมา โดยเฉพาะอย่างยิ่งพี่ไก่ สราญพงษ์ หนูยิ้มซ้าย พี่อุ๊ จตุรพิธ เกราะแก้วและน้องๆในห้องวิจัยการสื่อสารและคมนาคมชาญฉลาดที่ทำให้บรรยากาศในห้องวิจัยเป็นไปด้วยความสนุกสนานและเป็นกันเอง

คุณค่าและประโยชน์อันพึงมีจากวิทยานิพนธ์เล่มนี้ ผู้วิจัยขอมอบแด่ผู้มีพระคุณทุกท่าน

เกียรติศักดิ์ คิวขุนทด

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญรูป

สารบัญรูป (ต่อ)

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ในการเดินทางของอากาศยานโดยทั่วไปได้ถูกกำหนดไว้ว่าต้องมีนักบินคอยควบคุมให้สามารถเดินทางไปยังจุดหมายได้อย่างปลอดภัย และในกรณีฉุกเฉินนักบินจะสามารถควบคุมและเลือกลงจอดในจุดจอดหรือสนามบินที่ใกล้ที่สุดได้อย่างปลอดภัย ดังนั้นนักบินจึงเป็นทรัพยากรบุคคลที่มีค่ายิ่งต่อมาจึงมีการพัฒนาอากาศยานไร้นักบิน (UAV: Unmanned Air Vehicle) ขึ้นโดยอาศัยการควบคุมระยะไกลด้วยคอมพิวเตอร์เพื่อทดแทนการบังคับบนเครื่องด้วยนักบิน แต่การนำอากาศยานไร้นักบินใช้มาในงานทางทหารที่เป็นงานที่มีความเสี่ยงสูงและมีจุดหมายบางส่วนไม่สามารถกำหนดไว้ล่วงหน้าได้ เช่น สำรวจพื้นที่จุดหมายที่คาดว่าเป็นตำแหน่งปืนใหญ่ของฝ่ายตรงข้ามเพื่อใช้กำหนดจุดในการเข้าไปทำลายซึ่งมีความไม่แน่นอนเป็นต้น ดังนั้นการวางแผนเส้นทางบินจึงเป็นปัญหาที่สำคัญสำหรับความปลอดภัยในการเดินทางของอากาศยานไร้นักบิน

อากาศยานไร้นักบินเป็นอากาศยานประเภทหนึ่งที่ใช้คอมพิวเตอร์ช่วยควบคุมการทำงานเพียงบางส่วนหรือความคุมการทำงานทั้งหมด การความคุมการทำงานประกอบด้วย ระบบควบคุมบนอากาศยาน และระบบควบคุมภาคพื้นดิน โดยทั้งสองส่วนจะต้องติดต่อสื่อสารกันตลอดเวลา แต่ถ้าการสื่อสารขัดข้อง ระบบควบคุมบนอากาศยานจะต้องสามารถนำอากาศยานไร้นักบินเดินทางกลับมายังฐานหรือลงจอดฉุกเฉินได้อย่างปลอดภัย ระบบควบคุมภาคพื้นดินทำหน้าที่ส่งคำสั่งจากนักบินจากฐานไปยังอากาศยานและ รับข้อมูลและภาพกลับมาแสดงผลให้นักบินสามารถควบคุมอากาศยานได้โดยนักบินจะต้องมีความชำนาญในการควบคุมการทำงานของอากาศยานเนื่องจากคุณสมบัติของอากาศยานบางประเภทไม่สามารถทำได้ เช่น เลี้ยวมากเกินค่าองศาที่อากาศยานสามารถทำได้ เป็นต้น นักบินจะต้องรักษาระดับการเลี้ยวไม่ให้มากเกินที่อากาศยานทำได้ แต่ถ้าต้องการให้อากาศยานสามารถเดินทางโดยอัตโนมัติปราศจากการควบคุมจากนักบินจำเป็นอย่างยิ่งที่จะต้องนำเอาลักษณะการบินเคลื่อนที่ของอากาศยานมาประมวลผลต่างๆรวมถึงการวางแผนเส้นทางบินด้วย

การเดินทางบินของอากาศยานไร้นักบินเชิงพาณิชย์เป็นการเดินทางที่มีการกำหนดเส้นทางบินล่วงหน้าไว้แล้วซึ่งอาจจะมีการเปลี่ยนแปลงได้บ้างตามสถานการณ์ที่เกิดขึ้น เช่น สภาพอากาศที่แปรปรวน เป็นต้น การกำหนดเส้นทางบินต้องกำหนดไว้ล่วงหน้าเพื่อให้ง่ายต่อการจัดระเบียบการจราจรทางอากาศ แต่สำหรับงานทางทหาร การวางแผนการบินเพื่อเข้าไปในพื้นที่ซึ่งเราไม่สามารถระบุไว้ล่วงหน้าได้ว่า สถานที่ของสมรภูมิการรบและจุดหมายที่จะต้องเข้าสำรวจ เราจึงไม่สามารถวางแผนเส้นทางบินไว้ล่วงหน้าได้ดังนั้นจึงจำเป็นจะต้องมีการวางแผนการบินขึ้นท่ามกลาง

สิ่งกีดขวางต่างๆในสถานการณ์ปัจจุบันเช่น ภูเขาหรือปืนต่อสู้อากาศยาน เป็นต้น และต้องมีการวางแผนด้วยความรวดเร็วเพื่อทันต่อสถานการณ์ที่เกิดขึ้นด้วย

เมื่ออากาศยานไร้นักบินออกเดินทางบินจากจุดเริ่มต้นเพื่อไปยังจุดหมายที่กำหนดไว้ ในระหว่างนี้อาจมีเหตุการณ์ฉุกเฉินเกิดขึ้นซึ่งจำเป็นต้องลงจอดฉุกเฉินในทันทีทันใดบนสนามบินหรือสถานที่ที่สามารถลงจอดได้ที่ใกล้ที่สุด เราจำเป็นต้องวางแผนเส้นทางบินให้กับอากาศยานใหม่โดยมีเงื่อนไขเพิ่มเติมว่าจุดหมายหรือสถานีสำหรับลงจอดฉุกเฉินมีมากกว่าหนึ่งจุดหมาย การวางแผนนี้เป็นปัญหาการเลือกจุดหมายที่มีเส้นทางที่สั้นที่สุดจากจุดเริ่มต้นมายังจุดหมายเหล่านั้น

1.2 ความมุ่งหมายและวัตถุประสงค์ของการศึกษา

การเดินทางของอากาศยานไร้นักบินไปยังจุดหมายโดยมีสิ่งกีดขวางเป็นอุปสรรคในการเดินทางเป็นปัญหาที่มีผลต่อความปลอดภัยอย่างยิ่ง วิทยานิพนธ์นี้จึงได้มุ่งแก้ปัญหาการวางแผนเส้นทางบินของอากาศยานไร้นักบินให้สามารถเลี่ยงสิ่งกีดขวางและเป็นเส้นทางที่ได้ต้องนำมาใช้เดินทางได้จริงด้วยการนำลักษณะการบินที่ของอากาศยานไร้นักบิน มาพิจารณาร่วมด้วย ลักษณะการบินของอากาศยานไร้นักบินที่นำมาใช้ในวิทยานิพนธ์ได้แก่ มุมเลี้ยวสูงสุด และระยะทางบินไกลสุดซึ่งขึ้นอยู่กับปริมาณน้ำมันที่อากาศยานบรรจุได้

นอกจากนี้ยังจะมุ่งเน้นในการแก้ปัญหาการวางแผนเส้นทางบินของอากาศยานไร้นักบิน ที่มีจุดหมายให้เลือกเดินทางมากกว่าหนึ่งจุดหมาย ในวิทยานิพนธ์นี้จึงได้ศึกษาการวางแผนเส้นทางด้วยวิธีการต่างๆ และเสนอวิธีการวางแผนเส้นทางบินแบบใหม่ในการค้นหาเส้นทางบินที่สั้นที่สุดสำหรับหลายจุดหมาย

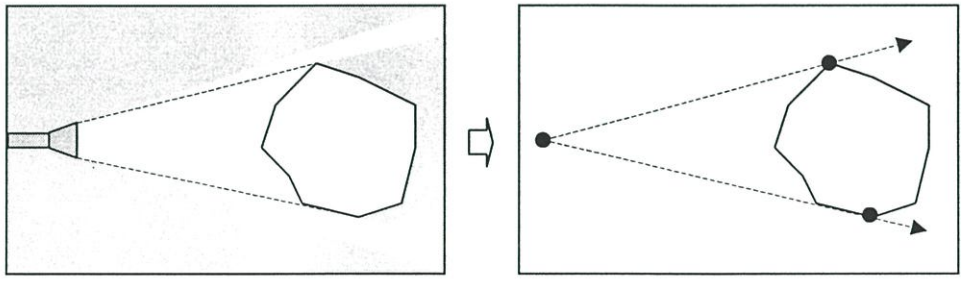
1.3 สมมติฐานของการศึกษา

การวางแผนเส้นทางบินสำหรับอากาศยานไร้นักบินแท้จริงแล้วเป็นการวางแผนเส้นทางการเดินทางใน 3 มิติซึ่งเป็นระนาบจำนวนจริงประกอบด้วยแกน X เป็นเส้นตรงตามแนวตะวันตก-ยังตะวันออก แกน Y เป็นเส้นตรงตามแนวเหนือ-ใต้ และแกน Z เป็นเส้นตรงที่ตั้งฉากจากพื้นโลก ซึ่งในแต่ละแกนของระนาบ 3 มิติมีค่าเป็นจำนวนจริงจึงทำให้เกิดมีเส้นทางการบินที่เป็นไปได้จำนวนอนันต์จึงเพิ่มความซับซ้อนในการวางแผนเส้นทางบิน ในวิทยานิพนธ์ฉบับนี้จึงไม่ขอนำความสูงในการบินมาพิจารณาด้วยจึงการวางแผนเส้นทางบินบนระนาบ 2 มิติ อย่างไรก็ตามถึงแม้ว่าจะมีเพียง 2 มิติ เส้นทางการบินที่เป็นไปได้ก็ยังคงมีจำนวนเป็นอนันต์ นอกจากนี้ยังได้กำหนดสิ่งกีดขวางที่ใช้ในวิทยานิพนธ์นี้เป็นรูปปิดล้อมสิ่งกีดขวางจริงเพื่อความปลอดภัยในการเดินทางของอากาศยานไร้นักบินที่ไม่สามารถบินเข้าใกล้สิ่งกีดขวางได้

การวางแผนเส้นทางในวิทยานิพนธ์นี้มุ่งเน้นในการแก้ปัญหาของการวางแผนเส้นทางบิน ซึ่งจำต้องทำก่อนการเดินทางจริง การวางแผนเส้นทางบินแบบนี้ได้สมมุติให้สิ่งกีดขวางไม่มีการเคลื่อนที่และสภาพแวดล้อมจะไม่เปลี่ยนแปลงขนาด รูปร่างและตำแหน่งในขณะเดินทาง จึงไม่จำเป็นต้องวางแผนเส้นทางบินใหม่

1.4 ทฤษฎีหรือแนวความคิดที่ใช้ในการวิจัย

วิทยานิพนธ์นี้ได้เลือกใช้อัลกอริธึมการค้นหาเอสตาร์ (A* search algorithm) ซึ่งเป็นการค้นหาแบบฮิวริสติก (Heuristic search) แบบหนึ่งที่สามารถหาผลลัพธ์ที่ดีที่สุด แต่เนื่องจากไม่สามารถนำมาใช้งานกับปัญหาที่มีเส้นทางที่เป็นไปได้จำนวนอนันต์ ดังนั้นอัลกอริธึมที่ใช้ในวิทยานิพนธ์นี้จึงเป็นอัลกอริธึมเอสตาร์ที่ได้ดัดแปลงเพื่อให้สามารถใช้วางแผนเส้นทางบินของอากาศยานไร้นักบินในกรณีที่จุดหมายจำนวน 1 จุดหมายหรือจำนวนอนันต์ได้ หลักการเลี่ยงสิ่งกีดขวางที่ใช้ในวิทยานิพนธ์นี้เป็นการเลี่ยงสิ่งกีดขวางที่ใช้ระยะที่สั้นที่สุดซึ่งจะเหมือนกับการฉายแสงไฟไปยังวัตถุ ลำแสงที่เกิดขึ้นจะเดินทางเป็นเส้นตรงไปยังวัตถุและมีลำแสงบางส่วนที่ไม่ชนวัตถุดังรูปที่ 1.1 เส้นประ คือ ลำแสงที่สัมผัสกับขอบของสิ่งกีดขวางโดยไม่ชนสิ่งกีดขวางเลย จากวิธีนี้ เราสามารถวางแผนเส้นทางโดยเลี่ยงสิ่งกีดขวางและใช้บริเวณจุดยอดของสิ่งกีดขวาง(วัตถุ) ที่เส้นตรง (ลำแสง) สัมผัสเป็นจุดเริ่มต้นในการหาเส้นทางเลี่ยงสิ่งกีดขวางอันถัดต่อมา ทำเช่นนี้เรื่อยไปจนกระทั่งถึงจุดหมาย จ ากวิธีการเลี่ยงสิ่งกีดขวางโดยวิธีการนี้ ในวิทยานิพนธ์มีการนิยามกราฟในการวางแผนเส้นทางบินตามลักษณะรูปร่างของสิ่งกีดขวางโดยเป็นรูปปิดหลายเหลี่ยม (Polygon) ทำให้สามารถสร้างเส้นสัมผัสสิ่งกีดขวางได้ ซึ่งกราฟในรูปแบบนี้จะนำไปใช้แก้ปัญหาในการค้นหาเส้นทางบินที่สั้นที่สุดของอากาศยานไร้นักบินสำหรับหนึ่งจุดหมาย หลายจุดหมาย และจุดหมายจำนวนไม่จำกัดต่อไป



รูปที่ 1.1 การเลี่ยงสิ่งกีดขวางใช้หลักการฉายแสงไปยังวัตถุ

1.5 ขอบเขตการวิจัย

วิทยานิพนธ์เล่มนี้ได้เสนออัลกอริธึมใหม่ที่ดัดแปลงมาจากอัลกอริธึมเอสตาร์ให้สามารถวางแผนเส้นทางบินของอากาศยานไร้นักบินบนระนาบ 2 มิติโดยก่อนการวางแผนเส้นทางบินจะต้องทราบตำแหน่งที่แน่ชัดของสิ่งกีดขวางและจุดหมายทั้งหมด วิทยานิพนธ์นี้ได้แสดงให้เห็นถึงประสิทธิภาพของอัลกอริธึมที่นำเสนอด้วยผลทดลองและวิเคราะห์การทำงานโดยเปรียบเทียบกับวิธีการอื่นๆ

1.6 ขั้นตอนของการศึกษา

วิทยานิพนธ์เล่มนี้แบ่งเนื้อหาออกเป็น 7 บท คือ

บทที่ 1 กล่าวถึงความเป็นมาของงานวิจัย ความมุ่งหมายและวัตถุประสงค์ สมมติฐาน ทฤษฎีที่ใช้ ขอบเขตของการวิจัย และขั้นตอนการศึกษา

บทที่ 2 กล่าวถึงงานวิจัยที่เกี่ยวข้องกับวิทยานิพนธ์ คือ การวางแผนเส้นทางที่ใช้อัลกอริธึมค้นหาแบบเอสตาร์, การวางแผนเส้นทางบินสำหรับอากาศยานไร้นักบิน และอัลกอริธึมการวางแผนเส้นทางสำหรับจำนวนจุดหมายที่มากกว่าหนึ่ง

บทที่ 3 ทฤษฎีที่เกี่ยวข้องกับการวางแผนเส้นทางซึ่งประกอบด้วยทฤษฎีกราฟและทฤษฎีการค้นหาเส้นทาง

บทที่ 4 กล่าวถึงการวางแผนเส้นทางบินที่สามารถหลีกเลี่ยงสิ่งขวางของอากาศยานไร้นักบินเริ่มตั้งแต่การสร้างกราฟแบบใหม่ที่เหมาะสำหรับอากาศยาน การนำเอาวิธีเอสตาร์มาประยุกต์ใช้งานในการวางแผนเส้นทาง

บทที่ 5 เป็นส่วนที่สองของงานวิจัย คือการค้นหาจุดหมายที่ดีที่สุดท่ามกลางจำนวนจุดหมายไม่จำกัด โดยเริ่มจากปัญหาของการวิจัย วิธีการและอัลกอริธึมที่นำเสนอ จากนั้นเป็นการประยุกต์ใช้งานกับการวางแผนเส้นทางบินของอากาศยานไร้นักบิน

บทที่ 6 เป็นการวิเคราะห์ประสิทธิภาพและผลการทดลอง

ส่วนบทที่ 7 จะสรุปผลการวิจัยทั้งหมด

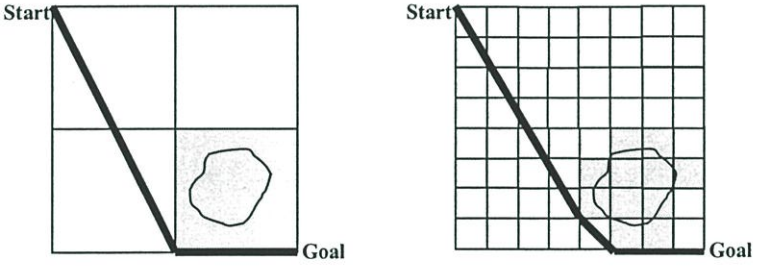
บทที่ 2
งานวิจัยที่เกี่ยวข้อง

บทนี้จะได้กล่าวถึงงานวิจัยที่เกี่ยวข้องกับการวางแผนเส้นทางบินของอากาศยานไร้นักบิน โดยเริ่มจากศึกษาถึงวิธีการสำหรับการกำหนดสเปซหรือกราฟสำหรับการวางแผนเส้นทาง จากนั้นเป็นงานวิจัยที่เกี่ยวข้องกับการวางแผนเส้นทางบินในกรณีที่มีจำหมายเดียว และสุดท้ายเป็นการวางแผนเส้นทางบินในกรณีที่มีจำนวนจุดหมายจำนวนมากมายเป็นอนันต์

2.1 งานวิจัยที่เกี่ยวกับการกำหนดสเปซการค้นหาในการวางแผนเส้นทางบิน

การกำหนดสเปซการค้นหาในการวางแผนเส้นทางบินนับเป็นหัวข้อหนึ่งที่มีงานวิจัยที่เกี่ยวข้องจำนวนไม่น้อย สเปซในการบินของอากาศยานเป็นระนาบ 3 มิติซึ่งในแต่ระนาบมีจุดจำนวนอนันต์ ทำให้เส้นทางที่เป็นไปได้ของเส้นทางบินมีจำนวนอนันต์เช่นกัน วิธีการหรืองานวิจัยที่มีการกำหนดสเปซการค้นหาในการวางแผนเส้นทางบินมีดังต่อไปนี้

การสร้างกริดเป็นวิธีการหนึ่งที่ช่วยลดจำนวนจำนวนจุดในการวางแผนเส้นทางบินลง ดังรูปที่ 2.1 คุณภาพของผลลัพธ์ในการวางแผนเส้นทางขึ้นอยู่กับความละเอียดของตารางกริด ถ้ากริดมีขนาดเล็กก็จะทำให้จำนวนโนดของสเปซในการค้นหามีมากขึ้นทำให้ต้องใช้เวลาในการค้นหามากขึ้นไปด้วยดังรูปที่ 2.1 ด้านซ้าย กับ 2.1ด้านขวา อย่างไรก็ดีถ้ามีจำนวนตารางมากก็จะส่งผลทำให้มีจำนวนโนดมากขึ้นตามไปด้วยซึ่งต้องใช้เวลาในการประมวลผลมากขึ้นนั่นเอง

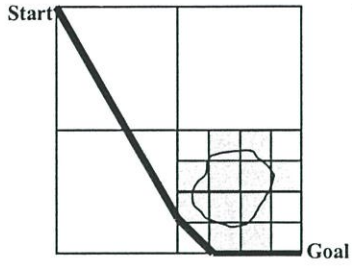


(a) 2x2 Grid (b) 8x8 Grid

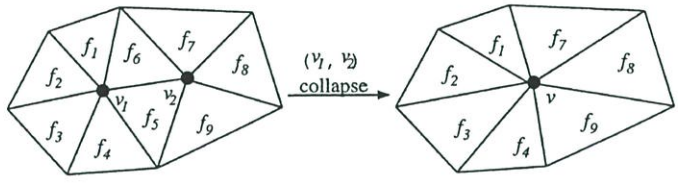
รูปที่ 2.1 การแบ่งกริดที่มีจำนวนต่างกัน

ในบางครั้งพบว่ามีโนดจำนวนหนึ่งในกริดที่ไม่ได้ใช้งานเลยจึงเป็นการสิ้นเปลือง ดังนั้นจึงมีการกำหนดสเปซในการค้นหาใหม่โดยใช้ Quad-tree แทน ได้แก่งานวิจัยของ Kambhampati [18] ซึ่งสร้างตารางเฉพาะส่วนที่จำเป็นเท่านั้น ลักษณะการสร้างโครงสร้างนี้จะสร้างตาราง 4 ช่องหรือขนาด 2 x 2 จากนั้นสร้างตารางย่อยจำนวน 4 ช่องลงในช่องใดช่องหนึ่งที่ต้องการของการสร้าง

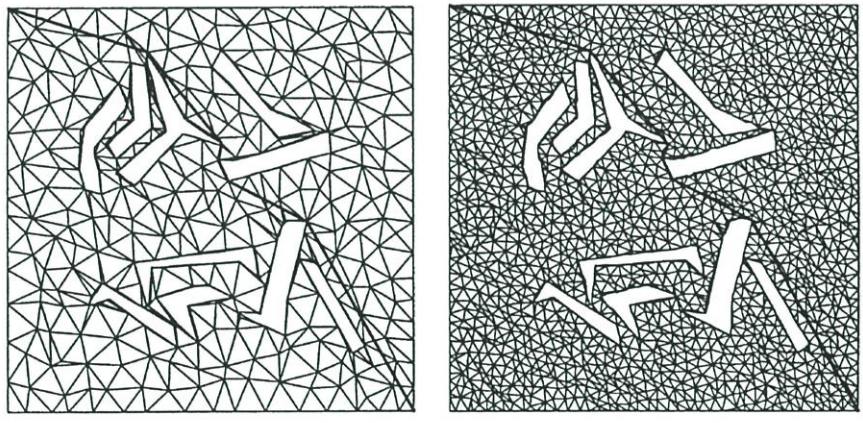
ตารางครั้งแรก และสามารถทำแบบนี้ได้ซ้ำอีกซึ่งขนาดของช่องหรือตารางจะเล็กลงไปอีก ดังรูปที่ 2.2 แต่ถ้ากำหนดให้เซลล์มีขนาดใหญ่ เส้นทางที่ได้อาจจะไม่ใช่เส้นทางที่สั้นที่สุด ดังนั้นเพื่อให้มีประสิทธิภาพสูงสุดจะต้องทำให้ช่องมีขนาดเล็กเพื่อบ่งบอกถึงขนาดที่ใกล้เคียงกับสิ่งกีดขวางมากที่สุด



รูปที่ 2.2 Quad tree



รูปที่ 2.3 เทคนิค Mesh Simplification

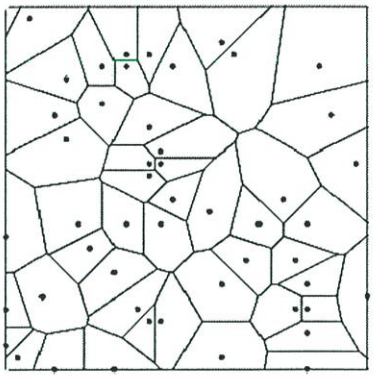


รูปที่ 2.4 การสร้างกราฟด้วยเทคนิค Mesh Simplification ด้วยขนาดของกริดที่แตกต่างกัน

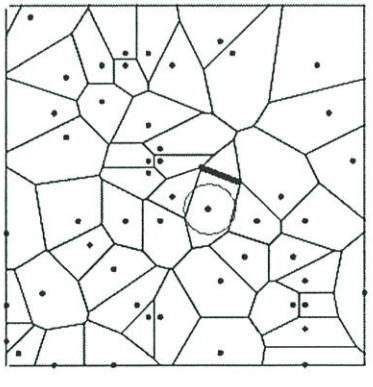
งานวิจัยของ Hwang และคณะ [7] ได้เสนอวิธีการวางแผนเส้นทางด้วยการปรับกราฟให้เหมาะสม (Path Graph Optimization) โดยเริ่มจากการสร้างกราฟจากตารางกริด จากนั้นนำเส้นเชื่อมจุดที่ตัดผ่านสิ่งกีดขวางออก สุดท้ายเป็นการลดรูปตารางด้วยเทคนิค Mesh Simplification ดังรูปที่ 2.3 ซึ่งเป็นเทคนิคการเพิ่มประสิทธิภาพในการแสดงภาพ 3 มิติโดยตัดจุดที่ไม่สำคัญออก เมื่อมี

จำนวนโนดลดลงก็จะทำให้มีจำนวนเส้นทางในการค้นหาน้อยลง จากรูปที่ 2.4 แสดงเส้นทางที่ได้ขึ้นอยู่กับตารางกริดในขั้นตอนแรกและแสดงให้เห็นข้อเสียของวิธีนี้คือมีโนดจำนวนหนึ่งถูกสร้างขึ้นเกินความจำเป็นในการใช้ค้นหาเส้นทาง

นอกจากการสร้างกริดแล้ว Bortoff [20] สามารถใช้วิธีการสร้างกราฟโดยใช้ไดอะแกรมโวโรนอย (Voronoi diagram) สร้างกรอบล้อมรอบจุดที่เป็นอันตราย (Treat point) และใช้เส้นตรงที่ใช้สร้างกรอบมาเชื่อมต่อกันเป็นเส้นทาง ดังรูปที่ 2.5 แต่ถ้าสิ่งที่เป็นอันตรายมีลักษณะเป็นพื้นที่หรือรัศมีที่เป็นอันตรายก็จะทำให้บางเส้นทางไม่สามารถใช้งานได้ เช่นในรูปที่ 2.6 สิ่งที่เป็นอันตรายมีลักษณะเป็นวงกลมซึ่งทำให้เส้นทางที่วงกลมตัดผ่านไม่สามารถใช้งานได้ นอกจากนี้แล้วพื้นที่บางส่วนสามารถตัดผ่านได้ตามเส้นทึบจึงเส้นทางบินที่เป็นไปได้



รูปที่ 2.5 กราฟที่ได้จากการสร้างโวโรนอยไดอะแกรม

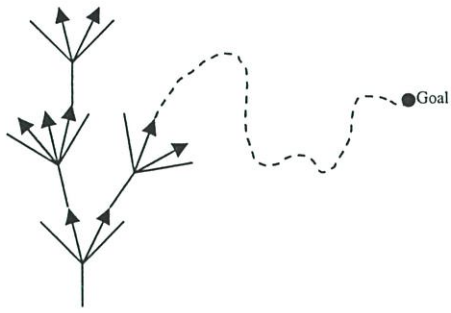


รูปที่ 2.6 สิ่งที่เป็นอันตรายเป็นวงกลมทำให้บางเส้นทางไม่สามารถใช้งานได้

จากที่กล่าวมาข้างต้นเป็นวิธีสร้างกราฟสเปซเส้นทางที่เป็นไปได้ก่อนการวางแผนเส้นทาง ต่อมาเป็นวิธีการสร้างเส้นทางในขณะวางแผนเส้นทางได้แก่ งานของ Szczerba [2] และคณะ [3] ซึ่งเสนอการสร้างเส้นทางในขณะการวางแผนเส้นทางโดยใช้ข้อจำกัดทางกายภาพและลักษณะการบินของอากาศยานซึ่งโดยปกติแล้วอากาศยานแต่ละประเภทมีข้อจำกัดที่สำคัญได้แก่

1) เส้นทางต้องมีระยะทางไม่เกินกว่าที่กำหนด ซึ่งถูกกำหนดโดยปริมาณที่สามารถบรรทุกได้อย่างจำกัด หรือเวลาที่ต้องใช้ในการเดินทางไปให้ถึงจุดหมาย
2) มุมเลี้ยวสูงสุดของอากาศยาน การเลี้ยวด้วยมุมที่มากเกินไปเพื่อเลี่ยงสิ่งกีดขวางอาจทำให้อากาศยานตกได้

ข้อจำกัดเหล่านี้เปลี่ยนแปลงขึ้นอยู่กับลักษณะของอากาศยานที่นำมาวางแผนเส้นทางบิน



รูปที่ 2.7 แสดงการสร้างโนดโดยใช้ลักษณะการบินของอากาศยาน

วิธีการและงานวิจัยที่ได้กล่าวมาทั้งหมดเป็นการแก้ปัญหาจำนวนโนดที่มีมากมายเป็นอนันต์ให้ลดลงจนมีจำนวนที่จำกัด จากนั้นจึงนำโนดเหล่านี้มาสร้างกราฟเพื่อนำใช้มาค้นหาเส้นทางต่อไป ซึ่งจะกล่าวถึงในหัวข้อต่อไป

2.2 งานวิจัยที่เกี่ยวข้องการวางแผนเส้นทางบินสำหรับอากาศยานไร้นักบิน

การแก้ปัญหาด้วยการค้นหาคำตอบในสเปซสำหรับค้นหาเป็นหลักการทางด้านปัญญาประดิษฐ์ที่มีความเป็นมาอันยาวนาน อัลกอริธึมที่ได้รับความนิยมอย่างแพร่หลายคือ อัลกอริธึมการค้นหาเอสตาร์ (A* search algorithm) ซึ่งสามารถใช้ค้นหาเส้นทางที่สั้นที่สุด มีงานวิจัยที่นำอัลกอริธึมนี้ไปใช้งานได้แก่

งานวิจัยของ Dobbs [1] ได้นำเอาการแก้ปัญหาด้วยวิธีเอสตาร์มาใช้ในการวางแผนเส้นทางบินสำหับอากาศยานไร้นักบินตั้งแต่ปี 1988 ซึ่งได้ศึกษาการทำงานของวิธีการเอสตาร์และวัดประสิทธิภาพและจากผลการทดลองของ Dobbs สามารถยืนยันได้ว่าวิธีเอสตาร์เป็นวิธีการที่มีประสิทธิภาพมาก อย่างไรก็ดีวิธีเอสตาร์มีข้อเสียในการใช้พื้นที่หน่วยความจำมากเนื่องมาจากต้องเก็บโนดที่พิจารณาแล้วทั้งหมดไว้ซึ่งไม่เหมาะสมที่จะนำมาใช้งานกับปัญหาที่สเปซมีโนดจำนวนมากๆ

งานวิจัยของ Szczerba [2] และคณะ [3] ได้นำวิธีการแก้ปัญหานี้ด้วยวิธีเอสตาร์มาปรับปรุงโดยใช้ข้อจำกัดทางกายภาพและลักษณะการทำงานของอากาศยาน ดังที่ได้กล่าวมาแล้วในหัวข้อที่

ผ่านมา การนำลักษณะการบินที่กำหนดมาประกอบการวางแผนเส้นทางด้วยวิธีเอสตาร์ Szczerba เรียกวิธีการนี้ว่า Sparse A* Search (SAS)

Yan และคณะ [5] ได้เสนอการวางแผนเส้นทางด้วยการนำข้อลักษณะการบินเคลื่อนที่ในระนาบ 3 มิติและนำกราฟนี้มาวางแผนเส้นทางด้วยวิธีการเอสตาร์ เช่นเดียวกับ Szczerba ซึ่งวิธีนี้สามารถหาผลลัพธ์ได้ไม่แตกต่างจากวิธีการของ Szczerba เท่าใดนัก

Hwang และคณะ [7] ได้นำสเปซที่สร้างจากตารางกริดและลดรูปด้วยเทคนิค Mesh Simplification มาใช้ค้นหาเส้นทางด้วยวิธีเอสตาร์

นอกจากวิธีการเอสตาร์สามารถวางแผนเส้นทางในสภาพแวดล้อมที่เป็นปกติแล้ว ยังมีงานวิจัยที่นำวิธีการเอสตาร์ไปแก้ไขเพื่อให้สามารถวางแผนเส้นทางในสภาพแวดล้อมที่ไม่แน่นอน เช่น งานวิจัยของ Stentz [4] ได้เสนอวิธีการวางเส้นทางบินที่สามารถทำงานได้ดีเมื่อมีการวางแผนเส้นทางใหม่ซึ่งเรียกว่า D* โดยวางแผนเส้นทางจากจุดหมายมายังจุดเริ่มต้น และนำข้อมูลในการวางแผนเส้นทางครั้งแรกหรือครั้งก่อนหน้ามาช่วยให้สามารถวางแผนเส้นทางใหม่เมื่อสภาพแวดล้อมเปลี่ยนแปลงได้รวดเร็วยิ่งขึ้น ซึ่งมีลักษณะการทำงานเหมือนเป็น Backward A* Search

นอกจากการใช้อัลกอริธึมการค้นหาแบบเอสตาร์มาวางแผนเส้นทางดังที่ได้กล่าวมาแล้วยังมีการใช้อัลกอริธึมแบบอื่นๆมาใช้ในการวางแผนเส้นทางเช่นเดียวกัน ได้แก่ งานวิจัยของ Rathbun และคณะ[6] ได้เสนอการวางแผนเส้นทางในสภาพแวดล้อมที่ไม่แน่นอนโดยใช้อัลกอริธึมแบบ Evolution โดยเริ่มจากการหาเส้นทางไปยังจุดหมายเป็นเส้นทางเริ่มต้น จากนั้นจะปรับปรุงเส้นทางนี้ให้ดีขึ้นเรื่อยๆ เหมือนกับวิวัฒนาการของสิ่งมีชีวิต การทำงานเช่นนี้สามารถนำไปใช้เมื่อสิ่งกีดขวางมีการเปลี่ยนแปลงได้เหมือนกับสภาพแวดล้อมในงานวิจัยของ Stentz [4] ซึ่งอาศัยอัลกอริธึมนี้การวางแผนเส้นทางซึ่งทำงานได้อย่างรวดเร็ว ใช้พื้นที่หน่วยความจำน้อย แต่อย่างไรก็ดีเส้นทางที่ได้ไม่สามารถยืนยันได้ว่าเป็นเส้นทางที่ดีที่สุดเนื่องจากอัลกอริธึมแบบนี้จะเก็บสถานะปัจจุบันเพื่อค้นหาคำตอบเท่านั้น

Hsiao และคณะ [12] ได้เสนออัลกอริธึมกองทัพมด (Ant Colony Optimization) สำหรับวางแผนเส้นทางซึ่งถือว่าจัดอยู่ในประเภทอัลกอริธึมฝูงแมลง (Swarm Optimization) แบบหนึ่ง หลักการทำงานของอัลกอริธึมนั้นเป็นการเลียนแบบการออกหาอาหารของกลุ่มมดซึ่งมดจะออกเดินทางไปหาอาหารจนกระทั่งพบอาหาร เมื่อพบอาหารแล้วจะเดินกลับรังของตัวเองตามเส้นทางเดิมและในระหว่างเดินทางกลับจะปล่อยกลิ่นหรือสารเคมี (Pheromone) เพื่อให้มดตัวอื่นรู้ว่ามีเส้นทางที่ไปถึงอาหารได้และใช้กลิ่นนี้ช่วยนำทางไปยังอาหารด้วย กลิ่นนี้จะจ่างลงไปเรื่อยๆ เส้นทางใดที่มีกลิ่นแรงแสดงว่าเป็นเส้นทางที่มดเดินผ่านจำนวนมากซึ่งน่าจะเป็นเส้นทางที่ดีที่สุด ตามหลักการหาอาหารของกองทัพมดสามารถนำมาใช้ค้นหาเส้นทางโดยสร้างเอเจนต์ (Agent) ทำหน้าที่เหมือนมด เอเจนต์แต่ละตัวทำการค้นหาเส้นทางอิสระจากกันโดยการเลือกว่าจะเดินเส้นทางไหนขึ้นอยู่กับกลิ่นที่มีอยู่ตามสภาพแวดล้อมที่เดินผ่านและเมื่อมีมดตัวหนึ่งสามารถไปถึงอาหาร เรา

ก็จะนำเส้นทางที่ได้มาเพิ่มค่ากลิ่นลงในสภาพแวดล้อมเพื่อให้มดตัวอื่นใช้ในการพิจารณา Hsiao ได้ใช้วิธีการนี้วางแผนเส้นทางได้อย่างมีประสิทธิภาพ แต่อย่างไรก็ตามยังไม่สามารถให้คำตอบที่ดีที่สุดได้เนื่องจากอัลกอริธึมนี้เป็นแบบ Local optimum

Chong และคณะ [13] ได้เสนออัลกอริธึมฝูงผึ้ง (Bee Colony Optimization) ซึ่งคล้ายกับงานของ Hsiao [12] แต่นำมาใช้ในการจัดการตารางการทำงานของเครื่องจักร พฤติกรรมการเคลื่อนที่ของผึ้งขึ้นอยู่กับข้อมูลของการเดินทางของผึ้งตัวก่อนหน้าที่สามารถไปยังจุดหมายหรือเกสรดอกไม้ด้วยหลักการนี้เราสามารถนำวิธีการนี้นำมาใช้งานวางแผนเส้นทางบนระนาบ 3 มิติได้ ซึ่งในวิทยานิพนธ์นี้จะไม่ขอกล่าวถึงการวางแผนเส้นทางบนระนาบ 3 มิติ เส้นทางที่ได้เป็นเส้นทางที่ฝูงผึ้งเดินทางบ่อยที่สุดนั่นเอง แต่เส้นทางเหล่านี้ไม่สามารถรับประกันได้ว่าเป็นเส้นทางที่สั้นที่สุด เช่นเดียวกับอัลกอริธึมกองทัพมด (Ant Colony Optimization)

2.3 งานวิจัยที่เกี่ยวข้องกับการวางแผนเส้นทางบินที่มีจุดหมายจำนวนมาก

การเดินทางของอากาศยานไร้นักบินบางครั้งอาจมีอุปสรรคที่ทำให้ไม่สามารถเดินทางไปถึงจุดหมายได้และต้องหาสถานที่ที่สามารถลงจอดฉุกเฉินได้ซึ่งสถานที่เหล่านี้อาจจะมีจำนวนมาก เราจึงได้ศึกษางานวิจัยที่เกี่ยวข้องกับการแก้ปัญหาเมื่อมีจุดหมายจำนวนมากดังนี้

งานวิจัยของ Dominik และคณะ [10] ใช้วิธีวางแผนเส้นทางแบบ 2 ทิศทาง (Bi-Directional) เพื่อหาจุดหมายแรกก่อนโดยการสุ่มแล้วนำมาหาเส้นทางแบบสองทิศทางจนกระทั่งได้คำตอบแรก จากนั้นเลือกจุดหมายอื่นมาวางแผนเส้นทางแบบสองทิศทางอีกครั้ง จากนั้นนำมาเปรียบเทียบกับคำตอบแรกเพื่อหาคำตอบที่ดีที่สุด และจะทำเช่นนี้กับจุดหมายที่เหลือจนหมด ซึ่งคำตอบสุดท้ายจะเป็นคำตอบที่ดีที่สุด ซึ่งวิธีการนี้ทำให้เวลาที่ใช้ในการวางแผนเส้นทางมากขึ้นตามจำนวนจุดหมายที่เพิ่มขึ้น

งานวิจัย Christian [11] เป็นงานวิจัยที่ใกล้เคียงกับปัญหานี้โดยได้เสนอวิธีการแก้ปัญหา MTP (Multi-goal path planning) ซึ่งเป็นปัญหาที่คล้ายกับ TSP (Traveling Salesman Problem) แต่เส้นทางของปัญหา MTP เป็นเส้นทางที่ผ่านทุกจุดหมายแต่ไม่กลับไปยังจุดเริ่มต้น และในการแก้ปัญหา MTP มีขั้นตอนหนึ่งที่เรียกว่า MTP control ซึ่งมีวิธีการเลือกจุดหมายด้วยอัลกอริธึมแบบต่างๆ เพื่อสร้างเส้นทางที่เหมาะสม ซึ่งพบว่าวิธีทั้งสองเป็นการนำการวางแผนเส้นทางไปประยุกต์ใช้กับเคลื่อนที่ของหุ่นยนต์ในโรงงานอุตสาหกรรม ซึ่งมีจำนวนจุดหมายจำกัด จึงไม่สามารถนำมาใช้แก้ปัญหาในกรณีที่มีจำนวนจุดหมายมากๆได้

Grimm [14] และ Gudaitis พร้อมคณะ [15] ได้เสนออัลกอริธึมเอสตาร์แบบขนานซึ่งสามารถวางแผนเส้นทางด้วยซีพียูหลายตัวพร้อมกันได้ อัลกอริธึมนี้วางแผนเส้นทางบนระนาบ 3 มิติซึ่งประสิทธิภาพการทำงานขึ้นอยู่สภาพแวดล้อมเช่นเดียวกับอัลกอริธึมเอสตาร์แบบมาตรฐาน

งานวิจัยของ Cvetanovic และคณะ [16] ได้เสนอวิธีการส่งผ่านข้อมูลระหว่างซีพียู (Message-passing architecture) สำหรับอัลกอริธึมเอสตาร์แบบขนาน และได้เปรียบเทียบการทำงานด้วยเทคนิคที่แตกต่างกัน คือ เทคนิคการใช้ข้อมูลร่วมกัน (Shared-List Algorithm) และเทคนิคการใช้ข้อมูลอิสระจากกัน (Static Distribution Algorithm) ผลการทำงานจะทำได้ดีกว่าเทคนิคที่นำมาเปรียบเทียบและมากยิ่งขึ้นเมื่อเพิ่มจำนวนซีพียู

นอกจากนี้ยังได้มีการสำรวจงานวิจัยที่เกี่ยวข้องกับการวางแผนเส้นทางแบบขนานของ Grama และคณะ [17] ซึ่งมีการสรุปประเด็นที่สำคัญของการวางแผนเส้นทางแบบขนานเมื่อใช้ซีพียูหลายตัว ได้แก่ ค่าใช้จ่ายที่เพิ่มขึ้น การสื่อสารข้อมูล สถาปัตยกรรมที่นำมาใช้ และความสามารถในการขยายการทำงาน ซึ่งทั้งหมดเป็นปัญหาที่นักวิจัยส่วนใหญ่กำลังให้ความสนใจอย่างมากเพื่อให้การวางแผนเส้นทางสามารถทำได้ภายในเวลาที่สั้นลง

แต่อย่างไรก็ดี การวางแผนเส้นทางบินของอากาศยานไร้นักบินไม่สามารถหาซีพียูจำนวนมากมาใช้งานได้ ดังนั้นในวิทยานิพนธ์นี้จึงได้ออกแบบอัลกอริธึมที่สามารถทำงานเมื่อมีซีพียูเดียวขณะที่มีจุดหมายมีจำนวนมาก ทั้งนี้ Russell [8] ได้เสนอวิธีการปรับปรุงฟังก์ชั่นฮิวริสติกของอัลกอริธึมเอสตาร์ใหม่เพื่อให้สามารถนำมาใช้กับกรณีที่มีจุดหมายจำนวนมากด้วยการเลือกฟังก์ชั่นฮิวริสติกที่ให้ค่าประเมินของแต่ละจุดหมายที่มีค่าน้อยที่สุดมาใช้ในฟังก์ชั่นประเมิน ซึ่งวิธีการนี้จะถูกนำมาใช้ในการเปรียบเทียบกับอัลกอริธึมที่นำเสนอในวิทยานิพนธ์นี้

บทที่ 3
ทฤษฎีที่เกี่ยวข้อง

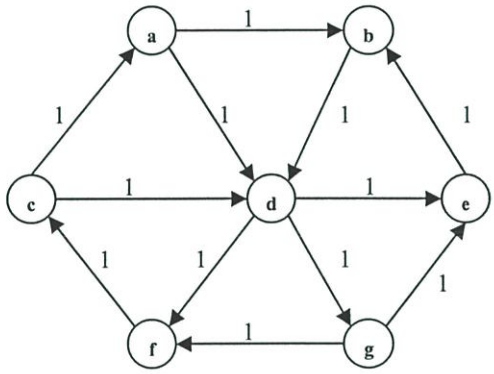
บทนี้กล่าวถึงทฤษฎีที่ใช้อธิบายการค้นหาเส้นทางซึ่งประกอบด้วยสเปซของการค้นหาและวิธีการการค้นหาเส้นทางแบบพื้นฐานต่างๆ

3.1 สเปซการค้นหา (Search Space)

สเปซเป็นพื้นที่หรือสภาพแวดล้อมของปัญหา เราสามารถแทนสเปซในรูปของกราฟได้ ซึ่งจากทฤษฎีกราฟได้กล่าวไว้ว่า

กราฟ G ประกอบด้วยเซตของโนด และเซตของเส้นขอบ (Edge) ที่เชื่อมระหว่างโนด ซึ่งเขียนแทนด้วยสัญลักษณ์ $G=(V,E)$ โดยที่ E เป็นเซตของเส้นขอบในกราฟ G ซึ่งเส้นขอบแต่ละเส้นเป็นคู่ของโนด (u,v) เมื่อ $u,v \in V$ และอาจมีค่าน้ำหนักประกอบด้วย

