บทที่ 4

ผลการดำเนินงานวิจัย

บทนี้เป็นการนำเสนอหัวข้อเกี่ยวกับผลการดำเนินงานวิจัย ซึ่งประกอบด้วย โปรแกรมที่ใช้ในงานวิจัย การเตรียมข้อมูลและรูปแบบปัญหาสำหรับปัญหาคลังสินค้าที่ใช้ทดสอบ ตัวแบบจำลองทางคณิตศาสตร์ การใช้โปรแกรม LINGO และ Microsoft Visual Basic C# เพื่อการหาค่าคำตอบ และการสรุปผลการดำเนินงานวิจัยจะทำการแสดงรายละเอียดดังต่อไปนี้

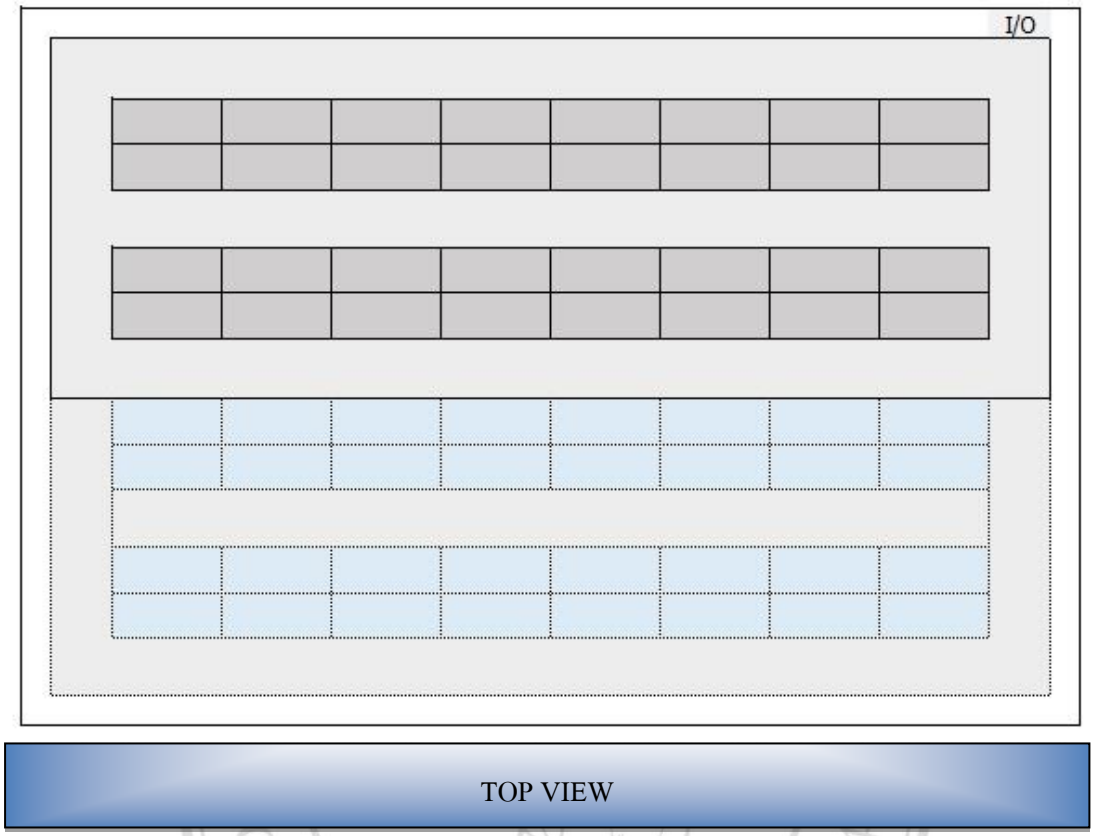
4.1 การเตรียมข้อมูลสำหรับปัญหาการจัดเก็บสินค้า

4.1.1 ขนาดของคลังสินค้า

ลักษณะแผนผังคลังสินค้า

ลักษณะคลังสินค้าที่ใช้สำหรับงานวิจัยนี้แสดงได้ดังรูปที่ 4.1 โดยคลังสินค้าสามารถอธิบายได้ดังนี้

- รูปแบบแผนผังคลังสินค้าจะมีลักษณะของช่องจัดเก็บสินค้า และช่องทางเดินเหมือนกันหมดทั้งคลัง
- มีประตูเข้า-ออกสำหรับเคลื่อนย้ายสินค้า 1 ประตู โดยประตูจะอยู่ที่บริเวณมุมใดมุมหนึ่งของคลังสินค้า
- มีช่องทางเดินล้อมรอบช่องจัดเก็บสินค้าเพื่อใช้เคลื่อนย้ายสินค้าในกิจกรรมจัดเก็บ ช่องทางเดินต้องมีความกว้างเพียงพอสำหรับพนักงานหรือรถโฟล์คลิฟท์เคลื่อนที่ได้
- จำนวนช่องจัดเก็บสำหรับใส่สินค้าสามารถลดและเพิ่มจำนวนได้ทุกมิติ ทั้งในแนวราบและแนวดิ่ง ทำให้สามารถลดหรือเพิ่มจำนวนช่องจัดเก็บสินค้าได้อย่างอิสระ



รูปที่ 4.1 แสดงลักษณะแผนผังคลังสินค้าที่ใช้ในการทดสอบปัญหา

ในส่วนถัดไปคือการแสดง ข้อมูลสินค้าที่ต้องจัดเก็บ(หัวข้อ 4.1.2) และแผนการจัดเก็บสินค้า(หัวข้อ 4.1.3) ซึ่งชุดข้อมูลดังกล่าว เป็นข้อมูลสินค้าสำหรับใช้ทดสอบตัวอย่างปัญหาเบื้องต้น ส่วนชุดข้อมูลเกี่ยวกับสินค้าทั้งหมดที่ทำการสร้างขึ้นเพื่อใช้ในงานวิจัยนี้จะถูกนำเสนอในหัวข้อที่ 4.3.2 การสร้างปัญหาตัวอย่างของการจัดเก็บสินค้า และรายละเอียดจะถูกแสดงในภาคผนวก

4.1.2 ข้อมูลสินค้าที่ต้องจัดเก็บ

ข้อมูลสำคัญเกี่ยวกับสินค้าที่ต้องนำมาใช้ในการทำวิจัยนี้ ได้แก่

1. ปริมาณการจัดเก็บสินค้า/นำสินค้าออก ต่อช่วงเวลาหนึ่ง ซึ่งชุดข้อมูลนี้กำหนดไว้ที่ 3 เดือน
2. ปริมาณการจัดเก็บสินค้า/นำสินค้าออก ต่อเดือน (การเฉลี่ยค่าจาก 3 เดือน)
3. จำนวนช่องเก็บที่สินค้าแต่ละชนิดต้องการใช้

ตารางที่ 4.1 แสดงข้อมูลสินค้าสำหรับตัวอย่างปัญหาเบื้องต้น

ชนิดสินค้า	ปริมาณการจัดเก็บสินค้า/นำสินค้าออก (3 เดือน) (1)	ปริมาณการจัดเก็บสินค้า/นำสินค้าออกต่อเดือน (2) = (1)/3	จำนวนช่องเก็บที่สินค้าแต่ละชนิดต้องการใช้ (3)
1	45	15	17
2	60	20	14
3	15	5	7
รวม	120	40	38

ข้อมูลสินค้าสำหรับตัวอย่างปัญหาเบื้องต้นเพื่อทดสอบแบบจำลองทางคณิตศาสตร์มีรายละเอียดดังต่อไปนี้

- สินค้ามีจำนวน 3 ชนิด
- ปริมาณการจัดเก็บสินค้า/นำสินค้าออก ต่อ 3 เดือน ของสินค้าแต่ละชนิดมีค่าเท่ากับ 45, 60, 1
- ปริมาณการจัดเก็บสินค้า/นำสินค้าออก ต่อเดือน ของสินค้าแต่ละชนิดมีค่าเท่ากับ 15, 20, 5
- จำนวนช่องจัดเก็บที่สินค้าแต่ละชนิดต้องการมีค่าเท่ากับ 17, 14, 7

4.1.3 แผนการจัดเก็บสินค้า

งานวิจัยนี้ได้เลือกใช้นโยบายจัดเก็บสินค้าเป็นแบบ Full Turnover Storage ซึ่งพิจารณาการจัดเก็บจากปริมาณการเคลื่อนย้ายสินค้าเข้าเพื่อจัดเก็บและการนำสินค้าออก สินค้าที่มีอัตราการเคลื่อนย้ายสูงจะถูกจัดวางอยู่ในช่องเก็บที่มีระยะทางใกล้กับทางเข้า-ออกของคลังสินค้า สินค้าที่มีอัตราในการเคลื่อนย้ายในลำดับรองลงมาให้อยู่ในช่องจัดเก็บระยะถัดไป ส่วนสินค้าที่มีอัตราการเคลื่อนย้ายต่ำที่สุดจะถูกจัดวางในช่องจัดเก็บที่มีระยะทางไกลจากทางเข้า-ออกคลังสินค้ามากที่สุด โดยแผนการจัดเก็บนี้สินค้าประเภทเดียวกันไม่จำเป็นต้องถูกจัดเก็บที่บริเวณเดียวกันเสมอไป เพราะคำนึงถึงตำแหน่งการจัดเก็บที่ทำให้ระยะทางของการเคลื่อนย้ายรวมของสินค้าทั้งหมดมีค่าที่น้อยที่สุด ซึ่งแผนการจัดเก็บสินค้าดังกล่าวสอดคล้องกับปัญหาใน

งานวิจัยนี้ที่มีชุดข้อมูลเกี่ยวกับสินค้าที่มีอัตราคงที่ ไม่มีการเปลี่ยนแปลงบ่อย และมีพื้นที่จัดเก็บมากเพียงพอ สามารถรองรับปัญหาการจัดเก็บในคลังสินค้าขนาดใหญ่ได้

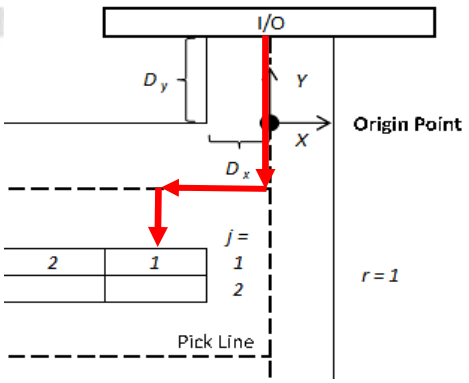
ตารางที่ 4.2 แสดงข้อมูลลำดับการจัดเก็บสินค้าสำหรับตัวอย่างปัญหาเบื้องต้น

ชนิดสินค้า	T_p	S_p	T_p / S_p	ลำดับความสำคัญ
2	20	14	1.429	1
1	15	17	0.882	2
3	5	7	0.714	3

ค่า T_p / S_p ที่แสดงในตารางที่ 4.2 เป็นค่าจากผลหารของปริมาณการจัดเก็บสินค้า/นำสินค้าออก (T_p) กับจำนวนช่องเก็บที่สินค้าแต่ละชนิดต้องการใช้ (S_p) ผลที่ได้คือค่าที่แสดงอัตราการจัดเก็บ/นำออกของสินค้า ซึ่งเป็นค่าบ่งบอกความสำคัญของสินค้าแต่ละชนิด สินค้าที่มีอัตราการจัดเก็บ/นำออกสินค้าสูงที่สุดควรจะถูกจัดเก็บไว้ที่ตำแหน่งใกล้ประตูคลังสินค้า ซึ่งเป็นตำแหน่งที่มีระยะทางการเคลื่อนที่น้อยที่สุดในกิจกรรมการจัดเก็บสินค้า

4.1.4 วิธีการวัดระยะด้วยวิธีการแบบ Rectilinear

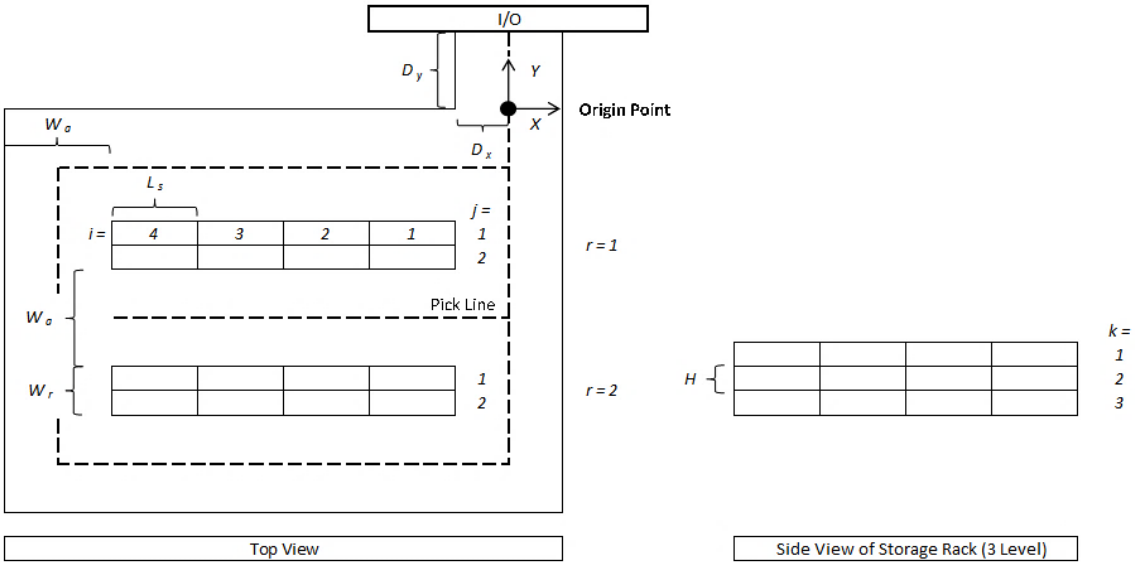
ในงานวิจัยนี้การแสดงระยะทางระหว่างจุดเข้าออกของสินค้า (I/O) และช่องจัดเก็บสินค้าในตำแหน่งต่างๆ โดยการวัดระยะใช้วิธีการคิดแบบ Rectilinear ซึ่งเป็นการวัดจากทางเดินที่เป็นเส้นตรงจากจุดเริ่มต้น ไปยังจุดศูนย์กลางของช่องวางสินค้านั้นๆ เพื่อให้ทราบค่าระยะทางจากจุดเข้าออกสินค้าไปยังช่องจัดเก็บสินค้าแต่ละช่อง ค่าระยะทางนี้จะถูกนำไปพิจารณาตำแหน่งการจัดเก็บสินค้าแต่ละชนิดต่อไป วิธีการดังกล่าวสามารถแสดงตัวอย่างระยะทางได้ดังรูปที่ 4.2



รูปที่ 4.2 แสดงตัวอย่างการวัดระยะด้วยวิธี Rectilinear

จากรูปเป็นตัวอย่างการแสดงวิธีการวัดระยะด้วยวิธีการแบบ Rectilinear โดยการวัดระยะทางจากจุดที่มีการเข้า-ออกของสินค้าไปยังจุดศูนย์กลางของช่องเก็บสินค้าแต่ละช่อง โดยค่าระยะทางได้จากผลรวมของการวัดระยะการเคลื่อนที่ทั้งแนวราบและแนวดิ่ง ระยะการเคลื่อนที่นี้จะถูกนำมาใช้พิจารณาตำแหน่งการจัดเก็บสินค้าแต่ละชนิด

4.2 ตัวแบบจำลองทางคณิตศาสตร์สำหรับปัญหางานวิจัย



รูปที่ 4.3 รูปแบบของแผนผังคลังสินค้าที่ใช้ในการสร้างแบบจำลองทางคณิตศาสตร์

<u>สมมติฐาน</u>

สมมติฐานเกี่ยวกับแผนผังคลังสินค้าที่ใช้สร้างแบบจำลองทางคณิตศาสตร์มีการกำหนดดังนี้

