# รายการอ้างอิง


**ภาษาไทย**

ศิริจันทร์ ทองประเสริฐ , 2537, การจำลองแบบปัญหา. กรุงเทพมหานคร: โรงพิมพ์จุฬาลงกรณ์มหาวิทยาลัย

สมศักดิ์ ตรีสัตย์, 2535. การออกแบบและวางผังโรงงาน. กรุงเทพมหานคร: สมาคมส่งเสริมเทคโนโลยีไทย-ญี่ปุ่น. ดวงกมลสมัย

**ภาษาอังกฤษ**

Askin, R.G., and Strandridge C.R. 1993. Modeling and Analysis of Manufacturing Systems. John Wiley & Sons.

Armour, G.C., and Buffa, E.S. 1963. A Heuristic Algorithm and Simulation Approach. A IIE Transactions. Vol.7 No.4: 432-437.

Bagchi, S. et.al 1991. Exploring Problem-Specific Recombination Operators for Job Shop Scheduling. International Conference on Genetic Algorithms. California: Morgan Kaufmann Publishers. 10-17.

Biegel, J.E., and Davern, J.J. 1990. Genetic Algorithms and Job Shop Scheduling. Computer Industrial Engineer. Vol.19 No.1-4: 81-89.

Bramlette, M.F. 1989. Initialization, Mutation and Selection Methods in Genetic Algorithms for Function Optimization. Proc. third Int. Conf. On Genetic Algorithms: George Mason University. 100-107

Cartwright, H.M., and Mott, G.F. 1989. Making Around: Using Clues from the Data Space to Guide Genetic Algorithm Searches. Proc. third Int. Conf. On Genetic Algorithms. George Mason University.108-114.

Croce, F.D., Tadei, R., and Guiuseppe, S. 1995. A Genetic Algorithm for the Job Shop Problem. Computer Operation Research. Vol.22 No.1: 15-24.

Chan, K.C, and Tansri, H. 1994. A Study of Genetic Crossover Operations on the Facility Layout Problems. Computers Industrial Engineering. Vol.26 No.3: 537-550.

Davis, L. 1989. Adapting Operator Probabilities In Genetic Algorithms. ICGA'89: 61-69.

Dagli, C., and Sittisathanchai, S. 1993. Genetic Neuro-Scheduling for Job Shop Scheduling. Computers and Industrial Engineering. Vol. 25 No.1-4: 267-290.

David, L. 1991. Handbook of Genetic Algorithms. New York: Van Nostrand Reinhold.

Falkenauer, E., and Bouffouix, S. 1991. A Genetic for Job Shop. Proceeding of the 1991 IEEE International Conference on Robotics and Automation: 824-829.

Fransis, R.L., McGinnis, L.F., and White, J. 1992. Facility Layout and Location: An Analytical approach. 2nd ed. Prentice Hall.

French, S. 1982. Sequencing And Scheduling: An Introduction to the Mathematic of the Job-Shop. John Wiley & Sons. 137-150.

Glover, F. 1995. Genetic Algorithms and Tabu Search: Hybrides for Optimization. Computer Operation Research. Vol.22 No.1: 111-134.

Goldberg, D.E. 1989. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison Wesley.

Holland, J.H. 1975. Adaptation in Natural and Artificial System. Ann Arbor. The University of Michigan Press.

Hytug et.al (1994), Genetic Learning of Dynamic Scheduling within A Simulation Environment, Computer Operation Research, Vol.21 No.8: 909-925.

John et.al 1995. A Real-World Problem Using Genetic Algorithms. Computer Industrial Engineer. Vol. 29 No. 1-4: 177-181.

Kim et al. 1996. Genetic Algorithms for Assembly Line Balancing with Various Objectives. Computers Industrial Engineering. Vol.30 No.3: 397-409.

Kubota, N. et.al 1996. Virus-Evolutionary Genetic Algorithm for a Self-Organizing Manufacturing System. Computer Industrial Engineer: Vol.30 No.4:1015-1026.

Kusiak, A. 1990. Intelligence Manufacturing System. Prentice-Hall. 289-319.

Lawrence, D. 1989. Adapting Operator Probability in Genetics Algorithms, Proc. International Conference on Genetic Algorithms and their Application.

Lee, C.Y, and Kim, S.J. 1995. Parallel Genetic Algorithms for the Earliness-Tardiness Job Scheduling Problem with General Penalty Weights. Computer Industrial Engineer. Vol. 28 No. 2: 231-241.

Levitin, V., and Rubinovitz J. 1993. Genetic Algorithm for Linear and Cyclic Assignment Problem Computer Operation Research. Vol.20 No.6:

Meller and Bozer 1996. A New Simulated Annealing Algorithm for the Facility Layout Problem. International Journal Production Research. Vol. 34 No. 6:1675-1692.

Michalewicz, Z., and Janikow, C.Z 1989. Handling Constraints in Genetic Algorithms. Proc. third Int. Conf. On Genetic Algorithms. George Mason University.

Michalewicz, Z. 1996. Genetic Algorithms + Data Structure = Evolution Programs 3rd rev and extended. New York: Springer Verlag Berlin Heidelberg.

Montgomery, C.D. 1996. Design and Analysis of Experiments. John Wiley & Sons.

Nakano, R., and Yamada, T. 1989. Conventional Genetic Algorithm Job Shop Problems. Proc. third Int.conf on Genetic Algorithms. Geourge Mason University.454-479.

Nugent, C.E., Vollman, R.E., and Ruml, J. 1968. An experimental comparison of techniques for the assignment of facilities to locations. Operation Research. Vol.16: 150-173.

Peng et.al 1996. Simulate Annealling for Quadratic Assignment Problem: A Further Study. Computer Industrial Engineer. Vol.31 No.3/4: 925-928.

Pereira, M.G. 1996 Artificial Intelligence – Techniques for Search Results in Programming Projects. Computer Industrial Engineer. Vol 31 No.1/2: 393-396.

Pierreval, H., and Tautou, L. 1997. Using evolution algorithms and simulation for the optimization of Manufacturing Systems. IIE Transaction: 181-189.

Poon, P.W., and Carter, J.N. 1995. Genetic Algorithm Crossover Operators for Ordering Application. Computer Operation Research. Vol.22 No.1: 135-147.

Reevest, C.R. 1995. A Genetic Algorithm for Flowshop Sequencing Computer Operation Research. Vol.22 No.1: 5-13.

Rubin, P.A., and Ragatz, G.L. 1995. Scheduling in a Sequence Dependent Setup Environment with Genetic Search. Computer Operation Research. Vol.22 No.1: 85-99.

Schaffer, J.D., and Eshelman, L.J. On Crossover as an Evolutionarily Viable Strategy. Proc. third Int. Conf. On Genetic Algorithms. George Mason University.

Sule, D.R. 1994. Manufacturing Facilities Location, Planning, and Design. PWS Publiching Company.

Suresh, G., Vinod V.V., and Sahu, S. 1995. A genetic algorithm for facility International Journal Production Research. Vol. 33 No.22: 3411-3423.

Tate, D.M. and Smith, A.E. 1995. A Genetic Approach to The Quadratic Assignment Problem. Computer Operation Research. Vol.22 No.1: 73-83.

Thangiah, S.R. 1991. GIDEON: A Genetic Algorithm System for Vehicle Routing Routing with Time Windows. IEEE Transactions on Systems. Man and Cybernatic: 322-328.

Thangiah, S.R., and Nygard, K.E. 1992. School Bus Routing Using Genetic Algorithms. SPIE Application of Artificial Intelligence X: Knowledge-Based System. Vol.1707: 387-398.

Venugopal, V., and Narendran, T.T. 1992. A Genetic Algorithms Approach to the Machine-Component Grouping Problem with Multiple Objective. Computer Industrial Engineer. Vol.22 No.4: 469-480.

Vignaux, G.A., and Michalewicz, Z. 1991. A Genetic Algorithm for the Linear Transportation Problem. IEEE Transactions on Systems. Man and Cybernatic. Vol. 21 No.2: 445-452.

Whitley, D. et.al 1989. Scheduling Problems and Traveling Salesman: The Genetic Edge Recombination Operator. Proc. third Int. Conf. On Genetic Algorithms:133-140.

Winston, P.H. 1992. Artificial Intelligence: Addison Westley. 505-528.

Wren, A. and Wren, D. 1995. A Genetic Algorithm for Public Transpor Driver Scheduling. Computer Operation Research. Vol.22 No.1:101-109.

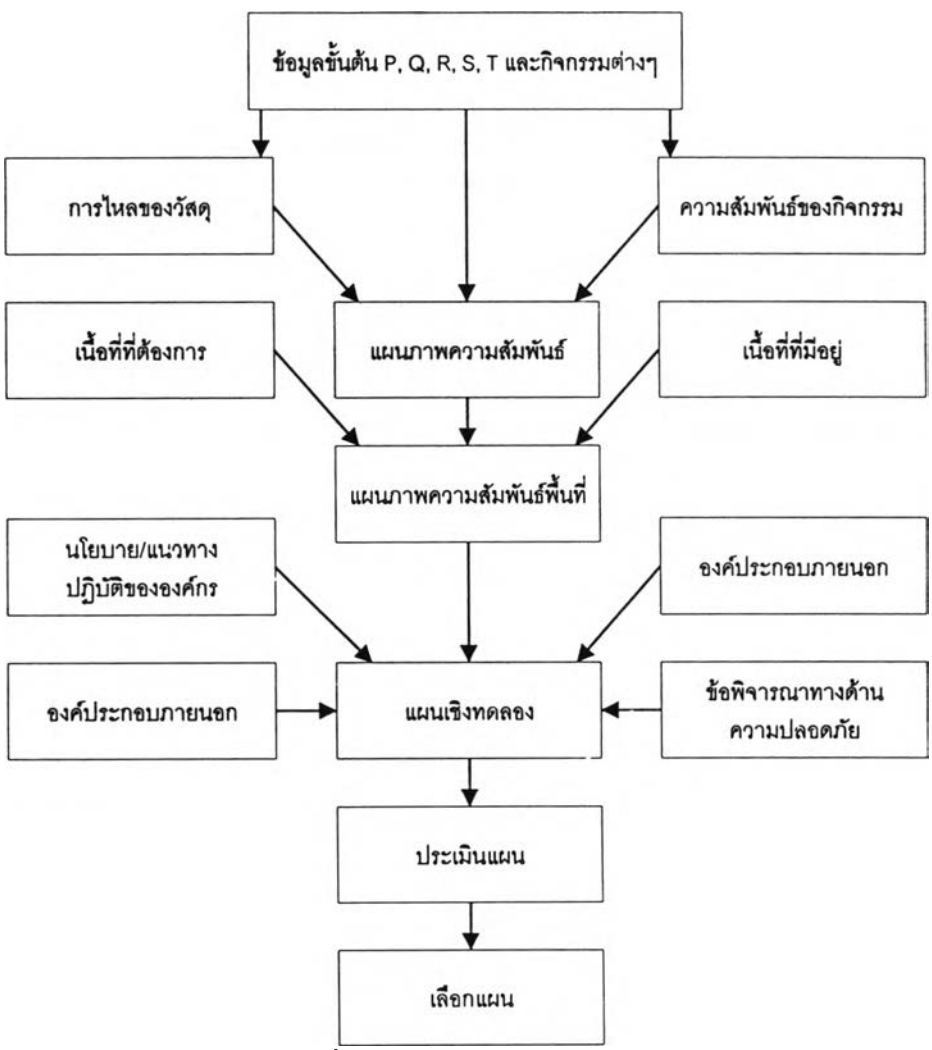
Zadeh, L. A. 1987. Fuzzy Set and Application: John Wiley & Sons.

Zadeh, L.A 1965 Fuzzy sets information and Control: 335-353.

# ภาคผนวก ก
# การออกแบบผังโรงงานอย่างมีระบบ

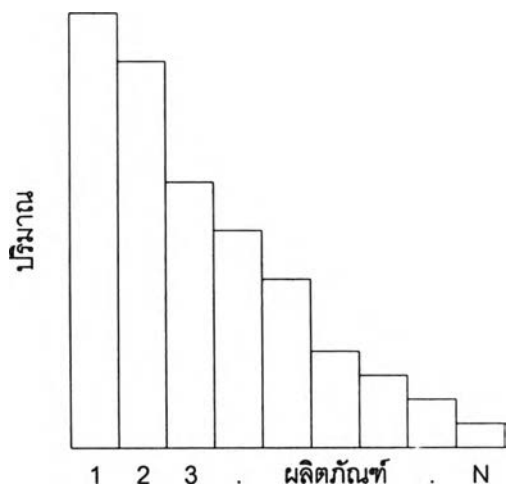
เนื้อหาในบทนี้กล่าวถึง วิธีการออกแบบผังโรงงานอย่างมีระบบ (Systematic Layout Planning) และขั้นตอนต่างๆอย่างละเอียดซึ่งจะนำไปสู่ที่มาของปัญหาการจัดผังโรงงาน

Muther (1973) ได้เสนอวิธีการออกแบบผังโรงงานอย่างมีระบบ หรือ SLP ในการออกแบบผังโรงงาน โดยมีแนวคิดพื้นฐานดังรูปที่ ก1 ซึ่งแสดงให้เห็นว่าผังโรงงานจะได้จากการวิเคราะห์รายละเอียดของกระบวนการตามวิธีการของ SLP โดยพิจารณาถึง ข้อมูล การตัดสินใจ วิธีการ ตามขั้นตอนต่างๆ ส่วนรายละเอียดจะกล่าวถึงต่อไป




**รูปที่ ก1** ขั้นตอนของการออกแบบผังโรงงานอย่างมีระบบ

**ขั้นตอนที่** 0 เก็บข้อมูล (Data Collection) ได้แก่ ข้อมูลของ ผลิตภัณฑ์ (Product) ปริมาณ (Quantity) กระบวนการผลิต (Routing) สิ่งสนับสนุนการผลิต (Supporting) และเวลา (Timing) ปริมาณและชนิดของผลิตภัณฑ์เป็นตัวกำหนดชนิดของผังโรงงาน (ผังโรงงานแบบผลิตภัณฑ์ ผังโรงงานแบบกระบวนการและผังโรงงานแบบที่ตั้งคงที่) ผลิตภัณฑ์ในโรงงานอาจมีหลายชนิดหรือชนิดเดียวก็ได้ ผลิตภัณฑ์เหล่านี้สามารถแสดงบนแผนภูมิผลิตภัณฑ์-ปริมาณ การวิเคราะห์ความสำคัญของผลิตภัณฑ์อาจวิเคราะห์ได้จากแผนภูมิพาเรโต (Pareto Diagram) ซึ่งสามารถแสดงได้ดังรูปที่ ก2



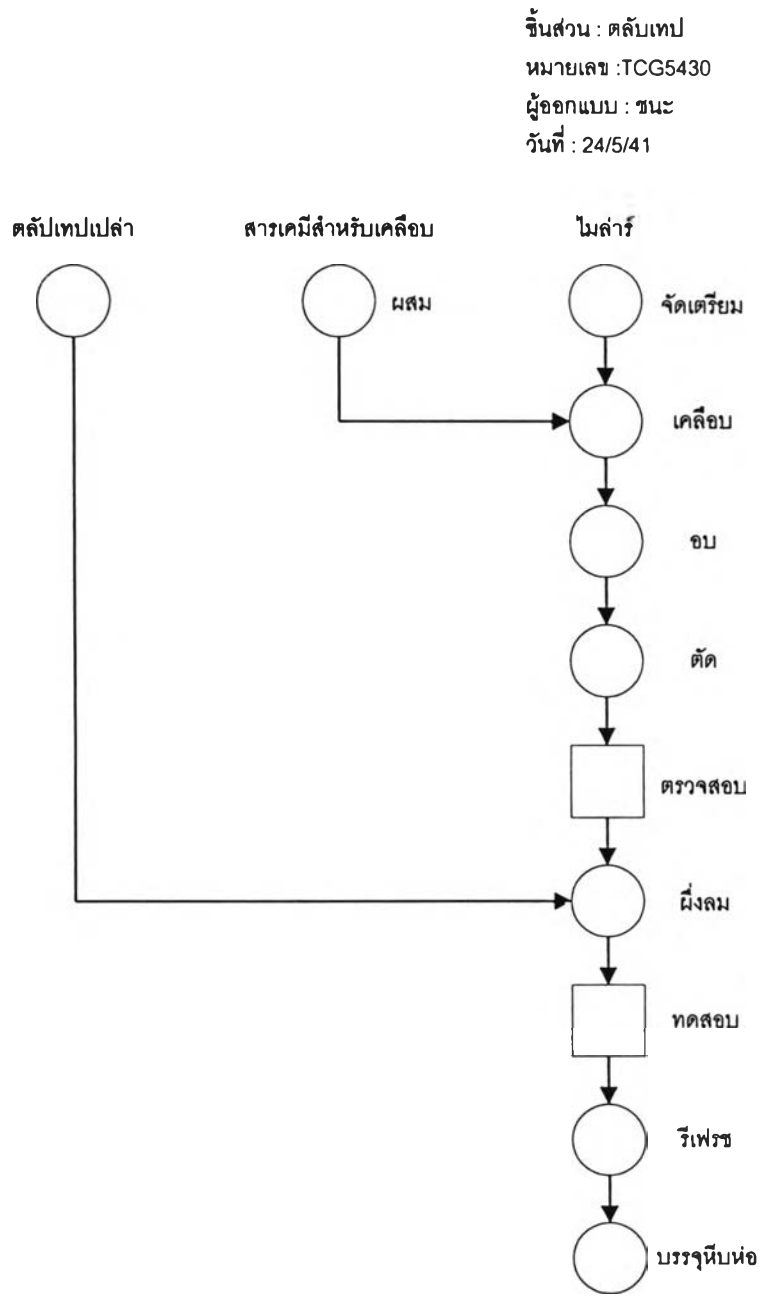

**รูปที่** ก2 แผนภูมิผลิตภัณฑ์-ปริมาณ

ข้อมูลของผลิตภัณฑ์ (P) ปริมาณ (Q) กระบวนการผลิต (R) สิ่งสนับสนุนการผลิต (S) และเวลา (T) จะถูกจัดเก็บอย่างละเอียดเช่น ข้อมูลของเครื่องจักรในสายงานผลิตได้แก่ข้อมูลเส้นทางการไหลของวัสดุและแผนการผลิตสามารถนำไปใช้ในการหาจำนวนของเครื่องจักรแต่ละชนิดและจำนวนพนักงานที่ต้องการได้ นอกจากนี้ข้อมูลที่จัดเก็บหลายๆด้านยังมีส่วนช่วยให้การวิเคราะห์ต่างๆมีความถูกต้องมากยิ่งขึ้น เช่น การหาพื้นที่ที่ต้องการใช้จากผู้ปฏิบัติงานอาจมีความต้องการใช้พื้นที่สูงแต่ถ้านำข้อมูลเหล่านี้มาทำการวิเคราะห์แล้วพื้นที่ที่ต้องการจริงๆอาจไม่สูงมากก็ได้

**ขั้นตอนที่** 1 วิเคราะห์การไหล (Flow Analysis) ก่อนการรวบรวมจัดทำข้อมูลข่าวสารผู้วิเคราะห์จะต้องกำหนดศูนย์กลางงาน (Work Center) ซึ่งมีความเกี่ยวข้องกับชนิดผังโรงงาน จากนั้นกำหนดการจัดพื้นที่ แล้วจึงทำการกำหนดสถานีงานอื่นรอบศูนย์กลางงานต่อไป แผนภูมิกระบวนการผลิต (Operation Chart) ดังรูปที่ ก3 ใช้ในการกำหนดการไหลของผลิตภัณฑ์ ข้อมูลการไหลของปริมาณสามารถสรุปได้เป็นแผนภูมิจาก-ไป (From-To Chart) แสดงดังตารางที่ ก1 ตารางนี้แสดงถึงปริมาณการขนถ่ายวัสดุจากสถานีงาน $i$ ไปยังสถานีงาน $j$ ปริมาณการไหลโดยรวม

ระหว่างสถานีงานสามารถตรวจสอบจากได้จากแผนภูมิ จาก-ไปและแผนภาพการไหล (Route Sheet) ของวัสดุแต่ละชนิดดังแสดงในรูปที่ ก4 ค่าใช้จ่ายโดยรวมของสถานีงานประกอบด้วยค่าใช้จ่ายในการขนถ่ายวัสดุกับระยะทาง กำหนดให้ระยะทางระหว่างแผนก $i$ และ $j$ เป็น $d_{ij}$ และค่าใช้จ่ายการเคลื่อนที่วัสดุของสถานีงาน $c_{ij}$ ดังสมการที่ 1 คือ

$$c_{ij} = w_{ij} d_{ij} \qquad (1)$$




**รูปที่ ก3** แผนภูมิการผลิตตลับเทป

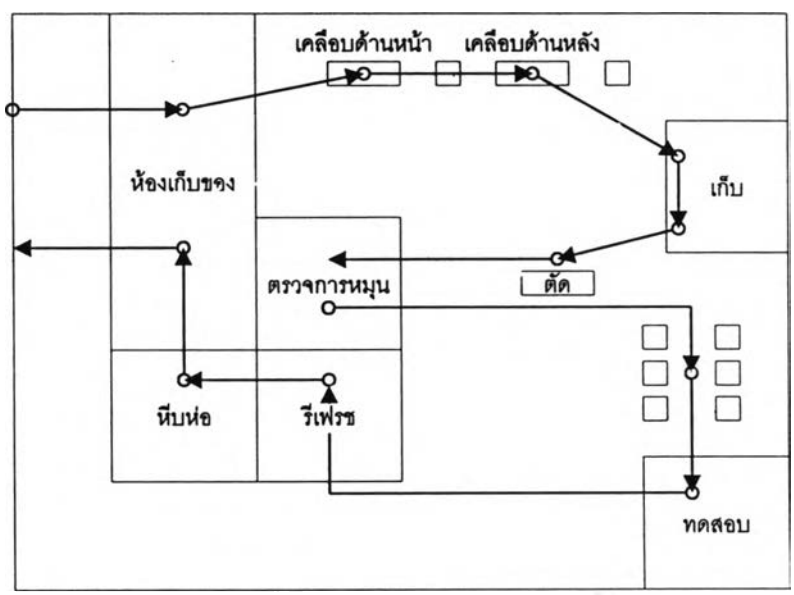
โดยที่ $w_{ij}$ คือค่าน้ำหนักของการไหลวัสดุระหว่างแผนกดังสมการที่ 2 คือ

$$w_{ij} = f_{ij} h_{ij} \qquad (2)$$

และ $f_{ij}$ คือ ปริมาณการไหลของวัสดุ (เที่ยว / เวลา) และ $h_{ij}$ คือ ค่าใช้จ่ายการขนถ่ายวัสดุ (ค่าใช้จ่าย/หน่วย ระยะทาง) อาจเป็นต้นทุนคงที่และต้นทุนผันแปรของการขนถ่ายวัสดุ ในเวลานั้นๆ หรืออีกนัยหนึ่งตัวประกอบตัวนี้อาจเป็นค่าถ่วงน้ำหนักที่รวมถึง ความปลอดภัย, ความสำคัญของลูกค้าและองค์ประกอบอื่นๆ ถ้าปริมาณการไหลทั้งหมดเกี่ยวเนื่องกัน ค่า $h_{ij}$ อาจมีค่าเป็น 1 ได้ (โดยทั่วไป $w_{ii}$ มีค่าเป็น 0)

ค่าใช้จ่ายโดยรวมคือ

$$C = \sum_{i=1}^{M} \sum_{j=1}^{M} c_{ij} \qquad (3)$$



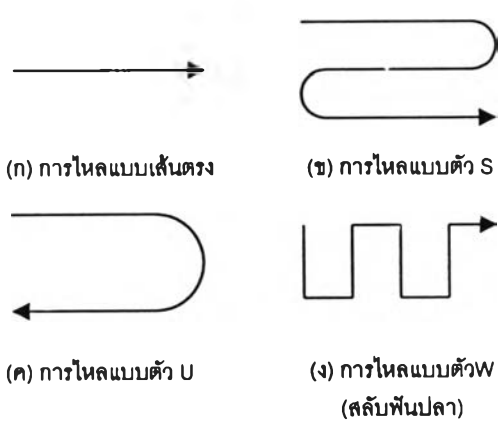

**รูปที่ n4** แผนภาพแสดงการไหลของกระบวนการผลิตตลับเทป

ดังนั้นการหาค่าใช้จ่ายอย่างคร่าวๆ จะประกอบไปด้วยข้อมูลสามส่วนได้แก่ แผนภูมิจาก-ไปของปริมาณการไหล แผนภูมิจาก-ไปของค่าใช้จ่ายในการเคลื่อนย้ายวัสดุและระยะทางระหว่างสถานีงาน สมการที่ 1 และ 3 ค่าใช้จ่ายในการขนถ่ายวัสดุ $c_{ij}$ คือบาทต่อช่วงระยะเวลา ปริมาณการไหลได้จากแผนการผลิตหรือข้อตกลงของแผนกต่างๆในโรงงาน ส่วนค่าใช้จ่ายของการขนถ่ายวัสดุขึ้นอยู่กับวิธีการและเครื่องมือที่ใช้ในการขนถ่ายวัสดุนั้นๆ ระยะทางขึ้นอยู่กับผังโรงงานที่ตัดสินใจเลือก การออกแบบผังโรงงานจึงสามารถทดลองเปลี่ยนแปลงค่าต่างๆได้ตามความเหมาะสม

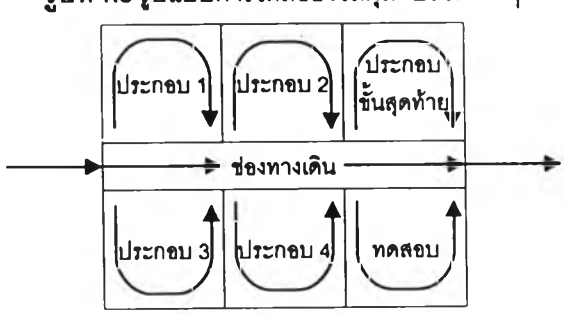
| จาก –ไป | SR | PC | PS | IC | XT | AT |
|---|---|---|---|---|---|---|
| SR | - | 40 | 10 | 30 | 10 | 50 |
| PC | | - | | | | 100 |
| PS | | | - | | | 102 |
| IC | | | | - | | 100 |
| XT | | | 100 | | - | |
| AT | 100 | 5 | 2 | 5 | 5 | - |

**ตารางที่ ก1** แผนภูมิจาก -ไป

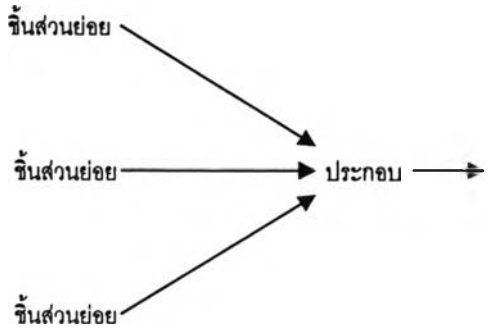
รูปแบบของการไหลมีหลายแบบ รูปที่ ก5 แสดงถึงรูปแบบการไหลที่เป็นไปได้ รูป ก5 ก แสดงรูปแบบการไหลแบบเส้นตรง รูป ก5 ข แสดงรูปแบบการไหลแบบตัวยู รูป ก5 ค แสดงรูปแบบการไหลแบบตัวเอส รูป ก5 ง แสดงรูปแบบการไหลแบบซิกแซ็ก รูปแบบการไหลที่นิยมอย่างหนึ่งคือ สพายน์ (Spine) ดังรูปที่ ก6 เป็นรูปแบบการไหลแบบผสมที่เป็นเส้นตรงแต่ถ้าเป็นการขนถ่ายภายในสถานีจะเป็นรูปตัวยู ส่วนแผนกรับ-ส่งสินค้าจะอยู่ด้านตรงข้ามซึ่งกันและกัน โรงงานทั่วไปมักเป็นประกอบชิ้นส่วนที่จะต้องส่งวัสดุไปยังไปยังแผนกประกอบสุดท้าย การไหลของวัสดุอาจเป็นชิ้นส่วนย่อยแล้วนำมาประกอบในขั้นสุดท้ายหรืออาจเป็นสายการผลิตที่มีชิ้นส่วนย่อยไหลเข้าตั้งฉากกับสายการผลิต ดังรูปที่ ก7 สิ่งที่สำคัญในการออกแบบผังโรงงานคือจะต้องหลีกเลี่ยงการออกแบบผังโรงงานที่ทำให้เกิดการไหลของวัสดุที่วกวนและสับสน



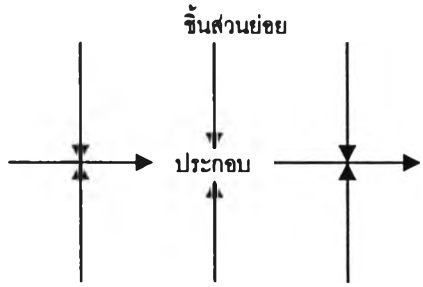

**รูปที่ ก5** รูปแบบการไหลของวัสดุลักษณะต่างๆ




**รูปที่ ก6** รูปแบบการไหลแบบสพายน์



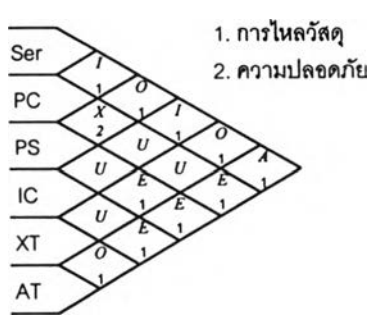

ก) การป้อนชิ้นส่วนแบบขนาน




ข) การป้อนชิ้นส่วนแบบตั้งฉาก

**รูปที่ ก7** รูปแบบการไหลของวัสดุเข้าสู่การประกอบ

**ขั้นตอนที่** 2 พิจารณาเชิงคุณภาพ (Qualitative Consideration) ในการระบุความสัมพันธ์ระหว่างสถานีงานสองสถานีสามารถกระทำได้ยากและไม่สามารถกำหนดได้แน่นอนตายตัว เช่นแผนกรับสินค้าและแผนกส่งสินค้าอาจจำเป็นต้องใช้อุปกรณ์ขนวัสดุร่วมกัน หรือการให้แผนกวิศวกรรมและแผนกจัดซื้ออยู่ติดกันเพื่อให้แน่ใจว่าผลิตภัณฑ์นั้นมีการตรวจสอบคุณภาพ มีการติดต่อสื่อสารที่สะดวก อย่างน้อยทำให้มีประสิทธิภาพมากกว่าการให้แผนกทั้งสองอยู่ด้านตรงข้ามกันในอาคาร ส่วนประกอบทางสภาวะแวดล้อมที่เกี่ยวข้องในกระบวนการก็จำเป็นที่จะต้องวิเคราะห์อย่างละเอียดเช่น แรงสั่นสะเทือน เสียง กลิ่น เป็นต้น ข้อมูลข่าวสารเหล่านี้จะถูกระบุอยู่ใน แผนภูมิแสดงความสัมพันธ์ (Relation Chart) หรือ เรียกสั้นๆว่า แผนภูมิ REL ดังตัวอย่างในรูป 8 สภาวะแวดล้อมพิเศษจะถูกระบุอยู่ในแผนภูมิด้วย เช่น แผนกสีมีวัสดุที่ติดไฟ (PS) ได้ง่ายจึงควรอยู่แยกกับแผนกพัสดุ (PC)




**รูปที่ ก8** แผนภาพ REL

แผนภูมิแสดงความสัมพันธ์จะประกอบด้วยสี่เหลี่ยมขนมเปียกปูนหลายรูป ลี่เหลี่ยมขนมเปียกปูนแต่ละรูปแสดงถึงความสัมพันธ์ของแต่ละแผนก ซึ่งมีค่าเป็น A E I O U และ X สัญลักษณ์เหล่านี้แสดงถึงระดับความต้องการความใกล้ชิดกันระหว่างสองแผนก โดยแบ่งเป็น 6 ระดับดังตารางที่ ก2

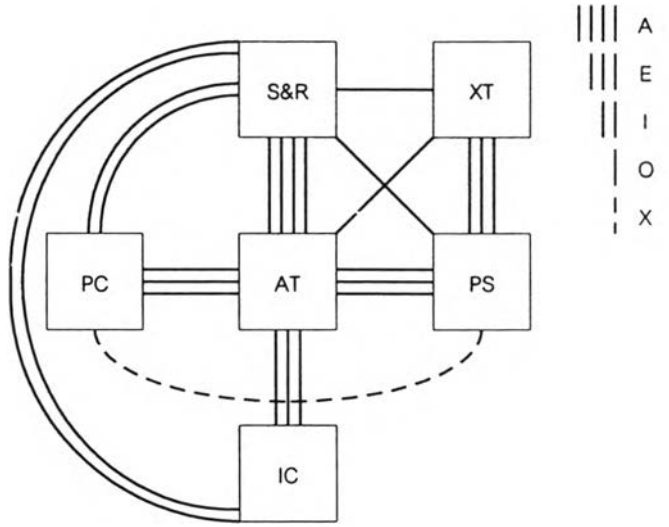
| ระดับความใกล้ชิด | | คะแนนเชิงคุณภาพแบบเชิงเส้น | คะแนนเชิงคุณภาพแบบเอ็กโปเนนเชียล |
|---|---|---|---|
| A | Absolutely Necessary | 4 | 81 |
| E | Especially Necessary | 3 | 27 |
| I | Important | 2 | 9 |
| O | Ordinary | 1 | 3 |
| U | Unimportant | 0 | 1 |
| X | Undesirable | -1 | -243 |

**ตารางที่ ก2 ระดับความสัมพันธ์ระหว่างแผนก**

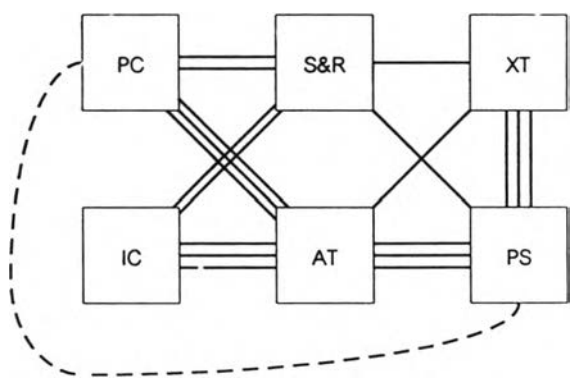
สัญลักษณ์เหล่านี้สามารถให้เป็นคะแนนเชิงคุณภาพเพื่อสะดวกในการคำนวณได้เป็นสองชนิดใหญ่คือ การให้คะแนนเชิงคุณภาพแบบเชิงเส้นและการให้คะแนนเชิงคุณภาพแบบเอ็กโปเนนเชียล ดังที่ตารางที่ ก2

ค่าเหล่านี้เป็นการให้คะแนนเชิงคุณภาพจึงไม่สามารถใช้ได้กับกระบวนการทางคณิตศาสตร์ได้ทุกกรณี เช่น ไม่อาจกล่าวได้ว่า A มีค่าเท่ากับ 2 I หรือ A ในด้านความปลอดภัยมีค่าเท่ากับ A ในด้านความสะดวกในการทำงาน แต่อย่างไรก็ดีมีการนำกระบวนการทางคณิตศาสตร์มาใช้ ควรระบุคำอธิบายแสดงความสัมพันธ์ในช่องสี่เหลี่ยมขนมเปียกปูนด้วย

**ขั้นตอนที่** 3 แผนภาพแสดงความสัมพันธ์ (Relation Diagram) แผนภาพแสดงความสัมพันธ์ (Relationship Diagram) แสดงถึงความสัมพันธ์ของข้อมูลเชิงปริมาณและข้อมูลเชิงคุณภาพ โดยมีกรอบรูปสี่เหลี่ยมแทนของสถานีงานแต่ละแผนก กรอบสี่เหลี่ยมเหล่านี้จะมีการจัดเรียงกันอย่างมีเหตุผลและจะถูกเชื่อมโยงกันด้วยเส้นต่างๆที่แสดงความสัมพันธ์ของแต่ละแผนก (A, E, I, O, U หรือ X) แผนภาพแสดงความสัมพันธ์อย่างง่ายเป็นดังรูป 9 กระบวนการจัดเรียงกรอบสี่เหลี่ยมเหล่านี้จะทำซ้ำหลายครั้งเพื่อให้ได้ผลลัพธ์ที่ดี นั่นคือ แผนกที่มีความสัมพันธ์เป็น A ควรจะอยู่ติดกัน




ก) แผนภาพแสดงความสัมพันธ์ของผังโรงงานเริ่มต้น




ข) แผนภาพแสดงความสัมพันธ์ของผังโรงงานเมื่อจัดเสร็จแล้ว

**รูปที่ ก9** แผนภาพแสดงความสัมพันธ์ของผังโรงงานก่อนการจัดและเมื่อจัดเสร็จแล้ว

ให้ $r_{ij}$ เป็นความสัมพันธ์ระหว่างสถานี $i$ และ $j$ โดยสามารถกำหนดค่า $r_{ij}$ ได้ตามความสัมพันธ์ และให้ $V(r_{ij})$ เป็นฟังก์ชั่น โดยฟังก์ชั่นนี้อาจมีค่าเป็น $V(A)$ = 81 $V(E)$ = 27 $V(I)$ = 9 $V(O)$ = 3 $V(U)$ = 1 และ $V(X)$ = -243 (ตารางที่ ก2) เมื่อคะแนนความสัมพันธ์เป็นแบบเอ็กซ์โปเนนเชียลการจัดเรียงแผนกในตอนเริ่มต้นจะพิจารณาถึง ค่าความใกล้ชิดโดยรวม (Total Closeness Rating) ดังสมการที่ 4

$$TCR_i = \sum_{j=1, j\neq i}^{M} V(r_{ij}) \qquad 4)$$

โดยที่ $M$ เป็นจำนวนสถานีงาน

สถานีงานใดที่มีค่า $TCR_i$ มากหมายความว่าสถานีนั้นมีความสัมพันธ์กับแผนกอื่นมากดังนั้นสถานีงานนั้นควรจะอยู่บริเวณกึ่งกลางของผังโรงงาน ขั้นตอนในการสร้างผังโรงงานขั้นตอนแรกคือ เรียงลำดับค่า $TCR_i$ จากมากไปหาน้อย จากนั้นเริ่มวางสถานีที่มีค่า $TCR_i$ มากที่สุดก่อนและจากนั้นผังสถานีอื่นๆต่อไปตามลำดับโดยรอบสถานีแรก ค่าความใกล้ชิด (Total Closeness Rating) ที่ใช้พิจารณามีสองอย่างคือ ค่าความใกล้ชิดสำหรับสถานีที่อยู่ติดกัน (Total Closeness Rating with Adjacent Department) ซึ่งจะต้องให้มีค่ามากที่สุด หรือค่าความใกล้ชิดที่พิจารณาระยะทางระหว่างแผนก (The Closeness Rating with Distance Between Department) ซึ่งจะต้องให้มีค่าน้อยที่สุด ดังสมการดังต่อไปนี้

$$\text{Maximize } V = \sum_{i=1}^{M-1} \sum_{j=i+1}^{M} \delta_{ij} V(r_{ij}) \qquad 5)$$

ค่า $\delta_{ij}$ เป็น 1 ถ้าแผนก $i$ และ $j$ อยู่ติดกัน และ ค่า $\delta_{ij}$ เป็น 0 ถ้าแผนก $i$ และ $j$ ไม่ได้อยู่ติดกัน

สมการค่าความใกล้ชิดที่พิจารณาระยะทางระหว่างแผนกคือ

$$\text{Minimize } Z = \sum_{i=1}^{M} \sum_{j=i+1}^{M-1} V(r_{ij}) d_{ij} \qquad 6)$$

การปรับเปลี่ยนการวางแต่ละแผนกต่างๆจะทำหลายครั้งเพื่อให้ได้คำตอบที่ดี โดยอาจพิจารณาค่าความใกล้ชิดหรือพิจารณาระยะทางก็ได้ มีวิธีการฮิวริสติกหลายอย่างที่ใช้ในการหาคำตอบประเภทนี้วิธีการที่นิยมอย่างหนึ่งได้แก่วิธีการ VNZ ฮิวริสติก และวิธี branch and bound ฯลฯ