กราฟที่ให้ความสำคัญกับลำดับของคู่โนดของเส้นขอบซึ่งถ้าลำดับโนดต่างกันจะมองว่าเป็นคนละเส้นขอบ เราเรียกกราฟนี้ว่า กราฟแบบมีทิศทาง (Directed Graph)



รูปที่ 3.1 กราฟแบบมีทิศทางที่ค่าน้ำหนักเท่ากัน

เส้นทางภายในกราฟ G เป็นชุดลำดับของโนด $w_1, w_2, w_3, ..., w_k$ โดยที่ $(w_i, w_{i+1}) \in E$ ซึ่ง $1 \le i < k$ และ k เป็นจำนวนโนดของเส้นทางใดๆ ความยาวของเส้นทางจะเท่ากับจำนวนเส้นขอบบนเส้นทางซึ่งเท่ากับ $k-1$

เส้นทางวงรอบ (Cycle path) คือเส้นทางที่มีโนดเริ่มต้นและโนดสุดท้ายเป็นจุดเดียวกันหรือ $w_1 = w_k$ ถ้า k เป็นโนดสุดท้าย และต้องมีโนดอย่างน้อย 1 โนดอยู่ระหว่างโนดเริ่มต้นและโนดสุดท้าย ในกรณีที่เป็นกราฟแบบไม่มีทิศทางจะมีเส้นทางเป็น u,v,u ซึ่งนำมาพิจารณาเนื่องจากเส้นขอบ (u,v) และ (v,u) เป็นเส้นตรงเดียวกัน ส่วนกราฟแบบมีทิศทาง เส้นขอบ (u,v) และ (v,u) เป็น

เส้นขอบคนละเส้นกัน และเรียกกราฟแบบมีทิศทางที่ไม่มีเส้นทางวงรอบเรียกว่า Directed acyclic graph หรือ DAG

3.2 การค้นหาเส้นทาง

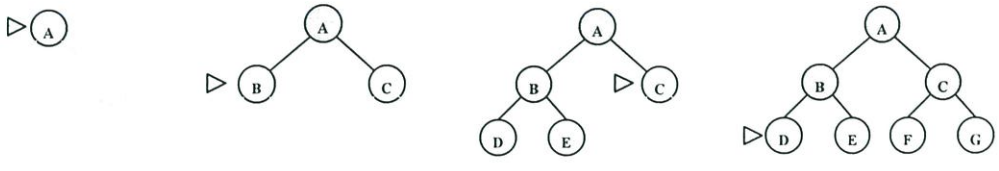
วิธีการค้นหาเส้นทางเป็นวิธีพื้นฐานที่สำคัญที่ใช้ในการวางแผนเส้นทางเหมาะสำหรับปัญหาทางด้านปัญญาประดิษฐ์ที่ต้องการคำตอบที่เป็นลำดับการทำงานหรือขั้นตอนจากจุดเริ่มต้นไปยังจุดหมาย ซึ่งมีหลักการทำงานของการค้นหาแบบต่างๆ ดังนี้

3.2.1 การค้นหาแบบขาดข้อมูลประกอบ (Uninformed Search)

หลักการทำงานของการค้นหาเริ่มจากการกำหนดสถานะต่างๆ เป็นโนดในกราฟ และให้ความสัมพันธ์ของสถานะหนึ่งที่เปลี่ยนไปอีกสถานะหนึ่งเป็นเส้นเชื่อมระหว่างโนด เมื่อกำหนดกราฟเป็นสเปซของปัญหาทั้งหมด การแก้ปัญหาจะเริ่มจากกำหนดสถานะเริ่มต้น และสถานะของจุดหมายที่ต้องการแก้ปัญหาเป็นโนดใดๆในกราฟ ซึ่งเรียกว่าโนดเริ่มต้นและโนดจุดหมายตามลำดับ จากนั้นนำโนดเริ่มต้นมาหาโนดใกล้เคียง (Neighbor Node) หรือโนดลูก (Child Node) ซึ่งเป็นสถานะหนึ่งที่เปลี่ยนมาจากสถานะเริ่มต้น เมื่อได้โนดลูกแล้ว เราจะเลือกเอาโนดลูกทั้งหมดมาหนึ่งโนดแล้วเพื่อหาโนดลูกของมันต่อไปเสมือนว่าโนดนี้เป็นโนดเริ่มต้นอีกครั้ง เราจะทำเช่นนี้เรื่อยไปจนกระทั่งเมื่อโนดที่เลือกมาเป็นโนดจุดหมาย การทำงานจึงสิ้นสุดลง เส้นทางจากโนดเริ่มต้นไปยังโนดจุดหมายเป็นคำตอบของการค้นหา

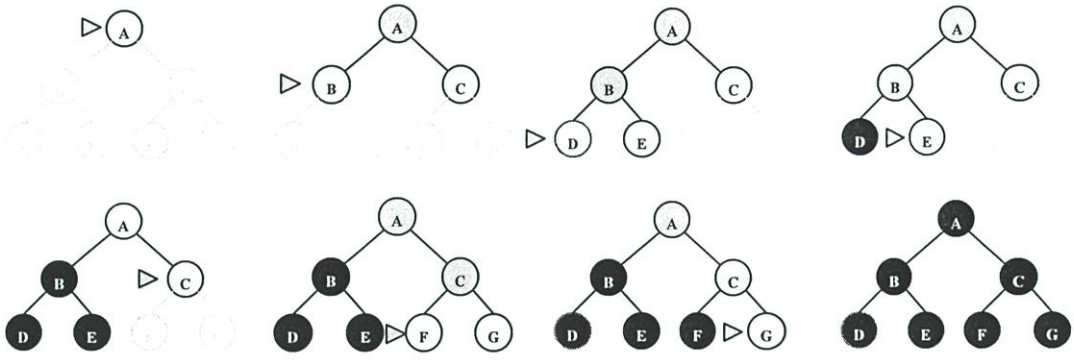
วิธีการเลือกโนดที่ได้มาจากการค้นหาโนดลูกด้วยการขาดข้อมูลประกอบการพิจารณาเรียกว่า Uninformed Search การทำงานทุกครั้งจะมีเพียง 2 ขั้นตอนคือถ้าไม่ใช่โนดจุดหมายจะทำการหาโนดลูกแต่ถ้าพบว่าเป็นโนดจุดหมายก็จะหยุดทำงานและถือว่าได้พบคำตอบแล้ว นอกจากการเลือกโนดแบบไม่มีข้อมูลประกอบแล้วยังมีวิธีที่สามารถทำงานได้ดีกว่า เรียกว่า การค้นหาแบบมีข้อมูลประกอบ (Informed Search) หรือ การค้นหาแบบฮิวริสติก (Heuristic Search) ซึ่งจะกล่าวถึงในหัวข้อถัดไป

การเลือกโนดแบบไม่มีข้อมูลประกอบมีวิธีการเลือกโนดได้หลายวิธีดังนี้



รูปที่ 3.2 การค้นหาแบบแนวกว้างบนไบนารี่ทรี

- Breadth-first Search เป็นการค้นหาตามแนวกว้างโดยการเลือกโนดเพื่อนำมาหาโนดลูก จะเลือกโนดที่มีความลึกเท่ากันจนครบทุกโนดเสียก่อนโดยเลือกโนดใดก่อนก็ได้ จากนั้นจึงจะเลือกโนดที่มีความลึกลงไปอีกขั้นหนึ่งเพื่อหาโนดลูกต่อไป ดังรูปที่ 3.2 จากรูปด้านซ้ายไปขวา กำหนดให้โนด A เป็นโนดเริ่มต้นแล้วหาโนดลูกของโนด A โดยในที่นี้เราเลือกโนดด้านซ้ายก่อน โนด A มีโนดลูกเป็นโนด B และ C จากนั้นเลือกโนด B มาหาโนดลูกและตามด้วยโนด C ทำจนกระทั่งไม่มีโนดที่มีความลึกระดับเดียวกันแล้วจึงเลือกโนดในความลึกลงอีกขั้นคือโนด D, E, F และ G ไปตามลำดับ ถ้าการทำงานของวิธีการค้นหานี้พบคำตอบที่ความลึก d และจำนวนโนดลูกของแต่ละโนดมีจำนวนเฉลี่ยอยู่ที่ b โนด (Branching factor) จะทำให้มีจำนวนโนดทั้งหมดที่ถูกสร้างขึ้นเป็น $O(b^{d+1})$ ซึ่งเมื่อต้องมีการค้นหาผลลัพธ์ที่ลงไปลึกเท่าไรก็จะใช้หน่วยความจำเป็นจำนวนทวีคูณ แต่อย่างไรก็ดียังมีวิธีการอื่นที่ใช้พื้นที่ในการค้นหาน้อยกว่าวิธีการค้นหานี้

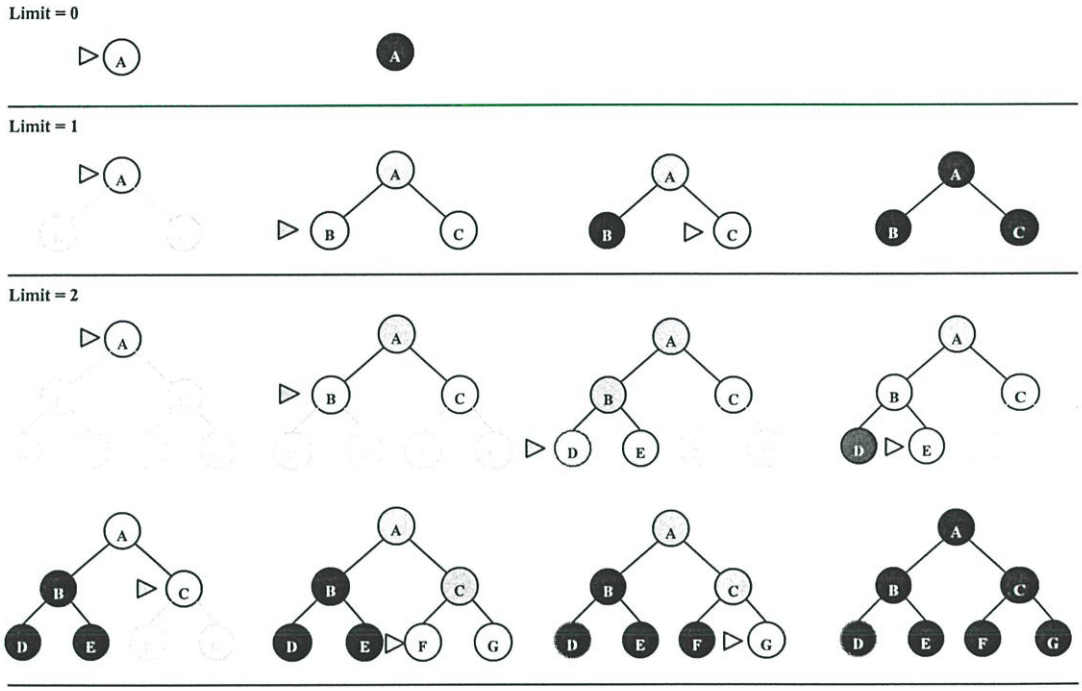


รูปที่ 3.3 การค้นหาตามแนวลึกบนไบนารี่ทรี

- Depth-first Search เป็นการค้นหาตามแนวลึกโดยจะเลือกโนดมาหาโนดลูกลงตามความลึกไปเรื่อยๆจนกระทั่งไม่สามารถหาโนดใกล้เคียงได้จึงย้อนกลับมาเลือกโนดอื่นๆต่อไป ดูจากรูปที่ 3.3 โนด A มีโนด B และ C เป็นโนดลูก จากนั้นเลือกโนด B มาหาโนดลูกได้เป็นโนด D และโนด E ซึ่งวิธีการนี้จะทำลึกลงไปก่อนซึ่งแตกต่างวิธีการค้นหาตามแนวกว้างโดยเลือกโนด D มาหาโนดลูกก่อนจะทำแบบนี้ไปจนกระทั่งโนดภายใต้โนด B ทั้งหมดหาโนดลูกหมดแล้วจึงจะย้อนกลับไปไปที่โนด C เพื่อหาโนดลูกต่อไป บางครั้งเรียกการค้นหาแบบนี้ว่า การค้นหาแบบย้อนกลับ (Backtracking Search) โนดบางส่วน(โนดที่เป็นสีดำ)จะไม่ถูกเก็บไว้ในหน่วยความจำด้วยการย้อนกลับทำให้ใช้พื้นที่หน่อยความจำเพียง $O(m)$ เมื่อ m เป็นความลึกสุดของทรี การทำงานของวิธีการนี้อาจทำให้เลือกเส้นทางผิดและเลือกโนดลึกมากหรือเจอลูบจนอาจจะไม่เจอที่สิ้นสุด ซึ่งอาจจะมีเส้นทางอื่นที่สามารถไปถึง

ผลลัพธ์ได้เร็วกว่าและลึกน้อยกว่า ดังนั้นวิธีการค้นหาตามแนวลึกไม่สามารถหาผลลัพธ์ที่ดีที่สุดได้เมื่อเกิดกรณีดังกล่าว

- Depth-limited Search เพื่อเป็นการแก้ปัญหาของการค้นหาตามแนวลึกที่มีทรีย่อยทางด้านซ้ายที่มีความลึกไม่จำกัด จึงกำหนดให้ความลึกในการค้นหาถูกจำกัดซึ่งเรียกวิธีการนี้ว่า การค้นหาตามแนวลึกแบบจำกัดความลึก และโดยการกำหนดความลึกให้กับการค้นหาทำให้ไม่สามารถขยายการค้นหาได้ลึกลงมากกว่า l เมื่อ l เป็นความลึกที่กำหนด และกำหนดให้ d เป็นความลึกที่ตื้นที่สุดของผลลัพธ์จากผลลัพธ์ที่เป็นไปได้ทั้งหมด ดังนั้นเมื่อ $l < d$ วิธีการค้นหานี้จะหยุดค้นหาก่อนที่จะถึงจุดหมายเสมอ



รูปที่ 3.4 การค้นหาตามแนวลึกแบบทำซ้ำที่ถูกจำกัดความลึกในการค้นหา

- Iterative Deepening Search หรือ Iterative Deepening Depth-First Search เป็นการค้นหาที่ได้ปรับปรุงมาจากการค้นหาตามแนวลึกแบบจำกัดความลึก เริ่มต้นการค้นหาโดยใช้การค้นหาตามแนวลึกแบบจำกัดความลึกที่มีความลึกเป็น 0 หรือทดสอบกับจุดเริ่มต้นโดยไม่มีการขยายโนดลูก แต่ถ้าจุดเริ่มต้นไม่ใช่จุดหมายจะเพิ่มความลึกจาก 0 เป็น 1 แล้วค้นหาด้วยวิธี Depth-limited Search ใหม่อีกครั้ง ถ้าไม่พบจุดหมายจะเพิ่มความลึกจาก 1 เป็น 2 และจะทำเช่นนี้ไปเรื่อยๆจนกระทั่งพบจุดหมาย ซึ่งจะพบผลลัพธ์ที่ความลึก d เราแสดงการทำงานได้ดังในรูปที่ 3.4 โดยเริ่มจากการค้นหาจากจุดเริ่มต้นที่มีความลึกเป็น 0 จากนั้นเพิ่มความลึกเป็น 1 จนครบทุกโนด และต่อมาเพิ่มความลึกเป็น 2 แล้วค้นหาใหม่ด้วยการค้นหาตามแนวลึกอีกครั้งจนครบทุกโนด ถ้ายังไม่พบโนดจุดหมาย ความลึกจะเพิ่ม

เป็น 3, 4, ... ไปจนกระทั่งพบโนดจุดหมาย วิธีการนี้ใช้เวลาในการทำงานมากเนื่องจากในการค้นหาแต่ละรอบการทำงานมีโนดลูกถูกสร้างใหม่ซ้ำๆกัน แต่มีข้อดีคือใช้หน่วยความจำน้อยเป็น $O(bd)$ เหมือนกับการค้นหาตามแนวลึก

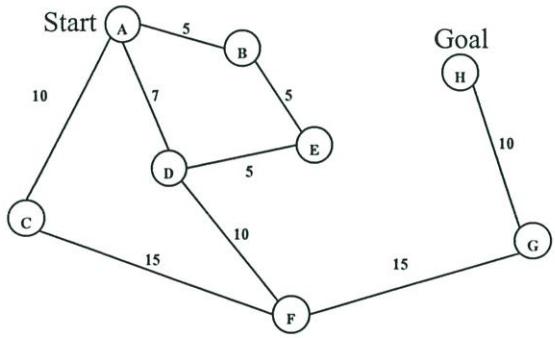
3.2.2 การค้นหาแบบฮิวริสติก (Heuristic Search)

การทำงานของการค้นหาแบบขาดข้อมูลประกอบส่วนใหญ่ไม่มีประสิทธิภาพจึงได้มีการนำข้อมูลมาประกอบการพิจารณาว่าโนดใดควรถูกเลือกมาค้นหาก่อน วิธีแบบง่ายๆที่นำข้อมูลอื่นมาร่วมพิจารณาด้วยคือการค้นหาแบบเลือกโนดที่ดีก่อน (Best-first Search) ซึ่งโนดที่อยู่ใกล้โนดจุดหมายจะถูกเลือกมาหาโนดลูกก่อนโนดอื่นๆ

การค้นหาแบบมีข้อมูลประกอบจะใช้ฟังก์ชั่นประเมินค่า (Evaluation function) สำหรับประเมินค่าสำหรับทุกๆโนดลูกที่จะค้นหาทั้งหมด โดยทั่วไปแล้วจะเลือกโนดที่ให้ค่าประเมินน้อยที่สุดมาหาโนดลูกต่อไปเนื่องจากค่าที่ประเมินเป็นค่าที่ได้จากระยะห่างจากโนดจุดหมาย ส่วนประกอบสำคัญของการค้นหาแบบนี้คือฟังก์ชั่นที่เรียกว่าฮิวริสติกฟังก์ชั่น (Heuristic function)

$h(n)$ = ค่าประมาณระยะห่างจากโนด n ถึงโนดจุดหมาย

โดยกำหนดให้ถ้า n เป็นโนดจุดหมาย จะได้ $h(n) = 0$



รูปที่ 3.5 กราฟตัวอย่าง

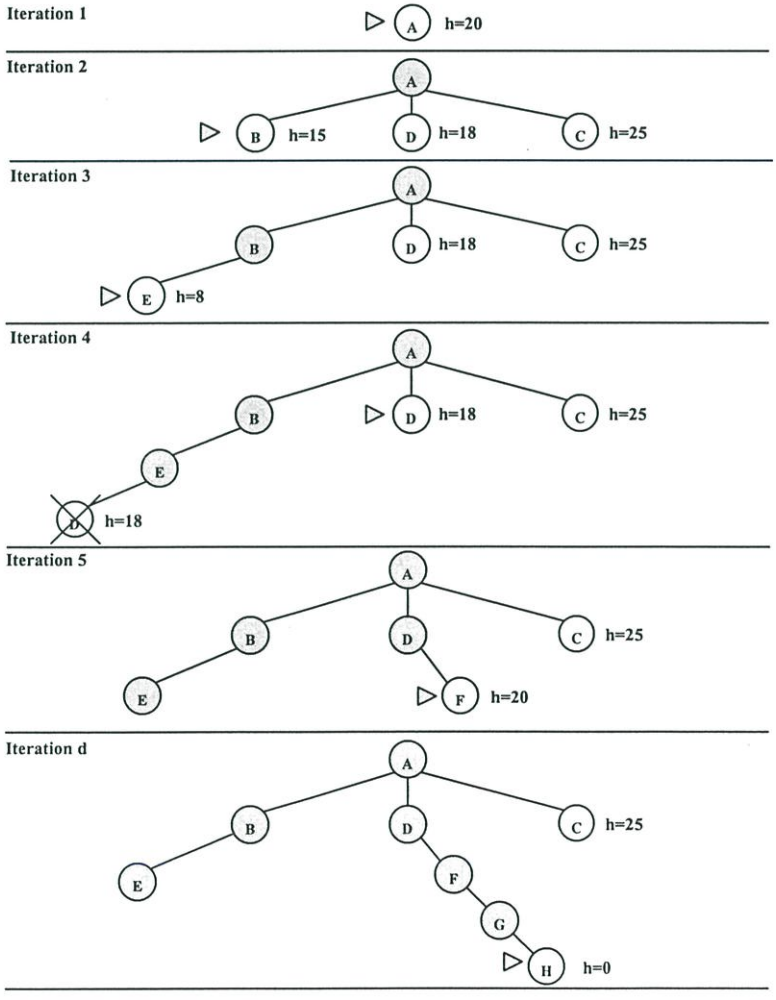
โดยทั่วไปแล้ว $h(n)$ จะเป็นค่าประมาณระยะทางของเส้นทางที่ลากจากโนด n ไปยังโนดจุดหมาย จากนี้จะแสดงการทำงานของวิธีการนี้โดยกำหนดกราฟตัวอย่างดังในรูปที่ 3.5 และกำหนดให้โนด A และโนด H เป็นโนดเริ่มต้นและโนดจุดหมายตามลำดับ และรูปที่ 3.5 สามารถแสดงการทำงานของการค้นหาแบบเลือกโนดที่ดีก่อน

เริ่มจากโนด A ที่มีค่า $h(n)$ สมมุติว่าเป็น 20 เนื่องจากมีโนดเดียวจึงเลือกโนด A มาหาโนดลูกต่อไป ในรอบที่ 2 จะได้โนดลูกของโนด A เป็นโนด B, C และ D ซึ่งมีค่า $h(n)$ สมมุติว่าเป็น 15, 25, 18 ตามลำดับซึ่งโนดที่มีค่าน้อยที่สุดคือโนด B

สำนักหอสมุดกลาง พระจอมเกล้าลาดกระบัง

รอบต่อมาเลือกโนด B มีโนด E เป็นโนดลูกมีค่า $h(n)$ สมมุติว่าเป็น 8 และโนด E ถูกเลือกเพราะมีค่าน้อยกว่าโนด C และ D รอบที่ 4 โนดลูกของโนด E คือโนด D แต่เคยถูกสร้างมาแล้วจึงตัดทิ้ง ดังนั้นจึงมีโนดให้เลือกระหว่างโนด C และโนด D ซึ่ง $h(n)$ ของ D มีค่าน้อยกว่าจึงถูกเลือกรอบที่ 5 และรอบต่อมา โนด C จะไม่ถูกเลือกเลยเนื่องจากอยู่ไกลสุด จนกระทั่งเมื่อถึงโนด H ซึ่งมีค่า $h(n)$ เป็น 0 แสดงว่าโนด H เป็นโนดจุดหมายจึงหยุดทำงานและได้เส้นทางจากโนดเริ่มต้นไปยังโนดจุดหมาย คือ A-> D->F->G->H

จากการทำงานจะเห็นว่าวิธีการนี้เหมือนการค้นหาตามแนวลึกซึ่งจะมุ่งไปโนดในทิศทางหนึ่งจนกระทั่งถึงทางตันก็จะเปลี่ยนทิศทาง จากตัวอย่างรอบที่ 3 การค้นหามุ่งไปยังโนด E ซึ่งเป็นเส้นทางที่ไกลกว่าไปในทิศทางของโนด D แต่เมื่อไม่สามารถค้นหาต่อได้จึงเปลี่ยนมายังทิศทางของโนด D ซึ่งมีผลเหมือนกับการค้นหาตามแนวลึกคือผลลัพธ์ที่ได้อาจไม่ใช่ผลลัพธ์ที่ดีที่สุดและอาจจะไม่พบผลลัพธ์ก็เป็นได้ ถ้าไม่ป้องกันการเกิดลูปจะทำให้การค้นหาไปติดในลูปแบบไม่รู้จบ



รูปที่ 3.6 การค้นหาแบบเลือกโนดที่ดีก่อน

74851

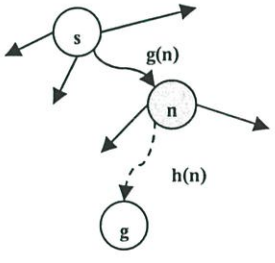
3.2.3 การค้นหาวิธีเอสตาร์ (A* Search)

การค้นหาแบบเอสตาร์ (A* Search) เป็นการค้นหาแบบใช้ข้อมูลประกอบการพิจารณาแบบหนึ่ง การค้นหาแบบเอสตาร์ได้ใช้ฟังก์ชั่นประเมินที่ต่างจากการค้นหาแบบเลือกโนดที่ดีก่อน ด้วยการเพิ่มข้อมูลค่าใช้จ่ายจริงในการค้นหาจากโนดเริ่มต้นมายังโนด n ซึ่งเขียนได้เป็น

$$f(n) = g(n) + h(n) \tag{3.1}$$

ซึ่งเป็นฟังก์ชั่นที่เกิดจากการรวมค่า $g(n)$ ซึ่งเป็นค่าระยะทางจริงจากจุดเริ่มต้นมาถึงโนด n และค่า $h(n)$ ซึ่งเป็นค่าประมาณระยะห่างจากโนด n ไปถึงจุดหมาย ดังรูปที่ 3.7 เส้นทึบเป็นค่าระยะทางจริง และเส้นประเป็นค่าระยะทางที่ได้จากการประมาณ

อัลกอริธึมในการค้นหาแบบเอสตาร์จะใช้ $f(n)$ ในการตัดสินใจเลือกโนดโดยเริ่มจากโนดเริ่มต้น s เพื่อขยายโนดลูกในแต่ละขั้นซึ่งทำให้เกิดเส้นทางที่ยังค้นหาไม่จบจนกว่าจะถึงโนดจุดหมาย เส้นทางเหล่านี้มีโนดปลายซึ่งถือว่าเป็นโนด n ของเส้นทางนั้นๆ ถ้าเส้นทางใดในบรรดาเส้นทางเหล่านี้ได้ค่าจาก $f(n)$ ของโนดมีค่าที่น้อยที่สุดในกลุ่มโนด n อื่นๆ อัลกอริธึมจะเลือกขยายขอบเขตการค้นหาต่อในที่เส้นทางนั้น การค้นหาแบบเอสตาร์จะทำทีละขั้นเช่นนี้ซ้ำแล้วซ้ำอีกไปเรื่อยๆจนกระทั่งพบโนดจุดหมาย g เส้นทางที่เกิดขึ้นที่เชื่อมจากโนด s ไปโนดจุดหมาย g จะเป็นเส้นทางที่ดีที่สุด



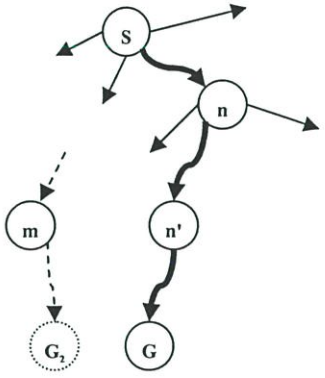
รูปที่ 3.7 ส่วนประกอบของฟังก์ชั่นฮิวริสติก

การค้นหาแบบเอสตาร์มีคุณสมบัติที่สามารถให้ผลลัพธ์ที่ดีที่สุด เมื่อเป็นการค้นหาเส้นทางบนทรี ค่าที่ได้จากฟังก์ชั่นฮิวริสติกต้องไม่เกินค่าใช้จ่ายจริงที่ได้จากการเดินทางจากโนด n ไปยังโนดจุดหมายหรือที่เรียกว่ามีคุณสมบัติ Admissible ดังนั้นเมื่อ $g(n)$ เป็นค่าใช้จ่ายจริงและ $h(n)$ เป็นค่าประมาณที่ไม่เกินค่าใช้จ่ายจริง จึงทำให้ $f(n)$ มีค่าไม่เกินค่าความเป็นจริงในการค้นหาจากโนด n ไปยังโนดจุดหมายซึ่งทำให้การค้นหาแบบเอสตาร์มีคุณสมบัติที่สามารถให้ผลลัพธ์ที่ดีที่สุดได้ ซึ่งเราสามารถพิสูจน์ได้ดังนี้

เริ่มจากการกำหนดให้ทั้ง G และ G_2 เป็นโนดจุดหมาย แต่เส้นทางที่ไปถึงโนดจุดหมาย G_2 ไม่ใช่เส้นทางที่สั้นที่สุด ขอเรียกโนด G_2 ว่า Suboptimal goal node และเส้นทางที่สั้นที่สุดไปถึงโนด G ซึ่งมีค่าใช้จ่ายจริงน้อยที่สุดเป็น C^* จากรูปที่ 3.8 เส้นทึบเป็นเส้นทางที่สั้นที่สุดมายัง G และเส้นประเป็นเส้นทางที่สั้นรองลงมาซึ่งมีค่ามากกว่าเส้นทางที่สั้นที่สุด

เมื่อ G_2 เป็นโนดจุดหมาย ทำให้ค่าประมาณมีค่าเป็น $h(G_2)=0$ หรือค่าประเมินของโนด G_2 จะเหลือเพียงค่าใช้จ่ายจริงของโนด G_2 นั่นเอง และมีค่ามากกว่าค่าระยะทางของเส้นทางที่สั้นที่สุดหรือ C^* เราจึงเขียนได้เป็น

$$f(G_2)=g(G_2)+h(G_2)=g(G_2)>C^* \tag{3.2}$$



รูปที่ 3.8 รูปประกอบการพิสูจน์คุณสมบัติ Optimality ของฟังก์ชั่นฮิวริสติก

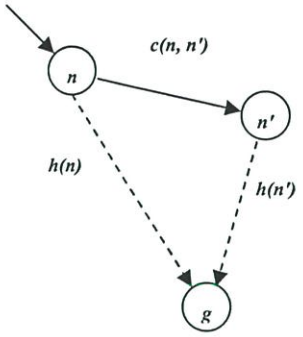
และเมื่อพิจารณาที่โนด n ใดๆที่อยู่บนเส้นทางที่สั้นที่สุดซึ่งอยู่ระหว่างโนดเริ่มต้นกับโนดจุดหมาย และ $h(n)$ เป็นค่าประมาณที่ไม่เกินค่าใช้จ่ายจริง ซึ่งทำให้ค่าประเมินของ n ไม่มากกว่า C^* จึงเขียนได้เป็น

$$f(n)=g(n)+h(n)\le C^* \tag{3.3}$$

จะเห็นได้ว่าโนด n ใดๆที่มีค่าประเมินน้อยกว่า C^* จะถูกเลือกมาพิจารณาในระหว่างการค้นหาก่อนที่จะพบจุดหมาย และเมื่อนำสมการ (3.2) และ (3.3) มาเขียนรวมกัน จึงสรุปได้เป็น $f(n)\le C^*<f(G_2)$ ซึ่งทำให้การค้นหาแบบเอสตาร์ไม่ได้เลือก G_2 มาพิจารณาก่อนเส้นทางที่สั้นที่สุด เนื่องจาก G_2 เป็น Suboptimal goal node

แต่เมื่อนำวิธีค้นหาแบบเอสตาร์มาใช้ค้นหาในกราฟ คุณสมบัติ Admissible ไม่สามารถใช้ได้ เนื่องจากการค้นหาบนกราฟอาจจะเกิดลูปหรือค่าประเมินมีค่าติดลบซึ่งจะส่งผลให้ไม่สามารถไปค้นหาในเส้นทางอื่นที่เป็นไปได้ จึงทำให้วิธีค้นหาแบบเอสตาร์ไม่มีคุณสมบัติ

Completeness ดังนั้นคุณสมบัติที่ช่วยให้สามารถป้องกันปัญหานี้ได้ ต้องอาศัยคุณสมบัติความคล้องจองกัน (Consistency) ที่เรียกว่า Monotonicity กล่าวคือทุกๆ โนด n และโนดลูก n' ค่าประมาณจากโนด n ถึงโนดจุดหมายต้องไม่มากกว่าผลรวมของค่าใช้จ่ายจากโนด n ไปยังโนดลูก n' รวมกับค่าประมาณจากโนด n' ถึงโนดจุดหมาย ซึ่งสามารถเขียนเป็นสมการได้ดังสมการ (3.4) แสดงได้ดังรูปที่ 3.9



รูปที่ 3.9 คุณสมบัติ Monotonicityของฟังก์ชั่นฮิวริสติก

$$h(n) \le c(n,n') + h(n') \tag{3.4}$$

ซึ่งจะพบว่าฟังก์ชั่นฮิวริสติกทุกๆฟังก์ชั่นที่มี Monotonicity จะมีคุณสมบัติ Admissible ด้วย และ*คุณสมบัติ Monotonicity จะมีความสมบูรณ์กว่าคุณสมบัติ Admissible* เนื่องจากค่าที่ได้จากฟังก์ชั่นฮิวริสติกของโนดที่เลือกมาพิจารณาในแต่ละครั้งจะเพิ่มขึ้นเรื่อยๆ แม้ว่าเส้นทางนั้นเป็นลูปก็ตาม จึงทำให้อัลกอริธึมเอสตาร์มีคุณสมบัติ Completeness เมื่อใช้ฟังก์ชั่นฮิวริสติกที่มีคุณสมบัติ Monotonicity

ถ้าการค้นหาแบบเอสตาร์ที่ฟังก์ชั่นฮิวริสติกมีคุณสมบัติเหล่านี้ เราจะกล่าวได้ว่า ทุกโนด n ที่ $f(n) < C^*$ จะถูกเลือกมาหาโนดลูกเสมอและบางโนดที่มีระยะทางเท่ากันหรือค่า $f(n) = C^*$ ด้วย ซึ่งทำให้โนดที่ $f(n) > C^*$ จะไม่ถูกนำมาพิจารณาเลย หรืออาจจะกล่าวได้ว่าโนดเหล่านี้ถูกตัดทิ้งจากการพิจารณา แต่ไม่ได้ตัดเส้นทางที่ดีที่สุดออกไป

อย่างไรก็ดี การค้นหาแบบเอสตาร์ใช้พื้นที่หน่วยความจำมากเนื่องจากจะเก็บทุกๆ โนดที่ยังค้นหาไม่แล้วเสร็จไว้ในหน่วยความจำ ด้วยเหตุนี้เองที่ทำให้*การค้นหาแบบเอสตาร์ไม่สามารถทำงานในกรณีที่มีโนดจำนวนมากๆ ได้* ในปัจจุบันการพัฒนาวิธีการค้นหาแบบใหม่ๆอื่นจึงได้พยายามการใช้หน่วยความจำลงเพื่อให้ประมวลผลได้เร็วขึ้นแต่จะส่งผลทำให้ไม่สามารถหาผลลัพธ์ที่ดีที่สุดได้หรือไม่สามารถหาผลลัพธ์ได้เลย

บทที่ 4
การวางแผนเส้นทางบินที่สามารถเลี่ยงสิ่งกีดขวางของอากาศยานไร้นักบิน

การวางแผนเส้นทางบินเป็นปัญหาที่สำคัญสำหรับความปลอดภัยในการเดินทางของอากาศยานไร้นักบิน ซึ่งมีงานวิจัยที่เกี่ยวข้องกับการวางแผนเส้นทางซึ่งนำไปใช้งานในเชิงพาณิชย์ และงานทางทหาร บทนี้จะกล่าวถึงการวางแผนเส้นทางบินที่สามารถเลี่ยงสิ่งกีดขวางโดยวิธีการค้นหาแบบเอสตาร์(สำหรับหนึ่งจุดหมาย)

4.1 แนวความคิด

การค้นหาเอสตาร์เป็นการค้นหาแบบหนึ่งที่สามารถหาผลลัพธ์ได้เป็นผลลัพธ์ที่ดีที่สุด แต่เนื่องจากไม่สามารถนำมาใช้งานกับปัญหาที่มีเส้นทางที่เป็นไปได้จำนวนอนันต์ เส้นทางบินจำนวนอนันต์เกิดจากลักษณะการเดินทางบินของอากาศยานซึ่งเป็นการเดินทางไปในระนาบ 3 มิติซึ่งเป็นระนาบจำนวนจริงประกอบด้วยแกน X เป็นเส้นตรงตามแนวตะวันตก-ตะวันออก แกน Y เป็นเส้นตรงตามแนวเหนือ-ใต้ และแกน Z เป็นเส้นตรงที่ตั้งฉากจากพื้นโลกซึ่งในแต่ละแกนของระนาบ 3 มิติมีค่าเป็นจำนวนจริงทำให้มีเส้นทางการบินที่เป็นไปได้จำนวนอนันต์ ดังนั้นในวิทยานิพนธ์นี้จึงลดความซับซ้อนในการวางแผนเส้นทางบินด้วยการไม่นำความสูงในการบินมาพิจารณาร่วมด้วยจึงเป็นการวางแผนเส้นทางบินบนระนาบ 2 มิติเท่านั่น ถึงอย่างไรก็ตามแม้ว่าจะพิจารณาเพียง 2 มิติเส้นทางการบินที่เป็นไปได้ก็ยังคงมีจำนวนเป็นอนันต์เช่นกัน

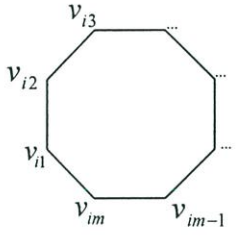
ในบทที่ 2 ที่ได้กล่าวไปแล้วว่าเราสามารถแทนสเปซของเส้นทางได้ด้วยการใช้กริด Quad-tree หรือ Mesh simplification เพื่อเป็นข้อแก้ไขข้อบกพร่องของทั้งสามวิธี เราจึงนิยามสเปซสำหรับการวางแผนเส้นทางบินของอากาศยานไร้นักบินขึ้นมาสำหรับในวิทยานิพนธ์นี้ ซึ่งจะได้กล่าวในหัวข้อต่อไปแล้ว

4.2 สเปซสำหรับการค้นหา (Search Space)

หลักการเคลื่อนที่เลี่ยงสิ่งกีดขวางที่ใช้ในวิทยานิพนธ์นี้เป็นการเลี่ยงสิ่งกีดขวางที่มีระยะทางที่สั้นที่สุดซึ่งเหมือนกับการฉายไฟส่องไปยังวัตถุ ลำแสงที่เกิดขึ้นจะเดินทางเป็นเส้นตรงไปยังวัตถุและมีลำแสงบางส่วนที่ไม่ชนวัตถุดังรูปที่ 1.1 ด้วยวิธีนี้ เราจึงได้นิยามกราฟแบบทั่วไปในบทที่ 3 ใหม่เป็นสเปซสำหรับการวางแผนเส้นทางบินโดยใช้การค้นหาแบบเอสตาร์ ดังต่อไปนี้

เริ่มจากขอพิจารณาที่สิ่งกีดขวางก่อน โดยกำหนดให้สิ่งกีดขวางที่ใช้ในวิธีการที่เสนอเป็นสิ่งกีดขวางสมมุติซึ่งล้อมรอบสิ่งกีดขวางจริงชิ้นหนึ่งเพื่อให้เส้นทางบินที่อยู่บริเวณรอบๆสิ่งกีดขวางสมมุติอยู่ในระยะปลอดภัยของอากาศยานต่อการชนกับสิ่งกีดขวางจริงและต่อจากนี้เราจะใช้สิ่งกีดขวางในความหมายนี้ตลอดทั้งวิทยานิพนธ์

สิ่งกีดขวางมีลักษณะเป็นรูปปิดทรงหลายเหลี่ยมสองมิติ (Polygon) ซึ่งสร้างจากการลากเส้นตรงเชื่อมจุดยอดแต่ละจุดของรูปปิด ซึ่งสิ่งกีดขวางใดๆสามารถเขียนให้อยู่ในรูปของคู่ลำดับของจุดยอดทั้งหมด คือ



รูปที่ 4.2 ลักษณะของสิ่งกีดขวาง

$$P_i = v_{i1}, v_{i2}, v_{i3}, \ldots, v_{im-1}, v_{im} \tag{4.1}$$

เมื่อ m เป็นจำนวนจุดยอด โดยมีคู่ลำดับ (v_{ij}, v_{ij+1}) ซึ่ง $1 \le j < m$ และคู่ลำดับ (v_{im}, v_{i1}) และสามารถกำหนดให้จุดยอดทั้งหมดของสิ่งกีดขวางใดๆแทนในรูปของเซตได้เป็น

$$O_i = \{v_{i1}, v_{i2}, v_{i3}, \ldots, v_{im-1}, v_{im}\} \tag{4.2}$$

กราฟ G ของโนดที่ใช้ค้นหาเส้นทางโดยอัลกอริธึมวางแผนเส้นทางประกอบด้วยเซตของจุดยอด (Vertex) ของสิ่งกีดขวางเป็นโนด และเซตของเส้นขอบ (Edge) ที่เชื่อมระหว่างโนด

กำหนดให้ E เป็นเซตของเส้นตรงในกราฟ G ซึ่งเส้นตรงแต่ละเส้นเป็นคู่ของโนด (u, v) โดยที่ $u, v \in V$ ซึ่งเหมือนกราฟทั่วไปแต่ได้นิยาม V ใหม่โดยเพิ่มโนดเริ่มต้น s และโนดจุดหมาย g ซึ่งเขียนได้เป็นสมการ (4.3) เมื่อมีสิ่งกีดขวางทั้งหมด n ตัว

$$V = \{s, g\} \cup \bigcup_{i=1}^{n} O_i \tag{4.3}$$

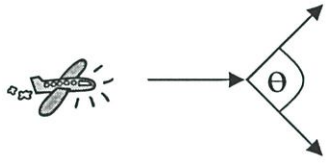
ดังนั้นกราฟที่นิยามข้างต้นมีเส้นทางที่เป็นไปได้ทั้งหมดสำหรับการเดินทางเลี่ยงสิ่งกีดขวางเป็นจำนวนจำกัด และจากนี้ไปในวิทยานิพนธ์จะใช้คำว่า “สเปซ หรือ กราฟ” แทนสเปซในความหมาย

นี้สำหรับการค้นหาแบบเอสตาร์ซึ่งได้ลดจำนวนเส้นทางที่เป็นไปได้ทั้งหมดจากที่เป็นอนันต์ให้เหลือเป็นจำนวนที่จำกัดแล้ว

4.3 ลักษณะการบินเคลื่อนที่ของอากาศยานไร้นักบิน

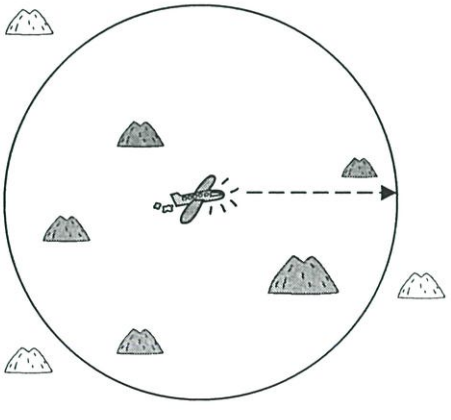
ลักษณะการบินเคลื่อนที่ของอากาศยานไร้นักบินก็เป็นอีกปัจจัยหนึ่งที่มีผลต่อการวางแผนเส้นทางบินเนื่องจากเส้นทางที่สั้นที่สุดสำหรับการเคลื่อนที่ไปซึ่งอาจเป็นเส้นทางที่อากาศยาน ไม่สามารถบินไปได้จริงเพราะขัดกับลักษณะการบินที่ปลอดภัย เช่น บางเส้นทางต้องทำมุมเลี้ยว 90 องศา ซึ่งอากาศยานประเภทเครื่องบินไม่สามารถทำได้ เป็นต้น สำหรับลักษณะการเคลื่อนที่ของอากาศยานไร้นักบินแต่ละประเภทก็มีความแตกต่างกันออกไป สำหรับในวิทยานิพนธ์นี้สามารถรองรับลักษณะการบินแต่ละประเภทได้ ซึ่งมักมีข้อจำกัดการเคลื่อนที่ที่สำคัญ ได้แก่

1. Maximum turning angle เป็นการกำหนดมุมการเลี้ยวสูงสุดที่อากาศยานสามารถทำได้



รูปที่ 4.3 มุมการเลี้ยวสูงสุด (Θ)

2. Route distance constraint เป็นการกำหนดระยะการบินที่ไกลสุด เพราะต้องพิจารณาเรื่องปริมาณน้ำมันที่มีจำกัดด้วย

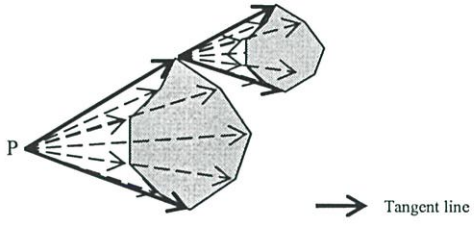


รูปที่ 4.4 ระยะทางไกลสุดของการบิน

งานวิจัยของ Szczerba [2] มีการเสนอให้นำเอาลักษณะการบินของอากาศยานไร้นักบินมาใช้ร่วมกับการค้นหาแบบเอสตาร์ ซึ่งสามารถช่วยลดจำนวนโนดที่การค้นหาแบบเอสตาร์ต้องพิจารณาลง ส่งผลทำให้เพิ่มประสิทธิภาพความเร็วในการวางแผนเส้นทาง ซึ่งในหัวข้อถัดไปจะแสดงการประยุกต์ใช้วิธีการนี้ด้วย