1. ประตูเข้าออกสินค้า มี 1 จุด อยู่ที่ตำแหน่งมุมใดมุมหนึ่งของคลังสินค้า
2. ช่องจัดเก็บสินค้า และช่องทางเดินจะมีลักษณะเหมือนกันทั้งหมด
3. ช่องจัดเก็บสินค้า 1 ช่อง สามารถเก็บสินค้าได้ 1 ชนิด
4. พิจารณาระยะทางการเคลื่อนที่ในการจัดเก็บ โดยอ้างอิงจากจุดกึ่งกลางของช่องทางเดิน และช่องจัดเก็บสินค้า
5. ระยะทางการเคลื่อนที่ที่นำมาคำนวณพิจารณาระยะทางขาเดียว เนื่องจากเส้นทางในการไป-กลับช่องจัดเก็บสินค้าต้องเป็นเส้นทางเดียวกัน

ดัชนีและตัวแปรที่ใช้ในแบบจำลอง คือ

p	=	ชนิดของสินค้า	(1, 2, 3, …, n)
i	=	ตำแหน่งหลักของช่องจัดเก็บสินค้า	(1, 2, 3, …, m)
j	=	ตำแหน่งแถวของช่องจัดเก็บสินค้า	(1, 2, 3, …, q)
k	=	ตำแหน่งชั้นของช่องจัดเก็บสินค้า	(1, 2, 3, …, k)
r	=	ตำแหน่งบล๊อกของช่องจัดเก็บสินค้า	(1, 2, 3, …, r)
D_x	=	ระยะทางจากประตูคลังสินค้ามายังจุดอ้างอิง วัดตามแนวแกน X	
D_y	=	ระยะทางจากประตูคลังสินค้ามายังจุดอ้างอิง วัดตามแนวแกน Y	
TD	=	ระยะทางการเคลื่อนที่รวมในการจัดเก็บสินค้า/นำสินค้าออก	
W_a	=	ความกว้างของช่องทางเดิน	
W_r	=	ความกว้างของช่องจัดเก็บสินค้า	
Ls	=	ความยาวของช่องจัดเก็บสินค้า	
H	=	ความสูงของช่องจัดเก็บสินค้า	
T_p	=	ปริมาณการจัดเก็บสินค้า/นำสินค้าออก ของสินค้าแต่ละชนิด	
S_p	=	จำนวนช่องจัดเก็บที่สินค้าแต่ละชนิดต้องการใช้	
I	=	ตัวเลขแสดงตำแหน่งหลักของช่องจัดเก็บสินค้า	
J	=	ตัวเลขแสดงตำแหน่งแถวของช่องจัดเก็บสินค้า	
K	=	ตัวเลขแสดงตำแหน่งชั้นของช่องจัดเก็บสินค้า	
R	=	ตัวเลขแสดงตำแหน่งบล๊อกของช่องจัดเก็บสินค้า	

ตัวแปรการตัดสินใจ คือ

$$X_{pijkr} = \begin{cases} 1 & \text{, ถ้าสินค้า } p \text{ ถูกจัดเก็บที่ ตำแหน่ง } i \text{ แถวที่ } j \text{ ชั้นที่ } k \text{ บล๊อกที่ } r \\ 0 & \text{, otherwise} \end{cases}$$

สมการวัตถุประสงค์

Minimize Z (ระยะทางการเคลื่อนที่รวม) =

$$\sum_{p=1}^{n}\sum_{i=1}^{m}\sum_{j=1}^{q}\sum_{k=1}^{k}\sum_{r=1}^{r}\left(\frac{Tp}{Sp}\right)X_{pijkr}D_{ijkr} \qquad (4.1)$$

สมการข้อกำหนด

$$D_{ijkr} = TD_x + TD_y + TD_z \quad (4.2)$$

$$TD_x = D_x + (I - 0.5)L_s \quad (4.3)$$

$$TD_y = D_y + (0.5J\,W_a) + [(J\text{-}1)(W_r + 0.5W_a)] + [(W_r + W_a)(R\text{-}1)] \quad (4.4)$$

$$TD_z = H(K\text{-}1) \quad (4.5)$$

$$\sum_{p=1}^{n} X_{pijkr} \leq 1 \qquad \forall\, i,j,k,r \quad (4.6)$$

$$\sum_{i=1}^{m}\sum_{j=1}^{q}\sum_{k=1}^{k}\sum_{r=1}^{r} X_{pijkr} = S_p \qquad \forall\, p \quad (4.7)$$

สมการวัตถุประสงค์ (สมการที่ (4.1)) คือการลดระยะทางการเคลื่อนที่ในกิจกรรมการจัดเก็บสินค้าให้มีระยะทางรวมน้อยที่สุด โดยระยะทางการเคลื่อนที่รวมนี้ต้องคำนึงถึงระยะทางการเคลื่อนที่ทั้งในแนวราบและแนวดิ่ง

สมการข้อกำหนด สมการที่ (4.2) เป็นการอธิบายวิธีการคำนวณระยะทางรวม ซึ่งประกอบไปด้วยการพิจารณาระยะทางการเคลื่อนที่ในแนวราบ 2 แกน ซึ่งคือค่า TD_x (สมการที่ (4.3)) และ TD_y (สมการที่ (4.4)) และระยะทางการเคลื่อนที่ในแนวดิ่งอีก 1 แกน ซึ่งคือค่า TD_z (สมการที่ (4.5))

สมการข้อกำหนด (สมการที่ (4.6)) คือข้อจำกัดสำหรับการจัดเก็บสินค้า โดยกำหนดให้ช่องจัดเก็บสินค้า 1 ช่อง จัดเก็บสินค้าได้เพียง 1 ชนิด ซึ่งจะไม่มีการคละชนิดสินค้าในแต่ละช่องจัดเก็บ

สมการข้อกำหนด (สมการที่ (4.7)) เป็นสมการเพื่อใช้กำหนดว่า ผลการจัดเก็บสินค้าแต่ละชนิดต้องมีจำนวนการจัดเก็บครบถ้วน เท่ากับจำนวนช่องจัดเก็บที่สินค้าแต่ละชนิดต้องการใช้

4.2.1 การทดสอบปัญหาเชิงตัวเลข

แบบจำลองทางคณิตศาสตร์ข้างต้นจะถูกมานำใช้ทดสอบปัญหาการจัดเก็บสินค้า โดยใช้รูปแบบและลักษณะคลังสินค้าตามที่กำหนดในรูปที่ 4.3 ซึ่งเบื้องต้นได้ทำการทดลองกับตัวอย่างปัญหาเบื้องต้น ซึ่งรายละเอียดของปัญหาแสดงดังตารางที่ 4.3

ตารางที่ 4.3 การกำหนดค่าตัวแปรที่ใช้ทดสอบตัวอย่างปัญหาเบื้องต้น

	ตัวแปร	ค่า	ตัวแปร	ค่า
เกี่ยวกับแผนผัง	D_x	2	W_a	4
	D_y	2	W_r	3
	L_s	2	H	1.7

ตารางที่ 4.3 การกำหนดค่าตัวแปรที่ใช้ทดสอบตัวอย่างปัญหาเบื้องต้น (ต่อ)

	ตัวแปร	ค่า	ตัวแปร	ค่า
เกี่ยวกับสินค้า	S_1	17	T_1	15
	S_2	14	T_2	20
	S_3	7	T_3	5

ในปัญหาตัวอย่างปัญหาเบื้องต้นที่ทำการทดสอบได้กำหนดให้คลังสินค้ามีขนาด 3 ชั้น มีจำนวนช่องเก็บสินค้า 48 ช่อง สินค้าที่ถูกจัดเก็บมี 3 ชนิด จำนวนสินค้าแต่ละชนิดที่ต้องการจัดเก็บ ซึ่งมีค่าเท่ากับจำนวนช่องจัดเก็บที่สินค้าแต่ละชนิดต้องการใช้ (S_p) มีค่า 17, 14 และ 7 ชิ้น ตามลำดับ ปริมาณการจัดเก็บสินค้า/นำสินค้าออก ของสินค้าแต่ละชนิด (T_p) มีค่า 15, 20 และ 5 ตามลำดับ หลังจากได้กำหนดค่าตัวแปรต่างๆ สำหรับทดสอบปัญหาแล้ว จึงทำการทดสอบตัวอย่างปัญหาการจัดเก็บสินค้าด้วยโปรแกรม LINGO เพื่อหาคำตอบและตรวจสอบความถูกต้องของสมการทางคณิตศาสตร์

4.2.2 การใช้โปรแกรม LINGO ในการหาค่าคำตอบที่ดีที่สุด

งานวิจัยนี้ได้ใช้โปรแกรม LINGO version 5 ในการหาคำตอบที่ดีที่สุดจากแบบจำลองทางคณิตศาสตร์ ซึ่งรายละเอียดการเขียนรหัสโปรแกรมสามารถแสดงได้ดังรูปที่ 4.4

```
Model:
Sets:
!ideices, parameters,decision variables;
Product/1..3/:S,T;
Position/1..4/:Q ;
Rack/1..2/:M ;
Level/1..3/:C ;
Row/1..2/:N ;
Links1(Product,Position,Rack,Level,Row): X;
Links2(Position,Rack,Level,Row): D;
Endsets

!Math Model;
!Objective Minimize Total Traveling Distance;
Min = @sum(Links1(p,i,j,k,r): (T(p)/S(p)*X(p,i,j,k,r)*D(i,j,k,r)));

!This is how the total distance is calculated;
@For(Position(i):@For(Rack(j):@For(Level(k):@For(Row(r):D(i,j,k,r)
= (Distx + ((Q(i)-0.5)*L))+(Disty +(0.5*M(j)*WA)+((M(j)-1)*(WR+(0.5*WA)))+((WR+WA)*(N(r)-
1)))+(H*(C(k)-1))))));

!Constraints;
@For(Position(i):@For(Rack(j):@For(Level(k):@For(Row(r):@sum(product(p):X(p,i,j,k,r))<=
1))));
@For(product(p):@sum(Links1(p,i,j,k,r):X(p,i,j,k,r))=S(p));
@FOR(Links1:@BIN(X));
```

```
!Here is data;
Data:
S= 17 14 7;
T= 15 20 5;
Distx = 2;
Disty = 2;
H = 1.7;
L = 2;
WA = 4;
WR = 3;
Q = 1 2 3 4;
M = 1 2;
N = 1 2;
C = 1 2 3;
Enddata

End
```

รูปที่ 4.4 แสดงการแก้ปัญหาแบบจำลองทางคณิตศาสตร์ด้วยโปรแกรม LINGO

แบบจำลองทางคณิตศาสตร์ข้างต้นจะถูกนำมาแปลงให้อยู่ในลักษณะรหัสโปรแกรม LINGO เพื่อใช้ประมวลผลหาคำตอบและตรวจสอบความถูกต้องของแบบจำลอง รหัสโปรแกรมดังกล่าวสามารถแสดงรายละเอียดได้ดังต่อไปนี้

```
Model:
1] Sets:
2] !ideices, Parameters,Decision Variables;
3] Product/1..3/:S,T;
4] Position/1..4/:Q ;
5] Rack/1..2/:M ;
6] Level/1..3/:C ;
7] Row/1..2/:N ;
8] Links1(Product,Position,Rack,Level,Row): X;
9] Links2(Position,Rack,Level,Row): D;
10] Endsets

11] !Math Model;
12] !Objective Minimize Total Traveling Distance;
13] Min = @sum(Links1(p,i,j,k,r):
(T(p)/S(p)*X(p,i,j,k,r)*D(i,j,k,r)));

20] !Total distance is calculated;
21]
@For(Position(i):@For(Rack(j):@For(Level(k):@For(Row(r):D(i,j,k,r)
22] = (Distx + ((Q(i)-0.5)*L))+(Disty +(0.5*M(j)*WA)+((M(j)- 1)
23] *(WR+(0.5*WA)))+((WR+WA)*(N(r)-1)))+(H*(C(k)-1))))));

24] !Constraints;
25] @For(Position(i):@For(Rack(j):@For(Level(k):
26] @For(Row(r):@sum(product(p): X(p,i,j,k,r))<= 1))));
27] @For(product(p):@sum(Links1(p,i,j,k,r):X(p,i,j,k,r))=S(p));
28] @For(Links1:@BIN(X));

29] !Data;
30] Data:
31] S= 17 14 7;
32] T= 15 20 5;
33] Distx = 2;
34] Disty = 2;
```

```
35] H = 1.7;
36] L = 2;
37] WA = 4;
38] WR = 3;
39] Q = 1 2 3 4;
40] M = 1 2;
41] N = 1 2;
42] C = 1 2 3;
43] Enddata

End
```

ข้อมูลข้างต้นคือรายละเอียดแบบจำลองทางคณิตศาสตร์ที่ถูกแปลงให้อยู่ในรูปแบบของโปรแกรม LINGO ซึ่งจะถูกใช้ทดสอบตัวอย่างปัญหาและตรวจสอบความถูกต้องของแบบจำลองทางคณิตศาสตร์ที่ใช้ในงานวิจัยนี้ รายละเอียดชุดคำสั่งในตัวโปรแกรมสามารถอธิบายได้ดังนี้

บรรทัดที่ 1-10 เป็นการประกาศกลุ่มตัวแปร ซึ่งประกอบไปด้วยตัวแปรที่เกี่ยวกับสินค้า ตัวแปรที่เกี่ยวกับตำแหน่งช่องจัดเก็บสินค้า ตัวแปรตัดสินใจ การกำหนดค่าเชื่อมโยงกับตำแหน่งและระยะทางในการจัดเก็บสินค้า

บรรทัดที่ 11-13 คือสมการวัตถุประสงค์ที่ต้องการลดระยะทางการเคลื่อนที่ในกิจกรรมการจัดเก็บ-นำออกสินค้าให้มีระยะทางรวมน้อยที่สุด

บรรทัดที่ 20-23 สมการข้อกำหนดเพื่อการคำนวณหาระยะทางรวมการเคลื่อนที่ ซึ่งคิดจากผลรวมระยะทางการเคลื่อนที่จากจุดเริ่มต้นไปยังช่องเก็บสินค้าแต่ละช่อง โดยคำนึงถึงการเคลื่อนที่ทั้งในแนวราบ 2 แกน และในแนวดิ่งอีก 1 แกน

บรรทัดที่ 24-28 คือสมการข้อจำกัด ซึ่งประกอบไปด้วยข้อจำกัดสำหรับการจัดเก็บสินค้าโดยช่องจัดเก็บ 1 ช่อง จะสามารถจัดเก็บสินค้าได้เพียง 1 ชนิด กล่าวคือจะไม่มีการคละชนิดสินค้าในแต่ละช่องจัดเก็บ และมีการตรวจสอบผลการจัดเรียงสินค้าแต่ละชนิด ต้องมีจำนวนเท่ากับจำนวนช่องจัดเก็บสินค้าที่สินค้าแต่ละชนิดต้องการใช้