**ขั้นตอนที่** 4 ความต้องการพื้นที่ (Space Requirement) ความต้องการพื้นที่สามารถหาได้หลายวิธี เช่นอาจใช้มาตรฐานทางอุตสาหกรรมในการกำหนดความต้องการพื้นที่ต่อหน่วยทรัพยากรสามารถนำไปใช้หาจำนวนพนักงานและเครื่องจักรในแผนกได้ วิธีการนี้จำเป็นต้องทำรายการอุปกรณ์ เครื่องจักร ชั้นวางของ ฯลฯ ที่จะต้องใช้พื้นที่ขึ้นมาก่อน เมื่อได้รายการอุปกรณ์เครื่องจักรแล้วจากนั้นจึงนำมาร่างผังโรงงานอย่างคร่าวๆแล้วจึงนำมาปรับเปลี่ยนให้เข้ากับมาตรฐาน พื้นที่การใช้งานเหล่านี้สามารถปรับเปลี่ยนได้ตามความเหมาะสม แต่สิ่งที่ผู้วิเคราะห์จะต้องคำนึงถึงคือจะต้องระมัดระวังการปรับเปลี่ยนมาตรฐานเพื่อให้เข้ากับสภาพที่เป็นอยู่และเทคโนโลยีที่เปลี่ยนแปลง

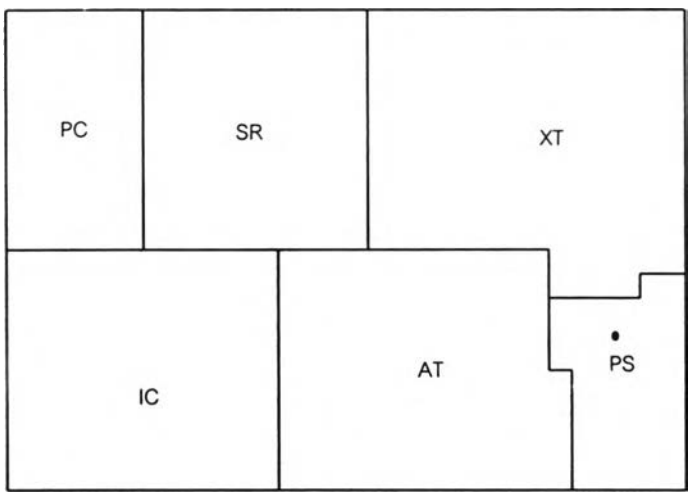
**ขั้นตอนที่ 5** การพิจารณาพื้นที่ที่มีอยู่ (Space Availability) หลังจากที่ได้ผังโรงงานอย่างเมื่อพิจารณาดูอาจพบว่าพื้นที่ที่มีอยู่จริงอาจน้อยกว่าพื้นที่ที่ต้องการก็ได้ ถ้ามีการเพิ่มแผนกขึ้นมาใหม่การออกแบบจำเป็นต้องเผื่อพื้นที่ของทรัพยากรด้วย เช่น วัตถุดิบ พลังงาน พนักงาน การไหลของวัสดุ เป็นต้น แต่อย่างไรก็ดีอาจถูกจำกัดทางด้านพื้นที่ที่มีอยู่จริงและงบประมาณก็ได้ ผู้ออกแบบผังโรงงานจึงจำเป็นต้องกำหนดและเตรียมการอธิบายเหตุผลที่ไม่สามารถจัดหาพื้นที่ตามที่กำหนดได้

**ขั้นตอนที่ 6** แผนภาพความสัมพันธ์พื้นที่ (Space Relationship Diagram) แผนภาพประเภทนี้แสดงความสัมพันธ์แสดงถึงผังโรงงานกับพื้นที่ที่มีอยู่จริง จากขั้นตอนที่แล้วการจัดผังโรงงานโดยสมมติให้แผนกต่างๆมีพื้นที่เท่ากันและจัดผังโรงงานโดยทำการสลับแผนกต่างๆเป็นคู่ๆเพื่อให้ได้คำตอบที่ดี แต่อย่างไรก็ตามในปัญหาการจัดผังโรงงานที่เกิดขึ้นจริงนั้นความต้องการพื้นที่และรูปร่างของแต่ละแผนกไม่เหมือนกัน แผนภาพความสัมพันธ์พื้นที่นี้จึงเปลี่ยนจากเดิมที่ให้แต่ละแผนกถูกแทนด้วยกรอบสี่เหลี่ยมซึ่งมีขนาดและรูปร่างเท่ากันมาเป็นให้แต่ละแผนกถูกแทนด้วยสี่เหลี่ยมเล็กๆที่มีขนาดเท่ากันหลายๆรูป (Grid) หรือกริดและนำสี่เหลี่ยมเล็กๆนี้มาประกอบเข้าด้วยกัน จะได้แผนกที่มีขนาดและรูปร่างซึ่งเป็นสัดส่วนตามความเป็นจริง และสี่เหลี่ยมเล็กๆเหล่านี้สามารถจัดเรียงกันใหม่เพื่อหาคำตอบของปัญหาจริงได้

การสลับแผนกในแผนภาพความสัมพันธ์พื้นที่ไม่ใช่สิ่งที่ง่าย แผนกต่างๆมีกริดเป็นสัดส่วนกับพื้นที่ของแผนก รูปที่ ก10 แสดงถึง กริดขนาด 20*30 ช่อง เป็นคำตอบเริ่มแรกของปัญหาตัวอย่างที่แล้ว โดยมีพื้นที่ของแต่ละแผนกจากตาราง 2

ในการสลับแผนกจะต้องทำการสลับกริดเพื่อรักษาขนาดของผังโรงงานและขนาดของแต่ละแผนกให้คงเดิม มีวิธีการและโปรแกรมหลายชนิดที่ช่วยในการสลับกริดแผนกที่อยู่ติดกันหรือแผนกที่มีขนาดเดียวกัน ส่วนแผนกมีขนาดเดียวกันอยู่แล้วการสลับจะทำได้ง่าย ในขณะที่แผนกที่อยู่ติดกันแต่มีขนาดต่างกันการสลับจะทำได้ยากกว่า วิธีการในการสลับแผนกที่มีขนาดต่างกันแต่อยู่ใกล้กันก็คือ เลือกกริดของแผนกที่มีขนาดใหญ่กว่าที่อยู่ไกลจุดเซ็นทรอยด์ของแผนกที่เล็กกว่ามากที่สุด จากนั้นเลือกกริดของแผนกที่ใหญ่กว่าให้เท่ากับจำนวนกริดของแผนกที่เล็กกว่าเพื่อทำการสลับกริดระหว่างสองแผนก สิ่งที่สำคัญคือจะต้องตรวจสอบว่าทั้งสองแผนกต้องเกาะกลุ่มกันไม่มีกริดใดกริดหนึ่งแตกแยกออกมา รูปที่ ก11 แสดงถึงการสลับแผนกที่มีขนาดไม่เท่ากัน ดังตารางที่ ก3 กริดที่มีเครื่องหมาย "*" คือกริดแรกของ PS ที่ย้ายมา ที่เลือกตำแหน่งนี้เนื่องจากอยู่ไกลจากจุดเซ็นทรอยด์ของ PS เก่ามากที่สุด จากนั้นจึงทำการเลือกกริดให้ครบ 45 กริด ได้เป็นเส้นขั้นบันได อย่างไรก็ดีรูปร่างของแผนกยังไม่เป็นที่ต้องการ จากนั้นจึงทดลองย้ายกริด 9 รูปเป็นรูปก้นหอยได้รูปร่างของแผนกใหม่เป็นรูปเส้นประ "...." ทางเลือกอีกทางหนึ่งที่เป็นเส้นประ "- - -" แสดง

ถึงลักษณะของผังโรงงานที่สามารถเพิ่มช่องทางเดินได้ เกณฑ์ในการตัดสินใจเลือกของผู้วิเคราะห์มีสองประการคือประการแรกจะต้องทำได้ง่าย ประการที่สองจะต้องเหมาะสมกับสภาพจริง ปัญหาอย่างหนึ่งที่เกิดขึ้นในการสลับตำแหน่งของแผนกหลายครั้งคือทำให้การคำนวณที่ผ่านมาลดความหมายลง

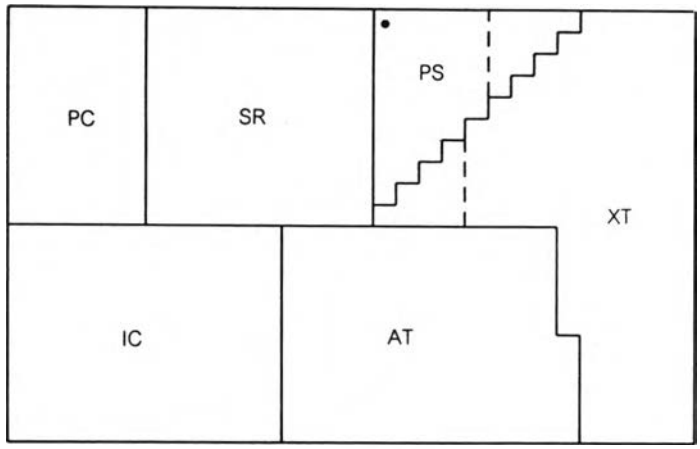



**รูปที่ ก10** แผนภาพแสดงความสัมพันธ์ของพื้นที่และผังโรงงานตามขนาดของกริด

**ตารางที่ ก3** แสดงขนาดพื้นที่ของปัญหาตัวอย่าง

| แผนก | พื้นที่ (ตารางฟุต) |
|---|---|
| SR | 10,000 |
| PC | 6,000 |
| PS | 45,00 |
| IC | 12,000 |
| XT | 15,000 |
| AT | 12,500 |
| รวม | 60,000 |

เมื่อสลับตำแหน่งแล้วจะทำให้จุดเซ็นทรอยด์ของแผนกอาจเปลี่ยนไปเนื่องจากรูปร่างของแต่ละแผนกเปลี่ยนแปลง เช่น แผนก PS ในรูปที่ ก11 มีจุดเซ็นทรอยด์เปลี่ยนไป เพื่อหลีกเลี่ยงการคำนวณหาจุดเซ็นทรอยด์ทุกๆครั้งที่ทำการสลับแผนก จำเป็นที่จะต้องหาจุดเซ็นทรอยด์ใหม่โดยการสลับจุดเซ็นทรอยด์ในผังโรงงานตั้งแต่เริ่มแรก ถ้าการสลับตำแหน่งแล้วได้ผลดีจะช่วยลดการคำนวณหาค่าจุดเซ็นทรอยด์และระยะทางของแต่ละแผนก




**รูปที่ ก11** ผังโรงงานหลังจากสลับกริดระหว่างแผนก PS และ XT

**ขั้นตอนที่ 7 และ 8 พิจารณาข้อกำหนดและข้อจำกัดต่างๆ** (Modifying Consideration and Limitation) **จากที่ผ่านมาผังโรงงานที่ได้ยังไม่มีรายละเอียดของกระบวนการและระบบที่ใช้ในผังโรงงาน** อาจจำเป็นต้องปรับผังโรงงานใหม่เพื่อให้เข้ากับสภาพข้อจำกัดทางด้านพื้นที่หรือข้อจำกัดทางด้านกระบวนการการผลิต เช่นระบบขนส่งทางน้ำ ทางบก อาจต้องอยู่ติดกับแผนกรับ-ส่งสินค้า พื้นภายในอาคารต้องสามารถรองรับแรงสั่นสะเทือนได้ สิ่งอำนวยความสะดวก เช่น ความร้อน การระบายอากาศ ระบบระบายอากาศ (HAVC) แสงสว่าง ระบบระบายของเสีย ระบบอัดอากาศ, ไฟฟ้าและระบบพลังงานต่างๆ ฯลฯ

การออกแบบบล็อกแพลนจะต้องพิจารณาถึงระบบขนถ่ายวัสดุ ช่องทางเดินควรเป็นเส้นตรงและอยู่ใกล้กับจุดรับส่งวัสดุโดยปราศจากสิ่งกีดขวางใดๆ

**ขั้นตอนที่** 9 การประเมินผล (Evaluation) **เมื่อถึงจุดนี้จะมีผังโรงงานให้เลือกหลายแบบ รูปที่แสดงถึงการไหลของวัสดุ** ตารางเปรียบเทียบข้อดีข้อเสียของผังโรงงาน ค่าใช้จ่ายของผังโรงงาน ปริมาณการไหลความถี่ ระยะทาง ค่าความใกล้ชิดของข้อมูลเชิงคุณภาพ องค์ประกอบเหล่านี้จะรวมถึง ความยืดหยุ่น (Flexibility) การบำรุงรักษา (Maintainability) ความสามารถในการเพิ่มเติมอุปกรณ์ตามความต้องการ (Modularity) ความปลอดภัย (Safety) และ ง่ายต่อการปฏิบัติงาน

# ภาคผนวก ข

## Quadratic Assignment Problems

เนื้อหาในภาคผนวกนี้กล่าวถึง รูปแบบของปัญหา QAP (Quadratic Assignment Problems) (Kusiak, 1990) และตัวแปรต่างที่เกี่ยวข้องรวมถึงรูปแบบสมการทางคณิตศาสตร์ที่เกี่ยวข้องกับปัญหานี้

Koopmans และ Beckmann (1957) เสนอแบบจำลองปัญหาการจัดผังโรงงานในรูปของการไหลของวัสดุของแผนกต่างๆ ซึ่งแบบจำลองนี้นำไปสู่รูปแบบปัญหา QAP โดยกำหนดให้

$n$ = จำนวนของแผนกต่างๆหรือจำนวนพื้นที่

$a_{ij}$ = รายได้จากการดำเนินการของแผนก $i$ ที่ตำแหน่ง $j$

$f_{ik}$ = การไหลของวัสดุจากแผนก $i$ ไปแผนก $k$

$c_{jl}$ = ค่าขนส่งหน่วยวัสดุ (unit of material) จากตำแหน่ง $j$ ไปตำแหน่ง $l$

$x_{ij}$ = 1 (ถ้าแผนก $i$ อยู่ที่ตำแหน่ง $j$), 0 (ถ้าแผนก $i$ ไม่อยู่ที่ตำแหน่ง $j$)

โดยมีข้อสมมติเพิ่มเติมคือ

$a_{ij}$ เป็นรายได้โดยรวม (Total Revenue) หักค่าลงทุนเริ่มแรกแต่ไม่รวมถึงค่าขนส่งวัสดุระหว่างแผนก

$f_{ik}$ ไม่ขึ้นอยู่กับตำแหน่งของแผนกต่างๆ

$c_{jl}$ ไม่ขึ้นอยู่กับแผนกต่างๆและค่าขนส่งโดยตรงจากแผนก $i$ ไปยังแผนก $k$ และถูกกว่าที่จะขนส่งผ่านแผนกที่ 3 ก่อน

จากตัวแปรที่ได้กล่าวมาแล้ว สามารถเขียนอยู่ในรูปสมการเป็น

$$\max \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}x_{ij} - \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} f_{ik}c_{jl}x_{ij}x_{kl} \qquad (1)$$

โดยที่

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1,\ldots,n \qquad (2)$$

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad j = 1,\ldots,n \qquad (3)$$

$$x_{ij} = 0 \text{ หรือ } 1 \qquad i = 1,\ldots,n \,,\, j = 1,\ldots,n \qquad (4)$$

ถ้า $a_{ij}$ เป็นค่าใช้จ่ายของการสร้างและดำเนินการของแผนก $i$ ตำแหน่ง $j$ แทนที่จะเป็นเป็นรายได้โดยรวมของแผนก $i$ ตำแหน่ง $j$ สมการที่ (1) อาจเขียนได้ใหม่เป็นดังสมการที่ (5)

$$\min \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}x_{ij} + \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} f_{ik}c_{jl}x_{ij}x_{kl} \qquad (1a)$$

Lawer (1963) ได้พิจารณาถึงค่าพารามิเตอร์ $b_{ijkl}$ โดยที่

$a_{ij}$ = ต้นทุนคงที่ของแผนก $i$ ตำแหน่ง $j$

$f_{ik}$ = การไหลวัสดุระหว่างแผนก $i$ ไปยังแผนก $k$

$c_{jl}$ = ต้นทุนการไหลวัสดุต่อหน่วยจากตำแหน่ง $j$ ไปยังตำแหน่ง $l$

และ

$$b_{ijkl} = f_{ik}c_{jl} + a_{ij} \quad \text{ถ้า } i = k \text{ และ } j = l$$

$$= f_{ik}c_{jl} \quad \text{ถ้า } i \neq k \text{ และ } j \neq l$$

เมื่อแทนค่า $b_{ijkl}$ ลงในสมการ 1a) ได้ว่า

$$\min \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} b_{ijkl}x_{ij}x_{kl} \qquad (1b)$$

จากสมการที่ผ่านมาสามารถกล่าวได้ว่าถ้า $i \neq k$ หมายความว่า $j \neq l$ ถ้า $j \neq l$ หมายความว่า $l \neq k$ ถ้า $i = k$ หมายความว่า $j = l$ ถ้า $j = l$ หมายความว่า $l = k$ เนื่องจากสมการที่ (2) และ (3) ดังนั้นจำนวนแผนกต่างๆจึงกำหนดให้เท่ากับจำนวนตำแหน่งที่ตั้ง หรือในบางปัญหาจำนวนแผนกอาจน้อยกว่าจำนวนที่ตั้ง (Steinberg, 1961) โดยให้บางแผนกเป็นสถานีหุ่น (Dummy) และกำหนดให้มีปริมาณการไหลเป็นศูนย์

ถ้า $a_{ij}$ มีค่าเป็นศูนย์หรือเหมือนกัน (identical) สมการที่ (1a) สามารถลดรูปได้เป็น

$$\min \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} f_{ij}c_{jl}x_{ij}x_{kl} \qquad (1c)$$

เนื่องจากสมการสามารถแสดงได้หลายรูปแบบ และโดยส่วนมากแล้วสมการที่ (1c) และสมการบังคับที่ (2) - (4) ถูกเรียกว่า Quadratic Assignment Problem

ปัญหา QAP กับสมการ (1a) และสมการบังคับที่ (2) - (4) ได้นำมาใช้ร่วมกับแบบจำลองของปัญหาการจัดผังโรงงาน ((Bazarra (1975) และ Burgard และ Stratmann (1978)) แต่ก็ไม่ได้หมายความว่าปัญหาการจัดผังโรงงานทั้งหมดจะอยู่ในรูปของ QAP ยกตัวอย่างเช่นการจัดวางตำแหน่งของเครื่องจักรในโรงงานโดยที่ไม่ทราบตำแหน่งที่ตั้งของเครื่องจักร ปัญหานี้ไม่สามารถที่จะหาคำตอบได้เนื่องจากไม่ทราบระยะทางที่แน่นอน และระยะทางของตำแหน่ง $j$ กับ $l$ จะมีความสัมพันธ์กับเครื่องจักรอื่นๆด้วย

ในบางสถานการณ์ปัญหาการจัดผังโรงงานก็อาจพบกับปัญหาที่ขนาดพื้นที่ของแต่ละแผนกไม่เท่ากัน ถ้าเป็นเช่นนี้แล้วการสลับตำแหน่งเพื่อทำการปรับปรุงผังโรงงานก็จะทำได้ยาก ผังโรงงานที่มีขนาดพื้นที่ของแต่ละแผนกไม่เท่ากันสามารถเขียนเป็นสมการได้ดังสมการที่ (5)

$$\min \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} f_{ij} c_{jl}^{K} x_{ij} x_{kl} \qquad (5)$$

โดยที่

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1,\ldots,n \qquad (6)$$

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad j = 1,\ldots,n \qquad (7)$$

$$x_{ij} = 0 \text{ หรือ } 1 \qquad i = 1,\ldots,n \,, j = 1,\ldots,n \qquad (8)$$

โดยที่ $c_{jl}^{K}$ คือค่าขนส่งของหน่วยวัสดุจากตำแหน่ง $j$ ไปยังตำแหน่ง $l$ ภายใต้การจัดเรียง $K$ และสมการบังคับที่ (6) – (8) เป็นไปเช่นเดียวกับ (2) – (4)

ให้ $K$ เป็นวิธีการจัดเรียงทั้งหมดที่เป็นไปได้ และขึ้นอยู่กับขนาดพื้นที่ของแต่ละแผนกซึ่งไม่จำเป็นต้องเท่ากับ $n!$ เนื่องจากว่าผังโรงงานบางผังอาจมีขนาดเท่ากันซึ่งอาจไม่ต้องทำการคำนวณทั้งหมด

# ภาคผนวก ค
# ปัญหา NP-hard

ปัญหา NP-hard คือปัญหาที่ใช้เวลาในการหาคำตอบยาวนานและเวลาในการหาคำตอบจะเพิ่มมากขึ้นเป็นแบบเอ็กซ์โปเนนเชียลเมื่อขนาดของปัญหาเพิ่มขึ้น ซึ่งไม่เหมาะกับการหาคำตอบด้วยวิธีการแบบตรงไปตรงมาในทางปฏิบัติ และโดยทั่วไปแล้วจะใช้ฮิวริสติกในการแก้ปัญหาประเภทนี้เพื่อให้ได้คำตอบที่ดีถึงแม้ว่าจะไม่ใช่คำตอบที่ดีที่สุดก็ตาม

ลักษณะของปัญหาแบบ NP-hard จะอยู่ในรูปของ $f(v)$ (Time Complexity Function) ซึ่งเป็นฟังก์ชั่นที่ใช้แสดงถึงเวลาสูงสุดของปัญหาที่มีขนาด v ตัวอย่างของเวลาในการคำนวณแสดงได้ดังตารางที่ ค1 เช่น เวลาที่ใช้ในการคำนวณของรูปแบบปัญหาที่มีฟังก์ชั่น $f(v)=v$ โดยกำหนดให้ v ขนาดเท่ากับ 10 และกำหนดให้เวลาที่ใช้ในการคำนวณในแต่ละขั้นตอนเท่ากับ 1 ไมโครวินาทีดังนั้นเวลาทั้งหมดที่ใช้ในการคำนวณทั้งหมดเท่ากับ 10 ไมโครวินาที (1x10) แต่ถ้าปัญหามีขนาดใหญ่ขึ้น เวลาที่ใช้ก็จะเพิ่มมากขึ้นเป็นแบบเส้นตรง แต่ถ้าปัญหาที่มีค่าของ $f(v)$ เป็น $2^v$ $3^v$ และ $v!$ เวลาที่ใช้จะเป็นแบบเอ็กซ์โปเนนเชียล

**ตารางที่ ค1** เวลาในการคำนวณที่อยู่ในรูป Time Complexity Function โดยมีสมมติฐานว่าการคำนวณในแต่ละครั้งใช้เวลา 1 ไมโครวินาที

| Time Complexity Function | v | | | | | |
|---|---|---|---|---|---|---|
| $f(v)$ | 10 | 20 | 30 | 40 | 50 | 60 |
| $v$ | 0.00001 sec | 0.00002 sec | 0.00003 sec | 0.00004 sec | 0.00005 sec | 0.00006 sec |
| $v^2$ | 0.001 sec | 0.0004 sec | 0.0009 sec | 0.0016 sec | 0.0025 sec | 0.0036 sec |
| $v^5$ | 0.1 sec | 3.2 sec | 24.3 sec | 1.7 min | 5.2 min | 13 min |
| $v^{10}$ | 2.7 hr | 118.5 days | 18.7 yrs | 3.3 centuries | 30.9 centuries | 192 centuries |
| $2^v$ | 0.001 sec | 1.0 sec | 17.9 min | 12.7 days | 35.7 yrs | 366 centuries |
| $3^v$ | 0.59 sec | 58 min | 6.5 yrs | 3855 centuries | $2*10^6$ centuries | $1.3*10^{13}$centuries |
| $v!$ | 3.6 sec | 770 centuries | $8.4*10^{16}$ yrs | $2.5*10^{32}$centuries | $9.6*10^{48}$centuries | $2.6*19^{66}$ centuries |

สมมติให้มีเครื่องคอมพิวเตอร์ที่มีความเร็วสูงกว่าเครื่องคอมพิวเตอร์จากตัวอย่างที่ผ่านมา 1,000 เท่า ถ้าปัญหาไม่มีความซับซ้อนมากนักและให้ระยะเวลาในการคำนวณเท่ากับเครื่องคอมพิวเตอร์จากปัญหาที่ผ่านมา ถ้าปัญหาที่มีฟังก์ชัน v ก็สามารถทำให้เวลาในการคำนวณเร็วขึ้น 1,000 เท่า แต่ถ้าปัญหามีความซับซ้อนมากคอมพิวเตอร์ที่มีความเร็วสูงก็สามารถช่วยในการคำนวณได้เร็วขึ้นในระดับหนึ่ง เช่นปัญหาที่มีฟังก์ชั่นเป็น $v!$ เครื่องคอมพิวเตอร์ที่มีการคำนวณเร็วกว่า 1,000 เท่า ช่วยให้การคำนวณได้เร็วขึ้นเล็กน้อย ดังตัวอย่างในตารางที่ ค2

**ตารางที่ ค2** ขนาดของปัญหาในการคำนวณของคอมพิวเตอร์ที่มีความเร็วสูงกว่า 1000 เท่า

| Time Complexity Function | ขนาดของปัญหาที่ถูกแก้ | |
|---|---|---|
| | คอมพิวเตอร์ธรรมดา | คอมพิวเตอร์ที่มีความเร็วสูงกว่า 1000 เท่า |
| $n$ | $N_1$ | $1000N_1$ |
| $n^2$ | $N_2$ | $31.62N_2$ |
| $n^3$ | $N_3$ | $3.98N_3$ |
| $n^{10}$ | $N_4$ | $1.99N_4$ |
| $2^n$ | $N_5$ | $N_5+10$ |
| $3^n$ | $N_6$ | $N_6+6$ |
| $n!$ | $N_7$ | $N_7+3$ $N_7 \leq 10$<br>$N_7+2$ $10 < N_7 \leq 30$<br>$N_7+1$ $30 < N_7 \leq 1000$ |

ปัญหา NP-hard เป็นปัญหาที่ใช้ระยะเวลาในการหาคำตอบยาวนาน ดังนั้นการหาคำตอบด้วยวิธีการแบบตรงไปตรงมาจึงเป็นไปได้ลำบาก และถึงแม้จะมีเครื่องคอมพิวเตอร์ที่มีความเร็วสูงมาช่วยในการคำนวณก็สามารถช่วยได้ในระดับหนึ่ง วิธีการหาคำตอบของปัญหารูปแบบนี้ได้แก่การใช้ฮิวริสติก หรือ อัลกอลิทึมต่างๆมาช่วยใช้ในการหาคำตอบ

# ภาคผนวก ง
# การจำลองแบบปัญหา

เนื้อหาในบทนี้จะกล่าวถึงวิธีการหาคำตอบของปัญหาการจัดผังโรงงาน ข้อดีของการใช้คอมพิวเตอร์ช่วยในการสร้างแบบจำลองของการจัดผังโรงงาน ขั้นตอนของการจำลองแบบปัญหา การสร้างแบบจำลอง การจัดเตรียมข้อมูล การทดสอบความถูกต้องของแบบจำลอง ตลอดจนการออกแบบการทดลองที่ใช้ในงานวิจัย ดังมีรายละเอียดดังต่อไปนี้

## ง.1 วิธีการหาคำตอบของปัญหาการจัดผังโรงงาน

โดยทั่วไปแล้ววิธีการหาคำตอบของปัญหาการจัดผังโรงหลายวิธีงานสามารถทำได้วิธี (Sule, 1994) ดังต่อไปนี้

1 Exact Mathematical Programming Procedure คือวิธีการหาคำตอบด้วยโปรแกรมทางการคำนวณทางคณิตศาสตร์โดยหาคำตอบที่เป็นไปได้ทุกวิธี โดยส่วนใหญ่แล้วจะเลือกใช้วิธี Branch and Bound (Askin และ Strandridge, 1993) วิธีการนี้สามารถหาคำตอบได้ดี แต่ไม่เหมาะกับปัญหาขนาดใหญ่ (จำนวนแผนก>15) เนื่องจากใช้ทรัพยากรสูงและใช้เวลาในการคำนวณนานไม่เหมาะสมที่จะนำมาใช้หาคำตอบในทางปฏิบัติ

2 Heuristics คือเทคนิคการแก้ปัญหาเฉพาะที่ให้คำตอบที่สามารถยอมรับได้ เนื่องจากการหาคำตอบของวิธีการที่กล่าวมาแล้วนั้นไม่เหมาะที่จะหาคำตอบของปัญหาขนาดใหญ่ วิธีการทางฮิวริสติก ได้แก่วิธีการ ของ CRAFT หรือ SDPI คอนสตรักชั่นฮิวริสติก วิธีการเหล่านี้เป็นที่นิยมเป็นอย่างมากแต่มีข้อเสียคือคำตอบที่ได้อาจไม่ใช่คำตอบที่ดีที่สุด

3 Probilistic Approach เป็นวิธีที่คล้ายกับวิธีการทั้งสองวิธีที่ผ่านมา แต่แตกต่างกันที่มีการใช้ความน่าจะเป็นมาช่วยในการหาคำตอบเพื่อทำการปรับปรุงผังโรงงานให้มีค่าใช้จ่ายถูกลง โดยการใช้ แบบจำลอง (Simulation) โดยจะต้องต้องสร้างผังโรงงานเบื้องต้นขึ้น และใช้โปรแกรมทำการหาคำตอบอื่นๆที่เป็นไปได้หลายๆครั้งเพื่อให้ได้คำตอบที่ดี การหาคำตอบในแต่ละครั้งก็จะทำให้ได้คำตอบที่ดีขึ้นไปเรื่อยๆ

4 Graph Theory เป็นการประยุกต์ใช้ทฤษฎีกราฟนำมาแก้ปัญหาการจัดผังโรงงาน (Askin และ Strandridge, 1993) แต่อย่างไรก็ตามวิธีการแบบนี้ก็ไม่เหมาะกับปัญหาขนาดใหญ่ซับซ้อน

จะเห็นได้ว่า การหาคำตอบของปัญหาการจัดผังโรงงานสามารถทำได้หลายวิธี วิธีที่ใช้ในงานวิจัยนี้ จะใช้ทั้งวิธี ฮิวริสติก Probilistic Approach โดยใช้คอมพิวเตอร์ช่วยในการคำนวณ

**ง.2 ข้อดีของการใช้คอมพิวเตอร์ช่วยในการสร้างแบบจำลองของการจัดผังโรงงาน**

การใช้คอมพิวเตอร์ช่วยในการคำนวณหาคำตอบมีข้อดีดังต่อไปนี้ (Sule, 1994)

1 คอมพิวเตอร์สามารถคำนวณหาคำตอบได้เร็วกว่ามนุษย์มากและมีค่าใช้จ่ายที่ถูกกว่า

2 คอมพิวเตอร์สามารถหาคำตอบของปัญหาขนาดใหญ่ซึ่งมีข้อมูลเป็นจำนวนมาก

3 คอมพิวเตอร์สามารถจัดผังโรงงานได้อย่างมีระบบ ตามสมการทางคณิตศาสตร์ที่สร้างขึ้น

**ง.3 การจำลองแบบปัญหา**

ในงานวิจัยได้ทดลองจัดผังโรงงานด้วยวิธีคอนสตรัคชั่นฮิวริสติก SDPI และ GAs ดังนั้นจึงต้องมีการศึกษาและวิเคราะห์การจัดผังโรงงานแต่ละวิธี อีกทั้งยังต้องเปรียบเทียบการจัดผังโรงงานแต่ละวิธี ในการศึกษาวิเคราะห์ และเปรียบเทียบดังกล่าวสามารถกระทำได้โดยการจำลองแบบปัญหา ซึ่งมีรายละเอียดดังต่อไปนี้

การจำลองแบบปัญหา คือกระบวนการในการออกแบบตัวแทนหรือแบบจำลองของระบบจริงและดำเนินการทดลองนั้นโดยมีจุดประสงค์เพื่อทำการศึกษาและทำความเข้าใจของระบบ หรือประเมินค่ากลยุทธ์ที่จะใช้ในทางปฏิบัติของระบบจริง (Pegden และคณะ,1995) ดังนั้นการจำลองแบบปัญหาในที่นี้จึงหมายความถึงการสร้างแบบจำลองปัญหาไปจนถึงการดำเนินการทดลองเพื่อศึกษาแบบจำลองนั้น

งานขั้นตอนต่างๆของการสร้างแบบจำลองปัญหาประกอบด้วย

1. การตั้งปัญหาและการให้คำจำกัดความของระบบงาน
2. การสร้างแบบจำลอง
3. การจัดเตรียมข้อมูล
4. การแปรรูปแบบจำลอง
5. การทดสอบความถูกต้อง
6. การออกแบบการทดลอง
7. การวางแผนการใช้การทดลอง
8. การดำเนินการทดลอง
9. การตีความผลการทดลอง
10. การนำไปใช้งาน
11. การจัดทำเอกสารการใช้งาน

### ง.3.1 การตั้งปัญหาและการให้คำจำกัดความระบบงาน

การตั้งปัญหาเป็นส่วนที่สำคัญที่สุดในการออกแบบและสร้างแบบจำลอง เนื่องจากการตั้งปัญหาที่ถูกต้องจะทำให้สามารถออกแบบและสร้างแบบจำลองให้ตรงกับความต้องการในการใช้งาน ซึ่งจะทำให้คำตอบที่ได้รับสามารถนำไปใช้ได้ตามวัตถุประสงค์

ในการตั้งปัญหาสำหรับการจำลองแบบปัญหานั้น ต้องกำหนดวัตถุประสงค์ในการสร้างแบบจำลองขึ้นก่อน โดยวัตถุประสงค์ในการสร้างแบบจำลองสำหรับงานวิจัยนี้คือ เพื่อใช้ในการศึกษาพฤติกรรมของการจัดผังโรงงานแต่ละวิธีทั้งแบบเชิงปริมาณและเชิงคุณภาพ รวมถึงการเปรียบเทียบการจัดผังโรงงานแต่ละประเภท โดยมีเกณฑ์ที่ใช้ในการเปรียบเทียบคือ

- ค่าใช้จ่ายสำหรับผังโรงงาน (สำหรับการเปรียบเทียบเชิงปริมาณ)
- TCR ของความใกล้ชิด (สำหรับการเปรียบเทียบเชิงคุณภาพ)
- TCR ของระยะทาง (สำหรับการเปรียบเทียบเชิงคุณภาพ)
- เวลาที่ใช้ในการหาคำตอบ โดยใช้จำนวนเจนเนอเรชั่น (สำหรับทั้งการเปรียบเทียบเชิงปริมาณและคุณภาพ)

จะเห็นได้ว่าจากวัตถุประสงค์ ทำให้สามารถสร้างฟังก์ชั่นเป้าหมายทางคณิตศาสตร์สำหรับเกณฑ์ในการเปรียบเทียบ (บทที่ 4) ซึ่งสำหรับเวลาที่ใช้ในการหาคำตอบของวิธีการจัดผังโรงงานนั้นสามารถวัดได้โดยอัตโนมัติจากเวลาในเครื่องคอมพิวเตอร์ ดังนั้นจึงจะไม่กล่าวถึงในการสร้างแบบจำลองปัญหานี้

เมื่อพิจารณาเฉพาะฟังก์ชั่นเป้าหมายในรูปค่าใช้จ่าย TCR ของความใกล้ชิด และ TCR ของระยะทางแล้ว ทำให้สามารถกำหนดองค์ประกอบของระบบงานได้ว่าจะต้องมีองค์ประกอบที่ให้ข้อมูลที่สามารถเชื่อมโยง

- ระยะทางระหว่างแผนก ค่าใช้จ่ายระหว่างแผนก และปริมาณการไหล
- ระยะทางระหว่างแผนกกับความสัมพันธ์ระหว่างแผนก
- ความใกล้ชิดระหว่างแผนกกับความสัมพันธ์ระหว่างแผนก

#### ง.3.1.1 การศึกษาข้อมูลของระบบงาน

ส่วนหนึ่งในการทำความเข้าใจระบบงานและปัญหาที่เกิดขึ้น ได้มาจากการศึกษาข้อมูลของระบบงานซึ่งมักจะอยู่ในรูปแบบของเอกสารต่างๆ โดยข้อมูลดังกล่าวสามารถหาจากการศึกษาข้อมูลของระบบงาน ในที่นี้ได้แก่

- เอกสารทางด้านบัญชี ได้แก่ ค่าขนส่งวัสดุจากแผนกหนึ่งไปยังอีกแผนกหนึ่ง

- เอกสารทางด้านวิศวกรรม ได้แก่ ข้อกำหนดทางด้านวิศวกรรมของการจัดเรียงแผนก การให้คะแนนความสัมพันธ์ระหว่างแผนก (ทั้งในรูป TCR ของระยะทาง และ TCR ของความใกล้ชิด) และระยะทางระหว่างแผนก
- เอกสารทางด้านพัสดุ ได้แก่ ปริมาณพัสดุที่มีการขนย้ายจากแผนกหนึ่งไปยังอีกแผนกหนึ่ง

### ง.3.1.2 การศึกษาการทำงานขององค์ประกอบระบบงาน

ในการวิเคราะห์ระบบงานมักจะกระทำโดยการศึกษาการเคลื่อนที่ขององค์ประกอบของระบบงาน สำหรับการจัดผังโรงงานนี้อาจพิจารณาองค์ประกอบที่เคลื่อนที่คือ พัสดุ โดยพัสดุจะเคลื่อนที่ระหว่างแผนกและก่อให้เกิดค่าใช้จ่าย แต่อย่างไรก็ตามการพิจารณาการเคลื่อนที่ของวัสดุไม่อาจแสดงให้เห็นถึงขั้นตอนการจัดเรียงผังโรงงานของแต่ละวิธี ดังนั้นในการวิเคราะห์ระบบงานโดยการศึกษาการเคลื่อนที่ขององค์ประกอบนี้ จึงเลือกพิจารณาในส่วนของสตริงของผังโรงงานที่อยู่ในระบบของ GAs ซึ่งมีขั้นตอนต่างๆดังต่อไปนี้

*ขั้นตอนที่ 1.*สุ่มสตริงเริ่มต้นจากฮิวริสติกและกำหนดให้สตริงคำตอบนี้เป็นประชากรรุ่นแรกสำหรับ GAs

*ขั้นตอนที่ 2.* คำนวณหาค่าฟิตเนสของสตริงทั้งหมดจากฟิตเนสฟังก์ชั่น

*ขั้นตอนที่ 3.* นำสตริงทั้งหมดเข้ากระบวนการรีโปรดักชั่นเพื่อทำการคัดเลือกประชากรรุ่นใหม่ สตริงที่มีค่าฟิตเนสสูงจะมีโอกาสที่จะถูกคัดเลือกได้มากกว่าสตริงที่มีค่าฟิตเนสต่ำ

*ขั้นตอนที่ 4.* ครอสโอเวอร์ทำการจับคู่เพื่อให้เกิดสตริงใหม่โดยทำการจับคู่อย่างสุ่ม

*ขั้นตอนที่ 5.* มิวเตชั่นทำการเปลี่ยนค่าของยีนบางส่วนเพื่อทำให้เกิดสตริงใหม่ขึ้นอย่างสุ่ม

*ขั้นตอนที่ 6.* กลับไปสู่ขั้นตอนที่ 2. จนกว่าจะครบจำนวนเจนเนอเรชั่นที่กำหนดหรือจนกว่าจะลู่เข้าหาคำตอบ

### ง.3.2 การสร้างแบบจำลอง

การสร้างแบบจำลองคือการเขียนแบบจำลองที่สามารถอธิบายถึงพฤติกรรมของระบบงานตามวัตถุประสงค์ในการศึกษา

แบบจำลองโดยทั่วๆไปแล้วมีอยู่สองประเภทคือ แบบจำลอง Isomorphic และ Homomorphic (ศิริจันทร์, 2537)

