DAFTAR PUSTAKA

Aziz, M. H., Qamar, S., Khasawneh, M. T., & Saha, C. (2020). Cloud manufacturing: A myth or future of global manufacturing? Journal of Manufacturing Technology Management, 31, 1325–1350.

Cao, Y., Wang, S., Kang, L., & Gao, Y. (2015). A TQCS-based service selection and scheduling strategy in Cloud Manufacturing. The International Journal of Advanced Manufacturing Technology, 82(1-4), 235–251. https://doi.org/10.1007/s00170-015-7350-5

Chaouch, I., Driss, O.B., & Ghedira, K. (2017). Elitist Ant System for the Distributed Job Shop Scheduling Problem. In: Benferhat, S., Tabia, K., Ali, M. (eds) Advances in Artificial Intelligence: From Theory to Practice. IEA/AIE 2017. Lecture Notes in Computer Science(), vol 10350. Springer, Cham. https://doi.org/10.1007/978-3-319-60042-0_12

Ghomi, E. J., Rahmani, A. M., & Qader, N. N. (2019). Cloud manufacturing: challenges, recent advances, open research issues, and future trends. The International Journal of Advanced Manufacturing Technology, 102(9–12), 3613–3639.

Helo, P., Hao, Y., Toshev, R., & Boldosova, V. (2021). Cloud manufacturing ecosystem analysis and design. Robotics and Computer-Integrated Manufacturing, 67, Article 102050.

Lee, Y. T., Kumaraguru, S., Jain, S., Robinson, S., Helu, M., Hatim, Q. Y., Rachuri, S., Dornfeld, D., Saldana, C. J., & Kumara, S. (2017). A Classification Scheme for Smart Manufacturing Systems' Performance Metrics. In Smart and Sustainable Manufacturing Systems (Vol. 1, Issue 1, p. 20160012). ASTM International. https://doi.org/10.1520/ssms20160012

Liu, Y., Wang, L., Wang, X. V., Xu, X., & Jiang, P. (2019). Cloud manufacturing: Key issues and future perspectives. International Journal of Computer Integrated Manufacturing, 32, 858–874.

Liu, Y., Wang, L., Wang, X. V., Xu, X., & Zhang, L. (2018). Scheduling in cloud manufacturing: State-of-the-art and research challenges. International Journal of Production Research, 57(15-16), 4854–4879. https://doi.org/10.1080/00207543.2018.1449978

Liu, Y., Xu, X., Zhang, L., Wang, L., & Zhong, R. (2017). Workload-based multi-task scheduling in Cloud Manufacturing. Robotics and Computer-Integrated Manufacturing, 45, 3–20. https://doi.org/10.1016/j.rcim.2016.09.008

Tao, F., L. Zhang, Y. Liu, Y. Cheng, L. Wang, & X. Xu. (2015). Manufacturing Service Management in Cloud Manufacturing: Overview and Future Research Directions. Journal of Manufacturing Science and Engineering137 (4): 040912.

Wu, B., Cheng, J., & Dong, M. (2020). Hybrid fruit fly optimisation algorithm for Field Service Scheduling Problem. International Journal of Automation and Control, 14(5/6), 554. https://doi.org/10.1504/ijaac.2020.110072

Wu, D., Greer, M. J., Rosen, D. W., & Schaefer, D. (2013). Cloud manufacturing: Strategic vision and state-of-the-art. *Journal of Manufacturing Systems*, *32*(4), 564–579. https://doi.org/10.1016/j.jmsy.2013.04.008

Wu, D., Rosen, D. W., Wang, L., & Schaefer, D. (2015). Cloud-based design and manufacturing: A new paradigm in digital manufacturing and Design Innovation. *Computer-Aided Design*, *59*, 1–14. https://doi.org/10.1016/j.cad.2014.07.006