4.4 การนำเอาวิธีเอสตาร์มาประยุกต์ใช้ในการวางแผนเส้นทางบิน

การวางแผนเส้นทางบินเป็นการเลี่ยงสิ่งกีดขวางโดยพิจารณาหาเส้นตรงที่สัมผัสกับจุดยอดต่างๆของรูปหลายเหลี่ยมและไม่ตัดผ่านสิ่งกีดขวางด้วย ดังแสดงในรูปที่ 4.5 หรืออาจกล่าวอีกนัยหนึ่งว่า เป็นการวางแผนเส้นทางบนกราฟที่เกิดจากการเชื่อมจุดยอดหนึ่งๆของสิ่งกีดขวางหนึ่งไปยังจุดยอดหนึ่งๆของอีกสิ่งกีดขวางหนึ่ง และใช้กราฟเหล่านี้สำหรับการค้นหาเส้นทาง



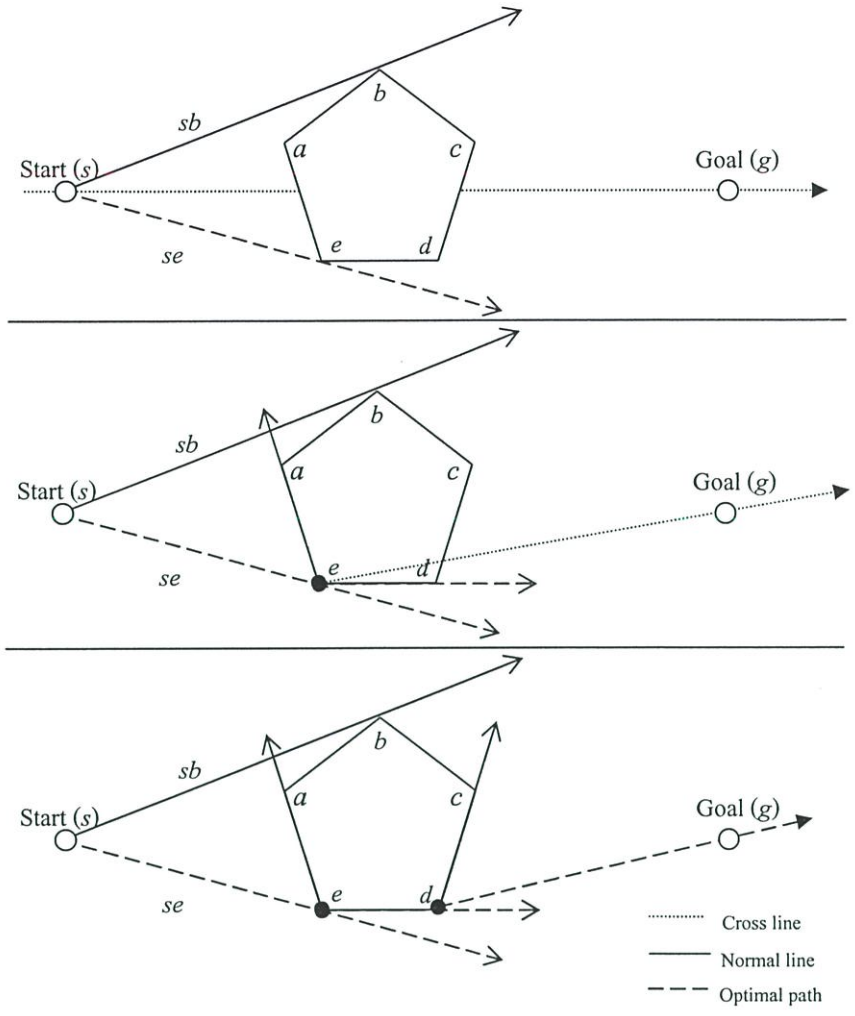
รูปที่ 4.5 การหาเส้นสัมผัสรูปหลายเหลี่ยม

จากนี้ไปเราจะใช้รูปที่ 4.6 เพื่ออธิบายวิธีการวางแผนเส้นทางบินเพื่อเลี่ยงสิ่งกีดขวางโดยอาศัยการลากเส้นเชื่อมจุดยอดของสิ่งกีดขวางอันหนึ่งไปยังจุดยอดของสิ่งกีดขวางอันถัดไปโดยทำทีละครั้งจนสามารถเลี่ยงสิ่งกีดขวางไปยังโนดเป้าหมายได้

พิจารณาเส้นตรงลากเชื่อมจากโนดเริ่มต้น s ไปยังจุดยอดต่างๆของสิ่งกีดขวางรวมทั้งโนดจุดหมาย g ด้วย จะได้เส้น sb , se และ sg ซึ่งเส้นตรง sg ตัดผ่านสิ่งกีดขวาง (Cross line) จะตัดทิ้งจากนั้นเลือกโนดที่สั้นที่สุดด้วยวิธีการค้นหาแบบเอสตาร์โดยกำหนดให้ $h(n)$ เป็นค่าความยาวของเส้นตรงที่เชื่อมโนดที่พิจารณาไปยังโนดจุดหมายโดยไม่สนใจว่าเส้นตรงจะตัดสิ่งกีดขวางหรือไม่ และให้ $g(n)$ เป็นระยะทางจริงจากโนดเริ่มต้นมาถึงโนดปัจจุบันซึ่งกำหนดให้เป็นโนด n มาถึงขั้นตอนนี้เราสมมุติให้ $f(e) < f(b)$ ดังนั้นการค้นหาแบบเอสตาร์จึงเลือกโนด e เพื่อค้นหาโนดลูกต่อไป

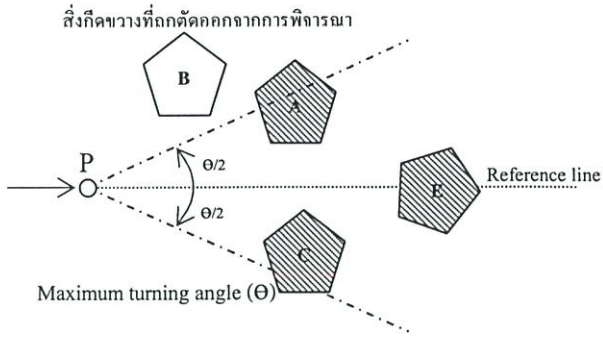
จากนั้นพิจารณาจากโนด e มีเส้นตรง ea , ed และ eg แต่ eg ตัดผ่านสิ่งกีดขวางจึงไม่นำมาพิจารณา เมื่อรวมโนดลูกทั้งหมดจะได้เป็นโนด a มีเส้นทาง $s->e->a$ b มีเส้นทาง $s->b$ และ d มีเส้นทาง $s->e->d$ ในกรณีนี้สมมุติให้ว่าการค้นหาแบบเอสตาร์เลือกโนด d ซึ่งได้เส้นตรง de dc และ dg จากการค้นหาพบว่ามีเส้นตรงเชื่อมจากโนด d ไปยังโนดจุดหมายโดยไม่

ตัดสิ่งกีดขวางใดๆ ดังนั้นเส้นทางที่ได้จากการค้นหาคือ $s->e->d->g$ ซึ่งเป็นเส้นทางเลี่ยงสิ่งกีดขวางจาก s ไปยัง g ที่สั้นที่สุด



รูปที่ 4.6 การเลี่ยงสิ่งกีดขวางเพียงสิ่งกีดขวางเดียว

ในขณะที่การค้นหาแบบเอสตาร์ขยายการค้นหาต่อ เราได้นำเอาข้อมูลลักษณะการบินของอากาศยานไร้นักบินในเรื่องมุมเลี้ยวสูงสุดมาใช้ในการพิจารณาคัดเลือกกลุ่มสิ่งกีดขวางที่จะนำมาใช้ค้นหาเส้นทางต่อ ทำให้เราสามารถตัดสิ่งกีดขวางบางส่วนที่อยู่นอกขอบเขตของมุมเลี้ยวสูงสุด (กำหนดให้เป็นค่า Θ) ออกไปได้ ดังแสดงในรูปที่ 4.7 ในที่นี้มุม Θ จะถูกแบ่งครึ่งโดยเส้นตรงอ้างอิง (Reference line) ซึ่งเป็นเส้นตรงแนวเดียวกับเส้นเวคเตอร์ที่ลากจากโนดก่อนหน้านี้มายังโนดที่กำลังพิจารณา ซึ่งค่ามุมเลี้ยวสูงสุดนี้ขึ้นอยู่กับลักษณะการบินของอากาศยานไร้นักบินแต่ละประเภท



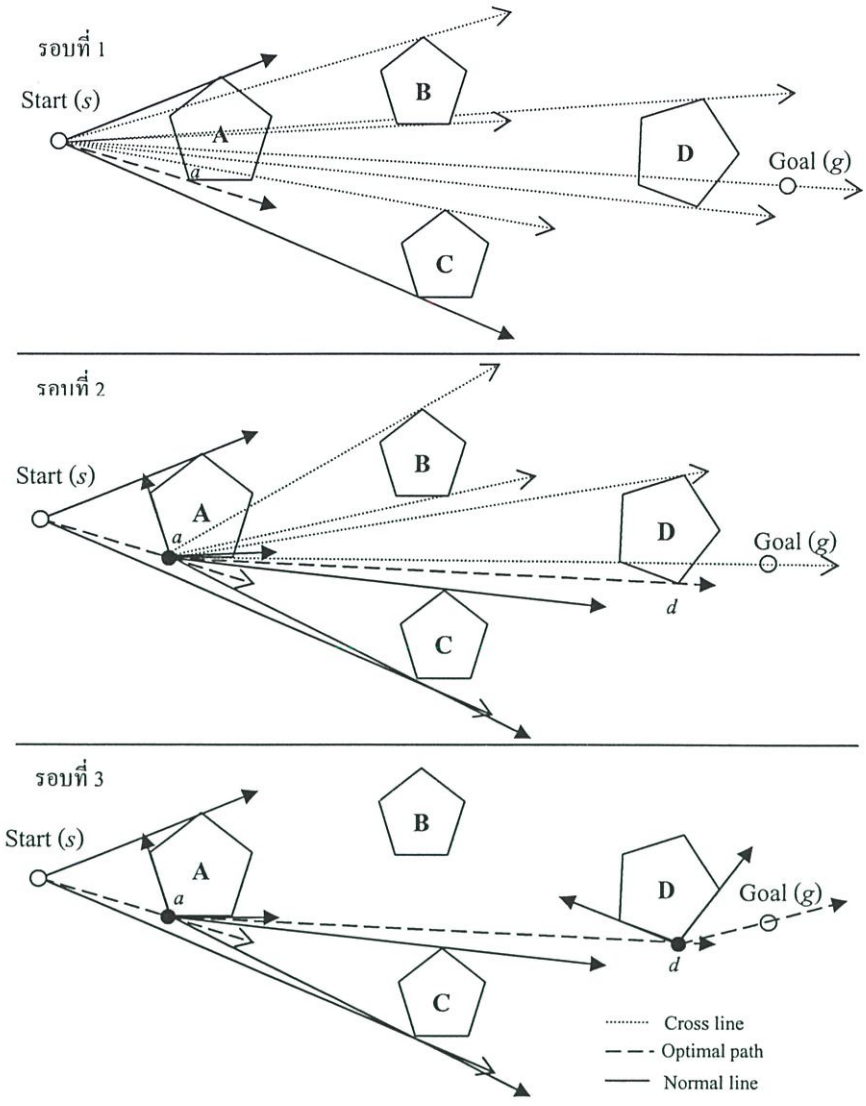
รูปที่ 4.7 แสดงการเลือกสิ่งกีดขวางที่อยู่ในช่วงองศาที่กำหนด

ยังมีอีกหนึ่งเงื่อนไขที่ใช้ในการตัดโนดออกจากการพิจารณาของการค้นหาแบบเอสตาร์คือ ใช้เงื่อนไขระยะทางไกลสุดในการบินซึ่งอธิบายได้ดังนี้ ถ้าโนดที่กำลังพิจารณามีค่าประเมินซึ่งได้จากระยะทางของเส้นทางจากโนดเริ่มต้นมาถึงโนดที่พิจารณาและค่าประมาณไปยังจุดหมายแล้วมากกว่าระยะทางไกลสุดในการบินซึ่งกำหนดให้มีค่าเป็น L แล้วเราสามารถตัดโนดนี้ออกจากการพิจารณาของการค้นหาแบบเอสตาร์ได้

มาถึง ณ ตอนนี้ เมื่อนำเอาลักษณะการบินทั้งสองเงื่อนไขมาพิจารณาร่วมด้วยในการวางแผนเส้นทาง เราสามารถสรุปขั้นตอนการค้นหาเส้นทางที่สมบูรณ์ได้ดังนี้

1. กำหนดให้โนดเริ่มต้นและโนดจุดหมายเป็นเสมือนจุดยอดของสิ่งกีดขวาง และเริ่มพิจารณาจากโนดเริ่มต้นเป็นโนดแรก โดยเก็บไว้ในลิสต์ที่ใช้เก็บโนดที่เรียงลำดับตามค่าประเมินจากน้อยไปหามาก
2. ถ้าลิสต์ที่ใช้เก็บโนดว่าง แสดงว่าไม่มีโนดที่นำมาค้นหาเส้นทางให้พิจารณาอีก จึงหยุดทำงาน
3. เลือกโนดที่มีค่า $f(n)$ น้อยที่สุดจากลิสต์
4. ไม่พิจารณาโนดที่มีค่าประเมินมากกว่า L
5. ถ้าโนดที่เลือกมาเป็นโนดจุดหมาย แสดงว่าได้พบเส้นทางที่สั้นที่สุดแล้ว ให้หยุดการทำงาน
6. ขยายโนดลูกด้วยการหาเส้นตรงที่ลากจากโนดที่เลือกไปยังจุดยอดของสิ่งกีดขวางอื่นๆ ทั้งหมดโดยไม่ตัดผ่านสิ่งกีดขวางใดๆ และมีช่วงองศาจากเส้นอ้างอิงไม่เกินค่า Θ/2 เก็บไว้ในลิสต์ และกลับไปทำขั้นตอนที่ 2

จากขั้นตอนการทำงานข้างต้น เราสามารถแสดงขั้นตอนการค้นหาเส้นทางได้ ดังรูปที่ 4.8 โดยกำหนดให้ Θ = 60 องศาและ L=500



รูปที่ 4.8 การวางแผนเส้นทางที่สามารถเลี่ยงสิ่งกีดขวาง

มีการทำงานวนซ้ำทั้งหมด 3 รอบ โดยแต่ละรูปแสดงการทำงานในแต่ละรอบ เริ่มจากรอบที่ 1 โดยเริ่มพิจารณาจากโนดเริ่มต้น s ไปยังสิ่งกีดขวางต่างๆรวมทั้งโนดจุดหมาย g ด้วย เมื่อพิจารณาเส้นทั้งหมดที่สัมผัสสิ่งกีดขวาง เราสามารถตัดเส้นตรงที่ตัดผ่านสิ่งกีดขวางออกจะเหลือเพียง 3 เส้น โดยอาศัยเงื่อนไข 1) มีมุมเลี้ยวสูงกว่า Θ 2) เป็นเส้นที่ตัดผ่านสิ่งกีดขวาง จากนั้นเลือกโนดทั้ง 3 โดยวิธีการค้นหาแบบเอสตาร์จะได้โนด a มาพิจารณาเป็นโนดต่อไปเหมือนเป็นโนดเริ่มต้นซึ่งโนด a เป็นจุดยอดของสิ่งกีดขวาง A ที่มีเส้นประเป็นเส้นสัมผัสโนด

รอบต่อมาจะทำงานเหมือนรอบที่ 1 เมื่อพิจารณาโนดที่มีค่า $f(n)$ น้อยที่สุดจะได้โนด d ของสิ่งกีดขวาง D มาพิจารณา และรอบสุดท้ายสามารถตัดสิ่งกีดขวาง A, B และ C ออกจากการพิจารณาเพราะมีค่ามุมที่อ้างอิงกับ Reference line มากกว่า Θ /2 จากนี้เราสามารถลากเส้นตรงจากโนด d ของสิ่งกีดขวาง D ไปยังโนดเป้าหมาย g ได้ แล้วจึงสิ้นสุดการค้นหาเส้นทางและได้เส้นทางที่สั้นที่สุดที่ค้นพบคือ $s \rightarrow a \rightarrow d \rightarrow g$

```
01 global obstacle_list is an array of obstales
02 global max_turn_angle is maximum turning angle
03 global max_distance is maximum distance
04
05 def ASTAR_SEARCH (initial, goal)
06    add( obstacle_list, goal) <- Add goal into obstacle_list.
07    list = [] <- Create an empty array for keep nodes which be expanded.
08    add( list, start) <-Add start into list.
09    while list is not empty:
10       n <- remove a node which is lowest evaluate f() from list.
11       if f(n) > max_distance then
12          exit function with null.
13       if n == goal then
14          exit function with n.
15       EXTEND (list, EXPAND(n)) <- Extend list, a array of nodes with children of n.
16    return null.
17
18 def EXPEND(n):
19    nodes = [] <- Create an empty list
20    for each obstacle in obstacle_list:
21       points <- get vertexes of obstacle
22       for each p in points:
23          line = create_line(n, p) <- Create line which consist of node point and p point.
24          If turning degree of p < max_turn_angle and not IS_CROSS( line ) then
25             add(nodes, p) <- Add p into nodes
26    return nodes.
27
28 def IS_CROSS( line )
29    for each obstacle in obstacle_list:
30       for each edge in obstacle :
31          if edge cross line then
32             return TRUE <- Exit function with TRUE.
01    return FALSE
```

รูปที่ 4.9 อัลกอริธึมการวางแผนเส้นทางบินที่สามารถเลี่ยงสิ่งกีดขวาง

จากขั้นตอนการวางแผนเส้นทางทั้ง 3 ขั้นตอนข้างต้น เราสามารถเขียนอยู่ในรูปอัลกอริธึมดังรูปที่ 4.9 ซึ่งประกอบด้วยฟังก์ชั่นหลักๆ ดังนี้

1. ฟังก์ชั่นหลัก SEARCH(*initial*, *goal*) เป็นฟังก์ชั่นที่ทำงานเหมือนกับการค้นหาเส้นทางแบบเอสตาร์แต่ได้ปรับปรุงเพิ่มเติมในบรรทัดที่ 11-12 คือจะหยุดทำงานเมื่อพบว่าเส้นทางมีระยะทางเกินระยะระยะการบินที่ไกลสุด
2. ฟังก์ชั่น EXPAND(*n*) เป็นฟังก์ชั่นที่สร้างขึ้นมาใหม่ซึ่งทำหน้าที่หาโนดของแต่ละสิ่งกีดขวาง ซึ่งต้องเป็นโนดที่ลากเส้นเชื่อมแล้วไม่ตัดผ่านสิ่งกีดขวาง
3. ฟังก์ชั่น IS_CROSS(*line*) ทำหน้าที่ทดสอบว่าเส้นตรง *line* ตัดผ่านสิ่งกีดขวางใดสิ่งกีดขวางหนึ่งหรือไม่
4. ฟังก์ชั่น EXTEND(*list*, *nodes*) ทำหน้าที่เพิ่มโนดในลิสต์

4.5 สรุป

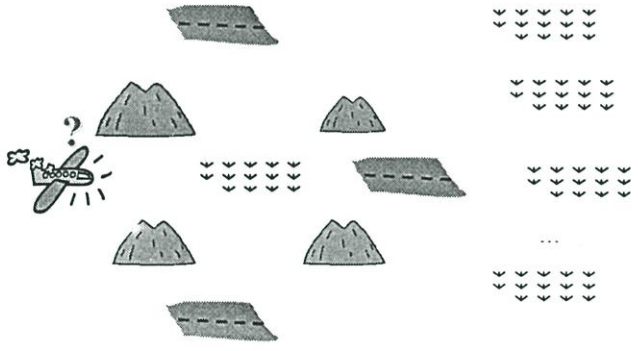
อัลกอริธึมเอสตาร์เป็นอัลกอริธึมที่สามารถวางแผนเส้นทางได้อย่างมีประสิทธิภาพและได้ผลลัพธ์ที่ดีที่สุดภายใต้เงื่อนไขที่ฟังก์ชั่นฮิวริสติกมีคุณสมบัติ Monotonicity สำหรับการค้นหาเส้นทางบนกราฟและคุณสมบัติ Admissible สำหรับการค้นหาบนทรี แต่การค้นหาที่ทำงานบนกราฟที่มีโนดจำนวนมากอาจเป็นอนันต์ที่จะทำให้ต้องใช้หน่วยความจำในการเก็บข้อมูลทั้งหมดไว้หรือไม่สามารถจัดเก็บได้หมด ดังนั้นในบทนี้จึงได้เสนอวิธีการลดจำนวนโนดในสเปซลงโดยใช้การเลี่ยงรูปปิดหลายเหลี่ยมแทนสิ่งกีดขวางจริงประกอบกับการใช้ข้อมูลลักษณะการเคลื่อนที่ของอากาศยานเป็นการลดจำนวนโนดในสเปซให้เป็นจำนวนที่จำกัดจึงจะสามารถใช้อัลกอริธึมเอสตาร์ได้

บทที่ 5
การค้นหาจุดหมายที่ดีที่สุด ในกรณีที่มีจำนวนจุดหมายมากมายหรือไม่จำกัด

จากที่ได้กล่าวมาในบทที่ 4 เราสามารถนำสเปซที่นิยามใหม่มาใช้กับการค้นหาแบบเอสตาร์ ซึ่งสามารถเลี่ยงสิ่งกีดขวางจากจุดเริ่มต้นไปยังจุดหมายหนึ่งๆ ได้ แต่ในบางครั้งการเดินทางของอากาศยานไร้นักบินอาจจะมีอุปสรรคที่ทำให้ไม่สามารถเดินทางไปถึงจุดหมายได้และต้องหาสถานที่ที่สามารถลงจอดฉุกเฉินได้ซึ่งสถานที่นี้อาจจะมีจำนวนมากหรือจำนวนไม่จำกัด ในบทนี้จะกล่าวถึงอัลกอริธึมที่ใช้แก้ปัญหานี้

5.1 ปัญหาการค้นหาจุดหมายที่ดีที่สุดจากหลายจุดหมายหรือมีจำนวนไม่จำกัด

การเดินทางในการค้นหาวางแผนเส้นทางบินของอากาศยานเป็นปัญหาที่มีความสำคัญ เพราะมีผลต่อความปลอดภัยในการเดินทาง ซึ่งมีปัญหาหลัก คือ ขนาดของสเปซที่มีไม่จำกัดอันเนื่องมาจากจำนวนโนดที่ต้องพิจารณามีจำนวนมากมายนับไม่ถ้วน เพราะเมื่ออากาศยานบินจากจุดเริ่มต้นหลบเลี่ยงสิ่งกีดขวางไปยังจุดหมายผ่านไปในสภาพแวดล้อมจริง ณ ตำแหน่งใดๆ มันสามารถเคลื่อนที่ไปได้หลายทิศทางและสามารถเลี้ยวได้ในหลายตำแหน่งเช่นเดียวกัน ทำให้เส้นทางบินของอากาศยานมีความเป็นไปได้มากมายนับไม่ถ้วนหรือเป็นอนันต์นั่นเอง ซึ่งได้มีการแก้ปัญหานี้ไปแล้วในบทที่ผ่านมา



รูปที่ 5.1 การค้นหาเส้นทางในกรณีที่มีจุดหมายจำนวนมาก

อย่างไรก็ตามการค้นหาเส้นทางโดยวิธีเอสตาร์ได้ถูกออกแบบมาสำหรับการค้นหาเส้นทางไปยังจุดหมายเดียว แต่การวางแผนเส้นทางบินในความเป็นจริงอาจมีจุดหมายจำนวนมากหรือนับไม่ถ้วน เช่น จากรูปที่ 5.1 ในกรณีที่มีเหตุการณ์ผิดปกติซึ่งอากาศยานจำเป็นจะต้องลงจอดฉุกเฉินที่

ใดก็ได้เพียงที่เดียว แต่มีสถานีหรือสถานที่ที่สามารถลงจอด (จุดหมาย) ได้หลายแห่งหรือลงอาจจอดบนพื้นที่ใดๆก็ได้ (จำนวนจุดหมายเป็นอนันต์) ซึ่งอาจจะเป็นถนน นาข้าว หรือทุ่งหญ้า จากปัญหาดังกล่าวและข้อจำกัดของอัลกอริธึมการค้นหาแบบเอสตาร์ที่ทำงานได้เมื่อมีจำนวนจุดหมายจำกัดทำให้ไม่สามารถนำมาใช้แก้ปัญหาในกรณีที่มีจำนวนจุดหมายมากมายเป็นอนันต์ได้

จำนวนอนันต์ ในที่นี้หมายถึง เมื่อกำหนดขอบเขตบริเวณที่จะทำการวางแผนเส้นทาง เราสามารถหาจำนวนของจุดหมายได้ แต่ในความเป็นจริง ขอบเขตนี้สามารถขยายไปได้เรื่อยๆไม่จำกัด ซึ่งทำให้จำนวนจุดหมายค่อยๆมีมากขึ้นเรื่อยๆ ไม่จำกัดเช่นกัน ซึ่งเรียกลักษณะเช่นนี้ว่า สเปซของการค้นหามีอยู่อย่างไม่จำกัด(Finite) และขอบเขตของการค้นหาจะขยายขออกไปได้เรื่อยๆอย่างไม่จำกัด (Unbound) อย่างไรก็ตามเราสามารถคาดคะเนระยะทางโดยประมาณของจุดหมายทั้งหมดได้เมื่อเทียบกับขอบเขตการค้นหานี้ จากนี้ไปเราจะใช้จำนวนอนันต์ในความหมายนี้ในวิทยานิพนธ์ฉบับนี้

การวางแผนเส้นทางที่มีจำนวนจุดหมายมากเป็นปัญหาในการเลือกจุดหมายที่ดีที่สุดเพียงจุดหมายเดียวซึ่งเป็นจุดหมายที่มีระยะทางที่สั้นที่สุดจากจุดเริ่มต้นมายังจุดหมายนั้นเมื่อเปรียบเทียบเส้นทางของจุดหมายอื่นๆ จำนวนจุดหมายที่มากขึ้นทำให้ต้องนำมาเปรียบเทียบมากขึ้นด้วย เราอาจสามารถแก้ปัญหานี้โดยการใช้การประมวลผลแบบขนานด้วยการใช้เครื่องคอมพิวเตอร์หลายๆตัวมาทำงานพร้อมๆ กัน และต้องออกแบบการติดต่อสื่อสารระหว่างซีพียูแต่ละตัว ได้แก่ งานวิจัยของ [14, 15, 16] ซึ่งได้กล่าวไว้แล้วในบทที่ 2 แต่วิธีการนี้ต้องใช้ทรัพยากรจำนวนมาก แต่ถ้าในกรณีที่มีจำนวนจุดหมายเป็นอนันต์ วิธีการดังกล่าว อาจจะไม่สามารถใช้ได้ ในวิทยานิพนธ์นี้จึงได้นำเสนออัลกอริธึมที่สามารถตัดจุดหมายส่วนหนึ่งออกจากการพิจารณาก่อน และเลือกจุดหมายที่ความว่าดีที่สุดกลุ่มหนึ่งมาพิจารณาก่อน ถ้าพบว่าจุดหมายไม่ได้อยู่ในกลุ่มนี้ ก็จะขยายขอบเขตการค้นหาออกไปอีกเพื่อนำจุดหมายอื่นๆมาร่วมพิจารณา

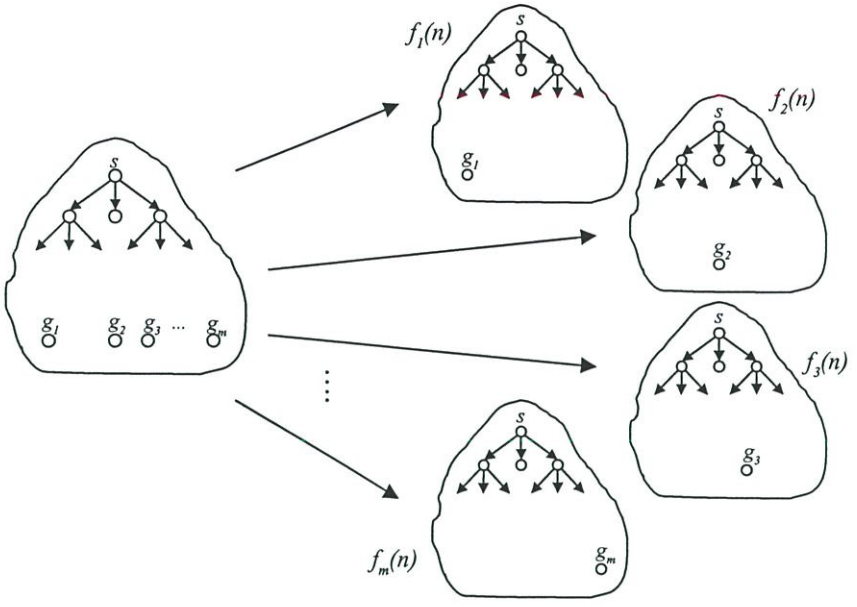
ในบทนี้จะเสนอวิธีการใหม่ในการวางแผนเส้นทางบินที่สั้นที่สุดที่มีจุดหมายจำนวนไม่จำกัดสำหรับอากาศยานไร้นักบินด้วยวิธีการค้นหาเอสตาร์แบบค้นหาลึกลงไป (Deepening A* Search) ซึ่งสามารถค้นหาจุดหมายที่ดีที่สุดจากจุดหมายจำนวนมากมายเป็นอนันต์ได้ โดยการดัดแปลงมาจากอัลกอริธึมเอสตาร์เพื่อรับประกันได้ว่าจุดหมายที่ได้ให้เส้นทางที่สั้นที่สุด

5.2 อัลกอริธึมเอสตาร์แบบค้นหาลึกลงไป

อัลกอริธึมเอสตาร์แบบค้นหาลึกลงไป (Deepening A* Search) สามารถใช้ค้นหาจุดหมายที่ดีที่สุดจากจุดหมายจำนวนอนันต์ได้อย่างมีประสิทธิภาพ ซึ่งแนวคิดที่มา สามารถอธิบานได้ดังนี้

เราจะพิจาณาว่าในสเปซการค้นหาเส้นทางที่มีหลายๆจุดหมาย ถ้าเราแยกพิจารณาการค้นหาของแต่ละจุดหมายออกเป็นการค้นหาที่อิสระจากกันแต่ทำหลายๆครั้ง การค้นหาแต่ละครั้งเรามุ่งพิจารณาไปที่จุดหมายหนึ่งๆ เพียงจุดหมายเดียว โดยอาศัย $f(n)$ ของจุดหมายนั้นซึ่งเสนอใน [14,

15, 16, 17] ดังรูปที่ 5.2 เราจะนำมาผลเส้นทางซึ่งเป็นคำตอบของการค้นหาของแต่ละจุดหมายมาเปรียบเทียบกันเพื่อหาเส้นทางที่สั้นที่สุด จุดหมายใดที่มีเส้นทางสั้นที่สุด จุดหมายนั้นก็เป็นจุดหมายที่ดีที่สุด



รูปที่ 5.2 การค้นหาเส้นทางโดยค้นหาเส้นทางของแต่ละจุดหมายแยกจากกันแต่ทำงานเป็นอิสระกัน

ด้วยวิธีการเช่นนี้ เราสามารถทำการประมวลผลแบบขนานได้ อย่างไรก็ตามการคำนวณแบบนี้มีความสิ้นเปลืองมากเนื่องจากการค้นหากราฟเกิดความซ้ำซ้อนกันมาก ดังนั้นเราจึงได้แก้ปัญหาด้วยวิธีการค้นหาด้วยการคำนวณแบบลำดับ (Sequential) ซึ่งใช้กราฟหรือสเปซของการค้นหาร่วมกันและกำหนดวิธีการเลือกพิจารณาจุดหมายเพียงจุดหมายเดียวในช่วงเวลาหนึ่งๆ โดยใช้ $f_i(n)$ เป็นตัวกำหนดให้สามารถเปลี่ยนจุดหมายได้ความเหมาะสมและทำให้ยังคงมีคุณสมบัติ Optimality ซึ่งจะขออธิบายวิธีการดังต่อไปนี้

กำหนดให้ฟังก์ชั่นประเมิน (Evaluation function) ที่ใช้พิจารณาเลือกโนดสำหรับการค้นหาเมื่อพิจารณาที่แต่ละจุดหมาย g_i คือ $f_i(n)$ ซึ่งเขียนได้เป็น

$$f_i(n) = g(n) + h_i(n) \tag{5.1}$$

โดยที่ $1 \le i \le m$ ซึ่ง m คือจำนวนจุดหมาย ซึ่งมีค่าเป็นอนันต์ได้ และ $h_i(n)$ เป็นฟังก์ชั่นฮิวริสติกย่อย ของแต่ละจุดหมาย g_i

และเขียนอธิบายขั้นตอนการทำงานของอัลกอริธึมการค้นหาเส้นทางได้ดังนี้

กำหนดให้ $F = \{f_1(n), f_2(n), ..., f_m(n)\}$ โดย N เป็นเซตของโนดที่ยังค้นหาไม่เสร็จและจะถูกเลือกสำหรับการค้นหาต่อ โดยเริ่มแรกให้ $N = \{start\}$ ก่อน ลำดับการค้นหามีดังต่อไปนี้

1. เลือกพิจารณาจุดหมายที่คาดว่าดีที่สุด โดยอาศัยการเรียงลำดับน้อยไปหามากตามค่า $f_i(n)$ ของแต่ละจุดหมายนำไปคำนวณร่วมกับ N แล้วเลือก $f_i(n)$ ที่ให้ค่าน้อยที่สุด (สังเกตว่า สำหรับ $f_i(n)$ ที่มีค่ามากๆ จะถูกเลือกมาพิจารณาหลังๆ ยิ่งถ้ามีค่ายิ่งมาก ก็อาจจะไม่ถูกพิจารณาเลย ทำให้จุดหมายที่อยู่ไกลๆ ออกไปหรือที่มีเป็นจำนวนอนันต์ถูกตัดทิ้งไปในที่สุด) สมมุติว่า ฟังก์ชั่นที่เลือกมาเป็น $f_s(n)$ ดังนั้น $f_s(n)$ คือฟังก์ชั่นของจุดหมาย s ที่เลือกมาพิจารณา
2. ขยายการค้นหาจากโนด *start* ลงไปยังโนดลูก $\{n_1, n_2, ..., n_k\}$ ของมัน แล้วนำเซตนี้ไปรวมกับ N
3. เลือกโนดใน N ที่มีค่า $f_s(n_i)$ น้อยที่สุดคือ $selected\_N = \min(\{f_s(n_p), ..., f_s(n_q)\})$
4. นำค่า $f_s(selected\_N)$ มาเปรียบเทียบกับค่า $f_i(selected\_N_i)$ ของจุดหมาย i ใดๆที่ไม่ใช่จุดหมาย s (โดยที่ $selected\_N_i$ เป็นโนดล่าสุดที่ถูกเลือกมาค้นหาในการพิจารณาของแต่ละจุดหมาย g_i ในการค้นหารอบก่อนๆ โดยกำหนดให้มีโนดเริ่มต้นเป็นโนด *start* สำหรับทุกๆ จุดหมาย)
    a. ถ้าค่าที่น้อยที่สุดยังคงเป็น $f_s(selected\_N)$ แล้ว ก็จะเป็นการพิจารณาที่จุดหมายเดิม โดยไปทำขั้นตอนที่ 5
    b. มิฉะนั้นแล้วจะเป็นการเปลี่ยนจุดหมายในการพิจารณาค้นหาโดยให้จดจำโนด $selected\_N$ ของจุดหมาย s แทนโนด $selected\_N_s$ เดิม ($selected\_N_s$ เริ่มต้นของแต่ละจุดหมายคือ *start*) ซึ่งจะถูกนำมาใช้พิจารณาอีกครั้งของรอบต่อไป และกำหนดให้ $f_i(n)$ ที่ค่าน้อยที่สุดเป็น $f_s(n)$ แล้วกลับไปทำขั้นตอนที่ 3 ซ้ำอีก
5. จากขั้นตอน 4 ตรวจดูว่า $selected\_N$ เป็นจุดหมายหรือไม่
    a. ถ้าเป็นแสดงว่าได้เส้นทางไปยังจุดหมายที่ดีที่สุดแล้ว จบการทำงาน
    b. มิฉะนั้นตรวจสอบว่าจะค้นหาลึกลงไปอีกระดับ โดยให้ $selected\_N$ เป็นเสมือนโนด *start* และนำ $selected\_N$ ออกจาก N เพื่อไม่ให้นำกลับมาพิจารณาซ้ำอีกซึ่งจะนำมาพิจารณาในขั้นตอนที่ 2

จากคำอธิบายการทำงานข้างต้น เราสามารถเขียนอัลกอริธึมได้ดังรูปที่ 5.3

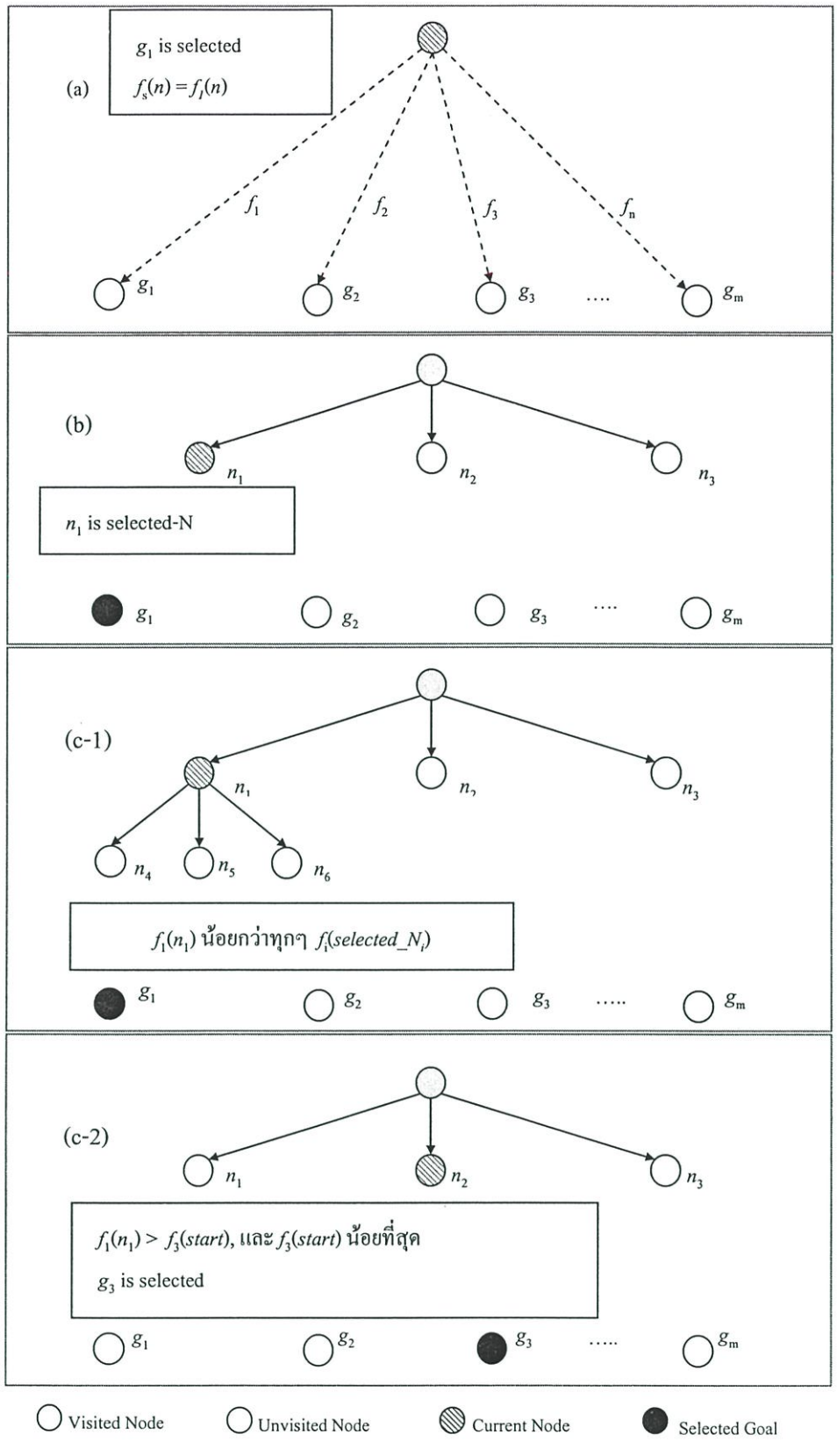
```
def Deepening-AStar-Search(start, GOALS):
   F = {f1, f2, f3, ....}
   N = {start}
   #L is an array of every previous selected node for each goal
   L = {start, start, start, ...}
   fs = select-min(F applied to list of selected_Ni)
   while N not empty
      Let selected_N be a node from N whose fs(n) is minimum
      Let fs' be the function giving min value among fs(selected_N) & fi(selected_Ni)
      if fs' = fs then
        if selected_N is gs then
           return selected_N as the best goal
        else
           N = N – {selected_N}
           N = N U expand(selected_N)
      else
        Store selected_N for gs in L
        fs = fs'
   return None

def expand(node):
   return child nodes of node
```

รูปที่ 5.3 อัลกอริธึมวิธีเอสตาร์แบบค้นหาลึกลงไป

ต่อจากนี้เราใช้รูปที่ 5.4 อธิบายขั้นตอนการทำงาน โดยเริ่มจากรูปที่ 5.4(a) เราเลือกจุดหมายที่ใกล้ที่สุดมาพิจารณาก่อน, สมมุติให้ g_1 เป็นจุดหมายที่ใกล้ที่สุด, และใช้ $f_1(n)$ สำหรับการคำนวณค่าเพื่อเลือกโนดด้วยวิธี A* หลังจากเลือกจุดหมายแล้วนำจุดเริ่มต้นมาขยายการค้นหาไปยังโนดลูก และเลือกโนดลูกที่มีค่า $f_1(n)$ น้อยที่สุด เราสมมุติให้โนดลูก n_1 เป็นโนดที่มีค่า $f_1(n_1)$ น้อยที่สุดดังรูปที่ 5.4(b) จากนั้นนำ $f_1(n_1)$ มาเปรียบเทียบกับค่า $f_i(selected\_N)$ สำหรับจุดหมายอื่นๆ เพื่อหาค่าน้อยที่สุด ถ้ายังเป็น $f_1(n)$ อยู่จะพิจารณาจุดหมายเดิมลึกลงไปอีกขั้นดังรูปที่ 5.4(c-1) แต่ถ้า $f_1(n)$ ไม่ใช่ค่าน้อยที่สุดโนด n_1 จะเป็น $selected\_N$ ล่าสุดของจุดหมาย g_1 ซึ่งจะถูกเก็บไว้ในอาร์เรย์สำหรับการเปรียบเทียบต่อไป เราสมมุติให้จุดหมายใหม่ที่ดีกว่า $f_1(n)$ คือ $f_3(n)$ เมื่อกลับไปเลือกโนดที่มีค่าน้อยที่สุดด้วย $f_3(n)$ แล้ว จะได้โนด n_2 มาพิจาณาดังรูปที่ 5.4(c-2) แล้วทำการขยายการค้นหาไปยังโนดลูกต่อไปซึ่งจะทำขั้นตอนดังรูป 5.4(a) ซ้ำอีกเสมือนว่า n_2 เป็นจุดเริ่มต้นอีกครั้งจนกระทั่งโนดที่ถูกเลือกมาจะเป็นจุดหมายซึ่งแสดงว่าได้เส้นทางที่สั้นที่สุดแล้ว ซึ่งเราก็จะบอกได้ว่าจุดหมายที่พบเป็นจุดหมายที่ดีที่สุด

เราจะสังเกตว่าอัลกอริธึมนี้มีการทำงานในสองลักษณะคือที่รูป 5.4(c-1) อัลกอริธึมนี้จะค้นหาในแนวลึกที่โนด $selected\_N$ อีก 1 ขั้น ส่วนในรูปที่ 5.4(c-2) จากการสลับเปลี่ยนไปพิจารณาจุดหมายที่จุดหมายอื่นแทนทำให้อัลกอริธึมมีการค้นหาโนดในแนวกว้างแทนแนวลึก จากลักษณะการค้นหาที่ทำแนวลึกสลับกับแนวกว้างแต่ยังใช้วิธีเอสตาร์ซึ่งขอเรียกอัลกอริธึมนี้ว่า "Deepening A* Search" หรือ "การค้นหาเอสตาร์แบบค้นหาลึกลงไป"