- แบบจำลองแบบ Isomorphic คือแบบจำลองเหมือนงานจริงทุกประการ โดยมีเงื่อนไขสองประการคือ ทุกๆองค์ประกอบของระบบงานจริงต้องมีองค์ประกอบที่เหมือนกันในแบบจำลอง และความสัมพันธ์ขององค์ประกอบในแบบจำลองจะต้องเหมือนกับความสัมพันธ์ขององค์ประกอบในระบบงานจริง
- แบบจำลองแบบ Homomorphic คือแบบจำลองจะเหมือนกับระบบงานจริงบางประการ เช่นมีรูปร่างหน้าตาเหมือน, ทำงานได้เหมือน

แบบจำลองที่ใช้ในงานวิจัยนี้เป็นแบบจำลองแบบ Homomorphic ในการทดลองค่าโอเปอร์เรเตอร์ต่างๆจะถูกกำหนดให้เป็นค่าพารามิเตอร์ ได้แก่ จำนวนประชากร ค่าความน่าจะเป็นในการครอสโอเวอร์ ค่าความน่าจะเป็นในการมิวเตชั่น ค่าพารามิเตอร์ในการทดลองได้มาจากการทำการศึกษาเบื้องต้น แบบจำลองที่ใช้ในงานเขียนขึ้นจาก ภาษา C ซึ่งมีรายละเอียดของโปรแกรม การใช้งานและข้อจำกัดต่างๆของโปรแกรมแสดงดังภาคผนวก จ

### ง.3.3 การจัดเตรียมข้อมูล

ข้อมูลในข้อที่ 3.1 เป็นข้อมูลต่างๆที่เกี่ยวข้องกับผังโรงงาน แต่ยังจำเป็นต้องกำหนดข้อมูลเพื่อใช้ในระแบบจำลอง โดยการ

1. การประมาณค่าคงที่และพารามิเตอร์ของโอเปอเรเตอร์ต่าง
2. การใช้การทดสอบความถูกต้องของผลที่ได้จากการจำลองปัญหา

ค่าคงที่และค่าพารามิเตอร์ต่างๆจากข้อ 1 และข้อ 2 สามารถหาได้โดยการทำการศึกษาเบื้องต้นหรืออีกวิธีที่นิยมกันคือ การใช้ค่าโดยประมาณจากงานวิจัยอื่นๆ สำหรับปัญหาของการจัดผังโรงงานโดยใช้ข้อมูลเชิงปริมาณและข้อมูลเชิงคุณภาพสำหรับงานวิจัยนี้เลือกปัญหาจาก Fransis และคณะ (1992)

### ง.3.4 การทดสอบความถูกต้องของแบบจำลอง

การทดสอบความถูกต้องของแบบจำลอง เป็นกระบวนการในการสร้างความมั่นใจให้กับผู้สร้างและผู้ใช้แบบจำลอง ว่าผลที่ได้จากแบบจำลองนั้นควรจะเป็นผลที่ถูกต้องนำไปใช้งานได้ตามวัตถุประสงค์ของการสร้างแบบจำลอง การทดสอบความถูกต้องนั้นไม่มี ."วิธีการทดสอบ" ที่จะบอกได้ว่าแบบจำลองนั้นเป็นแบบจำลองที่ถูกต้องของระบบหรือไม่ ความถูกต้องของแบบจำลองในที่นี้คือความมั่นใจว่ามันเป็นแบบจำลองที่ถูกต้องใช้งานได้ ความมั่นใจดังกล่าวจะได้จากความเข้าใจในระบบงาน ความละเอียดถี่ถ้วนในการตรวจสอบความเหมาะสมขององค์ประกอบ พฤติ

กรรมต่างๆขององค์ประกอบและค่าเชิงปริมาณที่ใช้แทนองค์ประกอบ และความสัมพันธ์ต่างๆ การทดสอบพฤติกรรมขององค์ประกอบงานจริง

กรรมวิธีที่ใช้ในการทดสอบความถูกต้องของแบบจำลองที่ใช้กันอยู่ ประกอบด้วยสามขั้นตอน คือ

**ง.3.4.1 การพิสูจน์ยืนยัน** (Verification)

เป็นการทำให้แน่ใจว่าแบบจำลองมีพฤติกรรมอย่างที่ผู้สร้างต้องการให้มันเป็น วิธีการที่ใช้ในขั้นตอนนี้ได้แก่

**ง.3.4.1.1 การถามความเห็นจากผู้เชี่ยวชาญ** (Face Validity)

การถามความเห็นจากผู้เชี่ยวชาญเป็นการถามความเห็นจากผู้ที่มีความรู้เชี่ยวชาญจากการใช้องค์ประกอบต่างๆในระบบงานและการใช้ระบบงาน ว่าองค์ประกอบและระบบงานนั้นๆมีพฤติกรรมอย่างไรภายใต้เงื่อนไขต่างๆ และการที่องค์ประกอบในแบบจำลองและแบบจำลองมีพฤติกรรมต่างๆ ควรจะเป็นพฤติกรรมที่สอดคล้องกับพฤติกรรมขององค์ประกอบและระบบงานจริงหรือไม่

**ง.3.4.1.2 การทดสอบความถูกต้องของกลไกภายในแบบจำลอง** (Internal Validity)

การทดสอบความถูกต้องของกลไกภายในแบบจำลองเป็นการทดสอบองค์ประกอบในแบบจำลอง โดยการใส่เงื่อนไข เช่นให้ตัวแปรขาเข้า (Input Variable) เป็นค่าคงที่ แล้วพิจารณาว่าผลที่ได้จากองค์ประกอบหรือแบบจำลองหลายๆครั้งมีความแปรปรวนมากน้อยแค่ไหน ถ้ามีความแปรปรวนมาก องค์ประกอบในแบบจำลองหรือแบบจำลองนั้นอาจไม่ถูกต้องและอาจจะต้องมีการแก้ไข

สำหรับงานวิจัยนี้ การทดสอบความถูกต้องของกลไกภายในแบบจำลองทำโดยการแบ่งส่วนของแบบจำลองออกเป็นส่วนๆแล้วทำการป้อนตัวแปรขาเข้าเพื่อให้โปรแกรมทำงานจากนั้นก็ดูผลที่ได้จากแบบจำลอง ซึ่งปรากฏว่าถูกต้องเป็นไปตามหลักการของ GAs

การพิสูจน์ยืนยันของงานวิจัยนี้ทำโดยการเขียนโปรแกรมเพิ่มเติมเพื่อให้แสดงผลออกมาเป็นแฟ้มข้อมูล แฟ้มข้อมูลแรกที่ใช้ในการทดสอบการทำงานของแบบจำลองอย่างละเอียดแสดงผลอยู่ในบทที่ 5 การทดสอบกลไกภายในแบบจำลอง มีการตรวจสอบโดยการเปลี่ยนแปลงค่า

พารามิเตอร์เพื่อให้แน่ใจว่าแบบจำลองสามารถทำงานได้อย่างถูกต้อง โดยอ้างอิงการทำงานของแบบจำลองจาก Michalewicz (1992) และ Goldberg (1974)

**ง.3.4.1.3 การทดสอบความถูกต้องของตัวแปรและพารามิเตอร์** (Variable-Parameters Validity)

การทดสอบความถูกต้องของตัวแปรและค่าพารามิเตอร์เป็นการทดสอบความไว (Sensitivity Testing) ของการเปลี่ยนแปลงค่าของตัวแปรและพารามิเตอร์ว่ามีผลกระทบต่อผลลัพธ์ที่ได้จากองค์ประกอบในแบบจำลองและแบบจำลองอย่างไร ถ้าผลที่ได้ มีการเปลี่ยนแปลงหรือไวต่อค่าตัวแปรหรือพารามิเตอร์ใด ก็เป็นเครื่องแสดงบอกให้เรารู้ว่าจะต้องระวังให้มากต่อการประมาณค่าตัวแปรและพารามิเตอร์เหล่านั้น นอกจากนั้นแล้ว การทดสอบความไวนี้ยังช่วยให้ผู้สร้างแบบจำลองได้เห็นว่าองค์ประกอบในแบบจำลองและแบบจำลองประพฤติตนอย่างที่ควรจะเป็นหรือไม่ จากการทดลองพบว่าค่าพารามิเตอร์ของการมิวเตชั่นและจำนวนประชากรไวต่อการหาคำตอบมากกว่าค่าพารามิเตอร์ของการครอสโอเวอร์ ดังนั้นถ้าต้องการนำไปใช้งานจริงจึงควรทดลองแปรค่าพารามิเตอร์ของการมิวเตชั่นและจำนวนประชากร ซึ่งค่าพารามิเตอร์ทั้งสองนี้มีผลในการหาคำตอบและเจนเนอเรชั่นที่พบคำตอบ

**ง.3.4.1.4 การทดสอบความถูกต้องของสมมุติฐาน** (Hypothesis Validity)

การทดสอบความถูกต้องของสมมติฐานเป็นการทดสอบความถูกต้องทางสถิติว่าผลที่ได้จากองค์ประกอบในแบบจำลองกับผลที่ได้จากองค์ประกอบของระบบงานจริงนั้นเหมือนกัน โดยอาจใช้เงื่อนไขต่างๆที่มีปรากฏในอดีต ใส่ให้กับองค์ประกอบในแบบจำลองกับผลที่ได้กับระบบงานจริงนั้นเหมือนกัน โดยอาจใช้เงื่อนไขต่างที่มีปรากฏจากข้อมูลในอดีต ใส่ให้กับองค์ประกอบในแบบจำลองแล้วเปรียบเทียบผลที่ได้กับผลที่ได้ในอดีตว่าสามารถยอมรับว่าเหมือนกันโดยมีระดับนัยสำคัญที่สามารถยอมรับได้

สำหรับงานวิจัยนี้เลือกคำตอบจากวิธีการทางฮิวริสติกนำมาเปรียบเทียบกันปรากฏว่าคำตอบถูกต้องเช่นเดียวกัน

**ง.3.4.2 การทดสอบความถูกต้อง** (Validation)

เป็นการทดสอบความสอดคล้องระหว่างพฤติกรรมของแบบจำลองกับพฤติกรรมจริง ทั้งนี้โดยอาศัยการเปรียบเทียบระหว่างข้อมูลที่ได้จากแบบจำลองกับข้อมูลในอดีตของระบบงานจริงที่เงื่อนไขของการใช้ระบบงานที่เหมือนกัน การวิเคราะห์ทำได้โดยอาศัยเทคนิคทางสถิติ ได้แก่

3.4.2.1การทดสอบสมมุติฐานในการเปรียบเทียบค่าพารามิเตอร์ของแบบจำลองกับของระบบงานจริง จากตัวอย่างการคำนวณในบทที่ 5 จะเห็นได้ว่าในส่วนของการกำหนดค่าความน่าจะเป็นของการมิวเตชั่นและการครอสโอเวอร์ จำนวนตำแหน่งที่เกิดมีค่าใกล้เคียงกับค่าที่กำหนดไว้

3.4.2.2 การทดสอบสมมุติฐานของลักษณะการกระจายของความน่าจะเป็นของข้อมูลจากแบบจำลองเปรียบเทียบกับระบบงานจริง การกระจายความน่าจะเป็นของข้อมูลใช้การกระจายแบบปรกติ

3.4.2.3 การประมาณค่าพารามิเตอร์ของแบบจำลองเปรียบเทียบกับค่าประมาณของพารามิเตอร์ของระบบงานจริง ก่อนทำการทดลองจำเป็นต้องทำการทดลองเบื้องต้นและทำการเปรียบเทียบคำตอบที่ได้กับวิธีการทางฮิวริสติกซึ่งการทำการทดลองเบื้องต้นมีรายละเอียดอยู่ในบทที่ 7

3.4.2.4 การพยากรณ์ความสัมพันธ์ระหว่างตัวแปรและพารามิเตอร์ในแบบจำลองเปรียบเทียบกับของระบบงานจริง เนื่องจากแบบจำลองที่ใช้งานไม่สามารถที่จะเปรียบเทียบกับระบบงานจริงได้จึงทำการทดลองเปรียบเทียบกับฮิวริสติกและทำการทดลองเปรียบเทียบกันเองโดยเปลี่ยนแปลงการชนิดของการครอสโอเวอร์

**ง.3.4.3 การวิเคราะห์ปัญหา** (Problems Analysis)

การวิเคราะห์ปัญหาเป็นการทดลองใช้แบบจำลองในการพยากรณ์พฤติกรรมต่างๆ ของระบบงานเปรียบเทียบกับพฤติกรรมจริงของระบบงาน

การทดสอบความถูกต้องของแบบจำลองในงานวิจัยฉบับนี้ ทำโดยการเปรียบเทียบกับฮิวริสติก และตรวจสอบผลที่ได้จากแบบจำลอง สมมุติฐานของลักษณะการกระจายของข้อมูลโดยส่วนใหญ่มีการกระจายแบบปรกติ และค่าพารามิเตอร์ที่ใช้ในการทดลองได้จากการทำการศึกษาเบื้องต้น

### ง.3.5 การออกแบบการทดลอง

เนื้อหาในส่วนนี้เกี่ยวกับการออกแบบการทดลองที่ใช้ในงานวิจัย รูปแบบของปัญหาในงานวิจัย ซึ่งการทดลองจะกล่าวอย่างละเอียดในบทที่ 7 ต่อไป


**รูปที่ ง.1** รูปแบบของปัญหาที่ใช้ข้อมูลเชิงปริมาณและข้อมูลเชิงคุณภาพในงานวิจัย

ลักษณะของปัญหาในแบบจำลองที่ใช้ในการทดลองแบ่งออกเป็นสองประเภทดังที่ได้กล่าวมาแล้ว สามารถสรุปได้ดังในตารางที่รูปที่ ง.1 ปัญหาที่ใช้ในการทดลองแบ่งออกเป็นสองประเภทดังที่ได้กล่าวมาแล้ว คือปัญหาการจัดผังโรงงานโดยใช้ข้อมูลเชิงปริมาณและปัญหาการจัดผังโรงงานโดยใช้ข้อมูลเชิงคุณภาพ

### ง.3.5.1 ปัญหาการจัดผังโรงงานโดยใช้ข้อมูลเชิงปริมาณ

การทดลองของปัญหาการจัดผังโรงงานโดยใช้ข้อมูลเชิงปริมาณในงานวิจัยนี้แบ่งออกเป็น 5 การทดลอง วัตถุประสงค์หลักคือทำการปรับปรุงระยะเวลาในการหาคำตอบของ GAs ดังมีรายละเอียดดังต่อไปนี้

การทดลองที่ 1 เป็นการทดลองในการหาคำตอบของปัญหาการจัดผังโรงงานด้วยวิธีฮิวริสติก โดยมีวัตถุประสงค์เพื่อ เปรียบเทียบคำตอบที่ได้จากฮิวริสติกและจะนำไปใช้เป็นคำตอบเริ่มต้นให้กับ GAs ต่อไป

การทดลองที่ 2 เป็นการทดลองเพื่อหาค่าพารามิเตอร์ที่เหมาะสมของ GAs วัตถุประสงค์เพื่อ หาค่าพารามิเตอร์ที่เหมาะสมเพื่อให้การหาคำตอบเป็นไปอย่างรวดเร็วและได้คำตอบที่ดีหรือใกล้เคียง

การทดลองที่ 3 เป็นการทดลองเปรียบเทียบการครอสโอเวอร์แบบ PMX, OX และ CX มีวัตถุประสงค์เพื่อ เปรียบเทียบประสิทธิภาพของการครอสโอเวอร์ทั้งสามแบบและหาวิธีการครอสโอเวอร์ที่เหมาะสมเพื่อใช้ในการการทดลองต่อไป

การทดลองที่ 4 เป็นการทดลองให้คำตอบที่ได้จากฮิวริสติกเริ่มต้นแก่ GAs โดยให้สตริงคำตอบเริ่มต้นเป็นแบบเดียวกัน วัตถุประสงค์เพื่อ ทำการปรับปรุงประสิทธิภาพของ GAs ให้หาคำตอบได้รวดเร็วยิ่งขึ้น

การทดลองที่ 5 เป็นการทดลองให้คำตอบที่ได้จากฮิวริสติกเริ่มต้นแก่ GAs โดยให้สตริงคำตอบเริ่มต้นที่แตกต่างกัน วัตถุประสงค์เพื่อ ทำการปรับปรุงประสิทธิภาพของ GAs ให้หาคำตอบได้รวดเร็วยิ่งขึ้น

การทดลองเหล่านี้จำเป็นต้องมีการกำหนดค่าพารามิเตอร์ต่างๆ ได้แก่

- จำนวนประชากร
- จำนวนประชากรคำตอบเริ่มต้น
- จำนวนเจนเนอเรชั่น
- ค่าความน่าจะเป็นของการครอลโอเวอร์
- ค่าความน่าจะเป็นของการมิวเตชั่น

และนอกจากนั้นข้อมูลเบื้องต้นของผังโรงงาน ได้แก่

- จำนวนแผนกหรือสถานีงานทั้งหมด
- ปริมาณการไหลของแต่ละแผนก (จากแผนภูมิจาก -ไป)
- ค่าใช้จ่ายของการไหลในแต่ละแผนก (จากแผนภูมิจาก -ไป)

ส่วนรายละเอียดในการทดลองอยู่ในบทที่ 6

### ง.3.5.2 ปัญหาการจัดผังโรงงานโดยใช้ข้อมูลเชิงคุณภาพ

การทดลองของปัญหาการจัดผังโรงงานโดยใช้ข้อมูลเชิงปริมาณในงานวิจัยนี้แบ่งออกเป็น 5 การทดลอง วัตถุประสงค์หลักคือของการทดลอง เพื่อเป็นการเปรียบเทียบคำตอบและเจนเนอเรชั่นที่พบคำตอบของการครอสโอเวอร์แบบ PMX, OX และ CX กับปัญหาแบบต่างๆและพิจารณาผลปัจจัยแต่ละปัจจัยและขนาดของผังโรงงานที่พิจารณาปฏิสัมพันธ์ที่มีผลต่อคำตอบและเจนเนอ

เรชั่นที่พบคำตอบ เพื่อศึกษาและพิจารณาเป็นแนวทางในการนำ GAs ในการแก้ปัญหาการจัดผังโรงงานโดยใช้ข้อมูลเชิงคุณภาพไปใช้งาน

การทดลองที่ 1 การจัดผังโรงงานโดยใช้ข้อมูลเชิงคุณภาพโดยพิจารณา TCR ระยะทางระหว่างแผนกและให้ระดับคะแนนแบบลิเนียร์ ของผังโรงงานขนาด 10 แผนก

การทดลองที่ 2 การจัดผังโรงงานโดยใช้ข้อมูลเชิงคุณภาพโดยพิจารณา TCR ระยะทางระหว่างแผนกและให้ระดับคะแนนแบบเอ็กซ์โปเนนเชียล ของผังโรงงานขนาด 10 แผนก

การทดลองที่ 3 การจัดผังโรงงานโดยใช้ข้อมูลเชิงคุณภาพโดยพิจารณา TCR ระยะทางระหว่างแผนกและให้ระดับคะแนนแบบเอ็กซ์โปเนนเชียล ของผังโรงงานขนาด 20 แผนก

การทดลองที่ 4 การจัดผังโรงงานโดยใช้ข้อมูลเชิงคุณภาพโดยพิจารณา TCR ความใกล้ชิดระหว่างแผนกและให้ระดับคะแนนแบบลิเนียร์ ของผังโรงงานขนาด 10 แผนก

การทดลองที่ 5 การจัดผังโรงงานโดยใช้ข้อมูลเชิงคุณภาพโดยพิจารณา TCR ความใกล้ชิดระหว่างแผนกและให้ระดับคะแนนแบบเอ็กซ์โปเนนเชียล ของผังโรงงานขนาด 10 แผนก

ค่าพารามิเตอร์ในการทดลองเหมือนกับการทดลองการจัดผังโรงงานโดยใช้ข้อมูลเชิงปริมาณในหัวข้อ ง.3.5.1 ส่วนข้อมูลของผังโรงงานเบื้องต้นได้แก่แผนภูมิแสดงความสัมพันธ์และขนาดของผังโรงงาน รายละเอียดของการทดลองแสดงดังในบทที่ 7

# ภาคผนวก จ
# คำอธิบายการใช้งานโปรแกรม

เนื้อหาในบทนี้เกี่ยวข้องกับ ลักษณะของโปรแกรมที่ใช้ในงานวิจัย ข้อจำกัดต่างๆในการใช้งานของโปรแกรม ลักษณะของโปรแกรมกับรูปแบบปัญหาและการใช้งานโปรแกรม

โปรแกรมที่ใช้ในงานวิจัยถูกเขียนด้วนภาษา C ซึ่งบางโปรแกรมมีทั้งลักษณะเป็นแบบโครงสร้าง (Structure) และบางโปรแกรมลักษณะเชิงวัตถุ หรือ OOP (Object Oriented Programming) โปรแกรมที่เป็นฮิวริสติกที่ใช้ในงานวิจัยได้แก่ คอนสตรักชั่นฮิวริสติก และ SDPI ได้ถูกพัฒนาเป็นแบบโครงสร้างและแบบเชิงวัตถุตามลำดับ ดังแสดงอยู่ในภาคผนวก ฎ และ ภาคผนวก ฎ ส่วนโปรแกรมในการหาคำตอบของ GAs แสดงอยู่ในภาคผนวก ช ถึง ภาคผนวก ญ

## จ.1 รายละเอียดของโปรแกรม

โปรแกรมในการแก้ปัญหาการจัดบล็อกแพลนในงานวิจัยนี้มี 2 ประเภทคือโปรแกรมที่ใช้ GAs ในการแก้ปัญหาและโปรแกรมที่เป็นฮิวริสติกในการแก้ปัญหา โปรแกรมที่ใช้ GAs ประกอบด้วยอยู่ 4 โปรแกรม ได้แก่ quaclose.exe, quadist.exe, quandist.exe และ quaninit.exe ซึ่งมีรายละเอียดดังต่อไปนี้

quaclose เป็นโปรแกรมสำหรับการจัดบล็อกแพลนสำหรับข้อมูลเชิงคุณภาพ โดยพิจารณาถึงความใกล้ชิด

quadist เป็นโปรแกรมสำหรับการจัดบล็อกแพลนสำหรับข้อมูลเชิงคุณภาพ โดยพิจารณาถึงระยะทาง

quandist เป็นโปรแกรมสำหรับการจัดบล็อกแพลนสำหรับข้อมูลเชิงมาณ

quaninit เป็นโปรแกรมสำหรับการจัดบล็อกแพลนสำหรับข้อมูลเชิงมาณ และสามารถกำหนดคำตอบเริ่มต้นได้

สำหรับโปรแกรมที่ใช้ฮิวริสติกประกอบด้วย 2 โปรแกรมคือ craft.exe และ construct.exe

craft เป็นโปรแกรมสำหรับการจัดบล็อกแพลนสำหรับข้อมูลเชิงมาณโดยใช้วิธีการ SDPI

construct เป็นโปรแกรมสำหรับการจัดบล็อกแพลนสำหรับข้อมูลเชิงมาณโดยใช้วิธีการ construction

โปรแกรมดังกล่าวทำงานบนระบบปฏิบัติการ WINDOWS 95 หรือ WINDOWS NT ส่วนทฤษฎีและหลักการของโปรแกรมเหล่านี้สามารถศึกษาได้จากในบทก่อน

## จ.2 ข้อจำกัดของโปรแกรมการในการใช้งาน

ข้อจำกัดของโปรแกรมที่ใช้ในงานวิจัยนี้ก็คือ

- แผนกต่างๆจะถูกกำหนดให้อยู่ภายในตำแหน่งเดียว ไม่มีตำแหน่งใดๆที่มีสถานีมากกว่าสองแผนกขึ้นไป
- ระยะห่างระหว่างจุดศูนย์กลางขนาดของแต่ละสถานีมีขนาด 1 หน่วยระยะทาง
- สถานีแรกที่ต้องกำหนดคือสถานีที่ 0 ไปจนถึงสถานีสุดท้าย ตำแหน่งพื้นที่แรกที่กำหนดคือตำแหน่งที่ 0 จนถึงตำแหน่งพื้นที่สุดท้าย
- จำนวนพื้นที่ต้องเป็นจำนวนคู่และต้องมากกว่าจำนวนสถานี ถ้าจำนวนพื้นที่มากกว่าจำนวนสถานีแล้วต้องกำหนดให้สถานีหุ่น (Dummy Station) เพื่อให้จำนวนสถานีเท่ากับจำนวนพื้นที่
- จำนวนแผนกไม่เกิน 256 แผนก
- จำนวนประชากรในทางทฤษฎีไม่เกิน 256 แผนก แต่ในทางปฏิบัติไม่ควรเกิน 100 แผนกเนื่องจากข้อจำกัดทางด้านความละเอียดของทศนิยม ซึ่งอาจมีผลทำให้การรีโปรดักชั่นผิดพลาดมาก
- ในกรณีที่แก้ปัญหาการจัดผังโรงงานแบบข้อมูลเชิงคุณภาพ ค่า TCR ของแต่ละแผนกต้องไม่เกิน +/- 32526
- ในกรณีที่แก้ปัญหาการจัดผังโรงงานแบบข้อมูลเชิงปริมาณ ค่าใช้จ่ายและความถี่ในการไหลของแต่ละแผนก TCR ของแต่ละแผนก ต้องไม่เกิน 32526

## จ.3 การใช้งานโปรแกรมแก้ปัญหาการจัดผังโรงงาน

การใช้งานโปรแกรมสามารถแบ่งออกเป็นสองส่วนคือ การใช้งานโปรแกรมในส่วนของ GAs และ การใช้งานโปรแกรมในส่วนของฮิวริสติก ซึ่งมีรานละเอียดดังต่อไปนี้

### จ.3.1 การแก้ปัญหาการวางผังโรงงานด้วย GAs

การใช้งานโปรแกรมในส่วนนี้จะประกอบไปด้วยขั้นตอนการเตรียมข้อมูล การป้อนค่าพารามิเตอร์ต่างๆและการอ่านผลซึ่งมีขั้นตอนดังต่อไปนี้

- การเตรียมข้อมูลเบื้องต้น

การเตรียมข้อมูลเบื้องต้นเป็นการจัดเตรียมข้อมูลของผังที่ต้องการจะออกแบบ ขึ้นอยู่กับว่าผู้วิเคราะห์ต้องการออกแบบโดยพิจารณาข้อมูลในด้านใด ก่อนการใช้งานโปรแกรมทุกครั้งต้องพิจารณาว่าจะใช้โปรแกรมใดในการวิเคราะห์เพื่อที่จะได้จัดเตรียมข้อมูลให้เหมาะสม

ข้อมูลเบื้องต้นในการใช้งานโปรแกรมได้แก่ ขนาดและจำนวนสถานี แผนภูมิจาก-ไป ของค่าใช้จ่ายและความถี่ หรือ แผนภูมิแสดงความสัมพันธ์ระหว่างสถานี ในกรณีที่พิจารณาถึงข้อมูลเชิงคุณภาพจะใช้ข้อมูลจากแผนภูมิแสดงความสัมพันธ์ แต่ถ้าพิจารณาถึงข้อมูลเชิงปริมาณจะใช้ข้อมูลจากแผนภูมิจาก-ไป ของค่าใช้จ่ายและความถี่ โดยมีจำนวนสถานีเป็นไปตามข้อจำกัดข้างต้น ข้อมูลที่เป็นอินพุทสำหรับโปรแกรม quaclose, quadist คือไฟล์ init.dat ข้อมูลที่เป็นอินพุทสำหรับ quandist คือไฟล์ frequency.dat และ cost.dat ข้อมูลที่เป็นอินพุทสำหรับ quaninit ไฟล์ init.dat, frequency.dat และ cost.dat ข้อมูลทั้งหมดจะถูกบันทึกอยู่ในรูปของข้อความ (Text) สามารถแก้ไขโดยใช้โปรแกรมเท็กซ์อีดิเตอร์ (Text Editer) ใดก็ได้ ลักษณะของแฟ้มข้อมูลต่างๆมีรายลักษณะดังต่อไปนี้

- ข้อมูลที่เป็นอินพุทสำหรับโปรแกรม quaclose, quadist คือไฟล์ init.dat โดยที่ไฟล์ดังกล่าวมีลักษณะดังต่อไปนี้

```
2
5
2
0
0
0
0
.
.
.
0
0
-1
2
0
3
```

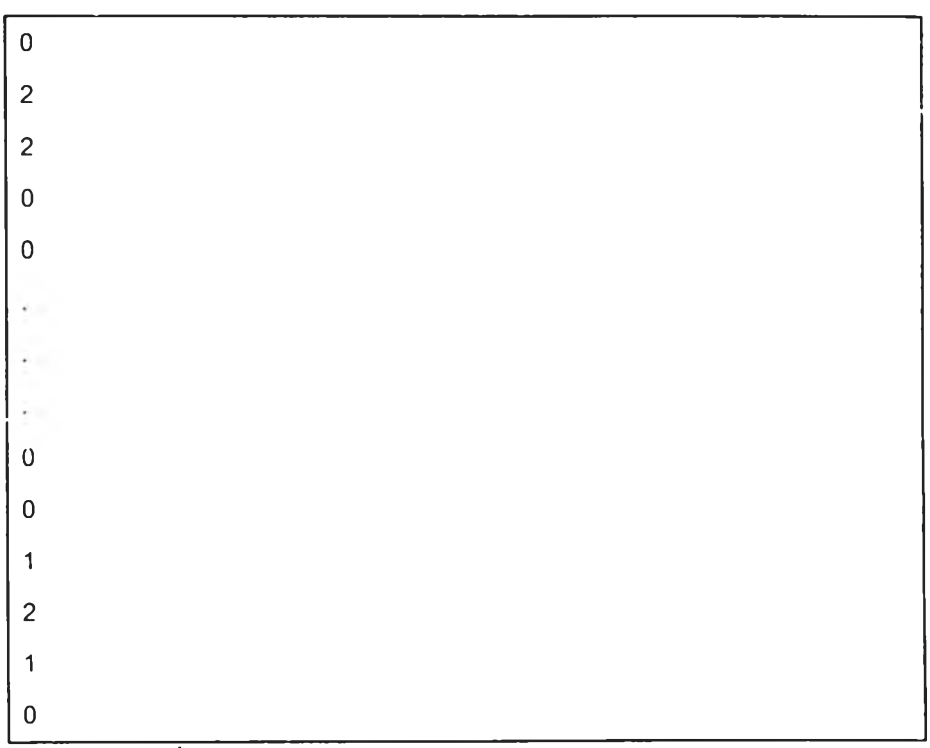
**รูปที่ จ1** ข้อมูลอินพุทสำหรับโปรแกรม quaclose และ quadist จากไฟล์ init.dat

การบันทึกข้อมูลจากแผนภูมิแสดงความสัมพันธ์ สองบรรทัดแรกแสดงถึง จำนวนแถวและจำนวนหลักของสถานีตามลำดับ ตั้งแต่บรรทัดที่ 3 ไปจนถึงบรรทัดสุดท้าย คือคะแนนของระดับความสัมพันธ์จากแผนภูมิแสดงความสัมพันธ์ จากตัวอย่างในรูป จ1 บรรทัดที่ 3 คือคะแนนระดับความสัมพันธ์ของแผนกที่ 0 และแผนกที่ 1 บรรทัดที่ 4 คือคะแนนระดับความสัมพันธ์ของแผนกที่ 0 และแผนกที่ 2 เป็นต้น

การให้คะแนนระดับความสัมพันธ์อาจกำหนดได้ทั้งรูปแบบ เอ็กซ์โปเนนเชียล, ลิเนียร์ หรือแล้วแต่ผู้ใช้กำหนดค่าก็ได้ โดยนำค่าจากแผนภูมิแสดงความสัมพันธ์มาป้อนโดยตรง

- ข้อมูลที่เป็นอินพุทสำหรับ quandist คือไฟล์ frequency.dat และ cost.dat ข้อมูลที่เป็นอินพุทสำหรับ quaninit ไฟล์ init.dat, frequency.dat และ cost.dat ไฟล์ init.dat เป็นไฟล์ที่ใช้เก็บคำตอบเริ่มต้นที่จะป้อนเข้า GAs

ไฟล์ frequency.dat และ cost.dat เป็น ไฟล์ที่ได้จากแผนภูมิจาก-ไป ซึ่งการป้อนข้อมูลจากแผนภูมิลงไฟล์นั้นมีข้อแตกต่างเพียงเล็กน้อยคือ ไฟล์ frequency.dat จำเป็นที่จะต้องกำหนดสองบรรทัดแรกเป็นขนาดของสถานี ส่วน cost.dat ไม่ต้องกำหนดสองบรรทัดแรกให้เป็นขนาดของสถานี สามารถกำหนดค่าจากแผนภูมิจากไปได้โดยตรง




**รูปที่ จ2** ข้อมูลอินพุทสำหรับโปรแกรม quaninit จากไฟล์ cost.dat

จากรูป จ2 แสดงถึงบรรทัดแรกแสดงถึงค่าใช้จ่ายของการไหลระหว่างแผนกที่ 0 และแผนกที่ 1 บรรทัดที่สองแสดงถึงค่าใช้จ่ายของการไหลระหว่างแผนกที่ 0 และแผนกที่ 2 เป็นต้น

```
2
5
0
2
1
0
0
.
.
.
0
1
1
2
0
```

**รูปที่ จ3 ข้อมูลอินพุทสำหรับโปรแกรม** quaninit **จากไฟล์** frequency.dat

จากรูป จ3 สองบรรทัดแรกแสดงถึงขนาดของผังโรงงาน บรรทัดที่สามแสดงถึงแสดงถึงค่าใช้จ่ายของการไหลระหว่างแผนกที่ 0 และแผนกที่ 1 บรรทัดที่สี่แสดงถึงค่าใช้จ่ายของการไหลระหว่างแผนกที่ 0 และแผนกที่ 2 เป็นต้น

ไฟล์ init.dat ในที่นี้หมายถึง ไฟล์ที่ใช้กำหนดสตริงเริ่มต้นให้กับ GAs โดยมีรูปแบบการกำหนดดังรูปที่ จ4 บรรทัดแรกของไฟล์ init.dat คือจำนวนสถานีที่จะกำหนดให้เป็นคำตอบเริ่มต้น โดยจะต้องไม่มีมากไปกว่าจำนวนประชากร จากตัวอย่างจะเห็นได้ว่ามีสตริงเริ่มต้นอยู่สองตัวคือ [ 0 1 5 4 7 2 3 6 8 9 ] และ [ 2 3 5 4 1 7 8 6 9 0 ]

**ข้อควรระวัง** การกำหนดค่าผิดอาจทำให้โปรแกรมทำงานผิดพลาดได้

```
2
0
1
5
4
7
2
3
```

```
6
8
9
2
3
5
4
1
7
8
6
9
0
```

**รูปที่ จ4** แสดงถึงข้อมูลอินพุทสำหรับโปรแกรม quaninit จากไฟล์ init.dat

- กำหนดค่าพารามิเตอร์ต่างของ GAs

กำหนดค่าพารามิเตอร์ต่างของ GAs เพื่อใช้กับโปรแกรม quaclose, quadist, quandist และ quaninit ได้แก่

- ค่าพารามิเตอร์ของจำนวนประชากร
- ค่าพารามิเตอร์ของจำนวนเจนเนอเรชั่น
- ค่าพารามิเตอร์ของความน่าจะเป็นของการครอสโอเวอร์ PMX, OX และ CX
- ค่าพารามิเตอร์ของความน่าจะเป็นของการมิวเตชั่น
- กำหนดจำนวนสตริงและจำนวนเจนเนอเรชั่นที่ต้องการให้แสดงคำตอบ

กำหนดจำนวนสตริงและจำนวนเจนเนอเรชั่นที่ต้องการให้แสดงคำตอบ ค่าพารามิเตอร์ในส่วนนี้ไม่มีผลในการคำนวณใดๆ แต่มีเพื่อความสะดวกในการพิจารณาคำตอบ

- การเรียกใช้งานโปรแกรม

การเรียกโปรแกรมใช้งานในการวิเคราะห์ด้วย GAs ที่ประกอบไปด้วย 4 โปรแกรมที่กล่าวมาแล้วได้แก่ quaclose, quadist, quandist และ quaninit สามารถเรียกใช้งานโปรแกรมเหล่านี้โดยเรียกชื่อโปรแกรม โดยที่จะต้องเตรียมไฟล์ข้อมูลต่างๆให้เรียบร้อยและอยู่ในไดเรคตอรี่ (directory) เดียวกันดังตารางที่ จ.1

| ลำดับ | โปรแกรม | ไฟล์ข้อมูล |
|---|---|---|
| 1 | quaclose.exe | init.dat |
| 2 | quadist.exe | init.dat |
| 3 | quandist.exe | frequenct.dat, cost.dat |
| 4 | quanini.exe | frequenct.dat, cost.dat, init.dat |

**ตารางที่ จ.1** สรุปไฟล์ข้อมูลที่ต้องเตรียมให้กับโปรแกรม

```
C:\quality\2_5clonesslinear\quaclose
Program Plant Layout version 1.01 for cloness
Last update 14/09/97
Enter Generation                :100
Enter Population                :10
Enter Probability PMX           :0.8
Enter Probability OX            :0
Enter Probability CX            :0
Enter Proability Mutation       :0.1
How many period for report      :20
How many report chromosome      :5
```

รูปที่ จ5 การป้อนค่าพารามิเตอร์เมื่อเรียกโปรแกรม quaclose

รูปที่ จ5 แสดงถึงการป้อนข้อมูลเมื่อเรียกโปรแกรม quaclose จะเห็นได้ว่าอันดับแรกเป็นการป้อนข้อมูลของจำนวนเจนเนอเรชั่น, จำนวนประชากรในเมทติ้งพูล, ค่าความน่าจะเป็นของการครอสโอเวอร์แบบ PMX, OX และ CX ค่าความน่าจะเป็นของการมิวเตชั่น, จำนวนช่วงระหว่างเจนเนอเรชั่นที่ต้องการบันทึกลงแฟ้มข้อมูล, จำนวนสตริงที่ดีที่สุดที่ต้องการบันทึกลงแฟ้มข้อมูล

จำนวนช่วงระหว่างเจนเนอเรชั่นที่ต้องการบันทึกลงแฟ้มข้อมูล จากตัวอย่างเป็นกำหนดให้เป็น 20 หมายความว่าทุกๆ 20 เจนเนอเรชั่นจะมีการบันทึกลงแฟ้มข้อมูล 1 ครั้งและจำนวนสตริงที่ดีที่สุดที่ต้องการบันทึกลงแฟ้มข้อมูล จากตัวอย่างกำหนดให้เป็น 5 หมายความว่าจำนวนสตริงที่ดีที่สุด 5 อันดับแรกจะถูกบันทึกลงแฟ้มข้อมูล

จากรูปที่ จ6 โปรแกรมจะทวนค่าข้อมูลในแฟ้ม input.dat เพื่อให้ผู้ใช้สามารถตรวจสอบจำนวนและลักษณะของข้อมูล และเมื่อโปรแกรมทำงานจะมีการแสดงจำนวนเจนเนอเรชั่นที่คำนวณผ่านไปแล้วเพื่อให้ผู้ใช้ทราบถึงระยะเวลาโดยประมาณได้เนื่องจากบางครั้งการคำนวณใน

บางครั้งอาจใช้ระยะเวลานาน ส่วนบรรทัดสุดท้ายเป็นระยะเวลาของการคำนวณทั้งหมดโดยรวมตั้งแต่เริ่มทำการคำนวณที่เจนเนอเรชั่นแรก มีหน่วยเป็นวินาที

```
MemVij is
2000002-1413344232004400-1400000001102000000-1203
Generation 0 Completed
Generation 1 Completed
Generation 2 Completed
Generation 3 Completed
.
.
.
Generation 97 Completed
Generation 98 Completed
Generation 99 Completed
Function successfully used time = 1.37 sec.
```