Yang, B., Wang, S., Cheng, Q., & Jin, T. (2021). Scheduling of field service resources in cloud manufacturing based on multi-population competitive-

cooperative GWO - ScienceDirect. Computers & Industrial Engineering, 154, 107104. https://doi.org/10.1016/j.cie.2021.107104

Yumbe, Y., Komoda, N., & Fujiwara, T. (2019). Workforce scheduling system to manage static optimization and dynamic re-optimization for Field Service. 2019 IEEE International Conference on Systems, Man and Cybernetics (SMC). https://doi.org/10.1109/smc.2019.8914507

Yusoff, M., & Othman, A. A. (2018). Genetic Algorithm with Elitist-Tournament for Clashes-Free Slots of Lecturer Timetabling Problem. In Indonesian Journal of Electrical Engineering and Computer Science (Vol. 12, Issue 1, p. 303). Institute of Advanced Engineering and Science. https://doi.org/10.11591/ijeecs.v12.i1.pp303-309

Zhou, L., Zhang, L., Zhao, C., Laili, Y., & Xu, L. (2017). Diverse task scheduling for individualized requirements in Cloud Manufacturing. Enterprise Information Systems, 12(3), 300–318. https://doi.org/10.1080/17517575.2017.1364428

LAMPIRAN

Lampiran 1. Tampilan *console* hasil eksekusi

TAMPILAN CONSOLE

```
List Client:

Client [clientId=0, clientName=Client 1, clientLocationId=0]

Client [clientId=1, clientName=Client 2, clientLocationId=1]

Client [clientId=2, clientName=Client 3, clientLocationId=2]

List Task:

Task [taskId=0, taskName=Task 1,1, taskLocationId=0]

Task [taskId=1, taskName=Task 1,2, taskLocationId=0]

Task [taskId=2, taskName=Task 2,1, taskLocationId=1]

Task [taskId=3, taskName=Task 2,2, taskLocationId=1]

Task [taskId=4, taskName=Task 3,1, taskLocationId=2]

Task [taskId=5, taskName=Task 3,2, taskLocationId=2]

List subTask:

SubTask [subTaskId=0, subTaskName=SubTask 1,1,1, subTaskManufacturingType=4,
subTaskLocationId=0, subTaskExecutionTime=2,49]

SubTask [subTaskId=1, subTaskName=SubTask 1,1,2, subTaskManufacturingType=2,
subTaskLocationId=0, subTaskExecutionTime=2,28]

SubTask [subTaskId=2, subTaskName=SubTask 1,2,1, subTaskManufacturingType=1,
subTaskLocationId=0, subTaskExecutionTime=2,18]

SubTask [subTaskId=3, subTaskName=SubTask 1,2,2, subTaskManufacturingType=1,
subTaskLocationId=0, subTaskExecutionTime=1,70]

SubTask [subTaskId=4, subTaskName=SubTask 2,1,1, subTaskManufacturingType=0,
subTaskLocationId=1, subTaskExecutionTime=1,40]

SubTask [subTaskId=5, subTaskName=SubTask 2,1,2, subTaskManufacturingType=0,
subTaskLocationId=1, subTaskExecutionTime=1,28]

SubTask [subTaskId=6, subTaskName=SubTask 2,2,1, subTaskManufacturingType=4,
subTaskLocationId=1, subTaskExecutionTime=2,10]

SubTask [subTaskId=7, subTaskName=SubTask 2,2,2, subTaskManufacturingType=4,
subTaskLocationId=1, subTaskExecutionTime=1,01]

SubTask [subTaskId=8, subTaskName=SubTask 3,1,1, subTaskManufacturingType=4,
subTaskLocationId=2, subTaskExecutionTime=1,49]

SubTask [subTaskId=9, subTaskName=SubTask 3,1,2, subTaskManufacturingType=3,
subTaskLocationId=2, subTaskExecutionTime=1,96]

SubTask [subTaskId=10, subTaskName=SubTask 3,2,1, subTaskManufacturingType=4,
subTaskLocationId=2, subTaskExecutionTime=1,07]

SubTask [subTaskId=11, subTaskName=SubTask 3,2,2, subTaskManufacturingType=0,
subTaskLocationId=2, subTaskExecutionTime=2,04]

List MnfcServiceProvider:
```