รูปที่ 5.4 อธิบายการทำงานของอัลกอริธึมวิธีเอสตาร์แบบค้นหาลึกลงไป

5.3 จุดหมายจำนวนอนันต์กับอัลกอริธึมการค้นหาเอสตาร์แบบค้นหาลึกลงไป

ถ้าเรากำหนดให้ $g_i \in GOALS$ และ $g_k \in GOALS$ การทำงานของอัลกอริธึมจะเริ่มจากเลือกจุดหมาย g_i แล้วใช้จุดหมายนี้ในการประเมินโนดต่างๆไปจนกระทั่งเส้นทางที่ไปหาจุดหมาย g_i มีแนวโน้มว่าไม่ใช่เส้นทางที่สั้นที่สุดและทำให้จุดหมายนั้นไม่ใช่จุดหมายที่ดีที่สุดโดยใช้ $f_i(n)$ ในการพิจารณา อัลกอริธึมจะเปลี่ยนจุดหมาย g_k ที่คาดว่ามีเส้นทางที่สั้นกว่าและใช้ $f_k(n)$ ในการพิจารณาแทน $f_i(n)$ และจะทำเช่นนี้ซ้ำๆไปจนกระทั่งมีเส้นทางไปถึงจุดหมายและไม่มีจุดหมายอื่นที่คาดว่ามีเส้นทางที่ดีกว่า อัลกอริธึมจะหยุดทำงานซึ่งจุดหมายที่มีค่าประเมินมากกว่าเส้นทางที่ดีที่สุดจะไม่ถูกนำมาพิจารณาเลย ด้วยเหตุผลนี้เองทำให้อัลกอริธึมการค้นหาเอสตาร์แบบค้นหาลึกลงไปสามารถตัดจุดหมายที่อยู่ไกลมากๆออกไปได้ หรือที่เรียกว่า Pruning ซึ่งทำให้จำนวนจุดหมายที่มากมายเป็นจำนวนอนันต์ถูกตัดทิ้งไปให้เหลือเป็นจำนวนที่จำกัด

การทำงานของอัลกอริธึมที่มีพื้นฐานมาจากอัลกอริธึมเอสตาร์จะมีข้อเสียในเรื่องการใช้พื้นที่หน่วยความจำ อัลกอริธึมเหล่านี้จะทำงานด้วยการสร้างโนดลูกไปเรื่อยๆ และในขณะเดียวกันก็จะเก็บโนดที่เคยพิจารณาแล้วไว้ซึ่งหน่วยความจำจะถูกใช้เพิ่มขึ้นตามไปด้วยและถ้ามีจำนวนโนดเป็นอนันต์โดยที่จุดหมายก็เป็นโนดด้วยเช่นกัน เราไม่สามารถหาหน่วยความจำที่มีขนาดที่ใหญ่เพียงพอในการทำงาน ทำให้ปัญหาการใช้หน่วยความจำมีความสำคัญมากกว่าเวลาที่ใช้ในการประมวลผลเสียอีก ดังนั้นเมื่อสามารถลดจำนวนโนดและจำนวนจุดหมายลงได้ (ไม่เป็นจำนวนอนันต์) แล้วเราสามารถใช้อัลกอริธึมการค้นหาเอสตาร์แบบค้นหาลึกลงไปมาทำงานได้และยังคงหาผลลัพธ์ที่ดีที่สุดได้อีกด้วย

5.4 การประยุกต์ใช้ในการวางแผนเส้นทางบิน

เราสามารถนำอัลกอริธึมเอสตาร์แบบค้นหาลึกลงไปมาใช้วางแผนเส้นทางบินสำหรับอากาศยานไร้นักบินในกรณีที่จุดหมายอาจจะมีมากกว่าหนึ่งจุดหมายโดยใช้สเปซซึ่งนิยามใหม่ด้วยและการนำข้อมูลลักษณะการบินเคลื่อนที่ของอากาศยานไร้นักบินได้แก่ ระยะทางบินได้ไกลที่สุดและมุมเลี้ยวสูงสุด ตามที่ได้กล่าวมาแล้วในบทที่ 3 เราจึงเขียนเป็น Pseudo code ดังในรูปที่ 5.5

5.5 ภาพรวมของอัลกอริธึมการค้นหาเอสตาร์แบบค้นหาลึกลงไป

เมื่อพิจารณาอัลกอริธึมเอสตาร์แบบค้นหาลึกลงไปโดยรวมแล้วจะเห็นว่าเป็นการผสมผสานระหว่าง Breadth-first Search และ A* Search คือเป็นการค้นหาในแนวกว้างแต่จะถูกชี้นำในการค้นหาทางแนวลึกโดยเอสตาร์เนื่องจากอัลกอริธึมนี้จะเลือกค้นหาโดยเลือกพิจารณาที่จุดหมายห่างจากจุดเริ่มต้นจากน้อยไปหามาก ในลักษณะ Breadth-first Search จึงรับประกันได้ว่า

จะพบจุดหมายที่ใกล้ที่สุดก่อน ซึ่งเราจะได้วิเคราะห์และแสดงให้เห็นว่าอัลกอริธึมยังคงคุณสมบัติของวิธีเอสตาร์ในบทต่อไป

```
global max_length, max_turn
def UAV_Path_Search( start, GOALS):
   F = {f1, f2, f3, ....}
   N = {start}
   #L is an array of every previous selected node for each goal
   L = {start, start, start, ...}
   fs = select-min(F applied to list of selected_Ni)
   while N not empty
      Let selected_N be a node from N whose fs(n) is minimum
      Let fs' be the function giving min value among fs(selected_N) & fi(selected_Ni)
      if fs' = = fs then
         if g( selected_N ) > max_length then
            Prune selected_N
         else if selected_N is gs then
            return selected_N as the best goal
         else
            N = N – {selected_N}
            N = N U expand(selected_N)
       else
          Store selected_N for gs in L
          fs = fs'
   return None

def expand( node ):
   points = {}
   for obstacle in obstacles
      pts = get_tangentvertex( obstacle, node)
      for pt in pts
         if degree(pt) < max_turn then
            points = points U { pt }
   return points
```

รูปที่ 5.5 อัลกอริธึมการวางแผนสำหรับอากาศยานไร้นักบิน

5.6 สรุป

การวางแผนเส้นทางที่มีจำนวนจุดหมายมากเป็นปัญหาในการเลือกจุดหมายที่ดีที่สุด บทนี้จึงเสนอวิธีการใหม่ในการวางแผนเส้นทางบินที่สั้นที่สุดที่มีจุดหมายจำนวนมากมายหรือไม่จำกัดสำหรับอากาศยานไร้นักบินด้วยวิธีการค้นหาเอสตาร์แบบค้นหาลึกลงไปหรือ Deepening A* Search ซึ่งสามารถค้นหาจุดหมายที่ดีที่สุดจากจุดหมายจำนวนมากมายเป็นอนันต์ได้ โดยตัดจุดหมายบางส่วนออกโดยไม่ตัดจุดหมายที่ดีที่สุดออกไป กล่าวคือ ถ้าเส้นทางที่สั้นที่สุดมีระยะทางเป็น C^* จุดหมาย i ใดๆที่มี $f_i(start)$ มากกว่า C^* จะไม่ถูกเลือกมาพิจารณาเลยทำให้สามารถตัดจุดหมายเหล่านี้ได้ นอกจากนี้อัลกอริธึมนี้ยังรับประกันได้ว่าจุดหมายที่ได้มีเส้นทางที่สั้นที่สุดโดยดัดแปลงมาจากอัลกอริธึมเอสตาร์แบบทั่วไปซึ่งจะอธิบายในบทถัดไป

บทที่ 6
การวิเคราะห์และผลการทดลอง

บทนี้ขออธิบายถึงการทำงานและแสดงประสิทธิภาพการทำงานของอัลกอริธึมเอสตาร์ทั่วไปสำหรับหนึ่งจุดหมายและจุดหมายจำนวนมาก จากนั้นเป็นการวิเคราะห์ประสิทธิภาพอัลกอริธึมเอสตาร์แบบใช้ฟังก์ชั่นฮิวริสติกที่ให้ค่าน้อยที่สุด และสุดท้ายเป็นการวิเคราะห์อัลกอริธึมเอสตาร์แบบค้นหาลึกลงไป

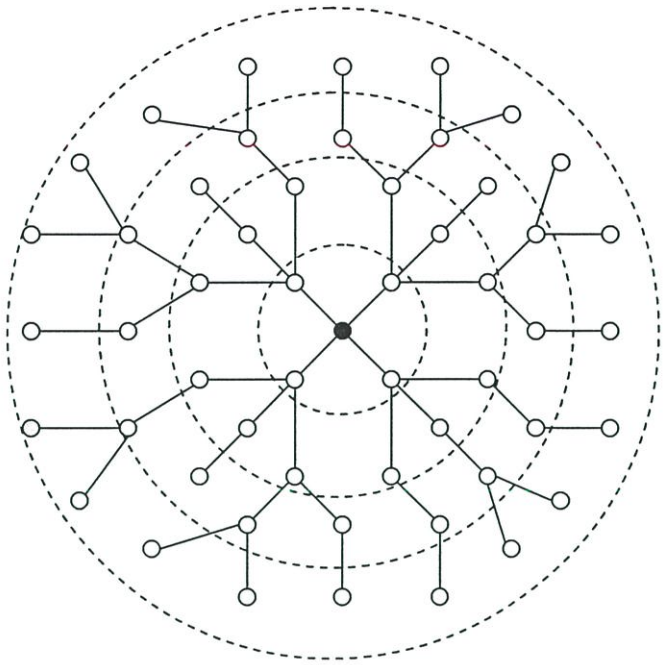
ก่อนที่จะเริ่มวิเคราะห์ประสิทธิภาพการทำงานของอัลกอริธึมในวิทยานิพนธ์ฉบับนี้ เราได้กำหนดสัญลักษณ์และเครื่องหมายต่างๆที่ใช้ในบทนี้ ดังนี้

$f(n)$	ฟังก์ชั่นสำหรับประเมินโนด n ซึ่ง $f(n) = g(n) + h(n)$
$g(n)$	ค่าใช้จ่ายจริงในการเดินทางผ่านโนดต่างจากจุดเริ่มต้นมาถึงโนด n
$h(n)$	ฟังก์ชั่นฮิวริสติกสำหรับประเมินค่าโดยประมาณของโนด n
$f_i(n)$	ฟังก์ชั่นสำหรับประเมินโนด n สำหรับเฉพาะอ้างอิงจุดหมาย i ซึ่ง $f_i(n) = g(n) + h_i(n)$
$h_i(n)$	ฟังก์ชั่นฮิวริสติกสำหรับประเมินค่าโดยประมาณของโนด n สำหรับเฉพาะอ้างอิงกับจุดหมาย i
$GOALS$	เซตของจุดหมาย
g_i	จุดหมาย i ใดๆซึ่ง $g_i \in GOALS$ และ $1 \le i \le m$ เมื่อ m คือจำนวนจุดหมายในเซต $GOALS$
b	จำนวนโนดลูกของโนดหนึ่งซึ่งเป็นค่าเฉลี่ย
d	ความลึกของเส้นทางที่สั้นที่สุด
C^*	ค่าใช้จ่ายที่น้อยที่สุดของเส้นทางที่ไปยังจุดหมาย
ε	ค่าใช้จ่ายโดยประมาณระหว่างโนดซึ่งเป็นเลขจำนวนจริงบวกที่มีค่าน้อยมาก
l	ความลึกโดยประมาณซึ่งหาได้จาก C^* / ε
$l_{\max}$	ความลึกโดยประมาณที่มีค่ามากที่สุดของสเปซการค้นหา

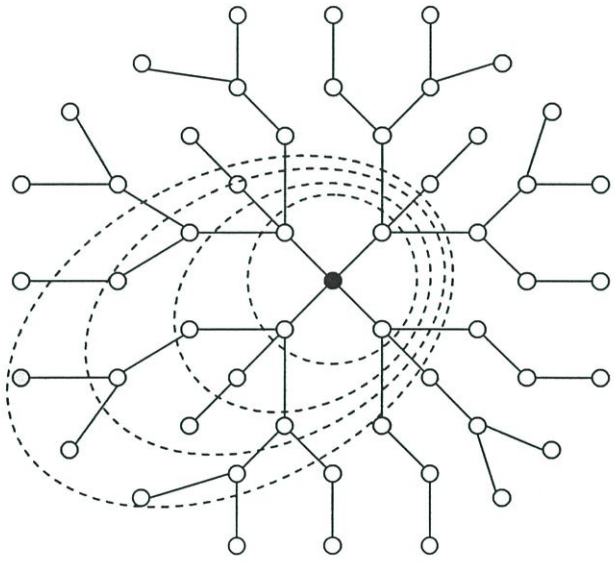
6.1 ประสิทธิภาพของอัลกอริธึมเอสตาร์ทั่วไป

งานวิจัยของ Douglas [19] ได้หาประสิทธิภาพของอัลกอริธึม Branch and Bound ซึ่งมีการทำงานที่คล้ายกับอัลกอริธึมเอสตาร์ทั่วไปโดยกำหนดให้ทรีเป็นแบบสุ่ม (Random tree) คือจำนวนโนดลูกและค่าน้ำหนักของเส้นได้จากความน่าจะเป็น จึงทำให้ค่าประสิทธิภาพอยู่ในเทอมของความ

น่าจะเป็นด้วย แต่ในวิทยานิพนธ์ฉบับนี้เลือกอธิบายการหาประสิทธิภาพตามของ Russell [8] ซึ่งเข้าใจได้ง่ายกว่าดังนี้



รูปที่ 6.1 การค้นหาตามแนวกว้าง – เส้นรอบปะแสดงระดับตามความลึก



รูปที่ 6.2 การค้นหาแบบเอสตาร์ - เส้นรอบปะมีลักษณะเอียงไปหาจุดหมาย

อัลกอริธึมเอสตาร์ทั่วไปทำงานโดยเลือกโนดที่มีค่าประเมินจาก $f(n)$ น้อยที่สุดแล้วนำมาพิจารณาว่าเป็นโนดจุดหมายหรือไม่ ถ้าไม่ใช่ โนดนี้จะขยายโนดลูกลงไปอีกขั้น และทำเช่นนี้ไปเรื่อยจนกระทั่งโนดที่เลือกมาเป็นโนดจุดหมาย ซึ่งจากการทำงานจะเห็นได้ว่าอัลกอริธึมเอสตาร์มีการทำงานคล้ายกับการค้นหาตามแนวกว้าง กล่าวคือการค้นหาตามแนวกว้างจะเลือกโนดที่มีความ

ลึกน้อยที่สุดมาพิจาณาก่อน จากรูปที่ 6.1 โนดที่อยู่ในระดับความลึกเดียวจะถูกเลือกมาพิจารณาโดยให้โนดสีดำเป็นจุดเริ่มต้น, โนดสีเทาเป็นจุดหมาย, และเส้นรอบประแสดงระดับความลึกของเส้นทางจากจุดเริ่มต้นโดยเส้นประที่อยู่ในสุดมีความลึกเป็น 1 เส้นที่อยู่นอกออกมาอีก 1 ระดับมีความลึกเป็น 2 และเส้นถัดมาจะมีความลึกเพิ่มขึ้นเรื่อยๆจนกระทั่งถึงจุดหมาย โนดที่อยู่ในความลึกเดียวกันจะถูกเลือกมาพิจารณาก่อนหรือหลังก็ได้

ส่วนการทำงานของการค้นหาแบบเอสตาร์จะเลือกโนดที่มีค่าที่ได้จากฟังก์ชั่นการประเมิน $f(n)$ จากสมการ (2.1) ดังในรูปที่ 6.2 เส้นรอบประจะมีลักษณะเอียงไปหาโนดจุดหมายเพราะโนดที่มีแนวโน้มว่าสามารถมีเส้นทางไปถึงจุดหมายได้สั้นจะถูกเลือกมาก่อน ลักษณะของเส้นรอบประคล้ายกับเส้น Contour หรือเส้นระดับความสูงเพื่อบอกระดับความสูงบนแผนที่

ก่อนที่จะวิเคราะห์การทำงานของอัลกอริธึมเอสตาร์ เราจะพูดถึงการค้นหาตามแนวกว้างหรือ Breadth-first Search ที่ทำงานค้นหาเส้นทางไปยังจุดหมายมาถึงระดับความลึก d ซึ่ง ณ ระดับความลึกนี้ โนดจุดหมายจะไม่สร้างโนดลูกทำให้ต้องใช้เวลาในการสร้างโนดลูก $b^{d+1}-b$ ดังนั้นเวลาที่ใช้ในการเส้นสร้างโนดทั้งหมดเป็น

$$b+b^2+b^3+\ldots+b^d+(b^{d+1}-b)=O(b^{d+1}) \tag{6.1}$$

และโนดที่ถูกสร้างทั้งหมดจะถูกเก็บไว้ในหน่วยความจำทำให้มีค่า Complexity ของพื้นที่หน่วยความจำจะเท่ากับเวลาที่ใช้สร้างโนดเป็น $O(b^{d+1})$

อย่างไรก็ดี การค้นหาตามแนวกว้างให้เส้นทางที่ดีที่สุด (Optimal) เมื่อค่าน้ำหนักของเส้นเชื่อมโนดทุกเส้นมีค่าเท่ากัน จึงทำให้โนดที่มีความลึกน้อยที่สุดเป็นโนดที่มีค่าใช้จ่ายน้อยที่สุดด้วย แต่ในกรณีที่ค่าน้ำหนักของเส้นเชื่อมไม่เท่ากัน จำเป็นต้องเลือกโนดที่มีค่าใช้จ่ายน้อยที่สุดแทนการเลือกโนดที่ลึกน้อยที่สุดซึ่งเหมือนกับการเลือกโนดของอัลกอริธึมเอสตาร์

การพิจารณาค่าใช้จ่ายเพื่อเลือกโนดที่มีค่าน้อยที่สุดอาจทำให้เกิดลูปขึ้นได้ ถ้ามีเส้นเชื่อมบางเส้นมีค่าเป็น 0 แล้วทำให้เลือกโนดเดิมมาพิจารณาซ้ำๆไม่รู้จบ ดังนั้นเพื่อให้สามารถรับประกันว่าสามารถทำงานได้สำเร็จ (Completeness) เส้นเชื่อมโนดแต่ละโนดจะต้องมีค่าเป็นบวกโดยสามารถแทนด้วย ε ซึ่งเป็นจำนวนบวกที่มีค่าน้อยๆ และเงื่อนไขนี้ก็จะทำให้มีคุณสมบัติที่สามารถหาเส้นทางที่ดีที่สุดได้

การค้นหาตามแนวกว้างที่มีค่าน้ำหนักไม่เท่ากันซึ่งค่าใช้จ่ายเป็นตัวกำหนดในการเลือกโนดแทนการใช้ค่าความลึก ซึ่งไม่สามารถหาค่า Complexity ที่อยู่ในรูปของ b และ d ได้เพราะความลึกของแต่ละเส้นทางที่เป็นไปได้ไม่เท่ากัน Russell [8] จึงได้กำหนดให้เส้นทางที่ดีที่สุดมีระยะทางเป็น C^* และเส้นเชื่อมโนดไม่ค่าน้อยกว่า ε แล้วจะทำให้การค้นหาแบบนี้มีค่า Complexity ของเวลาและพื้นที่ในกรณีที่แย่ที่สุดเป็น

$$O(b^{l}) \qquad \text{โดยที่ } l = C^{*}/\varepsilon \tag{6.2}$$

เมื่อ l เป็นความลึกโดยประมาณซึ่งเท่ากับ C^{*}/ε ค่า Complexity ของเวลาเท่ากับอัลกอริธึมเอสตาร์ในกรณีที่แย่ที่สุดเนื่องจากวิธีการเลือกจุดหมายเหมือนกันและ ε จะทำให้ฟังก์ชั่นฮิวริสติกมีคุณสมบัติ Monotonicity ด้วย

การหา Complexity ของอัลกอริธึมเอสตาร์ทั่วไปนั้นทำได้ยากเนื่องจากลักษณะของกราฟซึ่งมีทั้งค่าน้ำหนักของเส้นเชื่อมโนด, ตำแหน่งของจุดเริ่มต้น, และจุดหมาย และประสิทธิภาพของฟังก์ชั่นฮิวริสติกว่าสามารถช่วยให้หาเส้นทางไปยังจุดหมายได้แม่นยำเพียงใดด้วย ดังนั้นการเปรียบเทียบโค้ดการทำงานของทุกอัลกอริธึมจึงได้ปรับโค้ดในส่วนที่เหมือนกันจะแสดงในบรรทัดเดียวกัน โดยอัลกอริธึมเอสตาร์ทั่วไปแสดงในรูปที่ 6.3 บรรทัดที่ 2 จะกำหนดให้จุดเริ่มต้นเป็นโนดเริ่มต้นในการทำงาน ต่อมาบรรทัดที่ 7 และบรรทัดที่ 10 เลือกโนดที่มีค่าการประเมินน้อยที่สุดแล้วนำมาเปรียบเทียบว่าเป็นจุดหมายหรือไม่ ถ้าโนดที่เลือกมาพิจารณาไม่ใช่โนดจุดหมาย จะนำโนดที่เลือกมาออกจาก N และเพิ่มโนดลูกของโนดที่พิจารณาลงใน N ดังบรรทัดที่ 13 และ 14 ตามลำดับ

```
00 def Generic-AStar-Search(start, GOAL):
01
02    N = {start}
03
04
05
06    while N not empty
07       Let selected_N be a node from N whose f (n) is minimum
08
09
10       if selected_N is GOAL then
11          return selected_N is the best goal
12       else
13          N = N – {selected_N}
14          N = N U expand(selected_N)
15
16
17
18    return None
```

รูปที่ 6.3 อัลกอริธึมเอสตาร์ทั่วไป

6.2 ประสิทธิภาพอัลกอริธึมเอสตาร์ทั่วไปที่ใช้กับจุดหมายจำนวนมาก

เมื่อมีจุดหมายจำนวนมากให้เลือกว่าจุดหมายใดสามารถให้เส้นทางที่ดีที่สุดและถ้าเรามีเครื่องคอมพิวเตอร์ประสิทธิภาพสูงมากๆ เราสามารถกำหนดให้มีการค้นหาเส้นทางของแต่ละจุดหมายทำงานครั้งละจุดหมายดังรูปที่ 5.1 จนกระทั่งได้เส้นทางแล้วจะนำมาเปรียบเทียบกับเส้นทางของจุดหมายอื่นจนครบทุกจุดหมาย เราจะได้จุดหมายที่ดีที่สุดซึ่งมีเส้นทางสั้นที่สุด เวลาใน

การประมวลผลเกิดจากผลรวมของการค้นหาเส้นทางในแต่ละจุดหมายซึ่งมีระดับความลึกแต่ต่างกัน

$$b^l + b^{l_1} + b^{l_2} + \ldots + b^{l_{\max}} \tag{6.3}$$

จุดหมายใดๆมีระดับความลึกน้อยที่สุดคือ b^l และจุดหมายใดๆที่มีระดับความลึกมากที่สุดในสเปซการค้นหาคือ $b^{l_{\max}}$ ซึ่ง $l < l_1 < l_2 < \ldots < l_{\max}$ ถ้าสมมุติให้แต่ละจุดหมายอยู่ห่างกันเพียง 1 ระดับความลึก

$$\begin{aligned} b^l + b^{l_1} + b^{l_2} + \ldots + b^{l_{\max}} &= b^l + b^{l+1} + b^{l+2} + \ldots + b^{l_{\max}} \\ &= \frac{b(b^{l_{\max}+1} - 1)}{b-1} - \frac{b(b^l - 1)}{b-1} \\ &= \frac{b}{b-1}\left[(b^{l_{\max}+1} - 1) - (b^l - 1)\right] \\ &= \frac{b}{b-1}\left[b^{l_{\max}+1} - b^l\right] \\ &= O(b^{l_{\max}+1}) \end{aligned} \tag{6.4}$$

แต่ในความเป็นจริงแล้ว เราไม่สามารถหาเครื่องคอมพิวเตอร์ประสิทธิภาพสูงมาทำงานค้นหาเส้นทางทั้งหมดได้ และยิ่งไปกว่านั้นถ้าจำนวนจุดหมายเป็นอนันต์ จำนวนโนดก็เป็นอนันต์ด้วยเช่นกันซึ่งเครื่องคอมพิวเตอร์นี้ต้องมีหน่วยความจำที่ใหญ่มากเป็นอนันต์หรือขนาดของหน่วยความจำเท่ากับ $O(b^{l_{\max}+1})$ ดังนั้นในหัวข้อต่อไปเราจึงได้ศึกษาการลดขนาดหน่วยความจำด้วยวิธีการของ Russell [8] ซึ่งใช้การค้นหาเส้นทางใช้หน่วยความจำร่วมกันแต่ได้ใช้ฟังก์ชั่นประเมินที่ต่างออกไป จากนั้นจะนำมาเปรียบเทียบอัลกอริธึมการค้นหาเอสตาร์แบบทั่วไป

6.3 ประสิทธิภาพอัลกอริธึมเอสตาร์แบบใช้ฟังก์ชั่นฮิวริสติกที่ให้ค่าน้อยที่สุด

อัลกอริธึมเอสตาร์แบบใช้ฟังก์ชั่นฮิวริสติกที่มีค่าน้อยที่สุดเป็นวิธีการของ Russell [8] ซึ่งได้เสนอการปรับเปลี่ยนฟังก์ชั่นประเมินสำหรับจุดหมายเดียวให้สามารถทำงานได้ในกรณีหลายจุดหมาย ด้วยวิธีการนี้สามารถใช้หน่วยความจำร่วมกันได้จึงเหลือการใช้หน่วยความจำเพียงแค่ $O(b^{l+1})$ เท่านั้น

การวิเคราะห์ประสิทธิภาพของอัลกอริธึมเอสตาร์ทั่วไปหาได้จากการวิเคราะห์ว่าจำนวนครั้งที่สร้างโนดลูกมีมากเพียงใด แต่อัลกอริธึมเอสตาร์แบบเลือกใช้ฟังก์ชั่นฮิวริสติกที่ให้ค่าน้อยที่สุดมีส่วนที่เพิ่มเติมเนื่องจากจำนวนจุดหมายที่มากขึ้น ดังนั้นการวิเคราะห์ประสิทธิภาพจะวิเคราะห์เฉพาะส่วนที่เพิ่มเติมมาจากอัลกอริธึมเอสตาร์ทั่วไป

การทำงานของอัลกอริธึมเอสตาร์แบบเลือกใช้ฟังก์ชั่นฮิวริสติกที่ให้ค่าน้อยที่สุดเป็นอัลกอริธึมที่เพิ่มเติมจากอัลกอริธึมเอสตาร์ทั่วไปโดยปรับเปลี่ยนฟังก์ชั่นประเมินในส่วนของฟังก์ชั่นฮิวริสติกจากสมการ (3.1) ซึ่งเขียนเป็นอัลกอริธึมใหม่ได้ดังรูปที่ 6.4 บรรทัดที่ 7 และ 10

```
00 def Multigoal-AStar-Search(start, GOALS):
01
02    N = {start}
03
04
05
06    while N not empty
07       Let selected_N be a node from N whose f (n) is minimum
08
09
10       if selected_N is a goal in GOALS then
11          return selected_N is the best goal
12       else
13          N = N – {selected_N}
14          N = N U expand(selected_N)
15
16
17
18    return None
```

รูปที่ 6.4 อัลกอริธึมเอสตาร์แบบใช้ฟังก์ชั่นฮิวริสติกที่มีค่าน้อยที่สุด

เริ่มจากบรรทัดที่ 7 เป็นการเลือกโนดที่มีค่าน้อยที่สุดโดยใช้ฟังก์ชั่นฮิวริสติกโดยเลือกค่าที่ได้จากฟังก์ชั่นฮิวริสติกย่อยที่มีค่าน้อยที่สุดจากค่าที่ได้จากฟังก์ชั่นฮิวริสติกย่อยจากโนด n ไปยังจุดหมาย G_1, ไปยังจุดหมาย G_2, ไปยังจุดหมาย G_3, จนกระทั่งไปถึงค่าที่ได้จาก G_m เมื่อจุดหมายทั้งหมดมี m จุดหมาย จึงได้สมการ (6.4) ดังนั้นค่าประเมินของแต่ละโนดต้องมีการคำนวณฟังก์ชั่นฮิวริสติกย่อยของแต่ละจุดหมายเป็นจำนวน m ครั้ง

$$h(n) = \min(h_1(n), h_2(n), h_3(n), ..., h_{m-1}(n), h_m(n)) \tag{6.5}$$

ถ้ากำหนดให้ G_i เป็นจุดหมายที่มีเส้นทางที่สั้นที่สุดจากจุดหมายทั้งหมดหรือมีระยะของเส้นทางจากจุดเริ่มต้นที่สั้นที่สุดเป็น C^* ดังนั้นโนด n_x ใดๆที่อยู่ระหว่างเส้นทางจากจุดเริ่มต้นไปยัง G_i และ $f(n_x) = g(n_x) + h(n_x) < C^*$ โดยที่ $h(n_x) = h_i(n_x)$ ซึ่งมีค่าน้อยที่สุด และโนด n_y ใดๆซึ่ง $h_i(n_x)$ ไม่ใช้ค่าน้อยที่สุดหรือ $h(n_x) \neq h_i(n_x)$ จากฟังชั่นฮิวริสติกของแต่ละจุดหมาย แต่ $f(n_y) = g(n_y) + h(n_y) < C^*$ ดังนั้นจำนวนโนด n_x จะเท่ากับ $b(b^{l+1} - 1)/(b-1)$ แต่จำนวนโนด n_y ไม่สามารถหาได้อย่างแน่ชัด เราจึงกำหนดให้มีจำนวนเป็น y โนด

โนดที่สร้างขึ้นด้วยอัลกอริธึมเอสตาร์แบบเลือกใช้ฟังก์ชั่นฮิวริสติกที่ให้ค่าน้อยที่สุดมีจำนวนมากกว่าอัลกอริธึมเอสตาร์ทั่วไปคือมีมากกว่า y โนด แต่น้อยกว่าอัลกอริธึมเอสตาร์ทั่วไปที่ใช้กับจำนวนจุดหมายจำนวนมากอยู่มาก

การเปลี่ยนแปลงฟังก์ชั่นฮิวริสติกในอัลกอริธึมนี้ทำให้มีการประมวลผลเพิ่มขึ้นมา 2 ส่วน คือ เวลาที่ใช้คำนวณฟังก์ชั่นฮิวริสติกซึ่งได้จากการคำนวณฟังก์ชั่นฮิวริสติกย่อย และเวลาที่ใช้ในการเปรียบเทียบว่าโนดที่เลือกมาเป็นจุดหมายใดจุดหมายหนึ่งจากจุดหมายทั้งหมดหรือไม่

การคำนวณฟังก์ชั่นฮิวริสติกจากสมการ (6.5) ต้องคำนวณกับทุกโนดลูกที่สร้างขึ้นมาเพื่อใช้ประเมินโนดก่อนจะนำมาเปรียบเทียบหาโนดที่มีค่าประเมินน้อยที่สุด แสดงว่าต้องคำนวณฟังก์ชั่นฮิวริสติกย่อยจำนวน m ครั้งต่อโนดใหม่ 1 โนดที่สร้างขึ้นซึ่งใช้เวลา $t_h m$ โดยให้ t_h เป็นเวลาที่ใช้คำนวณฟังก์ชั่นฮิวริสติกย่อย 1 ครั้ง ดังนั้นเราสามารถหาเวลาใช้คำนวณฟังก์ชั่นฮิวริสติกได้เป็น

$$\begin{aligned}
&t_h m + t_h mb + t_h mb^2 + \ldots + t_h mb^l + t_h m(b^{l+1} - b) + t_h my \\
&\quad = t_h m[1 + b + b^2 + \ldots + b^l] + t_h m(b^{l+1} - b) + t_h my \\
&\quad = t_h m\sum_{i=0}^{l} b^i + t_h m(b^{l+1} - b)] + t_h my \\
&\quad = t_h m\left[\frac{b(b^{l+1} - 1)}{(b-1)}\right] + t_h m(b^{l+1} - b) + t_h my \\
&\quad = t_h m\left[\frac{b(b^{l+1} - 1)}{(b-1)} + (b^{l+1} - b) + y\right] \qquad (6.6)
\end{aligned}$$

การเปรียบเทียบว่าโนดที่เลือกมาเป็นจุดหมายหรือไม่ต้องทำงานมากขึ้นเช่นกันเมื่อจำนวนจุดหมายมากขึ้น โนดที่มีค่าประเมินน้อยกว่า C^* จะถูกเลือกมาพิจารณาและต้องนำเปรียบเทียบกับจุดหมายทั้งหมด m ครั้ง ดังนั้นเวลาที่ใช้ในการเปรียบเทียบของแต่ละโนดเป็น $t_g m$ ซึ่ง t_g เป็นเวลาที่ใช้เปรียบเทียบกับ 1 จุดหมาย ดังนั้นเวลาที่ใช้ทั้งหมด สามารถเขียนได้เป็น

$$\begin{aligned}
t_g m + t_g mb + t_g mb^2 + \ldots + t_g mb^l + t_h my &= t_g m[1 + b + b^2 + \ldots + b^l + y] \\
&= t_g m\left[\sum_{i=o}^{l} b^i + y\right] \\
&= t_g m\left[\frac{b(b^{l+1} - 1)}{(b-1)} + y\right] \qquad (6.7)
\end{aligned}$$

เวลาในการประมวลผลทั้งหมดของอัลกอริธึมได้จากการรวมเวลาที่ใช้ในการคำนวณฟังก์ชั่นฮิวริสติกย่อยและเวลาที่ใช้ในการเปรียบเทียบจุดหมายหรือรวมสมการ (6.6) และ (6.7) ตามลำดับจึงได้เป็นสมการ (6.8) และถ้าจำนวนโนด y มีค่าไม่มากนัก เราสามารถหาค่า Complexity โดยให้ t_h และ t_g เป็นค่าคงที่จะได้เป็น $O(mb^{l+1})$ ดังสมการ (6.9)

$$(6.6)+(6.7)= t_h m\left[\frac{b(b^{l+1}-1)}{(b-1)}+(b^{l+1}-b)+y\right]+t_g m\left[\frac{b(b^{l+1}-1)}{(b-1)}+y\right]$$

$$=(t_h+t_g)m\left[\frac{b(b^{l+1}-1)}{(b-1)}+y\right]+t_h m(b^{l+1}-b) \quad (6.8)$$

$$=O(mb^{l+1}) \quad (6.9)$$

ส่วนค่า Complexity ของการใช้หน่วยความจำได้เท่ากับการใช้หน่วยความจำของการค้นหาตามแนวกว้างคือ $O(b^{l+1})$

การทำงานของอัลกอริธึมเอสตาร์แบบใช้ฟังก์ชั่นฮิวริสติกที่มีค่าน้อยที่สุดใช้เวลาในการทำงานน้อยกว่าอัลกอริธึมเอสตาร์แบบทั่วไปสำหรับจุดหมายจำนวนมากซึ่งขึ้นอยู่จำนวน y โนด และยังสามารถลดพื้นที่หน่วยความจำลงได้เหลือเพียง $O(b^{l+1})$ ซึ่งมากกว่าอัลกอริธึมเอสตาร์แบบทั่วไปอยู่เท่ากับจำนวน y โนด <u>*อัลกอริธึมทั้งหมดที่ได้กล่าวมานี้ทำงานได้เมื่อจำนวนจุดหมายมีจำนวนอนันต์เท่านั่น*</u> ดังนั้นในวิทยานิพนธ์ฉบับนี้จึงได้เสนออัลกอริธึมเอสตาร์แบบค้นหาลึกลงไปที่กล่าวมาแล้วในบทที่ผ่านมาที่สามารถทำงานได้เมื่อจุดหมายเป็นจำนวนอนันต์ และในหัวข้อต่อไป เราได้วิเคราะห์ประสิทธิภาพการทำงานของอัลกอริธึมเอสตาร์แบบค้นหาลึกลงไป

6.4 ประสิทธิภาพอัลกอริธึมเอสตาร์แบบค้นหาลึกลงไป

เราจะสามารถเปรียบเทียบกับอัลกอริธึมอื่นได้เมื่อจุดหมายมีจำนวนจำกัดเท่านั่น เนื่องจากอัลกอริธึมเอสตาร์แบบใช้ฟังก์ชั่นฮิวริสติกที่มีค่าน้อยที่สุดและอัลกอริธึมเอสตาร์แบบทั่วไปสำหรับจุดหมายจำนวนมากจะสามารถทำงานได้เมื่อทราบจำนวนจุดหมายหรือไม่สามารถทำงานเมื่อมีจุดหมายจำนวนอนันต์ ดังนั้นเราจึงได้วิเคราะห์ประสิทธิภาพการทำงานของอัลกอริธึมเอสตาร์แบบค้นหาลึกลงไปเมื่อจุดหมายเป็นจำนวนจำกัดว่ามีประสิทธิภาพเพียงใดเมื่อเทียบกับอัลกอริธึมอื่น

เราใช้รูปที่ 6.5 ประกอบการวิเคราะห์ประสิทธิภาพการทำงานของอัลกอริธึมเอสตาร์แบบค้นหาลึกลงไป อัลกอริธึมนี้มีลักษณะการทำงานซึ่งได้ปรับปรุงมาจากอัลกอริธึมเอสตาร์แบบทั่วไปสำหรับจุดหมายจำนวนมาก เราสามารถเขียนอธิบายการทำงานได้ดังนี้

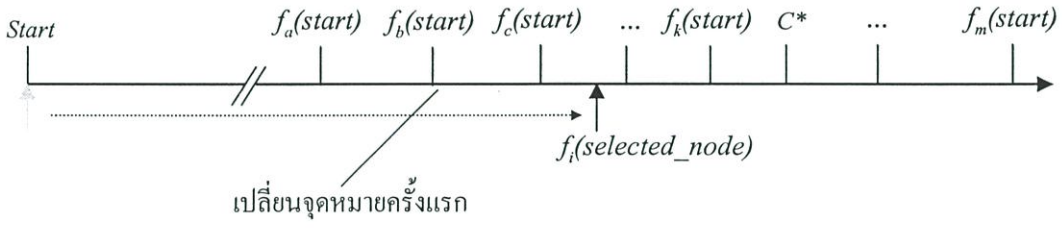
เริ่มจากบรรทัดที่ 5 เลือกฟังก์ชั่นประเมินของจุดหมายที่มีค่าประเมินจุดเริ่มต้นที่น้อยที่สุดและจากนั้นการนำมาประเมินโนดที่มีค่าน้อยที่สุดได้เป็นโนด *selected _ N* ดังบรรทัดที่ 7 เมื่อได้โนดมาแล้วจะนำมาตรวจสอบว่าควรเปลี่ยนจุดหมายในการพิจาณาหรือไม่ซึ่งในการทำงานรอบแรกจะไม่มีเปลี่ยนจุดหมาย ถ้าโนด *selected _ N* เป็นจุดหมายจะทำในบรรทัดที่ 11 คืออัลกอริธึมจะหยุดทำงานแล้วได้ผลลัพธ์เป็นโนด *selected _ N* แต่ถ้าไม่ใช่จุดหมายจะนำมาหาโนดลูกดังบรรทัดที่ 13 และ 14

```
00 def Deepening-AStar-Search(start, GOALS):
01    F = {f1, f2, f3, ...., fm}
02    N = {start}
03    #L is array of every previous selected node for each goal
04    L = {start, start, start, …, start}
05    fs = select-min(F applied to list of selected_Ni)
06    while N not empty
07       Let selected_N be a node from N whose fs(n) is minimum
08       Let fs' be the function giving min value among fs(selected_N) & fi(selected_Ni)
09       if fs' = = fs then
10          if selected_N is gs then
11             return selected_N is the best goal
12          else
13             N = N – {selected_N}
14             N = N U expand(selected_N)
15       else
16          Store selected_N for gs in L
17          fs = fs'
18    return None
```

รูปที่ 6.5 อัลกอริธึมเอสตาร์แบบค้นหาลึกลงไป

อัลกอริธึมจะทำงานไปเรื่อยๆ ถ้าจุดหมายที่ถูกเลือกนำมาประเมินกับโนด *selected _ N* แล้วมีค่าน้อยกว่าค่าประเมินของจุดหมายอื่น ต้องทำการเปลี่ยนจุดหมายใหม่โดยจะเก็บโนด *selected _ N* ไว้สำหรับจุดหมายเดิมไว้และเปลี่ยนฟังก์ชั่นประเมินของจุดหมายใหม่ดังบรรทัดที่ 16 และ 17 ตามลำดับ

จากอัลกอริธึมในรูปที่ 6.5 เราสามารถนำลักษณะการเพิ่มขึ้นของค่าประเมินโนดมาเขียนให้อยู่ในรูปของเส้นระยะทางจากจุดเริ่มต้นดังรูปที่ 6.6 โดยจุดที่อยู่ใกล้แสดงว่ามีระยะทางสั้นกว่าจุดที่อยู่ไกลออกไป ฟังก์ชั่นประเมินของจุดหมายต่างๆประเมินโนดเริ่มต้น (Start) ได้ค่าที่ต่างกัน โดยกำหนดให้ a, b, c, ..., m เป็นจุดหมายในเซตของจุดหมายและ $f_a(start) < f_b(start)$, $f_b(start) < f_c(start)$, ... ซึ่งจุดหมาย i ใดๆที่ $f_i(start) > C^*$ จะไม่ถูกเลือกมาพิจารณาเลย เนื่องจากค่าประมาณของจุดเริ่มต้นสำหรับจุดหมาย i ยังมีค่ามากกว่าแล้วค่าประมาณของโนดใดๆ สำหรับจุดหมาย i ย่อมมีค่ามากกว่า



รูปที่ 6.6 เส้นระยะทาง

ค่าประเมินของโนดที่ถูกเลือกเข้ามาในแต่ละครั้งจะเพิ่มขึ้นเรื่อยๆ จากรูปที่ 6.6 ถ้า $f_a(seleted\_node) > f_b(start)$ หรือค่าประเมินสำหรับจุดหมาย a มีค่ามากกว่าค่าประเมินสำหรับจุดหมาย b อัลกอริธึมจำเป็นต้องมีการเปลี่ยนจุดหมาย b มาพิจารณาแทนและเปลี่ยนฟังก์ชั่นประเมิน f_a ด้วยฟังก์ชั่นประเมินของจุดหมายที่เปลี่ยนใหม่ f_b ด้วย อัลกอริธึมจะทำงานเช่นนี้ไปเรื่อยๆจนกระทั่งโนดที่เลือกมาเป็นโนดจุดหมายซึ่งจะได้เส้นทางที่สั้นที่สุดมีระยะทางเป็น C^*

วิธีการหาจำนวนโนดของอัลกอริธึมเอสตาร์แบบค้นหาลึกลงไปจะคล้ายกับอัลกอริธึมเอสตาร์ทั่วไปที่ใช้ฟังก์ชั่นฮิวริวติกที่ให้ค่าน้อยที่สุด กล่าวคือ ถ้ากำหนดให้ G_i เป็นจุดหมายที่มีเส้นทางที่สั้นที่สุดจากจุดหมายทั้งหมดหรือมีระยะของเส้นทางจากจุดเริ่มต้นที่สั้นที่สุดเป็น C^* ดังนั้นโนด n_x ใดๆที่อยู่ระหว่างเส้นทางจากจุดเริ่มต้นไปยัง G_i และ $f_i(n_x) = g(n_x) + h_i(n_x) < C^*$ และโนด n_z ใดๆที่ $f_j(n_z) = g(n_z) + h_j(n_z) < C^*$ โดยที่จุดหมาย j ไม่ใช่จุดหมาย i ดังนั้นจำนวนโนด n_x จะเท่ากับ $b(b^{l+1} - 1)/(b-1)$ แต่จำนวนโนด n_z ไม่สามารถหาได้อย่างแน่ชัด เราจึงกำหนดให้มีจำนวนเป็น z โนด

โนดที่สร้างขึ้นด้วยอัลกอริธึมเอสตาร์แบบค้นหาลึกลงไปมีค่าน้อยกว่าอัลกอริธึมเอสตาร์แบบเลือกใช้ฟังก์ชั่นฮิวริสติกที่ให้ค่าน้อยที่สุดหรือ $z < y$ เนื่องจากจุดหมายบางส่วนไม่ถูกเลือกมาพิจารณาเลย หรือมีจำนวนโนดเท่ากันคือ $z = y$ เมื่อจุดหมายถูกเลือกมาพิจารณาทั้งหมด และทำให้มีจำนวนโนดน้อยกว่าอัลกอริธึมเอสตาร์ทั่วไปที่ใช้กับจำนวนจุดหมายจำนวนมากด้วยเช่นกัน

การวิเคราะห์ประสิทธิภาพการทำงานของอัลกอริธึมทำโดยการพิจารณาการทำงานในส่วนที่เพิ่มเติมจากอัลกอริธึมเอสตาร์แบบทั่วไปซึ่งประกอบด้วยการเปรียบเทียบกับจุดหมาย, การคำนวณค่าประเมินจากฟังก์ชั่นฮิวริสติกย่อย, และการเปลี่ยนจุดหมายในการประเมินโนดที่เลือกมา

เริ่มพิจารณาด้วยการเปรียบเทียบกับจุดหมาย อัลกอริธึมจะเปรียบเทียบโนดที่เลือกมากับจุดหมายเพื่อตรวจสอบว่าเป็นจุดหมายหรือไม่ ขั้นตอนนี้จะเปรียบเทียบกับจุดหมายที่กำลังพิจารณาเท่านั่น ทำให้แต่ละโนดที่เลือกมาถูกนำมาเปรียบเทียบเพียงครั้งเดียว ดังนั้นเราสามารถหาเวลาที่ใช้เปรียบเทียบทั้งหมดโดยให้ t_g เป็นเวลาที่ใช้เปรียบเทียบจุดหมาย จึงเขียนได้เป็น

$$\begin{aligned} t_g + t_g b + t_g b^2 + \ldots + t_g b^l + t_g z &= t_g[1 + b + b^2 + \ldots + b^l + z] \\ &= t_g\left[\sum_{i=o}^{l} b^i + z\right] \\ &= t_g\left[\frac{b(b^{l+1} - 1)}{(b-1)} + z\right] \end{aligned} \tag{6.10}$$