**รูปที่ จ6** การแสดงผลของโปรแกรม

- การอ่านผลจากที่ได้จากโปรแกรม

การเรียกข้อมูลที่ผ่านการคำนวณ ข้อมูลที่ได้จากโปรแกรมจะมีแฟ้มข้อมูลสองแฟ้มคือ result.txt และ summary.txt ข้อมูลจาก result.txt จะเป็นข้อมูลสรุปในทุกๆเจนเนอเรชั่นโดยประกอบด้วยข้อมูลของ คำตอบที่ดีที่สุดในเจนเนอเรชั่น (Min) คำตอบที่มีค่าที่มากที่สุดในเจนเนอเรชั่น (Max) ค่าเฉลี่ยของคำตอบในเจนเนอเรชั่น (Average) ผลรวมของคำตอบที่จะนำไปทำรีโปรดักชั่น (Sum) ค่าเบี่ยงเบนมาตรฐาน (STD) ของประชากรในเมทติ้งพูล และสุดท้ายคือสตริงคำตอบที่ดีที่สุด ในกรณีที่ข้อมูลเป็นแบบเชิงคุณภาพการวัดคำตอบจะเป็น TCR และถ้าข้อมูลเป็นแบบเชิงปริมาณวัดคำตอบจะเป็นค่าใช้จ่าย ดังมีรายละเอียดดังต่อไปนี้

รูปที่ จ7 แสดงถึงรายละเอียดของไฟล์ result.txt ดังที่ได้กล่าวมาแล้ว รูปที่ จ8 ในแฟ้ม summary.txt ภายในแฟ้มนี้จะประกอบด้วย สตริงที่ถูกจัดลำดับโดยสตริงที่มีค่าที่ดีที่สุดจะถูกจัดเรียงไว้เป็นอันดับแรกโดยมีหมายเลขของสตริงในเมทติ้งพูลนั้นๆระบุด้วย TCR ของสตริงแต่ละตัว และข้อมูลสรุปของสตริง ค่า TCR โดยเฉลี่ยภายในเจนเนอเรชั่นนั้นๆ

ข้อมูลจากแฟ้ม result.txt และ summary.txt สามารถดูได้จากโปรแกรมเท็กซ์อีดิเตอร์ทั่วๆไปได้ และสามารถนำไปวิเคราะห์ต่อไป

| Min | Max | Average | Sum | STD. | String |
|---|---|---|---|---|---|
| 19 | 29 | 23.3 | 233 | 3.802046 | [ 8 3 4 1 2 0 9 6 5 7 ] |
| 19 | 29 | 24.4 | 244 | 3.777124 | [ 8 3 4 1 2 0 9 6 5 7 ] |
| 19 | 30 | 24.4 | 244 | 4.273952 | [ 4 8 5 6 3 0 9 2 1 7 ] |
| 19 | 34 | 28.1 | 281 | 4.863698 | [ 1 6 5 0 9 3 4 2 8 7 ] |
| 23 | 34 | 27 | 270 | 4.2947 | [ 1 6 5 0 9 3 4 2 8 7 ] |
| . | . | . | . | . | . |
| . | . | . | . | . | . |
| . | . | . | . | . | . |
| . | . | . | . | . | . |
| 19 | 41 | 28 | 280 | 8.082904 | [ 6 1 4 7 9 2 5 3 8 0 ] |
| 19 | 41 | 31.6 | 316 | 7.426679 | [ 6 1 4 7 9 2 5 3 8 0 ] |
| 20 | 41 | 27.9 | 279 | 6.332456 | [ 6 1 4 7 9 2 5 3 8 0 ] |
| 22 | 41 | 28.8 | 288 | 7.299924 | [ 6 1 4 7 9 2 5 3 8 0 ] |
| 18 | 41 | 27.1 | 271 | 7.309811 | [ 6 1 4 7 9 2 5 3 8 0 ] |

**รูปที่ ผ.7** ลักษณะการจัดเรียงข้อมูลที่ได้จากไฟล์ result.txt

```
[Generation 0]
[8]     8 3 4 1 2 0 9 6 5 7 =     29.000000
[0]     8 3 4 1 2 0 9 6 5 7 =     29.000000
[6]     2 8 6 0 4 5 9 1 3 7 =     26.000000
[7]     9 2 6 1 3 5 4 0 7 8 =     25.000000
[1]     7 8 3 1 9 5 6 4 0 2 =     23.000000

The Best TCR is : 29.000000       [ 8 3 4 1 2 0 9 6 5 7 ]
Average is : 23.300000
STD .   is : 3.802046

[Generation 20]
[5]     5 6 8 9 3 2 1 7 4 0 =     39.000000
[2]     5 6 8 9 0 2 1 7 4 3 =     39.000000
[0]     5 6 8 9 3 2 1 7 4 0 =     39.000000
[8]     5 6 8 3 9 2 1 7 4 0 =     36.000000
```

```
[9]      5 6 0 4 3 2 9 8 1 7 =      33.000000

The Best TCR is : 39.000000      [ 5 6 8 9 3 2 1 7 4 0 ]
Average is : 32.000000
STD.    is : 6.548961
.
.
.
.
[Generation 80]
[1]      6 1 4 7 9 2 5 3 8 0 =      41.000000
[0]      6 1 4 7 9 2 5 3 8 0 =      41.000000
[9]      6 1 7 9 8 2 5 0 4 3 =      38.000000
[8]      8 1 5 9 7 2 6 3 4 0 =      32.000000
[2]      8 1 5 9 7 2 6 0 3 4 =      29.000000

The Best TCR is : 41.000000      [ 6 1 4 7 9 2 5 3 8 0 ]
Average is : 31.100000
STD.    is : 6.573516
```

**รูปที่ ผ.8** ลักษณะการจัดเรียงข้อมูลที่ได้จากไฟล์ summaryt.txt

### จ.3.2 การแก้ปัญหาการวางผังโรงงานด้วยฮิวริสติก

ตามที่ได้กล่าวมาแล้วว่าโปรแกรมทางด้านฮิวริสติกที่ใช้ในงานวิจัยนี้ประกอบไปด้วยสองโปรแกรมคือ craft และ construct ข้อแตกต่างของโปรแกรมที่หาคำตอบด้วยฮิวริสติกกับ GAs คือ ไม่จำเป็นต้องกำหนดค่าพารามิเตอร์ใดๆ ดังนั้นขั้นตอนต่างๆของการใช้งานโปรแกรมมีดังต่อไปนี้

- การกำหนดข้อมูลเบื้องต้น

แฟ้มข้อมูลที่เป็นอินพุทของโปรแกรม craft ก็คือ cflow.dat และ fflow.dat มีลักษณะเช่นเดียวกับ cost.dat และ frequency.dat แต่แตกต่างกันที่ fflow.dat ไม่จำเป็นที่จะต้องระบุขนาดของผังโรงงานในตัวไฟล์ โปรแกรมจะให้ผู้ใช้ป้อนข้อมูลของขนาดเอง

แฟ้มข้อมูลที่เป็นอินพุทของโปรแกรม construct ก็คือ cost.dat และ frequency.dat มีลักษณะการป้อนข้อมูลเช่นเดียวกับโปรแกรม

- การใช้งานโปรแกรมและการดูคำตอบ

การใช้งานโปรแกรมและการดูคำตอบแบ่งออกเป็นสองส่วนดังนี้

- การใช้งานโปรแกรม craft

เรียกใช้งานโปรแกรมโดยจะต้องมีแฟ้ม fflow.dat และ cflow.dat อยู่ในไดเร็คตอรี่เดียวกัน

```
C:\quantity\craft
Number of seed  :1
Number of Row   :2
Number of Column        :5
```

**รูปที่ จ9** การเรียกใช้งานและการป้อนขนาดของผังโรงงานให้กับโปรแกรม craft

หลังจากเรียกใช้งานโปรแกรมจำเป็นต้อง ป้อนค่า Seed Random Number และขนาดของผังที่ต้องการ หลังจากที่ป้อนข้อมูลเสร็จเรียบร้อยแล้วโปรแกรมจะแสดงวิธีการคำนวณให้ดู ดังตัวอย่างต่อไปนี้

```
Number of Iteration 0
Swap 0 1
7 1 4 0 9 8 2 5 6 3 is 399.827 diff is –27.2577
swap 0 2
4 7 1 0 9 8 2 5 6 3 is 431.768 diff is 4.68375
.
.
swap 7 8
2 3 6 5 0 9 1 8 4 7 is 305.308 diff is 22.8913
swap 7 9
2 3 6 5 0 9 7 1 4 8 is 318.993 diff is 36.5767
swap 8 9
2 3 6 5 0 9 7 8 1 4 is 316.206 diff is 33.7896
summary data
Number of total iteration is 8
Result is
2 3 6 5 0 9 7 8 4 1 is 282.416
total used time 3.13 seconds
```

**รูปที่ จ10** การแสดงผลของโปรแกรม craft ในแต่ละขั้นตอนและคำตอบ

โปรแกรมจะแสดงผลจำนวน Iteration ในการคำนวณตามวิธีการของ SDPI,ตำแหน่งของผังเมื่อทำการสลับตำแหน่ง ค่าใช้จ่ายของผัง รวมทั้งความแตกต่างของผังก่อนการสลับตำแหน่งและหลังการสลับตำแหน่ง, สตริงคำตอบรวมถึงค่าใช้จ่ายและเวลาที่ใช้ในการคำนวณ

- การใช้งานโปรแกรม construct

เรียกใช้งานโปรแกรมโดยจะต้องมีแฟ้ม frequency.dat และ cost.dat อยู่ในไดเร็คตอรี่เดียวกัน

```
C:\quantity\construct
Number of seed  :11
```

**รูปที่ จ11** การเรียกใช้งานและการป้อนขนาดของผังโรงงานให้กับโปรแกรม construct

หลังจากเรียกใช้งานโปรแกรมจำเป็นต้อง ป้อนค่า Seed Random Number ดังรูปที่ จ11 โปรแกรมจะไม่ถามขนาดของผัง แต่ผู้ใช้จะต้องกำหนดเองในแฟ้ม frequency.dat หลังจากที่ป้อนข้อมูลเสร็จเรียบร้อยแล้วโปรแกรมจะแสดงวิธีการคำนวณให้ดู ดังตัวอย่างต่อไปนี้

```
Consturct version 1
Last update 23/10/97
Row             = 2
Coulumn         = 5
Size            = 10
Station
[4 8 6 9 1 3 5 0 7 2]
State [1]
Cost [1]  = 3.000000
Cost [2]  = 6.000000
Cost [3]  = 9.000000
Cost [4]  = 12.000000
Cost [5]  = 3.000000
Cost [6]  = 4.4.242640
Cost [7]  = 6.708204
Cost [8]  = 9.486833
Cost [9]  = 12.369317
```

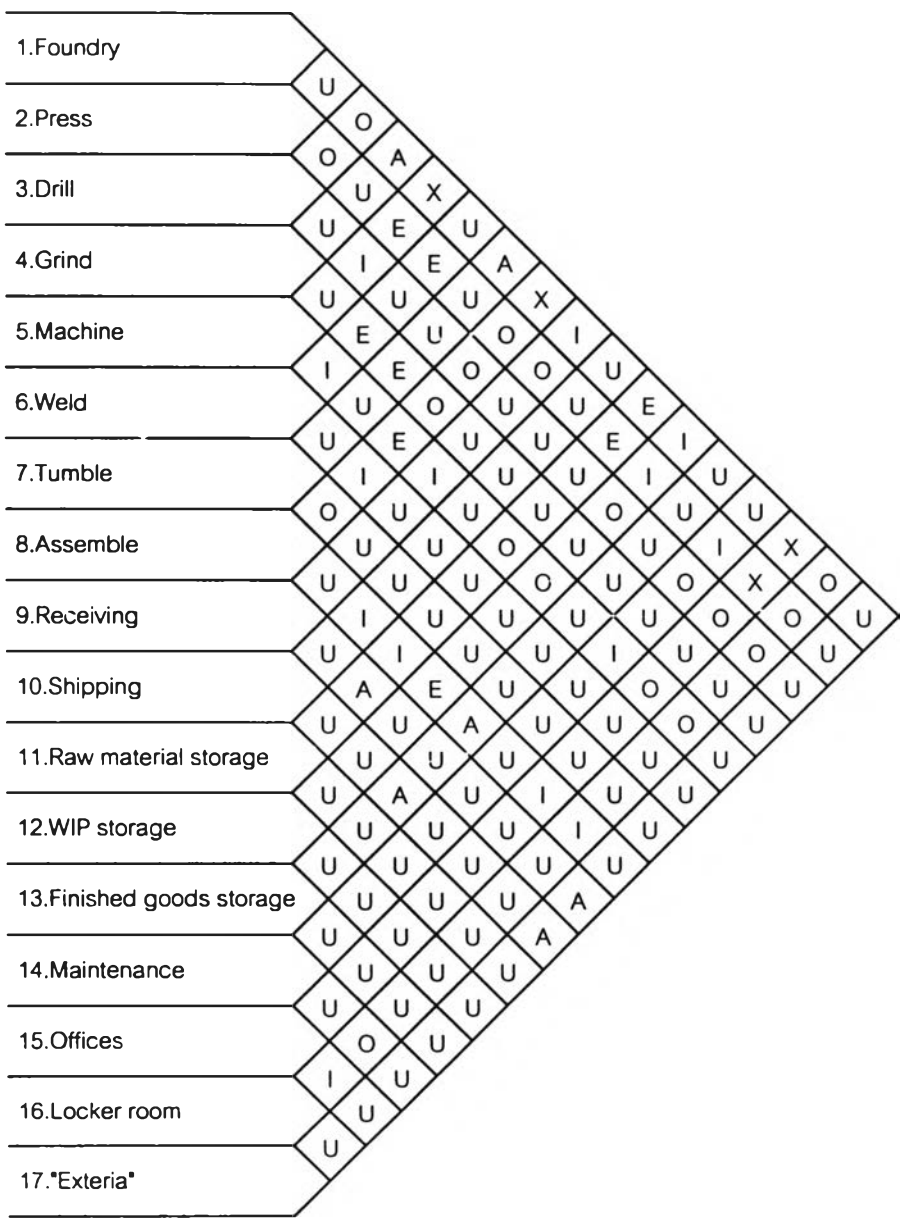
```
Best Position is 1


State [2]
Cost [2]  = 32.000000
Cost [3]  = 48.000000
Cost [4]  = 64.000000
Cost [5]  = 20.000000
Cost [6]  = 24.384777
Cost [7]  = 36.311523
Cost [8]  = 50.817814
Cost [9]  = 66.087212
Best Position is 5
.
.
.
State [9]
Cost [4]  = 445.608093
Best Position is 4
New station is
[ 4 8 1 5 2 6 9 3 0 7 ]
Function successfully used time = 0.39 sec.
```

**รูปที่ จ11** การแสดงผลจากโปรแกรม construct

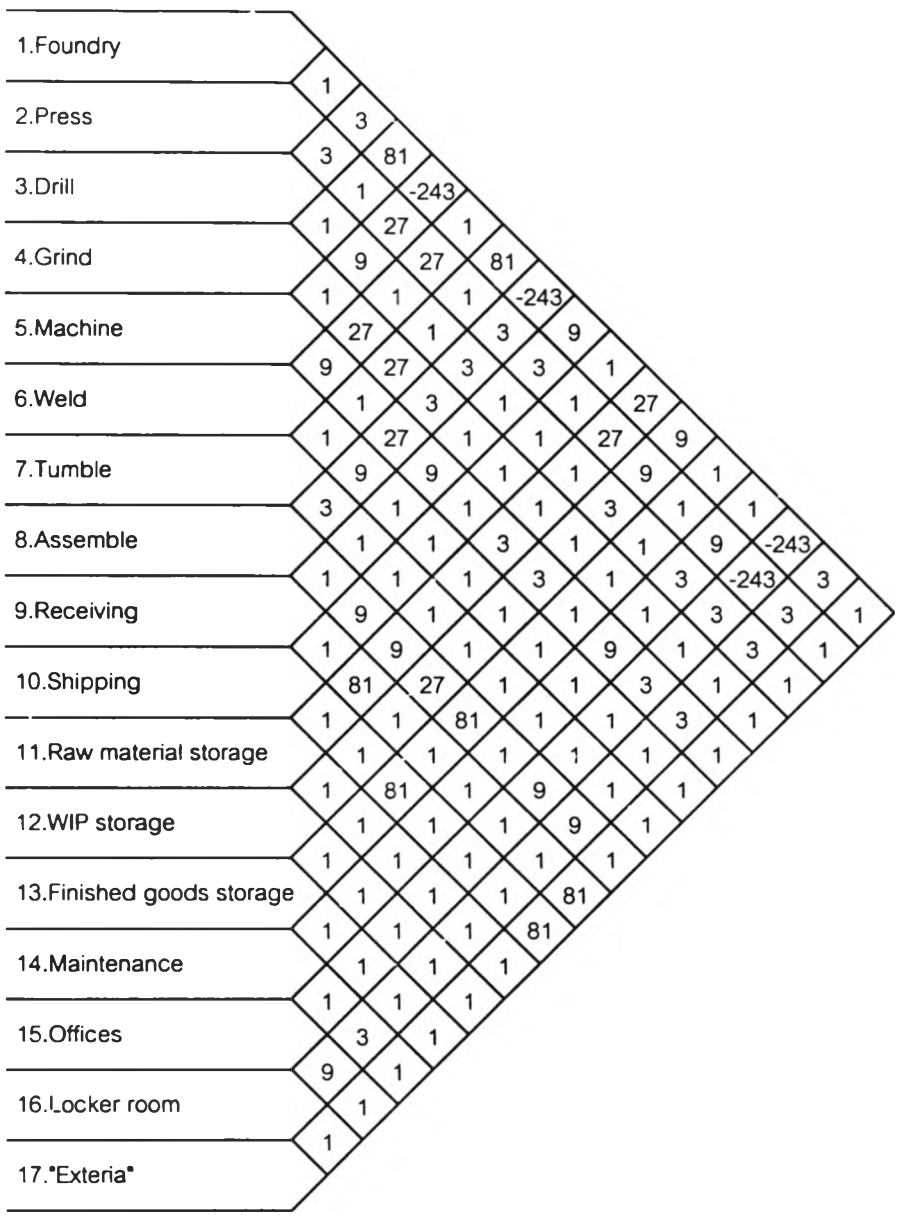
จากรูปที่ จ11 แสดงถึงผลและคำตอบที่ได้จากโปรแกรม construct ซึ่งประกอบด้วย State [ ] หมายถึงสถานี การจัดเรียงสถานีจากสตริงที่ทำการสุ่ม, Cost [ ] คือค่าใช้จ่ายของการวางสถานีลงในตำแหน่งนั้นๆ , Best Position คือตำแหน่งที่ดีที่สุดเมื่อวางสถานีนั้นๆลงในตำแหน่งนั้นๆ โปรแกรมจะสรุปคำตอบไว้ที่ตอนท้ายของการทำงาน โดยที่สามารถดูค่าใช้จ่ายทั้งหมดได้จาก State [ ] สุดท้ายของการคำนวณ และเวลาในการคำนวณทั้งหมดไว้ด้วย

# ภาคผนวก ฉ

ปัญหาตัวอย่างที่ใช้ในการทดลองขนาด 20 แผนก (Fransis, 1992) ซึ่งมีรายละเอียดดังแผนภูมิแสดงความสัมพันธ์ดังรูปที่ ฉ.1 และแผนภูมิแสดงความสัมพันธ์ของระดับคะแนนแบบเอ็กซ์โปเนนเชียลดังรูปที่ ฉ.2




**รูปที่ ฉ.1** แผนภูมิแสดงความสัมพันธ์ของผังโรงงานปัญหาตัวอย่างขนาด 20 แผนก




**รูปที่ ฉ.2** แผนภูมิแสดงความสัมพันธ์ของโรงงานปัญหาตัวอย่างโดยให้ระดับคะแนนแบบเอ็กซ์โปเนนเชียล

# ภาคผนวก ช
# รายละเอียดไฟล์ตัวหนังสือของโปรแกรม Quadist

เนื้อหารายละเอียดในส่วนนี้จะเสนอถึงรายละเอียดไฟล์ตัวหนังสือของโปรแกรม Quadist หรือโปรแกรมการแก้ปัญหาการจัดผังโรงงานโดยพิจารณาถึงข้อมูลเชิงคุณภาพและระยะทางระหว่างสถานี (Total Cloness Rating with Distance Between Station) ด้วย GAs โดยประกอบไปด้วยไฟล์จำนวนทั้งสิ้น 9 ไฟล์ ได้แก่

1. Genetic.cpp
2. CostSummary.cpp
3. CrossPMX.cpp
4. CrossOX.cpp
5. CrossCX.cpp
6. Distance.cpp
7. Elitist.cpp
8. Mutation.cpp
9. Reproduct.cpp