MnfcServiceProvider [mnfcServiceProviderId=0, mnfcServiceProviderName=Mnfc Service Provider 1, mnfcServiceProviderLocId=3]

MnfcServiceProvider [mnfcServiceProviderId=1, mnfcServiceProviderName=Mnfc Service Provider 2, mnfcServiceProviderLocId=4]

MnfcServiceProvider [mnfcServiceProviderId=2, mnfcServiceProviderName=Mnfc Service Provider 3, mnfcServiceProviderLocId=5]

MnfcServiceProvider [mnfcServiceProviderId=3, mnfcServiceProviderName=Mnfc Service Provider 4, mnfcServiceProviderLocId=6]

List Service:

Service [serviceId=0, serviceName=Service 1,1, serviceManufacturingType=3, serviceLocationIndex=3, serviceCostPerTime=62,96, serviceQuality=87,69, serviceReliability=95,02, serviceTravelVelocity=55,40, serviceTravelCostPerDistance=0,23]

Service [serviceId=1, serviceName=Service 1,2, serviceManufacturingType=1, serviceLocationIndex=3, serviceCostPerTime=84,90, serviceQuality=77,39, serviceReliability=92,90, serviceTravelVelocity=72,28, serviceTravelCostPerDistance=0,19]

Service [serviceId=2, serviceName=Service 1,3, serviceManufacturingType=1, serviceLocationIndex=3, serviceCostPerTime=80,07, serviceQuality=91,34, serviceReliability=92,56, serviceTravelVelocity=77,51, serviceTravelCostPerDistance=0,13]

Service [serviceId=3, serviceName=Service 1,4, serviceManufacturingType=3, serviceLocationIndex=3, serviceCostPerTime=93,47, serviceQuality=91,02, serviceReliability=89,01, serviceTravelVelocity=71,24, serviceTravelCostPerDistance=0,28]

Service [serviceId=4, serviceName=Service 2,1, serviceManufacturingType=3, serviceLocationIndex=4, serviceCostPerTime=95,24, serviceQuality=84,41, serviceReliability=98,97, serviceTravelVelocity=69,84, serviceTravelCostPerDistance=0,22]

Service [serviceId=5, serviceName=Service 2,2, serviceManufacturingType=4, serviceLocationIndex=4, serviceCostPerTime=70,31, serviceQuality=94,33, serviceReliability=98,19, serviceTravelVelocity=61,24, serviceTravelCostPerDistance=0,16]

Service [serviceId=6, serviceName=Service 2,3, serviceManufacturingType=2, serviceLocationIndex=4, serviceCostPerTime=59,03, serviceQuality=88,35, serviceReliability=99,57, serviceTravelVelocity=65,15, serviceTravelCostPerDistance=0,20]

Service [serviceId=7, serviceName=Service 2,4, serviceManufacturingType=4, serviceLocationIndex=4, serviceCostPerTime=71,62, serviceQuality=82,74, serviceReliability=95,35, serviceTravelVelocity=44,07, serviceTravelCostPerDistance=0,16]

Service [serviceId=8, serviceName=Service 3,1, serviceManufacturingType=2, serviceLocationIndex=5, serviceCostPerTime=48,97, serviceQuality=98,61, serviceReliability=97,55, serviceTravelVelocity=66,12, serviceTravelCostPerDistance=0,20]

Service [serviceId=9, serviceName=Service 3,2, serviceManufacturingType=0, serviceLocationIndex=5, serviceCostPerTime=71,89, serviceQuality=76,98, serviceReliability=89,08, serviceTravelVelocity=44,30, serviceTravelCostPerDistance=0,25]