เมื่อนำสมการ (6.7) ของอัลกอริธึมเอสตาร์แบบใช้ฟังก์ชั่นฮิวริสติกที่มีค่าน้อยที่สุดมาเปรียบเทียบกับสมการ (6.10) จะเห็นได้ว่าเวลาของสมการ (6.10) ไม่ขึ้นอยู่กับจำนวนจุดหมาย

ต่อมาเราจะพิจารณาในส่วนการคำนวณฟังก์ชั่นฮิวริสติกย่อย แต่ละโนดต้องคำนวณฟังก์ชั่นฮิวริสติกย่อยอย่างน้อยหนึ่งครั้งกับจุดหมายที่กำลังพิจารณา ซึ่งใช้เวลาเป็น

$$t_h + t_h b + t_h b^2 + \ldots + t_h b^l + t_h(b^l - b) + t_h z = t_h[1 + b + b^2 + \ldots + b^l + (b^l - b) + z]$$
$$= t_h\left[\frac{b(b^{l+1} - 1)}{(b-1)} + (b^{l+1} - b) + z\right] \qquad (6.11)$$

และเมื่อมีการเปลี่ยนจุดหมายในการพิจารณา เราต้องประมวลผลใหม่เพื่อให้สามารถเลือกน้อยที่มีค่าประเมินน้อยที่สุดและสมมุติให้ z มีค่าน้อยซึ่งทำให้ไม่มีผลต่อการประมวลผลใหม่ ดังนั้นเราสามารถกำหนดโครงสร้างข้อมูลได้ออกเป็น

- <u>การใช้ลิสต์เก็บข้อมูลโนดร่วมกัน</u> เมื่อมีการเปลี่ยนจุดหมายจำเป็นต้องมีการคำนวณฟังก์ชั่นฮิวริสติกย่อยใหม่และจัดเรียงโนดใน N ใหม่ทุกครั้ง ดังนั้นเวลาในการเปลี่ยนจุดหมายประกอบด้วย 2 ส่วน คือ

S_j เป็นเวลาที่ใช้ในการคำนวณฟังก์ชั่นฮิวริสติกย่อยใหม่อีกครั้งเมื่อเปลี่ยนจุดหมายซึ่งขึ้นอยู่กับจำนวนโนดทั้งหมดใน N ด้วย แต่ละรอบการทำงานมีโนดที่ถูกนำออกจาก N ครั้งละ 1 โนดและเพิ่มเข้าไป b โนด ดังนั้นจำนวนโนดจะเพิ่มขึ้นรอบละ $b-1$ โนด จึงทำให้เวลาที่ใช้ในการคำนวณฟังก์ชั่นฮิวริสติกย่อยใหม่ซึ่งมีจำนวนครั้งในการทำงานทั้งหมด b^l ครั้งและการเปลี่ยนจุดหมายที่รอบการทำงาน j ใดๆใช้เวลาเป็น $t_g j(b-1)$ และใน N มีโนดจำนวน $j(b-1)$ ซึ่ง $1 \le j \le b^l$ จึงเขียนสมการได้เป็น

$$S_j = t_h j(b-1) \qquad (6.12)$$

T_j เป็นเวลาที่ใช้ในจัดเรียงโนดใหม่เพื่อให้การเลือกโนดที่มีค่าประเมินน้อยที่สุดสามารถทำได้อย่างรวดเร็ว ต้องจัดเรียงโนดใหม่ตามลำดับจากน้อยไปหามากดังนั้นเวลาที่ใช้ในจัดเรียงโนดใหม่ เมื่อ $sort(j(b-1))$ เป็นฟังก์ชั่นที่ใช้จัดเรียงโนดจำนวน $j(b-1)$ โนด เราเขียนสมการได้เป็น

$$T_j = sort(j(b-1)) \qquad (6.13)$$

กำหนดให้ ST เป็นเซตของการเปลี่ยนจุดหมายทั้งหมด โดยให้ j เป็นรอบการทำงานใดๆ และการเปลี่ยนจุดหมายแต่ละครั้งประกอบด้วยเวลาการเปลี่ยนจุดหมายและเวลาเรียงโนดใหม่

$$ST = \{S_j + T_j \mid 1 \le j \le b^l\} \qquad (6.14)$$

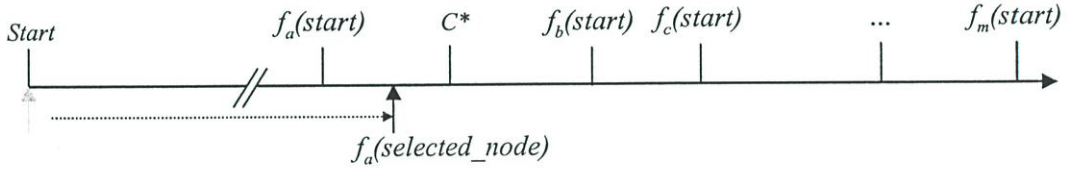
การเปลี่ยนจุดหมาย TC ของการทำงานของอัลกอริธึมเป็นซับเซตของการเปลี่ยนจุดหมายทั้งหมดหรือ $TC \subset ST$ ดังนั้นเวลาในการเปลี่ยนจุดหมายจึงเป็นผลรวมของสมาชิกใน TC เราจึงเขียนสมการได้เป็น

$$tc = \sum_{i \in TC} i \tag{6.15}$$

ดังนั้นประสิทธิภาพการทำงานของอัลกอริธึมเอสตาร์แบบค้นหาลึกลงไปเขียนได้เป็น

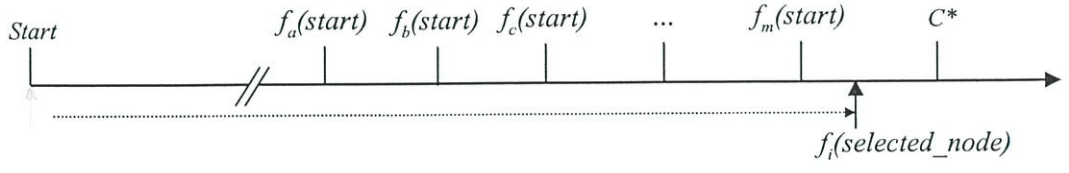
$$\begin{aligned}(6.10)+(6.11)+(6.15) &= t_g\left[\frac{b(b^{l+1}-1)}{(b-1)}+z\right]+t_h\left[\frac{b(b^{l+1}-1)}{(b-1)}+(b^{l+1}-b)+z\right]+tc \\ &= (t_h+t_g)\left[\frac{b(b^{l+1}-1)}{(b-1)}+z\right]+t_h(b^{l+1}-b)+tc \end{aligned} \tag{6.16}$$

เมื่อเราพิจารณาการทำงานของอัลกอริธึมในกรณีที่ดีที่สุดเมื่อจุดหมายแรกที่เลือกมาพิจาณาเป็นจุดหมายที่ดีที่สุด จะทำให้จุดหมาย a ที่มี $f_a(start)$ น้อยที่สุดจะถูกเลือกมาพิจารณาจึงทำให้อัลกอริธึมไม่มีการเปลี่ยนจุดหมายในการพิจารณาเลย ดังรูปที่ 6.7 และเวลาที่ใช้ในการทำงานจะไม่มีส่วนของเวลาในการเปลี่ยนจุดหมาย หรือจากสมการ (6.16) ที่ได้ตัด tc ออก



รูปที่ 6.7 เส้นระยะทางสำหรับกรณีที่ดีที่สุด

ส่วนในกรณีที่แย่ที่สุดจะเกิดขึ้นเมื่อ $C*$ มีค่ามากกว่าค่าประเมินของจุดเริ่มต้นในทุกๆ จุดหมาย ดังรูปที่ 6.8 แล้วทำให้จุดหมายทั้งหมดถูกนำมาพิจารณาอย่างน้อยหนึ่งครั้ง และบางจุดหมายอาจถูกเลือกมาพิจารณาหลายครั้ง



รูปที่ 6.8 เส้นระยะทางสำหรับกรณีที่แย่ที่สุด

ดังนั้นในกรณีแย่ที่สุดของอัลกอริธึม เราจึงกำหนดให้มีการเปลี่ยนจุดหมายทุกครั้งหลังจากสร้างโนดลูกซึ่ง $TC = ST$

$$
\begin{aligned}
tc = ST &= \sum_{i=0}^{b^l} S_i + \sum_{i=0}^{b^l} T_i \\
&= \sum_{i=0}^{b^l} t_h i(b-1) + \sum_{i=0}^{b^l} sort(i(b-1)) \\
&= t_h(b-1)\sum_{i=0}^{b^l} i + \sum_{i=0}^{b^l} sort(i(b-1)) \\
&= t_h(b-1)\frac{b^l(b^l-1)}{2} + \sum_{i=0}^{b^l} sort(i(b-1)) \\
&= t_h(b-1)\frac{(b^{2l}-b^l)}{2} + \sum_{i=0}^{b^l} sort(i(b-1))
\end{aligned}
\tag{6.17}
$$

เมื่อพิจารณาในสมการ (6.17) เวลาในการเปลี่ยนจุดหมายมีพจน์ที่มีค่ายกกำลังมากที่สุดคือ b^{2l} ซึ่งทำให้ประสิทธิภาพการทำงานของอัลกอริธึมในกรณีที่แย่ที่สุดจะขึ้นอยู่กับความลึกของเส้นทาง ดังนั้นเราสามารถสรุปได้ว่าการเปลี่ยนจุดหมายส่งผลต่อประสิทธิภาพของอัลกอริธึมเมื่อในการทำงานค้นหาเส้นทางมีการเปลี่ยนจุดหมายมากครั้งจะทำให้ใช้เวลาในการทำงานมากขึ้นด้วย

- <u>การใช้ลิสต์เก็บข้อมูลโนดหลายลิสต์สำหรับจุดหมายที่ถูกเปลี่ยนบ่อย</u> ด้วยวิธีการทำให้การวางแผนเส้นทางที่มีจุดหมายจำนวนมากที่ใช้ลิสต์เก็บข้อมูลร่วมกันใช้พื้นที่หน่วยความจำน้อย แต่จะใช้เวลาในการประมวลผลมากถ้าเส้นทางที่ดีที่สุดมีความลึกมาก เราสามารถแก้ปัญหานี้ด้วยการปรับปรุงการสร้างลิสต์สำหรับเก็บโนดของแต่ละจุดหมายเมื่อถูกเลือกเพื่อลดการจัดเรียงโนดใหม่ ถ้าจุดหมายทั้งหมดถูกเลือกมาพิจารณาจะทำให้พื้นที่หน่วยความจำเท่ากับการวางแผนเส้นทางแบบขนาน ดังนั้นเราจึงเลือกสร้างลิสต์เก็บโนดเฉพาะจุดหมายที่ถูกเลือกบ่อยๆ (Most recent used) ข้อดีของวิธีการนี้คือลดเวลาในการจัดเรียงโนดใหม่อันเนื่องจากปัญหาที่เกิดจากการใช้ลิสต์ข้อมูลร่วมกัน แต่ส่งผลให้พื้นที่หน่วยความจำมากขึ้นตามจำนวนลิสต์ที่สร้าง ผลการทดลองในหัวข้อถัดไปจะเปรียบเทียบการใช้โครงสร้างข้อมูลทั้งสองแบบด้วย

6.5 ผลการทดลอง

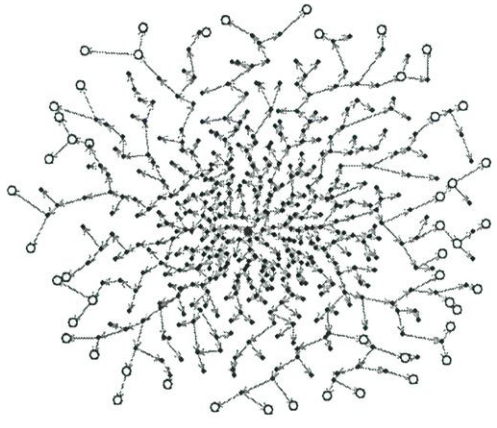
อัลกอริธึมเอสตาร์แบบค้นหาลึกลงไปและอัลกอริธึมเอสตาร์แบบใช้ฟังก์ชั่นฮิวริสติกที่มีค่าน้อยที่สุดมีคุณสมบัติ Optimal ของอัลกอริธึมเอสตาร์เหมือนกัน คือ สามารถหาเส้นทางที่ดีที่สุดได้ เนื่องจากโนดที่ถูกเลือกมาพิจารณาทั้งหมดมีค่า $f_i(n) < C^*$ ดังนั้นในการทดลองเพื่อเปรียบเทียบการทำงานทั้งสองอัลกอริธึม เราพิจารณาในส่วนที่เพิ่มเติมจากอัลกอริธึมเอสตาร์ทั่วไปคือ การเปรียบเทียบจุดหมาย, การตรวจสอบว่าโนดเป็นจุดหมายหรือไม่, และการเปลี่ยนจุดหมายซึ่งมีใน

อัลกอริธึมเอสตาร์แบบค้นหาลึกลงไปและทดลองกับการใช้โครงสร้างข้อมูลทั้งสองแบบ โดยการทดลองเริ่มจากเปรียบเทียบอัลกอริธึมเอสตาร์แบบใช้ฟังก์ชั่นฮิวริสติกที่มีค่าน้อยที่สุดกับอัลกอริธึมเอสตาร์แบบค้นหาลึกลงไปโดยใช้ลิสต์ร่วมกัน จากนั้นจะเปรียบเทียบระหว่างอัลกอริธึมเอสตาร์แบบค้นหาลึกลงไปแบบใช้ลิสต์ร่วมกันกับแบบใช้ลิสต์เก็บข้อมูลโนดหลายลิสต์สำหรับจุดหมายที่ถูกเปลี่ยนบ่อย

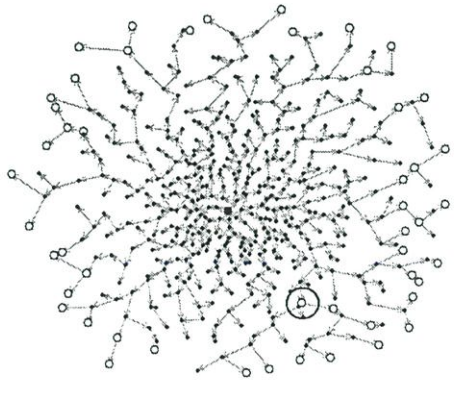
การทดลองแบ่งออกเป็น 2 ส่วนคือ การทดลองกับทรี และการทดลองวางแผนเส้นทางบินเลี่ยงสิ่งกีดขวาง

6.5.1 ผลการทดลองกับทรี

เรากำหนดจุดหมายในทรีหรือเป็นกราฟที่ไม่มีลูปนั่นเองโดยแบ่งกลุ่มจุดหมายที่ใช้ในการทดลองออกเป็น 2 กลุ่มคือ กลุ่มจุดหมายที่ให้อัลกอริธึมมีการเปลี่ยนจุดหมายมากครั้งหรือเป็นกรณีที่แย่ที่สุด และกลุ่มจุดหมายที่ให้อัลกอริธึมมีการเปลี่ยนจุดหมายน้อยครั้งหรือเป็นกรณีที่ดีที่สุด



รูปที่ 6.9 กลุ่มจุดหมาย A ที่จำนวน 50 จุดหมาย

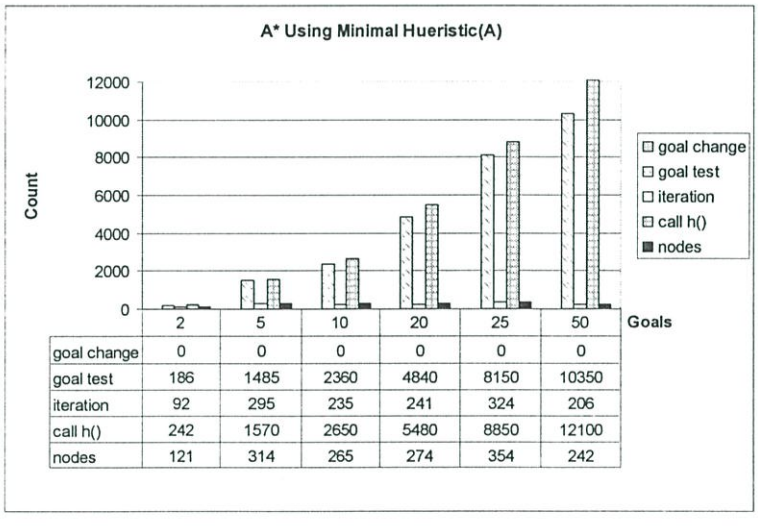


รูปที่ 6.10 จุดหมายกลุ่ม B ที่จำนวน 50 จุดหมาย

- กลุ่มจุดหมาย A เป็นกลุ่มจุดหมายที่ให้อัลกอริธึมมีการเปลี่ยนจุดหมายมากครั้งหรือเป็นกรณีที่แย่ที่สุดจะกำหนดให้จุดหมายกระจายตัวในระดับความลึกที่ใกล้เคียงกัน ดังรูปที่ 6.9 เป็นกราฟที่มีจุดหมายจำนวน 50 จุดหมายซึ่งเป็นวงกลมสีแดงด้านในสีเหลือง ส่วนจุดเริ่มต้นเป็นวงกลมสีแดงด้านในสีน้ำเงินซึ่งอยู่ตรงกลางของทรี จำนวนจุดหมายที่ใช้ในการทดลองมีตั้งแต่ 2, 5, 10, 20, 25, 50, และ 500 โนด

- กลุ่มจุดหมาย B เป็นกลุ่มจุดหมายที่ต่างออกไปเพื่อให้อัลกอริธึมที่ใช้ทดลองทำงานได้ดีที่สุดโดยกำหนดให้มีการเปลี่ยนจุดหมายน้อยครั้งที่สุด เราจึงกำหนดให้มีจุดหมายหนึ่งอยู่ใกล้จุดเริ่มต้นมากกว่าจุดหมายอื่นๆ ดังในรูปที่ 6.10 เป็นทรีที่มีจุดหมายจำนวน 50 จุดหมายเช่นเดียวกับรูปที่ 6.9 แต่แตกต่างกันที่มีจุดหมายหนึ่งที่ล้อมรอบด้วยวงกลมสีน้ำเงินจะอยู่ใกล้จุดเริ่มต้นมากกว่าจุดหมายอื่นๆ

เมื่อนิยามทรีที่ใช้ในการทดลองแล้ว เราจึงเริ่มการทดลองด้วยการการเปรียบเทียบอัลกอริธึมเอสตาร์แบบใช้ฟังก์ชั่นฮิวริสติกที่มีค่าน้อยที่สุดกับอัลกอริธึมเอสตาร์แบบค้นหาลึกลงไปแบบปกติซึ่งใช้ลิสต์ร่วมกัน



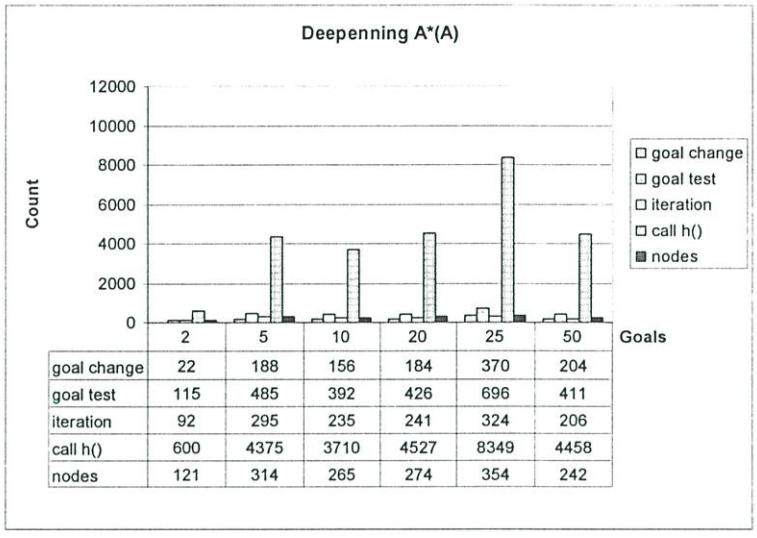
	2	5	10	20	25	50
goal change	0	0	0	0	0	0
goal test	186	1485	2360	4840	8150	10350
iteration	92	295	235	241	324	206
call h()	242	1570	2650	5480	8850	12100
nodes	121	314	265	274	354	242

รูปที่ 6.11 กลุ่มจุดหมาย A กับอัลกอริธึมเอสตาร์แบบใช้ฟังก์ชั่นฮิวริสติกที่ให้ค่าน้อยที่สุด

ผลการทดลองของกลุ่มจุดหมาย A แสดงในรูปที่ 6.11 และ 6.12 ซึ่งผลการทดลองกับอัลกอริธึมเอสตาร์แบบใช้ฟังก์ชั่นฮิวริสติกที่มีค่าน้อยที่สุดจากรูปที่ 6.11 แสดงให้เห็นว่าจำนวนครั้งในการคำนวณฟังก์ชั่นฮิวริสติกย่อยและการเปรียบเทียบจุดหมายเพิ่มขึ้นเมื่อจำนวนจุดหมายเพิ่มขึ้นซึ่งผลที่ได้เป็นไปตามสมการ (6.9) และมีค่า Complexity เป็นดังสมการ (6.9) คือ $O(mb^{l+1})$ เช่นกัน

ส่วนอัลกอริธึมเอสตาร์แบบค้นหาลึกลงไปเมื่อนำมาทดลองกับกลุ่มจุดหมาย A ผลที่ได้แตกต่างออกไปคือ เมื่อจำนวนจุดหมายเพิ่มขึ้นแต่จำนวนครั้งในการคำนวณฟังก์ชั่นฮิวริสติกย่อยและการเปรียบเทียบจุดหมายไม่เพิ่มขึ้นตามไปด้วย จากรูปที่ 6.12 ผลของ 50 จุดหมายมีจำนวนครั้ง

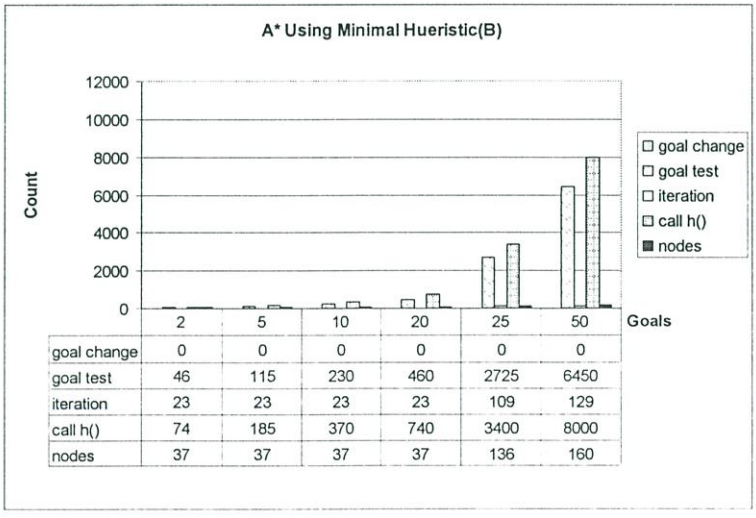
น้อยกว่าผลของ 25 จุดหมาย และนอกจากนี้การเปลี่ยนจุดหมายก็เป็นส่วนหนึ่งที่ทำให้จำนวนครั้งในการคำนวณฟังก์ชั่นฮิวริสติกย่อยและการเปรียบเทียบจุดหมายเพิ่มขึ้นซึ่งจำนวนครั้งในการคำนวณฟังก์ชั่นฮิวริสติกย่อยจะเพิ่มมากกว่า



	2	5	10	20	25	50
goal change	22	188	156	184	370	204
goal test	115	485	392	426	696	411
iteration	92	295	235	241	324	206
call h()	600	4375	3710	4527	8349	4458
nodes	121	314	265	274	354	242

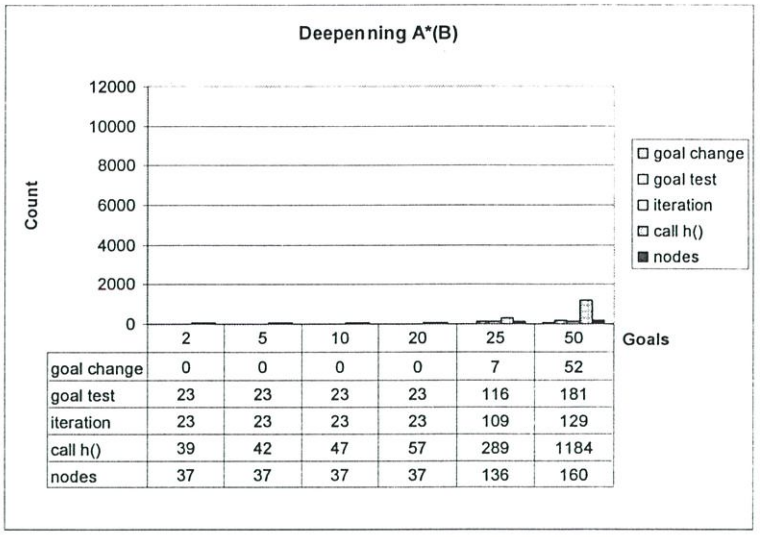
รูปที่ 6.12 จุดหมายกลุ่ม A กับอัลกอริธึมเอสตาร์แบบค้นหาลึกลงไป

ผลการทดลองของกลุ่มจุดหมาย B แสดงในรูปที่ 6.13 และ 6.14 ซึ่งรูปที่ 6.13 เป็นผลการทดลองของอัลกอริธึมเอสตาร์แบบใช้ฟังก์ชั่นฮิวริสติกที่มีค่าน้อยที่สุดซึ่งผลลัพธ์ไปในทิศทางเดียวกันกับกลุ่มจุดหมาย A ในรูปที่ 6.11 กล่าวคือ เพิ่มขึ้นตามจำนวนจุดหมาย



	2	5	10	20	25	50
goal change	0	0	0	0	0	0
goal test	46	115	230	460	2725	6450
iteration	23	23	23	23	109	129
call h()	74	185	370	740	3400	8000
nodes	37	37	37	37	136	160

รูปที่ 6.13 จุดหมายกลุ่ม B กับอัลกอริธึมเอสตาร์แบบใช้ฟังก์ชั่นฮิวริสติกที่ให้ค่าน้อยที่สุด



	2	5	10	20	25	50
goal change	0	0	0	0	7	52
goal test	23	23	23	23	116	181
iteration	23	23	23	23	109	129
call h()	39	42	47	57	289	1184
nodes	37	37	37	37	136	160

รูปที่ 6.14 จุดหมายกลุ่ม B กับอัลกอริธึมเอสตาร์แบบค้นหาลึกลงไป

ส่วนอัลกอริธึมเอสตาร์แบบค้นหาลึกลงไปเมื่อนำมาทดลองกับกลุ่มจุดหมาย B ดังในรูปที่ 6.14 ผลที่ได้จะดีกว่ากลุ่มจุดหมาย A ดังในรูปที่ 6.12 คือจำนวนครั้งในการคำนวณฟังก์ชั่นฮิวริสติกย่อยและใช้การเปรียบเทียบจุดหมายไม่มากเนื่องมาจากมีการเปลี่ยนจุดหมายลดลงนั่นเอง

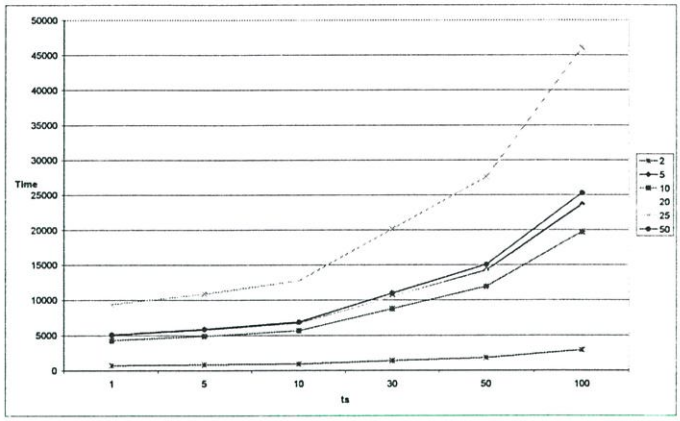
ผลการทดลองก็เป็นไปในทิศทางเดียวกันกับการวิเคราะห์ในหัวข้อที่ผ่านมาของอัลกอริธึมเอสตาร์แบบค้นหาลึกลงไปแบบใช้ลิสต์ร่วมกันคือ การเปลี่ยนจุดหมายมีผลต่อการทำงานของอัลกอริธึมเนื่องจากต้องทำการประเมินโนดที่เหลืออยู่ใน N และจัดเก็บโนดเหล่านี้ใหม่เพื่อความรวดเร็วในการเลือกโนดที่มีค่าที่น้อยที่สุดมาใช้งาน

ผลการทดลองแสดงให้เห็นว่าเวลาในการเปลี่ยนจุดหมายมีผลต่อการทำงานของอัลกอริธึมเอสตาร์แบบค้นหาลึกลงไปโดยใช้ลิสต์ร่วมกัน โดยจำนวนครั้งในการเรียกฟังก์ชั่นฮิวริสติกย่อยเกิดจาก 2 ส่วนคือ การเพิ่มโนดลงในลิสต์ซึ่งต้องนำค่าประเมินมาเรียงลำดับโนดจากค่าน้อยไปหามากและเมื่อต้องเปลี่ยนจุดหมายซึ่งค่าประเมินจากจุดหมายเดิมของแต่ละโนดไม่สามารถนำมาใช้งานกับจุดหมายใหม่ได้จึงต้องประเมินใหม่ เราจึงได้นำผลการทดลองมาหาเวลาการทำงานโดยรวมซึ่งคิดได้จาก

$$\begin{aligned} time &= (t_g * goal\ test) + (t_h * call\ h()_A) + (t_c * goal\ change) \\ &= (t_g * goal\ test) + (t_h * call\ h()_A) + [(t_h * call\ h()_B) + (t_s * sort\ node)] \\ &= (t_g * goal\ test) + (t_h * [call\ h()_A + call\ h()_B]) + (t_s * sort\ node) \end{aligned} \tag{6.18}$$

จากสมการ (6.18) เราทดลองกับจุดหมายกลุ่ม A ซึ่งเป็นการสมมุติให้อัลกอริธึมทำงานได้แย่ที่สุดโดยใช้ผลการทดลองในรูปที่ 6.13 และให้ t_h และ t_g มีค่าเท่ากันคือ 1 ส่วน t_s ให้มีค่าเป็น 1, 5,

10, 30, 50, และ 100 ซึ่งได้ผลดังในรูปที่ 6.15 เมื่อจุดหมายจำนวน 25 จุดหมาย อัลกอริธึมมีการเปลี่ยนจุดหมายหลายครั้งมากและเวลาที่ใช้ในการเปลี่ยนจุดหมายจึงใช้เวลานานและถ้าเวลาที่ใช้ในการจัดเรียงโนดหรือค่า t_s มีค่าสูงแล้วจะทำให้ใช้เวลาในการประมวลผลนานขึ้น



รูปที่ 6.15 กราฟเส้นแสดงเวลาในการเปลี่ยนจุดหมายของกลุ่ม A

ด้วยเหตุผลนี้ เราจึงพิจารณาใช้การสร้างลิสต์สำหรับจุดหมายที่ถูกเลือกบ่อยๆเพื่อลดการทำงานในการจัดเรียงจุดหมายใหม่ ต่อจากนี้จะเป็นการทดลองเปรียบเทียบการใช้ลิสต์ร่วมกันดังรูปที่ 6.16 และสร้างลิสต์สำหรับจุดหมายที่ถูกเลือกบ่อยๆ ที่มีจำนวนลิสต์เป็น 5, 10, 20, และไม่จำกัดจำนวน ดังรูปที่ 6.17, 6.18, 6.19 และ 6.20 ตามลำดับ ในแต่ละกราฟเส้นเป็นผลการทดลองกับกลุ่มข้อมูล A และ B เช่นเดียวกับการทดลองก่อนหน้านี้และยังได้เพิ่มจุดหมายอีก 1 ชุดคือจุดหมายจำนวน 500 จุดหมาย

การทดลองของแต่ละรูปมี 4 กราฟเส้นคือ

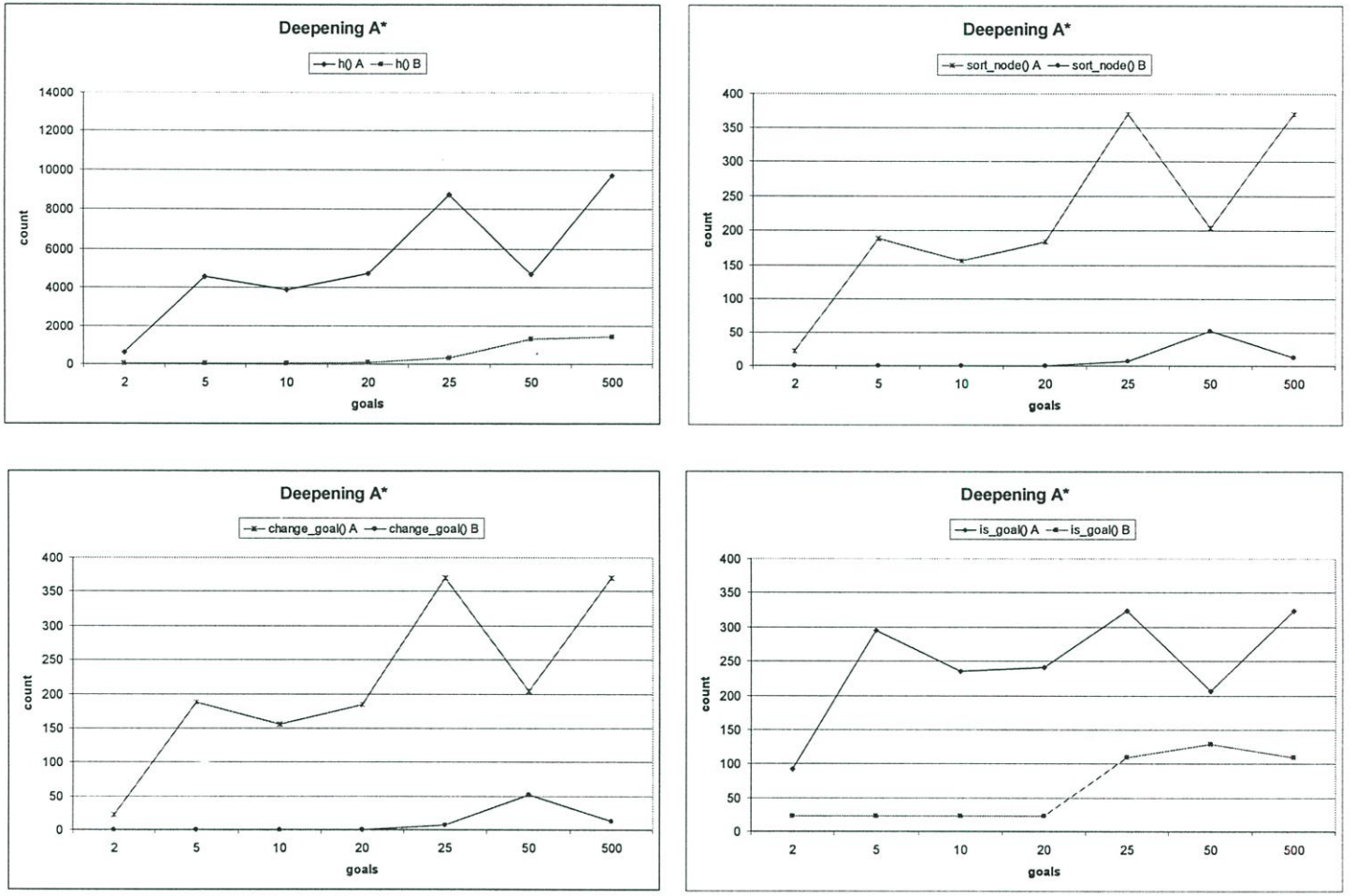
1. กราฟเส้นแสดงจำนวนครั้งในการคำนวณฟังก์ชั่นฮิวริสติกย่อยหรือกราฟ $h()$ การเรียกใช้ฟังก์ชั่นนี้เมื่อโนดถูกเพิ่มเข้าไปในลิสต์ และเมื่อต้องการจัดเรียงโนดใหม่จึงประเมินโนดในลิสต์ใหม่ทั้งหมดซึ่งเกิดขึ้นเมื่อการเปลี่ยนจุดหมายบางครั้งต้องการจัดเรียงโนดใหม่
2. กราฟเส้นแสดงจำนวนครั้งในการเรียกฟังก์ชั่นจัดเรียงโนดใหม่เมื่อไม่สามารถเพิ่มลิสต์ได้อีกจึงใช้ลิสต์เดิมหรือเรียกกราฟนี้ว่า *sort _ node*() โนดในลิสต์ที่เลือกมาจะต้องนำมาประเมินค่าใหม่ กราฟนี้ในการทดลองรูปที่ 6.20 มีจำนวนครั้งเป็น 0 หรือไม่มีการเรียกใช้ฟังก์ชั่นนี้
3. กราฟเส้นแสดงจำนวนครั้งในการเรียกฟังก์ชั่นเปลี่ยนจุดหมายหรือกราฟ *change _ goal*() การเรียกใช้ฟังก์ชั่นนี้เมื่อค่าประเมินที่น้อยที่สุดของจุดหมายที่กำลังพิจารณาอยู่นั่นมีค่ามากกว่าของจุดหมายอื่นจึงจำเป็นต้องเปลี่ยนจุดหมายในการพิจารณา

4. กราฟเส้นแสดงจำนวนครั้งในการเรียกฟังก์ชั่นในการทดสอบจุดหมายหรือกราฟ *is _ goal*()

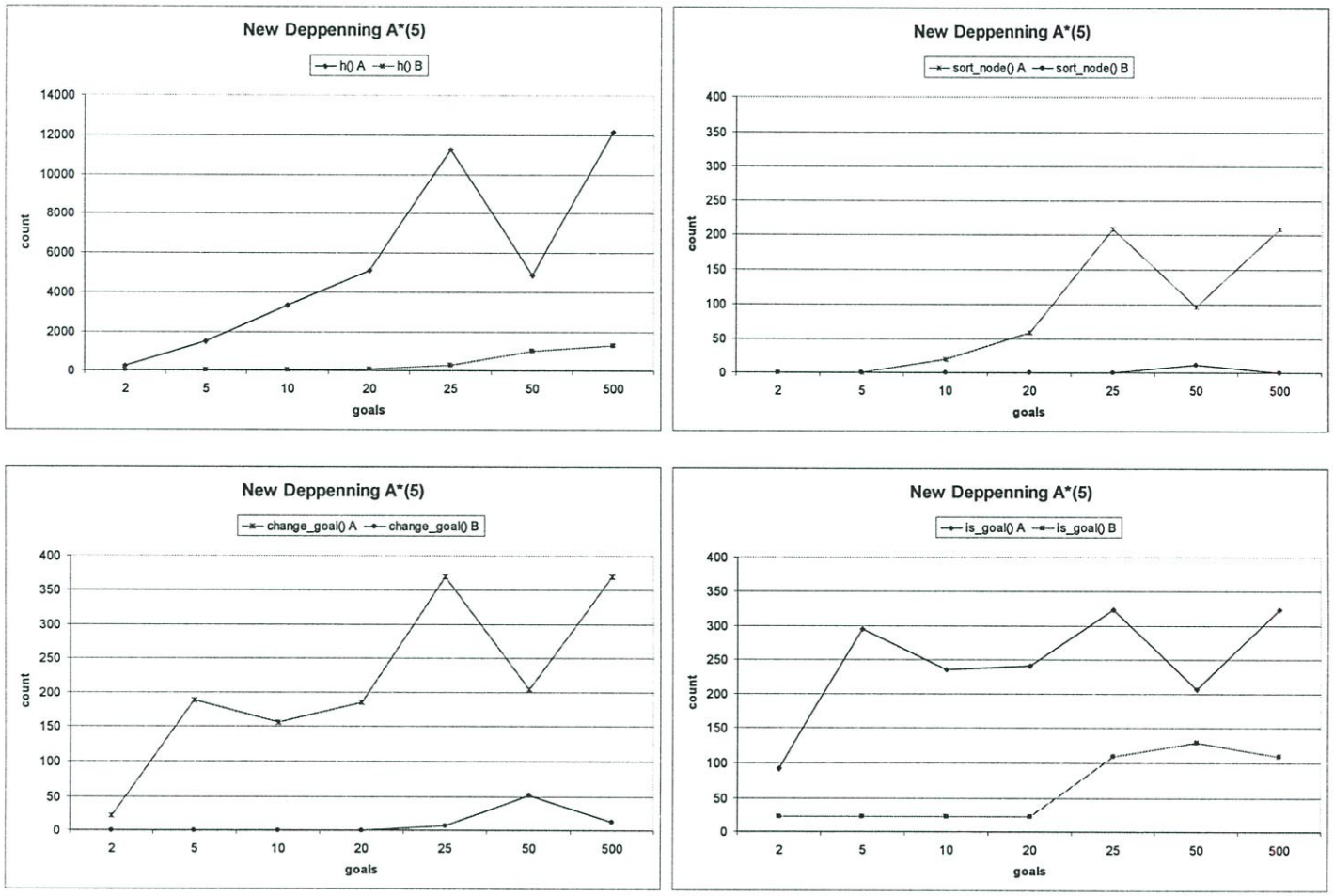
จากผลการทดลอง เมื่อเราพิจารณากราฟเส้นจำนวนครั้งการทดสอบจุดหมายและกราฟเส้นจำนวนครั้งในการเปลี่ยนจุดหมายในการพิจารณาของทุกๆการทดลองมีค่าเท่ากันหมดก็เนื่องจากจุดหมายใดที่มีค่าประเมินของจุดเริ่มต้นน้อยกว่าเส้นทางที่สั้นที่สุดจะถูกเลือกมาพิจารณาและด้วยคุณสมบัตินี้ทำให้การเพิ่มจำนวนลิสต์ไม่ส่งผลต่อคุณสมบัติการหาผลลัพธ์ที่ดีที่สุดหรือ Optimality

การกำหนดจำนวนลิสต์ที่เพิ่มมากขึ้นได้ส่งผลต่อ

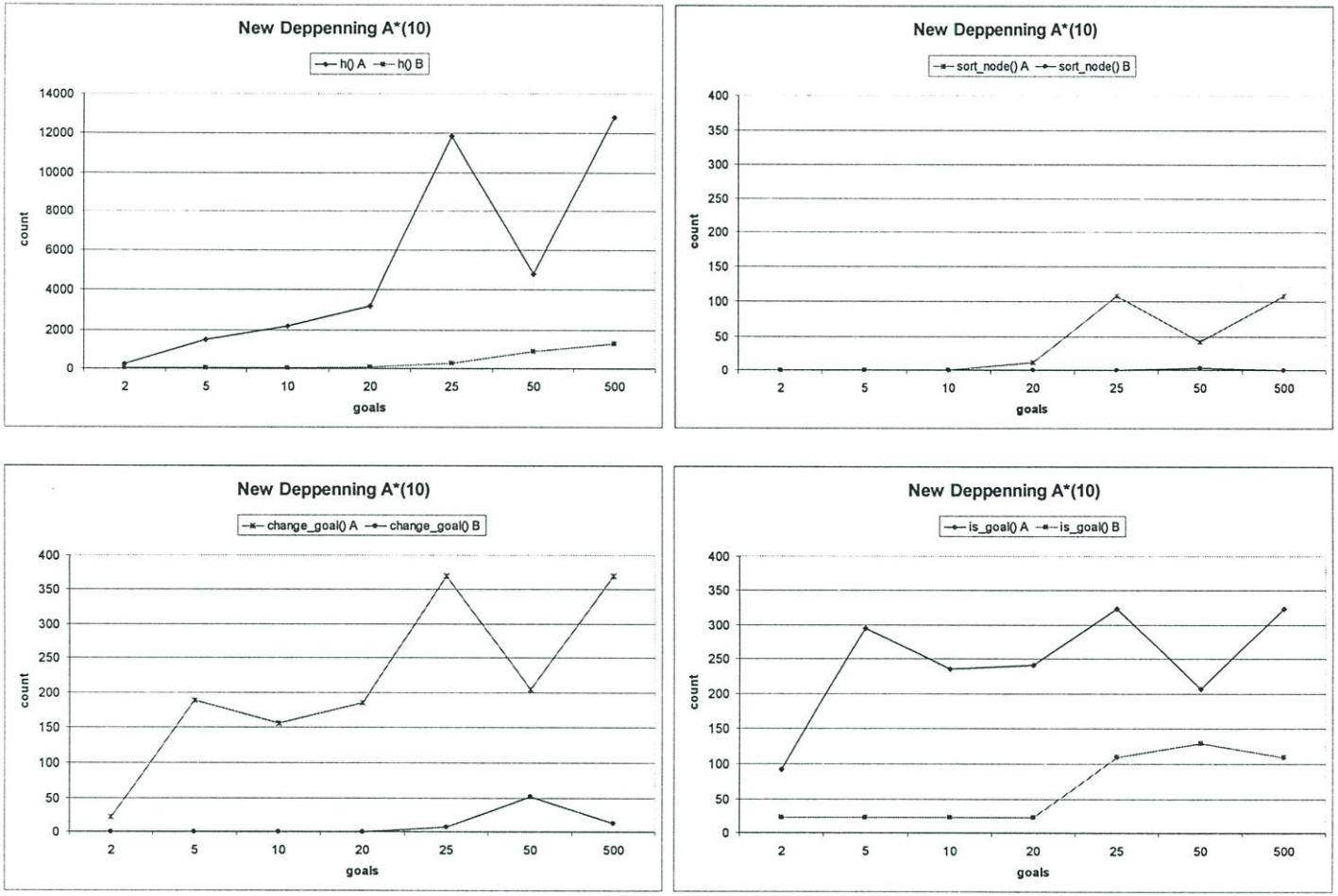
1. พื้นที่หน่วยความจำในการประมวลผลที่เพิ่มมากขึ้น
2. จำนวนครั้งในการจัดเรียงโนดซึ่งลดลงจากที่ต้องเรียงโนดใหม่ทุกครั้งที่เปลี่ยนจุดหมายเป็นเรียงโนดใหม่เป็นบางครั้ง
3. เวลาที่ใช้ในการเปลี่ยนจุดหมายโดยเกิดจากผลรวมของจำนวนครั้งในการคำนวณฟังก์ชั่นฮิวริสติกได้จากกราฟ *h*() และการเรียงโนดใหม่ได้จากกราฟ *sort _ node*() ลดลง ซึ่งเป็นผลต่อเนื่องมาจากข้อ 2