ดังมีรายละเอียดดังต่อไปนี้

```
////////////////////////////////////////////////////////////////////////////////
// File genetic.cpp
// Last update 31/05/97
////////////////////////////////////////////////////////////////////////////////
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

extern unsigned char *Chromosome;
extern signed int    *MemVij;
extern int            Col,Row,size;

int RandomNumber(int interval);
void RandomToPlantLayout(void);
float Evaluate(void);
float Dij(int i,int j);

int RandomNumber(int interval) // interval of random value (0-interval)
{
       return rand()%interval;
}

void RandomToPlantLayout(void)
{
       int temp;
       int attrib;
       int i;

       for(i=0;i<size;i++)
       {
              if(i == 0)
              {
```

```
                        temp=RandomNumber(size);
                        Chromosome[i] = (char)temp;
                }
                else
                {
                        temp=RandomNumber(size);
                        for(attrib=0;attrib<i;attrib++)
                        {
                                if(Chromosome[attrib] == (unsigned char)temp)
                                {
                                        temp=RandomNumber(size);
                                        attrib = -1;
                                }
                        if(Chromosome[attrib] != (unsigned char)temp && attrib == i-1)
                                break;
                        }
                        Chromosome[attrib+1] = (unsigned char)temp;
                }
        }
}

float Evaluate(void)
{
        int   i,j,N=0;
        float cost;
        cost=(float)0;
        for(i=0;i<size;i++)
        {
                for(j=i+1;j<size;j++)
                {
                        cost+=(Dij(i,j)*(float)MemVij[N]);
                        N++;
                }
        }
        return cost;
}

float Dij(int i,int j)
{
        int x1,x2,y1,y2;
        int N,M;

        for(N=0;N<size;N++)
                if(Chromosome[N]==i) break;
        for(M=0;M<size;M++)
                if(Chromosome[M]==j) break;
        x1 = N/Col;
        y1 = N%Col;
        x2 = M/Col;
        y2 = M%Col;
        return (float)sqrt(pow((x1-x2),2)+pow((y1-y2),2));
}

////////////////////////////////////////////////////////////////////////////////
// File CostSummary.cpp
// Last update 31/05/97
////////////////////////////////////////////////////////////////////////////////
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <time.h>

extern unsigned char   *ChromosomeBuffer;    // Buffer for String
extern int             population,size;
extern float           *CostBuffer;
extern int             generation;

float  CostMin(void);
float  CostMax(void);
float  CostAvg(void);
float  CostSum(void);
double STD(void);

float CostSum(void)
{
        int   i;
```

```
        float sum;
        for(i=0;i<population;i++)
                sum+=CostBuffer[i];
        return sum;
}

float CostMin(void)
{
        int    i;
        float  min;
        min=CostBuffer[0];
        for(i=0;i<population;i++)
                if(min > CostBuffer[i])
                        min=CostBuffer[i];
        return min;
}

float CostMax(void)
{
        int    i,j;
        float  max;
        max=CostBuffer[0];
        for(i=0;i<population;i++)
                if(max < CostBuffer[i])
                {
                        max=CostBuffer[i];
                        j=i;
                }
        return max;
}

float CostAvg(void)
{
        return CostSum()/(float)population;
}

double STD(void)
{
        int i;
        double avg;
        double std=(double)0;
        avg=CostAvg();
        for(i=0;i<population;i++)
                std+=pow((double)(CostBuffer[i]-avg),(double)2);
        return sqrt((double)std/(double)(population-1));
}

////////////////////////////////////////////////////////////////////////////////
// File CrossPMX.cpp
// Last update 31/05/97
////////////////////////////////////////////////////////////////////////////////
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>

// Function are not is this file
void CopyItem(unsigned char *Soure,int soure,unsigned char *Destination,int
destination);
void UpdateCost(void);

extern unsigned char *ChromosomeBuffer;      // Buffer for String
extern int                    population;
extern int                    size;
extern FILE *file;

// Function are in this file
void AllocRanDomProbPMX(void);
void AllocIndexProbPMX(void);
void CrossOver_PMX(float Prob_PMX);
void CrossPMX(int first,int second);
void Map(unsigned char *Temp1,int i,unsigned char *Temp2,int j);
void ShowValuePMX(void);
void AllocParentPMX(int first,int second);
void DefineInterval(void);
void SelectProbPMX(float prob);
```

```
float  *RanDomProb;
int        *IndexProb;
int        count;
unsigned char *Parent1 ,*Parent2,temp;
int        interval1,interval2;

void AllocRanDomProbPMX(void)
{
        int i;
        RanDomProb = (float *)malloc(sizeof(float)*population);
        if(RanDomProb == NULL)
        {
                printf("Not enough memmory for RanDomProb...\n");
                exit(0);
        }
//      srand( (unsigned)time( NULL ) );
        for(i=0;i<population;i++)
        {
                RanDomProb[i]=(float)(rand()*rand()%1000000)/(float)1000000;
//              fprintf(file,"%f\t",RanDomProb[i]);
        }
}

void AllocIndexProbPMX(void)
{
        IndexProb=(int *)malloc(sizeof(int)*(count+1));
        if(IndexProb == NULL)
        {
                printf("Not enough memmory for IndesPMX...\n");
                exit(0);
        }
}

void Map(unsigned char *Temp1,int i,unsigned char *Temp2,int j)
{
        Temp1[i]=Temp2[j];
}

void ShowValuePMX(void)
{
        int i;

        fprintf(file,"[ ");
        for(i=0;i<size;i++)
        {
                if(i== interval1 || i == interval2+1)
                        fprintf(file,"| ");
                fprintf(file,"%d ",Parent1[i]);
        }
        fprintf(file,"]\n");
        fprintf(file,"[ ");
        for(i=0;i<size;i++)
        {
                if(i == interval1 || i == interval2+1)
                        fprintf(file,"| ");
                fprintf(file,"%d ",Parent2[i]);
        }
        fprintf(file,"]\n\n");
}

void AllocParentPMX(int first,int second)
{
        Parent1=(unsigned char *)malloc(sizeof(char)*size);
        if(Parent1 == NULL)
        {
                printf("Not enough memmory for Parent1...\n");
                exit(0);
        }
        Parent2=(unsigned char *)malloc(sizeof(char)*size);
        if(Parent2 ==NULL)
        {
                printf("Not enough memmory for Parent2...\n");
                exit(0);
        }
        CopyItem(ChromosomeBuffer,first,Parent1,0);
        CopyItem(ChromosomeBuffer,second,Parent2,0);
}
```

```
void DefineInterval(void)
{
        do
                interval1=rand()%size;
        while(interval1 < 1 || interval1 >= size-2);
//      fprintf(file,"Interval 1 = %d\n",interval1);
        do
                interval2=rand()%size;
        while(interval2 <= interval1 || interval2 > size-2);
//      fprintf(file,"Interval 2 = %d\n",interval2);
}

void CrossPMX(int first,int second)
{
        int  i,j;
        AllocParentPMX(first,second);
        DefineInterval();
//      ShowValuePMX();
        for(i=interval1;i<=interval2;i++)
        {
                temp=Parent1[i];
                Parent1[i]=Parent2[i];
                Parent2[i]=temp;
        }
        for(i=0;i<interval1;i++)
        {
                for(j=interval1;j<=interval2;j++)
                {
                        if(Parent1[i]==Parent1[j])
                        {
                                Map(Parent1,i,Parent2,j);
                                j=interval1-1;
                        }
                }
                for(j=interval1;j<=interval2;j++)
                {
                        if(Parent2[i]==Parent2[j])
                        {
                                Map(Parent2,i,Parent1,j);
                                j=interval1-1;
                        }
                }
        }
        for(i=interval2+1;i<size;i++)
        {
                for(j=interval1;j<=interval2;j++)
                {
                        if(Parent1[i]==Parent1[j])
                        {
                                Map(Parent1,i,Parent2,j);
                                j=interval1-1;
                        }
                }
                for(j=interval1;j<=interval2;j++)
                {
                        if(Parent2[i]==Parent2[j])
                        {
                                Map(Parent2,i,Parent1,j);
                                j=interval1-1;
                        }
                }
        }
        CopyItem(Parent1,0,ChromosomeBuffer,first);
        CopyItem(Parent2,0,ChromosomeBuffer,second);
//      ShowValuePMX();
        free(Parent1);
        free(Parent2);
}

void SelectProbPMX(float prob)
{
        int i,j;
//      fprintf(file,"Select .....\n");
        j=0;
        for(i=0;i<population;i++)
        {
```

```
                if(RanDomProb[i] <= prob)
                {
                        IndexProb[j]=i;
                        j++;
//                      if(i%6 == 5) fprintf(file,"\n");fprintf(file,"%d\t",i);
                }
        }
}

void CrossOver_PMX(float Prob_PMX)
{
        int       i;
        if(Prob_PMX == 0) return;
        AllocRanDomProbPMX();
        count=0;
        for(i=0;i<population;i++)
                if(RanDomProb[i] <= Prob_PMX) count++;
//      fprintf(file,"\nCount = %d\n",count);
        AllocIndexProbPMX();
        SelectProbPMX(Prob_PMX);
//      fprintf(file,"\n");
        if(count%2 == 1) count--;
        for(i=0;i<count;i+=2)
        {
//              fprintf(file,"[%d %d]\n",IndexProb[i],IndexProb[i+1]);
                CrossPMX(IndexProb[i],IndexProb[i+1]);
        }
        UpdateCost();
        free(IndexProb);
        free(RanDomProb);
}

////////////////////////////////////////////////////////////////////////////////
// File CrossOX.cpp
// Last update 31/05/97
////////////////////////////////////////////////////////////////////////////////
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>

extern unsigned char *ChromosomeBuffer;      // Buffer for String
extern int size;
extern int population;
extern FILE *file;

void CopyItem(unsigned char *Soure,int soure,unsigned char *Destination,int
destination);
void UpdateCost(void);

unsigned char *ParentOX1,*ParentOX2;
int interval1OX,interval2OX;
float  *RanDomProbOX;
int    *IndexProbOX;
int    CountOX;
char   hole='*';

unsigned char *Temp1,*Temp2,*TempF,*TempM;
void ShowValueOX(void);
void CrossOX(void);
void AllocRanDomProbOX();
void DefineIntervalOX(void);
void AllocParentOX(int first,int second);
void SelectProbOX(float prob);
void AllocIndexProbOX();

void ShowValueOX(void)
{
        int i;
        fprintf(file,"[ ");
        for(i=0;i<size;i++)
        {
                if(i== interval1OX || i == interval2OX+1)
                        fprintf(file,"| ");
                if(ParentOX1[i] == hole) fprintf(file,"# ");
                else fprintf(file,"%d ",ParentOX1[i]);
        }
```

```
        fprintf(file,"]\n");
        fprintf(file,"[ ");
        for(i=0;i<size;i++)
        {
                if(i == interval1OX || i == interval2OX+1)
                        fprintf(file,"| ");
                if(ParentOX2[i] == hole) fprintf(file,"# ");
                else fprintf(file,"%d ",ParentOX2[i]);
        }
        fprintf(file,"]\n");
}

void AllocRanDomProbOX(void)
{
        int i;
        RanDomProbOX = (float *)malloc(sizeof(float)*population);
        if(RanDomProbOX == NULL)
        {
                printf("Not enough memmory for RanDomProbOX...\n");
                exit(0);
        }
//      srand( (unsigned)time( NULL ) );
        for(i=0;i<population;i++)
        {
                RanDomProbOX[i]=(float)(rand()*rand()%1000000)/(float)1000000;
//              if(i == 5) fprintf(file,"\n");
//              fprintf(file,"%f\t",RanDomProbOX[i]);
        }
}

void AllocIndexProbOX(void)
{
        IndexProbOX=(int *)malloc(sizeof(int)*(CountOX+1));
        if(IndexProbOX == NULL)
        {
                printf("Not enough memmory for IndesOX...\n");
                exit(0);
        }
}

void AllocParentOX(int first,int second)
{
        ParentOX1=(unsigned char *)malloc(sizeof(char)*size);
        if(ParentOX1 == NULL)
        {
                printf("Not enough memmory for ParentOX1...\n");
                exit(0);
        }
        ParentOX2=(unsigned char *)malloc(sizeof(char)*size);
        if(ParentOX2 ==NULL)
        {
                printf("Not enough memmory for ParentOX2...\n");
                exit(0);
        }

        CopyItem(ChromosomeBuffer,first,ParentOX1,0);
        CopyItem(ChromosomeBuffer,second,ParentOX2,0);
}

void DefineIntervalOX(void)
{
        do
                interval1OX=rand()%size;
        while(interval1OX < 1 || interval1OX >= size-2);
//      fprintf(file,"IntervalOX 1 = %d\n",interval1OX);
        do
                interval2OX=rand()%size;
        while(interval2OX <= interval1OX || interval2OX > size-2);
//      fprintf(file,"IntervalOX 2 = %d\n",interval2OX);
}

void SelectProbOX(float prob)
{
        int i,j;
//      fprintf(file,"\nSelect\n");
        j=0;
        for(i=0;i<population;i++)
```

```
        {
                if(RanDomProbOX[i] <= prob)
                {
                        IndexProbOX[j]=i;
                        j++;
//                      if(i%6 ==5) fprintf(file,"\n");
//                      fprintf(file,"%d\t",i);
                }
        }
//      fprintf(file,"\n"),
}

void CrossOX(int first,int second)
{
        int i,j;
        AllocParentOX(first,second);
        DefineIntervalOX();
//      ShowValueOX();
//      fprintf(file,"-------------------------\n");
        for(i=0;i<size;i++)
        {
                TempF[i]=Temp1[i]=ParentOX1[i];
                TempM[i]=Temp2[i]=ParentOX2[i];
        }
        for(i=0;i<size;i++)
        {
                for(j=interval1OX;j<=interval2OX;j++)
                if(ParentOX1[i]==Temp2[j]) ParentOX1[i]=hole;        //  Hole = 0xff
                for(j=interval1OX;j<=interval2OX;j++)
                if(ParentOX2[i]==Temp1[j]) ParentOX2[i]=hole;        //  Hole = 0xff
        }
//      ShowValueOX();
//      fprintf(file,"-------------------------\n");
        i=0;
        for(j=interval2OX+1;j<size;j++)
        {
                Temp1[i]=ParentOX1[j];
                Temp2[i]=ParentOX2[j];
                i++;
        }
        for(j=0;j<=interval2OX;j++)
        {
                Temp1[i]=ParentOX1[j];
                Temp2[i]=ParentOX2[j];
                i++;
        }
        for(i=0;i<size;i++)
        {
                if(Temp1[i] == hole)
                {
                        for(j=i;j<size;j++)
                        {
                                if(Temp1[j] != hole) break;
                        }

                        if(j!=size)
                        {
                                Temp1[i]=Temp1[j];
                                Temp1[j]=hole;
                        }
                        else Temp1[i]=Temp1[j];
                }
        }
        for(i=0;i<size;i++)
        {
                if(Temp2[i] == hole)
                {
                        for(j=i;j<size;j++)
                        {
                                if(Temp2[j] != hole) break;
                        }
                        if(j!=size)
                        {
                                Temp2[i]=Temp2[j];
                                Temp2[j]=hole;
                        }
                        else Temp2[i]=Temp2[j];
```

```
                }
        }
        i=0;
        for(j=interval2OX+1;j<size;j++)
        {
                ParentOX1[j]=Temp1[i];
                ParentOX2[j]=Temp2[i];
                i++;
        }
        for(j=0;j<=interval2OX;j++)
        {
                ParentOX1[j]=Temp1[i];
                ParentOX2[j]=Temp2[i];
                i++;
        }
        for(i=interval1OX;i<=interval2OX;i++)
        {
                ParentOX1[i]=TempM[i];
                ParentOX2[i]=TempF[i];
        }
        CopyItem(ParentOX1,0,ChromosomeBuffer,first);
        CopyItem(ParentOX2,0,ChromosomeBuffer,second);
//      ShowValueOX();
        free(ParentOX1);
        free(ParentOX2);
}

void CrossOver_OX(float Prob_OX)
{
        int i;
        unsigned char eos1,eos2;
        if(Prob_OX == 0) return;
        Temp1=(unsigned char *)malloc(sizeof(char)*size);
        Temp2=(unsigned char *)malloc(sizeof(char)*size);
        TempF=(unsigned char *)malloc(sizeof(char)*size);
        TempM=(unsigned char *)malloc(sizeof(char)*size);
        eos1=Temp1[size];
        eos2=Temp2[size];
        Temp1[size]=hole;
        Temp2[size]=hole;
        AllocRanDomProbOX();
        CountOX=0;
        for(i=0;i<population;i++)
                if(RanDomProbOX[i] <= Prob_OX) CountOX++;
//      fprintf(file,"CountOX = %d\n",CountOX);
        AllocIndexProbOX();
        SelectProbOX(Prob_OX);
//      fprintf(file,"\n");
        if(CountOX%2 == 1) CountOX--;
        for(i=0;i<CountOX;i+=2)
        {
//              fprintf(file,"[%d %d]\n",IndexProbOX[i],IndexProbOX[i+1]);
                CrossOX(IndexProbOX[i],IndexProbOX[i+1]);
        }
//      fprintf(file,"\n");
        Temp1[size]=eos1;
        Temp2[size]=eos2;
        UpdateCost();
        free(Temp1);
        free(Temp2);
        free(TempF);
        free(TempM);
        free(IndexProbOX);
        free(RanDomProbOX);
}

////////////////////////////////////////////////////////////////////////////////
// File CrossCX.cpp
// Last update 31/05/97
////////////////////////////////////////////////////////////////////////////////
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>
void CopyItem(unsigned char *Soure,int soure,unsigned char *Destination,int
destination);
void UpdateCost(void);
```

```
extern unsigned char *ChromosomeBuffer;      // Buffer for String
extern int                    population;
extern int                    size;
extern FILE *file;
void AllocParentCX(int first,int second);
void AllocRanDomProbCX(void);
void SelectProbCX(float prob);
void AllocIndexProbCX(void);
void ShowValueCX(void);
unsigned char *ParentCX1,*ParentCX2;
float  *RanDomProbCX;
int    *IndexProbCX;
int    CountCX;

void ShowValueCX(void)
{
       int i;
       for(i=0;i<size;i++)
              fprintf(file,"%d ",ParentCX1[i]);
       fprintf(file,"\n");
       for(i=0;i<size;i++)
              fprintf(file,"%d ",ParentCX2[i]);
       fprintf(file,"\n");
}

void AllocRanDomProbCX(void)
{
       int i;
       RanDomProbCX = (float *)malloc(sizeof(float)*population);
       if(RanDomProbCX == NULL)
       {
              printf("Not enough memmory for RanDomProbOX...\n");
              exit(0);
       }
//     srand( (unsigned)time( NULL ) );
       for(i=0;i<population;i++)
       {
              RanDomProbCX[i]=(float)(rand()*rand()%1000000)/(float)1000000;
//            if(i%6 == 5) fprintf(file,"\n");
//            fprintf(file,"%f\t",RanDomProbCX[i]);
       }
}
void AllocIndexProbCX(void)
{
       IndexProbCX=(int *)malloc(sizeof(int)*(CountCX+1));
       if(IndexProbCX == NULL)
       {
              printf("Not enough memmory for IndesCX...\n");
              exit(0);
       }
}

void AllocParentCX(int first,int second)
{
       ParentCX1=(unsigned char *)malloc(sizeof(char)*size);
       if(ParentCX1 == NULL)
       {
              printf("Not enough memmory for ParentCX1...\n");
              exit(0);
       }
       ParentCX2=(unsigned char *)malloc(sizeof(char)*size);
       if(ParentCX2 ==NULL)
       {
              printf("Not enough memmory for ParentCX2...\n");
              exit(0);
       }
       CopyItem(ChromosomeBuffer,first,ParentCX1,0);
       CopyItem(ChromosomeBuffer,second,ParentCX2,0);
}

void SelectProbCX(float prob)
{
       int i,j;
//     fprintf(file,"Select\n");
       j=0;
       for(i=0;i<population;i++)
       {
```

```
                if(RanDomProbCX[i] <= prob)
                {
                        IndexProbCX[j]=i;
                        j++;
//                      if(i==5) fprintf(file,"\n");
//                      fprintf(file,"%d\t",i);
                }
        }
//      fprintf(file,"\n\n");
}

void CrossCX(int first,int second)
{
        AllocParentCX(first,second);
        int i,mark=0;
        unsigned char first_value,swap_tmp,temp;
//      ShowValueCX();
        temp=ParentCX2[0];
        first_value=ParentCX1[0];
        for(i=0;i<size;i++)
        {
                if(ParentCX1[i]==temp && i != mark)
                {
                        swap_tmp=ParentCX1[i];
                        ParentCX1[i]=ParentCX2[i];
                        ParentCX2[i]=swap_tmp;
                        temp=ParentCX1[i];
                        mark=i;
                        i = -1;
                }
        }
        for(i=0;i<size;i++)
        {
                swap_tmp=ParentCX1[i];
                ParentCX1[i]=ParentCX2[i];
                ParentCX2[i]=swap_tmp;
        }
//      ShowValueCX();fprintf(file,"\n");
        CopyItem(ParentCX1,0,ChromosomeBuffer,first);
        CopyItem(ParentCX2,0,ChromosomeBuffer,second);
}

void CrossOver_CX(float Prob_CX)
{
        int i;
        if(Prob_CX == 0) return;
        CountCX=0;
        AllocRanDomProbCX();
        for(i=0;i<population;i++)
                if(RanDomProbCX[i] <= Prob_CX) CountCX++;
//      fprintf(file,"CountCX = %d\n",CountCX);
        AllocIndexProbCX();
        SelectProbCX(Prob_CX);
//      fprintf(file,"\n");
        if(CountCX%2 == 1) CountCX--;
        for(i=0;i<CountCX;i+=2)
        {
//              fprintf(file,"[%d %d]\n",IndexProbCX[i],IndexProbCX[i+1]);
                CrossCX(IndexProbCX[i],IndexProbCX[i+1]);
        }
//      fprintf(file,"\n");
        UpdateCost();
}

////////////////////////////////////////////////////////////////////////////////
// File Distance.cpp
// Last update 31/05/97
////////////////////////////////////////////////////////////////////////////////
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <malloc.h>
#include <string.h>
#include <time.h>

unsigned char *Chromosome,*ChromosomeBuffer,*ChromosomeTemp,*BestChromosome; // Buffer
for String
```

```
signed int    *MemVij;                        // Buffer for store Vij
float            Prob_PMX,Prob_OX,Prob_CX,Prob_M,mutat;
int              Row,Col,size;
float           *Cost,*CostBuffer,*CostTemp,BestCost;
int              population;
int              generation;
int              mark[20];                    //used for check number of first string
int              interval,total;              //used for check interval
FILE            *file,*summary;

// Functions are not in this file.
void RandomToPlantLayout(void);
float Evaluate(void);
float CostMin(void);
float CostMax(void);
float CostAvg(void);
float CostSum(void);
double STD(void);
void  CostReproduct(void);
void  CrossOver_PMX(float Prob_PMX);
void  CrossOver_OX(float Prob_OX);
void  CrossOver_CX(float Prob_CX);
void  Mutate(float mutate);
void  Elitist(void);

// Functions are in this file.
int TotalWeight(int N);
void AllocDij(void);
void AllocVij(void);
void initial(void);
void AllocCostBuffer(void);
void AllocChromosome(void);
void SavePlantLayout(int pop);
void SaveToFile(int Generation);
void ShowChromosome(void);
void SaveBest(void);
void UpdateCost(void);
void SummaryToFile(int gen);
int  NeverMark(int end,int key);

int  NeverMark(int end,int key)
{
        int i;
        for(i=0;i<end;i++)
                if(mark[i] == key) return 0;
        return 1;
}

void SummaryToFile(int gen)
{
        int i,j,k;
        float min;
        fprintf(summary,"[Generation %d]\n",gen);
        min=CostMax();
        for(i=0;i<total;i++)
        {
                for(k=0;k<population;k++)
                {
                        if(CostBuffer[k]<=min && NeverMark(i,k))
                        {
                                min=CostBuffer[k];
                                mark[i]=k;
                        }
                }
                fprintf(summary,"[%d]\t",mark[i]);
                for(j=0;j<size;j++)
                        fprintf(summary,"%d ",ChromosomeBuffer[size*mark[i]+j]);
                fprintf(summary," =\t%f\n",CostBuffer[mark[i]]);
                min=CostMax();
        }
        fprintf(summary,"\nThe Best Cost is : %f\t",BestCost);
        fprintf(summary,"[ ");
        for(i=0;i<size;i++)
                fprintf(summary,"%d ",BestChromosome[i]);
        fprintf(summary,"]\n");
        fprintf(summary,"Averang is : %f\n",CostAvg());
        fprintf(summary,"STD     is : %f\n\n",STD());
```

```
}

void UpdateCost(void)
{
        int i,j;
        for(i=0;i<population;i++)
        {
                for(j=0;j<size;j++)
                        Chromosome[j]=ChromosomeBuffer[i*size+j];
                CostBuffer[i]=Evaluate();
        }
}

void ShowChromosome(void)
{
        int i;
/*
        for(i=0;i<population;i++)
        {
                fprintf(file,"[%d]\t",i);
                for(j=0;j<size;j++)
                        fprintf(file,"%d ",ChromosomeBuffer[size*i+j]);
                fprintf(file," =\t%f\n",CostBuffer[i]);
        }
*/
        fprintf(file,"%f\t",CostMin());
        fprintf(file,"%f\t",CostMax());
        fprintf(file,"%f\t",CostAvg());
        fprintf(file,"%f\t",CostSum());
        fprintf(file,"%f\t",STD());
//      fprintf(file,"The Best Cost is = %f\n",BestCost);
        fprintf(file,"[ ");
        for(i=0;i<size;i++)
                fprintf(file,"%d ",BestChromosome[i]);
        fprintf(file,"]\n");
}

int TotalWeight(int N) // Formular N=N-1+N-2+N-3...+1
{
        if(N > 1)
                return N+TotalWeight(N-1);
        else return 1;
}

void AllocDij(void)    // Allocate memory for construct string
{
        Chromosome = (unsigned char *)malloc(size*sizeof(char));
        if(Chromosome == NULL)
        {
                printf("Not enough memmory for \"Chromosome\"\n");
                exit(0);
        }
}

// Allocate memmory for weight table and input weight in term continue
// Input : None
// Output: Enter value in pointer MemVij
void AllocVij(void)
{
        FILE *input;
        char buffer[5];
        int  weight,i;
        input=fopen("input.dat","rt");
        if(input == NULL)
        {
                printf("Can not open file input.dat\n");
                exit(0);
        }
        fgets(buffer,5,input);
        sscanf(buffer,"%d",&Row);      // Get ROW
        fgets(buffer,5,input);
        sscanf(buffer,"%d",&Col);      // Get COL
        size=Row*Col;
        weight = TotalWeight(size-1);                  // Find total of weight
        printf("weight = %d\n",weight);
        MemVij = (int *)malloc(weight*sizeof(weight));// Alloc memmory for input weight
        if(MemVij == NULL)
```

```
        {
                printf("Not enough memmory for \"MemVij\"\n");
                exit(0);
        }
        for(i=0;i<weight;i++)
        {
                fgets(buffer,15,input);
                sscanf(buffer,"%d",&MemVij[i]);
        }
        printf("MemVij is\n");
        for(i=0;i<weight;i++)
                printf("%d ",MemVij[i]);
        printf("\n");
        fclose(input);
}

void AllocCostBuffer(void)
{
        CostBuffer=(float *)malloc(sizeof(float)*population);
        if(CostBuffer == NULL)
        {
                printf("Not enough memmory for \"CostBuffer\"\n");
                exit(0);
        }
        CostTemp=(float *)malloc(sizeof(float)*population);
        if(CostTemp == NULL)
        {
                printf("Not enough memmory for \"CostTemp\"\n");
                exit(0);
        }
}

void AllocChromosomeBuffer(void)
{
        ChromosomeBuffer=(unsigned char *)malloc(size*sizeof(char)*population);
        if(ChromosomeBuffer == NULL)
        {
                printf("Not enough memmory for \"ChromosomeBuffer\"\n");
                exit(0);
        }
        ChromosomeTemp=(unsigned char *)malloc(size*sizeof(char)*population);
        if(ChromosomeTemp == NULL)
        {
                printf("Not enough memmory for \"ChromosomeTemp\"\n");
                exit(0);
        }
        BestChromosome=(unsigned char *)malloc(size*sizeof(char));
        if(ChromosomeTemp == NULL)
        {
                printf("Not enough memmory for \"Best Chromosome\"\n");
                exit(0);
        }
}

void SavePlantLayout(int pop)
{
        int i;
        for(i=0;i<size;i++)
                ChromosomeBuffer[pop*size+i]=Chromosome[i];
}

void SaveToFile(int Generation)
{
        fclose(file);
        fclose(summary);
}

void initial(void)
{
        int i;
        printf("Enter Generation : ");scanf("%d",&generation);
        printf("Enter Population : ");scanf("%d",&population);
        printf("Enter Probability PMX : ");scanf("%f",&Prob_PMX);
        printf("Enter Probability OX  : ");scanf("%f",&Prob_OX);
        printf("Enter Probability CX  : ");scanf("%f",&Prob_CX);
        printf("Enter Mutate : ");scanf("%f",&mutat);
        printf("How many period for report : ");scanf("%d",&interval);
```

```
        printf("How many report chromosome : ");scanf("%d",&total);
        AllocVij();             // Build relationship chart table
        AllocCostBuffer();
        AllocDij();             // Build plant layout
        AllocChromosomeBuffer();
        file=fopen("result.txt","wt");
        if(file == NULL)
        {
                printf("Can not open file \"RESULT.TXT\" \n");
                exit(0);
        }
        summary=fopen("summary.txt","wt");
        if(file == NULL)
        {
                printf("Can not open file \"SUMMARY.TXT\" \n");
                exit(0);
        }
//      srand( (unsigned)time( NULL ) );
        for(i=0;i<population;i++)
        {
                RandomToPlantLayout();
                SavePlantLayout(i);
                CostBuffer[i]=Evaluate();
        }
}

void main(void)
{
        int i;
        clock_t start, finish;
        double  use;
        printf("Program Plant Layout version 1.00\n");       // Show version
        printf("Last update 08/03/97\n\n");
        initial();
        fprintf(file,"Program Plant Layout version 1.00\n");
        fprintf(file,"Last update 08/03/97\n\n");
        start=clock();
        BestCost=CostMax();
        for(i=0;i<generation;i++)
        {
//              fprintf(file,"Generation %d\n",i);
                SaveBest();
//              ShowChromosome();
//              fprintf(file,"\n[Reproduct]\n");
                CostReproduct();
                SaveBest();
//              ShowChromosome();
//              fprintf(file,"\n[Cross Over Prob PMX]\n");
                CrossOver_PMX(Prob_PMX);
                SaveBest();
//              ShowChromosome();
//              fprintf(file,"\n[Cross Over Prob OX]\n");
                CrossOver_OX(Prob_OX);
                SaveBest();
//              ShowChromosome();
//              fprintf(file,"\n[Cross Over Prob CX]\n");
                CrossOver_CX(Prob_CX);
                SaveBest();
//              ShowChromosome();
//              fprintf(file,"\n[Mutate]\n");
                Mutate(mutat);
                SaveBest();
                Elitist();
                ShowChromosome();
                printf("Generation %d Completed\n",i);
                if(i%interval == 0) SummaryToFile(i);
        }
        SaveToFile(i);
        finish=clock();
        use=(double)(finish-start)/CLOCKS_PER_SEC;
        printf("Function sucessfully used time = %.2f sec.\n",use);
        free(BestChromosome);
        free(ChromosomeBuffer);
        free(CostBuffer);
        free(ChromosomeTemp);
        free(CostTemp);
        free(MemVij);
```

```
}

////////////////////////////////////////////////////////////////////////////////
// File Elitist.cpp
// Last update 31/05/97
////////////////////////////////////////////////////////////////////////////////
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

float  CostMin(void);
void   CopyItem(unsigned char *Soure,int soure,unsigned char *Destination,int
destination);
float  CostMax(void);
void UpdateCost(void);
void   SaveBest(void);
extern unsigned char *Chromosome,*ChromosomeBuffer,*BestChromosome; // Buffer for
String
extern float          *CostBuffer,BestCost;
extern int            population;
extern int            size;
extern FILE *file;

void SaveBest(void)
{
        int i;
        if(CostMin() < BestCost)
        {
                BestCost=CostMin();
                for(i=0;i<population;i++)
                        if(CostBuffer[i]==BestCost) break;
                CopyItem(ChromosomeBuffer,i,BestChromosome,0);
        }
}

void ReplaceBestChromosome(void)
{
        int i;
        float BadCost;
        BadCost=CostMax();
        for(i=0;i<population;i++)
                if(CostBuffer[i]==BadCost) break;
        CopyItem(BestChromosome,0,ChromosomeBuffer,i);
}

void Elitist(void)
{
        ReplaceBestChromosome();
        UpdateCost();
}

////////////////////////////////////////////////////////////////////////////////
// File Mutatiom.cpp
// Last update 31/05/97
////////////////////////////////////////////////////////////////////////////////
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>

void UpdateCost();
extern int size;
extern int population;
extern unsigned char *ChromosomeBuffer;
extern FILE *file;
void InitialMutate(void);
void RandomMutate(void);
void SelectMutate(void);
void Mutation(int position);
void ShowChromosome(int pop,int siz,int swaper);
void Mutate(float mutat);
float *Gene,mutate;
void InitialMutate(void)
{
        Gene=(float *)malloc(sizeof(float)*size*population);
        if(Gene == NULL)
        {
```

```
                printf("Not enough memory for Gene\n");
                exit(0);
        }
}

/*
void RandomChromosome(void)
{
        int Size,Population;
        for(Population=0;Population<population;Population++)
        {
                for(Size=0;Size<size;Size++)
                {
                        do
                        {
                                ChromosomeBuffer[(Population*size)+Size]=rand()%size;
                        } while(Redundant(Population,Size));
                }
        }
        for(Population=0;Population<population;Population++)
        {
                printf("Level %d \t[ ",Population);
                for(Size=0;Size<size;Size++)
                {
                        printf("%d ",ChromosomeBuffer[(Population*size)+Size]);
                }
                printf("]\n");
        }
}

int Redundant(int pop,int siz)
{
        int i;
        for(i=0;i<siz;i++)
        {
                if(ChromosomeBuffer[(pop*size)+siz] == ChromosomeBuffer[(pop*size)+i])
                        return 1;
        }
        return 0;
}
*/

void RandomMutate(void)
{
        int i;
//      srand( (unsigned)time( NULL ) );
        for(i=0;i<population*size;i++)
        {
                Gene[i]=(float)(rand()*rand()%1000000)/(float)1000000;
//              if(i%6 == 5) fprintf(file,"\n");
//              fprintf(file,"%f\t",Gene[i]);
        }
}

void SelectMutate(void)
{
        int i;

//      fprintf(file,"\nSelect Mutate\n");
        for(i=0;i<population*size;i++)
        {
                if(Gene[i] <= mutate)
                {
                        Mutation(i);
                }
        }
}

void Mutation(int position)
{
        int pop,siz;
        int swaper;
        unsigned char temp;
        pop = position/size;
        siz = position%size;
//      fprintf(file,"[Expected Mutation=%d]\tPercent mutate = %f\n   Chromosome =
%d\n   Logus = %d\n",position,Gene[position],pop,siz);
```

```
        swaper=size;
//      ShowChromosome(pop,siz,swaper);
//      srand( (unsigned)time( NULL ) );
        do
        {
                swaper=rand()%size;
        } while(swaper == siz);
//      ShowChromosome(pop,siz,swaper);
        temp=ChromosomeBuffer[(pop*size)+siz];
        ChromosomeBuffer[(pop*size)+siz] = ChromosomeBuffer[(pop*size)+swaper];
        ChromosomeBuffer[(pop*size)+swaper] = temp;
//      ShowChromosome(pop,siz,swaper);
}

void ShowChromosome(int pop,int siz,int swaper)
{
        int i;
//      fprintf(file,"Chromosome %d \t[",pop);
        for(i=0;i<size;i++)
        {
                if(i == siz || i == swaper) fprintf(file,"|");
                else fprintf(file," ");
                fprintf(file,"%d",ChromosomeBuffer[(pop*size )+i]);
                if(i == siz || i == swaper) fprintf(file,"|");
                else fprintf(file," ");
        }
        fprintf(file,"]\n");
}

void Mutate(float mutat)
{
        mutate=mutat;
        InitialMutate();
        RandomMutate();
        SelectMutate();
        UpdateCost();
        free(Gene);
}

////////////////////////////////////////////////////////////////////////////////
// File Reproduct.cpp
// Last update 31/05/97
////////////////////////////////////////////////////////////////////////////////
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <time.h>
extern unsigned char *ChromosomeBuffer,*ChromosomeTemp;     // Buffer for String
extern int           population,size;
extern float            *CostBuffer,*CostTemp;
extern FILE *file;
// Function are not in this file
float  CostSum(void);
void   UpdateCost(void);
// Function are in this file
void AllocIndex(void);
void AllocProb(void);
void AllocCumulative(void);
void CostReproduct(void);
void CopyItem(unsigned char *Soure,int soure,unsigned char *Destination,int
destination);
int             *Index;
float           *Prob,*Cumulative;

void AllocIndex(void)
{
        Index=(int *)malloc(population*sizeof(int));
        if(Index == NULL)
        {
                printf("Not enough memmory for Index...\n");
                exit(0);
        }
}

void AllocProb(void)
{
```

```
        Prob=(float *)malloc(population*sizeof(float));
        if(Prob == NULL)
        {
                printf("Notenough memmory for Prob...\n");
                exit(0);
        }
}

void AllocCumulative(void)
{
        Cumulative=(float *)malloc(population*sizeof(float));
        if(Cumulative == NULL)
        {
                printf("Not enough memmory for Cumulative...\n");
                exit(0);
        }
}

void CopyItem(unsigned char *Soure,int soure,unsigned char *Destination,int
destination)
{
        int i;
        for(i=0;i<size;i++)
                Destination[destination*size+i]=Soure[soure*size+i];
        if(Soure == ChromosomeTemp)
                CostBuffer[destination]=CostTemp[soure];
        if(Soure == ChromosomeBuffer)
                CostTemp[destination]=CostBuffer[soure];
}

void CostReproduct(void)
{
        float sum,roulette;
        int   i,j;
        AllocIndex();
        AllocProb();
        AllocCumulative();
//      fprintf(file,"CostSum()-Cost\nCostSum = %d\n",CostSum());
        for(i=0;i<population;i++)
        {
                Cumulative[i] = CostSum()-CostBuffer[i];
//              fprintf(file,"[x%d]\t%f - %f = %f\n",i,CostSum(),CostBuffer
[i],Cumulative[i]);
        }
////////////////////////////////////////////////////////////////////////////
        sum=(float)0;
        for(i=0;i<population;i++)
                sum+=Cumulative[i];
//      fprintf(file,"\nSum of (CostSum-Cost) = %f\n",sum);
////////////////////////////////////////////////////////////////////////////
//      fprintf(file,"Prob Sum(CostSum-Cost)/(CostSum-Cost)\n");
        for(i=0;i<population;i++)
        {
                Prob[i]=Cumulative[i]/sum;
//              fprintf(file,"[%d]\t%f / %f = %f\n",i,Cumulative[i],sum,Prob[i]);
        }

////////////////////////////////////////////////////////////////////////////
        Cumulative[0]=Prob[0];
        for(i=1;i<population;i++)
                Cumulative[i]=Prob[i]+Cumulative[i-1];
//      fprintf(file,"\nCumulative\n");
//      for(i=0;i<population;i++)
//      {
//              if(i == 5) fprintf(file,"\n");
//              fprintf(file,"%f\t",Cumulative[i]);
//      }
//      fprintf(file,"\n\n");
////////////////////////////////////////////////////////////////////////////
//      fprintf(file,"Roulette\n");
//      srand( (unsigned)time( NULL ) );
        for(j=0;j<population;j++)
        {
                roulette=(float)(rand()*rand()%1000000)/(float)1000000;
//              if(j==5) fprintf(file,"\n");fprintf(file,"%f\t",roulette);
                for(i=0;i<population;i++)
                {
```

```
                        if(roulette > Cumulative[i] && roulette <= Cumulative[i+1])
                        {
                                Index[j]=i+1;
                                break;
                        }
                        if(roulette <= Cumulative[0])
                        {
                                Index[j]=0;
                                break;
                        }
                }
        }
//      fprintf(file,"\n\n");
//      for(i=0;i<population;i++)
//      {
//              if(i == 5) fprintf(file,"\n");
//              fprintf(file,"Index[%d] = %d\t",i,Index[i]);
//      }
//      fprintf(file,"\n\n");
////////////////////////////////////////////////////////////////////////////////
        for(i=0;i<population;i++)
                CopyItem(ChromosomeBuffer,Index[i],ChromosomeTemp,i);
//      for(i=0;i<population;i++)
//      {
//              fprintf(file,"[%d]\t[ ",i);
//              for(j=0;j<size;j++)
//                      fprintf(file,"%d ",ChromosomeBuffer[size*i+j]);
//              fprintf(file,"] = %f  ",CostBuffer[i]);
//              fprintf(file,"[%d]\t[ ",Index[i]);
//              for(j=0;j<size;j++)
//                      fprintf(file,"%d ",ChromosomeTemp[size*i+j]);
//              fprintf(file,"] = %f ",CostTemp[i]);
//              fprintf(file,"\n");
//      }
//      fprintf(file,"\n");
        for(i=0;i<population;i++)
                CopyItem(ChromosomeTemp,i,ChromosomeBuffer,i);

        UpdateCost();
////////////////////////////////////////////////////////////////////////////////
        free(Cumulative);
        free(Prob);
        free(Index);
}
```

# ภาคผนวก ช.
# รายละเอียดไฟล์ตัวหนังสือของโปรแกรม Quaclose

เนื้อหารายละเอียดในส่วนนี้จะเสนอถึงรายละเอียดไฟล์ตัวหนังสือของโปรแกรม Quaclose หรือโปรแกรมการแก้ปัญหาการจัดผังโรงงานโดยพิจารณาถึงข้อมูลเชิงคุณภาพและความใกล้ชิดระหว่างสถานี (Total Cloness Rating with Cloness between Station) โดยประกอบไปด้วยไฟล์จำนวนทั้งสิ้น 9 ไฟล์ ได้แก่

1. Genetic.cpp
2. CostSummary.cpp
3. CrossPMX.cpp
4. CrossOX.cpp
5. CrossCX.cpp
6. Cloness.cpp
7. Elitist.cpp
8. Mutation.cpp
9. Reproduct.cpp