บรรทัดที่ 29-30 เป็นการประกาศค่าของข้อมูลที่ต้องใช้ในการคำนวณ

บรรทัดที่ 31 จำนวนช่องเก็บสินค้าที่สินค้าแต่ละชนิดต้องการ

บรรทัดที่ 32 ปริมาณการจัดเก็บสินค้า/นำสินค้าออก ของสินค้าแต่ละชนิด

บรรทัดที่ 33 ค่าระยะทางจากประตูสินค้ามายังจุดอ้างอิง วัดตามแนวราบแกนที่ 1

บรรทัดที่ 34 ค่าระยะทางจากประตูสินค้ามายังจุดอ้างอิง วัดตามแนวราบแกนที่ 2

บรรทัดที่ 35 ค่าความสูงของช่องจัดเก็บสินค้า

บรรทัดที่ 36 ค่าความยาวของช่องจัดเก็บสินค้า

บรรทัดที่ 37 ค่าความกว้างของช่องทางเดิน

บรรทัดที่ 38 ค่าความกว้างของช่องจัดเก็บสินค้า

บรรทัดที่ 39 จำนวนหลักของช่องจัดเก็บสินค้า

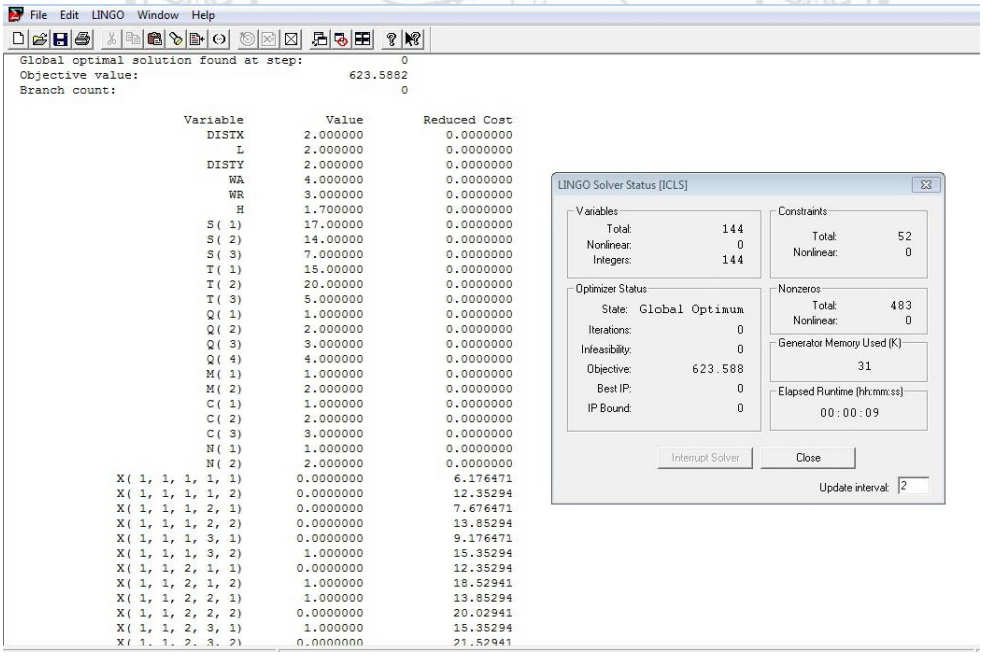
บรรทัดที่ 40 จำนวนแถวของช่องจัดเก็บสินค้า

บรรทัดที่ 41 จำนวนบล๊อกของช่องจัดเก็บสินค้า

บรรทัดที่ 42 จำนวนชั้นของช่องจัดเก็บสินค้า

4.2.3 ผลการทดสอบจากโปรแกรม LINGO

จากการคำนวณผ่านโปรแกรม LINGO version 5 ผลลัพธ์ที่ได้จากตัวอย่างปัญหาคือ ระยะทางรวมทั้งหมดที่น้อยที่สุดในกิจกรรมการจัดเก็บสินค้า 3 ชนิด มีค่าเท่ากับ 623.588 เมตร ผลที่ได้จากโปรแกรม LINGO แสดงดังรูปที่ 4.5



รูปที่ 4.5 ผลลัพธ์จากการคำนวณตัวอย่างปัญหาด้วยโปรแกรม LINGO

เมื่อนำตัวอย่างปัญหาเบื้องต้นมาทำการประมวลผลด้วยโปรแกรม LINGO จะได้ผลการจัดเก็บสินค้า ผลที่ได้นี้จะถูกนำไปตรวจความถูกต้องของผลลัพธ์ในขั้นตอนต่อไป ชุดข้อมูลคำตอบจากโปรแกรมจะถูกนำเสนอในภาคผนวก

จากผลลัพธ์จากการคำนวณด้วยโปรแกรม LINGO เมื่อนำผลการจัดเก็บสินค้ามาทำการตรวจสอบแล้ว สามารถสรุปผลการทดสอบปัญหาการจัดเก็บสินค้าด้วยตัวอย่างปัญหาเบื้องต้นได้ดังตารางที่ 4.4 และตารางที่ 4.5

ตารางที่ 4.4 ระยะทางการจัดเก็บ/นำออก สินค้าแต่ละชนิดสำหรับตัวอย่างปัญหาเบื้องต้น

ชนิดสินค้า	ระยะทางในการเคลื่อนย้ายสินค้า
1 (ชิ้นที่ 1)	15.7
1 (ชิ้นที่ 2)	16.0
1 (ชิ้นที่ 3)	16.0
1 (ชิ้นที่ 4)	16.4
1 (ชิ้นที่ 5)	17.4
1 (ชิ้นที่ 6)	17.4
1 (ชิ้นที่ 7)	17.7
1 (ชิ้นที่ 8)	17.7
1 (ชิ้นที่ 9)	18.0
1 (ชิ้นที่ 10)	18.0
1 (ชิ้นที่ 11)	19.4
1 (ชิ้นที่ 12)	19.4
1 (ชิ้นที่ 13)	19.7
1 (ชิ้นที่ 14)	19.7
1 (ชิ้นที่ 15)	20.0
1 (ชิ้นที่ 16)	20.0
1 (ชิ้นที่ 17)	21.0
2 (ชิ้นที่ 1)	7.0
2 (ชิ้นที่ 2)	8.7
2 (ชิ้นที่ 3)	9.0
2 (ชิ้นที่ 4)	10.4
2 (ชิ้นที่ 5)	10.7
2 (ชิ้นที่ 6)	11.0

ตารางที่ 4.4 ระยะทางการจัดเก็บ/นำออก สินค้าแต่ละชนิดสำหรับตัวอย่างปัญหาเบื้องต้น (ต่อ)

ชนิดสินค้า	ระยะทางในการเคลื่อนย้ายสินค้า
2 (ชิ้นที่ 7)	12.4
2 (ชิ้นที่ 8)	12.7
2 (ชิ้นที่ 9)	13.0
2 (ชิ้นที่ 10)	14.0
2 (ชิ้นที่ 11)	14.0
2 (ชิ้นที่ 12)	14.4
2 (ชิ้นที่ 13)	14.7
2 (ชิ้นที่ 14)	15.7
3 (ชิ้นที่ 1)	21.4
3 (ชิ้นที่ 2)	21.4
3 (ชิ้นที่ 3)	21.7
3 (ชิ้นที่ 4)	21.7
3 (ชิ้นที่ 5)	22.7
3 (ชิ้นที่ 6)	23.0
3 (ชิ้นที่ 7)	23.4

ตารางที่ 4.5 สรุปผลการทดสอบระยะทางการจัดเก็บ/นำออกสินค้า สำหรับตัวอย่างปัญหาเบื้องต้น

ชนิดสินค้า	T_p	S_p	T_p / S_p	ลำดับความสำคัญ	ระยะทางในการเคลื่อนย้ายสินค้า
1	15	17	0.882	2	15.7 – 21.0
2	20	14	1.429	1	7.0 - 15.7
3	5	7	0.714	3	21.4 – 23.4

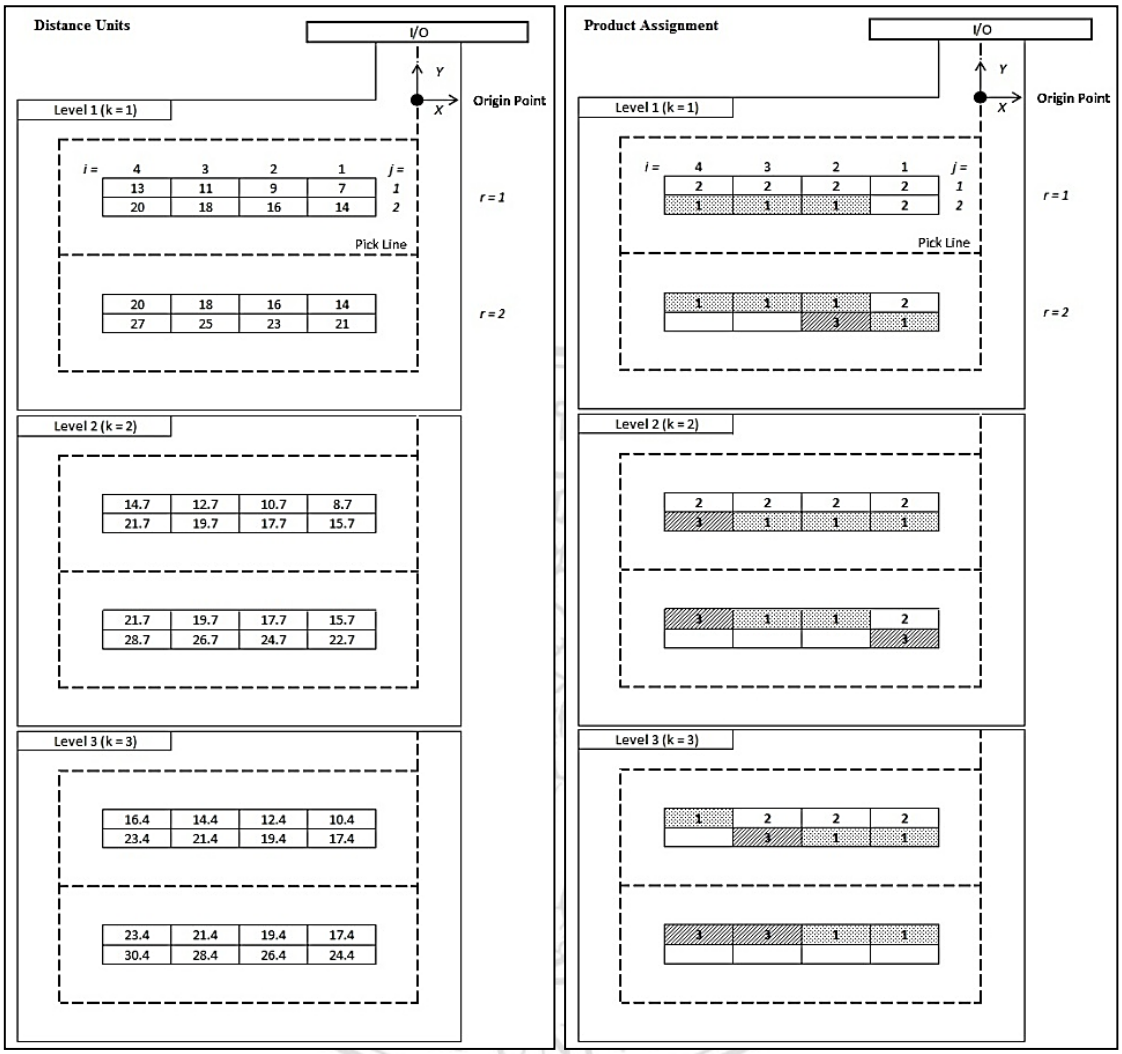
4.3 ผลการตรวจสอบความถูกต้อง

4.3.1 การตรวจสอบความถูกต้องตัวอย่างปัญหาเบื้องต้น

ค่าปริมาณการจัดเก็บสินค้า/นำสินค้าออก ของสินค้าแต่ละชนิด (T_p) และ จำนวนช่องจัดเก็บที่สินค้าแต่ละชนิดต้องการใช้ (S_p) ถือเป็นส่วนสำคัญในการพิจารณาปัญหาดังกล่าว จากตารางสรุปผลจะเห็นได้ว่าสินค้าชนิดที่มีค่า T_p / S_p สูงสุดจะถูกนำไปจัดเก็บที่ช่องใส่สินค้าเป็นลำดับแรก และสินค้าที่มีค่า T_p / S_p น้อยกว่าจะถูกจัดเก็บในลำดับถัดไปจนครบทุกชนิดสินค้า

ผลจากการกำหนดค่าและทำการคำนวณด้วยโปรแกรม LINGO ได้ผลลัพธ์ว่า สินค้าชนิดที่ 2 มีค่า T_p / S_p สูงที่สุด ซึ่งหมายถึงเป็นสินค้าชนิดที่มีความสำคัญที่สุด จะมีกิจกรรมการจัดเก็บหรือนำสินค้าออกจากคลังสินค้าบ่อยเพราะมีอัตราการจัดเก็บ/นำออกสินค้าสูงควรจัดเก็บไว้ใกล้ประตูคลังสินค้า กล่าวคือสินค้าชนิดที่ 2 ควรถูกจัดเก็บไว้ในตำแหน่งที่มีระยะทางการเคลื่อนที่น้อยที่สุด ดังนั้นสินค้าชนิดนี้จะถูกจัดเก็บเข้าที่ช่องจัดเก็บเป็นลำดับแรก เมื่อจัดเก็บสินค้าชนิดที่ 2 ครบทุกชิ้น จึงทำการจัดเก็บสินค้าชนิดที่ 1 และ สินค้าชนิดที่ 3 ตามลำดับ ซึ่งสามารถแสดงผลการจัดเก็บสินค้าได้ดังรูปที่ 4.6 และเมื่อสินค้าทุกชนิดถูกจัดเก็บตามตำแหน่งที่เหมาะสมแล้ว จะทำให้ทราบระยะทางการเคลื่อนที่รวมที่น้อยที่สุดได้คือ 623.588

ผลลัพธ์จากการคำนวณด้วยโปรแกรม LINGO ให้ผลที่สอดคล้องกับแบบจำลองทางคณิตศาสตร์ที่สร้างขึ้นในงานวิจัยนี้ สามารถใช้แก้ปัญหาการจัดเก็บสินค้าได้ตามที่ต้องการ คำตอบมีความถูกต้องทั้งผลการจัดเก็บ และผลรวมระยะทางการเคลื่อนที่ จึงสรุปได้ว่าแบบจำลองทางคณิตศาสตร์ที่สร้างขึ้นมีความถูกต้อง สามารถนำไปใช้ทดสอบตัวอย่างปัญหาในงานวิจัยนี้ได้



รูปที่ 4.6 ระยะทางการเคลื่อนที่เพื่อนำสินค้าไปยังช่องจัดเก็บ และผลการจัดเรียงการจัดเก็บสินค้า

4.3.2 การสร้างตัวอย่างปัญหาการจัดเก็บสินค้า

หลังจากที่ได้ทำการทดสอบตัวอย่างปัญหาเบื้องต้น ซึ่งมีจำนวนช่องเก็บสินค้าทั้งหมด 48 ช่อง จำนวนสินค้า 3 ชนิด และตรวจสอบความถูกต้องการเขียนโปรแกรม LINGO สำหรับแบบจำลองทางคณิตศาสตร์ที่นำมาใช้แก้ปัญหาการจัดเก็บสินค้าแล้ว ผู้วิจัยได้สร้างตัวอย่างปัญหาของการจัดเก็บสินค้าในคลังสินค้าให้มีความซับซ้อนมากขึ้นโดยมีขนาดของคลังสินค้าที่ใหญ่ขึ้น จำนวนชนิดของสินค้าเพิ่มมากขึ้น และปริมาณสินค้าแต่ละชนิดที่ต้องจัดเก็บเพิ่มมากขึ้น ตัวอย่างปัญหาทั้งหมดที่ได้สร้างขึ้นเพื่อทำการทดสอบในงานวิจัยนี้ แสดงดังตารางที่ 4.6 โดยขนาดของตัวอย่างปัญหาจะถูกกำหนดด้วยค่าของ จำนวนชนิดสินค้า จำนวนช่องจัดเก็บที่ต้องการใช้ และจำนวนช่องเก็บสินค้าทั้งหมดในคลังสินค้า ตามลำดับ

ตารางที่ 4.6 ตัวอย่างปัญหาของการจัดเก็บสินค้าที่สร้างขึ้น

ตัวอย่างปัญหา	จำนวนชนิดสินค้า	จำนวนช่องจัดเก็บที่ต้องการใช้	จำนวนช่องจัดเก็บในคลังสินค้า
WH01	3	38	48
WH02	60	259	300
WH03	135	840	1000
WH04	270	1675	2000
WH05	500	2825	3360
WH06	600	2534	3360
WH07	500	2825	4992
WH08	800	3993	4992
WH09	500	2825	6300
WH10	1000	5040	6300

ตัวอย่างปัญหาทั้ง 10 กรณีที่สร้างขึ้นเป็นการจัดเก็บสินค้าในคลังสินค้าขนาดเล็กที่มีจำนวนช่องจัดเก็บปริมาณน้อย ไปจนถึงคลังสินค้าขนาดใหญ่ที่มีจำนวนช่องจัดเก็บปริมาณมาก ข้อมูลเกี่ยวกับจำนวนช่องเก็บสินค้าซึ่งมีผลต่อแผนผังคลังสินค้าของตัวอย่างปัญหาทั้ง 10 กรณี แสดงดังตารางที่ 4.7