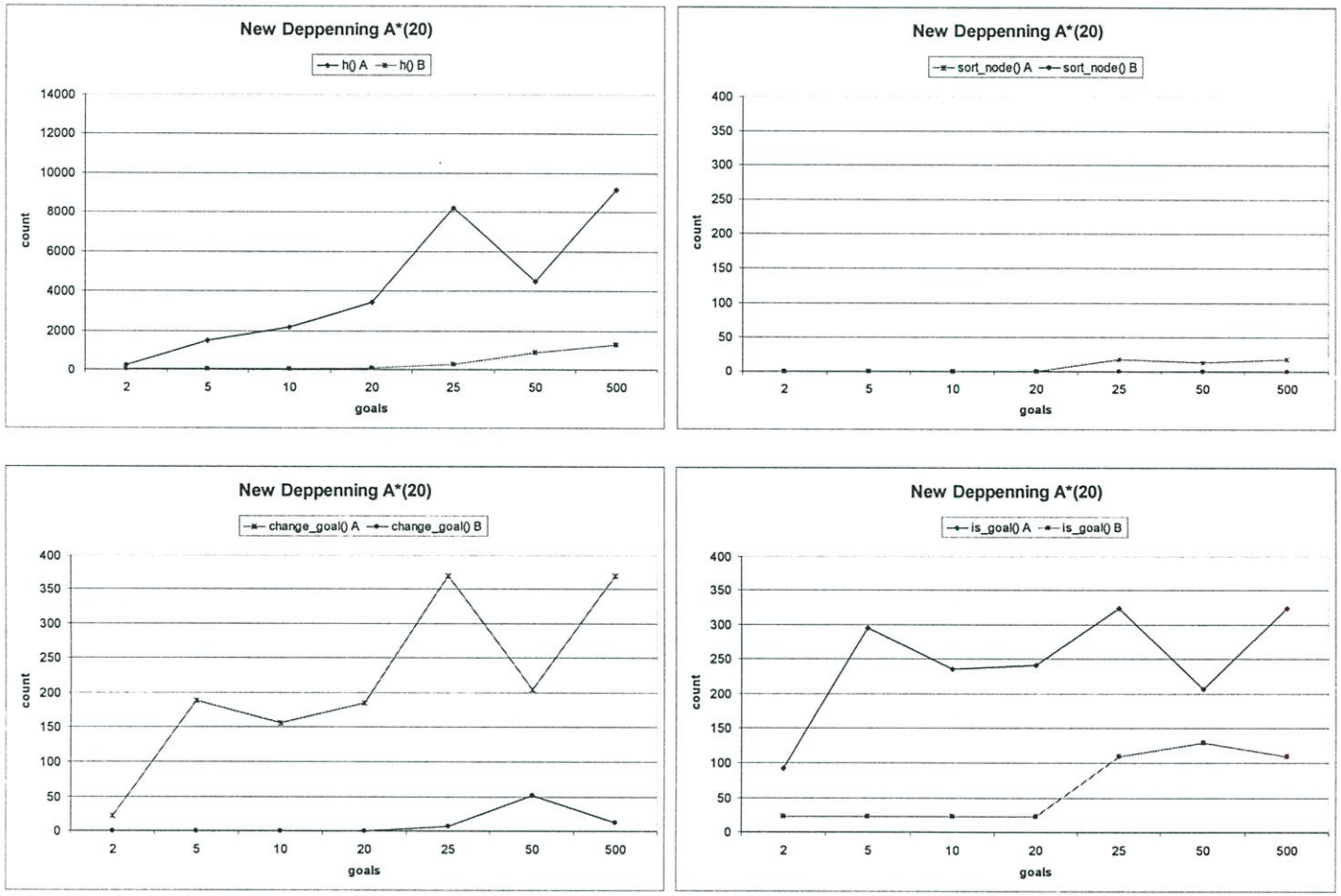
รูปที่ 6.16 กราฟเส้นแสดงผลการทดลองเมื่อใช้ลิสต์ร่วมกัน



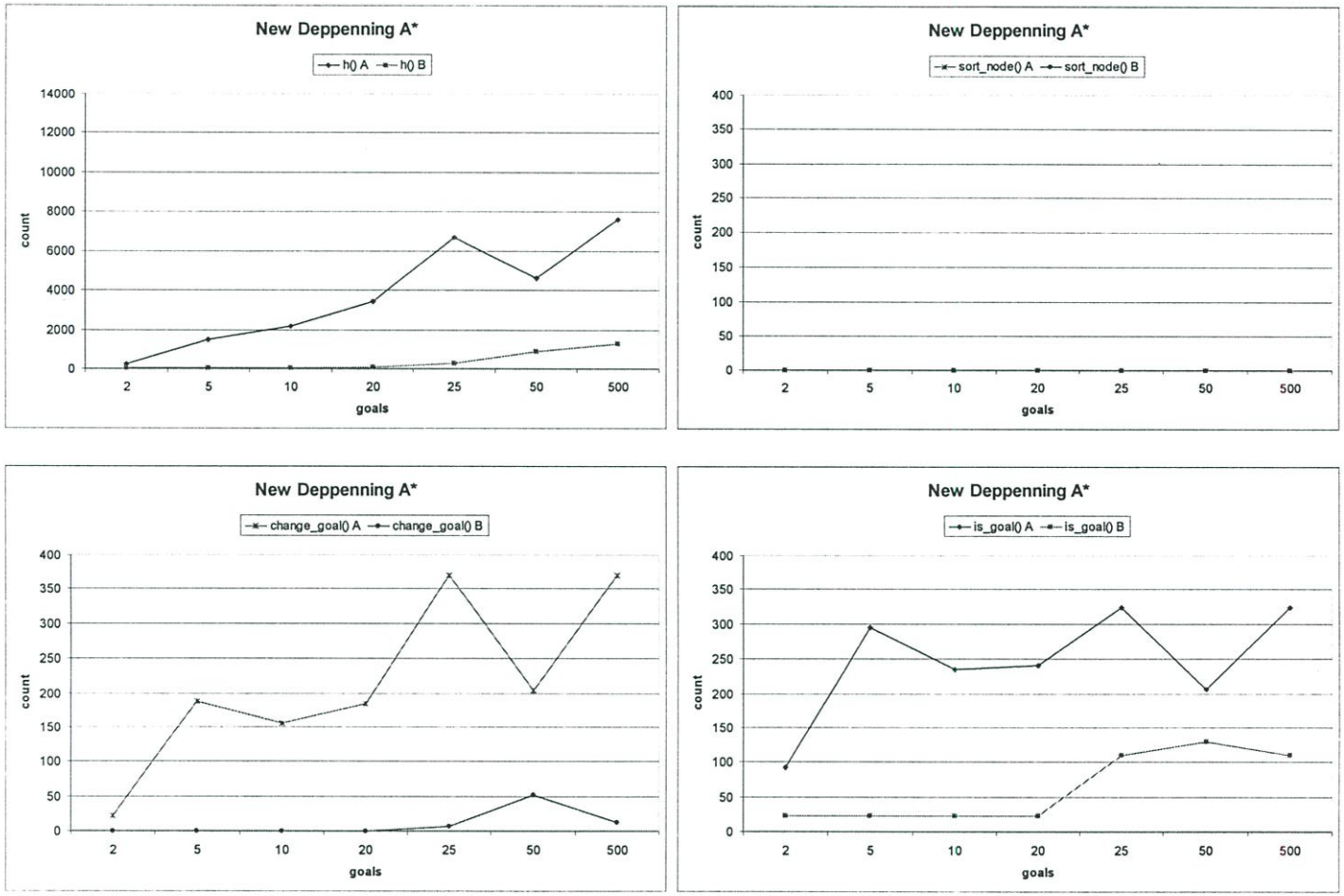
รูปที่ 6.17 กราฟเส้นแสดงผลการทดลองเมื่อกำหนดให้มีไม่เกิน 5 ลิสต์



รูปที่ 6.18 กราฟเส้นแสดงผลการทดลองเมื่อกำหนดให้มีไม่เกิน 10 ลิสต์



รูปที่ 6.19 กราฟเส้นแสดงผลการทดลองเมื่อกำหนดให้มีไม่เกิน 20 ลิสต์



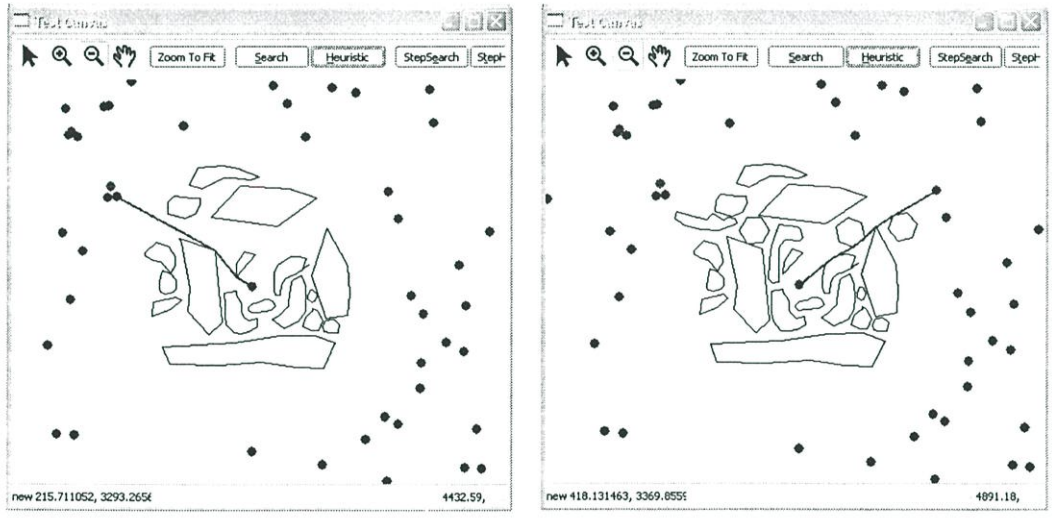
รูปที่ 6.20 กราฟเส้นแสดงผลการทดลองเมื่อกำหนดให้จำนวนลิสต์ไม่จำกัด

เราสามารถสรุปผลการทดลองสำหรับทรีได้ว่า อัลกอริธึมการค้นหาเอสตาร์แบบค้นหาลึกลงไปทำให้จำนวนครั้งในการคำนวณฟังก์ชั่นฮิวริสติกและจำนวนครั้งในการตรวจสอบจุดหมายได้ไม่ขึ้นจำนวนจุดหมายที่เพิ่มขึ้น เมื่อเรานำมาเปรียบเทียบกับอัลกอริธึมการค้นหาเอสตาร์ที่ใช้ฟังก์ชั่นฮิวริสติกที่ให้ค่าน้อยที่สุดซึ่งผลการทดลองขึ้นอยู่กับจำนวนจุดหมาย

แต่อัลกอริธึมที่นำเสนอก็มีข้อเสียเช่นกัน คือ เวลาที่ใช้ในการประมวลผลขึ้นอยู่กับจำนวนครั้งในการเปลี่ยนจุดหมายซึ่งทุกครั้งต้องมีการประเมินโนดและจัดเรียงโนดใหม่ทำให้เสียเวลา ดังนั้นเราจึงได้เสนอจากการใช้ลิสต์ร่วมกับเพียงลิสต์เดียวเป็นให้สามารถเพิ่มจำนวนลิสต์ได้และเลือกเก็บเฉพาะลิสต์ของจุดหมายที่มีถูกเลือกมาพิจารณาบ่อยๆ ด้วยวิธีการแบบนี้ นอกจากจะทำให้อัลกอริธึมยังคงมีคุณสมบัติ Optimality แล้วยังช่วยลดการจัดเรียงโนดใหม่และทำให้เวลาในการประมวลผลลดลง ซึ่งแปรผันผกผันกับพื้นที่หน่วยความจำ กล่าวคือ ถ้าใช้พื้นที่หน่วยความจำน้อยคือเพียงลิสต์เดียวจะใช้เวลาประมวลผลมาก แต่ถ้าเพิ่มพื้นที่หน่วยความจำจะทำให้ใช้เวลาลดลง

6.5.2 ผลการทดลองวางแผนเส้นทางบินเลี่ยงสิ่งกีดขวาง

เมื่อเราอัลกอริธึมการค้นหาเอสตาร์แบบค้นหาลึกลงไปกับทรีแล้วทั้งแบบใช้ลิสต์ร่วมและแบบใช้หลายลิสต์ เราจึงนำมาวางแผนเส้นทางบินเลี่ยงสิ่งกีดขวางโดยใช้อัลกอริธึมจากรูปที่ 5.5 มาประมวลผลและนำมาแสดงดังรูปที่ 6.21



รูปที่ 6.21 การวางแผนเส้นทางบินบนระนาบ 2 มิติ

บทที่ 7
สรุปผลการวิจัยและข้อเสนอแนะ

7.1 สรุปผลการทดลอง

วิทยานิพนธ์ฉบับนี้นำเสนอวิธีการวางแผนเส้นทางบินที่ดีที่สุดสำหรับอากาศยานไร้นักบินเพื่อเลี่ยงสิ่งกีดขวางซึ่งประยุกต์ใช้วิธีการค้นหาเส้นทางแบบเอสตาร์ที่สามารถหาเส้นทางที่สั้นที่สุดได้ โดยอาศัยข้อมูล 1) สมมุติฐานที่ว่าเราสามารถทราบตำแหน่งที่แน่ชัดของสิ่งกีดขวางก่อนการวางแผน 2) การวิเคราะห์ให้ทราบเส้นทางที่สามารถเลี่ยงสิ่งกีดขวางได้เร็วที่สุด และ 3) ลักษณะการบินของ UAV

การทำวิจัยในวิทยานิพนธ์นี้ เราเริ่มด้วยการแก้ปัญหาการวางแผนเส้นทางบินเพื่อเลี่ยงสิ่งกีดขวางที่มีจำนวนจุดหมายเพียงหนึ่งเดียวก่อน เนื่องด้วยวิธีการเอสตาร์แบบเดิมไม่สามารถใช้งานได้ในกรณีที่จำนวนโนดมีจำนวนมากเป็นอนันต์ เราจึงกำหนดสเปซในการวางแผนเส้นทางใหม่โดยสร้างรูปปิดล้อมรอบสิ่งกีดขวางจริงและใช้จุดยอดของรูปปิดเป็นโนดในสเปซในการวางแผนเส้นทาง ด้วยหลักการนี้ทำให้จำนวนของเส้นทางบินที่เป็นไปได้ลดลงเป็นจำนวนจำกัด จากนั้นเราจึงสามารถนำวิธีการเอสตาร์มาใช้โดยเพิ่มข้อกำหนดเกี่ยวกับข้อจำกัดการบินของอากาศยานเพื่อให้สามารถนำวิธีการวางแผนเส้นทางนี้ไปใช้งานได้จริงโดยไม่ขัดกับข้อจำกัดการบินของอากาศยาน

หลังจากนั้น เราได้แก้ปัญหาในกรณีที่จุดหมายจำนวนมากเป็นอนันต์โดยปรับปรุงอัลกอริธึมจากที่ได้อธิบายไปก่อนหน้านี้เสียใหม่ โดยเรียกอัลกอริธึมนี้ว่า “อัลกอริธึมการค้นหาเอสตาร์แบบค้นหาลึกลงไป” หรือ “Deepening A* Search” ซึ่งสามารถตัดจุดหมายที่อยู่ไกลมากๆ ออกจากการพิจารณาได้ และเรานำได้เปรียบเทียบผลกับอัลกอริธึมเอสตาร์ที่ได้เปลี่ยนมาใช้ฟังก์ชั่นฮิวริสติกของ Russell ที่เลือกใช้ค่าของฮิวริสติกที่มีค่าน้อยที่สุดจากหลายๆค่าซึ่งประสิทธิภาพของมันขึ้นอยู่กับจำนวนจุดหมาย ผลการวิเคราะห์ประสิทธิภาพและผลการทดลองแสดงให้เห็นว่าอัลกอริธึมที่นำเสนอสามารถวางแผนเส้นทางได้อย่างมีประสิทธิภาพโดยไม่ขึ้นอยู่กับจำนวนจุดหมาย นอกจากนี้ยังได้เพิ่มประสิทธิภาพด้วยการเพิ่มจำนวนลิสต์ที่ช่วยการประมวลผลใหม่

วิธีการวางแผนเส้นทางบินที่ดีที่สุดสำหรับอากาศยานไร้นักบินที่นำเสนอในวิทยานิพนธ์ฉบับนี้สามารถวางแผนเส้นทางบินเลี่ยงสิ่งกีดขวางในกรณีที่มีจุดหมายมีเพียงหนึ่งจุดหมายหรือมีมากมายเป็นอนันต์ได้อย่างมีประสิทธิภาพ

7.2 ปัญหาและอุปสรรค

ปัญหาในการในการทำงานวิจัยนี้ คือ เราไม่สามารถเปรียบเทียบประสิทธิภาพการทำงานของอัลกอริธึมเอสตาร์แบบค้นหาลึกลงไปได้อย่างแน่ชัดด้วยสมการคณิตศาสตร์

7.3 ข้อเสนอแนะ

1. ควรหาสมการทางคณิตศาสตร์ที่แสดงให้เห็นถึงประสิทธิภาพการทำงานของอัลกอริธึมเอสตาร์แบบค้นหาลึกลงไปอย่างแน่ชัด
2. ควรหาอัตราส่วนของจำนวนลิสต์ต่อจำนวนจุดหมายที่เหมาะสมเพื่อให้มีประสิทธิภาพการทำงานสูงสุดโดยใช้เวลาและหน่วยความจำน้อย
3. ควรออกแบบหรือกำหนดสเปซบนระนาบ 3 มิติ เพื่อให้สามารถนำอัลกอริธึมที่นำเสนอไปใช้วางแผนเส้นทางบินบนระนาบ 3 มิติได้
4. ควรพัฒนาอัลกอริธึมให้สามารถวางแผนเส้นทางบินในกรณีที่สภาพแวดล้อมหรือสเปซมีการเปลี่ยนแปลงได้ ซึ่งในกรณีนี้จำเป็นต้องวางแผนเส้นทางใหม่เมื่อสเปซมีการเปลี่ยนแปลง

เอกสารอ้างอิง

[1] Dobbs, V., Davis, H., and Lizza, C., "**An application of heuristic search techniques to the problem of flight path generator in a military hostile environment**", *International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, pp 273 – 280, 1988.

[2] Szczerba, J., Galkowski, P., Glickstein, S., and Ternullo, N., "**Robust algorithm for real-time route planning**", *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 36, No.3 July 2000.

[3] Szczerba, J. "**Threat netting for real-time, intelligent route planners**", *Proceedings on Information Decision and Control 99*, pp. 377-382, February 1999.

[4] Stentz, A. "**Optimal and efficient path planning for partial-known environment**", *Proceedings on the IEEE International Conference on Robotics and Automation*, 1994.

[5] Yan, P., Ding, M., Zhou, C., and Zheng, C., "**A path replanning algorithm based on roadmap-diagram**", *Proceedings on the 5th World Congress on Intelligent Control and Automation*, Jun 15-19 2004.

[6] Rathburn, D., Kragelund, S., and Pongpunwattana, A., "**An evolution based path planning algorithm for autonomous motion of UAV through uncertain environments**", *Proceedings on AIAA's 1st Technical Conferences and Workshop on Unmanned Aerospace Vehicle, System, Technologies, and Operations, Portsmouth*, 2002.

[7] Hwang, Y. J., Kim, S. J., Lim, S. S., and Park, H. K., "**A fast path planning by path graph optimization**", *IEEE Transaction on System, Man, and Cybernetics-part a: System and Humans*, Vol. 33, No.1 January 2003.

[8] Russell, S., Norvig, P., **Artificial Intelligence: A Modern Approach**, Prentice Hall, 2003.

[9] Weiss, Mark A., **Data Structures and Algorithm Analysis in C**, Addison Wesley Longman, 1997.

[10] Dominik, H., Christian, W., and Heinz, W., "**Multi-directional search with goal switching for robot path planning**", *The 11th International Conference on Industrial & Engineering Application of Artificial Intelligence & Expert Systems*,

Castellan, Spain, Jul. 1-4, 1998.

[11] W. Christian, H. Dominik, and W. Heinz, "**Multi-goal path planning for industrial robots**", *International Conference on Robotics and Application (RA'99)*, Santa Barbara, USA, Oct. 28-90, 1999.

[12] Hsiao, Y. T., Chuang, C. L., and Chien, C. C., "**Ant Colony Optimization for Best Path Planning**", *International Symposium on Communications and Information Technologies 2004 (ISCIT 2004)*, Sapporo, Japan, Oct.26-29, 2004.

[13] Chong, C. S., Low, M. Y. H., Sivakumar, A. I., and Gay, K. L., "**A Bee Colony Optimization Algorithm to Job Shop Scheduling**", *Proceedings of the 2006 Winter Simulation Conference*, Monterey, California, United States, 2006.

[14] Grimm, J. J., Lamont, G. B., and Terzouli, A. J., "**A Parallelized Search Strategy for Solving a Multicriteria Aircraft Routing Problem**", *Proceedings of the 1993 ACM/SIGAPP symposium on Applied computing: states of the art and practice*, Indianapolis, Indiana, United States, 1993.

[15] Gudatis, M. S., Lamont, G. B., and Terzouli, A. J., "**Multicriteria Vehicle Route-Planning using Parallel A* Search**", *Proceedings of the 1995 ACM symposium on Applied computing*, Nashville, Tennessee, United States, 1995.

[16] Cvetonovic, Z. and Nofsinger, C., "**Parallel Astar Search on Message-passing Architecture**", *System Sciences, 1990., Proceedings of the Twenty-Third Annual Hawaii International Conference on*, Kailua-Kana, Hawaii, United States, Jan. 1-4, 1990.

[17] Grama, A. Y. and Kumar, V., "**A Survey of Parallel Search Algorithms for Discrete Optimization Problems**", *ORSA J. Computing*, 1995

[18] Kambhampati, S. and Davis, L.S., "**Multiresolution path planning for mobile robots**", *IEEE J. Robot. Automat., vol. RA-2*, pp.135-145, Sept. 1986.

[19] Douglas R. Smith., "**Random trees and the analysis of branch and bound procedures**", *Journal of the ACM, vol. 31 No.1*, Jan. 1984.

[20] Bortoff, S. A, "**Path Planning for UAVs**", *Proceedings of American Control Conference*, Chicago, Illinois, United States, June 2000.

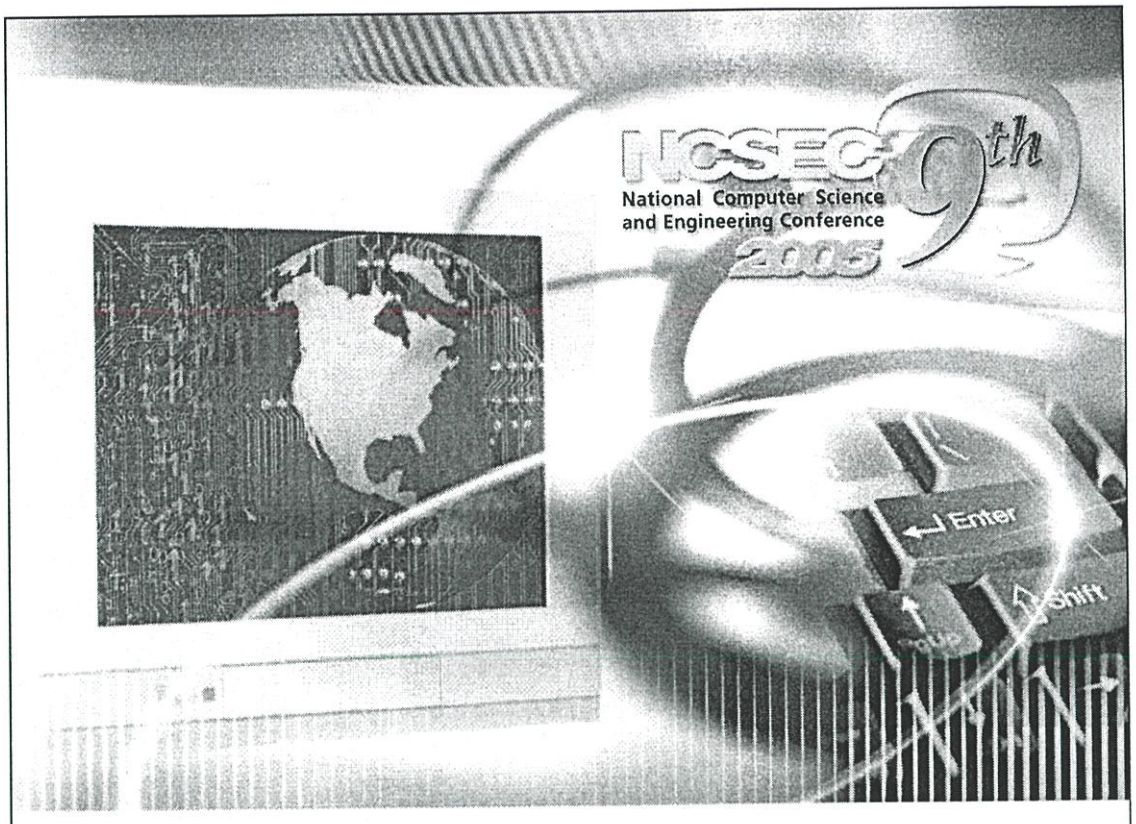
ภาคผนวก

ภาคผนวก ก.

ผลงานวิจัยที่ได้รับการตีพิมพ์

[1] เกียรติศักดิ์ คิวขุนทด, วิศิษฏ์ หิรัญกิตติ, "การวางแผนเส้นทางบินที่สามารถเลี่ยงสิ่งกีดขวางสำหรับอากาศยานไร้นักบิน", *การประชุมเชิงวิชาการวิทยาการคอมพิวเตอร์และวิศวกรรมคอมพิวเตอร์แห่งชาติครั้งที่ 9 (NCSEC2005), กรุงเทพ, 2548*

[2] เกียรติศักดิ์ คิวขุนทด, วิศิษฏ์ หิรัญกิตติ, "การวางแผนเส้นทางบินที่มีจำนวนไม่จำกัดด้วยวิธีเอสตาร์แบบค้นหาลึกลงไป", *การประชุมเชิงวิชาการวิทยาการคอมพิวเตอร์และวิศวกรรมคอมพิวเตอร์แห่งชาติครั้งที่ 10 (NCSEC2006), ขอนแก่น, 2549* (ได้รับการคัดเลือกตีพิมพ์ใน NECTEC Technical Journal 2007)



The 9th National Computer Science and Engineering Conference

October 27-28, 2005

University of Thai Chamber of Commerce, Bangkok Thailand

Organized by:

Department of Computer Engineering, School of Engineering,
University of Thai Chamber of Commerce

In Cooperation with:

Electrical Engineering; Electronics, Computer, Telecommunications and Information Technology Association of Thailand (ECTI)

IEEE Communications Society, Thailand Chapter

Sponsored by:

University of Thai Chamber of Commerce

National Electronics and Computer Technology Center (NECTEC)

Sun Microsystems (Thailand)

CS Loxinfo Public Company Limited

Pearson Education Indochina Limited

The OGA Group

การวางแผนเส้นทางบินที่สามารถเลี่ยงสิ่งกีดขวางสำหรับอากาศยานไร้นักบิน

A Path Planning Capable of Obstacle Avoidance for Unmanned Aerial Vehicles

เกียรติศักดิ์ คิวขุนทด และวิศิษฏ หิรัญกิตติ
หองวิจัยการสื่อสารและคมนาคมชาญฉลาด ภาควิชาวิศวกรรมคอมพิวเตอร
สถาบันเทคโนโลยีพระจอมเกลาเจาคุณทหารลาดกระบัง กรุงเทพ 10520 ประเทศไทย
E-mail: k_keattisak@hotmail.com, v_hirankitti@gmail.com

บทคัดย่อ

บทความนี้นำเสนอวิธีใหม่ในการวางแผนเส้นทางบินที่สามารถเลี่ยงสิ่งกีดขวางสำหรับอากาศยานไร้นักบินหรือ UAV ด้วยวิธีการค้นหาเส้นทางแบบ A ที่มีข้อดีคือสามารถให้เส้นทางที่สั้นที่สุด แต่มีข้อเสียคือในกรณีที่เส้นทางที่เป็นไปได้ทั้งหมดมีจำนวนอนันต์จะทำให้ A* ไม่สามารถใช้งานได้ ดังนั้นเราจึงได้ดัดแปลงใช้ A* ให้เหมาะสมกับการวางแผนเส้นทางบินของ UAV ที่มีลักษณะ Real time ด้วยการพยายามลดจำนวนโหนดในกราฟที่ A* ต้องพิจารณาลง โดยอาศัยข้อมูล 1) สมมุติฐานที่ว่าเราสามารถทราบตำแหน่งที่แน่ชัดของสิ่งกีดขวางก่อนการวางแผน 2) การวิเคราะห์ให้ทราบเส้นทางที่สามารถเลี่ยงสิ่งกีดขวางได้เร็วที่สุด และ 3) ลักษณะการบินเคลื่อนที่ของ UAV*

Abstract

This paper presents a new approach for path planning, capable of obstacle avoidance, used by an unmanned aerial vehicle (UAV). This approach is adapted from the A algorithm which can be used in path planning to find a shortest path. However, A* may not be workable for path planning for a flying object, like UAV, since the number possible paths for a UAV is infinite whilst A* presumes a finite search space. In this paper we have adapted A* to use for UAV path planning in order to avoid obstacles in the real-time manner. In our approach the number of candidate nodes, to be considered by A*, in the search space is reduced substantially by using the following information: (1) the assumption that the positions of the obstacles to be avoided by the UAV are certain and known in advance of planning (2) the results from the analysis to identify shortest ways to avoid the obstacles (3) the knowledge of flying characteristic for a certain type of UAVs.*

Key words: Path planning, A* Algorithm, Unmanned aerial vehicles

1. บทนำ

การวางแผนเส้นทางบินเป็นปัญหาที่สำคัญสำหรับความปลอดภัยในการเดินทางของ UAV (Unmanned Air Vehicle) ในบทความนี้ได้ศึกษางานวิจัยที่เกี่ยวข้องกับการวางแผนเส้นทางซึ่งนำไปใช้งานในเชิงพาณิชย์ และงานทางทหาร [1] การวางแผนเส้นทางมีการนำเสนอไว้หลายวิธีดังนี้ วิธี Heuristic search [1, 2, 5], วิธีการค้นหาย้อนหลัง (Backward search) เสนอโดย Stentz [4] และ Yan [5] ได้กล่าวถึงวิธีที่รียกว่า D* ซึ่งมีข้อดีที่สามารถช่วยให้วางแผนเส้นทางใหม่ (Replanning) ทำได้อย่างรวดเร็วยิ่งขึ้น หรือการวางแผนเส้นทางด้วย Evolutionary Algorithm [6] ซึ่งการหาค่าน้ำหนักที่เหมาะสมในทุกกรณีด้วยวิธีนี้ทำได้ยาก

ปัญหาที่สำคัญอย่างหนึ่งของการวางแผนเส้นทางบินสำหรับ UAV คือ จำนวนโหนดที่ต้องพิจารณาในการวางแผนมีจำนวนนับไม่ถ้วนเนื่องจากเมื่อ UAV บินจากจุดเริ่มต้นไปยังเปาหมายผ่านไปในสภาพแวดล้อมจริง ณ ตำแหน่งใดๆ มันสามารถเคลื่อนที่ไปได้หลายทิศทางและสามารถเลี้ยว ณ หลายตำแหน่ง

เช่นกัน ทำให้เส้นทางบินของ UAV มีความเป็นไปได้มากมายนับไม่ถ้วน (เป็นอนันต์) มีผลทำให้การวางแผนเส้นทางโดยวิธี A* ไม่สามารถใช้ได้กับกรณีนี้เนื่องจาก A* ใช้วางแผนได้ก็ต่อเมื่อเส้นทางที่เป็นไปได้ทั้งหมดมีจำนวนจำกัด นอกจากนี้ได้มีงานวิจัยหลายชิ้นแก้ปัญหาการใช้ A*โดยได้เสนอวิธีการลดจำนวนโหนดในกราฟที่ต้องพิจารณาลงและทำให้เส้นทางที่มีจำนวนอนันต์กลายเป็นจำนวนที่จำกัดได้ เช่น วิธีการแบ่งสเปซออกเป็นกริด แต่เส้นทางที่ได้เป็นเส้นทางที่ไม่มีความแม่นยำ หรืองานวิจัยของ Hwang [7] ได้ลดจำนวนโหนดที่ต้องพิจารณาลงด้วยการสร้างโครงข่ายสามเหลี่ยมที่ครอบสิ่งกีดขวางและค้นหาเส้นทางบนโครงข่ายเหล่านั้นด้วยการหาเส้นทางโดยวิธี Dijkstra ข้อดีของงานวิจัยชิ้นนี้คือสามารถลดจำนวนโหนดที่ต้องพิจารณาลงได้ และได้เส้นทางที่ดีกว่าวิธีการแบ่งเซลล์ สำหรับงานวิจัยของ Szczerba [2] ได้อาศัยลักษณะการเคลื่อนที่ของ UAV มาใช้ประโยชน์ในการลดจำนวนเส้นทางบินที่อาจเป็นได้ที่ต้องพิจารณาลง

บทความนี้ได้นำเสนอการวางแผนเส้นทางบินที่สามารถเลี่ยงสิ่งกีดขวางที่มีประสิทธิภาพโดยอาศัยวิธี A* โดยมีการคำนวณอย่างรวดเร็วเพื่อให้สามารถใช้งานในการวางแผนเส้นทางการบินของ UAV ในลักษณะ Real time ได้

สำหรับส่วนที่เหลือของบทความนี้หัวข้อที่ 2 จะอธิบายถึงปัญหาการวางแผนเส้นทางการบินของ UAV หัวข้อที่ 3 กล่าวถึงสเปซของกราฟเส้นทางที่เป็นไปได้ ส่วนหัวข้อที่ 4 เป็นอัลกอริธึมการวางแผนเส้นทางที่เสนอในบทความนี้ ตามมาด้วยผลการทดลองในหัวข้อที่ 5 แล้วเป็นการเปรียบเทียบกับงานวิจัยอื่นในหัวข้อถัดไปและท้ายที่สุดจะสรุปในหัวข้อที่ 7

2. ปัญหาการวางแผนเส้นทางบินของ UAV

การวางแผนเส้นทางการบินของ UAV จากจุดเริ่มต้นไปยังเป้าหมายโดยไม่ชนเข้ากับสิ่งกีดขวางเป็นงานที่มีความซับซ้อน สิ่งกีดขวางเป็นสิ่งมีผลต่อการเดินทางของ UAV โดยตรง สิ่งกีดขวางที่อยู่ในพื้นที่ที่พิจารณาอาจเป็นสิ่งกีดขวางที่ไม่เป็นอันตราย เช่น ภูเขา หรือ ตึกสูง เป็นต้น หรือเป็นสิ่งกีดขวางที่เป็นอันตรายสามารถคุกคาม หรือทำลาย UAV ได้ เช่น ปนต่อสู้อากาศยาน รถถัง เป็นต้น ดังนั้นลักษณะของสิ่งกีดขวางประเภทหลังนี้จึงถูกกำหนดเป็นขอบเขตของบริเวณที่เป็นอันตราย ดังนั้นเพื่อให้ UAV สามารถเดินทางได้อย่างปลอดภัยจึงจำเป็นต้องทราบตำแหน่งและกำหนดลักษณะที่แน่ชัดของสิ่งกีดขวางไว้ก่อนการวางแผน

ลักษณะการบินเคลื่อนที่ของ UAV ก็เป็นอีกปัจจัยหนึ่งที่มีผลต่อการวางแผนเส้นทางบินเนื่องจากเส้นทางที่สั้นที่สุดสำหรับการเคลื่อนที่ไป อาจเป็นเส้นทางที่ UAV ไม่สามารถบินไปได้จริงเพราะขัดกับลักษณะการบินที่ปลอดภัย เช่น บางเส้นทางต้องทำมุมเลี้ยว 90 องศา ซึ่ง UAV ประเภทเครื่องบินไม่สามารถทำได้ เป็นต้น สำหรับลักษณะการเคลื่อนที่ของ UAV แต่ละประเภทก็มีความแตกต่างกันออกไป สำหรับวิธีการที่เสนอนั้น สามารถรองรับลักษณะการบินของ UAV ของแต่ละประเภทได้ซึ่งมักมีข้อจำกัดการเคลื่อนที่ที่สำคัญได้แก่

1. Maximum turning angle เป็นการกำหนดมุมการเลี้ยวมากที่สุดที่ UAV สามารถทำได้

2. Route distance constraint เป็นการกำหนดระยะทางไกลสุดของการบิน เพราะต้องพิจารณาเรื่องปริมาณน้ำมันที่มีจำกัดด้วย

มีการเสนอให้นำเอาลักษณะข้อจำกัดในการบินของ UAV มาใช้ร่วมกับ A* ดังเช่น งานวิจัยของ Szczerba [2] ซึ่งสามารถช่วยลดจำนวนโหนดที่ A* ต้องพิจารณาลง ส่งผลทำให้เพิ่มประสิทธิภาพความเร็วในการวางแผนเส้นทาง ซึ่งวิธีการวางแผนเส้นทางที่นำเสนอจะได้นำวิธีการนี้มาประยุกต์ใช้เช่นเดียวกัน

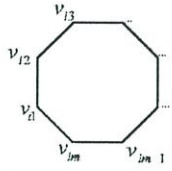
3. สเปซ (Space) ของกราฟเส้นทางที่เป็นไปได้

เพื่อแก้ปัญหาที่ว่าจำนวนเส้นทางที่เป็นไปได้ของการบินของ UAV มีค่าอนันต์ ดังนั้นเราจึงจะพิจารณาเฉพาะเส้นทางที่เป็นไปได้ที่อยู่ใกล้เคียงกับสิ่งกีดขวางเพื่อลดจำนวนเส้นทางที่เป็นไปได้ที่จะค้นหาให้เหลือเพียงจำนวนจำกัด จากนี้เราขอกำหนดสเปซสำหรับการค้นหาดังต่อไปนี้

เริ่มจากขอพิจารณาสิ่งกีดขวางก่อน โดยกำหนดให้สิ่งกีดขวางที่ใช้ในวิธีการที่เสนอเป็นสิ่งกีดขวางสมมุติโดยลอมรอบ

สิ่งกีดขวางจริงเพื่อให้เส้นทางบินที่อยู่บริเวณรอบๆสิ่งกีดขวางสมมุติอยู่ในระยะปลอดภัยต่อ UAV และต่อจากนี้เราจะใช้สิ่งกีดขวางในความหมายนี้สำหรับวิธีการที่เสนอ

สิ่งกีดขวางมีลักษณะเป็นรูปปิดทรงหลายเหลี่ยมสองมิติ (Polygon) ซึ่งสร้างจากการลากเส้นตรงเชื่อมจุดยอดแต่ละจุดของรูปปิด ซึ่งสิ่งกีดขวางใดๆสามารถเขียนให้อยู่ในรูปของคู่อันดับของจุดยอดทั้งหมด



รูปที่ 1. แสดงแสดงสิ่งกีดขวางหนึ่งๆ

$$P_i = v_{i1}, v_{i2}, v_{i3}, \ldots, v_{im-1}, v_{im} \quad (1)$$

เมื่อ m เป็นจำนวนจุดยอด โดยมีคู่อันดับ (v_{ij}, v_{ij+1}) ซึ่ง $1 \le j < m$ และคู่อันดับ (v_{im}, v_{i1})

สำหรับจุดยอดทั้งหมดของสิ่งกีดขวางใดๆจะแทนในรูปของเซตได้เป็น

$$O_i = \{v_{i1}, v_{i2}, v_{i3}, \ldots, v_{im-1}, v_{im}\} \quad (2)$$

กราฟของโหนดที่ใช้ค้นหาเส้นทางโดยอัลกอริธึมวางแผนเส้นทางประกอบด้วยเซตของจุดยอด (Vertex) ของสิ่งกีดขวาง ซึ่งถือว่าเป็นโหนด และเซตของเส้นตรง (Edge) ที่เชื่อมระหว่างโหนด หรืออาจเขียนสัญลักษณ์ได้เป็น $G = (V, E)$

โดยที่เซตของโหนด V เป็นผลมาจากการยูเนียนของเซตของโหนดของสิ่งกีดขวางทั้งหมด n ตัวแล้วนำมายูเนียนกับเซตของโหนดเริ่มต้น s และโหนดเป้าหมาย g สำหรับการค้นหาเส้นทาง ดังในสมการที่ 3

$$V = \{s, g\} \cup \bigcup_{i=1}^{n} O_i \quad (3)$$

และเส้นตรง E ในกราฟ G เป็นคู่ของโหนด (u, v) เมื่อ $u, v \in V$

ดังนั้นเราสามารถหาสเปซเส้นทางที่เป็นไปได้ทั้งหมดจากกราฟ G ในรูปของเซตของชุดลำดับของโหนด $w_1, w_2, w_3, \ldots, w_n$ โดยที่ $(w_i, w_{i+1}) \in E$ ซึ่ง $1 \le i < n$

สเปซของกราฟที่มีขนาดใหญ่ใช้แทนเส้นทางที่เป็นไปได้ทั้งหมดสำหรับการเดินทางเลี่ยงสิ่งกีดขวาง จากนี้ไปเราจะใช้สเปซในความหมายนี้สำหรับการค้นหาวิธี A* ซึ่งได้ลดจำนวนเส้นทางที่เป็นไปได้ทั้งหมดจากที่เป็นอนันต์ให้เหลือเป็นจำนวนที่จำกัดแล้ว

4. อัลกอริธึมการวางแผนเส้นทางสำหรับ UAV

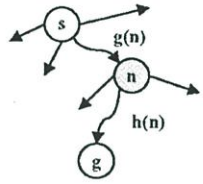
วิธีการวางแผนเส้นทางบินแบบใหม่สำหรับ UAV ที่นำเสนอนี้ได้ดัดแปลงมาจากวิธีค้นหา A* แบบมาตรฐาน ก่อนอธิบายการทำงานของวิธีวางแผนเส้นทางบินแบบใหม่ เราจะอธิบายการทำงานของ A*แบบมาตรฐานอย่างคร่าวๆ ก่อน

4.1. การค้นหาแบบ A*

ในงานวิจัยนี้เราได้ประยุกต์ใช้การค้นหาแบบเอสตาร์ ซึ่งใช้ Heuristic function

$$f(n) = g(n) + h(n) \quad (4)$$

ในการชี้นำการค้นหาให้ไปสู่โหนดเป้าหมายได้อย่างรวดเร็ว โดย g(n) หมายถึงค่าใช้จ่าย (Cost) จริงที่ใช้ในการเดินทางจากโหนดเริ่มต้น s จนไปถึงโหนดปัจจุบัน n (ซึ่งค่าใช้จ่ายในที่นี้อาจเป็นระยะทาง เวลา ฯลฯ) และค่า h(n) หมายถึงค่าใช้จ่ายประมาณการที่ใช้ในการเดินทางจากโหนดปัจจุบัน n ไปถึงโหนดโหนดเป้าหมาย g ดังรูปที่ 2



รูปที่ 2. ส่วนประกอบของ Heuristic function f(n)