ดังมีรายละเอียดดังต่อไปนี้

```
////////////////////////////////////////////////////////////////////////////////
// File Genetic.cpp
// Last update 31/05/97
////////////////////////////////////////////////////////////////////////////////
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
extern unsigned char *Chromosome;
extern signed int    *MemVij;
extern int           Col,Row,size;
int RandomNumber(int interval);
void RandomToPlantLayout(void);
float Evaluate(void);
float Dij(int i,int j);

int RandomNumber(int interval) // interval of random value (0-interval)
{
        return rand()%interval;
}

void RandomToPlantLayout(void)
{
        int temp;
        int attrib;
        int i;
        for(i=0;i<size;i++)
        {
                if(i == 0)
                {
                        temp=RandomNumber(size);
                        Chromosome[i] = (char)temp;
                }
```

```
                else
                {
                        temp=RandomNumber(size);
                        for(attrib=0;attrib<i;attrib++)
                        {
                                if(Chromosome[attrib] == (unsigned char)temp)
                                {
                                        temp=RandomNumber(size);
                                        attrib = -1;
                                }
                        if(Chromosome[attrib] != (unsigned char)temp && attrib == i-1)
                        break;
                        }
                        Chromosome[attrib+1] = (unsigned char)temp;
                }
        }
}

float Evaluate(void)
{
        int   i,j,N=0;
        float cost;
        cost=(float)0;
        for(i=0;i<size;i++)
        {
                for(j=i+1;j<size;j++)
                {
                        cost+=(Dij(i,j)*(float)MemVij[N]);
                        N++;
                }
        }
        return cost;
}

float Dij(int i,int j)
{
        int x1,x2,y1,y2;
        int N,M;
        int X,Y;
        for(N=0;N<size;N++)
                if(Chromosome[N]==i) break;
        for(M=0;M<size;M++)
                if(Chromosome[M]==j) break;
        x1 = N/Col;
        y1 = N%Col;
        x2 = M/Col;
        y2 = M%Col;
//      return (float)sqrt(pow((x1-x2),2)+pow((y1-y2),2));
        X = abs(x1-x2);
        Y = abs(y1-y2);
        if(X <= 1 && Y <= 1) return (float)1;
        else return (float)0;
}
////////////////////////////////////////////////////////////////////////////////
// File CostSummary.cpp
// Last update 31/05/97
////////////////////////////////////////////////////////////////////////////////
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <time.h>

extern unsigned char *ChromosomeBuffer;      // Buffer for String
extern int           population,size;
extern float            *CostBuffer;
extern int                      generation;
float  CostMin(void);
float  CostMax(void);
float  CostAvg(void);
float  CostSum(void);
double STD(void);

float CostSum(void)
{
        int   i;
        float sum;
```

```
        for(i=0;i<population;i++)
                sum+=CostBuffer[i];
        return sum;
}

float CostMin(void)
{
        int    i;
        float  min;
        min=CostBuffer[0];
        for(i=0;i<population;i++)
                if(min > CostBuffer[i])
                        min=CostBuffer[i];
        return min;
}

float CostMax(void)
{
        int    i,j;
        float  max;
        max=CostBuffer[0];
        for(i=0;i<population;i++)
                if(max < CostBuffer[i])
                {
                        max=CostBuffer[i];
                        j=i;
                }
        return max;
}

float CostAvg(void)
{
        return CostSum()/(float)population;
}

double STD(void)
{
        int i;
        double avg;
        double std=(double)0;
        avg=CostAvg();
        for(i=0;i<population;i++)
                std+=pow((double)(CostBuffer[i]-avg),(double)2);
        return sqrt((double)std/(double)(population-1));
}

////////////////////////////////////////////////////////////////////////////////
// File CrossPMX.cpp
// Last update 31/05/97
////////////////////////////////////////////////////////////////////////////////
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>

// Function are not is this file
void CopyItem(unsigned char *Soure,int soure,unsigned char *Destination,int
destination);
void UpdateCost(void);
extern unsigned char *ChromosomeBuffer;      // Buffer for String
extern int                    population;
extern int                    size;
extern FILE *file;
// Function are in this file
void AllocRanDomProbPMX(void);
void AllocIndexProbPMX(void);
void CrossOver_PMX(float Prob_PMX);
void CrossPMX(int first,int second);
void Map(unsigned char *Temp1,int i,unsigned char *Temp2,int j);
void ShowValuePMX(void);
void AllocParentPMX(int first,int second);
void DefineInterval(void);
void SelectProbPMX(float prob);

float  *RanDomProb;
int       *IndexProb;
int       count;
```

```
unsigned char *Parent1 ,*Parent2,temp;
int        interval1,interval2;

void AllocRanDomProbPMX(void)
{
        int i;
        RanDomProb = (float *)malloc(sizeof(float)*population);
        if(RanDomProb == NULL)
        {
                printf("Not enough memmory for RanDomProb...\n");
                exit(0);
        }
//      srand( (unsigned)time( NULL ) );
        for(i=0;i<population;i++)
        {
                RanDomProb[i]=(float)(rand()*rand()%1000000)/(float)1000000;
//              fprintf(file,"%f\t",RanDomProb[i]);
        }
}

void AllocIndexProbPMX(void)
{
        IndexProb=(int *)malloc(sizeof(int)*(count+1));
        if(IndexProb == NULL)
        {
                printf("Not enough memmory for IndesPMX...\n");
                exit(0);
        }
}

void Map(unsigned char *Temp1,int i,unsigned char *Temp2,int j)
{
        Temp1[i]=Temp2[j];
}

void ShowValuePMX(void)
{
        int i;
        fprintf(file,"[ ");
        for(i=0;i<size;i++)
        {
                if(i== interval1 || i == interval2+1)
                        fprintf(file,"| ");
                fprintf(file,"%d ",Parent1[i]);
        }
        fprintf(file,"]\n");
        fprintf(file,"[ ");
        for(i=0;i<size;i++)
        {
                if(i == interval1 || i == interval2+1)
                        fprintf(file,"| ");
                fprintf(file,"%d ",Parent2[i]);
        }
        fprintf(file,"]\n\n");
}

void AllocParentPMX(int first,int second)
{
        Parent1=(unsigned char *)malloc(sizeof(char)*size);
        if(Parent1 == NULL)
        {
                printf("Not enough memmory for Parent1...\n");
                exit(0);
        }
        Parent2=(unsigned char *)malloc(sizeof(char)*size);
        if(Parent2 ==NULL)
        {
                printf("Not enough memmory for Parent2...\n");
                exit(0);
        }
        CopyItem(ChromosomeBuffer,first,Parent1,0);
        CopyItem(ChromosomeBuffer,second,Parent2,0);
}

void DefineInterval(void)
{
        do
```

```
        interval1=rand()%size;
        while(interval1 < 1 || interval1 >= size-2);
//      fprintf(file,"Interval 1 = %d\n",interval1);
        do
                interval2=rand()%size;
        while(interval2 <= interval1 || interval2 > size-2);
//      fprintf(file,"Interval 2 = %d\n",interval2);
}

void CrossPMX(int first,int second)
{
        int  i,j;
        AllocParentPMX(first,second);
        DefineInterval();
//      ShowValuePMX();
        for(i=interval1;i<=interval2;i++)
        {
                temp=Parent1[i];
                Parent1[i]=Parent2[i];
                Parent2[i]=temp;
        }
        for(i=0;i<interval1;i++)
        {
                for(j=interval1;j<=interval2;j++)
                {
                        if(Parent1[i]==Parent1[j])
                        {
                                Map(Parent1,i,Parent2,j);
                                j=interval1-1;
                        }
                }
                for(j=interval1;j<=interval2;j++)
                {
                        if(Parent2[i]==Parent2[j])
                        {
                                Map(Parent2,i,Parent1,j);
                                j=interval1-1;
                        }
                }
        }
        for(i=interval2+1;i<size;i++)
        {
                for(j=interval1;j<=interval2;j++)
                {
                        if(Parent1[i]==Parent1[j])
                        {
                                Map(Parent1,i,Parent2,j);
                                j=interval1-1;
                        }
                }
                for(j=interval1;j<=interval2;j++)
                {
                        if(Parent2[i]==Parent2[j])
                        {
                                Map(Parent2,i,Parent1,j);
                                j=interval1-1;
                        }
                }
        }
        CopyItem(Parent1,0,ChromosomeBuffer,first);
        CopyItem(Parent2,0,ChromosomeBuffer,second);
//      ShowValuePMX();
        free(Parent1);
        free(Parent2);
}

void SelectProbPMX(float prob)
{
        int i,j;
//      fprintf(file,"Select .....\n");
        j=0;
        for(i=0;i<population;i++)
        {
                if(RanDomProb[i] <= prob)
                {
                        IndexProb[j]=i;
                        j++;
```

```
//                      if(i%6 == 5) fprintf(file,"\n");fprintf(file,"%d\t",i);
                }
        }
}

void CrossOver_PMX(float Prob_PMX)
{
        int     i;
        if(Prob_PMX == 0) return;
        AllocRanDomProbPMX();
        count=0;
        for(i=0;i<population;i++)
                if(RanDomProb[i] <= Prob_PMX) count++;
//      fprintf(file,"\nCount = %d\n",count);
        AllocIndexProbPMX();
        SelectProbPMX(Prob_PMX);
//      fprintf(file,"\n");
        if(count%2 == 1) count--;
        for(i=0;i<count;i+=2)
        {
//              fprintf(file,"[%d %d]\n",IndexProb[i],IndexProb[i+1]);
                CrossPMX(IndexProb[i],IndexProb[i+1]);
        }
        UpdateCost();
        free(IndexProb);
        free(RanDomProb);
}

////////////////////////////////////////////////////////////////////////////////
// File CrossOX.cpp
// Last update 31/05/97
////////////////////////////////////////////////////////////////////////////////
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>

extern unsigned char *ChromosomeBuffer;       // Buffer for String
extern int size;
extern int population;
extern FILE *file;

void CopyItem(unsigned char *Soure,int soure,unsigned char *Destination,int
destination);
void UpdateCost(void);
unsigned char *ParentOX1,*ParentOX2;
int interval1OX,interval2OX;
float  *RanDomProbOX;
int    *IndexProbOX;
int    CountOX;
char   hole='*';
unsigned char *Temp1,*Temp2,*TempF,*TempM;
void ShowValueOX(void);
void CrossOX(void);
void AllocRanDomProbOX();
void DefineIntervalOX(void);
void AllocParentOX(int first,int second);
void SelectProbOX(float prob);
void AllocIndexProbOX();

void ShowValueOX(void)
{
        int i;
        fprintf(file,"[ ");
        for(i=0;i<size;i++)
        {
                if(i== interval1OX || i == interval2OX+1)
                        fprintf(file,"| ");
                if(ParentOX1[i] == hole) fprintf(file,"# ");
                else fprintf(file,"%d ",ParentOX1[i]);
        }
        fprintf(file,"]\n");
        fprintf(file,"[ ");
        for(i=0;i<size;i++)
        {
                if(i == interval1OX || i == interval2OX+1)
                        fprintf(file,"| ");
```

```
                if(ParentOX2[i] == hole) fprintf(file,"# ");
                else fprintf(file,"%d ",ParentOX2[i]);
        }
        fprintf(file,"]\n");
}

void AllocRanDomProbOX(void)
{
        int i;
        RanDomProbOX = (float *)malloc(sizeof(float)*population);
        if(RanDomProbOX == NULL)
        {
                printf("Not enough memmory for RanDomProbOX...\n");
                exit(0);
        }
//      srand( (unsigned)time( NULL ) );
        for(i=0;i<population;i++)
        {
                RanDomProbOX[i]=(float)(rand()*rand()%1000000)/(float)1000000;
//              if(i == 5) fprintf(file,"\n");
//              fprintf(file,"%f\t",RanDomProbOX[i]);
        }
}

void AllocIndexProbOX(void)
{
        IndexProbOX=(int *)malloc(sizeof(int)*(CountOX+1));
        if(IndexProbOX == NULL)
        {
                printf("Not enough memmory for IndesOX...\n");
                exit(0);
        }
}

void AllocParentOX(int first,int second)
{
        ParentOX1=(unsigned char *)malloc(sizeof(char)*size);
        if(ParentOX1 == NULL)
        {
                printf("Not enough memmory for ParentOX1...\n");
                exit(0);
        }
        ParentOX2=(unsigned char *)malloc(sizeof(char)*size);
        if(ParentOX2 ==NULL)
        {
                printf("Not enough memmory for ParentOX2...\n");
                exit(0);
        }
        CopyItem(ChromosomeBuffer,first,ParentOX1,0);
        CopyItem(ChromosomeBuffer,second,ParentOX2,0);
}

void DefineIntervalOX(void)
{
        do
                interval1OX=rand()%size;
        while(interval1OX < 1 || interval1OX >= size-2);
//      fprintf(file,"IntervalOX 1 = %d\n",interval1OX);
        do
                interval2OX=rand()%size;
        while(interval2OX <= interval1OX || interval2OX > size-2);
//      fprintf(file,"IntervalOX 2 = %d\n",interval2OX);
}

void SelectProbOX(float prob)
{
        int i,j;
//      fprintf(file,"\nSelect\n");
        j=0;
        for(i=0;i<population;i++)
        {
                if(RanDomProbOX[i] <= prob)
                {
                        IndexProbOX[j]=i;
                        j++;
//                      if(i%6 ==5) fprintf(file,"\n");
//                      fprintf(file,"%d\t",i);
```

```
                }
        }
//      fprintf(file,"\n");
}

void CrossOX(int first,int second)
{
        int i,j;
        AllocParentOX(first,second);
        DefineIntervalOX();
//      ShowValueOX();
//      fprintf(file,"-------------------------\n");
        for(i=0;i<size;i++)
        {
                TempF[i]=Temp1[i]=ParentOX1[i];
                TempM[i]=Temp2[i]=ParentOX2[i];
        }
        for(i=0;i<size;i++)
        {
                for(j=interval1OX;j<=interval2OX;j++)
                if(ParentOX1[i]==Temp2[j]) ParentOX1[i]=hole;        //  Hole = 0xff

                for(j=interval1OX;j<=interval2OX;j++)
                if(ParentOX2[i]==Temp1[j]) ParentOX2[i]=hole;        //  Hole = 0xff
        }
//      ShowValueOX();
//      fprintf(file,"-------------------------\n");
        i=0;
        for(j=interval2OX+1;j<size;j++)
        {
                Temp1[i]=ParentOX1[j];
                Temp2[i]=ParentOX2[j];
                i++;
        }
        for(j=0;j<=interval2OX;j++)
        {
                Temp1[i]=ParentOX1[j];
                Temp2[i]=ParentOX2[j];
                i++;
        }
        for(i=0;i<size;i++)
        {
                if(Temp1[i] == hole)
                {
                        for(j=i;j<size;j++)
                        {
                                if(Temp1[j] != hole) break;
                        }
                        if(j!=size)
                        {
                                Temp1[i]=Temp1[j];
                                Temp1[j]=hole;
                        }
                        else Temp1[i]=Temp1[j];
                }
        }
        for(i=0;i<size;i++)
        {
                if(Temp2[i] == hole)
                {
                        for(j=i;j<size;j++)
                        {
                                if(Temp2[j] != hole) break;
                        }
                        if(j!=size)
                        {
                                Temp2[i]=Temp2[j];
                                Temp2[j]=hole;
                        }
                        else Temp2[i]=Temp2[j];
                }
        }
        i=0;
        for(j=interval2OX+1;j<size;j++)
        {
                ParentOX1[j]=Temp1[i];
                ParentOX2[j]=Temp2[i];
```

```
                i++;
        }
        for(j=0;j<=interval2OX;j++)
        {
                ParentOX1[j]=Temp1[i];
                ParentOX2[j]=Temp2[i];
                i++;
        }
        for(i=interval1OX;i<=interval2OX;i++)
        {
                ParentOX1[i]=TempM[i];
                ParentOX2[i]=TempF[i];
        }
        CopyItem(ParentOX1,0,ChromosomeBuffer,first);
        CopyItem(ParentOX2,0,ChromosomeBuffer,second);
//      ShowValueOX();
        free(ParentOX1);
        free(ParentOX2);
}

void CrossOver_OX(float Prob_OX)
{
        int i;
        unsigned char eos1,eos2;
        if(Prob_OX == 0) return;
        Temp1=(unsigned char *)malloc(sizeof(char)*size);
        Temp2=(unsigned char *)malloc(sizeof(char)*size);
        TempF=(unsigned char *)malloc(sizeof(char)*size);
        TempM=(unsigned char *)malloc(sizeof(char)*size);
        eos1=Temp1[size];
        eos2=Temp2[size];
        Temp1[size]=hole;
        Temp2[size]=hole;
        AllocRanDomProbOX();
        CountOX=0;
        for(i=0;i<population;i++)
                if(RanDomProbOX[i] <= Prob_OX) CountOX++;
//      fprintf(file,"CountOX = %d\n",CountOX);
        AllocIndexProbOX();
        SelectProbOX(Prob_OX);
//      fprintf(file,"\n");
        if(CountOX%2 == 1) CountOX--;
        for(i=0;i<CountOX;i+=2)
        {
//              fprintf(file,"[%d %d]\n",IndexProbOX[i],IndexProbOX[i+1]);
                CrossOX(IndexProbOX[i],IndexProbOX[i+1]);
        }
//      fprintf(file,"\n");
        Temp1[size]=eos1;
        Temp2[size]=eos2;
        UpdateCost();
        free(Temp1);
        free(Temp2);
        free(TempF);
        free(TempM);
        free(IndexProbOX);
        free(RanDomProbOX);
}

////////////////////////////////////////////////////////////////////////////////
// File CrossCX.cpp
// Last update 31/05/97
////////////////////////////////////////////////////////////////////////////////
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>

void CopyItem(unsigned char *Soure,int soure,unsigned char *Destination,int
destination);
void UpdateCost(void);
extern unsigned char *ChromosomeBuffer;      // Buffer for String
extern int                    population;
extern int                    size;
extern FILE *file;
void AllocParentCX(int first,int second);
void AllocRanDomProbCX(void);
```

```
void SelectProbCX(float prob);
void AllocIndexProbCX(void);
void ShowValueCX(void);
unsigned char *ParentCX1,*ParentCX2;
float  *RanDomProbCX;
int    *IndexProbCX;
int    CountCX;

void ShowValueCX(void)
{
        int i;
        for(i=0;i<size;i++)
                fprintf(file,"%d ",ParentCX1[i]);
        fprintf(file,"\n");
        for(i=0;i<size;i++)
                fprintf(file,"%d ",ParentCX2[i]);
        fprintf(file,"\n");
}

void AllocRanDomProbCX(void)
{
        int i;
        RanDomProbCX = (float *)malloc(sizeof(float)*population);
        if(RanDomProbCX == NULL)
        {
                printf("Not enough memmory for RanDomProbOX...\n");
                exit(0);
        }
//      srand( (unsigned)time( NULL ) );
        for(i=0;i<population;i++)
        {
                RanDomProbCX[i]=(float)(rand()*rand()%1000000)/(float)1000000;
//              if(i%6 == 5) fprintf(file,"\n");
//              fprintf(file,"%f\t",RanDomProbCX[i]);
        }
}

void AllocIndexProbCX(void)
{
        IndexProbCX=(int *)malloc(sizeof(int)*(CountCX+1));
        if(IndexProbCX == NULL)
        {
                printf("Not enough memmory for IndesCX...\n");
                exit(0);
        }
}

void AllocParentCX(int first,int second)
{
        ParentCX1=(unsigned char *)malloc(sizeof(char)*size);
        if(ParentCX1 == NULL)
        {
                printf("Not enough memmory for ParentCX1...\n");
                exit(0);
        }
        ParentCX2=(unsigned char *)malloc(sizeof(char)*size);
        if(ParentCX2 ==NULL)
        {
                printf("Not enough memmory for ParentCX2...\n");
                exit(0);
        }
        CopyItem(ChromosomeBuffer,first,ParentCX1,0);
        CopyItem(ChromosomeBuffer,second,ParentCX2,0);
}

void SelectProbCX(float prob)
{
        int i,j;
//      fprintf(file,"Select\n");
        j=0;
        for(i=0;i<population;i++)
        {
                if(RanDomProbCX[i] <= prob)
                {
                        IndexProbCX[j]=i;
                        j++;
//                      if(i==5) fprintf(file,"\n");
```

```
//                      fprintf(file,"%d\t",i);
                }
        }
//      fprintf(file,"\n\n");
}

void CrossCX(int first,int second)
{
        AllocParentCX(first,second);
        int i,mark=0;
        unsigned char first_value,swap_tmp,temp;
//      ShowValueCX();
        temp=ParentCX2[0];
        first_value=ParentCX1[0];
        for(i=0;i<size;i++)
        {
                if(ParentCX1[i]==temp && i != mark)
                {
                        swap_tmp=ParentCX1[i];
                        ParentCX1[i]=ParentCX2[i];
                        ParentCX2[i]=swap_tmp;
                        temp=ParentCX1[i];
                        mark=i;
                        i = -1;
                }
        }
        for(i=0;i<size;i++)
        {
                swap_tmp=ParentCX1[i];
                ParentCX1[i]=ParentCX2[i];
                ParentCX2[i]=swap_tmp;
        }
//      ShowValueCX();fprintf(file,"\n");
        CopyItem(ParentCX1,0,ChromosomeBuffer,first);
        CopyItem(ParentCX2,0,ChromosomeBuffer,second);
}

void CrossOver_CX(float Prob_CX)
{
        int i;
        if(Prob_CX == 0) return;
        CountCX=0;
        AllocRanDomProbCX();
        for(i=0;i<population;i++)
                if(RanDomProbCX[i] <= Prob_CX) CountCX++;
//      fprintf(file,"CountCX = %d\n",CountCX);
        AllocIndexProbCX();
        SelectProbCX(Prob_CX);
//      fprintf(file,"\n");
        if(CountCX%2 == 1) CountCX--;
        for(i=0;i<CountCX;i+=2)
        {
//              fprintf(file,"[%d %d]\n",IndexProbCX[i],IndexProbCX[i+1]);
                CrossCX(IndexProbCX[i],IndexProbCX[i+1]);
        }
//      fprintf(file,"\n");
        UpdateCost();
}

////////////////////////////////////////////////////////////////////////////////
// File Cloness.cpp
// Last update 31/05/97
////////////////////////////////////////////////////////////////////////////////
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <malloc.h>
#include <string.h>
#include <time.h>

unsigned char *Chromosome,*ChromosomeBuffer,*ChromosomeTemp,*BestChromosome;       //
Buffer for String
signed int    *MemVij;                // Buffer for store Vij
float           Prob_PMX,Prob_OX,Prob_CX,Prob_M,mutat;
int           Row,Col,size;
float           *Cost,*CostBuffer,*CostTemp,BestCost;
int                     population;
```

```
int                     generation;
int                     mark[100];
int                     interval,total;
FILE            *file,*summary;
// Functions are not in this file.
void RandomToPlantLayout(void);
float Evaluate(void);
float CostMin(void);
float CostMax(void);
float CostAvg(void);
float CostSum(void);
double STD(void);
void  CostReproduct(void);
void  CrossOver_PMX(float Prob_PMX);
void  CrossOver_OX(float Prob_OX);
void  CrossOver_CX(float Prob_CX);
void  Mutate(float mutate);
void  Elitist(void);
// Functions are in this file.
int TotalWeight(int N);
void AllocDij(void);
void AllocVij(void);
void initial(void);
void AllocCostBuffer(void);
void AllocChromosome(void);
void SavePlantLayout(int pop);
void SaveToFile(int Generation);
void ShowChromosome(void);
void SaveBest(void);
void UpdateCost(void);
void SummaryToFile(int gen);
int  NeverMark(int end,int key);

int  NeverMark(int end,int key)
{
        int i;
        for(i=0;i<end;i++)
                if(mark[i] == key) return 0;
        return 1;
}

void SummaryToFile(int gen)
{
        int i,j,k;
        float max,maxtemp;
        fprintf(summary,"[Generation %d]\n",gen);
        maxtemp=CostMin();
        for(i=0;i<total;i++)
        {
                max=maxtemp;
                for(k=0;k<population;k++)
                {
                        if(CostBuffer[k] >= max && NeverMark(i,k))
                        {
                                max=CostBuffer[k];
                                mark[i]=k;
                        }
                }
                fprintf(summary,"[%d]\t",mark[i]);
                for(j=0;j<size;j++)
                        fprintf(summary,"%d ",ChromosomeBuffer[size*mark[i]+j]);
                fprintf(summary," =\t%f\n",CostBuffer[mark[i]]);
        }
        fprintf(summary,"\nThe Best Cost is : %f\t",BestCost);
        fprintf(summary,"[ ");
        for(i=0;i<size;i++)
                fprintf(summary,"%d ",BestChromosome[i]);
        fprintf(summary,"]\n");
        fprintf(summary,"Averang is : %f\n",CostAvg());
        fprintf(summary,"STD     is : %f\n\n",STD());
}

void UpdateCost(void)
{
        int i,j;
        for(i=0;i<population;i++)
        {
```

```
                for(j=0;j<size;j++)
                        Chromosome[j]=ChromosomeBuffer[i*size+j];
                CostBuffer[i]=Evaluate();
        }
}

void ShowChromosome(void)
{
        int i;
/*
        for(i=0;i<population;i++)
        {
                printf("[%d]\t",i);

                for(j=0;j<size;j++)
                        printf("%d ",ChromosomeBuffer[size*i+j]);
                printf(" =\t%f\n",CostBuffer[i]);
        }
*/
        fprintf(file,"%f\t",CostMin());
        fprintf(file,"%f\t",CostMax());
        fprintf(file,"%f\t",CostAvg());
        fprintf(file,"%f\t",CostSum());
        fprintf(file,"%f\t",STD());

        fprintf(file,"[ ");
        for(i=0;i<size;i++)
                fprintf(file,"%d ",BestChromosome[i]);
        fprintf(file,"]\n");
}

int TotalWeight(int N) // Formular N=N-1+N-2+N-3...+1
{
        if(N > 1)
                return N+TotalWeight(N-1);
        else return 1;
}

void AllocDij(void)    // Allocate memory for construct string
{
        Chromosome = (unsigned char *)malloc(size*sizeof(char));
        if(Chromosome == NULL)
        {
                printf("Not enough memmory for \"Chromosome\"\n");
                exit(0);
        }
}

// Allocate memmory for weight table and input weight in term continue
// Input : None
// Output: Enter value in pointer MemVij
void AllocVij(void)
{
        FILE *input;
        char buffer[10];
        int  weight,i;
        input=fopen("input.dat","rt");
        if(input == NULL)
        {
                printf("Can not open file input.dat\n");
                exit(0);
        }
        fgets(buffer,7,input);
        sscanf(buffer,"%d",&Row);      // Get ROW
        fgets(buffer,7,input);
        sscanf(buffer,"%d",&Col);      // Get COL
        size=Row*Col;
        weight = TotalWeight(size-1);                  // Find total of weight
        printf("weight = %d\n",weight);
        MemVij = (int *)malloc(weight*sizeof(weight));// Alloc memmory for input weight
        if(MemVij == NULL)
        {
                printf("Not enough memmory for \"MemVij\"\n");
                exit(0);
        }
        for(i=0;i<weight;i++)
        {
```

```
                fgets(buffer,7,input);
                sscanf(buffer,"%d",&MemVij[i]);
        }
        printf("MemVij is\n");
        for(i=0;i<weight;i++)
                printf("%d ",MemVij[i]);
        printf("\n");
        fclose(input);
}

void AllocCostBuffer(void)
{
        CostBuffer=(float *)malloc(sizeof(float)*population);
        if(CostBuffer == NULL)
        {
                printf("Not enough memmory for \"CostBuffer\"\n");
                exit(0);
        }
        CostTemp=(float *)malloc(sizeof(float)*population);
        if(CostTemp == NULL)
        {
                printf("Not enough memmory for \"CostTemp\"\n");
                exit(0);
        }
}

void AllocChromosomeBuffer(void)
{
        ChromosomeBuffer=(unsigned char *)malloc(size*sizeof(char)*population);
        if(ChromosomeBuffer == NULL)
        {
                printf("Not enough memmory for \"ChromosomeBuffer\"\n");
                exit(0);
        }
        ChromosomeTemp=(unsigned char *)malloc(size*sizeof(char)*population);
        if(ChromosomeTemp == NULL)
        {
                printf("Not enough memmory for \"ChromosomeTemp\"\n");
                exit(0);
        }
        BestChromosome=(unsigned char *)malloc(size*sizeof(char));
        if(ChromosomeTemp == NULL)
        {
                printf("Not enough memmory for \"Best Chromosome\"\n");
                exit(0);
        }
}

void SavePlantLayout(int pop)
{
        int i;
        for(i=0;i<size;i++)
                ChromosomeBuffer[pop*size+i]=Chromosome[i];
}

void SaveToFile(int Generation)
{
        fclose(file);
        fclose(summary);
}

void initial(void)
{
        int i;
        printf("Enter Generation : ");scanf("%d",&generation);
        printf("Enter Population : ");scanf("%d",&population);
        printf("Enter Probability PMX : ");scanf("%f",&Prob_PMX);
        printf("Enter Probability OX  : ");scanf("%f",&Prob_OX);
        printf("Enter Probability CX  : ");scanf("%f",&Prob_CX);
        printf("Enter Mutate : ");scanf("%f",&mutat);
        printf("How many period for report : ");scanf("%d",&interval);
        printf("How many report chromosome : ");scanf("%d",&total);
        AllocVij();             // Build relationship chart table
        AllocCostBuffer();
        AllocDij();             // Build plant layout
        AllocChromosomeBuffer();
        file=fopen("result.txt","wt");
```

```
        if(file == NULL)
        {
                printf("Can not open file \"RESULT.TXT\" \n");
                exit(0);
        }
        summary=fopen("summary.txt","wt");
        if(file == NULL)
        {
                printf("Can not open file \"SUMMARY.TXT\" \n");
                exit(0);
        }
//      srand( (unsigned)time( NULL ) );
        for(i=0;i<population;i++)
        {
                RandomToPlantLayout();
                SavePlantLayout(i);
                CostBuffer[i]=Evaluate();
        }
}

void main(void)
{
        int i;
        clock_t start, finish;
        double  use;
        char *version="Program Plant Laout version 1.01 for ClossNess\nLast update
14/09/97\n";

        printf("%s",version);  // Show version
        initial();
        fprintf(file,"%s",version);
        start=clock();
        BestCost=(float)0;
        for(i=0;i<generation;i++)
        {
                SaveBest();
                CostReproduct();
                SaveBest();
                CrossOver_PMX(Prob_PMX);
                SaveBest();
                CrossOver_OX(Prob_OX);
                SaveBest();
                CrossOver_CX(Prob_CX);
                SaveBest();
                Mutate(mutat);
                SaveBest();
                Elitist();
                ShowChromosome();
                printf("Generation %d Completed\n",i);
                if(i%interval == 0)
                SummaryToFile(i);
        }

        SaveToFile(i);
        finish=clock();
        use=(double)(finish-start)/CLOCKS_PER_SEC;
        printf("Function sucessfully used time = %2.2f sec.\n",use);
        free(BestChromosome);
        free(ChromosomeBuffer);
        free(CostBuffer);
        free(ChromosomeTemp);
        free(CostTemp);
        free(MemVij);
}

////////////////////////////////////////////////////////////////////////////////
// File Elitist.cpp
// Last update 31/05/97
////////////////////////////////////////////////////////////////////////////////
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

float  CostMin(void);
void   CopyItem(unsigned char *Soure,int soure,unsigned char *Destination,int
destination);
float  CostMax(void);
```

```
void UpdateCost(void);
void   SaveBest(void);
extern unsigned char *Chromosome,*ChromosomeBuffer,*BestChromosome; // Buffer for
String
extern float            *CostBuffer,BestCost;
extern int                      population;
extern int                      size;
extern FILE *file;

void SaveBest(void)
{
        int i;
        if(BestCost < CostMax())
        {
                BestCost=CostMax();
                for(i=0;i<population;i++)
                        if(CostBuffer[i]==BestCost) break;
                CopyItem(ChromosomeBuffer,i,BestChromosome,0);
        }
}

void ReplaceBestChromosome(void)
{
        int i;
        float BedCost;
        BedCost=CostMin();
        for(i=0;i<population;i++)
                if(CostBuffer[i]==BedCost) break;
        CopyItem(BestChromosome,0,ChromosomeBuffer,i);
}

void Elitist(void)
{
        ReplaceBestChromosome();
        UpdateCost();
}

////////////////////////////////////////////////////////////////////////////////
// File Mutation.cpp
// Last update 31/05/97
////////////////////////////////////////////////////////////////////////////////
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>

void UpdateCost();
extern int size;
extern int population;
extern unsigned char *ChromosomeBuffer;
extern FILE *file;
void InitialMutate(void);
void RandomMutate(void);
void SelectMutate(void);
void Mutation(int position);
void ShowChromosome(int pop,int siz,int swaper);
void Mutate(float mutat);
float *Gene,mutate;

void InitialMutate(void)
{
        Gene=(float *)malloc(sizeof(float)*size*population);
        if(Gene == NULL)
        {
                printf("Not enough memory for Gene\n");
                exit(0);
        }
}
/*
void RandomChromosome(void)
{
        int Size,Population;
        for(Population=0;Population<population;Population++)
        {
                for(Size=0;Size<size;Size++)
                {
                        do
```

```
                        {
                                ChromosomeBuffer[(Population*size)+Size]=rand()%size;
                        } while(Redundant(Population,Size));
                }
        }
        for(Population=0;Population<population;Population++)
        {
                printf("Level %d \t[ ",Population);
                for(Size=0;Size<size;Size++)
                {
                        printf("%d ",ChromosomeBuffer[(Population*size)+Size]);
                }
                printf("]\n");
        }
}

int Redundant(int pop,int siz)
{
        int i;
        for(i=0;i<siz;i++)
        {
                if(ChromosomeBuffer[(pop*size)+siz] == ChromosomeBuffer[(pop*size)+i])
                        return 1;
        }
        return 0;
}

*/

void RandomMutate(void)
{
        int i;
//      srand( (unsigned)time( NULL ) );
        for(i=0;i<population*size;i++)
        {
                Gene[i]=(float)(rand()*rand()%1000000)/(float)1000000;
//              if(i%6 == 5) fprintf(file,"\n");
//              fprintf(file,"%f\t",Gene[i]);
        }
}

void SelectMutate(void)
{
        int i;

//      fprintf(file,"\nSelect Mutate\n");
        for(i=0;i<population*size;i++)
        {
                if(Gene[i] <= mutate)
                {
                        Mutation(i);
                }
        }
}

void Mutation(int position)
{
        int pop,siz;
        int swaper;
        unsigned char temp;
        pop = position/size;
        siz = position%size;
//      fprintf(file,"[Expected Mutation=%d]\tPercent mutate = %f\n   Chromosome  =
%d\n   Logus = %d\n",position,Gene[position],pop,siz);
        swaper=size;
//      ShowChromosome(pop,siz,swaper);
//      srand( (unsigned)time( NULL ) );
        do
        {
                swaper=rand()%size;
        } while(swaper == siz);
//      ShowChromosome(pop,siz,swaper);
        temp=ChromosomeBuffer[(pop*size)+siz];
        ChromosomeBuffer[(pop*size)+siz] = ChromosomeBuffer[(pop*size)+swaper];
        ChromosomeBuffer[(pop*size)+swaper] = temp;
//      ShowChromosome(pop,siz,swaper);
}
```

```
void ShowChromosome(int pop,int siz,int swaper)
{
        int i;
//      fprintf(file,"Chromosome %d \t[",pop);
        for(i=0;i<size;i++)
        {
                if(i == siz || i == swaper) fprintf(file,"|");
                else fprintf(file," ");
                fprintf(file,"%d",ChromosomeBuffer[(pop*size )+i],;
                if(i == siz || i == swaper) fprintf(file,"|");
                else fprintf(file," ");
        }
        fprintf(file,"]\n");
}

void Mutate(float mutat)
{
        mutate=mutat;
        InitialMutate();
        RandomMutate();
        SelectMutate();
        UpdateCost();
        free(Gene);
}

////////////////////////////////////////////////////////////////////////////////
// File Reproduct.cpp
// Last update 31/05/97
////////////////////////////////////////////////////////////////////////////////
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <time.h>

extern unsigned char *ChromosomeBuffer,*ChromosomeTemp,     // Buffer for String
extern int           population,size;
extern float            *CostBuffer,*CostTemp;
extern FILE *file;
// Function are not in this file
float  CostSum(void);
void   UpdateCost(void);
// Function are in this file
void AllocIndex(void);
void AllocProb(void);
void AllocCumulative(void);
void CostReproduct(void);
void CopyItem(unsigned char *Soure,int soure,unsigned char *Destination,int
destination);
int                     *Index;
float           *Prob,*Cumulative;

void AllocIndex(void)
{
        Index=(int *)malloc(population*sizeof(int));
        if(Index == NULL)
        {
                printf("Not enough memmory for Index...\n");
                exit(0);
        }
}

void AllocProb(void)
{
        Prob=(float *)malloc(population*sizeof(float));
        if(Prob == NULL)
        {
                printf("Notenough memmory for Prob...\n");
                exit(0);
        }
}

void AllocCumulative(void)
{
        Cumulative=(float *)malloc(population*sizeof(float));
        if(Cumulative == NULL)
```

```
        {
                printf("Not enough memmory for Cumulative...\n");
                exit(0);
        }
}

void CopyItem(unsigned char *Soure,int soure,unsigned char *Destination,int
destination)
{
        int i;
        for(i=0;i<size;i++)
                Destination[destination*size+i]=Soure[soure*size+i];
        if(Soure == ChromosomeTemp)
                CostBuffer[destination]=CostTemp[soure];
        if(Soure == ChromosomeBuffer)
                CostTemp[destination]=CostBuffer[soure];
}

void CostReproduct(void)
{
        float sum,roulette;
        int   i,j;
        AllocIndex();
        AllocProb();
        AllocCumulative();
        sum=(float)0;
        for(i=0;i<population;i++)
                sum+=CostBuffer[i];
        for(i=0;i<population;i++)
                Prob[i]=CostBuffer[i]/sum;
        Cumulative[0]=Prob[0];
        for(i=1;i<population;i++)
                Cumulative[i]=Prob[i]+Cumulative[i-1];
        for(j=0;j<population;j++)
        {
                roulette=(float)(rand()*rand()%1000000)/(float)1000000;
                for(i=0;i<population;i++)
                {
                        if(roulette > Cumulative[i] && roulette <= Cumulative[i+1])
                        {
                                Index[j]=i+1;
                                break;
                        }
        //              if(roulette <= Cumulative[0])
                        else
                        {
                                Index[j]=0;
                                break;
                        }
                }
        }
        for(i=0;i<population;i++)
                CopyItem(ChromosomeBuffer,Index[i],ChromosomeTemp,i);
        for(i=0;i<population;i++)
                CopyItem(ChromosomeTemp,i,ChromosomeBuffer,i);
        UpdateCost();
        free(Cumulative);
        free(Prob);
        free(Index);
}
```

# ภาคผนวก ฌ
# รายละเอียดไฟล์ตัวหนังสือของโปรแกรม Quantity

เนื้อหารายละเอียดในส่วนนี้จะเสนอถึงรายละเอียดไฟล์ตัวหนังสือของโปรแกรม Quantity หรือโปรแกรมการแก้ปัญหาการจัดผังโรงงานโดยพิจารณาถึงข้อมูลเชิงปริมาณ ด้วย GAs ซึ่งประกอบไปด้วยไฟล์จำนวนทั้งสิ้น 9 ไฟล์ ได้แก่

1. Genetic.cpp
2. CostSummary.cpp
3. CrossPMX.cpp
4. CrossOX.cpp
5. CrossCX.cpp
6. Distance.cpp
7. Elitist.cpp
8. Mutation.cpp
9. Reproduct.cpp