ตารางที่ 4.7 ข้อมูลเกี่ยวกับช่องเก็บสินค้าสำหรับตัวอย่างปัญหาที่สร้างขึ้น

ตัวอย่างปัญหา	จำนวนหลัก (i) ช่องเก็บสินค้า	จำนวนแถว (j) ช่องเก็บสินค้า	จำนวนชั้น (k) ช่องเก็บสินค้า	จำนวนบล๊อก (r) ช่องเก็บสินค้า
WH01	4	2	3	2
WH02	10	2	3	5
WH03	10	2	5	10
WH04	20	2	5	10
WH05	24	2	5	14
WH06	24	2	5	14

ตารางที่ 4.7 ข้อมูลเกี่ยวกับช่องเก็บสินค้าสำหรับตัวอย่างปัญหาที่สร้างขึ้น (ต่อ)

ตัวอย่างปัญหา	จำนวนหลัก (i) ช่องเก็บสินค้า	จำนวนแถว (j) ช่องเก็บสินค้า	จำนวนชั้น (k) ช่องเก็บสินค้า	จำนวนบล๊อก (r) ช่องเก็บสินค้า
WH07	26	2	6	16
WH08	26	2	6	16
WH09	30	2	7	15
WH10	30	2	7	15

หลังจากได้สร้างตัวอย่างปัญหาของการจัดเก็บสินค้าให้มีขนาดใหญ่และซับซ้อนมากขึ้นแล้วนั้น ผู้วิจัยจึงได้ทำการทดสอบประสิทธิภาพในการหาคำตอบของโปรแกรม LINGO กับตัวอย่างปัญหาทั้ง 10 กรณี ผลการหาคำตอบของโปรแกรม LINGO กับตัวอย่างปัญหาการจัดเก็บสินค้าทั้งหมด แสดงได้ดังตารางที่ 4.8

ตารางที่ 4.8 ผลการหาคำตอบของโปรแกรม LINGO กับตัวอย่างปัญหาการจัดเก็บสินค้า

ตัวอย่างปัญหา	จำนวนชนิดสินค้า	จำนวนช่องจัดเก็บที่ต้องการใช้	จำนวนช่องจัดเก็บในคลังสินค้า	ค่าคำตอบจาก LINGO	เวลาที่ใช้
WH01	3	38	48	588.4625	8 วินาที
WH02	60	259	300	47273.04	7 วินาที
WH03	135	840	1000	112705.2	4 นาที 40 วินาที
WH04	270	1675	2000	204283.6	36 นาที 13 วินาที
WH05	500	2825	3360	290062.9	2 ชั่วโมง 24 นาที
WH06	600	2534	3360	Undetermined	-
WH07	500	2825	4992	Undetermined	-
WH08	800	3993	4992	Undetermined	-
WH09	500	2825	6300	Undetermined	-
WH10	1000	5040	6300	Undetermined	-

เนื่องจากโปรแกรม LINGO ใช้วิธีการหาคำตอบโดยใช้วิธีตรง (Exact Algorithm) ดังนั้นคำตอบที่ได้จะเป็นคำตอบที่ดีที่สุด (Optimal Solution) แต่ข้อจำกัดโดยทั่วไปของโปรแกรม LINGO

คือ เมื่อปัญหามีขนาดใหญ่ขึ้นหรือมีความซับซ้อนที่มากขึ้น โปรแกรมมักจะไม่สามารถหาคำตอบได้ในเวลาที่ยอมรับได้เนื่องจากใช้เวลาในการประมวลผลนานมาก หรือในกรณีที่ปัญหามีความซับซ้อนมาก โปรแกรม LINGO อาจจะไม่สามารถหาคำตอบได้เลย

จากตารางที่ 4.8 จะเห็นได้ว่าโปรแกรม LINGO สามารถหาคำตอบที่ดีที่สุดได้ในปัญหาที่ WH01 – WH05 แต่หากปัญหามีขนาดใหญ่ขึ้น เช่นปัญหาที่ WH06 – WH10 ซึ่งมีจำนวนชนิดสินค้าที่เพิ่มขึ้น ปริมาณสินค้าที่มากขึ้น หรือมีปริมาณช่องเก็บสินค้าที่มากขึ้นนั้น โปรแกรม LINGO จะไม่สามารถหาคำตอบได้เนื่องจากเกินขีดจำกัดในการประมวลผลของโปรแกรม โดยเมื่อโปรแกรมเริ่มทำการประมวลผลจนกระทั่งเวลาผ่านไปประมาณ 30 นาที โปรแกรมจะแจ้งเตือนความผิดพลาดว่าไม่สามารถทำการประมวลผลและแสดงค่าใดๆ ของแบบจำลองได้ เนื่องจากตัวโปรแกรมมีพื้นที่ไม่เพียงพอสำหรับการสร้างชุดข้อมูลและประมวลผล ถึงแม้ว่าจะทำการปรับตั้งค่าโปรแกรมให้มีพื้นที่หน่วยความจำการประมวลผลให้มีค่าสูงสุดแล้วก็ตาม

การเพิ่มการทดสอบสำหรับปัญหาที่มีขนาดใหญ่ขึ้นทำให้ทราบถึงข้อจำกัดในการใช้โปรแกรม LINGO ในการแก้ปัญหาการจัดเก็บสินค้าในงานวิจัยนี้ ซึ่งข้อจำกัดนี้นำไปสู่ที่มาของการพัฒนาวิธีการคำตอบโดยวิธีการดิฟเฟอเรนเชียลอิโวลูชั่น ซึ่งเป็นหนึ่งในวิธีการประเภท Evolutionary Algorithm โดยสามารถแก้ปัญหาที่มีขนาดใหญ่และซับซ้อนได้ในเวลาที่รวดเร็ว

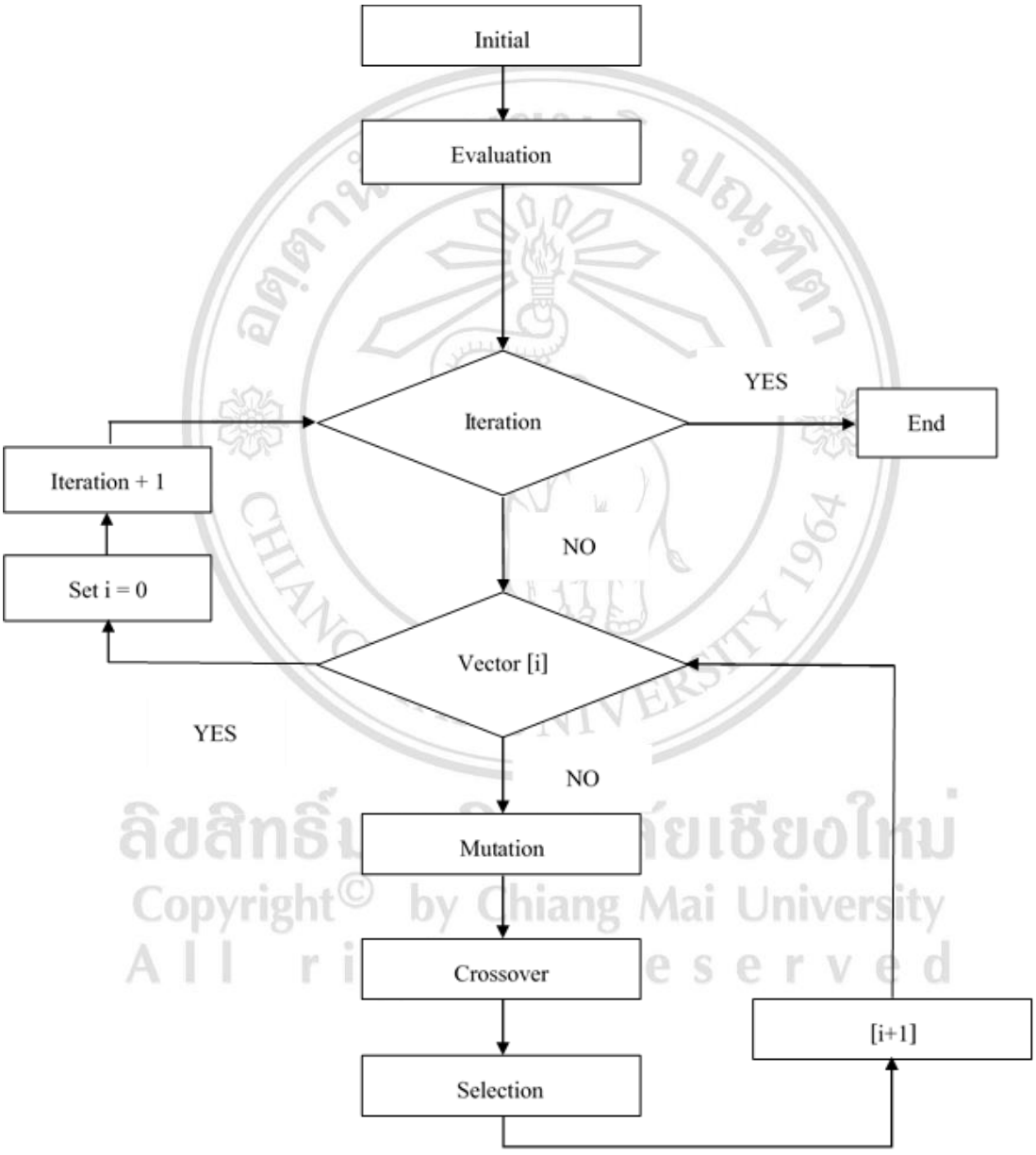
4.4 ผลการพัฒนาการหาคำตอบด้วยวิธีดิฟเฟอเรนเชียลอิโวลูชัน

4.4.1 การประยุกต์ใช้วิธีดิฟเฟอเรนเชียลอิโวลูชันสำหรับปัญหาการจัดเก็บสินค้า

งานวิจัยนี้ได้พัฒนาวิธีการหาคำตอบสำหรับปัญหาการจัดเก็บสินค้าโดยวิธีการดิฟเฟอเรนเชียลอิโวลูชัน (Differential Evolution, DE) เนื่องจากวิธี DE นี้มีหลักการในการหาคำตอบที่รวดเร็วและมีประสิทธิภาพ วิธีนี้ไม่สามารถรับประกันได้ว่าคำตอบที่ได้จะเป็นคำตอบที่ดีที่สุด แต่ก็เป็นคำตอบที่ดีเพียงพอที่จะนำไปใช้ประโยชน์ ซึ่งเป็นผลมาจากวิธีการหาค่าคำตอบแบบครอบคลุม โดยใช้วิธีเชิงพันธุกรรมและการวิวัฒนาการคำตอบเช่นเดียวกับวิธี GA แต่มีข้อดีที่โดดเด่นกว่าคือมีโครงสร้างของระเบียบวิธีที่ซับซ้อนน้อยกว่า มีความยืดหยุ่นมากกว่า ในขณะประมวลผลแต่ละรอบเมื่อคำนวณพบค่าคำตอบที่ดีแล้ว จะนำค่านั้นไปใช้เป็นค่าเป้าหมายตั้งต้นเพื่อหาค่าคำตอบที่ดีกว่าทันที จึงลดระยะเวลาในการคัดเลือกคำตอบได้ดีกว่า GA ที่ต้องประมวลค่าคำตอบให้ครบตามกำหนดก่อนแล้วจึงนำค่าคำตอบมาเรียงลำดับเพื่อคัดเลือกคำตอบ ผลคือสามารถแก้ปัญหาที่มีขนาดใหญ่และซับซ้อนได้ในเวลาอันรวดเร็ว จึงมีความเหมาะสมสำหรับการนำมาประยุกต์ใช้แก้ปัญหาการจัดเก็บสินค้าในงานวิจัยนี้

4.4.2 การหาคำตอบด้วยวิธีดิฟเฟอเรนเชียลอิโวลูชัน และรหัสโปรแกรม C#

การพัฒนาวิธีการหาคำตอบเพื่อแก้ปัญหาการจัดเก็บสินค้าโดยวิธีการดิฟเฟอเรนเชียลอิโวลูชัน (Differential Evolution, DE) ในงานวิจัยนี้จะใช้ภาษา Microsoft Visual Basic C# โดยขั้นตอนการแก้ปัญหาโดยวิธี DE ที่ใช้ในงานวิจัยนี้แสดงได้ดังรูปที่ 4.7



รูปที่ 4.7 แผนผังการทำงานของวิธีดิฟเฟอเรนเชียลอิโวลูชันสำหรับงานวิจัยนี้

จากรูปที่ 4.7 เป็นผังงาน (Flow Chart) ที่แสดงขั้นตอนการทำงานของ DE ที่นำมาประยุกต์ใช้กับปัญหาในงานวิจัยนี้

ขั้นตอนการหาคำตอบด้วยวิธี DE สามารถอธิบายขยายความจากรูปที่ 4.7 ได้ดังนี้

1. ขั้นตอนเริ่มต้น เป็นการสุ่มค่าประชากรเริ่มต้น การกำหนดจำนวนเวกเตอร์คำตอบและการกำหนดมิติซึ่งเป็นขนาดของเวกเตอร์เพื่อนำไปใช้ในการหาคำตอบ
2. ขั้นตอนการแปลงค่า เป็นการนำค่าประชากรเริ่มต้นที่สร้างขึ้น มาทำการแปลงค่าให้ตรงกับองค์ประกอบของปัญหาที่ต้องการนำมาประมวลผล
3. กำหนดจำนวนรอบการคำนวณหาคำตอบ เป็นการกำหนดจำนวนรอบการประมวลผล
4. ขั้นตอน Mutation ทำการพัฒนาคำตอบด้วยสร้างเวกเตอร์ตัวใหม่ ด้วยการใช้สมการ

$$V_{i,g} = X_{r1,g} + F(X_{r2,g} - X_{r3,g})$$

เวกเตอร์ $X_{r1,}, X_{r2,}$ และ X_{r3} คือเวกเตอร์ที่ถูกสุ่มจากประชากรเริ่มต้น ซึ่งต้องเป็นเวกเตอร์ที่ไม่ซ้ำกัน ส่วนค่า $_F$ คือ Scale Factor ที่มีค่าเป็นบวก โดยค่า $_F$ จะเป็นตัวกำหนดค่าความแตกต่างของเวกเตอร์

5. ขั้นตอน Crossover เป็นกระบวนการเพิ่มความหลากหลายของคำตอบ ในงานวิจัยนี้ได้เลือกใช้การ Crossover แบบ Single Point Crossover โดยตำแหน่งการ Crossover จะทำการสุ่มตำแหน่งใหม่ในทุกๆ รอบการประมวลผล ค่า Cr คือค่าความน่าจะเป็นในการ Crossover ซึ่งค่าจะอยู่ในช่วง [0,1]
6. ขั้นตอนการเลือกคำตอบ เป็นการเลือกค่าคำตอบที่ดีที่สุดจากเวกเตอร์ที่ผ่านการประมวลผล ค่าคำตอบที่ดีที่สุดในแต่ละรอบการคำนวณจะถูกนำไปเป็นตัวอย่างคำตอบในการคำนวณรอบถัดไป เมื่อทำครบจำนวนรอบการคำนวณที่ได้กำหนดไว้ จะได้ค่าคำตอบที่ดีที่สุด