อัลกอริธึมในการค้นหาแบบ A* จะใช้ f(n) ในการตัดสินใจเลือกโหนดของกราฟเริ่มจากโหนดเริ่มต้น s เพื่อค้นหาโหนดถูกในแต่ละขั้น ซึ่งแต่ละขั้นจะทำให้เกิดเส้นทางที่ยังค้นหาไม่จบ เส้นทางเหล่านี้มีโหนดปลาย ซึ่งแต่ละอันถือว่าเป็นโหนด n ของเส้นทางนั้นๆ ถ้าเส้นทางใดในบรรดาเส้นทางเหล่านี้ให้ค่า f(n) ของโหนด n ของตนเป็นค่าที่น้อยที่สุดในกลุ่มโหนด n

อื่นๆ อัลกอริธึมจะเลือกขยายขอบเขตการค้นหาต่อในที่เส้นทางนั้น โดยขยายการค้นหาที่โหนด n ของเส้นที่เลือกลงไปยังโหนดลูกของมัน ถ้ามีโหนดลูกตัวใดปรากฏในเส้นทางที่กำลังค้นหาค้างอยู่ก็จะถูกตัดทิ้งไป การค้นหาโดย A* จะทำทีละขั้น เช่นนี้ซ้ำแล้วซ้ำอีกไปเรื่อยๆจนกระทั่งพบโหนดเป้าหมาย g เส้นทางที่เกิดขึ้นที่เชื่อมจาก s ไป g จะถือว่าเป็นเส้นทางที่ดีที่สุดที่เลือกโดยวิธี A*

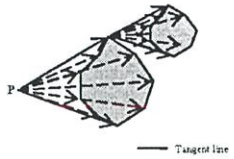
วิธี A* จะสามารถเลือกเส้นทางที่ดีที่สุด (Optimum solution) ได้ก็ต่อเมื่อฟังก์ชั่น g(n) ซึ่งเป็นการคิดค่าใช้จ่ายจากโหนดเริ่มต้น s ถึงโหนดปัจจุบัน n จะต้องเป็นฟังก์ชั่นที่คิดค่าออกมาแล้วไม่ต่ำกว่าค่าใช้จ่ายจริงของเส้นทางที่สั้นที่สุดที่เชื่อมจาก s ถึง n ส่วน h(n) ซึ่งเป็นการประมาณการค่าใช้จ่ายจากโหนดปลาย n ถึงโหนดเป้าหมาย g จะต้องเป็นการประมาณค่าที่ไม่เกินจากค่าใช้จ่ายจริงของเส้นทางที่สั้นที่สุดที่เชื่อมจาก n ถึง g [8]

ปัญหาของการค้นหาแบบ A* ประการที่หนึ่งคือในกรณีที่สเปซเส้นทางการค้นหาที่เป็นไปได้ทั้งหมดมีลักษณะอนันต์จะทำให้ A* จึงไม่สามารถใช้งานได้ ประการที่สอง ถ้าโหนดที่อยู่ในกราฟมีจำนวนมาก การค้นหาเส้นทางโดยใช้วิธี A* ก็จะใช้เวลานานมาก

เนื่องจากสเปซการวางแผนเส้นทางบินของ UAV มีลักษณะเป็นอนันต์ทั้งนั้นการนำเอา A* มาใช้งานจึงต้องปรับปรุงสเปซของการค้นหาให้เป็นลักษณะจำกัด (Finite) ก่อนดังที่เราได้ดำเนินการและนิยามไว้แล้วในหัวข้อที่ 3

4.2. การวางแผนเส้นทางการบินที่นำเสนอ

การวางแผนเส้นทางแบบใหม่ที่นำเสนอเป็นการเลี่ยงสิ่งกีดขวางโดยพิจารณาหาเส้นตรงที่สัมผัสกับจุดยอดต่างๆของรูปหลายเหลี่ยมและไม่ตัดผ่านสิ่งกีดขวางด้วย ดังแสดงในรูปที่ 3 หรืออาจกล่าวอีกนัยหนึ่งว่า เป็นการวางแผนเส้นทางบนกราฟที่เกิดจากการเชื่อมจุดยอดหนึ่งๆของสิ่งกีดขวางหนึ่งไปยังจุดยอดหนึ่งๆของอีกสิ่งกีดขวางหนึ่ง และใช้กราฟเหล่านี้สำหรับการค้นหาเส้นทาง



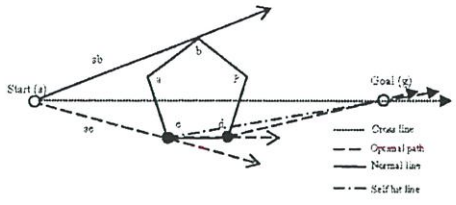
รูปที่ 3. การหาเส้นสัมผัสรูปหลายเหลี่ยม

จากนี้ไปเราจะใช้รูปที่ 4 เพื่ออธิบายวิธีการวางแผนเส้นทางเพื่อเลี่ยงสิ่งกีดขวางโดยอาศัยการลากเส้นเชื่อมจุดยอดของสิ่งกีดขวางอันหนึ่งไปยังจุดยอดของสิ่งกีดขวางอันถัดไปโดยทำทีละครั้งจนสามารถเลี่ยงสิ่งกีดขวางไปยังโหนดเป้าหมายได้

พิจารณาเส้นตรงลากเชื่อมจากโหนดเริ่มต้น s (โหนดเริ่มต้นของ UAV) ไปยังจุดยอดต่างๆของสิ่งกีดขวางรวมทั้งโหนดเป้าหมาย g (โหนดเป้าหมายของ UAV) ด้วย ที่แต่ละสิ่งกีดขวางจะได้เส้น sb และ se จากนั้นเลือกโหนดที่สั้นที่สุดด้วยวิธีการแบบ A* โดยกำหนดให้ h(n) เป็นค่าความยาวของเส้นตรงที่เชื่อมโหนดที่พิจารณาไปยังโหนดเป้าหมายโดยไม่พิจารณาว่าเส้นตรงจะตัดสิ่งกีดขวาง (Cross line) หรือไม่ และให้ g(n) เป็นระยะทางจริงจากโหนดเริ่มต้นมาถึงโหนดปัจจุบัน สมมุติให้ f(e) < f(b) ดังนั้น A* จึงเลือกโหนด e เพื่อค้นหาต่อ

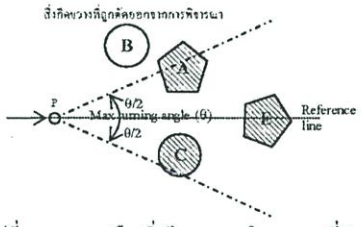
จากนั้นพิจารณาจากโหนด e มีเส้นตรง eg ลากไปยังโหนดเป้าหมายแต่ตัดผ่าน (ชน) สิ่งกีดขวางซึ่งเป็นตัวมันเองซึ่งมีโหนด e เป็นจุดยอดของสิ่งกีดขวางนั้นขอเรียกเส้นตรง eg ที่เกิดกรณีเช่นนี้ว่า "Self hit line" ในกรณีนี้ให้เลือกโหนดข้างเคียงที่ใกล้เป้าหมายมากที่สุดมาเป็นโหนดเชื่อมคือโหนด d ได้เส้นตรง ed จากนั้นนำโหนด d มาพิจารณาเพื่อค้นหาเส้นทางต่อไป

มาถึงตอนนี้ ถ้า f(d) < f(b) A* จะเลือกโหนด b เพื่อค้นหาต่อไป มิฉะนั้นจะเลือก d เพื่อค้นหาต่อไป ในกรณีนี้สมมุติให้ว่า A* เลือกโหนด d จากการค้นหาต่อไปพบว่าสามารถลากเส้นเชื่อมจากโหนด d ไปยังโหนดเป้าหมายได้โดยไม่ตัดสิ่งกีดขวางใดๆ ดังนั้นเส้นทางที่ได้จากการค้นหาคือ s → e → d → g ซึ่งเป็นเส้นทางเลี่ยงสิ่งกีดขวางจาก s ไปยัง g ที่สั้นที่สุด



รูปที่ 4. การหาเส้นสัมผัส

จากที่อธิบายนี้เป็นเฉพาะกรณีสิ่งกีดขวางหนึ่งตัว ถ้ามีสิ่งกีดขวางหลายๆตัว ก็จะทำให้เกิดกรณี e และ b (และ d ในกรณีที่เกิด Self hit line ขึ้น) ของสิ่งกีดขวางแต่ละตัว ซึ่ง A* จะต้องเลือกโหนดใดโหนดหนึ่งจากโหนดเหล่านี้ออกมา จากนั้นสร้างเส้นเชื่อมจากโหนดนี้ไปยังจุดยอดของแต่ละสิ่งกีดขวางอื่นอีกก็จะได้ e และ b (และ d ในกรณีที่เกิด Self hit line ขึ้น) ใหม่ขึ้นมาให้ A* ได้เลือกอีก และจะทำเช่นนี้ซ้ำอีกเป็นลำดับจน A* สามารถเชื่อมเส้นตรงไปถึงโหนดเป้าหมาย (Goal) ได้สำเร็จ



รูปที่ 5. แสดงการเลือกสิ่งกีดขวางภายในระยะมุมที่กำหนด

ในขณะที่ A* ขยายการค้นหาต่อ เรายังได้นำเอาข้อมูลลักษณะการบินของ UAV ที่เป็นเรื่องมุมเลี้ยวสูงสุดมาใช้ในการพิจารณาคัดเลือกกลุ่มสิ่งกีดขวางที่จะนำมาใช้ค้นหาเส้นทางต่อ ทำให้เราสามารถตัดสิ่งกีดขวางบางส่วนที่อยู่นอกขอบเขตของมุมเลี้ยวสูงสุด (กำหนดให้เป็นค่า θ) ออกไปได้ ดังแสดงในรูปที่ 5 ในที่นี้มุม θ จะ ถูกแบ่งครึ่งโดยเส้นตรง Reference line ซึ่งเป็นเส้นตรงแนวเดียวกับเส้นเวคเตอร์ที่ลากจากโหนดก่อนหน้ามายังโหนดที่กำลังพิจารณา ซึ่งค่า Maximum turning angle นี้ขึ้นอยู่กับลักษณะการบินของ UAV แต่ละประเภท

ยังมีอีกหนึ่งเงื่อนไขที่ใช้ในการตัดโหนดออกจากการพิจารณาของ A* คือเราใช้เงื่อนไขระยะทางไกลสุดในการบิน ซึ่งอธิบายได้ดังนี้ ถ้าโหนดที่กำลังพิจารณามีระยะทางของ Path ถึงโหนดเริ่มต้นมากกว่าระยะทางไกลสุดในการบินซึ่งกำหนดให้มีค่าเป็น L แล้วเราสามารถตัดโหนดนี้ออกจากการพิจารณาของ A* ได้

มาถึง ณ ตอนนี้ เมื่อนำเอาลักษณะการบินทั้งสองเงื่อนไขมาพิจารณาร่วมด้วยในการวางแผนเส้นทาง เราสามารถสรุปขั้นตอนการค้นหาเส้นทางที่สมบูรณ์ได้ดังนี้

1. กำหนดให้โหนดเริ่มต้นเสมือนเป็นจุดยอดของสิ่งกีดขวาง และเริ่มพิจารณาจากโหนดเริ่มต้นเป็นโหนดแรก

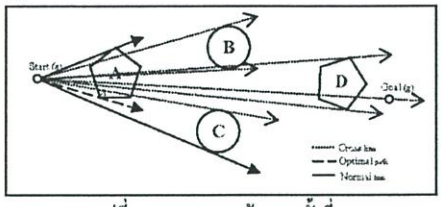
2. หาเส้นตรงที่ลากจากโหนดที่เลือกไปยังจุดยอดของสิ่งกีดขวางอื่นๆทั้งหมดโดยไม่ตัดผ่านสิ่งกีดขวางใดและอยู่ในช่วงองศาไม่เกินค่า θ แล้วให้เก็บไว้ในลิสท์ และถ้ามี Self hit line อยู่ด้วยให้เลือกโหนดใกล้เคียงบนสิ่งกีดขวางเดียวกันที่ใกล้เป้าหมายมากที่สุดนำไปรวมไว้ในลิสท์ของโหนดที่จะเป็นตัวเลือกในการค้นหาด้วย

3. เลือกโหนดที่มีค่า f(n) น้อยที่สุดจากลิสท์มาพิจารณาและจะหยุดการทำงานก็ต่อเมื่อ 1) โหนดที่เลือกมาเป็นโหนดเป้าหมาย 2) Path ที่เกิดจากโหนดที่เลือกนั้นมีระยะทางมากเกินกว่า L 3) ไม่มีโหนดที่จะเป็นตัวเลือกในการค้นหาให้พิจารณาอีก นอกเหนือจากนี้กลับไปทำขั้นตอนที่ 2 ซ้ำอีก

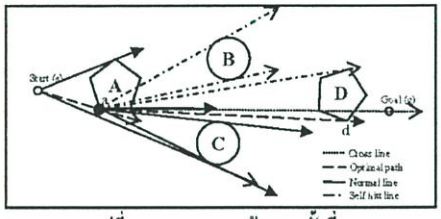
จากขั้นตอนการทำงานข้างต้น เราสามารถแสดงขั้นตอนการค้นหาเส้นทางได้ ดังรูปที่ 6 7 และ 8 โดยเรากำหนดให้ θ = 60 องศาและ L=500

มีการทำงานวนซ้ำทั้งหมด 3 รอบโดยแต่ละรูปแสดงการทำงานในแต่ละรอบ เริ่มจากรูปที่ 6 โดยเริ่มพิจารณาจากโหนดเริ่มต้น s ไปยังสิ่งกีดขวางต่างๆรวมทั้งโหนดเป้าหมาย g ด้วย เมื่อพิจารณาเส้นทั้งหมดที่สัมผัสสิ่งกีดขวาง เราสามารถตัดเส้นตรงที่ตัดผ่านสิ่งกีดขวางออกจะเหลือเพียง 3 เส้นโดยอาศัยเงื่อนไข 1) มีมุมเลี้ยวสูงสุด (θ) 2) เป็น Cross line จากนั้นเลือกโหนดทั้ง 3 โดยวิธีการ A* จะได้โหนด a ของสิ่งกีดขวาง A ที่มีเส้นประเป็นเส้นสัมผัสโหนดมาพิจารณาเป็นโหนดต่อไป เหมือนเป็นโหนดเริ่มต้น ในขั้นตอนนี้เราสามารถไม่นำสิ่งกีดขวาง A พิจารณาด้วยเนื่องจากมีค่ามุมที่อ้างอิงกับ Reference line มีค่ามากกว่า θ/2 ดังรูปที่ 7 เมื่อพิจารณาโหนดที่มีค่า f(n) น้อยที่สุดจะได้โหนด d ของสิ่งกีดขวาง D มาพิจารณาโดย

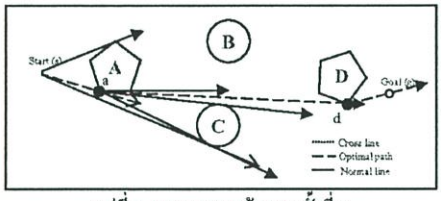
สามารถตัดสิ่งกีดขวาง A, B และ C ออกจากการพิจารณาเพราะมีค่ามุมที่อ้างอิงกับ Reference line มากกว่า θ/2 เช่นเดียวกับการตัดสิ่งกีดขวาง A ออกจากการพิจารณาดังที่ได้ทำในขั้นตอนนี้แล้ว ดังในรูปที่ 8 จากนี้เราสามารถลากเส้นตรงจากโหนด d ของสิ่งกีดขวาง D ไปยังโหนดเป้าหมาย g ได้ แล้วจึงสิ้นสุดการค้นหาเส้นทางและได้เส้นทางที่สั้นที่สุดที่ค้นพบคือ s → a → d → g



รูปที่ 6. การวางแผนเส้นทางครั้งที่ 1



รูปที่ 7. การวางแผนเส้นทางครั้งที่ 2



รูปที่ 8. การวางแผนเส้นทางครั้งที่ 3

จากขั้นตอนการวางแผนเส้นทางทั้ง 3 ขั้นตอนข้างต้น เราสามารถเขียนอยู่ในรูปอัลกอริธึมโดยใช้ syntax ของภาษา Python ซึ่งประกอบด้วยฟังก์ชั่นหลักและฟังก์ชั่นย่อยต่างๆ ดังนี้

1. ฟังก์ชั่นหลัก SEARCH(initial, goal) ทำหน้าที่เลือกโหนดที่ให้ค่า f(n) น้อยที่สุดแล้วนำมาพิจารณา ดังในรูปที่ 8 การค้นหาจะหยุดได้ใน 3 กรณีคือ 1) พบโหนดเป้าหมายที่แสดงว่าได้เส้นทางที่ต้องการแล้ว (บรรทัดที่ 13-14) 2) Path ที่เกิดจากโหนดที่กำลังพิจารณามีเส้นทางเกินระยะ Maximum distance (บรรทัดที่ 11-12) หรือ 3) ไม่มีเส้นทางเหลือที่จะค้นหาอีกแล้ว (บรรทัดที่ 9)

```
01 global obstacle_list = []
02 global max_turn_angle
03 global max_distance
04
05 def SEARCH(initial, goal):
06   obstacle_list.append(goal)
07   list = []
08   list.append( initial )
09   while list.is_empty():
10     node = list.lowest_node()
11     if cost(node) > max_distance:
12       return None
13     if node == goal:
14       return node
15     EXTEND(list, EXPAND(node))
16   return None
```

รูปที่ 9. ฟังก์ชั่น SEARCH

2. ฟังก์ชั่น EXTEND(list1, list2) ทำหน้าที่เก็บโหนดโดยเรียงลำดับจากที่ให้ค่า f(n) น้อยไปหามาก

3. ฟังก์ชั่น EXPAND(node) เป็นฟังก์ชั่นที่สำคัญมากทำหน้าที่หาโหนดของแต่ละสิ่งกีดขวางแล้วมาเลือกโหนดที่ลากเส้นเชื่อมแล้วไม่ตัดผ่านสิ่งกีดขวาง (บรรทัดที่ 7-8) และเลือกหาโหนดใกล้เคียงในกรณีที่เป็น Self hit line ออกมาดังแสดงในรูปที่ 10

```
01 def EXPEND(node):
02   nodes = []
03   for obs in obstacle_list:
04     points = FIND_VERTEX( obs, node)
05     for point in points:
06       if degree(point) < max_turn_angle:
07         if LINE(node, point)==NormalLine:
08           nodes.append(point)
09         elif LINE(node, point)==SelfHitLine:
10           nodes.append(
11                       NEAR_VERTEX(obs, point))
12   return nodes
```

รูปที่ 10. ฟังก์ชั่น EXTEND

4. ฟังก์ชั่น FIND_VERTEX(obs, point) ทำหน้าที่หาโหนดของสิ่งกีดขวางโดยที่เส้นตรงที่ลากผ่านโหนดนี้จะไม่ตัดผ่านสิ่งกีดขวาง

```
01 def FIND_VERTEX(obs, point):
02   points = []
03   for v in vertex of obs:
04     if LINE(point,v) <> Cross:
05       points.append(points)
06   return points
```

รูปที่ 11. ฟังก์ชั่น FIND_VERTEX

5. ฟังก์ชั่น LINE(p1, p2) ใช้ตรวจสอบว่าเส้นตรงจากโหนด p1 ไปยัง p2 ตัดผ่านสิ่งกีดขวางหรือไม่โดยให้ผลลัพธ์ที่เป็นไปได้ 3 ค่าคือ NormalLine, CrossLine และ SelfHitLine

```
01 def LINE(p1, p2):
02   for obs in obstacles_list:
03     if cross(obs):
04       if p1 or p2 is a vertex of obs:
05         return SelfHitLine
06       else:
07         return CrossLine
08   return Normal
```

รูปที่ 12. ฟังก์ชั่น LINE

5. ผลการทดลอง

เราได้นำการวางแผนเส้นทางวิธีการใหม่ที่นำเสนอไปทดลองเปรียบเทียบประสิทธิภาพกับการค้นหาเส้นทางแบบ Best first search ซึ่งให้เส้นทางออกมาแสดงด้วยเส้นบาง ส่วนเส้นทางที่ได้จาก A* search จะแสดงด้วยเส้นทึบ (ดูรูปที่ 13 14 15 และ 16) ซึ่งผลการทดลองทั้งหมดนี้ได้มาจากการรันโปรแกรมที่เขียนด้วยภาษา Python บนเครื่องคอมพิวเตอร์ใช้ซีพียู Celeron-M ความเร็ว 2.0 GHz หน่วยความจำ 512 MB ภายใต้สเปซการค้นหาเดียวกัน

จากผลการทดลองในตารางที่ 1 เป็นการเปรียบเทียบโดยให้ θ =60 และ L=2000 และสิ่งกีดขวางมีจำนวน 10 ตัวและ 40 ตัว ตามลำดับ การค้นหาแบบ A* ทำงานได้รวดเร็วกว่าและเส้นทางที่ได้มีระยะทางสั้นกว่า Best first search ดังแสดงเส้นทางผลลัพธ์ในรูปที่ 13 และ 14

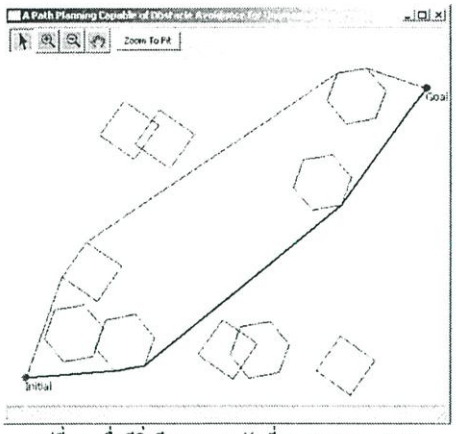
ตารางที่ 1. ตารางผลการทดลอง ที่ θ =60 และ L=2000

Graph Search	10 Obstacles		40 Obstacles	
	Time	Distance	Time	Distance
Best first	0.185290	1088.610148	0.367487	1143.196721
A*	0.081164	1037.227637	0.249005	996.024246

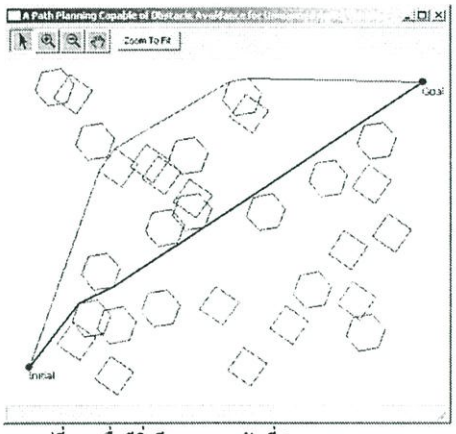
ส่วนในตารางที่ 2 เปรียบเทียบโดยให้จำนวนสิ่งกีดขวางและให้ θ คงที่แต่ปรับค่าระยะไกลสุดในการบินให้แตกต่างกัน จะเห็นได้ว่าเมื่อกำหนดให้ L มีค่า 1100 ซึ่งมีไม่เพียงพอทำให้การค้นหาแบบ Best first search ไม่พบเส้นทาง ทั้งนี้เนื่องมาจากเส้นทางที่เลือกโดย Best first search มีระยะทางมากกว่าค่า Maximum distance จึงถูกตัดออกจากการพิจารณาของการค้นหาเส้นทางโดยเงื่อนไขของ Maximum distance ดังแสดงผลของเส้นทางผลลัพธ์ในรูปที่ 15 และ 16

ตารางที่ 2. ตารางผลการทดลองกับ 40 สิ่งกีดขวาง ที่ θ =90

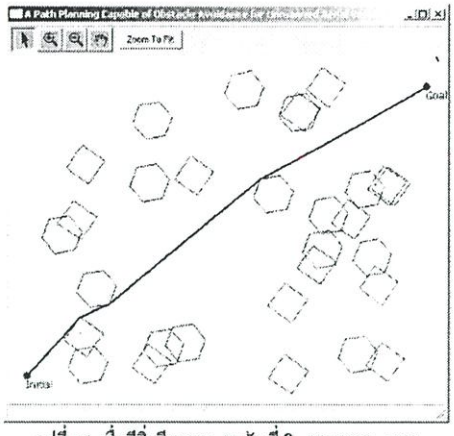
Max Distance	Best first		A*	
	Time	Distance	Time	Distance
1100	-	-	0.223328	997.053639
1500	0.612898	1167.142752	0.315880	992.883542



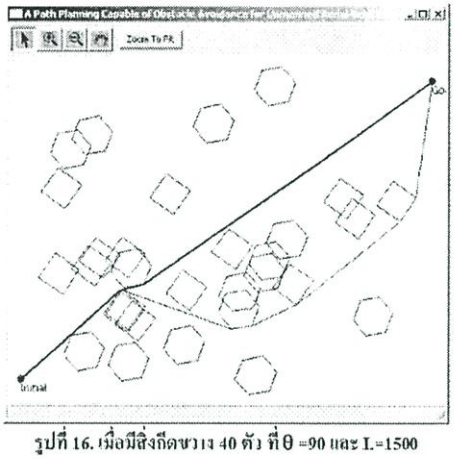
รูปที่ 13. เมื่อมีสิ่งกีดขวาง 10 ตัว ที่ θ =60 และ L=2000



รูปที่ 14. เมื่อมีสิ่งกีดขวาง 40 ตัว ที่ θ =60 และ L=2000



รูปที่ 15. เมื่อมีสิ่งกีดขวาง 40 ตัว ที่ θ =90 และ L=1100

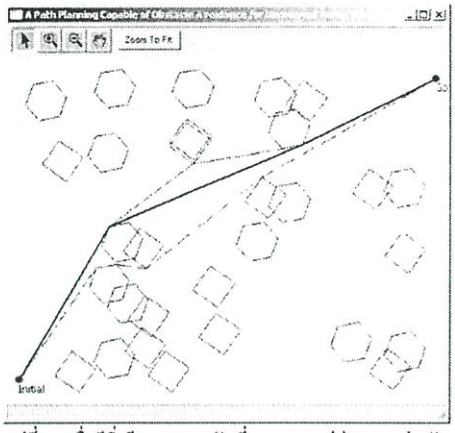


รูปที่ 16. เมื่อมีสิ่งกีดขวาง 40 ตัว ที่ θ =90 และ L=1500

สุดท้ายเป็นผลการทดลองเปรียบเทียบการคนหา A* ที่กำหนดให้ L=1500 และจำนวนสิ่งกีดขวาง 40 ตัวแต่ปรับค่ามุมเลี้ยวสูงสุดมีค่าแตกต่างๆ โดยเริ่มจากไม่นำค่า θ มาพิจารณาซึ่งแทนด้วยค่า 360 องศาซึ่งเส้นทางที่ได้แสดงด้วยเส้นประนอกจากนั้นเป็นทั้งกรณีที่กำหนดให้ θ มีค่าเป็น 90 และ 60 ซึ่งเส้นทางที่ได้แสดงด้วยเส้นทึบและเส้นบางตามลำดับ ดังแสดงในรูปที่ 17

ตารางที่ 3. ตารางผลการทดลอง A* และ L=1500

Maximum turning angel	วิธีการค้นหาแบบA*ที่มีค่า θ ต่างๆ	
	Time(s)	Distance
360	0.699733	1026.623659
90	0.275807	1030.356080
60	0.450999	1041.542323



รูปที่ 17. เมื่อมีสิ่งกีดขวาง 40 ตัว ที่ L=1500 แต่ค่าθ แตกต่างกัน

ผลการทดลองที่ปรากฏในตารางที่ 3 แสดงให้เห็นว่า การไม่นำ θ มาพิจารณาด้วยทำให้ได้เส้นทางที่มีระยะทางสั้นที่สุดแต่ใช้เวลาในการคำนวณมากที่สุด เนื่องจาก A* ต้องพิจารณาเส้นทางที่เป็นได้ทั้งหมด โดยไม่ถูกจำกัดโดย Maximum turning angle

ในกรณีที่นำ θ มาพิจารณาโดยกำหนดให้มีค่า 90 องศามีผลทำให้ใช้เวลาในการประมวลผลลดลงเพราะเส้นทางบางส่วนได้ถูกตัดออกจากการพิจารณา จึงทำให้เส้นทางที่ได้มีระยะทางมากกว่าการไม่นำ θ มาพิจารณา นั่นคือไม่ได้เส้นทางที่สั้นที่สุดจริง แต่ได้เส้นทางที่สั้นที่สุดภายใต้เงื่อนไขของ Maximum turning angle

แต่เมื่อปรับให้ค่า θ ลดลงเป็น 60 องศา เวลาในการค้นหาไม่น้อยกว่ากรณี θ =90 เนื่องจากเมื่อการค้นหาเส้นทางที่ผ่านไปยังโหนดเป้าหมายแล้วกลับไปมีเส้นทางที่ไปยังโหนดเป้าหมาย

ภายในขอบเขตของ θ จึงทำให้ต้องไปพิจารณาเส้นทางอื่นที่เป็นไปได้ต่อไป ทำให้ได้เส้นทางที่ใกล้มากขึ้นด้วย แต่ยังใช้เวลาน้อยกว่ากรณีที่ไม่นำ θ มาพิจารณา

6. เปรียบเทียบกับงานวิจัยอื่น

เมื่อเปรียบเทียบกับงานวิจัยของ Szczerba [2] ที่ได้อาศัยข้อจำกัดการเคลื่อนที่ของ UAV ช่วยลดจำนวนโหนด และด้วยข้อจำกัด Minimum Route Leg Length [2] ทำให้การค้นหาเส้นทางในกรณีที่ระยะจากโหนดที่พิจารณาห่างจากสิ่งกีดขวางมากกว่าค่า Minimum Route Leg Length วิธีนี้จะต้องพิจารณาหลายโหนดจึงจะสามารถเลี่ยงสิ่งกีดขวางได้ซึ่งมีผลทำให้เส้นทางที่ได้ประกอบด้วยเส้นตรงสั้นๆจำนวนมาก และมีความคดเคี้ยว แต่การวางแผนเส้นทางการบินแบบใหม่ที่นำเสนอนี้ทำงานได้รวดเร็วกว่าเนื่องจากการเลี่ยงสิ่งกีดขวางหนึ่งจะพิจารณาเพียงโหนดเดียว และได้เส้นทางที่คดเคี้ยวน้อยกว่าเพราะการเลี่ยงสิ่งกีดขวางในงานวิจัยนี้ใช้เส้นตรงจำนวนน้อยกว่า

ส่วนในงานวิจัยของ [5] มีแนวคิดที่แตกต่างกันไม่มากระหว่างการพิจารณาโหนดของสิ่งกีดขวางโดยวิธีที่นำเสนอกับการพิจารณาโหนดบนโครงข่ายสามเหลี่ยมที่เสนอโดย [5] แต่วิธีค้นหาเส้นทางในงานวิจัย [5] จะทำงานช้ากว่าเนื่องจากต้องเสียเวลาไปกับการสร้างโครงข่ายสามเหลี่ยมก่อนการวางแผนค้นหาเส้นทาง

7. บทสรุป

เราได้นำเสนอวิธีการวางแผนเส้นทางบินสำหรับ UAV ในลักษณะ Real time โดยเป็นการดัดแปลงใช้วิธี A* ในกรณีสเปซการค้นหาเป็นอนันต์ โดยสามารถลดจำนวนโหนดที่ใช้พิจารณาลงด้วยวิธีต่างๆ ทำให้ A* สามารถค้นหาเส้นทางการบินได้อย่างรวดเร็ว และให้เส้นทางที่สั้นที่สุดภายใต้เงื่อนไขลักษณะการบินของ UAV ประเภทนั้นๆ

8. แนวทางการวิจัยต่อไป

ในงานวิจัยต่อไปเราจะนำเอาวิธีการนี้ไปปรับปรุงเพื่อใช้กับวิธีวางแผนเส้นทางบินของ UAV บนภูมิประเทศที่แสดงด้วยภาพสามมิติ

9. กิตติกรรมประกาศ

งานวิจัยนี้ได้รับทุนสนับสนุนการวิจัยจากสำนักงานกองทุนสนับสนุนการวิจัยและสำนักงานวิจัยและพัฒนาการทหารกลาโหม ภายใต้โครงการวิจัยอากาศยานไร้นักบิน

10. เอกสารอ้างอิง

[1] Dobbs, S., Davis, W., and Carl Lizza, "An application of heuristic search techniques to the problem of flight path generator in a military hostile environment ", *International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, pp 273 – 280, 1988.

[2] Szczerba, J., Galkowski, P., Glickstein, S., and Ternullo, N."Robust algorithm for real-time route planning", *IEEE Transactions on Aerospace and Electronic Systems*,Vol. 36, No.3 July 2000.

[3] Szczerba, J. "Threat netting foe real-time, intelligent route planners", *Proceedings on Information Decision and Control 99*, pp. 377-382, February 1999.

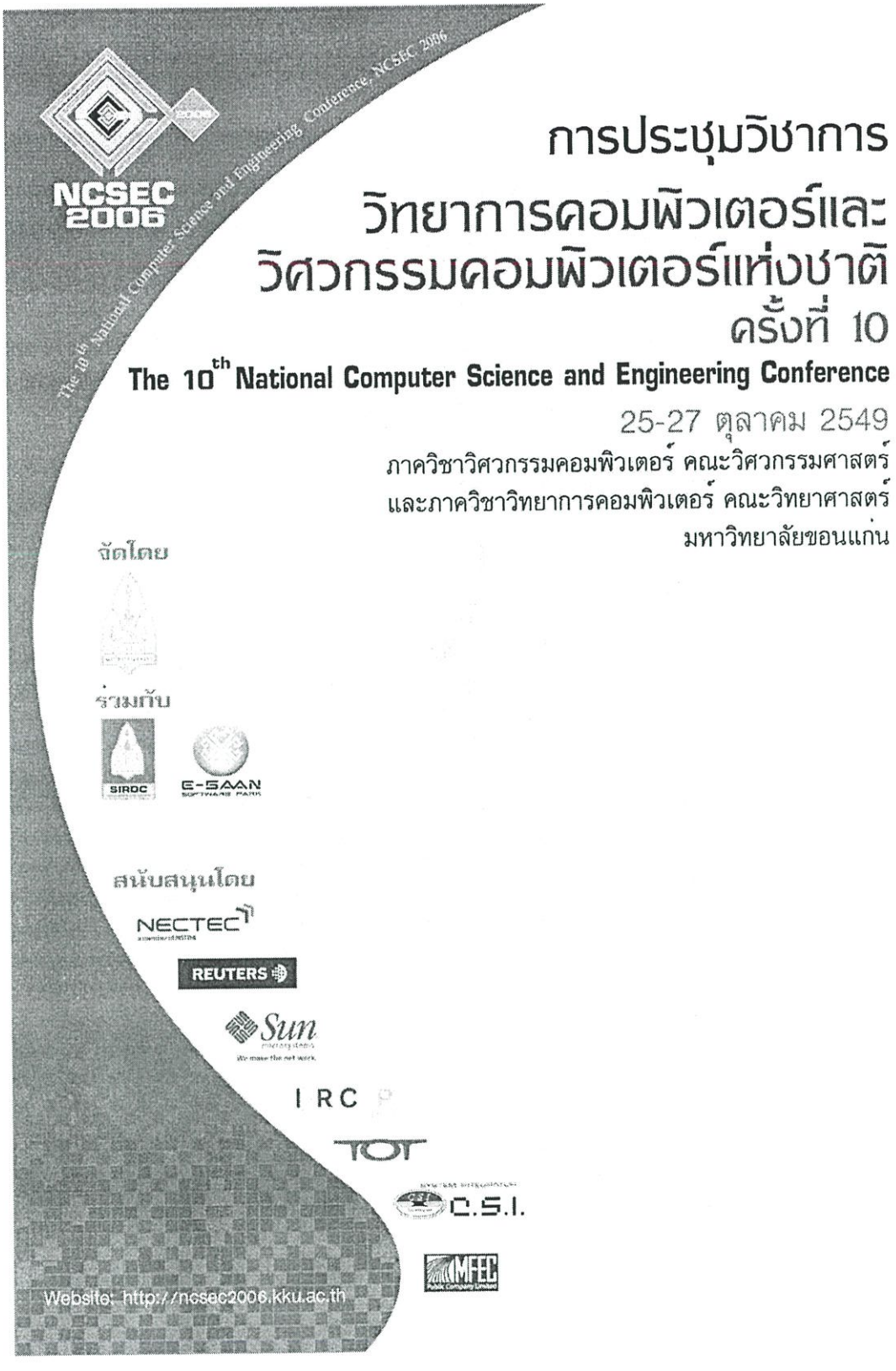
[4] Stentz, A. "Optimal and efficient path planning for partial-known environment", *Proceedings on the IEEE International Conference on Robotics and Automation*, May 1994.

[5] Yan, P., Ding, M., Zhou, C., and Zheng, C., "A path replanning algorithm based on roadmap-diagram", *Proceedings on the 5th World Congress on Intelligent Control and Automation*, Jun 15-19 2004.

[6] Rathbum, D., Kragelund, S., and Pongpunwattana, A., "An evolution based path planning algorithm for autonomous motion of UAV through uncertain environments", *Proceedings on AIAA's 1st Technical Conferences and Workshop on Unmanned Aerospace Vehicle, System, Technologies, and Operations. Portsmouth*, 2002.

[7] Hwang, Y. J., Kim, S. J., Lim, S. S., and Park, H. K., "A fast path planning by path graph optimization", *IEEE Transaction on System, Man, and Cybernetics-part a: System and Humans*, Vol. 33, No.1 January 2003.

[8] Russel, S., Norvig, P., *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2003.


NCSEC 2006
The 10th National Computer Science and Engineering Conference, NCSEC 2006
การประชุมวิชาการ
วิทยาการคอมพิวเตอร์และ
วิศวกรรมคอมพิวเตอร์แห่งชาติ
ครั้งที่ 10
The 10th National Computer Science and Engineering Conference
25-27 ตุลาคม 2549
ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์
และภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์
มหาวิทยาลัยขอนแก่น
จัดโดย
ร่วมกับ
SIRDC
E-SAAN SOFTWARE PARK
สนับสนุนโดย
NECTEC
REUTERS
Sun
IRC
TOT
C.S.I.
MFEC
Website: http://ncsec2006.kku.ac.th

National Computer Science and Engineering Conference (NCSEC2006)
25 - 27 October 2006

P22

การวางแผนเส้นทางบินที่มีจำนวนจุดหมายไม่จำกัดโดยวิธีเอสตาร์แบบค้นหาลึกลงไป

Flight Path Planning with an Unlimited Number of Goals using a Deepening A* Search

เกียรติศักดิ์ ศิวขุนทด และ วิศิษฏ์ หิรัญกิตติ
ห้องวิจัยการสื่อสารและคมนาคมชาญฉลาด ภาควิชาวิศวกรรมคอมพิวเตอร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง กรุงเทพ 10520 ประเทศไทย
E-mail: k_keattisak@hotmail.com, v_hirankitti@gmail.com

บทคัดย่อ

การวางแผนเส้นทางบินในความเป็นจริงจะต้องเลี่ยงสิ่งกีดขวางได้และอาจจะมีจุดหมายเดียว หลายจุดหมาย หรือมีจุดหมายจำนวนเป็นอนันต์ สำหรับวิธีการค้นหาเส้นทางที่สั้นที่สุดสำหรับหนึ่งจุดหมายของอากาศยานไร้นักบินได้มีการเสนอไว้ใน [1] โดยได้ประยุกต์ใช้วิธี A ซึ่งพบว่าไม่มีประสิทธิภาพสูงพอที่จะนำมาปรับใช้ในการวางแผนในกรณีที่มีจุดหมายจำนวนมากหรือเป็นอนันต์ได้ และในบางกรณีก็อาจมีจุดหมายที่คาดว่าจะไปถึงได้แต่กลับมีอุปสรรคขวางกั้นการเดินทางจนสามารถตัดออกจากการพิจารณาได้ ดังนั้นในบทความนี้จึงขอเสนอวิธีการใหม่ที่เอาชนะปัญหาดังกล่าวในการค้นหาเส้นทางบินที่สั้นที่สุดที่มีจุดหมายจำนวนไม่จำกัดเป็นอนันต์สำหรับอากาศยานไร้นักบินด้วยวิธี A* แบบค้นหาลึกลงไป โดยเราจะแสดงให้เห็นว่าวิธีการนี้ให้คำตอบที่เป็นการเลือกจุดหมายที่ดีที่สุดได้อย่างมีประสิทธิภาพยิ่ง*

Abstract

Flight planning, to be practical, must be able to avoid obstacles while searching for a single goal, multiple goals and even infinite goals. An approach based-on A was proposed in [1] for an unmanned aerial vehicle to find the shortest path for a single goal. Apparently, this approach is not efficient enough to apply for searching of a large number or even infinite number of goals. Additionally, in some circumstances some candidate goals may be almost completely blocked by obstacles and therefore can be ignored during the search. To overcome such problems, in this paper we shall propose a new approach for path planning with an infinite number of goals by using a deepening A* search algorithm. We will demonstrate that this approach is very efficient to give an optimum solution to the problems.*

Key words: Path planning, A* Algorithm, Multi-goal search, Unmanned aerial vehicles

1. บทนำ

การวางแผนเส้นทางบินของอากาศยานเป็นปัญหาที่มีความสำคัญเพราะมีผลต่อความปลอดภัยในการเดินทาง โดยมีปัญหาหลัก คือ ขนาดของสเปซที่มีไม่จำกัดอันเนื่องมาจากจำนวนโหนดที่ต้องพิจารณามีจำนวนมากมายนับไม่ถ้วน เพราะเมื่ออากาศยานบินจากจุดเริ่มต้นหลบเลี่ยงสิ่งกีดขวางไปยังจุดหมายผ่านไปในสภาพแวดล้อมจริง ณ ตำแหน่งใดๆ มันสามารถเคลื่อนที่ไปได้หลายทิศทางและสามารถเลี้ยวได้ในหลายตำแหน่งเช่นเดียวกัน ทำให้เส้นทางบินของอากาศยานมีความเป็นไปได้มากมายนับไม่ถ้วนหรือเป็นอนันต์นั่นเอง จึงมีผลทำให้การวางแผนเส้นทางโดยอัลกอริธึมค้นหาที่ค้นหาเส้นทางบนสเปซที่มีขอบเขตจำกัด ดังเช่น วิธี A* ไม่สามารถนำมาใช้กับกรณีนี้ได้

NCSEC 2006

ในงานวิจัยของ [1] ได้มีการนำเอาวิธี A* มาดัดแปลงเพื่อแก้ปัญหานี้โดยอาศัยข้อมูลลักษณะของการบินของอากาศยานไร้นักบินหรือ UAV (Unmanned Aerial Vehicle) มาใช้ในการลดจำนวนเส้นทางบินที่เป็นได้ที่ต้องพิจารณาลง มีผลทำให้วิธี A* ใช้งานได้ดี อย่างไรก็ตามการค้นหาเส้นทางวิธีนี้ออกแบบมาสำหรับค้นหาเส้นทางไปยังจุดหมายเดียวเท่านั้น แต่การวางแผนเส้นทางบินในความเป็นจริงอาจมีจุดหมายจำนวนมากหรือนับไม่ถ้วน เช่นในกรณีที่มีเหตุการณ์ผิดปกติซึ่งอากาศยานจำเป็นจะต้องลงจอดฉุกเฉิน และมีสถานีที่สามารถลงจอด (จุดหมาย) ได้หลายแห่งหรือลงอาจจอดบนพื้นที่ใดๆก็ได้ (จำนวนจุดหมายเป็นอนันต์) ซึ่งวิธีที่นำเสนอใน [1] จะไม่สามารถนำมาแก้ไขปัญหาเหล่านี้ได้

มีหลายบทความวิจัยได้นำเสนอวิธีการแก้ปัญหานี้ ได้แก่ [5] ใช้วิธีการวางแผนเส้นทางแบบ 2 ทิศทาง (bi-directional) สำหรับทุกจุดหมายโดยเลือกจุดหมายที่เหมาะสมมาทีละจุดหมายในการค้นหาในแต่ละรอบ แล้วทำซ้ำใน 2 ทิศทาง จนกว่าจะได้คำตอบ ส่วนจะได้คำตอบที่ดีที่สุดหรือไม่ ก็ขึ้นอยู่กับวิธีการที่เลือกมาใช้ในการเลือกค้นหาในแต่ละรอบนั้นๆ ส่วน [6] เสนอวิธีการแก้ปัญหา MTP (Multi-goal path planning) ซึ่งเป็นปัญหาที่คล้ายกับ TSP (Traveling Salesman Problem) แต่เส้นทางของปัญหา MTP เป็นเส้นทางที่ผ่านทุกจุดหมายแต่ไม่กลับไปยังจุดเริ่มต้น และในการแก้ปัญหา MTP มีขั้นตอนหนึ่งที่เรียกว่า MTP control ได้เสนอการเลือกจุดหมายด้วยวิธีการหรืออัลกอริธึมแบบต่างๆ เพื่อสร้างเส้นทางที่เหมาะสม เราพบว่าวิธีทั้งสองเป็นการนำการวางแผนเส้นทางไปประยุกต์ใช้กับเคลื่อนที่ของหุ่นยนต์ในโรงงานอุตสาหกรรม ซึ่งมีจำนวนจุดหมายจำกัด จึงไม่สามารถนำมาใช้แก้ปัญหาในกรณีที่มีจำนวนจุดหมายมากมายเป็นอนันต์ได้