**ดังมีรายละเอียดดังต่อไปนี้**

```
////////////////////////////////////////////////////////////////////////////////
// File Genetic.cpp
// Last update 31/05/97
////////////////////////////////////////////////////////////////////////////////
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

extern unsigned char *Chromosome;
extern signed int    *MemVij,*MemFij;
extern int            Col,Row,size;
int RandomNumber(int interval);
void RandomToPlantLayout(void);
float Evaluate(void);
float Dij(int i,int j);

int RandomNumber(int interval) // interval of random value (0-interval)
{
        return rand()%interval;
}

void RandomToPlantLayout(void)
{
        int temp;
        int attrib;
        int i;
        for(i=0;i<size;i++)
        {
                if(i == 0)
                {
                        temp=RandomNumber(size);
                        Chromosome[i] = (char)temp;
                }
                else
                {
```

```
                        temp=RandomNumber(size);
                        for(attrib=0;attrib<i;attrib++)
                        {
                                if(Chromosome[attrib] == (unsigned char)temp)
                                {
                                        temp=RandomNumber(size);
                                        attrib = -1;
                                }
                        if(Chromosome[attrib] != (unsigned char)temp && attrib == i-1)
                        break;
                        }
                        Chromosome[attrib+1] = (unsigned char)temp;
                }
        }
}

float Evaluate(void)
{
        int   i,j,N=0;
        float cost;
        cost=(float)0;
        for(i=0;i<size;i++)
        {
                for(j=0;j<size;j++)
                {
                        cost+=(Dij(i,j)*(float)MemVij[N]*(float)MemFij[N]);
                        N++;
                }
        }
        return cost;
}

float Dij(int i,int j)
{
        int x1,x2,y1,y2;
        int N,M;
        for(N=0;N<size;N++)
                if(Chromosome[N]==i) break;
        for(M=0;M<size;M++)
                if(Chromosome[M]==j) break;
        x1 = N/Col;
        y1 = N%Col;
        x2 = M/Col;
        y2 = M%Col;
        return (float)sqrt(pow((x1-x2),2)+pow((y1-y2),2));
}

////////////////////////////////////////////////////////////////////////////////
// File CostSummary.cpp
// Last update 31/05/97
////////////////////////////////////////////////////////////////////////////////
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <time.h>

extern unsigned char *ChromosomeBuffer;      // Buffer for String
extern int            population,size;
extern float         *CostBuffer;
extern int            generation;
float  CostMin(void);
float  CostMax(void);
float  CostAvg(void);
float  CostSum(void);
double STD(void);

float CostSum(void)
{
        int   i;
        float sum;
        for(i=0;i<population;i++)
                sum+=CostBuffer[i];
        return sum;
}

float CostMin(void)
```

```
{
        int    i;
        float  min;

        min=CostBuffer[0];
        for(i=0;i<population;i++)
                if(min > CostBuffer[i])
                        min=CostBuffer[i];
        return min;
}

float CostMax(void)
{
        int    i,j;
        float  max;
        max=CostBuffer[0];
        for(i=0;i<population;i++)
                if(max < CostBuffer[i])
                {
                        max=CostBuffer[i];
                        j=i;
                }
        return max;
}

float CostAvg(void)
{
        return CostSum()/(float)population;
}

double STD(void)
{
        int i;
        double avg;
        double std=(double)0;
        avg=CostAvg();
        for(i=0;i<population;i++)
                std+=pow((double)(CostBuffer[i]-avg),(double)2);
        return sqrt((double)std/(double)(population-1));
}

////////////////////////////////////////////////////////////////////////////////
// File CrossPMX.cpp
// Last update 31/05/97
////////////////////////////////////////////////////////////////////////////////
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>

// Function are not is this file
void CopyItem(unsigned char *Soure,int soure,unsigned char *Destination,int
destination);
void UpdateCost(void);
extern unsigned char *ChromosomeBuffer;      // Buffer for String
extern int     population;
extern int     size;
extern FILE *file;

// Function are in this file
void AllocRanDomProbPMX(void);
void AllocIndexProbPMX(void);
void CrossOver_PMX(float Prob_PMX);
void CrossPMX(int first,int second);
void Map(unsigned char *Temp1,int i,unsigned char *Temp2,int j);
void ShowValuePMX(void);
void AllocParentPMX(int first,int second);
void DefineInterval(void);
void SelectProbPMX(float prob);
float   *RanDomProb;
int     *IndexProb;
int     count;
unsigned char *Parent1 ,*Parent2,temp;
int     interval1,interval2;

void AllocRanDomProbPMX(void)
{
```

```
        int i;
        RanDomProb = (float *)malloc(sizeof(float)*population);
        if(RanDomProb == NULL)
        {
                printf("Not enough memmory for RanDomProb...\n");
                exit(0);
        }
//      srand( (unsigned)time( NULL ) );
        for(i=0;i<population;i++)
        {
                RanDomProb[i]=(float)(rand()*rand()%1000000)/(float)1000000;
//              fprintf(file,"%f\t",RanDomProb[i]);
        }
}

void AllocIndexProbPMX(void)
{
        IndexProb=(int *)malloc(sizeof(int)*(count+1));
        if(IndexProb == NULL)
        {
                printf("Not enough memmory for IndesPMX...\n");
                exit(0);
        }
}

void Map(unsigned char *Temp1,int i,unsigned char *Temp2,int j)
{
        Temp1[i]=Temp2[j];
}

void ShowValuePMX(void)
{
        int i;
        fprintf(file,"[ ");
        for(i=0;i<size;i++)
        {
                if(i== interval1 || i == interval2+1)
                        fprintf(file,"| ");
                fprintf(file,"%d ",Parent1[i]);
        }
        fprintf(file,"]\n");
        fprintf(file,"[ ");
        for(i=0;i<size;i++)
        {
                if(i == interval1 || i == interval2+1)
                fprintf(file,"| ");
                fprintf(file,"%d ",Parent2[i]);
        }
        fprintf(file,"]\n\n");
}

void AllocParentPMX(int first,int second)
{
        Parent1=(unsigned char *)malloc(sizeof(char)*size);
        if(Parent1 == NULL)
        {
                printf("Not enough memmory for Parent1...\n");
                exit(0);
        }
        Parent2=(unsigned char *)malloc(sizeof(char)*size);
        if(Parent2 ==NULL)
        {
                printf("Not enough memmory for Parent2...\n");
                exit(0);
        }
        CopyItem(ChromosomeBuffer,first,Parent1,0);
        CopyItem(ChromosomeBuffer,second,Parent2,0);
}

void DefineInterval(void)
{
        do
                interval1=rand()%size;
        while(interval1 < 1 || interval1 >= size-2);

//      fprintf(file,"Interval 1 = %d\n",interval1);
```

```
        do
                interval2=rand()%size;
        while(interval2 <= interval1 || interval2 > size-2);

//      fprintf(file,"Interval 2 = %d\n",interval2);
}

void CrossPMX(int first,int second)
{
        int                     i,j;
        AllocParentPMX(first,second);
        DefineInterval();
//      ShowValuePMX();
        for(i=interval1;i<=interval2;i++)
        {
                temp=Parent1[i];
                Parent1[i]=Parent2[i];
                Parent2[i]=temp;
        }
        for(i=0;i<interval1;i++)
        {
                for(j=interval1;j<=interval2;j++)
                {
                        if(Parent1[i]==Parent1[j])
                        {
                                Map(Parent1,i,Parent2,j);
                                j=interval1-1;
                        }
                }
                for(j=interval1;j<=interval2;j++)
                {
                        if(Parent2[i]==Parent2[j])
                        {
                                Map(Parent2,i,Parent1,j);
                                j=interval1-1;
                        }
                }
        }
        for(i=interval2+1;i<size;i++)
        {
                for(j=interval1;j<=interval2;j++)
                {
                        if(Parent1[i]==Parent1[j])
                        {
                                Map(Parent1,i,Parent2,j);
                                j=interval1-1;
                        }
                }
                for(j=interval1;j<=interval2;j++)
                {
                        if(Parent2[i]==Parent2[j])
                        {
                                Map(Parent2,i,Parent1,j);
                                j=interval1-1;
                        }
                }
        }
        CopyItem(Parent1,0,ChromosomeBuffer,first);
        CopyItem(Parent2,0,ChromosomeBuffer,second);
//      ShowValuePMX();
        free(Parent1);
        free(Parent2);
}

void SelectProbPMX(float prob)
{
        int i,j;
//      fprintf(file,"Select .....\n");
        j=0;
        for(i=0;i<population;i++)
        {
                if(RanDomProb[i] <= prob)
                {
                        IndexProb[j]=i;
                        j++;
//                      if(i%6 == 5) fprintf(file,"\n");fprintf(file,"%d\t",i);
                }
```

```
        }
}

void CrossOver_PMX(float Prob_PMX)
{
        int        i;
        if(Prob_PMX == 0) return;
        AllocRanDomProbPMX();
        count=0;
        for(i=0;i<population;i++)
                if(RanDomProb[i] <= Prob_PMX) count++;
//      fprintf(file,"\nCount = %d\n",count);
        AllocIndexProbPMX();
        SelectProbPMX(Prob_PMX);
//      fprintf(file,"\n");
        if(count%2 == 1) count--;
        for(i=0;i<count;i+=2)
        {
//              fprintf(file,"[%d %d]\n",IndexProb[i],IndexProb[i+1]);
                CrossPMX(IndexProb[i],IndexProb[i+1]);
        }
        UpdateCost();
        free(IndexProb);
        free(RanDomProb);
}

////////////////////////////////////////////////////////////////////////////////
// File CrossOX.cpp
// Last update 31/05/97
////////////////////////////////////////////////////////////////////////////////
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>

extern unsigned char *ChromosomeBuffer;       // Buffer for String
extern int size;
extern int population;
extern FILE *file;
void CopyItem(unsigned char *Soure,int soure,unsigned char *Destination,int
destination);
void UpdateCost(void);
unsigned char *ParentOX1,*ParentOX2;
int interval1OX,interval2OX;
float  *RanDomProbOX;
int    *IndexProbOX;
int    CountOX;
char   hole='*';

unsigned char *Temp1,*Temp2,*TempF,*TempM;
void ShowValueOX(void);
void CrossOX(void);
void AllocRanDomProbOX();
void DefineIntervalOX(void);
void AllocParentOX(int first,int second);
void SelectProbOX(float prob);
void AllocIndexProbOX();

void ShowValueOX(void)
{
        int i;
        fprintf(file,"[ ");
        for(i=0;i<size;i++)
        {
                if(i== interval1OX || i == interval2OX+1)
                fprintf(file,"| ");
                if(ParentOX1[i] == hole) fprintf(file,"# ");
                else fprintf(file,"%d ",ParentOX1[i]);
        }
        fprintf(file,"]\n");
        fprintf(file,"[ ");
        for(i=0;i<size;i++)
        {
                if(i == interval1OX || i == interval2OX+1)
                        fprintf(file,"| ");
                if(ParentOX2[i] == hole) fprintf(file,"# ");
                else fprintf(file,"%d ",ParentOX2[i]);
```

```
        }
        fprintf(file,"]\n");
}

void AllocRanDomProbOX(void)
{
        int i;
        RanDomProbOX = (float *)malloc(sizeof(float)*population);
        if(RanDomProbOX == NULL)
        {
                printf("Not enough memmory for RanDomProbOX...\n");
                exit(0);
        }
//      srand( (unsigned)time( NULL ) );
        for(i=0;i<population;i++)
        {
                RanDomProbOX[i]=(float)(rand()*rand()%1000000)/(float)1000000;
//              if(i == 5) fprintf(file,"\n");
//              fprintf(file,"%f\t",RanDomProbOX[i]);
        }
}

void AllocIndexProbOX(void)
{
        IndexProbOX=(int *)malloc(sizeof(int)*(CountOX+1));
        if(IndexProbOX == NULL)
        {
                printf("Not enough memmory for IndexOX...\n");
                exit(0);
        }
}

void AllocParentOX(int first,int second)
{
        ParentOX1=(unsigned char *)malloc(sizeof(char)*size);
        if(ParentOX1 == NULL)
        {
                printf("Not enough memmory for ParentOX1...\n");
                exit(0);
        }
        ParentOX2=(unsigned char *)malloc(sizeof(char)*size);
        if(ParentOX2 ==NULL)
        {
                printf("Not enough memmory for ParentOX2...\n");
                exit(0);
        }
        CopyItem(ChromosomeBuffer,first,ParentOX1,0);
        CopyItem(ChromosomeBuffer,second,ParentOX2,0);
}

void DefineIntervalOX(void)
{
        do
        interval1OX=rand()%size;
        while(interval1OX < 1 || interval1OX >= size-2);

//      fprintf(file,"IntervalOX 1 = %d\n",interval1OX);

        do
                interval2OX=rand()%size;
        while(interval2OX <= interval1OX || interval2OX > size-2);
//      fprintf(file,"IntervalOX 2 = %d\n",interval2OX);
}

void SelectProbOX(float prob)
{
        int i,j;
//      fprintf(file,"\nSelect\n");
        j=0;
        for(i=0;i<population;i++)
        {
                if(RanDomProbOX[i] <= prob)
                {
                        IndexProbOX[j]=i;
                        j++;
//                      if(i%6 ==5) fprintf(file,"\n");
//                      fprintf(file,"%d\t",i);
```

```
        }
    }
//      fprintf(file,"\n");
}

void CrossOX(int first,int second)
{
        int i,j;
        AllocParentOX(first,second);
        DefineIntervalOX();
//      ShowValueOX();
//      fprintf(file,"-------------------------\n");
        for(i=0;i<size;i++)
        {
                TempF[i]=Temp1[i]=ParentOX1[i];
                TempM[i]=Temp2[i]=ParentOX2[i];
        }
        for(i=0;i<size;i++)
        {
                for(j=interval1OX;j<=interval2OX;j++)
                if(ParentOX1[i]==Temp2[j]) ParentOX1[i]=hole;        //  Hole = 0xff
                for(j=interval1OX;j<=interval2OX;j++)
                if(ParentOX2[i]==Temp1[j]) ParentOX2[i]=hole;        //  Hole = 0xff
        }
//      ShowValueOX();
//      fprintf(file,"-------------------------\n");
        i=0;
        for(j=interval2OX+1;j<size;j++)
        {
                Temp1[i]=ParentOX1[j];
                Temp2[i]=ParentOX2[j];
                i++;
        }
        for(j=0;j<=interval2OX;j++)
        {
                Temp1[i]=ParentOX1[j];
                Temp2[i]=ParentOX2[j];
                i++;
        }
        for(i=0;i<size;i++)
        {
                if(Temp1[i] == hole)
                {
                        for(j=i;j<size;j++)
                        {
                                if(Temp1[j] != hole) break;
                        }
                        if(j!=size)
                        {
                                Temp1[i]=Temp1[j];
                                Temp1[j]=hole;
                        }
                        else Temp1[i]=Temp1[j];
                }
        }
        for(i=0;i<size;i++)
        {
                if(Temp2[i] == hole)
                {
                        for(j=i;j<size;j++)
                        {
                                if(Temp2[j] != hole) break;
                        }
                        if(j!=size)
                        {
                                Temp2[i]=Temp2[j];
                                Temp2[j]=hole;
                        }
                        else Temp2[i]=Temp2[j];
                }
        }
        i=0;
        for(j=interval2OX+1;j<size;j++)
        {
                ParentOX1[j]=Temp1[i];
                ParentOX2[j]=Temp2[i];
                i++;
```

```
        }
        for(j=0;j<=interval2OX;j++)
        {
                ParentOX1[j]=Temp1[i];
                ParentOX2[j]=Temp2[i];
                i++;
        }
        for(i=interval1OX;i<=interval2OX;i++)
        {
                ParentOX1[i]=TempM[i];
                ParentOX2[i]=TempF[i];
        }
        CopyItem(ParentOX1,0,ChromosomeBuffer,first);
        CopyItem(ParentOX2,0,ChromosomeBuffer,second);
//      ShowValueOX();
        free(ParentOX1);
        free(ParentOX2);
}

void CrossOver_OX(float Prob_OX)
{
        int i;
        unsigned char eos1,eos2;
        if(Prob_OX == 0) return;
        Temp1=(unsigned char *)malloc(sizeof(char)*size);
        Temp2=(unsigned char *)malloc(sizeof(char)*size);
        TempF=(unsigned char *)malloc(sizeof(char)*size);
        TempM=(unsigned char *)malloc(sizeof(char)*size);
        eos1=Temp1[size];
        eos2=Temp2[size];
        Temp1[size]=hole;
        Temp2[size]=hole;
        AllocRanDomProbOX();
        CountOX=0;
        for(i=0;i<population;i++)
                if(RanDomProbOX[i] <= Prob_OX) CountOX++;
//      fprintf(file,"CountOX = %d\n",CountOX);
        AllocIndexProbOX();
        SelectProbOX(Prob_OX);
//      fprintf(file,"\n");
        if(CountOX%2 == 1) CountOX--;
        for(i=0;i<CountOX;i+=2)
        {
//              fprintf(file,"[%d %d]\n",IndexProbOX[i],IndexProbOX[i+1]);
                CrossOX(IndexProbOX[i],IndexProbOX[i+1]);
        }
//      fprintf(file,"\n");
        Temp1[size]=eos1;
        Temp2[size]=eos2;
        UpdateCost();
        free(Temp1);
        free(Temp2);
        free(TempF);
        free(TempM);
        free(IndexProbOX);
        free(RanDomProbOX);
}

////////////////////////////////////////////////////////////////////////////////
// File CrossCX.cpp
// Last update 31/05/97
////////////////////////////////////////////////////////////////////////////////
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>

void CopyItem(unsigned char *Soure,int soure,unsigned char *Destination,int
destination);
void UpdateCost(void);

extern unsigned char *ChromosomeBuffer;       // Buffer for String
extern int     population;
extern int     size;
extern FILE *file;

void AllocParentCX(int first,int second);
```

```
void AllocRanDomProbCX(void);
void SelectProbCX(float prob);
void AllocIndexProbCX(void);
void ShowValueCX(void);
unsigned char *ParentCX1,*ParentCX2;
float  *RanDomProbCX;
int    *IndexProbCX;
int    CountCX;

void ShowValueCX(void)
{
        int i;
        for(i=0;i<size;i++)
                fprintf(file,"%d ",ParentCX1[i]);
        fprintf(file,"\n");
        for(i=0;i<size;i++)
                fprintf(file,"%d ",ParentCX2[i]);
        fprintf(file,"\n");
}

void AllocRanDomProbCX(void)
{
        int i;
        RanDomProbCX = (float *)malloc(sizeof(float)*population);
        if(RanDomProbCX == NULL)
        {
                printf("Not enough memmory for RanDomProbOX...\n");
                exit(0);
        }
//      srand( (unsigned)time( NULL ) );
        for(i=0;i<population;i++)
        {
                RanDomProbCX[i]=(float)(rand()*rand()%1000000)/(float)1000000;
//              if(i%6 == 5) fprintf(file,"\n");
//              fprintf(file,"%f\t",RanDomProbCX[i]);
        }
}

void AllocIndexProbCX(void)
{
        IndexProbCX=(int *)malloc(sizeof(int)*(CountCX+1));
        if(IndexProbCX == NULL)
        {
                printf("Not enough memmory for IndesCX...\n");
                exit(0);
        }
}

void AllocParentCX(int first,int second)
{
        ParentCX1=(unsigned char *)malloc(sizeof(char)*size);
        if(ParentCX1 == NULL)
        {
                printf("Not enough memmory for ParentCX1...\n");
                exit(0);
        }

        ParentCX2=(unsigned char *)malloc(sizeof(char)*size);
        if(ParentCX2 ==NULL)
        {
                printf("Not enough memmory for ParentCX2...\n");
                exit(0);
        }

        CopyItem(ChromosomeBuffer,first,ParentCX1,0);
        CopyItem(ChromosomeBuffer,second,ParentCX2,0);
}

void SelectProbCX(float prob)
{
        int i,j;
//      fprintf(file,"Select\n");
        j=0;
        for(i=0;i<population;i++)
        {
                if(RanDomProbCX[i] <= prob)
                {
```

```
                        IndexProbCX[j]=i;
                        j++;
//                      if(i==5) fprintf(file,"\n");
//                      fprintf(file,"%d\t",i);
                }
        }
//      fprintf(file,"\n\n");
}

void CrossCX(int first,int second)
{
        AllocParentCX(first,second);
        int i,mark=0;
        unsigned char first_value,swap_tmp,temp;
//      ShowValueCX();
        temp=ParentCX2[0];
        first_value=ParentCX1[0];
        for(i=0;i<size;i++)
        {
                if(ParentCX1[i]==temp && i != mark)
                {
                        swap_tmp=ParentCX1[i];
                        ParentCX1[i]=ParentCX2[i];
                        ParentCX2[i]=swap_tmp;
                        temp=ParentCX1[i];
                        mark=i;
                        i = -1;
                }
        }
        for(i=0;i<size;i++)
        {
                swap_tmp=ParentCX1[i];
                ParentCX1[i]=ParentCX2[i];
                ParentCX2[i]=swap_tmp;
        }
//      ShowValueCX();fprintf(file,"\n");
        CopyItem(ParentCX1,0,ChromosomeBuffer,first);
        CopyItem(ParentCX2,0,ChromosomeBuffer,second);
}

void CrossOver_CX(float Prob_CX)
{
        int i;
        if(Prob_CX == 0) return;
        CountCX=0;
        AllocRanDomProbCX();
        for(i=0;i<population;i++)
                if(RanDomProbCX[i] <= Prob_CX) CountCX++;
//      fprintf(file,"CountCX = %d\n",CountCX);
        AllocIndexProbCX();
        SelectProbCX(Prob_CX);
//      fprintf(file,"\n");
        if(CountCX%2 == 1) CountCX--;
        for(i=0;i<CountCX;i+=2)
        {
//              fprintf(file,"[%d %d]\n",IndexProbCX[i],IndexProbCX[i+1]);
                CrossCX(IndexProbCX[i],IndexProbCX[i+1]);
        }
//      fprintf(file,"\n");
        UpdateCost();
}

////////////////////////////////////////////////////////////////////////////////
// File Distance.cpp
// Last update 31/05/97
////////////////////////////////////////////////////////////////////////////////
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <malloc.h>
#include <string.h>
#include <time.h>

unsigned char *Chromosome,*ChromosomeBuffer,*ChromosomeTemp,*BestChromosome;       //
Buffer for String
signed int      *MemVij,*MemFij;                // Buffer for store Vij
float           Prob_PMX,Prob_OX,Prob_CX,Prob_M,mutat;
```

```
int             Row,Col,size;
float           *Cost,*CostBuffer,*CostTemp,BestCost;
int             population;
int             generation;
int             mark[20];
int             interval,total;
FILE            *file,*summary;

// Functions are not in this file.
void RandomToPlantLayout(void);
float Evaluate(void);
float CostMin(void);
float CostMax(void);
float CostAvg(void);
float CostSum(void);
double STD(void);
void  CostReproduct(void);
void  CrossOver_PMX(float Prob_PMX);
void  CrossOver_OX(float Prob_OX);
void  CrossOver_CX(float Prob_CX);
void  Mutate(float mutate);
void  Elitist(void);
// Functions are in this file.
void AllocDij(void);
void AllocVij(void);
void initial(void);
void AllocCostBuffer(void);
void AllocChromosome(void);
void SavePlantLayout(int pop);
void SaveToFile(int Generation);
void ShowChromosome(void);
void SaveBest(void);
void UpdateCost(void);
void SummaryToFile(int gen);
int  NeverMark(int end,int key);

int  NeverMark(int end,int key)
{
        int i;
        for(i=0;i<end;i++)
                if(mark[i] == key) return 0;
        return 1;
}

void SummaryToFile(int gen)
{
        int i,j,k;
        float min;
        fprintf(summary,"[Generation %d]\n",gen);

        min=CostMax();
        for(i=0;i<total;i++)
        {
                for(k=0;k<population;k++)
                {
                        if(CostBuffer[k]<=min && NeverMark(i,k))
                        {
                                min=CostBuffer[k];
                                mark[i]=k;
                        }
                }

                fprintf(summary,"[%d]\t",mark[i]);
                for(j=0;j<size;j++)
                        fprintf(summary,"%d ",ChromosomeBuffer[size*mark[i]+j]);
                fprintf(summary," =\t%f\n",CostBuffer[mark[i]]);
                min=CostMax();
        }
        fprintf(summary,"\nThe Best Cost is : %f\t",BestCost);
        fprintf(summary,"[ ");
        for(i=0;i<size;i++)
                fprintf(summary,"%d ",BestChromosome[i]);
        fprintf(summary,"]\n");
        fprintf(summary,"Averang is : %f\n",CostAvg());
        fprintf(summary,"STD     is : %f\n\n",STD());
}
```

```
void UpdateCost(void)
{
        int i,j;
        for(i=0;i<population;i++)
        {
                for(j=0;j<size;j++)
                        Chromosome[j]=ChromosomeBuffer[i*size+j];
                CostBuffer[i]=Evaluate();
        }
}

void ShowChromosome(void)
{
        int i;
/*
        for(i=0;i<population;i++)
        {
                fprintf(file,"[%d]\t",i);
                for(j=0;j<size;j++)
                        fprintf(file,"%d ",ChromosomeBuffer[size*i+j]);
                fprintf(file," =\t%f\n",CostBuffer[i]);
        }
*/
        fprintf(file,"%f\t",CostMin());
        fprintf(file,"%f\t",CostMax());
        fprintf(file,"%f\t",CostAvg());
        fprintf(file,"%f\t",CostSum());
        fprintf(file,"%f\t",STD());
//      fprintf(file,"The Best Cost is = %f\n",BestCost);
        fprintf(file,"[ ");
        for(i=0;i<size;i++)
                fprintf(file,"%d ",BestChromosome[i]);
        fprintf(file,"]\n");
}

void AllocDij(void)    // Allocate memory for construct string
{
        Chromosome = (unsigned char *)malloc(size*sizeof(char));
        if(Chromosome == NULL)
        {
                printf("Not enough memmory for \"Chromosome\"\n");
                exit(0);
        }
}

// Allocate memmory for weight table and input weight in term continue
// Input : None
// Output: Enter value in pointer MemVij
void AllocVij(void)
{
        FILE *frequency,*cost;
        char buffer[5];
        int  weight,i;
        frequency=fopen("frequency.dat","rt");
        if(frequency == NULL)
        {
                printf("Can not open file frequency.dat\n");
                exit(0);
        }
        cost=fopen("cost.dat","rt");
        if(cost == NULL)
        {
                printf("Can not open file cost.dat\n");
                exit(0);
        }
        fgets(buffer,5,frequency);
        sscanf(buffer,"%d",&Row);      // Get ROW
        fgets(buffer,5,frequency);
        sscanf(buffer,"%d",&Col);      // Get COL
        size=Row*Col;
        weight = size*size;                             // Find total of weight
        printf("weight = %d\n",weight);
        MemVij = (int *)malloc(weight*sizeof(weight));// Alloc memmory for input weight
        if(MemVij == NULL)
        {
                printf("Not enough memmory for \"MemVij\"\n");
                exit(0);
```

```
        }
        MemFij = (int *)malloc(weight*sizeof(weight));// Alloc memmory for input weight
        if(MemVij == NULL)
        {
                printf("Not enough memmory for \"MemVij\"\n");
                exit(0);
        }
        for(i=0;i<weight;i++)
        {
                fgets(buffer,5,frequency);
                sscanf(buffer,"%d",&MemVij[i]);
        }
        for(i=0;i<weight;i++)
        {
                fgets(buffer,5,cost);
                sscanf(buffer,"%d",&MemFij[i]);
        }
        printf("MemVij is\n");
        for(i=0;i<weight;i++)
                printf("%d ",MemVij[i]);
        printf("\n");
        printf("MemVij is\n");
        for(i=0;i<weight;i++)
                printf("%d ",MemFij[i]);
        printf("\n");
        fclose(frequency);
        fclose(cost);
}

void AllocCostBuffer(void)
{
        CostBuffer=(float *)malloc(sizeof(float)*population);
        if(CostBuffer == NULL)
        {
                printf("Not enough memmory for \"CostBuffer\"\n");
                exit(0);
        }
        CostTemp=(float *)malloc(sizeof(float)*population);
        if(CostTemp == NULL)
        {
                printf("Not enough memmory for \"CostTemp\"\n");
                exit(0);
        }
}

void AllocChromosomeBuffer(void)
{
        ChromosomeBuffer=(unsigned char *)malloc(size*sizeof(char)*population);
        if(ChromosomeBuffer == NULL)
        {
                printf("Not enough memmory for \"ChromosomeBuffer\"\n");
                exit(0);
        }
        ChromosomeTemp=(unsigned char *)malloc(size*sizeof(char)*population);
        if(ChromosomeTemp == NULL)
        {
                printf("Not enough memmory for \"ChromosomeTemp\"\n");
                exit(0);
        }
        BestChromosome=(unsigned char *)malloc(size*sizeof(char));
        if(ChromosomeTemp == NULL)
        {
                printf("Not enough memmory for \"Best Chromosome\"\n");
                exit(0);
        }
}

void SavePlantLayout(int pop)
{
        int i;
        for(i=0;i<size;i++)
                ChromosomeBuffer[pop*size+i]=Chromosome[i];
}

void SaveToFile(int Generation)
{
        fclose(file);
```

```
        fclose(summary);
}

void initial(void)
{
        int i;
        printf("Enter Generation : ");scanf("%d",&generation);
        printf("Enter Population : ");scanf("%d",&population);
        printf("Enter Probability PMX : ");scanf("%f",&Prob_PMX);
        printf("Enter Probability OX  : ");scanf("%f",&Prob_OX);
        printf("Enter Probability CX  : ");scanf("%f",&Prob_CX);
        printf("Enter Mutate : ");scanf("%f",&mutat);
        printf("How many period for report : ");scanf("%d",&interval);
        printf("How many report chromosome : ");scanf("%d",&total);
        AllocVij();             // Build relationship chart table
        AllocCostBuffer();
        AllocDij();             // Build plant layout
        AllocChromosomeBuffer();
        file=fopen("result.txt","wt");
        if(file == NULL)
        {
                printf("Can not open file \"RESULT.TXT\" \n");
                exit(0);
        }
        summary=fopen("summary.txt","wt");
        if(file == NULL)
        {
                printf("Can not open file \"SUMMARY.TXT\" \n");
                exit(0);
        }
//      srand( (unsigned)time( NULL ) );
        for(i=0;i<population;i++)
        {
                RandomToPlantLayout();
                SavePlantLayout(i);
                CostBuffer[i]=Evaluate();
        }
}

void main(void)
{
        int i;
        clock_t start, finish;
        double  use;
        char *version="Program Plant Layout version 1.02 for Distance\nLast update
14/09/97\n";
        printf("%s",version);  // Show version
        initial();
        fprintf(file,"%s",version);
        start=clock();
        BestCost=CostMax();
        for(i=0;i<generation;i++)
        {
                SaveBest();
                CostReproduct();
                SaveBest();
                CrossOver_PMX(Prob_PMX);
                SaveBest();
                CrossOver_OX(Prob_OX);
                SaveBest();
                CrossOver_CX(Prob_CX);
                SaveBest();
                Mutate(mutat);
                SaveBest();
                Elitist();
                ShowChromosome();
                printf("Generation %d Completed\n",i);
                if(i%interval == 0) SummaryToFile(i);
        }
        SaveToFile(i);
        finish=clock();
        use=(double)(finish-start)/CLOCKS_PER_SEC;
        printf("Function sucessfully used time = %2.2f sec.\n",use);
        free(BestChromosome);
        free(ChromosomeBuffer);
        free(CostBuffer);
        free(ChromosomeTemp);
```

```
        free(CostTemp);
        free(MemVij);
}

////////////////////////////////////////////////////////////////////////////////
// File Elitist.cpp
// Last update 31/05/97
////////////////////////////////////////////////////////////////////////////////
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

float  CostMin(void);
void   CopyItem(unsigned char *Soure,int soure,unsigned char *Destination,int
destination);
float  CostMax(void);
void   UpdateCost(void);
void   SaveBest(void);

extern unsigned char *Chromosome,*ChromosomeBuffer,*BestChromosome; // Buffer for
String
extern float            *CostBuffer,BestCost;
extern int                      population;
extern int                      size;
extern FILE *file;

void SaveBest(void)
{
        int i;
        if(CostMin() < BestCost)
        {
                BestCost=CostMin();
                for(i=0;i<population;i++)
                        if(CostBuffer[i]==BestCost) break;
                CopyItem(ChromosomeBuffer,i,BestChromosome,0);
        }
}

void ReplaceBestChromosome(void)
{
        int i;
        float BadCost;
        BadCost=CostMax();
        for(i=0;i<population;i++)
                if(CostBuffer[i]==BadCost) break;
        CopyItem(BestChromosome,0,ChromosomeBuffer,i);
}

void Elitist(void)
{
        ReplaceBestChromosome();
        UpdateCost();
}

////////////////////////////////////////////////////////////////////////////////
// File Mutation.cpp
// Last update 31/05/97
////////////////////////////////////////////////////////////////////////////////
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>

void UpdateCost();
extern int size;
extern int population;
extern unsigned char *ChromosomeBuffer;
extern FILE *file;
void InitialMutate(void);
void RandomMutate(void);
void SelectMutate(void);
void Mutation(int position);
void ShowChromosome(int pop,int siz,int swaper);
void Mutate(float mutat);
float *Gene,mutate;

void InitialMutate(void)
```

```
{
        Gene=(float *)malloc(sizeof(float)*size*population);
        if(Gene == NULL)
        {
                printf("Not enough memory for Gene\n");
                exit(0);
        }
}

/*
void RandomChromosome(void)
{
        int Size,Population;
        for(Population=0;Population<population;Population++)
        {
                for(Size=0;Size<size;Size++)
                {
                        do
                        {
                                ChromosomeBuffer[(Population*size)+Size]=rand()%size;
                        } while(Redundant(Population,Size));
                }
        }
        for(Population=0;Population<population;Population++)
        {
                printf("Level %d \t[ ",Population);
                for(Size=0;Size<size;Size++)
                {
                        printf("%d ",ChromosomeBuffer[(Population*size)+Size]);
                }
                printf("]\n");
        }
}

int Redundant(int pop,int siz)
{
        int i;
        for(i=0;i<siz;i++)
        {
                if(ChromosomeBuffer[(pop*size)+siz] == ChromosomeBuffer[(pop*size)+i])
                        return 1;
        }
        return 0;
}

*/

void RandomMutate(void)
{
        int i;
//      srand( (unsigned)time( NULL ) );
        for(i=0;i<population*size;i++)
        {
                Gene[i]=(float)(rand()*rand()%1000000)/(float)1000000;
//              if(i%6 == 5) fprintf(file,"\n");
//              fprintf(file,"%f\t",Gene[i]);
        }
}

void SelectMutate(void)
{
        int i;
//      fprintf(file,"\nSelect Mutate\n");
        for(i=0;i<population*size;i++)
        {
                if(Gene[i] <= mutate)
                {
                        Mutation(i);
                }
        }
}

void Mutation(int position)
{
        int pop,siz;
        int swaper;
        unsigned char temp;
```

```
        pop = position/size;
        siz = position%size;
//      fprintf(file,"[Expected Mutation=%d]\tPercent mutate = %f\n   Chromosome  =
%d\n   Logus = %d\n",position,Gene[position],pop,siz);
        swaper=size;
//      ShowChromosome(pop,siz,swaper);
//      srand( (unsigned)time( NULL ) );
        do
        {
                swaper=rand()%size;
        } while(swaper == siz);
//      ShowChromosome(pop,siz,swaper);
        temp=ChromosomeBuffer[(pop*size)+siz];
        ChromosomeBuffer[(pop*size)+siz] = ChromosomeBuffer[(pop*size)+swaper];
        ChromosomeBuffer[(pop*size)+swaper] = temp;
//      ShowChromosome(pop,siz,swaper);
}

void ShowChromosome(int pop,int siz,int swaper)
{
        int i;
//      fprintf(file,"Chromosome %d \t[",pop);
        for(i=0;i<size;i++)
        {
                if(i == siz || i == swaper) fprintf(file,"|");
                else fprintf(file," ");
                fprintf(file,"%d",ChromosomeBuffer[(pop*size )+i]);
                if(i == siz || i == swaper) fprintf(file,"|");
                else fprintf(file," ");
        }
        fprintf(file,"]\n");
}

void Mutate(float mutat)
{
        mutate=mutat;
        InitialMutate();
        RandomMutate();
        SelectMutate();
        UpdateCost();
        free(Gene);
}

////////////////////////////////////////////////////////////////////////////////
// File Reproduct.cpp
// Last update 31/05/97
////////////////////////////////////////////////////////////////////////////////
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <time.h>

extern unsigned char *ChromosomeBuffer,*ChromosomeTemp;     // Buffer for String
extern int           population,size;
extern float            *CostBuffer,*CostTemp;
extern FILE *file;
// Function are not in this file
float  CostSum(void);
void   UpdateCost(void);
// Function are in this file
void AllocIndex(void);
void AllocProb(void);
void AllocCumulative(void);
void CostReproduct(void);
void CopyItem(unsigned char *Soure,int soure,unsigned char *Destination,int
destination);

int     *Index;
float   *Prob,*Cumulative;

void AllocIndex(void)
{
        Index=(int *)malloc(population*sizeof(int));
        if(Index == NULL)
        {
                printf("Not enough memmory for Index...\n");
```

```
                exit(0);
        }
}

void AllocProb(void)
{
        Prob=(float *)malloc(population*sizeof(float));
        if(Prob == NULL)
        {
                printf("Notenough memmory for Prob...\n");
                exit(0);
        }
}

void AllocCumulative(void)
{
        Cumulative=(float *)malloc(population*sizeof(float));
        if(Cumulative == NULL)
        {
                printf("Not enough memmory for Cumulative...\n");
                exit(0);
        }
}

void CopyItem(unsigned char *Soure,int soure,unsigned char *Destination,int
destination)
{
        int i;
        for(i=0;i<size;i++)
                Destination[destination*size+i]=Soure[soure*size+i];
        if(Soure == ChromosomeTemp)
                CostBuffer[destination]=CostTemp[soure];
        if(Soure == ChromosomeBuffer)
                CostTemp[destination]=CostBuffer[soure];
}


void CostReproduct(void)
{
        float sum,roulette;
        int   i,j;
        AllocIndex();
        AllocProb();
        AllocCumulative();
        sum=CostSum();
        for(i=0;i<population;i++)
                Cumulative[i] = sum-CostBuffer[i];
        sum=(float)0;
        for(i=0;i<population;i++)
                sum+=Cumulative[i];
        for(i=0;i<population;i++)
                Prob[i]=Cumulative[i]/sum;
        Cumulative[0]=Prob[0];
        for(i=1;i<population;i++)
                Cumulative[i]=Prob[i]+Cumulative[i-1];
        for(j=0;j<population;j++)
        {
                roulette=(float)(rand()*rand()%1000000)/(float)1000000;
                for(i=0;i<population;i++)
                {
                        if(roulette > Cumulative[i] && roulette <= Cumulative[i+1])
                        {
                                Index[j]=i+1;
                                break;
                        }
                        if(roulette <= Cumulative[0])
                        {
                                Index[j]=0;
                                break;
                        }
                }
        }
        for(i=0;i<population;i++)
                CopyItem(ChromosomeBuffer,Index[i],ChromosomeTemp,i);
        for(i=0;i<population;i++)
                CopyItem(ChromosomeTemp,i,ChromosomeBuffer,i);
        UpdateCost();
```

```
        free(Cumulative);
        free(Prob);
        free(Index);
}
```

# ภาคผนวก ญ
# รายละเอียดไฟล์ตัวหนังสือของโปรแกรม Quannit

เนื้อหารายละเอียดในส่วนนี้จะเสนอถึงรายละเอียดไฟล์ตัวหนังสือของโปรแกรม Quaninit หรือโปรแกรมการแก้ปัญหาการจัดผังโรงงานโดยพิจารณาถึงข้อมูลเชิงปริมาณที่สามารถให้คำตอบเริ่มต้นได้ ด้วย GAs ซึ่งประกอบไปด้วยไฟล์จำนวนทั้งสิ้น 9 ไฟล์ ได้แก่

1. Genetic.cpp
2. CostSummary.cpp
3. CrossPMX.cpp
4. CrossOX.cpp
5. CrossCX.cpp
6. Distance.cpp
7. Elitist.cpp
8. Mutation.cpp
9. Reproduct.cpp

ดังมีรายละเอียดดังต่อไปนี้

```
////////////////////////////////////////////////////////////////////////////////
// File Genetic.cpp
// Last update 31/05/97
////////////////////////////////////////////////////////////////////////////////
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

extern unsigned char *Chromosome;
extern signed int    *MemVij,*MemFij;
extern int           Col,Row,size;
int RandomNumber(int interval);
void RandomToPlantLayout(void);
float Evaluate(void);
float Dij(int i,int j);

int RandomNumber(int interval) // interval of random value (0-interval)
{
       return rand()%interval;
}

void RandomToPlantLayout(void)
{
       int temp;
       int attrib;
       int i;
       for(i=0;i<size;i++)
       {
              if(i == 0)
              {
                     temp=RandomNumber(size);
                     Chromosome[i] = (char)temp;
              }
              else
              {
```

```
                        temp=RandomNumber(size);
                        for(attrib=0;attrib<i;attrib++)
                        {
                                if(Chromosome[attrib] == (unsigned char)temp)
                                {
                                        temp=RandomNumber(size);
                                        attrib = -1;
                                }
                        if(Chromosome[attrib] != (unsigned char)temp && attrib == i-1)
                        break;
                        }
                        Chromosome[attrib+1] = (unsigned char)temp;
                }
        }
}

float Evaluate(void)
{
        int   i,j,N=0;
        float cost;
        cost=(float)0;
        for(i=0;i<size;i++)
        {
                for(j=0;j<size;j++)
                {
                        cost+=(Dij(i,j)*(float)MemVij[N]*(float)MemFij[N]);
                        N++;
                }
        }
        return cost;
}

float Dij(int i,int j)
{
        int x1,x2,y1,y2;
        int N,M;
        for(N=0;N<size;N++)
                if(Chromosome[N]==i) break;
        for(M=0;M<size;M++)
                if(Chromosome[M]==j) break;
        x1 = N/Col;
        y1 = N%Col;
        x2 = M/Col;
        y2 = M%Col;
        return (float)sqrt(pow((x1-x2),2)+pow((y1-y2),2));
}

////////////////////////////////////////////////////////////////////////////////
// File CostSummary.cpp
// Last update 31/05/97
////////////////////////////////////////////////////////////////////////////////
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <time.h>
extern unsigned char   *ChromosomeBuffer;     // Buffer for String
extern int             population,size;
extern float           *CostBuffer;
extern int             generation;
float  CostMin(void);
float  CostMax(void);
float  CostAvg(void);
float  CostSum(void);
double STD(void);

float CostSum(void)
{
        int   i;
        float sum;
        for(i=0;i<population;i++)
                sum+=CostBuffer[i];
        return sum;
}

float CostMin(void)
{
```

```
        int    i;
        float  min;
        min=CostBuffer[0];
        for(i=0;i<population;i++)
                if(min > CostBuffer[i])
                        min=CostBuffer[i];
        return min;
}

float CostMax(void)
{
        int    i,j;
        float  max;
        max=CostBuffer[0];
        for(i=0;i<population;i++)
                if(max < CostBuffer[i])
                {
                        max=CostBuffer[i];
                        j=i;
                }
        return max;
}

float CostAvg(void)
{
        return CostSum()/(float)population;
}

double STD(void)
{
        int i;
        double avg;
        double std=(double)0;
        avg=CostAvg();
        for(i=0;i<population;i++)
                std+=pow((double)(CostBuffer[i]-avg),(double)2);
        return sqrt((double)std/(double)(population-1));
}

////////////////////////////////////////////////////////////////////////////////
// File CrossPMX.cpp
// Last update 31/05/97
////////////////////////////////////////////////////////////////////////////////
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>

// Function are not is this file
void CopyItem(unsigned char *Soure,int soure,unsigned char *Destination,int
destination);
void UpdateCost(void);
extern unsigned char *ChromosomeBuffer;      // Buffer for String
extern int                    population;
extern int                    size;
extern FILE *file;
// Function are in this file
void AllocRanDomProbPMX(void);
void AllocIndexProbPMX(void);
void CrossOver_PMX(float Prob_PMX);
void CrossPMX(int first,int second);
void Map(unsigned char *Temp1,int i,unsigned char *Temp2,int j);
void ShowValuePMX(void);
void AllocParentPMX(int first,int second);
void DefineInterval(void);
void SelectProbPMX(float prob);
float  *RanDomProb;
int        *IndexProb;
int        count;
unsigned char *Parent1 ,*Parent2,temp;
int        interval1,interval2;

void AllocRanDomProbPMX(void)
{
        int i;
        RanDomProb = (float *)malloc(sizeof(float)*population);
        if(RanDomProb == NULL)
```

```
        {
                printf("Not enough memmory for RanDomProb...\n");
                exit(0);
        }
//      srand( (unsigned)time( NULL ) );
        for(i=0;i<population;i++)
        {
                RanDomProb[i]=(float)(rand()*rand()%1000000)/(float)1000000;
//              fprintf(file,"%f\t",RanDomProb[i]);
        }
}

void AllocIndexProbPMX(void)
{
        IndexProb=(int *)malloc(sizeof(int)*(count+1));
        if(IndexProb == NULL)
        {
                printf("Not enough memmory for IndesPMX...\n");
                exit(0);
        }
}

void Map(unsigned char *Temp1,int i,unsigned char *Temp2,int j)
{
        Temp1[i]=Temp2[j];
}

void ShowValuePMX(void)
{
        int i;
        fprintf(file,"[ ");
        for(i=0;i<size;i++)
        {
                if(i== interval1 || i == interval2+1)
                        fprintf(file,"| ");
                fprintf(file,"%d ",Parent1[i]);
        }
        fprintf(file,"]\n");
        fprintf(file,"[ ");
        for(i=0;i<size;i++)
        {
                if(i == interval1 || i == interval2+1)
                        fprintf(file,"| ");
                fprintf(file,"%d ",Parent2[i]);
        }
        fprintf(file,"]\n\n");
}

void AllocParentPMX(int first,int second)
{
        Parent1=(unsigned char *)malloc(sizeof(char)*size);
        if(Parent1 == NULL)
        {
                printf("Not encugh memmory for Parent1...\n");
                exit(0);
        }
        Parent2=(unsigned char *)malloc(sizeof(char)*size);
        if(Parent2 ==NULL)
        {
                printf("Not enough memmory for Parent2...\n");
                exit(0);
        }
        CopyItem(ChromosomeBuffer,first,Parent1,0);
        CopyItem(ChromosomeBuffer,second,Parent2,0);
}

void DefineInterval(void)
{
        do
                interval1=rand()%size;
        while(interval1 < 1 || interval1 >= size-2);
//      fprintf(file,"Interval 1 = %d\n",interval1);
        do
                interval2=rand()%size;
        while(interval2 <= interval1 || interval2 > size-2);
//      fprintf(file,"Interval 2 = %d\n",interval2);
}
```

```
void CrossPMX(int first,int second)
{
	int	i,j;
	AllocParentPMX(first,second);
	DefineInterval();
//	ShowValuePMX();
	for(i=interval1;i<=interval2;i++)
	{
		temp=Parent1[i];
		Parent1[i]=Parent2[i];
		Parent2[i]=temp;
	}
	for(i=0;i<interval1;i++)
	{
		for(j=interval1;j<=interval2;j++)
		{
			if(Parent1[i]==Parent1[j])
			{
				Map(Parent1,i,Parent2,j);
				j=interval1-1;
			}
		}
		for(j=interval1;j<=interval2;j++)
		{
			if(Parent2[i]==Parent2[j])
			{
				Map(Parent2,i,Parent1,j);
				j=interval1-1;
			}
		}
	}
	for(i=interval2+1;i<size;i++)
	{
		for(j=interval1;j<=interval2;j++)
		{
			if(Parent1[i]==Parent1[j])
			{
				Map(Parent1,i,Parent2,j);
				j=interval1-1;
			}
		}
		for(j=interval1;j<=interval2;j++)
		{
			if(Parent2[i]==Parent2[j])
			{
				Map(Parent2,i,Parent1,j);
				j=interval1-1;
			}
		}
	}
	CopyItem(Parent1,0,ChromosomeBuffer,first);
	CopyItem(Parent2,0,ChromosomeBuffer,second);
//	ShowValuePMX();
	free(Parent1);
	free(Parent2);
}

void SelectProbPMX(float prob)
{
	int i,j;
//	fprintf(file,"Select .....\n");
	j=0;
	for(i=0;i<population;i++)
	{
		if(RanDomProb[i] <= prob)
		{
			IndexProb[j]=i;
			j++;
//			if(i%6 == 5) fprintf(file,"\n");fprintf(file,"%d\t",i);
		}
	}
}

void CrossOver_PMX(float Prob_PMX)
{
	int	i;
```

```
        if(Prob_PMX == 0) return;
        AllocRanDomProbPMX();
        count=0;
        for(i=0;i<population;i++)
                if(RanDomProb[i] <= Prob_PMX) count++;
//      fprintf(file,"\nCount = %d\n",count);
        AllocIndexProbPMX();
        SelectProbPMX(Prob_PMX);
//      fprintf(file,"\n");
        if(count%2 == 1) count--;
        for(i=0;i<count;i+=2)
        {
//              fprintf(file,"[%d %d]\n",IndexProb[i],IndexProb[i+1]);
                CrossPMX(IndexProb[i],IndexProb[i+1]);
        }
        UpdateCost();
        free(IndexProb);
        free(RanDomProb);
}

////////////////////////////////////////////////////////////////////////////////
// File CrossOX.cpp
// Last update 31/05/97
////////////////////////////////////////////////////////////////////////////////
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>

extern unsigned char *ChromosomeBuffer;       // Buffer for String
extern int size;
extern int population;
extern FILE *file;
void CopyItem(unsigned char *Soure,int soure,unsigned char *Destination,int
destination);
void UpdateCost(void);
unsigned char *ParentOX1,*ParentOX2;
int interval1OX,interval2OX;
float  *RanDomProbOX;
int    *IndexProbOX;
int    CountOX;
char   hole='*';
unsigned char *Temp1,*Temp2,*TempF,*TempM;
void ShowValueOX(void);
void CrossOX(void);
void AllocRanDomProbOX();
void DefineIntervalOX(void);
void AllocParentOX(int first,int second);
void SelectProbOX(float prob);
void AllocIndexProbOX();

void ShowValueOX(void)
{
        int i;
        fprintf(file,"[ ");
        for(i=0;i<size;i++)
        {
                if(i== interval1OX || i == interval2OX+1)
                        fprintf(file,"| ");
                if(ParentOX1[i] == hole) fprintf(file,"# ");
                else fprintf(file,"%d ",ParentOX1[i]);
        }
        fprintf(file,"]\n");
        fprintf(file,"[ ");
        for(i=0;i<size;i++)
        {
                if(i == interval1OX || i == interval2OX+1)
                        fprintf(file,"| ");
                if(ParentOX2[i] == hole) fprintf(file,"# ");
                else fprintf(file,"%d ",ParentOX2[i]);
        }
        fprintf(file,"]\n");
}

void AllocRanDomProbOX(void)
{
        int i;
```

```
        RanDomProbOX = (float *)malloc(sizeof(float)*population);
        if(RanDomProbOX == NULL)
        {
                printf("Not enough memmory for RanDomProbOX...\n");
                exit(0);
        }
//      srand( (unsigned)time( NULL ) );
        for(i=0;i<population;i++)
        {
                RanDomProbOX[i]=(float)(rand()*rand()%1000000)/(float)1000000;
//              if(i == 5) fprintf(file,"\n");
//              fprintf(file,"%f\t",RanDomProbOX[i]);
        }
}

void AllocIndexProbOX(void)
{
        IndexProbOX=(int *)malloc(sizeof(int)*(CountOX+1));
        if(IndexProbOX == NULL)
        {
                printf("Not enough memmory for IndesOX...\n");
                exit(0);
        }
}

void AllocParentOX(int first,int second)
{
        ParentOX1=(unsigned char *)malloc(sizeof(char)*size);
        if(ParentOX1 == NULL)
        {
                printf("Not enough memmory for ParentOX1...\n");
                exit(0);
        }
        ParentOX2=(unsigned char *)malloc(sizeof(char)*size);
        if(ParentOX2 ==NULL)
        {
                printf("Not enough memmory for ParentOX2...\n");
                exit(0);
        }
        CopyItem(ChromosomeBuffer,first,ParentOX1,0);
        CopyItem(ChromosomeBuffer,second,ParentOX2,0);
}

void DefineIntervalOX(void)
{
        do
                interval1OX=rand()%size;
        while(interval1OX < 1 || interval1OX >= size-2);
//      fprintf(file,"IntervalOX 1 = %d\n",interval1OX);
        do
                interval2OX=rand()%size;
        while(interval2OX <= interval1OX || interval2OX > size-2);
//      fprintf(file,"IntervalOX 2 = %d\n",interval2OX);
}

void SelectProbOX(float prob)
{
        int i,j;
//      fprintf(file,"\nSelect\n");
        j=0;
        for(i=0;i<population;i++)
        {
                if(RanDomProbOX[i] <= prob)
                {
                        IndexProbOX[j]=i;
                        j++;
//                      if(i%6 ==5) fprintf(file,"\n");
//                      fprintf(file,"%d\t",i);
                }
        }
//      fprintf(file,"\n");
}

void CrossOX(int first,int second)
{
        int i,j;
        AllocParentOX(first,second);
```

```
        DefineIntervalOX();
//      ShowValueOX();
//      fprintf(file,"-------------------------\n");
        for(i=0;i<size;i++)
        {
                TempF[i]=Temp1[i]=ParentOX1[i];
                TempM[i]=Temp2[i]=ParentOX2[i];
        }
        for(i=0;i<size;i++)
        {
                for(j=interval1OX;j<=interval2OX;j++)
                if(ParentOX1[i]==Temp2[j]) ParentOX1[i]=hole;        //  Hole = 0xff
                for(j=interval1OX;j<=interval2OX;j++)
                if(ParentOX2[i]==Temp1[j]) ParentOX2[i]=hole;        //  Hole = 0xff
        }
//      ShowValueOX();
//      fprintf(file,"-------------------------\n");
        i=0;
        for(j=interval2OX+1;j<size;j++)
        {
                Temp1[i]=ParentOX1[j];
                Temp2[i]=ParentOX2[j];
                i++;
        }
        for(j=0;j<=interval2OX;j++)
        {
                Temp1[i]=ParentOX1[j];
                Temp2[i]=ParentOX2[j];
                i++;
        }
        for(i=0;i<size;i++)
        {
                if(Temp1[i] == hole)
                {
                        for(j=i;j<size;j++)
                        {
                                if(Temp1[j] != hole) break;
                        }

                        if(j!=size)
                        {
                                Temp1[i]=Temp1[j];
                                Temp1[j]=hole;
                        }
                        else Temp1[i]=Temp1[j];
                }
        }
        for(i=0;i<size;i++)
        {
                if(Temp2[i] == hole)
                {
                        for(j=i;j<size;j++)
                        {
                                if(Temp2[j] != hole) break;
                        }
                        if(j!=size)
                        {
                                Temp2[i]=Temp2[j];
                                Temp2[j]=hole;
                        }
                        else Temp2[i]=Temp2[j];
                }
        }
        i=0;
        for(j=interval2OX+1;j<size;j++)
        {
                ParentOX1[j]=Temp1[i];
                ParentOX2[j]=Temp2[i];
                i++;
        }
        for(j=0;j<=interval2OX;j++)
        {
                ParentOX1[j]=Temp1[i];
                ParentOX2[j]=Temp2[i];
                i++;
        }
        for(i=interval1OX;i<=interval2OX;i++)
```

```
        {
                ParentOX1[i]=TempM[i];
                ParentOX2[i]=TempF[i];
        }
        CopyItem(ParentOX1,0,ChromosomeBuffer,first);
        CopyItem(ParentOX2,0,ChromosomeBuffer,second);
//      ShowValueOX();
        free(ParentOX1);
        free(ParentOX2);
}

void CrossOver_OX(float Prob_OX)
{
        int i;
        unsigned char eos1,eos2;
        if(Prob_OX == 0) return;
        Temp1=(unsigned char *)malloc(sizeof(char)*size);
        Temp2=(unsigned char *)malloc(sizeof(char)*size);
        TempF=(unsigned char *)malloc(sizeof(char)*size);
        TempM=(unsigned char *)malloc(sizeof(char)*size);
        eos1=Temp1[size];
        eos2=Temp2[size];
        Temp1[size]=hole;
        Temp2[size]=hole;
        AllocRanDomProbOX();
        CountOX=0;
        for(i=0;i<population;i++)
                if(RanDomProbOX[i] <= Prob_OX) CountOX++;
//      fprintf(file,"CountOX = %d\n",CountOX);
        AllocIndexProbOX();
        SelectProbOX(Prob_OX);
//      fprintf(file,"\n");
        if(CountOX%2 == 1) CountOX--;
        for(i=0;i<CountOX;i+=2)
        {
//              fprintf(file,"[%d %d]\n",IndexProbOX[i],IndexProbOX[i+1]);
                CrossOX(IndexProbOX[i],IndexProbOX[i+1]);
        }
//      fprintf(file,"\n");
        Temp1[size]=eos1;
        Temp2[size]=eos2;
        UpdateCost();
        free(Temp1);
        free(Temp2);
        free(TempF);
        free(TempM);
        free(IndexProbOX);
        free(RanDomProbOX);
}

////////////////////////////////////////////////////////////////////////////////
// File CrossCX.cpp
// Last update 31/05/97
////////////////////////////////////////////////////////////////////////////////
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>

void CopyItem(unsigned char *Soure,int soure,unsigned char *Destination,int
destination);
void UpdateCost(void);
extern unsigned char *ChromosomeBuffer;      // Buffer for String
extern int                    population;
extern int                    size;
extern FILE *file;
void AllocParentCX(int first,int second);
void AllocRanDomProbCX(void);
void SelectProbCX(float prob);
void AllocIndexProbCX(void);
void ShowValueCX(void);
unsigned char *ParentCX1,*ParentCX2;
float  *RanDomProbCX;
int    *IndexProbCX;
int    CountCX;

void ShowValueCX(void)
```

```
{
        int i;
        for(i=0;i<size;i++)
                fprintf(file,"%d ",ParentCX1[i]);
        fprintf(file,"\n");
        for(i=0;i<size;i++)
                fprintf(file,"%d ",ParentCX2[i]);
        fprintf(file,"\n");
}

void AllocRanDomProbCX(void)
{
        int i;
        RanDomProbCX = (float *)malloc(sizeof(float)*population);
        if(RanDomProbCX == NULL)
        {
                printf("Not enough memmory for RanDomProbOX...\n");
                exit(0);
        }
//      srand( (unsigned)time( NULL ) );
        for(i=0;i<population;i++)
        {
                RanDomProbCX[i]=(float)(rand()*rand()%1000000)/(float)1000000;
//              if(i%6 == 5) fprintf(file,"\n");
//              fprintf(file,"%f\t",RanDomProbCX[i]);
        }
}

void AllocIndexProbCX(void)
{
        IndexProbCX=(int *)malloc(sizeof(int)*(CountCX+1));
        if(IndexProbCX == NULL)
        {
                printf("Not enough memmory for IndesCX...\n");
                exit(0);
        }
}

void AllocParentCX(int first,int second)
{
        ParentCX1=(unsigned char *)malloc(sizeof(char)*size);
        if(ParentCX1 == NULL)
        {
                printf("Not enough memmory for ParentCX1...\n");
                exit(0);
        }
        ParentCX2=(unsigned char *)malloc(sizeof(char)*size);
        if(ParentCX2 ==NULL)
        {
                printf("Not enough memmory for ParentCX2...\n");
                exit(0);
        }
        CopyItem(ChromosomeBuffer,first,ParentCX1,0);
        CopyItem(ChromosomeBuffer,second,ParentCX2,0);
}

void SelectProbCX(float prob)
{
        int i,j;
//      fprintf(file,"Select\n");
        j=0;
        for(i=0;i<population;i++)
        {
                if(RanDomProbCX[i] <= prob)
                {
                        IndexProbCX[j]=i;
                        j++;
//                      if(i==5) fprintf(file,"\n");
//                      fprintf(file,"%d\t",i);
                }
        }
//      fprintf(file,"\n\n");
}

void CrossCX(int first,int second)
{
        AllocParentCX(first,second);
```

```
        int i,mark=0;
        unsigned char first_value,swap_tmp,temp;
//      ShowValueCX();
        temp=ParentCX2[0];
        first_value=ParentCX1[0];
        for(i=0;i<size;i++)
        {
                if(ParentCX1[i]==temp && i != mark)
                {
                        swap_tmp=ParentCX1[i];
                        ParentCX1[i]=ParentCX2[i];
                        ParentCX2[i]=swap_tmp;
                        temp=ParentCX1[i];
                        mark=i;
                        i = -1;
                }
        }
        for(i=0;i<size;i++)
        {
                swap_tmp=ParentCX1[i];
                ParentCX1[i]=ParentCX2[i];
                ParentCX2[i]=swap_tmp;
        }
//      ShowValueCX();fprintf(file,"\n");
        CopyItem(ParentCX1,0,ChromosomeBuffer,first);
        CopyItem(ParentCX2,0,ChromosomeBuffer,second);
}

void CrossOver_CX(float Prob_CX)
{
        int i;
        if(Prob_CX == 0) return;
        CountCX=0;
        AllocRanDomProbCX();
        for(i=0;i<population;i++)
                if(RanDomProbCX[i] <= Prob_CX) CountCX++;
//      fprintf(file,"CountCX = %d\n",CountCX);
        AllocIndexProbCX();
        SelectProbCX(Prob_CX);
//      fprintf(file,"\n");
        if(CountCX%2 == 1) CountCX--;
        for(i=0;i<CountCX;i+=2)
        {
//              fprintf(file,"[%d %d]\n",IndexProbCX[i],IndexProbCX[i+1]);
                CrossCX(IndexProbCX[i],IndexProbCX[i+1]);
        }
//      fprintf(file,"\n");
        UpdateCost();
}

////////////////////////////////////////////////////////////////////////////////
// File Distance.cpp
// Last update 31/05/97
////////////////////////////////////////////////////////////////////////////////
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <malloc.h>
#include <string.h>
#include <time.h>

unsigned char *Chromosome,*ChromosomeBuffer,*ChromosomeTemp,*BestChromosome;       //
Buffer for String
signed int      *MemVij,*MemFij;                // Buffer for store Vij
float           Prob_PMX,Prob_OX,Prob_CX,Prob_M,mutat;
int             Row,Col,size;
float           *Cost,*CostBuffer,*CostTemp,BestCost;
int             population;
int             generation;
int             mark[20];
int             interval,total;
FILE            *file,*summary;

// Functions are not in this file.
void RandomToPlantLayout(void);
float Evaluate(void);
float CostMin(void);
```

```
float CostMax(void);
float CostAvg(void);
float CostSum(void);
double STD(void);
void  CostReproduct(void);
void  CrossOver_PMX(float Prob_PMX);
void  CrossOver_OX(float Prob_OX);
void  CrossOver_CX(float Prob_CX);
void  Mutate(float mutate);
void  Elitist(void);
void CopyItem(unsigned char *Soure,int soure,unsigned char *Destination,int
destination);
// Functions are in this file.
void AllocDij(void);
void AllocVij(void);
void initial(void);
void AllocCostBuffer(void);
void AllocChromosome(void);
void SavePlantLayout(int pop);
void SaveToFile(int Generation);
void ShowChromosome(void);
void SaveBest(void);
void UpdateCost(void);
void SummaryToFile(int gen);
int  NeverMark(int end,int key);
void InitChromosome(void);

void InitChromosome(void)
{
        FILE *initchromosome;
        char buffer[10];
        int  InitValue,Temp,i,j;
        unsigned char *TempChro;
        initchromosome = fopen("init.dat","rt");
        if(initchromosome == NULL)
        {
                printf("Can not find init.dat\n");
                exit(0);
        }
        fgets(buffer,10,initchromosome);
        sscanf(buffer,"%d",&InitValue);
        TempChro = (unsigned char *)malloc(sizeof(char)*size*InitValue);
        printf("Initial Chromosome is\n");
        for(i=0;i<InitValue;i++)
        {
                for(j=0;j<size;j++)
                {
                        fgets(buffer,10,initchromosome);
                        sscanf(buffer,"%d",&Temp);
                        TempChro[j]=(unsigned char)Temp;
                        printf("%d ",TempChro[j]);
                }
                CopyItem(TempChro,0,ChromosomeBuffer,i);
                printf("\n");
        }
        free(TempChro);
        fclose(initchromosome);
}

int  NeverMark(int end,int key)
{
        int i;
        for(i=0;i<end;i++)
                if(mark[i] == key) return 0;
        return 1;
}

void SummaryToFile(int gen)
{
        int i,j,k;
        float min;
        fprintf(summary,"[Generation %d]\n",gen);
        min=CostMax();
        for(i=0;i<total;i++)
        {
                for(k=0;k<population;k++)
                {
```

```
                        if(CostBuffer[k]<=min && NeverMark(i,k))
                        {
                                min=CostBuffer[k];
                                mark[i]=k;
                        }
                }

                fprintf(summary,"[%d]\t",mark[i]);
                for(j=0;j<size;j++)
                        fprintf(summary,"%d ",ChromosomeBuffer[size*mark[i]+j]);
                fprintf(summary," =\t%f\n",CostBuffer[mark[i]]);
                min=CostMax();
        }
        fprintf(summary,"\nThe Best Cost is : %f\t",BestCost);
        fprintf(summary,"[ ");
        for(i=0;i<size;i++)
                fprintf(summary,"%d ",BestChromosome[i]);
        fprintf(summary,"]\n");
        fprintf(summary,"Averang is : %f\n",CostAvg());
        fprintf(summary,"STD     is : %f\n\n",STD());
}

void UpdateCost(void)
{
        int i,j;
        for(i=0;i<population;i++)
        {
                for(j=0;j<size;j++)
                        Chromosome[j]=ChromosomeBuffer[i*size+j];
                CostBuffer[i]=Evaluate();
        }
}

void ShowChromosome(void)
{
        int i;
        fprintf(file,"%f\t",CostMin());
        fprintf(file,"%f\t",CostMax());
        fprintf(file,"%f\t",CostAvg());
        fprintf(file,"%f\t",CostSum());
        fprintf(file,"%f\t",STD());
        fprintf(file,"[ ");
        for(i=0;i<size;i++)
                fprintf(file,"%d ",BestChromosome[i]);
        fprintf(file,"]\n");
}

void AllocDij(void)    // Allocate memory for construct string
{
        Chromosome = (unsigned char *)malloc(size*sizeof(char));
        if(Chromosome == NULL)
        {
                printf("Not enough memmory for \"Chromosome\"\n");
                exit(0);
        }
}

// Allocate memmory for weight table and input weight in term continue
// Input : None
// Output: Enter value in pointer MemVij
void AllocVij(void)
{
        FILE *frequency,*cost;
        char buffer[5];
        int  weight,i;
        frequency=fopen("frequency.dat","rt");
        if(frequency == NULL)
        {
                printf("Can not open file frequency.dat\n");
                exit(0);
        }

        cost=fopen("cost.dat","rt");
        if(cost == NULL)
        {
                printf("Can not open file cost.dat\n");
                exit(0);
```

```
        }
        fgets(buffer,15,frequency);
        sscanf(buffer,"%d",&Row);      // Get ROW
        fgets(buffer,15,frequency);
        sscanf(buffer,"%d",&Col);      // Get COL
        size=Row*Col;
        weight = size*size;                          // Find total of weight
        printf("weight = %d\n",weight);
        MemVij = (int *)malloc(weight*sizeof(weight));// Alloc memmory for input weight
        if(MemVij == NULL)
        {
                printf("Not enough memmory for \"MemVij\"\n");
                exit(0);
        }
        MemFij = (int *)malloc(weight*sizeof(weight));// Alloc memmory for input weight
        if(MemVij == NULL)
        {
                printf("Not enough memmory for \"MemVij\"\n");
                exit(0);
        }
        for(i=0;i<weight;i++)
        {
                fgets(buffer,15,frequency);
                sscanf(buffer,"%d",&MemVij[i]);
        }
        for(i=0;i<weight;i++)
        {
                fgets(buffer,15,cost);
                sscanf(buffer,"%d",&MemFij[i]);
        }
        printf("COST is\n");
        for(i=0;i<weight;i++)
                printf("%d ",MemVij[i]);
        printf("\n");
        printf("Frequency is\n");
        for(i=0;i<weight;i++)
                printf("%d ",MemFij[i]);
        printf("\n");
        fclose(frequency);
        fclose(cost);
}

void AllocCostBuffer(void)
{
        CostBuffer=(float *)malloc(sizeof(float)*population);
        if(CostBuffer == NULL)
        {
                printf("Not enough memmory for \"CostBuffer\"\n");
                exit(0);
        }
        CostTemp=(float *)malloc(sizeof(float)*population);
        if(CostTemp == NULL)
        {
                printf("Not enough memmory for \"CostTemp\"\n");
                exit(0);
        }
}

void AllocChromosomeBuffer(void)
{
        ChromosomeBuffer=(unsigned char *)malloc(size*sizeof(char)*population);
        if(ChromosomeBuffer == NULL)
        {
                printf("Not enough memmory for \"ChromosomeBuffer\"\n");
                exit(0);
        }
        ChromosomeTemp=(unsigned char *)malloc(size*sizeof(char)*population);
        if(ChromosomeTemp == NULL)
        {
                printf("Not enough memmory for \"ChromosomeTemp\"\n");
                exit(0);
        }
        BestChromosome=(unsigned char *)malloc(size*sizeof(char));
        if(ChromosomeTemp == NULL)
        {
                printf("Not enough memmory for \"Best Chromosome\"\n");
                exit(0);
```

```
        }
}

void SavePlantLayout(int pop)
{
        int i;
        for(i=0;i<size;i++)
                ChromosomeBuffer[pop*size+i]=Chromosome[i];
}

void SaveToFile(int Generation)
{
        fclose(file);
        fclose(summary);
}

void initial(void)
{
        int i;
        printf("Enter Generation : ");scanf("%d",&generation);
        printf("Enter Population : ");scanf("%d",&population);
        printf("Enter Probability PMX : ");scanf("%f",&Prob_PMX);
        printf("Enter Probability OX  : ");scanf("%f",&Prob_OX);
        printf("Enter Probability CX  : ");scanf("%f",&Prob_CX);
        printf("Enter Mutate : ");scanf("%f",&mutat);
        printf("How many period for report : ");scanf("%d",&interval);
        printf("How many report chromosome : ");scanf("%d",&total);
        AllocVij();             // Build relationship chart table
        AllocCostBuffer();
        AllocDij();             // Build plant layout
        AllocChromosomeBuffer();
        file=fopen("result.txt","wt");
        if(file == NULL)
        {
                printf("Can not open file \"RESULT.TXT\" \n");
                exit(0);
        }
        summary=fopen("summary.txt","wt");
        if(file == NULL)
        {
                printf("Can not open file \"SUMMARY.TXT\" \n");
                exit(0);
        }
//      srand( (unsigned)time( NULL ) );
        for(i=0;i<population;i++)
        {
                RandomToPlantLayout();
                SavePlantLayout(i);
//              CostBuffer[i]=Evaluate();
        }
        UpdateCost();
}

void main(void)
{
        int i;
        clock_t start, finish;
        double  use;
        char    *version="Program Plant Layout version 1.02 for Distance\nLast update
14/09/97\n";

//      printf("%s",version);  // Show version
        initial();
//      fprintf(file,"%s",version);
        start=clock();
        BestCost=CostMax();
        for(i=0;i<generation;i++)
        {
                SaveBest();
                CostReproduct();
                if(i == 0)
                {
                        InitChromosome();
                        UpdateCost();
                }
                SaveBest();
                CrossOver_PMX(Prob_PMX);
```

```
                SaveBest();
                CrossOver_OX(Prob_OX);
                SaveBest();
                CrossOver_CX(Prob_CX);
                SaveBest();
                Mutate(mutat);
                SaveBest();
                Elitist();
                ShowChromosome();
                printf("Generation %d Completed\n",i);
                if(i%interval == 0) SummaryToFile(i);
        }
        SaveToFile(i);
        finish=clock();
        use=(double)(finish-start)/CLOCKS_PER_SEC;
        printf("Function sucessfully used time = %2.2f sec.\n",use);
        free(BestChromosome);
        free(ChromosomeBuffer);
        free(CostBuffer);
        free(ChromosomeTemp);
        free(CostTemp);
        free(MemVij);
}

////////////////////////////////////////////////////////////////////////////////
// File Elitist.cpp
// Last update 31/05/97
////////////////////////////////////////////////////////////////////////////////
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

float  CostMin(void);
void   CopyItem(unsigned char *Soure,int soure,unsigned char *Destination,int
destination);
float  CostMax(void);
void UpdateCost(void);
void   SaveBest(void);
extern unsigned char *Chromosome,*ChromosomeBuffer,*BestChromosome; // Buffer for
String
extern float   *CostBuffer,BestCost;
extern int     population;
extern int     size;
extern FILE *file;

void SaveBest(void)
{
        int i;
        if(CostMin() < BestCost)
        {
                BestCost=CostMin();
                for(i=0;i<population;i++)
                        if(CostBuffer[i]==BestCost) break;
                CopyItem(ChromosomeBuffer,i,BestChromosome,0);
        }
}

void ReplaceBestChromosome(void)
{
        int i;
        float BadCost;
        BadCost=CostMax();
        for(i=0;i<population;i++)
                if(CostBuffer[i]==BadCost) break;
        CopyItem(BestChromosome,0,ChromosomeBuffer,i);
}

void Elitist(void)
{
        ReplaceBestChromosome();
        UpdateCost();
}

////////////////////////////////////////////////////////////////////////////////
// File Mutation.cpp
// Last update 31/05/97
////////////////////////////////////////////////////////////////////////////////
```