ในส่วนถัดไปจะเป็นตัวอย่างการแสดงรหัสข้อมูลด้วยโปรแกรม C# ที่ประยุกต์ใช้วิธี DE เพื่อหาคำตอบสำหรับตัวอย่างปัญหาการจัดเก็บสินค้า โดยได้ทำการยกตัวอย่างตัวอย่างปัญหา WH08 รายละเอียดในโปรแกรมสามารถแสดงได้ดังรูปที่ 4.8 ถึงรูปที่ 4.20

```
{
    class DataBase
    {
        public static string Case = "Case 8 ";

        public double[] SP(int numberofproduct)
        {
            String input = File.ReadAllText(Case + "SP Data.txt");

            double[] SP_Value = new double[numberofproduct];
            int i = 0, j = 0;
            foreach (var row in input.Split('\n'))
            {
                j = 0;
                foreach (var col in row.Trim().Split('\t'))
                {
                    SP_Value[j] = double.Parse(col.Trim());
                    j++;
                }
                i++;
            }
            return SP_Value;
        }

        public double[] TP(int numberofproduct)
        {
            String input = File.ReadAllText(Case + "TP Data.txt");

            double[] TP_Value = new double[numberofproduct];
            int i = 0, j = 0;
            foreach (var row in input.Split('\n'))
            {
                j = 0;
                foreach (var col in row.Trim().Split('\t'))
                {
                    TP_Value[j] = double.Parse(col.Trim());
                    j++;
                }
                i++;
            }
            return TP_Value;
        }
    }
}
```


รูปที่ 4.8 รหัสโปรแกรม C# ส่วนที่ 1 สำหรับตัวอย่างปัญหาการจัดเก็บสินค้า

การกรอกข้อมูลนำเข้า เป็นข้อมูลเกี่ยวกับสินค้าที่ต้องจัดเก็บ ซึ่งประกอบไปด้วย T_p จำนวนครั้งเฉลี่ยในการจัดเก็บสินค้าแต่ละชนิด และ S_p จำนวนช่องจัดเก็บที่สินค้าแต่ละชนิดต้องการใช้

```
{
 class Distance
  {
 public double[] SetDistanceMatrix(int numberofslot, int I , int J, int K ,int R)
        {
            double []distance = new double [numberofslot];
            double Distx = 2.0;
            double Disty = 2.0;
            double L = 2.0;
            double Wa = 4.0;
            double Wr = 3.0;
            double H = 1.7;
            int m = 0;
                for (int i = 1; i < (I + 1); i++)
                {
                    for (int j = 1; j < (J + 1); j++)
                    {
                        for (int k = 1; k < (K + 1); k++)
                        {
                            for (int r = 1; r < (R + 1); r+
                            {
                             distance[m] = (((((Distx + ((i - 0.5) * L)) +
                                                (Disty + (0.5 * j * Wa) +
                                            ((j - 1.0) * (Wr + (0.5 * Wa))) +
                                    ((Wr + Wa) * (r - 1.0))) + (H * (k - 1.0))))));
                              //Console.WriteLine("distance: {0}", distance[m]);
                                    m++;
                            }
                        }
                    }
                }
            return distance;
        }
  }
}
```

รูปที่ 4.9 รหัสโปรแกรม C# ส่วนที่ 2 สำหรับตัวอย่างปัญหาการจัดเก็บสินค้า

การกำหนดค่าตัวแปร และวิธีคำนวณระยะทางการจัดเก็บสินค้า โดยระยะทางเริ่มจากจุดเริ่มต้นไป ยังช่องเก็บสินค้าแต่ละช่อง ซึ่งประกอบไปด้วยการพิจารณาระยะทางการเคลื่อนที่ทั้งในแนวราบและแนวดิ่ง

```
public class Variables
{
 public static int Availiable_Slot = 4992;// dimension
 public static int I = 26;
 public static int J = 2;
 public static int K = 6;
 public static int R = 16;
 public static int Number_of_product = 801; //number of product +1 for empty
product
 public static int Interation = 500;
}
```

รูปที่ 4.10 รหัสโปรแกรม C# ส่วนที่ 3 สำหรับตัวอย่างปัญหาการจัดเก็บสินค้า

ในส่วนนี้ประกอบด้วยขั้นตอนการประกาศตัวแปรสำหรับการกำหนดจำนวนช่องจัดเก็บซึ่งมีผลต่อแผนผังคลังสินค้า การกำหนดจำนวนชนิดสินค้าทั้งหมดที่ต้องถูกจัดเก็บ และการกำหนดจำนวนรอบที่ต้องการใช้ในการหาคำตอบ

```
public class Data
{
  public static double [] SP_Data = new double[Variables.Number_of_product];
  public static double [] TP_Data = new double[Variables.Number_of_product];
  public static double [] TP_SP_Divide = new double[Variables.Number_of_product];
  public static double [] TPbySP_Support = new
double[Variables.Number_of_product];
  public static double [] Encode_Product = new
double[Variables.Number_of_product];
  public static double [] ReNumbering = new double[Variables.Number_of_product];
  public static double [] DimensionSort = new double[Variables.Availiable_Slot];
  public static double [] Slotput = new double[Variables.Availiable_Slot];
}
public class Fitness
{
  public static double [] Isolate = new double[Variables.Availiable_Slot];
  public static double min;
}
```

รูปที่ 4.11 รหัสโปรแกรม C# ส่วนที่ 4 สำหรับตัวอย่างปัญหาการจัดเก็บสินค้า

การประกาศกลุ่มข้อมูล (Data Array) การประกาศชุดค่าคำตอบ (Fitness Array) และการประกาศการ ซึ่งกลุ่มข้อมูลดังกล่าวใช้สำหรับบรจุค่าต่างๆ เพื่อนำไปใช้ในกระบวนการวิเคราะห์ปัญหาของแบบจำลอง

```
public class DistanceMatrix
{
  public static double[] Set = new double[Variables.Availiable_Slot];
}
public Form1()
{
  InitializeComponent();
  Distance Get = new Distance();
  DistanceMatrix.Set =
Get.SetDistanceMatrix(Variables.Availiable_Slot,Variables.I,Variables.J,Variables
.K,
Variables.R);
            Array.Sort(DistanceMatrix.Set);
```

รูปที่ 4.12 รหัสโปรแกรม C# ส่วนที่ 5 สำหรับตัวอย่างปัญหาการจัดเก็บสินค้า

การประกาศค่าระยะทางที่ใช้ในการจัดเก็บสินค้า และการเรียกใช้ค่าดังกล่าวมาทำการคำนวณและจัดเก็บข้อมูลระยะทางการเคลื่อนที่ไว้ใน (DistanceMatrix.Set)

```
DataBase Set = new DataBase();
Data.SP_Data = Set.SP(Variables.Number_of_product);
Data.TP_Data = Set.TP(Variables.Number_of_product);
    for (int i = 0; i <= Data.TP_Data.Length - 1; i++)
      {
        Data.TP_SP_Divide[i] = (double)Data.TP_Data[i] / (double)Data.SP_Data[i];
      }

Array.Copy(Data.TP_SP_Divide,Data.TPbySP_Support,Data.TP_SP_Divide.Length);
TP_SP_Rearrange Sort = new TP_SP_Rearrange();
Data.Encode_Product                                                    =
Sort.SortOrder(Data.TP_SP_Divide,Variables.Number_of_product);
Data.ReNumbering =
```

```
Sort.SortProductNumber(Data.Encode_Product,Data.SP_Data,Variables.Number_of_produ
ct);
}
```

รูปที่ 4.13 รหัสโปรแกรม C# ส่วนที่ 6 สำหรับตัวอย่างปัญหาการจัดเก็บสินค้า

การเรียกใช้ค่าข้อมูลเกี่ยวกับสินค้าที่ต้องจัดเก็บ ซึ่งคือค่าจำนวนครั้งเฉลี่ยในการจัดเก็บ-นำออกของสินค้าแต่ละชนิด (T_p) และจำนวนช่องจัดเก็บที่สินค้าแต่ละชนิดต้องการใช้ (S_p) และสร้างชุดคำสั่งให้ทำการหารค่าทั้งสอง เพื่อสร้างชุดข้อมูลลำดับการจัดเก็บสินค้า

```
public static double minest = double.MaxValue;
public static double GetFitnessvalue(double[] Fitarr,int NoSlot)
  {
    DimensionSort Sort = new DimensionSort();
    Data.DimensionSort =
Sort.Dimension_Rearrange(Fitarr,Variables.Availiable_Slot);
     //double [] Sorted = new double [Variables.Availiable_Slot];
     //Array.Copy(Data.DimensionSort, Sorted, Data.DimensionSort.Length);
     int Slot_Number_Run = 0;
     int Each_Product_count = 1;
     int Product_idx_Run = 0;
       do
         {
           if (Product_idx_Run <= Variables.Number_of_product - 1)
             {
               if (Each_Product_count <= Data.ReNumbering[Product_idx_Run])
                {

Data.Slotput[Convert.ToInt32(Data.DimensionSort[Slot_Number_Run])] =
                              Data.Encode_Product[Product_idx_Run];
                        //Console.WriteLine(Data.Slotput[Slot_Number_Run]);
                        Each_Product_count++;
                        Slot_Number_Run++;
                }
                else
                {
                    Each_Product_count = 1;
                    Product_idx_Run++;
                }
               }
            } while (Slot_Number_Run <= Variables.Availiable_Slot - 1);

           Data.Slotput[0] = Data.Encode_Product[0];

           double [] Fitness_Isolate = new double [Variables.Availiable_Slot];
           double Fitness_Sum = 0;
           for (int i = 0; i <= Variables.Availiable_Slot - 1; i++)
           {
               Fitness.Isolate[i] =
           Math.Round(Data.TPbySP_Support[Convert.ToInt32(Data.Slotput[i])] *
                                                   DistanceMatrix.Set[i], 4);
           }
```

```
                for (int i = 0; i <= Variables.Availiable_Slot-1; i++)
                {
                    Fitness_Sum += Fitness.Isolate[i];
                }
                return Fitness_Sum;
        }
```

รูปที่ 4.14 รหัสโปรแกรม C# ส่วนที่ 7 สำหรับตัวอย่างปัญหาการจัดเก็บสินค้า

ส่วนที่ 1 เป็นการคำนวณเพื่อจัดลำดับการจัดเก็บสินค้า

ส่วนที่ 2 เป็นการกำหนดวิธีที่จะใช้ในการหาค่าคำตอบ ซึ่งคือค่าระยะทางรวมที่น้อยที่สุด โดยการแปลงค่าคำตอบให้ทราบชนิดสินค้า ทราบว่าสินค้าแต่ละชิ้นถูกจัดเก็บที่ช่องจัดเก็บไหน แล้วจึงทำการแปลงค่าเป็นระยะทางการจัดเก็บ โดยทำการหาระยะทางจัดเก็บทีละช่องเก็บสินค้า แล้วจึงนำระยะทางทั้งหมดมารวมกันจึงได้คำตอบ

```
private void Form1_Load(object sender, EventArgs e)
        {

        }

        public void DE()
        {
            Stopwatch wacth = new Stopwatch();
            wacth.Start();
            int Dimension = Variables.Availiable_Slot;
            int Interation = Variables.Interation;
            int Random_Position_1;
            int Random_Position_2;

            int Vector_Amount = 20;    //
            double[] OriginalFitnessvalue = new double[Vector_Amount];
            double[] TrialFitnessvalue = new double[Vector_Amount];
            double[] TempVector = new double[Dimension];
            double[] SubTempVector = new double[Dimension];
            double F = 15;    //

            //double min = 0;
            double Cross_Over_Rate = 1.0;    //
```

```
            Random Rand = new Random();
            double[,] OriginalVector = new double[Vector_Amount, Dimension];
            double[,] MutantVector = new double[Vector_Amount, Dimension];
            double[,] TrialVector = new double[Vector_Amount, Dimension];
            double[] Chartinteration = new double[Interation];
            double[] ChartFitness = new double[Interation];
```

รูปที่ 4.15 รหัสโปรแกรม C# ส่วนที่ 8 สำหรับตัวอย่างปัญหาการจัดเก็บสินค้า

การประกาศตัวแปรที่ต้องถูกนำมาใช้ในการหาคำตอบด้วยวิธีดิฟเฟอเรนเชียลอิโวลูชัน ซึ่งประกอบด้วยการกำหนดค่า F ที่ต้องใช้ในขั้นตอน Mutation และ ค่า Crossover Rate ที่ต้องใช้ในขั้นตอนสร้างความหลากหลายของชุดคำตอบ ส่วนสุดท้ายคือจำนวนเวกเตอร์เพื่อใช้บันทึกค่าคำตอบ การสร้างกลุ่มเวกเตอร์คำตอบเบื้องต้น และการสร้างกลุ่มเวกเตอร์ที่ผ่านการพัฒนาคำตอบ ซึ่งขนาดของกลุ่มเวกเตอร์ทั้ง 2 ชนิดนี้ จะมีค่าเท่ากับจำนวนช่องจัดเก็บสินค้าในการทดสอบปัญหากรณีนั้นๆ

```
for (int j = 0; j <= Vector_Amount - 1; j++)
            {
                for (int i = 0; i <= Dimension - 1; i++)
                {
                    OriginalVector[j, i] = Rand.NextDouble();
                }
            }
```

รูปที่ 4.16 รหัสโปรแกรม C# ส่วนที่ 9 สำหรับตัวอย่างปัญหาการจัดเก็บสินค้า

ขั้นตอนการเริ่มต้นของกระบวนการหาคำตอบของวิธี DE ซึ่งจะทำการสุ่มค่าสำหรับประชากรเริ่มต้นให้กับทุกกลุ่มเวกเตอร์

```
//Start interation==================================================
            for (int interation = 1; interation <= Interation; interation++)
            {
                for (int vector_count = 0; vector_count <= Vector_Amount - 1;

                     vector_count++)
                {
                    for (int i = 0; i <= Dimension - 1; i++)
                    {
                        TempVector[i] = OriginalVector[vector_count, i];
                    }
                    OriginalFitnessvalue[vector_count] =

GetFitnessvalue(TempVector,Variables.Availiable_Slot);
```

รูปที่ 4.17 รหัสโปรแกรม C# ส่วนที่ 10 สำหรับตัวอย่างปัญหาการจัดเก็บสินค้า

จากรูปที่ 4.17 เป็นการสั่งให้ค่าประชากรเวกเตอร์คำตอบหาค่าให้ครบทุกจำนวนรอบการคำนวณ

```
//Mutation Method
  for (int i = 0; i <= Vector_Amount - 1; i++)
     {
      do
         {
          Random_Position_1 = Rand.Next(0, Vector_Amount - 1);
          Random_Position_2 = Rand.Next(0, Vector_Amount - 1);
         }
        while (Random_Position_1 == Random_Position_2 ||
         Random_Position_1== i || Random_Position_2 == i);

          for (int dim = 0; dim <= Dimension - 1; dim++)
           {
             MutantVector[i, dim] = OriginalVector[i, dim] + (F *
             (OriginalVector[i, Random_Position_1] - OriginalVector[i,
              Random_Position_2]));
           }
      }
```

รูปที่ 4.18 รหัสโปรแกรม C# ส่วนที่ 11 สำหรับตัวอย่างปัญหาการจัดเก็บสินค้า

ขั้นตอนการ Mutation ทำการพัฒนาคำตอบด้วยสร้างเวกเตอร์ตัวใหม่ ด้วยการใช้สมการ

$$V_{i,g} = X_{r1,g} + F(X_{r2,g} - X_{r3,g})$$

```
          int CrossOverPoint;
              CrossOverPoint = Rand.Next(0, Dimension-1);

                for (int i = 0; i <= Vector_Amount - 1; i++)
                {
                    //Crossover Method
                    double Random_Cross_Over_Number = Rand.NextDouble();
                    if (Random_Cross_Over_Number <= Cross_Over_Rate)
                    {
                        for (int j = 0; j <= Dimension - 1; j++)
                        {
                            if (j <= CrossOverPoint)
                            {
                                TrialVector[i, j] = MutantVector[i, j];
                            }
                            else
                            {
                                TrialVector[i, j] = OriginalVector[i, j];
                            }
                        }
                    } //Console.WriteLine(" reach =-============ ");
                }
```