Service [serviceId=10, serviceName=Service 3,3, serviceManufacturingType=0, serviceLocationIndex=5, serviceCostPerTime=58,23, serviceQuality=86,26, serviceReliability=85,02, serviceTravelVelocity=32,34, serviceTravelCostPerDistance=0,22]

Service [serviceId=11, serviceName=Service 3,4, serviceManufacturingType=4, serviceLocationIndex=5, serviceCostPerTime=77,28, serviceQuality=81,13, serviceReliability=98,21, serviceTravelVelocity=60,09, serviceTravelCostPerDistance=0,12]

Service [serviceId=12, serviceName=Service 4,1, serviceManufacturingType=3, serviceLocationIndex=6, serviceCostPerTime=82,95, serviceQuality=92,17, serviceReliability=91,57, serviceTravelVelocity=57,77, serviceTravelCostPerDistance=0,16]

Service [serviceId=13, serviceName=Service 4,2, serviceManufacturingType=3, serviceLocationIndex=6, serviceCostPerTime=50,68, serviceQuality=76,59, serviceReliability=93,30, serviceTravelVelocity=62,98, serviceTravelCostPerDistance=0,17]

Service [serviceId=14, serviceName=Service 4,3, serviceManufacturingType=1, serviceLocationIndex=6, serviceCostPerTime=59,75, serviceQuality=91,98, serviceReliability=89,76, serviceTravelVelocity=65,55, serviceTravelCostPerDistance=0,17]

Service [serviceId=15, serviceName=Service 4,4, serviceManufacturingType=3, serviceLocationIndex=6, serviceCostPerTime=36,22, serviceQuality=81,42, serviceReliability=93,93, serviceTravelVelocity=43,69, serviceTravelCostPerDistance=0,29]

List Location:

Location [locationId=0, locaitonName=Location Client 1, x=83, y=15]

Location [locationId=1, locaitonName=Location Client 2, x=83, y=35]

Location [locationId=2, locaitonName=Location Client 3, x=60, y=65]

Location [locationId=3, locaitonName=Location Mnfc Service Provider 1, x=80, y=70]

Location [locationId=4, locaitonName=Location Mnfc Service Provider 2, x=15, y=92]

Location [locationId=5, locaitonName=Location Mnfc Service Provider 3, x=73, y=21]

Location [locationId=6, locaitonName=Location Mnfc Service Provider 4, x=53, y=50]

Logistical weight:

Logistical weight [Location Client 1, Location Client 1] : 0,00

Logistical weight [Location Client 1, Location Client 2] : 20,00

Logistical weight [Location Client 1, Location Client 3] : 55,04

Logistical weight [Location Client 1, Location Mnfc Service Provider 1] : 55,08

Logistical weight [Location Client 1, Location Mnfc Service Provider 2] : 102,73

Logistical weight [Location Client 1, Location Mnfc Service Provider 3] : 11,66

Logistical weight [Location Client 1, Location Mnfc Service Provider 4] : 46,10

Logistical weight [Location Client 2, Location Client 1] : 20,00

Logistical weight [Location Client 2, Location Client 2] : 0,00

Logistical weight [Location Client 2, Location Client 3] : 37,80

Logistical weight [Location Client 2, Location Mnfc Service Provider 1] : 35,13

Logistical weight [Location Client 2, Location Mnfc Service Provider 2] : 88,73

Logistical weight [Location Client 2, Location Mnfc Service Provider 3] : 17,20

Logistical weight [Location Client 2, Location Mnfc Service Provider 4] : 33,54

Logistical weight [Location Client 3, Location Client 1] : 55,04

Logistical weight [Location Client 3, Location Client 2] : 37,80

Logistical weight [Location Client 3, Location Client 3] : 0,00

Logistical weight [Location Client 3, Location Mnfc Service Provider 1] : 20,62

Logistical weight [Location Client 3, Location Mnfc Service Provider 2] : 52,48

Logistical weight [Location Client 3, Location Mnfc Service Provider 3] : 45,88

Logistical weight [Location Client 3, Location Mnfc Service Provider 4] : 16,55

Logistical weight [Location Mnfc Service Provider 1, Location Client 1] : 55,08

Logistical weight [Location Mnfc Service Provider 1, Location Client 2] : 35,13

Logistical weight [Location Mnfc Service Provider 1, Location Client 3] : 20,62

Logistical weight [Location Mnfc Service Provider 1, Location Mnfc Service Provider 1] : 0,00