ในบทความวิจัยนี้ เราขอเสนอวิธีการใหม่ในการวางแผนเส้นทางบินที่สั้นที่สุดที่มีจุดหมายจำนวนไม่จำกัดสำหรับ UAV ด้วยวิธี A* แบบค้นหาลึกลงไปซึ่งสามารถค้นหาจุดหมายที่ดีที่สุดจากจุดหมายจำนวนมากมายเป็นอนันต์ได้

เนื้อหาในบทความนี้จะเริ่มจากหัวข้อที่ 2 ซึ่งอธิบายถึงข้อกำหนดของสเปซเส้นทางที่เป็นไปได้ที่ใช้ในการวางแผนส่วนหัวข้อที่ 3 จะอธิบายถึงวิธีการวางแผนเส้นทางบินสำหรับอากาศยานไร้นักบินในกรณีที่มีจุดหมายเดียว เพื่อเป็นพื้นฐานในการทำความเข้าใจการวางแผนเส้นทางในกรณีที่มีหลายจุดหมาย หัวข้อที่ 4 กล่าวถึงปัญหาความซับซ้อนในการวางแผนเส้นทางในกรณีที่มีจุดหมายจำนวนมากหรือเป็นจำนวนอนันต์ หัวข้อที่ 5 อธิบายถึงวิธีการที่นำเสนอ จากนั้นเป็นผลการทดลองในหัวข้อที่ 6 ตามมาด้วยการวิเคราะห์อัลกอริธึมและบทสรุป

2. สเปซของการวางแผนเส้นทางบินของ UAV

ในหัวข้อนี้จะขอนิยามสเปซในการวางแผนเส้นทางบินของ UAV จากจุดเริ่มต้นไปยังจุดหมายที่ยังไม่ได้เลี่ยงสิ่งกีดขวาง

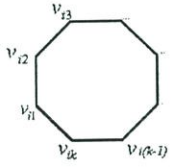
เริ่มแรกเราจะให้สเปซของปัญหาเป็นกราฟที่ประกอบด้วยจุดยอดของสิ่งกีดขวางทั้งหมด ซึ่งสิ่งกีดขวางเหล่านี้แท้ที่จริงแล้วเป็นกรอบล้อมรอบสิ่งกีดขวางจริงเพื่อให้เส้นทางบินอยู่ในระยะที่ปลอดภัยจากการชนของ UAV กับสิ่งกีดขวางจริง โดยสิ่งกีดขวาง (เสมือน) นี้มีลักษณะเป็นรูปปิดหลายเหลี่ยมสองมิติ (Polygon) ดังในรูปที่ 1 ซึ่งสร้างจากการลากเส้นตรงเชื่อมจุดยอดแต่ละจุดของรูปปิด ดังนั้นสิ่งกีดขวางใดๆสามารถสร้างจากการลากเส้นตรงเชื่อมต่อชุดของจุดยอด เมื่อ k เป็นจำนวนจุดยอด โดยมีคู่ลำดับ (v_{ij}, v_{ij+1}) ซึ่ง $1 \le j < k$ ตามด้วยคู่ลำดับ (v_{ik}, v_{i1}) เราจึงขอกำหนดให้สิ่งกีดขวาง R_i ใดๆ แทนด้วย

$$R_i = v_{i1}, v_{i2}, \ldots, v_{i(k-1)}, v_{ik} \quad (1)$$

โดยที่ $1 \le i < n$ และ n เป็นจำนวนสิ่งกีดขวางทั้งหมด

นอกจากนี้จุดยอดทั้งหมดของแต่ละสิ่งกีดขวางสามารถเขียนแทนในรูปของเซตคือ

$$O_i = \{v_{i1}, v_{i2}, \ldots, v_{i(k-1)}, v_{ik}\} \quad (2)$$



รูปที่ 1. แสดงสิ่งกีดขวาง

กราฟของโหนดที่ใช้ในการค้นหาเส้นทางโดยอัลกอริธึมวางแผนเส้นทางประกอบด้วยเซตของจุดยอด (Vertex) ของสิ่ง

กีดขวาง ซึ่งถือว่าเป็นโหนด และเซตของเส้นตรง (Edge) ที่เชื่อมระหว่างโหนด หรืออาจเขียนได้เป็น $G = (V, E)$ โดยมี V เป็นเซตของโหนดซึ่งเป็นผลมาจากการยูเนียนของเซตของโหนดของสิ่งกีดขวางทั้งหมด n ตัวกับเซตของโหนดเริ่มต้น s และเซตของจุดหมายทั้งหมด m จุดหมายซึ่งแทนด้วย $GOALS$ ดังนี้

$$V = \bigcup_{i=1}^{n} O_i \cup \{s\} \cup GOALS \quad (3)$$

$$GOALS = \{g_1, g_2, \ldots, g_m\} \quad (4)$$

E เป็นเซตของเส้นตรงซึ่งเป็นคู่ของโหนด (u, v) เมื่อ $u \in V$ และ $v \in V$ โดยที่ $u \neq v$

ดังนั้นเราสามารถหาสเปซของเส้นทางที่เป็นไปได้ทั้งหมดจากกราฟ G ส่วนเส้นทางในกราฟ G เกิดจากการนำเส้นตรงที่เป็นสมาชิกของ E มาต่อกัน

สเปซของกราฟที่นิยามข้างต้นใช้แทนเส้นทางที่เป็นไปได้ทั้งหมดซึ่งได้ลดจำนวนเส้นทางที่เป็นไปได้ทั้งหมดจากที่เป็นอนันต์ให้เหลือเป็นจำนวนที่จำกัด ซึ่งจากนี้ไปเราจะนำสเปซในความหมายนี้ไปใช้ในการอธิบายส่วนต่อๆไป

เพื่อเป็นการลดขนาดของสเปซลงไปอีก [1] ได้นำเสนอให้นำข้อมูลลักษณะการเคลื่อนที่ของ UAV มาช่วยในการลดจำนวนโหนดที่ต้องพิจารณาลง โดยอาศัยมุมเลี้ยวสูงสุดที่ UAV จะสามารถทำได้และระยะทางบินที่ไกลที่สุดที่ทำได้ด้วยข้อจำกัดของปริมาณน้ำมันเชื้อเพลิงที่ UAV สามารถบรรทุกได้

3. การวางแผนเส้นทางบินที่มีจุดหมายเดียวโดยวิธี A*

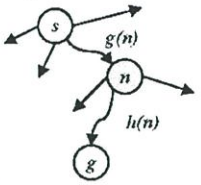
เพื่อสร้างความเข้าใจในวิธีการวางแผนเส้นทางบินสำหรับ UAV ในกรณีหลายจุดหมายโดยวิธี A* วิธีหนึ่งที่เราจะนำเสนอในหัวข้อ 5 ก่อนอื่นจะขออธิบายถึงวิธีการค้นหาแบบ A* แล้วแสดงการนำไปประยุกต์ใช้ในการวางแผนเส้นทางบินสำหรับกรณีหนึ่งจุดหมาย จากนั้นจะชี้ให้เห็นถึงข้อจำกัดและนำไปสู่การพัฒนาวิธีการวางแผนเส้นทางสำหรับหลายๆ จุดหมาย

3.1. การค้นหาเส้นทางแบบ A*

วิธีการค้นหาแบบ A* อาศัยฟังก์ชั่นประเมินค่า (Evaluation function)

$$f(n) = g(n) + h(n) \quad (6)$$

ในการชี้นำการค้นหาให้ไปสู่โหนดจุดหมายได้อย่างรวดเร็ว โดย $g(n)$ หมายถึงค่าใช้จ่าย (Cost) จริงที่ใช้ในการเดินทางจากโหนดเริ่มต้น s จนไปถึงโหนดปัจจุบัน n (ซึ่งค่าใช้จ่ายในที่นี้อาจเป็นระยะทาง เวลา ฯลฯ) และค่า $h(n)$ หมายถึงค่าใช้จ่ายประมาณการที่ใช้ในการเดินทางจากโหนดปัจจุบัน n ไปถึงโหนดจุดหมาย g ดังรูปที่ 2



รูปที่ 2. ส่วนประกอบของ Heuristic function f(n)

อัลกอริธึมวิธีการค้นหาแบบ A* จะใช้ $f(n)$ ในการตัดสินใจเลือกโหนดของกราฟ โดยเริ่มจากโหนดเริ่มต้น s เพื่อค้นหาโหนดลูกในแต่ละขั้น ซึ่งแต่ละขั้นจะทำให้เกิดเส้นทางที่ยังค้นหาไม่จบ เส้นทางเหล่านี้มีโหนดปลาย ซึ่งแต่ละอันถือว่าเป็นโหนด n ของเส้นทางนั้นๆ ถ้าเส้นทางใดในบรรดาเส้นทางเหล่านี้ให้ค่า $f(n)$ ของโหนด n ของตนเป็นค่าที่น้อยที่สุดในกลุ่มโหนด n อื่นๆ อัลกอริธึมจะเลือกขยายขอบเขตการค้นหาต่อในที่เส้นทางนั้น โดยขยายการค้นหาที่โหนด n ของเส้นที่เลือกลงไปยังโหนดลูกของมัน ถ้ามีโหนดลูกตัวใดปรากฏในเส้นทางที่กำลังค้นหาค้างอยู่ก็จะถูกตัดทิ้งไป การค้นหาโดย A* จะทำทีละขั้นตอนเช่นนี้ซ้ำแล้วซ้ำอีกไปเรื่อยๆจนกระทั่งพบโหนดจุดหมาย g แล้วเส้นทางที่เกิดขึ้นที่เชื่อมจาก s ไปยัง g จะถือว่าเป็นเส้นทางที่ดีที่สุดที่เลือกโดยวิธี A*

วิธี A* จะสามารถเลือกเส้นทางที่ดีที่สุด (Optimum solution) ได้ก็ต่อเมื่อฟังก์ชั่น $g(n)$ ซึ่งเป็นการคิดค่าใช้จ่ายจากโหนดเริ่มต้น s ถึงโหนดปัจจุบัน n จะต้องเป็นฟังก์ชั่นที่คิดค่าใช้จ่ายจริงของเส้นทางที่เชื่อมจาก s ถึง n ส่วน $h(n)$ ซึ่งเป็นการประมาณการค่าใช้จ่ายจากโหนด n ถึงโหนดจุดหมาย g จะต้องเป็นการประมาณการที่ไม่เกินค่าใช้จ่ายจริงของเส้นทางที่สั้นที่สุดที่เชื่อมจาก n ถึง g (คุณสมบัติ Admissible [7])

ปัญหาของการค้นหาแบบ A* คือในกรณีที่สเปซเส้นทางการค้นหาที่เป็นไปได้ทั้งหมดเป็นอนันต์ จะทำให้ A* ไม่

NCSEC 2006

สามารถใช้งานได้และการที่สเปซการวางแผนเส้นทางบินของ UAV มีลักษณะอนันต์ดังนั้นการจะนำเอา A* มาใช้จำเป็นต้องปรับปรุงขนาดสเปซของการค้นหาให้มีขนาดจำกัด (Finite) เสียก่อน

3.2. การวางแผนเส้นทางบินในกรณีที่มีจุดหมายเดียว

การวางแผนเส้นทางบินที่มีจุดหมายเดียวเพื่อเลี่ยงสิ่งกีดขวางของ UAV อาศัยการพิจารณาหาเส้นตรงที่สัมผัส (Tangent line) กับจุดยอดต่างๆของรูปหลายเหลี่ยมและไม่ตัดผ่านสิ่งกีดขวาง และเราจะใช้กราฟนี้สำหรับการวางแผนเส้นทางบิน

เราจะแสดงขั้นตอนการทำงานอย่างง่าย โดยใช้รูปที่ 2(a), (b) และ(c) ประกอบการอธิบายโดยมีข้อกำหนดดังนี้

- กำหนดให้ค่ามุมเลี้ยวสูงสุดเป็น θ–60 องศา
- ค่าระยะทางไกลสูงสุดเป็น L–500 เมตร
- Cross line เป็นเส้นตรงที่ตัดผ่านสิ่งกีดขวาง
- Reference line เป็นเส้นตรงที่บอกทิศทางในขณะ UAV เคลื่อนที่ผ่านจุดหนึ่งไปยังอีกจุดหนึ่ง
- Self hit line เป็นเส้นตรงที่ตัดผ่านสิ่งกีดขวางซึ่งจุดหนึ่งของเส้นตรงเป็นจุดยอดของสิ่งกีดขวางนั้นเอง กรณีนี้เราจะสร้างเส้นตรงที่เชื่อมจุดข้างเคียงทั้งสองจุดเพื่อแทนเส้นตรงนี้ เป็นการสร้างเส้นทางอ้อมสิ่งกีดขวาง

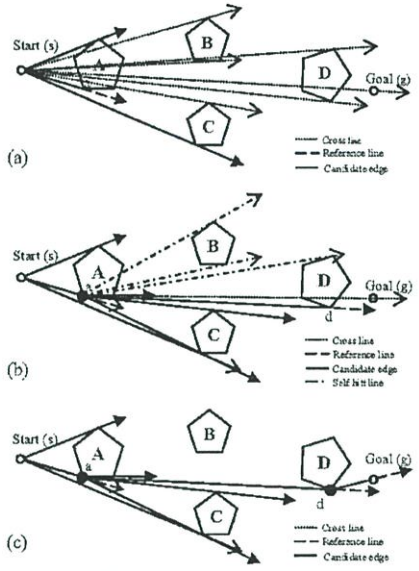
เริ่มจากนี้รูป 2(a) เราจะพิจารณาจากโหนดเริ่มต้น s ไปยังสิ่งกีดขวางต่างๆรวมทั้งโหนดจุดหมาย g ด้วย เราจะได้เส้นสัมผัสของสิ่งกีดขวางทั้งหมดโดยมีบางเส้นถูกคัดออกโดยอาศัยเงื่อนไขว่า 1) เป็นเส้นตรงที่ตัดผ่านสิ่งกีดขวาง (Cross line, Self hit line) 2) เป็นโหนดที่เกิดจากเส้นสัมผัสมีลักษณะที่ขัดกับลักษณะการเคลื่อนที่ของ UAV คือมีค่ามุมเลี้ยวมากกว่าค่ามุมเลี้ยวสูงสุด (θ) และมีระยะทางไกลกว่าค่าระยะทางสูงสุด (L) จากรูปที่ 2(a) เราตัดเส้นสัมผัสออกเหลือเพียง 3 เส้นซึ่งเส้นสัมผัสนี้ได้ให้โหนดมา 3 โหนด จากนั้นเราจะเลือกโหนด 3 โหนดด้วยวิธีการ A* ซึ่งจะเลือกโหนดที่มีค่า $f(n)$ น้อยที่สุด โดยอาศัยฟังก์ชัน Heuristic $h(n)$ ที่เป็นการคำนวณค่าระยะห่างระหว่างจุด n และ g จากสมการหาระยะทางระหว่าง 2 จุด

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \qquad (7)$$

สมมุติว่าโหนดที่ถูกเลือกคือโหนด a ของสิ่งกีดขวาง A เราจะใช้โหนด a พิจารณาในขั้นตอนต่อไป

ขั้นตอนต่อมาเราจะหาเส้นสัมผัสสิ่งกีดขวางอีกครั้ง โดยเราจะไม่นำจุดยอดของสิ่งกีดขวาง A พิจารณาด้วยในตอนนี้ แต่จะนำมาพิจารณาอีกครั้งเมื่อเกิด Self-hit line จากโหนด a ไปยังสิ่งกีดขวางอื่นๆที่จะเกิดขึ้นตามมา ดังรูปที่ 2(b) เมื่อหาเส้นสัมผัสจากโหนด a ไปยังสิ่งกีดขวางทั้งหมดได้แล้ว เกิด Self-hit line เราจะสร้างเส้นตรงจุดโหนด a ไปยังจุดยอดข้างเคียงแทนซึ่งในขั้นตอนนี้เราสมมุติให้โหนด d ของสิ่งกีดขวาง D ถูกเลือก

ขั้นตอนต่อไป เราไม่นำสิ่งกีดขวาง A, B และ C มาพิจารณาเนื่องจากมีค่ามุมข้างอิงมากกว่า θ/2 และเราจะหาเส้นสัมผัสสิ่งกีดขวางและเลือกโหนดเช่นนี้ไปเรื่อยๆจนกระทั่งพบโหนด g ซึ่งจากตัวอย่างนี้เราสามารถไปถึงจุดหมายได้เลยโดยลากเส้นตรงจากโหนด d ไปยังโหนดจุดหมาย g ดังในรูปที่ 2(c) แล้วเราจึงหยุดการค้นหาและได้เส้นทางที่สั้นที่สุดผ่านโหนดต่างๆ ดังนี้ $s \rightarrow a \rightarrow d \rightarrow g$



รูปที่ 2. แสดงการวางแผนเส้นทางบิน

NCSEC 2006

4. ปัญหาการวางแผนเส้นทางบินที่มีหลายจุดหมาย

เมื่อนำวิธี A* มาใช้วางแผนเส้นทางในกรณีที่มีจุดหมายจำนวนมาก สมมุติว่าให้แต่ละจุดหมายเป็น g_i เราก็จะมีฟังก์ชั่น $h(n)$ สำหรับแต่ละ g_i ดังนั้นถ้ามีจุดหมายจำนวน m ตัวก็จะมี ฟังก์ชั่น $h(n)$ ทั้งหมด m ฟังก์ชั่น ใน [7] ได้อธิบายถึงวิธีการนำ $h(n)$ หลายๆฟังก์ชั่นมาพิจารณาร่วมกันเพื่อใช้แทนค่า $h(n)$ รวมโดยเลือกเอาค่า $h_i(n)$ ที่มีค่าน้อยที่สุดมาใช้ดังนี้

$$h(n) = \min(h_1(n), h_2(n), \ldots, h_m(n)) \qquad (8)$$

โดยที่ m คือจำนวนจุดหมายทั้งหมดและ $h_i(n)$ เป็นฟังก์ชั่น Heuristic สำหรับจุดหมาย g_i ใดๆ

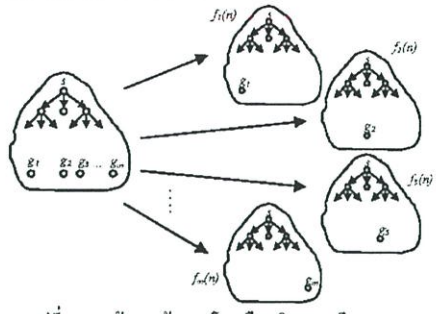
วิธีการนี้ยังคงรับประกันได้ว่า A* ให้เส้นทางที่ดีที่สุดเสมอถ้าแต่ละ $h_i(n)$ ยังคงมีคุณสมบัติที่เป็น Admissible [7]

เราพบว่าวิธีการนี้สามารถนำมาใช้ในการวางแผนเส้นทางบินของ UAV ในกรณีที่มีหลายจุดหมายได้โดยการนำเอาวิธีการค้นหาแบบหนึ่งจุดหมายซึ่งได้อธิบายไปก่อนหน้านี้มาประยุกต์ใช้ โดยในการเลือกโหนด n แต่ละครั้งจะพิจารณาจากค่า $f(n)$ ซึ่งคำนวณจาก $h(n)$ ตาม (8) แต่วิธีนี้มีข้อเสียคือ เมื่อนำไปใช้กับกรณีที่มีจุดหมายจำนวนมากจะทำให้เวลาที่ใช้ในประมวลผลมีมากขึ้นมากตามไปด้วย และยังไม่สามารถใช้ได้กับการวางแผนที่จุดหมายมีจำนวนอนันต์ได้เนื่องจาก m ใน (8) จะต้องเป็นจำนวนจำกัดเท่านั้น

นอกจากนี้โดยอาศัยข้อมูลการเคลื่อนที่ของ UAV เพื่อคัดโหนดบางตัวออกไป อาจทำให้เกิดปัญหาคือ เกิดมีบางเส้นทางที่ UAV ไม่สามารถเดินทางไปได้เนื่องจากขัดต่อลักษณะการเคลื่อนที่ของมัน เราจึงไม่นำมาพิจารณา ทำให้จุดหมายบางอันที่ถูกปิดกั้นโดยสิ่งกีดขวางสามารถตัดออกจากการพิจารณาในการค้นหาได้ ถ้าวิธีการค้นหาใดไม่สามารถตัดจุดหมายดังกล่าวออกจากการพิจารณาก็จะเป็นอุปสรรคต่อการค้นหา ทำให้เสียเวลากับจุดหมายที่มีอุปสรรคกีดขวางเส้นทางนี้ ซึ่งวิธีการที่จะเสนอต่อไปสามารถเอาชนะปัญหานี้ได้

5. การวางแผนเส้นทางบินที่มีจุดหมายจำนวนอนันต์

เพื่อให้การวางแผนเส้นทางบินที่มีจุดหมายจำนวนอนันต์ทำงานได้อย่างมีประสิทธิภาพ ก่อนที่จะอธิบายถึงวิธีการที่นำเสนอเราขอนิยามจุดหมายที่มีจำนวนอนันต์ดังต่อไปนี้



รูปที่ 3. การค้นหาเส้นทางโดยเลือกพิจารณาทีละจุดหมาย

5.1 จุดหมายที่มีจำนวนอนันต์

ปัญหาของการค้นหาเส้นทางที่มีจุดหมายไม่จำกัดหรือเป็นอนันต์จะเกิดขึ้นในกรณีที่สเปซการค้นหาสามารถขยายขอบเขตออกไปได้เรื่อยๆ ไม่มีที่สิ้นสุด ดังเช่น กรณีการหาจุดจอดของ UAV ซึ่งจอดบนพื้นที่ได้ไม่จำกัดคราบใดที่มีพื้นดินขยายต่อออกไปเรื่อยๆ วิธีการค้นหาเส้นทางที่นำเสนอในที่นี้สามารถค้นหาจุดหมายที่ดีที่สุดจากสเปซการค้นหาที่สามารถขยายขอบเขตออกไปได้ไม่มีสิ้นสุด โดยอัลกอริธึมของเราจะเลือกพิจารณาจุดหมายที่ใกล้เมื่อได้คำตอบจุดหมายที่ใกล้ที่สุดแล้ว จุดหมายอื่นๆที่เหลือจะถูกตัดออกจากการพิจารณาแต่ล้ากลุ่มจุดหมายที่อยู่ใกล้ถูกเงื่อนไขบางอย่างตัดทิ้งไปหมด ขอบเขตของสเปซการค้นหาจะถูกขยายออกไป เพื่อเอากลุ่มจุดหมายที่ไกลออกไปมาพิจารณา

5.2 การวางแผนเส้นทางบินที่จำนวนจุดหมายจำนวนอนันต์

เราจะพิจารณาว่าในสเปซการค้นหาเส้นทางที่มีหลายๆ จุดหมาย ถ้าเราแยกพิจารณาการค้นหาออกเป็นการค้นหาที่อิสระจากกันแต่ทำหลายๆครั้งซึ่งแต่ละการค้นหาเรามุ่งพิจารณาไปที่จุดหมายหนึ่งๆ เพียงจุดหมายเดียว โดยอาศัย $f(n)$ ของแต่ละจุดหมายนั้นๆ โดยไม่ต้องสนใจจุดหมายอื่นๆที่เหลือดังในรูปที่ 3 เมื่อการค้นหาในแต่ละครั้งสำหรับแต่ละจุดหมายได้

เส้นทางคำตอบออกมาทั้งหมดแล้วเราจะนำมาเปรียบเทียบกัน คำตอบที่ดีที่สุดที่ได้ก็จะบอกถึงจุดหมายที่ดีที่สุด ซึ่งวิธีการเช่นนี้อาจจะทำได้โดยการคำนวณแบบขนานหรือการทยอยทำ แต่อย่างไรก็ตามการคำนวณแบบนี้มีความสิ้นเปลืองมาก เนื่องจากการคำนวณและการค้นหากราฟในลักษณะเช่นนี้เกิดความซ้ำซ้อนกันมาก พบว่าเราสามารถลดความซ้ำซ้อนนี้ได้โดยใช้วิธีการค้นหาด้วยการคำนวณแบบเรียงลำดับ (Sequential) ซึ่งใช้กราฟหรือสเปซการค้นหาร่วมกันและมีวิธีการที่จะเลือกพิจารณาจุดหมายโดยใช้ $f_i(n)$ ช่วยจากนั้นทยอยเปลี่ยนจุดหมายที่จะพิจารณาตามความเหมาะสมซึ่งทำให้เกิดเป็นวิธีการที่จะนำเสนอดังต่อไปนี้

กำหนดให้ฟังก์ชั่นประเมินค่าที่ใช้พิจารณาเลือกโหนดสำหรับการค้นหาเมื่อพิจารณาที่แต่ละจุดหมาย g_i เป็น

$$f_i(n) = g(n) + h_i(n) \qquad (9)$$

โดยที่ $1 \le i \le m$ ซึ่ง m คือจำนวนจุดหมาย (ซึ่งอาจมีค่าเป็นอนันต์) และ $h_i(n)$ เป็นฟังก์ชั่น Heuristic สำหรับแต่ละจุดหมาย g_i

อัลกอริธึมการค้นหาเส้นทางมีขั้นตอนการทำงานดังนี้

กำหนดให้ $F = \{f_1(n), f_2(n), ..., f_m(n)\}$ โดยที่ m อาจจะเป็นค่าอนันต์ และให้ N เป็นเซตของโหนดที่ยังค้นหาไม่เสร็จและจะถูกเลือกสำหรับการค้นหาต่อ โดยเริ่มแรกให้ $N = \{start\}$ ก่อน ลำดับการค้นหามีดังต่อไปนี้

1. เลือกพิจารณาจุดหมายที่คาดว่าดีที่สุด โดยอาศัยการเรียงลำดับน้อยไปหามากตามค่า $f_i(n)$ ของแต่ละจุดหมายโดยการนำไปคำนวณร่วมกับ N แล้วเลือก $f_i(n)$ ที่ให้ค่าน้อยที่สุด (สังเกตว่าสำหรับ $f_i(n)$ ที่มีค่ามากๆ จะถูกเลือกมาพิจารณาหลังๆ ยิ่งถ้ามีค่าขึ้งมาก ก็อาจจะไม่ถูกพิจารณาเลย ทำให้จุดหมายที่อยู่ใกลๆ ออกไปหรือที่มีเป็นจำนวนอนันต์ถูกคัดทิ้งไปในที่สุด) สมมุติว่า ฟังก์ชั่นที่เลือกมาเป็น $f_s(n)$ ดังนั้น $f_s(n)$ คือฟังก์ชั่นของจุดหมาย s ที่เลือกมาพิจารณา
2. ขยายการค้นหาจากโหนด $start$ ลงไปยังโหนดลูก $\{n_1, n_2, ..., n_k\}$ ของมัน ตัดโหนด $start$ ออกจาก N แล้วนำเซตนี้ไปยูเนียน N
3. เลือกโหนดใน N ที่ให้ค่าของ $f_s(n_i)$ น้อยที่สุดแล้วกำหนดเป็นโหนด $selected\_N$
4. นำค่า $f_s(selected\_N)$ มาเปรียบเทียบกับค่า $f_t(selected\_N_t)$ ของจุดหมายอื่นๆที่ไม่ใช่จุดหมาย s (โดยที่ $selected\_N_t$ นั้น เป็นโหนดล่าสุดที่ถูกเลือกมาค้นหาเมื่อพิจารณาที่แต่ละจุดหมาย g_t ในการค้นหารอบก่อนๆ โดยกำหนดให้มีค่าเริ่มแรกเป็นโหนด $start$ สำหรับทุกๆ จุดหมาย)
5. จากขั้นตอน 4 เลือกค่าที่น้อยที่สุดมา ตรวจดูว่า $selected\_N$ เป็นจุดหมายหรือไม่ ถ้าเป็นแสดงว่าได้เส้นทางไปยังจุดหมายที่ดีที่สุดแล้ว จบการทำงาน มิฉะนั้นตรวจสอบว่า ถ้าค่าที่น้อยที่สุดยังคงเป็น $f_s(selected\_N)$ แล้ว ก็จะเป็นการพิจารณาที่จุดหมายเดิมแล้วค้นหาลึกลงไปอีกระดับ โดยกำหนดให้ $selected\_N$ เป็นเสมือนโหนด $start$ (ให้ $start = selected\_N$ ก่อน) แล้วกลับไปทำขั้นตอนที่ 2 ซ้ำอีก มิฉะนั้นแล้ว จะเป็นการเปลี่ยนจุดหมายในการพิจารณาค้นหา ให้จดจำโหนด $selected\_N$ ของจุดหมายเดิมไว้ (โดย $selected\_N$ ล่าสุดของแต่ละจุดหมายนี้จะถูกนำมาใช้พิจารณาอีกในขั้นตอน 4 รอบต่อไป) คง N ไว้ จากนั้นกำหนดให้ $f_t(selected\_N_t)$ นั้นเป็นเสมือน $f_s(n)$ แล้วกลับไปทำขั้นตอนที่ 3 ซ้ำอีก

จากคำอธิบายข้างต้นสามารถเขียนเป็นอัลกอริธึมได้รูปที่ 4

ต่อจากนี้เราใช้รูปที่ 5 อธิบายขั้นตอนการทำงาน โดยเริ่มจากรูปที่ 5(a) เราจะเลือกจุดหมายที่ใกล้ที่สุดมาพิจารณาก่อน สมมุติให้ g_1 เป็นจุดหมายที่ใกล้ที่สุด ดังนั้นเราจะเลือก $f_1(n)$ มาแล้วใช้สำหรับการประเมินค่าเพื่อเลือกโหนดด้วยวิธี A* โดยโดยนำจุดเริ่มต้นมาขยายการค้นหาไปยังโหนดลูกและเลือกโหนดลูกที่มีค่า $f_1(n_i)$ น้อยที่สุด สมมุติให้โหนดลูก n_1 เป็น

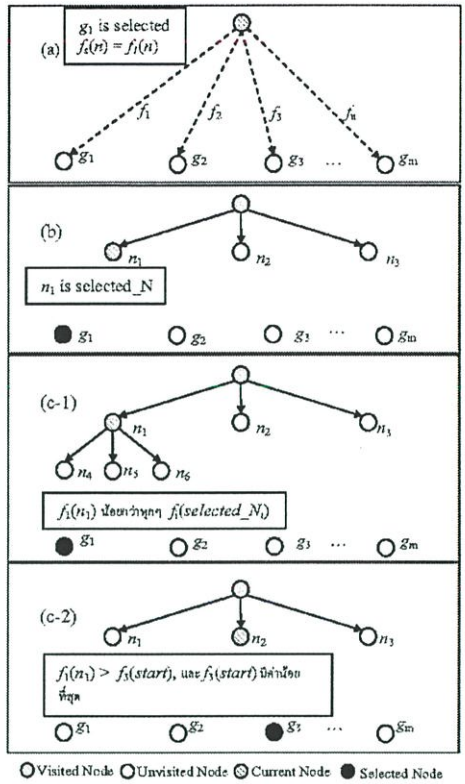
โหนดที่มีค่า $f_1(n_1)$ น้อยที่สุดดังรูปที่ 5(b) จากนั้นนำ $f_1(n_1)$ มาเปรียบเทียบกับค่า $f_1(selected\_N)$ สำหรับจุดหมายอื่นๆ เพื่อหาค่าน้อยที่สุด ถ้ายังเป็น $f_1(n)$ อยู่ จะพิจารณาจุดหมายเดิมลึกลงไปอีกขั้นดังรูปที่ 4(c-1) แต่ถ้า $f_1(n)$ ไม่ใช่ค่าน้อยที่สุด โหนด n_1 จะเป็น $selected\_N$ ล่าสุดของจุดหมาย g_1 ซึ่งจะถูกเก็บไว้ในอาร์เรย์สำหรับการเปรียบเทียบครั้งต่อไป สมมุติให้จุดหมาย g_3 ลึกว่า g_1 ซึ่งหมายถึง $f_3(n) < f_1(n)$ เมื่อกลับไปเลือกโหนดที่มีค่าน้อยที่สุดด้วย $f_3(n)$ แล้ว จะได้โหนด n_2 มาพิจารณาดังรูปที่ 5(c-2) แล้วทำการขยายการค้นหาไปยังโหนดลูกต่อไปซึ่งจะทำขั้นตอนดังรูป 5(a) ซ้ำอีกเสมือนว่า n_2 เป็นจุดเริ่มต้นอีกครั้ง จนกระทั่งโหนดที่ถูกเลือกมาจะเป็นจุดหมายซึ่งแสดงว่าได้เส้นทางที่สั้นที่สุดแล้ว ซึ่งเราก็จะบอกได้ว่าจุดหมายที่พบนั้นเป็นจุดหมายที่ดีที่สุด

```
def Deepening-A*-Search(start, goals):
  F = {f1, f2, f3, ....}
  N = {start}
  #L is array of every previous selected node for each goal
  L = {start, start, start, ...}
  ft = select-min(F applied to list of selected_Ni)
  while N not empty
    Let selected_N be a node from N whose ft(n) is min
    if selected_N is one of goals then
      return selected_N as the best goal
    else
      Let ft' be the function giving min value among
                              ft(selected_N) & fi(selected_Ni)
      if ft' = ft then
        start = selected_N
        N = N – {selected_N}
        N = N ∪ expand(selected_N)
      else
        store selected_N for goalt in L
        ft = ft'
  return None

def expand(N):
  return set of child nodes of N
```

รูปที่ 4. อัลกอริธึมวิธีเอสตาร์แบบลึกลงไป

จะสังเกตว่าอัลกอริธึมนี้มีการทำงานในสองลักษณะคือ ที่รูป 5(c-1) อัลกอริธึมนี้จะค้นหาในแนวลึกที่โหนด $selected\_N$ อีก 1 ขั้น ส่วนในรูปที่ 5(c-2) จากการสลับเปลี่ยนไป พิจารณาที่จุดหมายอื่นแทนทำให้อัลกอริธึมมีการค้นหาโหนดในแนวกว้างแทนแนวลึก จากลักษณะการค้นหาที่ทำแนวลึกสลับกับแนวกว้างแต่ยังใช้วิธี A* จึงขอเรียกอัลกอริธึมนี้ว่า "Deepening A*" หรือ "การค้นหา A* แบบค้นหาลึกลงไป"



รูปที่ 5.แสดงการทำงานของอัลกอริธึมเอสตาร์แบบค้นหาลึกลงไป

เราได้นำเอาอัลกอริธึมนี้มาประยุกต์ใช้ในการวางแผนเส้นทางบินสำหรับ UAV ในกรณีมีจุดหมายเป็นจำนวนอนันต์ ซึ่งวิธีการวางแผนเส้นทางบินเขียนได้ดัง Pseudo code ในรูปที่ 6 ส่วนที่เพิ่มเติมจากอัลกอริธึมก่อนหน้านี้ (ตัวอักษรเอียง) คือ การใส่เงื่อนไขการไม่พิจารณาโหนดที่มีระยะทางเกินกว่าระยะทางบินไกลสุดของ UAV (max_length) และเงื่อนไขการ

ขยายโหนดลูกด้วยการเชื่อมระหว่างจุดยอดของสิ่งกีดขวางต่างๆ ซึ่งอาจจะมีบางโหนดไม่ถูกนำมาพิจารณาเนื่องจากมุมเลี้ยวที่จะไปถึงโหนดนั้นๆมีค่ามากกว่ามุมเลี้ยวสูงสุดของ UAV (max_turn) ซึ่งได้กล่าวมาแล้วในหัวข้อที่ 3.2

```
global max_length, max_turn
def UAV_Path_Search( start, goals):
  F = {f1, f2, f3, ....}
  N = {start}
  #L is array of every previous selected node for each goal
  L = {start, start, start, ... }
  fi = select-min(F applied to list of selected_Ni)
  while N not empty
    Let selected_N be a node from N whose fi(n) is min
    if g( selected_N ) > max_length then
      prune selected_N
    else if selected_N is one of goals then
      return selected_N as the best goal
    else
      Let fi' be the function giving min value among
                              fi(selected_N) & fi(selected_Ni)
      if fi' = fi then
        start = selected_N
        N = N – {selected_N}
        N = N ∪ expand(selected_N)
      else
        store selected_N for goali in L
        fi = fi'
  return None

def expand( node ):
  points = {}
  for obstacle in onstacles
    pts = get_tangentvertex( obstacle, node)
    for pt in pts
      if degree(pt) < max_turn then
        points = points ∪ { pt }
  return points
```

รูปที่ 6. อัลกอริธึมการวางแผนสำหรับ UAV

Goal Number	Time(second)	Found
5	4.127218	True
10	4.155721	True
30	6.467129	True
50	8.153260	True
100	9.217061	True
300	55.491070	True
500	69.106475	True
1000	6634.694455	False
3000	2999.302302	True
5000	7212.235892	True

ตาราง 1. แสดงผลการค้นหาเส้นทางที่มีจำนวนจุดหมายต่างๆกัน

6. ผลการทดลอง

จาก Pseudo code ข้างต้น เราเขียนเป็นโปรแกรมโดยใช้ภาษา Python แล้วนำมาทดลองวางแผนเส้นทางโดยมีสิ่งกีดขวางจำนวน 28 ชิ้น กำหนดมุมเลี้ยวสูงสุดไว้ที่ 60 องศาและระยะทางบินไกลที่สุดไว้ที่ 60 กิโลเมตร

ตาราง 1 จะแสดงเวลาที่ใช้ในการค้นหาเส้นทางที่จำนวนจุดหมายต่างๆกัน การวางแผนเส้นทางในบางครั้งอาจไม่ให้ผลลัพธ์ (กรณี 1000 จุดหมาย) อันเป็นผลมาจากเส้นทางที่เป็นไปได้เกิดการขัดแย้งกับลักษณะการบินของ UAV จึงถูกตัดทิ้งไปทั้งหมด

7. การวิเคราะห์อัลกอริธึมวิธีเอสตาร์แบบค้นหาลึกลงไป

ถ้าพิจารณาอัลกอริธึม A* แบบค้นหาลึกลงไปอย่างคร่าวๆ จะเห็นว่าเป็นการผสมผสานระหว่าง Breadth-first Search และ A* Search คือเป็นการค้นหาในแนวกว้างแต่จะถูกชี้นำในการค้นหาทางแนวลึกโดย A* เนื่องจากอัลกอริธึมนี้จะเลือกค้นหาโดยเลือกพิจารณาที่จุดหมายห่างจากจุดเริ่มต้นจากน้อยไปหามาก ในลักษณะ Breadth-first Search จึงรับประกันได้ว่าจะพบจุดหมายที่ใกล้ที่สุดก่อน

เราได้วิเคราะห์เพื่อเปรียบประสิทธิภาพการทำงานระหว่างอัลกอริธึมที่กล่าวถึงในหัวข้อ 4 กับอัลกอริธึมของเรา พบว่าลิสต์ N จากรูปที่ 6ใช้สำหรับเก็บโหนดเพื่อรอการนำไปหาโหนดลูกในแต่ละรอบการค้นหาเส้นทางโหนด การคำนวณหาค่า $h(n)$ ของแต่ละโหนดจากสมการ (8) ของอัลกอริธึมแรก เมื่อมีการทำงานไปแล้วจำนวน m รอบ มีจำนวนครั้งเป็น

$$g_{(1)} + bg_{(2)} + bg_{(3)} + \ldots + bg_{(m)} = g(mb - b + 1) \qquad (10)$$

โดยที่ b คือ ค่า Branching factor และ g คือจำนวนจุดหมายทั้งหมด

ส่วนอัลกอริธึมของเราที่ทำงานที่จำนวนรอบเท่ากันจะมีจำนวนครั้งในการคำนวณค่า $h(n)$ เป็น

$$1_{(1)} + b_{(2)} + b_{(3)} + \ldots + b_{(m)} = mb - b + 1 \qquad (11)$$

ซึ่งจำนวนครั้งที่ได้นี้เป็นกรณีที่ไม่มีการเปลี่ยนแปลงจุดหมายที่ใช้พิจารณาเลย ดังนั้นเมื่อมีการเปลี่ยนจุดหมายซึ่งการเปลี่ยนจุดหมายทุกครั้งต้องเรียงลำดับโหนดที่เหลืออยู่ใหม่

ทั้งหมด และต้องคำนวณค่า $h(n)$ ของโหนดใหม่ด้วย ดังนั้นถ้าเราสมมุติให้มีการเปลี่ยนจุดหมาย ณ รอบการทำงานที่ j จะได้จำนวนโหนดที่แลกออกทั้งหมดเป็น $b(j-1)+1$ โหนดและจำนวนโหนดที่พิจารณาแล้วเป็น j โหนด จึงมีโหนดที่เหลืออยู่เท่ากับ $bj-b-j+1$ ซึ่งจะต้องคำนวณค่า $h(n)$ ใหม่ ดังนั้นจะมีจำนวนครั้งในการคำนวณ $h(n)$ เป็น

$$1_{(1)}+b_{(2)}+...+b_{(j)}+...+b_{(m)}=b(m+j-2)-j+2 \qquad (12)$$

ซึ่งจะเห็นว่าวิธีการที่นำเสนอมีจำนวนครั้งของการคำนวณค่า $h(n)$ ที่ไม่ขึ้นอยู่กับจำนวนจุดหมายเลย จึงสามารถทำงานได้กับกรณีจำนวนจุดหมายเป็นอนันต์

8. บทสรุป

เราได้พัฒนาอัลกอริธึม A* แบบค้นหาลึกลงไป เพื่อวางแผนเส้นทางบินสำหรับ UAV ในกรณีที่มีจำนวนจุดหมายมากและอาจจะมีเป็นจำนวนอนันต์ซึ่งเป็นอัลกอริธึมที่มีประสิทธิภาพมากกว่าอัลกอริธึม A* แบบเดิม

9. เอกสารอ้างอิง

[1] เกียรติศักดิ์ ศิวขุนทด, วิทิษฎ์ ทิรัญกิตติ, "การวางแผนเส้นทางบินที่สามารถเลี่ยงสิ่งกีดขวางสำหรับอากาศยานไร้นักบิน". *การประชุมวิชาการและวิศวกรรมคอมพิวเตอร์แห่งชาติครั้งที่ 9 (NCSEC2005), กรุงเทพ, 2548*

[2] Dobbs, S., Davis, W., and Carl Lizza, "An application of heuristic search techniques to the problem of flight path generator in a military hostile environment ", *International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, pp 273 280, 1988.

[3] Hwang, Y. J., Kim, S. J., Lim, S. S., and Park, H. K., "A fast path planning by path graph optimization", *IEEE Transaction on System, Man, and Cybernetics-part a: System and Humans*, Vol. 33, No.1 January 2003.

[4] Szczerba, J., Galkowski, P., Glickstein, S., and Ternullo, N., "Robust algorithm for real-time route planning", *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 36, No.3 July 2000.

[5] H. Dominik, W. Christian, and W. Heinz, "Multi-directional search with goal switching for robot path planning", *The 11th International Conference on Industrial & Engineering Application of Artificial Intelligence & Expert Systems*, Castellan, Spain. 1-4. July, 1998.

[6] W. Christian, H. Dominik, and W. Heinz, "Multi-goal path planning for industrial robots", *International Conference on Robotics and Application (RA'99)*, Santa Barbara, USA, Oct. 28-90, 1999.

[7] Russel, S., Norvig, P., *Artificial Intelligence: A Modern Approach*, Prentice Hall, 2003.

ประวัติผู้เขียน

ชื่อ-นามสกุล	นายเกียรติศักดิ์ คิวขุนทด
วันเดือนปีเกิด	วันที่ 6 เดือนมิถุนายน พุทธศักราช 2522
ประวัติการศึกษา	สำเร็จการศึกษาระดับปริญญาตรี หลักสูตรวิศวกรรมศาสตรบัณฑิต สาขาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ปีการศึกษา 2545
ผลงาน/งานวิจัย	ปี 2545 ผู้ช่วยนักวิจัย "โครงการวิจัยและพัฒนาอากาศยานไร้นักบิน" สนับสนุนทุนวิจัยโดย สำนักงานกองทุนสนับสนุนการวิจัย (สกว.) และสำนักงานวิจัยและพัฒนาการทหารกลาโหม (สวพ.กห.)
	ปี 2547 ได้รับรางวัลรองชนะเลิศอันดับ 1 จากการประกวด "Hucth Navi Developper Contest 2004" จัดโดย บริษัท ฮัทชิสัน ซีเอที ไวร์เลสมัลติมีเดีย จำกัด
	ปี 2548 ผู้ช่วยนักวิจัย "โครงการระบบจัดส่งรถให้บริการขนส่งและบริหารการขนส่งอย่างชาญฉลาดผ่านเครือข่ายการสื่อสารซีดีเอ็มเอ" สนับสนุนทุนวิจัยโดย บริษัท กสท. โทรคมนาคม จำกัด (มหาชน)
	ปี 2550 ผู้ช่วยนักวิจัย "โครงการระบบให้บริการยายพาหนะอัจฉริยะ" สนับสนุนทุนวิจัยโดย บริษัท กสท. โทรคมนาคม จำกัด(มหาชน)