รูปที่ 4.19 รหัสโปรแกรม C# ส่วนที่ 12 สำหรับตัวอย่างปัญหาการจัดเก็บสินค้า

การ Crossover คำตอบ แบบ Single-Point โดยทำการสุ่มตำแหน่งทุกรอบในการ Crossover ทำให้ในการหาคำตอบแต่ละรอบ ตำแหน่งการ Crossover จะแตกต่างกันในลักษณะการสุ่มนั่นเอง ซึ่งประโยชน์ที่ได้จากการกำหนดลักษณะนี้ คือได้คำตอบที่มีความหลากหลาย ทำให้มีโอกาสในการหาค่าคำตอบที่ดีที่สุดได้มากขึ้น

```
//Selection Method

      for (int vector_count = 0; vector_count <= Vector_Amount - 1;
vector_count++)
                {//Separate Dimension Array
                    for (int i = 0; i <= Dimension - 1; i++)
                    {
                        SubTempVector[i] = TrialVector[vector_count, i];
                    }
                    TrialFitnessvalue[vector_count] =

      GetFitnessvalue(SubTempVector,Variables.Availiable_Slot);
                }

                for (int k = 0; k <= Vector_Amount - 1; k++)
                {
                    if (TrialFitnessvalue[k] < OriginalFitnessvalue[k])
                    {
                        for (int l = 0; l <= Dimension - 1; l++)
                        {
                            OriginalVector[k, l] = TrialVector[k, l];
                        }
                    }
                }

                Array.Clear(TrialVector, 0, TrialVector.Length);
                if (TrialFitnessvalue.Min() > OriginalFitnessvalue.Min())
                {
                    Fitness.min = OriginalFitnessvalue.Min();
                }
                else
                {
                    Fitness.min = TrialFitnessvalue.Min();
                }
```

รูปที่ 4.20 รหัสโปรแกรม C# ส่วนที่ 13 สำหรับตัวอย่างปัญหาการจัดเก็บสินค้า

ทำการเปรียบเทียบค่าคำตอบของเวกเตอร์เริ่มต้น กับ เวกเตอร์ที่ถูกพัฒนา จากนั้นทำการเลือกค่าตัวที่ดีที่สุด เพื่อนำไปเป็นค่าคำตอบตั้งต้นในการคำนวณของโปรแกรมรอบถัดไป และเมื่อหาคำตอบจนครบจำนวนรอบที่กำหนดไว้ จะทำให้ได้ค่าคำตอบที่ดีที่สุด ซึ่งในงานวิจัยนี้คือค่า ระยะทางรวมในการจัดเก็บ-นำออกสินค้า ที่มีระยะทางน้อยที่สุดนั่นเอง นอกจากนี้ผู้วิจัยได้ทำการออกแบบวิธีการแปลงคำตอบที่เหมาะสม ซึ่งมีความสำคัญต่อ รหัสโปรแกรมในรูปที่ 4.14 การออกแบบดังกล่าวจะถูกนำเสนอในหัวข้อ 4.4.3

4.4.3 การพัฒนาและออกแบบวิธีการแปลงคำตอบ

ในงานวิจัยนี้ได้ทำการพัฒนาและออกแบบวิธีการแปลงคำตอบ (Solution Mapping) ให้มีความสอดคล้องกับตัวอย่างปัญหาการจัดเก็บสินค้า เพื่อให้ผลลัพธ์จากการประมวลผลมีความเหมาะสมและสามารถนำคำตอบจากการเขียนโปรแกรมมาวิเคราะห์ผลได้อย่างถูกต้อง

ประชากรหรือเวกเตอร์ใน DE นั้นประกอบด้วยค่าของแต่ละมิติซึ่งเป็นค่าสุ่ม จำนวนของมิติทั้งหมดในแต่ละเวกเตอร์มีขนาดเท่ากับ N มิติ ในงานวิจัยนี้ เวกเตอร์ขนาด N มิติถูกกำหนดให้มีขนาดเท่ากับปริมาณช่องเก็บสินค้าทั้งหมดในคลังสินค้า
การแสดงวิธีการแปลงคำตอบของการเขียนโปรแกรมที่ทำงานร่วมกับวิธี DE ในงานวิจัยนี้สามารถแสดงรายละเอียดได้ตามตัวอย่างปัญหาดังต่อไปนี้

ในตัวอย่างปัญหาขนาดเล็ก (WH01) จะกำหนดให้คลังสินค้ามีตำแหน่งหลักของช่องจัดเก็บสินค้ามีจำนวน 4 ตำแหน่ง (i = 1, 2, 3, 4) ตำแหน่งแถวของช่องจัดเก็บสินค้ามีจำนวน 2 ตำแหน่ง (j = 1, 2) มีจำนวนชั้นสำหรับช่องจัดเก็บสินค้า 3 ชั้น (k = 1, 2, 3) ตำแหน่งกลุ่มของช่องจัดเก็บสินค้ามีจำนวน 2 กลุ่ม (r = 1, 2) และมีสินค้าจำนวน 3 ชนิด แสดงดังตารางที่ 4.9

ตารางที่ 4.9 แสดงชุดข้อมูลของสินค้าที่ใช้ในตัวอย่างปัญหาขนาดเล็ก (WH01)

ชนิดสินค้า	T_p	S_p	T_p / S_p
1	15	17	0.882
2	20	14	1.429
3	5	7	0.714

ในตัวอย่างปัญหา WH01 ขนาดของเวกเตอร์จะถูกกำหนดให้มีขนาดเท่ากับปริมาณช่องเก็บสินค้าทั้งหมดในคลังสินค้าซึ่งมีค่าเท่ากับ 48

ขั้นตอนเริ่มต้นการแปลงคำตอบให้อยู่ในรูปแบบของ DE จะมีการกำหนดค่าเริ่มต้นของแต่ละมิติในเวกเตอร์เป็นค่าสุ่มระหว่าง 0 ถึง 1 สำหรับตัวอย่างการสมมติการสุ่มเลขประชากรของตัวอย่างปัญหา WH01 สามารถแสดงได้ดังรูปที่ 4.21

มิติ	1	2	3	4	5	6	7	8	9	10	11	12
ค่าจากการสุ่ม	0.29	0.61	0.47	0.32	0.12	0.37	0.56	0.94	0.44	0.13	0.33	0.68

มิติ	13	14	15	16	17	18	19	20	21	22	23	24
ค่าจากการสุ่ม	0.64	0.75	0.15	0.89	0.58	0.78	0.20	0.02	0.21	0.54	0.98	0.59

มิติ	25	26	27	28	29	30	31	32	33	34	35	36
ค่าจากการสุ่ม	0.17	0.24	0.71	0.23	0.08	0.69	0.88	0.63	0.25	0.27	0.18	0.83

มิติ	37	38	39	40	41	42	43	44	45	46	47	48
ค่าจากการสุ่ม	0.19	0.06	0.79	0.66	0.26	0.85	0.72	0.51	0.91	0.22	0.41	0.80

รูปที่ 4.21 แสดงขั้นตอนเริ่มต้นการแปลงคำตอบของตัวอย่างปัญหา WH01

ขั้นตอนถัดมาคือการเรียงค่าในแต่ละมิติของเวกเตอร์จากน้อยไปมาก เพื่อกำหนดตำแหน่งการจัดเก็บสินค้าแต่ละชนิดในช่องเก็บสินค้า จากข้อมูลในตารางที่ 4.9 สินค้าชนิดที่ 2 มีค่าอัตราการหมุนเวียน (*Tp / Sp*) ที่สูงกว่าสินค้าชนิดที่ 1 และ 3 ดังนั้นสินค้าชนิดที่ 2 ทั้งหมดจะต้องถูกจัดเก็บก่อน ในการจัดสรรตำแหน่งการจัดเก็บนั้น สินค้าชนิดที่ 2 ทั้งหมดจะถูกจัดเก็บในตำแหน่งที่มีค่าของมิติต่ำที่สุดก่อนจนกระทั่งสินค้าชนิดที่ 2 ทั้งหมดถูกจัดสรรลงไป หลังจากนั้นสินค้าชนิดที่ 1 และ 3 จึงจะถูกจัดเก็บลงตำแหน่งที่มีค่าของมิติมากขึ้นไปตามลำดับ

และเนื่องจากจำนวนช่องเก็บในคลังสินค้าทั้งหมดมี 48 ช่อง แต่ผลรวมจำนวนสินค้าทั้ง 3 ชนิดมีความต้องการใช้ช่องจัดเก็บจำนวน 38 ช่อง ดังนั้นจะมี 10 ช่องเก็บที่ไม่มีสินค้าอยู่ ผลการแปลงคำตอบในขั้นตอนการเรียงลำดับการจัดเก็บสินค้า จะแสดงได้ดังรูปที่ 4.22

มิติ	20	38	29	5	10	15	25	35	37	19	21	46
ค่าจากการสุ่ม	0.02	0.06	0.08	0.12	0.13	0.15	0.17	0.18	0.19	0.20	0.21	0.22
ชนิดสินค้า	2	2	2	2	2	2	2	2	2	2	2	2

มิติ	28	26	33	41	34	1	4	11	6	47	9	3
ค่าจากการสุ่ม	0.23	0.24	0.25	0.26	0.27	0.29	0.32	0.33	0.37	0.41	0.44	0.47
ชนิดสินค้า	2	2	1	1	1	1	1	1	1	1	1	1

มิติ	44	22	7	17	24	2	32	13	40	12	30	27
ค่าจากการสุ่ม	0.51	0.54	0.56	0.58	0.59	0.61	0.63	0.64	0.66	0.68	0.69	0.71
ชนิดสินค้า	1	1	1	1	1	1	1	3	3	3	3	3

มิติ	43	14	18	39	48	36	42	31	16	45	8	23
ค่าจากการสุ่ม	0.72	0.75	0.78	0.79	0.80	0.83	0.85	0.88	0.89	0.91	0.94	0.98
ชนิดสินค้า	3	3	-	-	-	-	-	-	-	-	-	-

รูปที่ 4.22 แสดงขั้นตอนการแปลงคำตอบการเรียงลำดับการจัดเก็บสินค้า ตัวอย่างปัญหา WH01

ประโยชน์ของการแปลงคำตอบที่ในงานวิจัยนี้คือ สามารถสร้างคำตอบที่เป็นไปได้ทั้งหมด (Feasible Solution) ผลของการแปลงคำตอบตำแหน่งการจัดเก็บสินค้า ตัวอย่างปัญหา WH01 ซึ่งเป็นตัวอย่างปัญหาขนาดเล็ก สามารถแสดงได้ดังรูปที่ 4.23

มิติ	1	2	3	4	5	6	7	8	9	10	11	12
ค่าจากการสุ่ม	0.29	0.61	0.47	0.32	0.12	0.37	0.56	0.94	0.44	0.13	0.33	0.68
ชนิดสินค้า	1	1	1	1	2	1	1	-	1	2	1	3
ตำแหน่งช่องเก็บ	18	30	24	19	4	21	27	47	23	5	20	34

มิติ	13	14	15	16	17	18	19	20	21	22	23	24
ค่าจากการสุ่ม	0.64	0.75	0.15	0.89	0.58	0.78	0.20	0.02	0.21	0.54	0.98	0.59
ชนิดสินค้า	3	3	2	-	1	-	2	2	2	1	-	1
ตำแหน่งช่องเก็บ	32	38	6	45	28	39	10	1	11	26	48	29

มิติ	25	26	27	28	29	30	31	32	33	34	35	36
ค่าจากการสุ่ม	0.17	0.24	0.71	0.23	0.08	0.69	0.88	0.63	0.25	0.27	0.18	0.83
ชนิดสินค้า	2	2	3	2	2	3	-	1	1	1	2	-
ตำแหน่งช่องเก็บ	7	14	36	13	3	35	44	31	15	17	8	42

มิติ	37	38	39	40	41	42	43	44	45	46	47	48
ค่าจากการสุ่ม	0.19	0.06	0.79	0.66	0.26	0.85	0.72	0.51	0.91	0.22	0.41	0.80
ชนิดสินค้า	2	2	-	3	1	-	3	1	-	2	1	-
ตำแหน่งช่องเก็บ	9	2	40	33	16	43	37	25	46	12	22	41

รูปที่ 4.23 แสดงขั้นตอนการแปลงคำตอบตำแหน่งการจัดเก็บสินค้า ตัวอย่างปัญหา WH01

4.4.4 ผลการประยุกต์ใช้วิธีดิฟเฟอเรนเชียลอิโวลูชัน (DE) เพื่อแก้ปัญหาการจัดเก็บสินค้า

ในขั้นตอนนี้เป็นหาคำตอบโดยวิธีการดิฟเฟอเรนเชียลอิโวลูชัน ที่ได้พัฒนาเพื่อการแก้ปัญหาการจัดเก็บสินค้าสำหรับตัวอย่างปัญหา WH01 – WH10 ซึ่งเป็นการทดสอบปัญหาขนาดเล็ก ขนาดกลาง และขนาดใหญ่ เพื่อนำผลลัพธ์ทั้งหมดมาทำการเปรียบเทียบคำตอบเชิงคุณภาพ กับผลลัพธ์ที่ได้จากการโปรแกรม LINGO ทำการประมวลผลบนระบบปฏิบัติการวินโดวส์ 7 (Windows 7) โดยเครื่องคอมพิวเตอร์ที่ใช้มีหน่วยประมวลผลกลาง (CPU) AMD A6-3420M 1.5 GHz และหน่วยความจำ (RAM) ขนาด 4.00 GB

1) การตั้งค่าพารามิเตอร์ในวิธีดิฟเฟอเรนเชียลอิโวลูชัน

ในงานวิจัยนี้พารามิเตอร์ที่เป็นส่วนประกอบสำคัญได้แก่

- จำนวณรอบการคำนวณ (Iteration)
- จำนวนเวกเตอร์ หรือที่เรียกว่าจำนวนประชากรคำตอบของ DE
- ค่า Parameter Scale Factor (f) ซึ่งใช้ในขั้นตอน Mutation
- ค่า Parameter Crossover Rate, (Cr) ซึ่งใช้ในขั้นตอนการสร้างความหลากหลายของคำตอบ

ค่าพารามิเตอร์ดังกล่าวต้องถูกนำมาทดลองเบื้องต้น เพื่อทำการหาการตั้งค่าพารามิเตอร์ที่เหมาะสม เพื่อความเป็นมาตรฐานในการทดสอบตัวอย่างปัญหา และได้ผลลัพธ์คำตอบที่ดีที่สุด การทดสอบการตั้งค่าพารามิเตอร์เบื้องต้น ในงานวิจัยนี้ได้เลือกตัวอย่างปัญหา WH02, WH05 และ WH08 ซึ่งเป็นตัวอย่างปัญหาขนาดเล็ก ขนาดกลาง และขนาดใหญ่ เพื่อให้สามารถหาการตั้งค่าพารามิเตอร์ที่มีความครอบคลุมมากที่สุด

2) การทดสอบเพื่อหาค่าพารามิเตอร์ f และค่าพารามิเตอร์ Cr ที่เหมาะสม

ในขั้นตอนเริ่มต้นทำการทดสอบเพื่อหาค่า f และค่า Cr ที่เหมาะสม ได้ทำการกำหนดจำนวนการคำนวณฟังก์ชัน (Number of Function Evaluation) ให้เป็นค่าคงที่มีค่าเท่ากับ 1000 ซึ่งประกอบไปด้วยจำนวนรอบการคำนวณ 200 ครั้ง และจำนวนเวกเตอร์ 5 ตัว

ตารางที่ 4.10 แสดงค่าที่ใช้ทดสอบการตั้งค่าพารามิเตอร์ f และ Cr

ค่า f	ค่า Cr
0.5	0.1
1	0.5
5	0.9

จากตารางที่ 4.10 เมื่อทำการจับคู่ค่าพารามิเตอร์ f และค่า Cr จากนั้นทำการสลับคู่ทดสอบจนครบทุกตัวอย่างปัญหา ได้ผลลัพธ์ว่าการตั้งค่าพารามิเตอร์ Cr ให้มีค่าเท่ากับ 0.9 จะทำให้ได้ผลลัพธ์ที่ดีที่สุด ซึ่งสอดคล้องกับความหมายของค่าพารามิเตอร์ Cr ที่เป็นการกำหนดความน่าจะเป็นในกระบวนการเพิ่มความหลากหลายของคำตอบ การตั้งค่าต้องอยู่ในช่วง [0,1] หากทำการตั้งค่า Cr ให้มี

ค่ายิ่งมากผลลัพธ์ของคำตอบจะยิ่งมีประสิทธิภาพ ดังนั้นในงานวิจัยนี้จะทำการตั้งค่าพารามิเตอร์ Cr ไว้แบบคงที่ โดยมีค่าเท่ากับ 0.9