Logistical weight [Location Mnfc Service Provider 1, Location Mnfc Service Provider 2] : 68,62

Logistical weight [Location Mnfc Service Provider 1, Location Mnfc Service Provider 3] : 49,50

Logistical weight [Location Mnfc Service Provider 1, Location Mnfc Service Provider 4] : 33,60

Logistical weight [Location Mnfc Service Provider 2, Location Client 1] : 102,73

Logistical weight [Location Mnfc Service Provider 2, Location Client 2] : 88,73

Logistical weight [Location Mnfc Service Provider 2, Location Client 3] : 52,48

Logistical weight [Location Mnfc Service Provider 2, Location Mnfc Service Provider 1] : 68,62

Logistical weight [Location Mnfc Service Provider 2, Location Mnfc Service Provider 2] : 0,00

Logistical weight [Location Mnfc Service Provider 2, Location Mnfc Service Provider 3] : 91,68

Logistical weight [Location Mnfc Service Provider 2, Location Mnfc Service Provider 4] : 56,64

Logistical weight [Location Mnfc Service Provider 3, Location Client 1] : 11,66

Logistical weight [Location Mnfc Service Provider 3, Location Client 2] : 17,20

Logistical weight [Location Mnfc Service Provider 3, Location Client 3] : 45,88

Logistical weight [Location Mnfc Service Provider 3, Location Mnfc Service Provider 1] : 49,50

Logistical weight [Location Mnfc Service Provider 3, Location Mnfc Service Provider 2] : 91,68

Logistical weight [Location Mnfc Service Provider 3, Location Mnfc Service Provider 3] : 0,00

Logistical weight [Location Mnfc Service Provider 3, Location Mnfc Service Provider 4] : 35,23

Logistical weight [Location Mnfc Service Provider 4, Location Client 1] : 46,10

Logistical weight [Location Mnfc Service Provider 4, Location Client 2] : 33,54

Logistical weight [Location Mnfc Service Provider 4, Location Client 3] : 16,55

Logistical weight [Location Mnfc Service Provider 4, Location Mnfc Service Provider 1] : 33,60

Logistical weight [Location Mnfc Service Provider 4, Location Mnfc Service Provider 2] : 56,64

Logistical weight [Location Mnfc Service Provider 4, Location Mnfc Service Provider 3] : 35,23

Logistical weight [Location Mnfc Service Provider 4, Location Mnfc Service Provider 4] : 0,00