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>

void UpdateCost();
extern int size;
extern int population;
extern unsigned char *ChromosomeBuffer;
extern FILE *file;
void InitialMutate(void);
void RandomMutate(void);
void SelectMutate(void);
void Mutation(int position);
void ShowChromosome(int pop,int siz,int swaper);
void Mutate(float mutat);
float *Gene,mutate;

void InitialMutate(void)
{
        Gene=(float *)malloc(sizeof(float)*size*population);
        if(Gene == NULL)
        {
                printf("Not enough memory for Gene\n");
                exit(0);
        }
}

/*
void RandomChromosome(void)
{
        int Size,Population;

        for(Population=0;Population<population;Population++)
        {
                for(Size=0;Size<size;Size++)
                {
                        do
                        {
                                ChromosomeBuffer[(Population*size)+Size]=rand()%size;
                        } while(Redundant(Population,Size));
                }
        }

        for(Population=0;Population<population;Population++)
        {
                printf("Level %d \t[ ",Population);
                for(Size=0;Size<size;Size++)
                {
                        printf("%d ",ChromosomeBuffer[(Population*size)+Size]);
                }
                printf("]\n");
        }
}
int Redundant(int pop,int siz)
{
        int i;
        for(i=0;i<siz;i++)
        {
                if(ChromosomeBuffer[(pop*size)+siz] == ChromosomeBuffer[(pop*size)+i])
                        return 1;
        }
        return 0;
}
*/

void RandomMutate(void)
{
        int i;
//      srand( (unsigned)time( NULL ) );
        for(i=0;i<population*size;i++)
        {
                Gene[i]=(float)(rand()*rand()%1000000)/(float)1000000;
//              if(i%6 == 5) fprintf(file,"\n");
//              fprintf(file,"%f\t",Gene[i]);
        }
}
```

```
void SelectMutate(void)
{
        int i;

//      fprintf(file,"\nSelect Mutate\n");
        for(i=0;i<population*size;i++)
        {
                if(Gene[i] <= mutate)
                {
                        Mutation(i);
                }
        }
}

void Mutation(int position)
{
        int pop,siz;
        int swaper;
        unsigned char temp;
        pop = position/size;
        siz = position%size;
//      fprintf(file,"[Expected Mutation=%d]\tPercent mutate = %f\n   Chromosome  =
%d\n   Logus = %d\n",position,Gene[position],pop,siz);
        swaper=size;
//      ShowChromosome(pop,siz,swaper);
//      srand( (unsigned)time( NULL ) );
        do
        {
                swaper=rand()%size;
        } while(swaper == siz);
//      ShowChromosome(pop,siz,swaper);
        temp=ChromosomeBuffer[(pop*size)+siz];
        ChromosomeBuffer[(pop*size)+siz] = ChromosomeBuffer[(pop*size)+swaper];
        ChromosomeBuffer[(pop*size)+swaper] = temp;
//      ShowChromosome(pop,siz,swaper);
}

void ShowChromosome(int pop,int siz,int swaper)
{
        int i;
//      fprintf(file,"Chromosome %d \t[",pop);
        for(i=0;i<size;i++)
        {
                if(i == siz || i == swaper) fprintf(file,"|");
                else fprintf(file," ");
                fprintf(file,"%d",ChromosomeBuffer[(pop*size )+i]);
                if(i == siz || i == swaper) fprintf(file,"|");
                else fprintf(file," ");
        }
        fprintf(file,"]\n");
}

void Mutate(float mutat)
{
        mutate=mutat;
        InitialMutate();
        RandomMutate();
        SelectMutate();
        UpdateCost();
        free(Gene);
}

////////////////////////////////////////////////////////////////////////////////
// File Reproduct.cpp
// Last update 31/05/97
////////////////////////////////////////////////////////////////////////////////
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <time.h>

extern unsigned char *ChromosomeBuffer,*ChromosomeTemp;     // Buffer for String
extern int           population,size;
extern float            *CostBuffer,*CostTemp;
extern FILE *file;
```

```
// Function are not in this file
float  CostSum(void);
void   UpdateCost(void);
// Function are in this file
void AllocIndex(void);
void AllocProb(void);
void AllocCumulative(void);
void CostReproduct(void);
void CopyItem(unsigned char *Soure,int soure,unsigned char *Destination,int
destination);
int     *Index;
float   *Prob,*Cumulative;
void AllocIndex(void)
{
        Index=(int *)malloc(population*sizeof(int));
        if(Index == NULL)
        {
                printf("Not enough memmory for Index...\n");
                exit(0);
        }
}

void AllocProb(void)
{
        Prob=(float *)malloc(population*sizeof(float));
        if(Prob == NULL)
        {
                printf("Notenough memmory for Prob...\n");
                exit(0);
        }
}

void AllocCumulative(void)
{
        Cumulative=(float *)malloc(population*sizeof(float));
        if(Cumulative == NULL)
        {
                printf("Not enough memmory for Cumulative...\n");
                exit(0);
        }
}

void CopyItem(unsigned char *Soure,int soure,unsigned char *Destination,int
destination)
{
        int i;
        for(i=0;i<size;i++)
                Destination[destination*size+i]=Soure[soure*size+i];
        if(Soure == ChromosomeTemp)
                CostBuffer[destination]=CostTemp[soure];
        if(Soure == ChromosomeBuffer)
                CostTemp[destination]=CostBuffer[soure];
}

void CostReproduct(void)
{
        float sum,roulette;
        int   i,j;
        AllocIndex();
        AllocProb();
        AllocCumulative();
        sum=CostSum();
        for(i=0;i<population;i++)
                Cumulative[i] = sum-CostBuffer[i];
        sum=(float)0;
        for(i=0;i<population;i++)
                sum+=Cumulative[i];
        for(i=0;i<population;i++)
                Prob[i]=Cumulative[i]/sum;
        Cumulative[0]=Prob[0];
        for(i=1;i<population;i++)
                Cumulative[i]=Prob[i]+Cumulative[i-1];
        for(j=0;j<population;j++)
        {
                roulette=(float)(rand()*rand()%1000000)/(float)1000000;
                for(i=0;i<population;i++)
```

```
            {
                    if(roulette > Cumulative[i] && roulette <= Cumulative[i+1])
                    {
                            Index[j]=i+1;
                            break;
                    }
                    else(roulette <= Cumulative[0]);
                    {
                            Index[j]=0;
                            break;
                    }
            }
    }
    for(i=0;i<population;i++)
            CopyItem(ChromosomeBuffer,Index[i],ChromosomeTemp,i);
    for(i=0;i<population;i++)
            CopyItem(ChromosomeTemp,i,ChromosomeBuffer,i);
    UpdateCost();
    free(Cumulative);
    free(Prob);
    free(Index);
}
```

# ภาคผนวก ฎ
# รายละเอียดไฟล์ตัวหนังสือของโปรแกรม Construct

เนื้อหารายละเอียดในส่วนนี้จะเสนอถึงรายละเอียดไฟล์ตัวหนังสือของโปรแกรม Construct หรือโปรแกรมการแก้ปัญหาการจัดผังโรงงานโดยพิจารณาถึงข้อมูลเชิงปริมาณที่โดยใช้คอนสตรัคชั่นฮิวริสติก ซึ่งประกอบไปด้วยไฟล์จำนวนทั้งสิ้น 2 ไฟล์ ได้แก่

1. Construct.cpp
2. Cost.cpp

ดังมีรายละเอียดดังต่อไปนี้

```
////////////////////////////////////////////////////////////////////////////////
// File Construct.cpp
// Last update 23/10/97
////////////////////////////////////////////////////////////////////////////////
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>

#define Version          "Construct version 1\n"
#define LastUpdate "Last update 23/10/97\n"
#define MaxBuffer      5
#define mark           0xf
#define unmark         0
char *station,*Mark;
int  col,row,size,table;
float  *cost,*frequency;

// Functions are in this file
void PrintVersion(void);
void Process(void);
void Initial(void);
void ShowParameter(void);
void GetSize(void);
void GetFrequency(void);
void GetCost(void);
void PrepareStation(void);
void ShowStation(void);
void ShowMark(void);
// Function are not is this file
float Evaluate(void);
FILE *FreqTable,*CostTable;
char buffer[MaxBuffer];
char  *Buffer;

/*
Function ShowMark
Input : NONE
Output: NONE
Describe : Show Mark bits
*/
void ShowMark(void)
{
        int i;
        printf("Mark = [ ");
        for(i=0;i<size;i++)
        {
                if(Mark[i] == mark) printf("1 ");
                else printf("0 ");
        }
        printf("]\n\n");
}
```

```
/*
Function ShowStation
Input : NONE
Output: NONE
Describe : Show all stations
*/
void ShowStation(void)
{
        int i;
        printf("Station \n[");
        for(i=0;i<size;i++)
                printf("%d ",station[i]);
        printf("\b]\n");
/*
        printf("Frequency\n");
        for(i=0;i<table;i++)
                printf("Frequency [%d] = %2.2f\n",i,frequency[i]);
        getch();
        printf("Cost\n");
        for(i=0;i<table;i++)
                printf("Cost [%d] = %2.2f\n",i,cost[i]);
        getch();
*/
 }

/*
Function ParepareStation
Input : NONE
Output: NONE
Describe : Alloc memory for station table and random value to table
*/
void PrepareStation(void)
{
        int i,j;
        station = (char *)malloc(sizeof(char)*size);
        if(station == NULL)
        {
                printf("Insufficent memory for station table\n");
                exit(0);
        }
        for(i=0;i<size;i++)
        {
                station[i] = rand()%size;
                for(j=0;j<i;j++)
                {
                        if(station[i] == station[j])
                        {
                                station[i] = rand()%size;
                                j = -1; // Start loop agrain
                        }
                }
        }
}

/*
Function GetCost
Input : NONE
Output: NONE
Describe : Open INPUT.DAT file for get Cost table
*/
void GetCost(void)
{
        int i;
        cost = (float *)malloc(sizeof(float)*table);
        if(frequency == NULL)
        {
                printf("Insufficent memory for cost table\n");
                exit(0);
        }
        for(i=0;i<table;i++)
        {
                fgets(buffer,MaxBuffer,CostTable);
                sscanf(buffer,"%f",&cost[i]);
        }
}
```

```
/*
Function GetFrequency
Input : NONE
Output: NONE
Describe : Open INPUT.DAT file for get Frequency table
*/
void GetFrequency(void)
{
        int i;
        frequency = (float *)malloc(sizeof(int)*table);
        if(frequency == NULL)
        {
                printf("Insufficent memory for frequency table\n");
                exit(0);
        }
        for(i=0;i<table;i++)
        {
                fgets(buffer,MaxBuffer,FreqTable);
                sscanf(buffer,"%f",&frequency[i]);
        }
}

/*
Function GetSize
Input : NONE
Output: NONE
Describe : Open INPUT.DAT file for get row,col
                   evaluate size by formula "Plant.size = row*col"
                   and evaluate size of frequency & cost table
                   by formula Table.size = 2*(row*col);
*/
void GetSize(void)
{
        fgets(buffer,MaxBuffer,FreqTable);
        sscanf(buffer,"%d",&row);
        fgets(buffer,MaxBuffer,FreqTable);
        sscanf(buffer,"%d",&col);
        size = row * col;              // Evaluate size of Plant
        table = size * size;   // Evaluate size of Table
}

/*
Function ShowParameter
Input : NONE
Output: NONE
Describe : Show all Parameter structure
*/
void ShowParameter(void)
{
        printf("\n");
        printf("Row    = %d\n",row);
        printf("Columm = %d\n",col);
        printf("Size   = %d\n",size);
        printf("\n");
}

/*
Function Initial
Input : NONE
Output: NONE
Describe : Input value of Parameter structure such as
                        - genertion of chromosome
                        - population of chromosome
                        - mutate value
                        - etc.
                        and open any files for used
*/
void Initial(void)
{
        if((FreqTable = fopen("Freq.dat","rt")) == NULL)
        {
                printf("Can not open file \"Freq.DAT\"\n");
                exit(0);
        }
        if((CostTable = fopen("Cost.dat","rt")) == NULL)
        {
                printf("Can not open file \"Cost.dat\"\n");
```

```
                exit(0);
        }
        GetSize();
        GetFrequency();
        GetCost();
        PrepareStation();
        ShowParameter();
        ShowStation();
        fclose(FreqTable);
        fclose(CostTable);
}

/*
Function Process
Input : NONE
Output: NONE
Describe : This function is created for decrease line of main program
*/
void Process(void)
{
        float BestCost,Cost;
        int   i,j,BestPosition;
        Buffer   = (char *)malloc(sizeof(char)*size);
        for(i=0;i<size;i++)
                Buffer[i]=(char)0xff;
        Buffer[0]=station[0];
        for(j=1;j<size;j++)
        {
                printf("State [%d]\n",j);
                BestCost=(float)3.4E+38;
                for(i=1;i<size;i++)
                {
                        if(Buffer[i] == (char)0xff)
                        {
                                Buffer[i]=station[j];
                                Cost=Evaluate();
                                printf("\tCost[%d] = %f\n",i,Cost);
                                if(Cost < BestCost)
                                {
                                        BestCost=Cost;
                                        BestPcsitiCn=i;
                                }
                                Buffer[i]=(char)0xff;
                        }
                }
                printf("Best position is %d\n\n",BestPosition);getch();
                Buffer[BestPosition]=station[j];
        }
        printf("New station is\n[ ");
        for(i=0;i<size;i++)
        {
                printf("%d ",Buffer[i]);
                station[i]=Buffer[i];
        }
        printf("]\n");
}

/*
Function PrintVersion
Input : NONE
Output: NONE
Describe : Print Version & Last update of program
*/
void PrintVersion(void)
{
        printf(Version);
        printf(LastUpdate);
}

void main(void)
{
        clock_t start, finish;
        double use;

        PrintVersion();
        Initial();
        start=clock();
```

```
        Process();
        finish=clock();
        use=(double)(finish-start)/CLOCKS_PER_SEC;
        printf("Function sucessfully used time = %2.2f sec.\n",use);
}

////////////////////////////////////////////////////////////////////////////////
// File Cost.cpp
// Last update 31/05/97
////////////////////////////////////////////////////////////////////////////////
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define mark            0xf
#define unmark          0
extern char *station,*Buffer;
extern int  col,row,size,table;
extern float *frequency,*cost;
float Evaluate(void);
float Dij(int i,int j);

/*
Function GetFrequency
Input : NONE
Output: NONE
Describe : Open INPUT.DAT file for get Frequency table
*/
float Evaluate(void)
{
        int   i,j,N=0;
        float Cost;

        Cost=(float)0;
        for(i=0;i<size;i++)
        {
                for(j=0;j<size;j++)
                {
                        Cost+=(Dij(i,j)*(float)frequency[N]*(float)cost[N]);
                        N++;
                }
        }
        return Cost;
}

/*
Function GetFrequency
Input : NONE
Output: NONE
Describe : Open INPUT.DAT file for get Frequency table
*/
float Dij(int i,int j)
{
        int x1,x2,y1,y2;
        int N,M;
        for(N=0;N<size;N++)
        {
                if(Buffer[N]==i) break;
        }
        if(N==size) return (float)0;

        for(M=0;M<size;M++)
        {
                if(Buffer[M]==j) break;
        }
        if(M==size) return (float)0;
        x1 = N/col;
        y1 = N%col;
        x2 = M/col;
        y2 = M%col;
        return (float)sqrt(pow((x1-x2),2)+pow((y1-y2),2));
}
```

# ภาคผนวก ฏ
# รายละเอียดไฟล์ตัวหนังสือของโปรแกรม SDPI

เนื้อหารายละเอียดในส่วนนี้จะเสนอถึงรายละเอียดไฟล์ตัวหนังสือของโปรแกรม SDPI หรือโปรแกรมการแก้ปัญหาการจัดผังโรงงานโดยพิจารณาถึงข้อมูลเชิงปริมาณที่โดยใช้ SDPI ฮิวริสติก ซึ่งประกอบไปด้วยไฟล์ Craft.cpp ซึ่งมีรายละเอียดดังต่อไปนี้

```
////////////////////////////////////////////////////////////////////////////////
// File Craft.cpp
// Last update 30/11/97
////////////////////////////////////////////////////////////////////////////////


#include <iostream.h>

#include <time.h>

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fstream.h>
#include <process.h>
#include <math.h>
#include <conio.h>

const int LEN = 100;
int Row,Column;                         //Row and Column of Plant
int StringLength;                       //Row * Coulmn = number of station
int totalarr;
int counter;
float costchange;                       //different between cost
float min;                              //store minimun cost change
int *costflow;                          //cost of material flow
int *freflow;                           //frequency of material flow
char *buffer1;
char *buffer2;

class String
{
private:
        int StringPlant[LEN];           //String plant layout
        float cost;                     //Cost of plant

public:
////////////////////////////////////////////////////////////////////////////////
        String()                        //constructor no argument
        {
                cost = (float)0.0;//initial cost and array of StringPlant to be zero
                for(int a=0;a<StringLength;a++)
                        {
                                StringPlant[a]=0;
                        }

         /* for(int j=0;j<totalarr;j++)       //used for check total cost array
                cout<<costflow[j]<<" ";
            cout<<endl;
            for(int l=0;l<totalarr;l++)       //used for check total fre array
                cout<<freflow[l]<<" ";        */

        }
////////////////////////////////////////////////////////////////////////////////
        ~String()
        //destructor
        {
}
////////////////////////////////////////////////////////////////////////////////
```

```
int RandomNumber(int StringLength)          //Random Funtion
{
        return rand()%StringLength;
}
////////////////////////////////////////////////////////////////////////////////
void initstring()                           //Initial Random String
{
        int temp;
        for(int i=0;i<StringLength;i++)
        {
                if(i==0)
                {
                        temp=RandomNumber(StringLength);
                        StringPlant[0] = temp;
                }
                else
                {
                        temp=RandomNumber(StringLength);
                        for(int j=0;j<i;j++)
                        {
                                if(StringPlant[j] == temp)
                                {
                                        temp=RandomNumber(StringLength);
                                        j=-1;
                                }
                                if(StringPlant[j] != temp && j==i-1)
                                break;
                        }
                        StringPlant[j+1]=temp;
                }
        }
}
////////////////////////////////////////////////////////////////////////////////
void evaluate(void)                 //calculate cost
{
        int   i,j,N=0;
        cost = (float)0.0;
        for(i=0;i<StringLength;i++)
                {
                        for(j=0;j<StringLength;j++)
                        {
                                cost+=(Distij(i,j)*costflow[N]*freflow[N]);
                                N++;
                        }
                }
}
////////////////////////////////////////////////////////////////////////////////
float Distij(int i,int j)
{
        int x1,x2,y1,y2;
        int N,M;
        for(N=0;N<StringLength;N++)
                if(StringPlant[N]==i) break;

        for(M=0;M<StringLength;M++)
                if(StringPlant[M]==j) break;

        x1 = N/Column;
        y1 = N%Column;
        x2 = M/Column;
        y2 = M%Column;
        return (float)sqrt(pow((x1-x2),2)+pow((y1-y2),2));
}
////////////////////////////////////////////////////////////////////////////////
void swap(int x,int y)                              //swap bit position
{
        int temp;
        temp=StringPlant[x];
        StringPlant[x]=StringPlant[y];
        StringPlant[y]=temp;
}
////////////////////////////////////////////////////////////////////////////////
String& copy(const String& st)
{

        for(int i=0;i<StringLength;i++)
        {
```

```
                StringPlant[i]=st.StringPlant[i];
            }
            return *this;
        }
////////////////////////////////////////////////////////////////////////////////
        float difference(String right)
        {
            float diff = cost-right.cost;
            return diff;
        }

////////////////////////////////////////////////////////////////////////////////
        void display()                      //display string
        {
            cout<<"\n";
            for(int j=0;j<StringLength;j++)
            {
                cout<<StringPlant[j]<<" ";
            }
            cout<<"is "<<cost;
        }
};
////////////////////////////////////////////////////////////////////////////////

main()
{
        cout<<"Number of station are not more than 30";
        cout<<"\nNumber of Row:";
            cin >> Row;
        cout<<"\nNumber of Column:";
            cin >> Column;
        StringLength = Row*Column;          //Boundary of StringPlant
        totalarr = StringLength*StringLength;

        costflow = new int[totalarr*sizeof(int)];
        freflow  = new int[totalarr*sizeof(int)];
        buffer1  = new char[totalarr*sizeof(char)];  //pointer to each line(cost)
        buffer2  = new char[totalarr*sizeof(char)];  //pointer to each line(freq)
////////////////////////////////////////////////////////////////////////////////
            ifstream infile1("cflow.dat");      //input data
                if(infile1==NULL)
                {
                    cerr<<"\nCould not open cflow.dat file";exit(1);
                }
                for(int i=0;i<totalarr;i++)
                    {
                        infile1.getline(buffer1,totalarr,'\n');
        //read line of text
                        costflow[i]=atoi(buffer1);
                    }
                ifstream infile2("fflow.dat");      //input data
                if(infile2==NULL)
                {
                    cerr<<"\nCould not open freflow.dat file";exit(1);
                }
                for(int k=0;k<totalarr;k++)
                    {
                        infile2.getline(buffer2,totalarr,'\n');
        //read line of text
                        freflow[k]=atoi(buffer2);
                    }

        /*      for(int y=0;y<totalarr;y++)   //used for check input file data
                {
                    cout<<costflow[y]<<" ";
                }
                cout<<"\n";
                for(int h=0;h<totalarr;h++)
                {
                    cout<<freflow[h]<<" ";
                }       */
////////////////////////////////////////////////////////////////////////////////
        clock_t start, finish;              //calculate time in process
        double  duration;
        start = clock();
////////////////////////////////////////////////////////////////////////////////
        String st1,st2,st3;
```

```
        st1.initstring();
        st1.evaluate();
//      st1.display();
        counter = 0;
                do
                {
                min = (float)0.0;
                cout<<"\nNumber of iteration"<<counter;
                for(int t=0;t<StringLength-1;t++)
                        for(int l=t+1;l<StringLength;l++)
                {
                        st1.evaluate();
                        st2.copy(st1);                          //st2 copy from st1
                        st2.evaluate();
//                      st2.display();                          //used for check up string
                        st2.swap(t,l);
                        cout<<"\n"<<"swap"<<t<<" "<<l;
                        st2.evaluate();
                        st2.display();
                                costchange=st2.difference(st1);       //cal st2 - st1
                                cout<<" diff is "<<costchange;
                                if(costchange<0)
                                {
                                        if(costchange<min)
                                        {
                                                min=costchange;
                                                st3.copy(st2);        //st3 copy from st2
                                        }
                                }
                }
                st3.evaluate();
        //      st3.display();
                st1.copy(st3);
                counter++;
                }while(min<0);
////////////////////////////////////////////////////////////////////////////////
cout<<"\nSummary data";                        //display summary
cout<<"\nNumber of total iterations is "<<counter;
cout<<"\nresult is ";
st3.display();
////////////////////////////////////////////////////////////////////////////////
        finish = clock();                       //calculate time in process
        duration = (double)(finish - start) / CLOCKS_PER_SEC;
        cout<<"\ntotal used time cpp"<<duration<<" seconds";
////////////////////////////////////////////////////////////////////
        delete []costflow;
        delete []freflow;
        delete []buffer1;                       //delete buffer for cost
        delete []buffer2;                       //delete buffer for frequency
        return 0;
}
////////////////////////////////////////////////////////////////////////////////
```

# ประวัติผู้เขียน

นายชนะ เยี่ยงกมลสิงห์ เกิดวันที่ 29 ธันวาคม 2513 จังหวัด กรุงเทพ ฯ สำเร็จการศึกษาปริญญาตรีวิศวกรรมศาสตรบัณฑิต สาขาวิศวกรรมการวัดคุมทางอุตสาหกรรม ภาควิชาวิศวกรรมการวัดคุมทางอุตสาหกรรม คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ในปีการศึกษา 2537 หลังจากนั้นได้เข้าทำงานที่บริษัทสยามนิสสันออโตโมบิล จำกัด แล้วเข้าศึกษาต่อในหลักสูตรวิศวกรรมศาสตรมหาบัณฑิต สาขาวิศวกรรมอุตสาหการ จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2538