สำหรับค่าพารามิเตอร์ f ที่มีการตั้งค่าในการทดสอบเบื้องต้นเท่ากับ 0.5, 1.0 และ 5 พบว่าหากทำการตั้งค่าให้มากกว่า 5 มีแนวโน้มว่าจะทำให้ผลลัพธ์ดียิ่งขึ้นได้อีก และเมื่อการทดสอบข้างต้นสามารถระบุค่าที่เหมาะสมสำหรับพารามิเตอร์ Cr ได้แล้ว ดังนั้นขั้นตอนถัดไปคือการทดสอบหาค่าพารามิเตอร์ f ที่เหมาะสมที่สุด

ตารางที่ 4.11 แสดงค่าและตัวอย่างปัญหาที่ใช้ทดสอบการตั้งค่าพารามิเตอร์ f

การทดสอบ	ตัวอย่างปัญหา	ค่า f	ค่า Cr	จำนวนรอบในการคำนวณ	จำนวนเวกเตอร์
1	WH2	0.5	0.9	200	5
2	WH2	3	0.9	200	5
3	WH2	15	0.9	200	5
4	WH5	0.5	0.9	200	5
5	WH5	3	0.9	200	5
6	WH5	15	0.9	200	5
7	WH8	0.5	0.9	200	5
8	WH8	3	0.9	200	5
9	WH8	15	0.9	200	5

จากตารางที่ 4.11 ค่าพารามิเตอร์Cr จำนวนรอบในการคำนวณ และจำนวนเวกเตอร์ จะถูกตั้งให้เป็นค่าคงที่ และเพิ่มจำนวนครั้งการทดสอบด้วยตัวอย่างปัญหาที่ WH2, WH05 และ WH08 รวมทั้งหมด 9 การทดสอบ เพื่อใช้หาค่าพารามิเตอร์ f ที่เหมาะสม ในแต่ละตัวอย่างปัญหาจะมีการตั้งค่า f ที่แตกต่างกัน โดยมีการกำหนดค่า f เพื่อใช้ในการทดสอบจำนวน 3 ค่า คือ 0.5, 3 และ 15 การทดสอบดังกล่าวสามารถแสดงได้ตามตารางที่ 4.12 ถึงตารางที่ 4.20

ตารางที่ 4.12 การทดสอบที่ 1 เพื่อหาค่าพารามิเตอร์ f ที่เหมาะสม

ตัวอย่างปัญหา	ค่า f	ผล Global Optimum (LINGO)	ผลจากวิธี DE (C#)	% ความคลาดเคลื่อน	ค่าเฉลี่ย (ผลจากวิธี DE)	ค่าเบี่ยงเบนมาตรฐาน (ผลจากวิธี DE)
WH02	0.5	47273.03	47419.82	0.31	47404.53	53.19
WH02	0.5	47273.03	47362.99	0.19		
WH02	0.5	47273.03	47364.99	0.19		
WH02	0.5	47273.03	47490.48	0.46		
WH02	0.5	47273.03	47384.35	0.24		

ตารางที่ 4.13 การทดสอบที่ 2 เพื่อหาค่าพารามิเตอร์ f ที่เหมาะสม

ตัวอย่างปัญหา	ค่า f	ผล Global Optimum (LINGO)	ผลจากวิธี DE (C#)	% ความคลาดเคลื่อน	ค่าเฉลี่ย (ผลจากวิธี DE)	ค่าเบี่ยงเบนมาตรฐาน (ผลจากวิธี DE)
WH02	3	47273.03	47342.01	0.15	47333.05	29.08
WH02	3	47273.03	47362.80	0.19		
WH02	3	47273.03	47319.80	0.10		
WH02	3	47273.03	47351.28	0.17		
WH02	3	47273.03	47289.35	0.03		

ตารางที่ 4.14 การทดสอบที่ 3 เพื่อหาค่าพารามิเตอร์ f ที่เหมาะสม

ตัวอย่างปัญหา	ค่า f	ผล Global Optimum (LINGO)	ผลจากวิธี DE (C#)	% ความคลาดเคลื่อน	ค่าเฉลี่ย (ผลจากวิธี DE)	ค่าเบี่ยงเบนมาตรฐาน (ผลจากวิธี DE)
WH02	15	47273.03	47296.24	0.05	47307.50	32.02
WH02	15	47273.03	47294.21	0.04		
WH02	15	47273.03	47364.64	0.19		
WH02	15	47273.03	47292.37	0.04		
WH02	15	47273.03	47290.02	0.04		

ตารางที่ 4.15 การทดสอบที่ 4 เพื่อหาค่าพารามิเตอร์ f ที่เหมาะสม

ตัวอย่างปัญหา	ค่า f	ผล Global Optimum (LINGO)	ผลจากวิธี DE (C#)	% ความคลาดเคลื่อน	ค่าเฉลี่ย (ผลจากวิธี DE)	ค่าเบี่ยงเบนมาตรฐาน (ผลจากวิธี DE)
WH05	0.5	290062.88	293808.40	1.29	292437.29	956.46
WH05	0.5	290062.88	292978.11	1.01		
WH05	0.5	290062.88	291380.30	0.45		
WH05	0.5	290062.88	291964.30	0.66		
WH05	0.5	290062.88	292055.36	0.69		

ตารางที่ 4.16 การทดสอบที่ 5 เพื่อหาค่าพารามิเตอร์ f ที่เหมาะสม

ตัวอย่างปัญหา	ค่า f	ผล Global Optimum (LINGO)	ผลจากวิธี DE (C#)	% ความคลาดเคลื่อน	ค่าเฉลี่ย (ผลจากวิธี DE)	ค่าเบี่ยงเบนมาตรฐาน (ผลจากวิธี DE)
WH05	3	290062.88	291901.40	0.63	291810.56	797.10
WH05	3	290062.88	292607.67	0.88		
WH05	3	290062.88	291290.31	0.42		
WH05	3	290062.88	292509.75	0.84		
WH05	3	290062.88	290743.66	0.23		

ตารางที่ 4.17 การทดสอบที่ 6 เพื่อหาค่าพารามิเตอร์ f ที่เหมาะสม

ตัวอย่างปัญหา	ค่า f	ผล Global Optimum (LINGO)	ผลจากวิธี DE (C#)	% ความคลาดเคลื่อน	ค่าเฉลี่ย (ผลจากวิธี DE)	ค่าเบี่ยงเบนมาตรฐาน (ผลจากวิธี DE)
WH05	15	290062.88	290955.22	0.31	291344.51	1020.90
WH05	15	290062.88	293081.21	1.04		
WH05	15	290062.88	291359.54	0.45		
WH05	15	290062.88	290472.32	0.14		
WH05	15	290062.88	290854.26	0.27		

ตารางที่ 4.18 การทดสอบที่ 7 เพื่อหาค่าพารามิเตอร์ f ที่เหมาะสม

ตัวอย่างปัญหา	ค่า f	ผล Global Optimum (LINGO)	ผลจากวิธี DE (C#)	% ความคลาดเคลื่อน	ค่าเฉลี่ย (ผลจากวิธี DE)	ค่าเบี่ยงเบนมาตรฐาน (ผลจากวิธี DE)
WH08	0.5	520550.52	546338.37	4.95	548091.11	1949.22
WH08	0.5	520550.52	549832.30	5.62		
WH08	0.5	520550.52	548996.32	5.46		
WH08	0.5	520550.52	545659.93	4.82		
WH08	0.5	520550.52	549628.64	5.59		

ตารางที่ 4.19 การทดสอบที่ 8 เพื่อหาค่าพารามิเตอร์ f ที่เหมาะสม

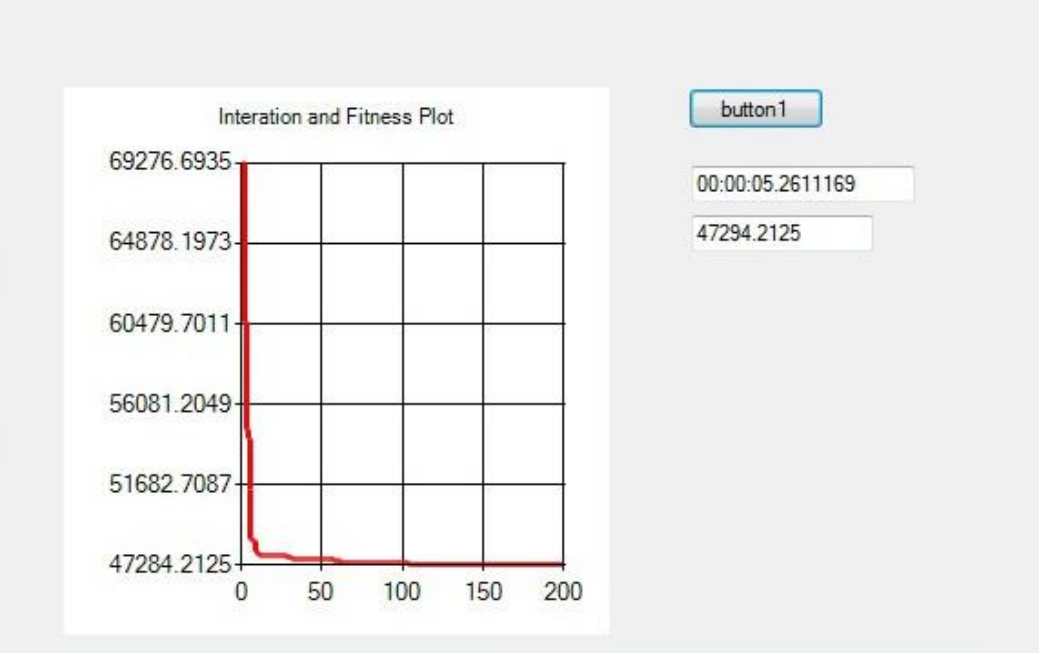
ตัวอย่างปัญหา	ค่า f	ผล Global Optimum (LINGO)	ผลจากวิธี DE (C#)	% ความคลาดเคลื่อน	ค่าเฉลี่ย (ผลจากวิธี DE)	ค่าเบี่ยงเบนมาตรฐาน (ผลจากวิธี DE)
WH08	3	520550.52	542390.54	4.20	543998.73	1549.00
WH08	3	520550.52	542939.91	4.30		
WH08	3	520550.52	546006.92	4.89		
WH08	3	520550.52	543422.48	4.39		
WH08	3	520550.52	545233.78	4.74		

ตารางที่ 4.20 การทดสอบที่ 9 เพื่อหาค่าพารามิเตอร์ f ที่เหมาะสม

ตัวอย่างปัญหา	ค่า f	ผล Global Optimum (LINGO)	ผลจากวิธี DE (C#)	% ความคลาดเคลื่อน	ค่าเฉลี่ย (ผลจากวิธี DE)	ค่าเบี่ยงเบนมาตรฐาน (ผลจากวิธี DE)
WH08	15	520550.52	542427.04	4.20	542010.62	536.40
WH08	15	520550.52	541547.55	4.03		
WH08	15	520550.52	542171.83	4.15		
WH08	15	520550.52	541348.97	4.00		
WH08	15	520550.52	542557.70	4.23		

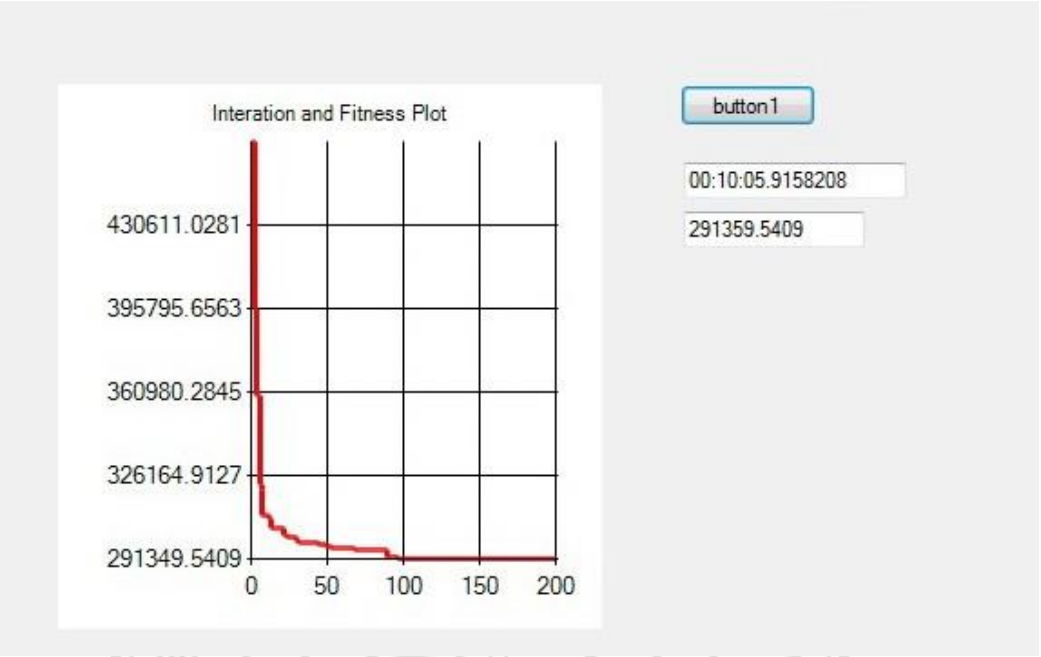
จากการทดสอบปัญหาเพื่อหาการตั้งค่าพารามิเตอร์ f ที่เหมาะสม การทดสอบทั้ง 9 กรณี จะถูกทำการทดสอบ กรณีละ 5 ครั้ง ทำให้สามารถหาค่าเฉลี่ยและค่าเบี่ยงเบนมาตรฐานของชุดคำตอบในแต่ละการทดสอบเพื่อใช้วิเคราะห์ผลได้ จากการทดสอบข้างต้นสามารถสรุปผลได้ว่า

- การกำหนดค่า f ให้มีค่าเท่ากับ 3 และ 15 จะทำให้ค่าคำตอบที่ดี เมื่อนำชุดคำตอบไปทำการเปรียบเทียบกับคำตอบที่ดีที่สุดจากโปรแกรม LINGO จะมีความคลาดเคลื่อนน้อย
- การกำหนดค่า f มีค่าเท่ากับ 3 และ 15 ผลคำตอบในตัวอย่างปัญหาขนาดเล็กและขนาดกลาง จะมีค่าเบี่ยงเบนมาตรฐานที่ใกล้เคียงกัน แต่ในตัวอย่างปัญหาขนาดใหญ่ เช่นในตัวอย่างปัญหาWH08 การกำหนดค่า f ให้มีค่าเท่ากับ 15 จะทำให้ค่าเบี่ยงเบนมาตรฐานมีค่าน้อยอย่างชัดเจน
- การกำหนดค่า f ให้มีค่าเท่ากับ 15 ชุดผลคำตอบที่ได้จะมีค่าเฉลี่ยที่ดีที่สุด มีค่าที่เข้าใกล้ค่าคำตอบที่ดีที่สุดมากกว่าการกำหนดค่า f เท่ากับ 3 และ 0.5



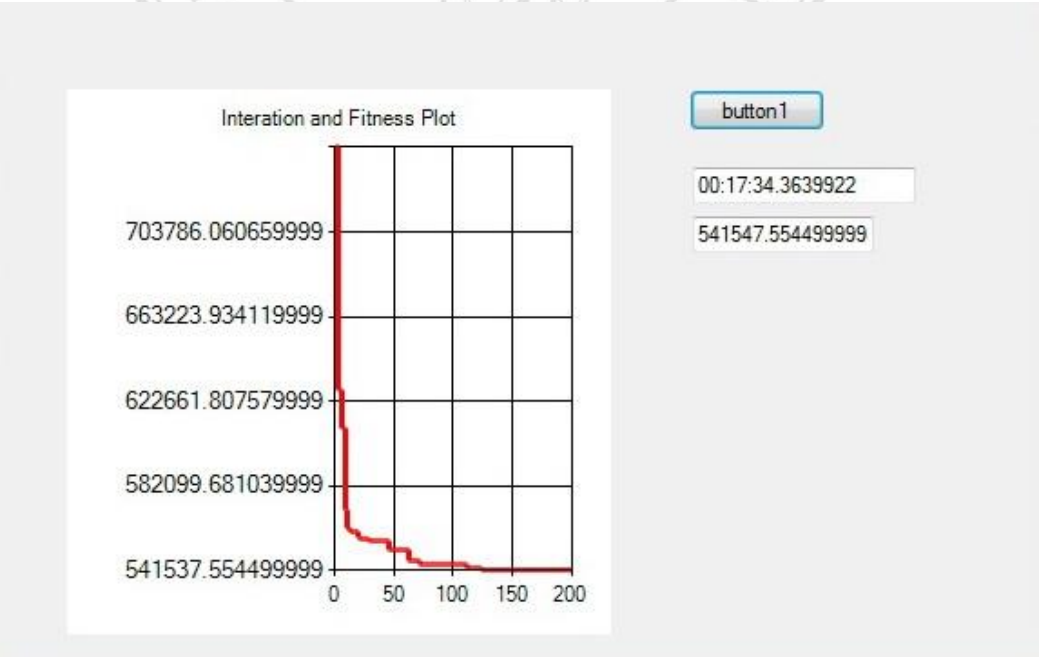
รูปที่ 4.24 แสดงตัวอย่างผลการทดสอบที่ 3 ตัวอย่างปัญหา WH02 เพื่อหาค่าพารามิเตอร์ f ที่เหมาะสม

ผลคำตอบจากการประมวลผลมีค่าเท่ากับ 47294.21 ใช้เวลาในการทดสอบ 5 นาที 26 วินาที



รูปที่ 4.25 แสดงตัวอย่างผลการทดสอบที่ 3 ตัวอย่างปัญหา WH05 เพื่อหาค่าพารามิเตอร์ f ที่เหมาะสม

ผลคำตอบจากการประมวลผลมีค่าเท่ากับ 291359.54 ใช้เวลาในการทดสอบ 10 นาที 6 วินาที



รูปที่ 4.26 แสดงตัวอย่างผลการทดสอบที่ 3 ตัวอย่างปัญหา WH08 เพื่อหาค่าพารามิเตอร์ f ที่เหมาะสม

ผลคำตอบจากการประมวลผลมีค่าเท่ากับ 541547.55 ใช้เวลาในการทดสอบ 17 นาที 34 วินาที

ผลจากจากการทดสอบข้างต้น ทำให้ได้ค่าที่เหมาะสมสำหรับค่าพารามิเตอร์ f และ Cr ซึ่งในขณะทำการทดสอบ ได้กำหนดค่าพารามิเตอร์อีก 2 ชนิดโดยกำหนดให้เป็นค่าคงที่ นั่นคือการกำหนดจำนวนรอบการคำนวณมีค่าเท่ากับ 200 ครั้ง และจำนวนเวกเตอร์ที่ใช้หาคำตอบมีการกำหนดเท่ากับ 5 ตัว หลังจากการพิจารณาผลคำตอบที่ได้ ทำให้เห็นได้ว่าการตั้งค่าจำนวนรอบการคำนวณและจำนวนเวกเตอร์ด้วยค่าดังกล่าว มีความเพียงพอต่อการใช้หาผลคำตอบในทุกขนาดปัญหา และค่าผลคำตอบที่ได้มีประสิทธิภาพดี ดังตัวอย่างผลการทดสอบในรูปที่ 4.24 ถึง 4.26 ดังนั้นการตั้งค่าพารามิเตอร์สำหรับวิธี DE เพื่อใช้ทดสอบตัวอย่างปัญหาทั้งหมดในงานวิจัยนี้จะมีค่าดังต่อไปนี้

ตารางที่ 4.21 แสดงการตั้งค่าพารามิเตอร์ในวิธีดิฟเฟอเรนเชียลอิโวลูชันสำหรับงานวิจัยนี้

ค่า f	ค่า Cr	จำนวนรอบในการคำนวณ	จำนวนเวกเตอร์
15	0.9	200	5

ขั้นตอนถัดไปคือการทดสอบตัวอย่างปัญหาการจัดเก็บสินค้าทั้งหมดด้วยวิธี DE ซึ่งผลคำตอบของตัวอย่างปัญหาทั้งหมดจะถูกแสดงในหัวข้อถัดไป

3) ผลการทดสอบของวิธีดิฟเฟอเรนเชียลอิโวลูชัน กับปัญหาตัวอย่างของการจัดเก็บสินค้า

ประสิทธิภาพของการแก้ปัญหาโดยวิธีการดิฟเฟอเรนเชียลอิโวลูชันที่พัฒนาสำหรับงานวิจัยนี้ ได้ถูกนำมาทดสอบกับปัญหาตัวอย่างที่ได้สร้างขึ้นมาทั้ง 10 ตัวอย่าง ผลการหาค่าคำตอบทั้งหมดจะถูกแสดงไว้ในตารางที่ 4.22

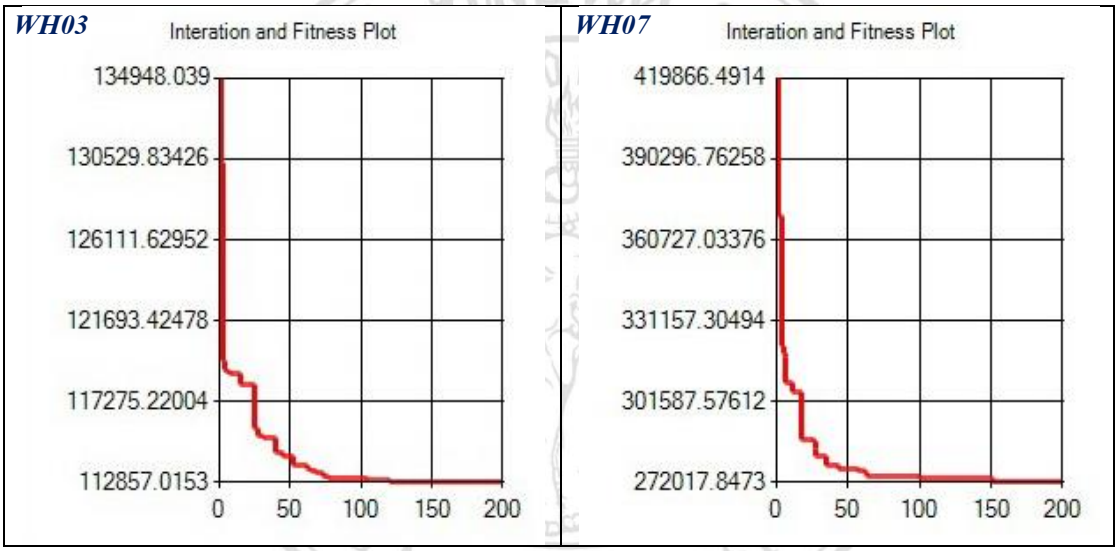
ตารางที่ 4.22 แสดงผลคำตอบของปัญหาตัวอย่างด้วยวิธีดิฟเฟอเรนเชียลอิโวลูชัน

ตัวอย่างปัญหา	ค่าของสมการวัตถุประสงค์					
	ผลครั้งที่ 1	ผลครั้งที่ 2	ผลครั้งที่ 3	ผลครั้งที่ 4	ผลครั้งที่ 5	ค่าเฉลี่ย
WH01	588.46	588.59	588.46	588.46	588.46	588.49
WH02	47296.24	47294.21	47364.64	47292.37	47290.02	47307.50
WH03	113010.79	113102.06	112867.02	112833.34	113330.33	113028.71
WH04	204525.56	204572.63	204724.86	204468.90	204705.63	204599.52
WH05	290955.22	293081.21	291359.54	290472.32	290854.26	291344.51
WH06	381887.74	381867.35	382290.17	381314.34	381728.21	381817.56
WH07	271910.05	272675.28	273454.65	272027.85	272361.79	272485.92
WH08	542427.04	541547.55	542171.83	541348.97	542557.70	542010.62

ตารางที่ 4.22 แสดงผลคำตอบของปัญหาตัวอย่างด้วยวิธีดิฟเฟอเรนเชียลอิโวลูชัน (ต่อ)

ตัวอย่างปัญหา	ค่าของสมการวัตถุประสงค์					
	ผลครั้งที่ 1	ผลครั้งที่ 2	ผลครั้งที่ 3	ผลครั้งที่ 4	ผลครั้งที่ 5	ค่าเฉลี่ย
WH09	261996.42	262414.59	263076.44	259507.28	259886.94	261376.33
WH10	746529.89	744872.81	744936.72	746233.06	744216.49	745357.79

ในส่วนของการแสดงตัวอย่างพฤติกรรมการลู่เข้าหาคำตอบของวิธีการ DE จะถูกนำเสนอด้วยการยกตัวอย่างปัญหา WH03, WH07, WH09 และ WH10 แสดงได้ดังรูปที่ 4.27 และ 4.28



รูปที่ 4.27 แสดงตัวอย่างพฤติกรรมการลู่เข้าหาคำตอบของวิธี DE ตัวอย่างปัญหา WH03 และ WH07

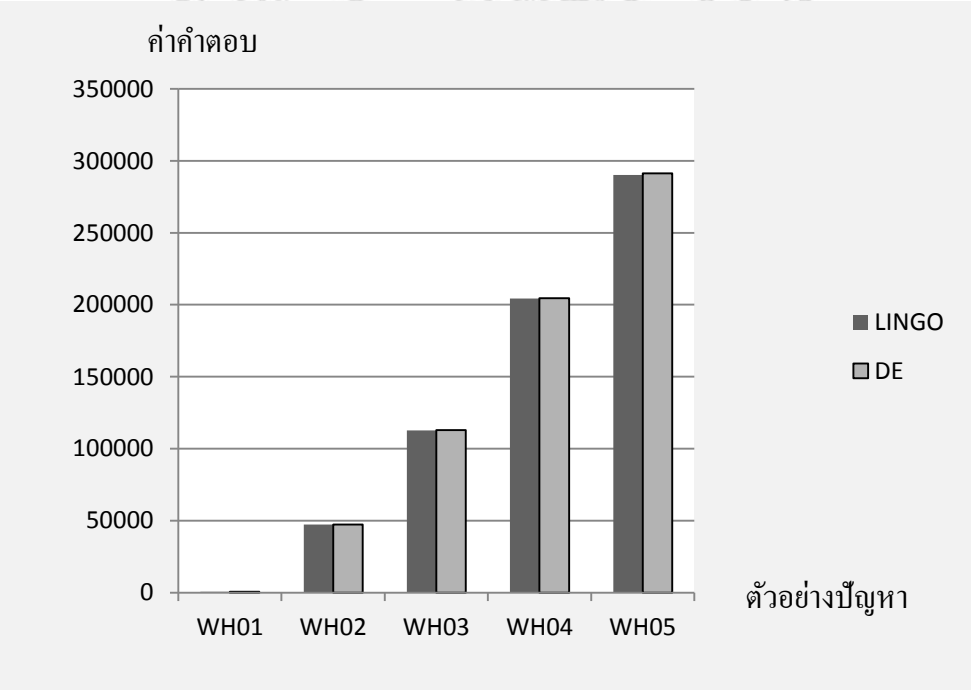
รูปที่ 4.28 แสดงตัวอย่างพฤติกรรมการลู่เข้าหาคำตอบของวิธี DE ตัวอย่างปัญหา WH09 และ WH10

4.5 การประเมินประสิทธิภาพของการหาคำตอบด้วยวิธีดิฟเฟอเรนเชียลอิโวลูชัน

การนำเสนอด้านคุณภาพของคำตอบจะทำการเปรียบเทียบด้วยตัวอย่างปัญหา WH01 ถึง WH05 การเปรียบเทียบคำตอบที่ได้และเวลาที่ใช้ในการคำนวณค่าคำตอบ ระหว่างวิธีตรงจากโปรแกรม LINGO และค่าคำตอบจากวิธี DE จะถูกแสดงผลด้วยตารางและรูปภาพดังต่อไปนี้

ตารางที่ 4.23 การเปรียบเทียบคำตอบที่ได้จากวิธีตรงและวิธีดิฟเฟอเรนเชียลอิโวลูชัน

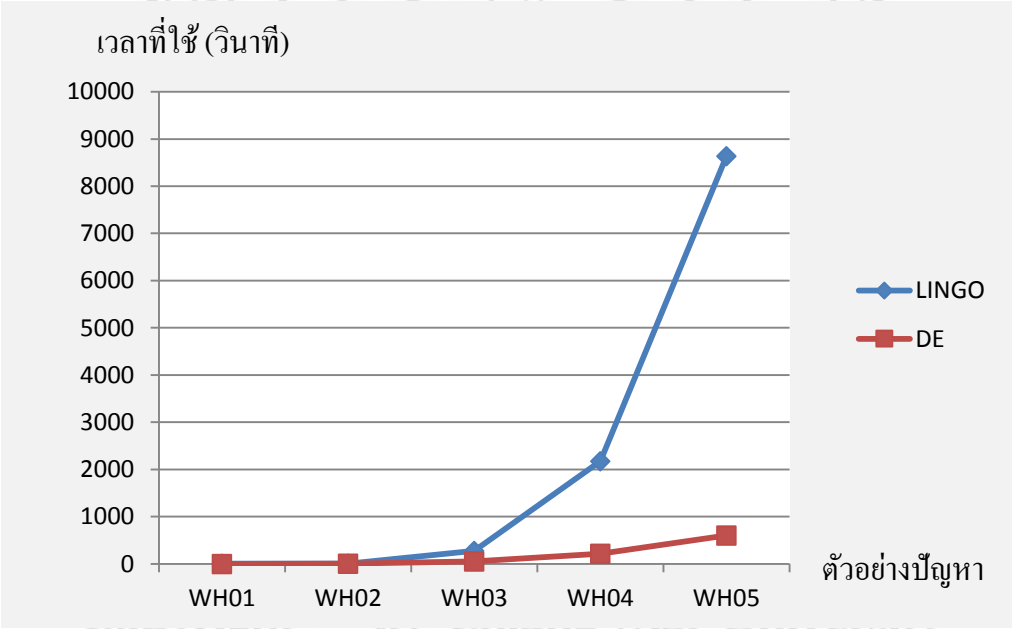
ตัวอย่างที่	ผลจาก LINGO (Global Optimum)	ผลจาก C# DE (ค่าเฉลี่ย)	ความแตกต่างของคำตอบ (%)
WH01	588.46	588.49	0.0042
WH02	47273.04	47307.50	0.0729
WH03	112705.20	113028.71	0.2870
WH04	204283.60	204599.52	0.1546
WH05	290062.90	291344.51	0.4418



รูปที่ 4.29 การเปรียบเทียบคำตอบที่ได้จากวิธีตรงและวิธีดิฟเฟอเรนเชียลอิโวลูชัน

ตารางที่ 4.24 การเปรียบเทียบเวลาที่ใช้หาคำตอบของวิธีตรงและวิธีดิฟเฟอเรนเชียลอิโวลูชัน

ตัวอย่างที่	ผลจาก LINGO เวลาที่ใช้ ชั่วโมง:นาที:วินาที	ผลจาก C# เวลาที่ใช้ ชั่วโมง:นาที:วินาที
WH01	0:00:08	0:00:01
WH02	0:00:07	0:00:05
WH03	0:04:40	0:00:54
WH04	0:36:13	0:03:33
WH05	2:24:00	0:10:06



รูปที่ 4.30 การเปรียบเทียบเวลาที่ใช้หาคำตอบของวิธีตรงและวิธีดิฟเฟอเรนเชียลอิโวลูชัน

จากการเปรียบเทียบผลด้านคำตอบและเวลาที่ใช้ในการคำนวณ ระหว่างค่าคำตอบจากโปรแกรม LINGO และค่าคำตอบจากวิธี DE ผู้วิจัยได้ทำการสรุปและวิเคราะห์ผล ซึ่งจะนำเสนอในบทที่ 5 การสรุปผลดำเนินงานวิจัย