---------- Normal Genetic Algorithm (GA) ----------

```
 populatioNormalGA : Population [schedules=[Schedule
[mnfcTypeArray=[5, 6, 1, 1, 10, 10, 7, 11, 5, 0, 11, 10], sequenceSchedule=[5, 0, 0, 1, 1,
3, 2, 5, 3, 4, 4, 2], fitness=0.1823674835886451]
, Schedule
[mnfcTypeArray=[5, 8, 2, 1, 10, 10, 5, 11, 11, 12, 7, 9], sequenceSchedule=[3, 5, 4, 1, 0,
2, 2, 5, 0, 1, 4, 3], fitness=0.1891438082540812]
, Schedule
[mnfcTypeArray=[5, 6, 14, 1, 10, 9, 11, 7, 7, 13, 5, 9], sequenceSchedule=[4, 4, 0, 2, 1,
5, 3, 3, 1, 2, 0, 5], fitness=0.19769170502879208]
, Schedule
[mnfcTypeArray=[7, 6, 14, 14, 10, 9, 5, 11, 5, 0, 11, 9], sequenceSchedule=[2, 5, 1, 5, 3,
3, 0, 0, 4, 2, 4, 1], fitness=0.20692924666264012]
, Schedule
[mnfcTypeArray=[7, 8, 1, 1, 9, 9, 5, 7, 7, 0, 11, 10], sequenceSchedule=[2, 0, 5, 1, 2, 1,
0, 4, 5, 3, 3, 4], fitness=0.16482462354265612]
, Schedule
[mnfcTypeArray=[5, 6, 14, 14, 9, 10, 11, 5, 11, 3, 7, 10], sequenceSchedule=[2, 5, 0, 1,
4, 5, 4, 3, 3, 1, 2, 0], fitness=0.1980931578151253]
, Schedule
[mnfcTypeArray=[7, 6, 2, 2, 10, 10, 7, 11, 11, 13, 5, 9], sequenceSchedule=[4, 0, 3, 1, 5,
2, 1, 2, 3, 4, 0, 5], fitness=0.16591653097657477]
, Schedule
[mnfcTypeArray=[11, 8, 2, 2, 9, 10, 11, 7, 11, 3, 5, 9], sequenceSchedule=[3, 0, 4, 5, 1,
3, 2, 1, 5, 0, 4, 2], fitness=0.20882182629019713]
, Schedule
[mnfcTypeArray=[5, 6, 2, 2, 10, 9, 11, 5, 7, 13, 5, 10], sequenceSchedule=[4, 0, 5, 4, 3,
0, 2, 1, 3, 2, 5, 1], fitness=0.22502520268966394]
, Schedule
[mnfcTypeArray=[11, 6, 2, 14, 9, 9, 7, 11, 7, 15, 5, 9], sequenceSchedule=[2, 2, 0, 4, 5,
1, 3, 0, 1, 4, 3, 5], fitness=0.19841300013801495]
]]


---------- Elistist Genetic Algorithm (EGA) ------------


 Population : Population [schedules=[Schedule
[mnfcTypeArray=[5, 6, 1, 1, 10, 10, 7, 11, 5, 0, 11, 10], sequenceSchedule=[5, 0, 0, 1, 1,
3, 2, 5, 3, 4, 4, 2], fitness=0.1823674835886451]
, Schedule
[mnfcTypeArray=[5, 8, 2, 1, 10, 10, 5, 11, 11, 12, 7, 9], sequenceSchedule=[3, 5, 4, 1, 0,
2, 2, 5, 0, 1, 4, 3], fitness=0.1891438082540812]
, Schedule
[mnfcTypeArray=[5, 6, 14, 1, 10, 9, 11, 7, 7, 13, 5, 9], sequenceSchedule=[4, 4, 0, 2, 1,
5, 3, 3, 1, 2, 0, 5], fitness=0.19769170502879208]
, Schedule
[mnfcTypeArray=[7, 6, 14, 14, 10, 9, 5, 11, 5, 0, 11, 9], sequenceSchedule=[2, 5, 1, 5, 3,
3, 0, 0, 4, 2, 4, 1], fitness=0.20692924666264012]
, Schedule
[mnfcTypeArray=[7, 8, 1, 1, 9, 9, 5, 7, 7, 0, 11, 10], sequenceSchedule=[2, 0, 5, 1, 2, 1,
0, 4, 5, 3, 3, 4], fitness=0.16482462354265612]
, Schedule
[mnfcTypeArray=[5, 6, 14, 14, 9, 10, 11, 5, 11, 3, 7, 10], sequenceSchedule=[2, 5, 0, 1,
4, 5, 4, 3, 3, 1, 2, 0], fitness=0.1980931578151253]
, Schedule
[mnfcTypeArray=[7, 6, 2, 2, 10, 10, 7, 11, 11, 13, 5, 9], sequenceSchedule=[4, 0, 3, 1, 5,
2, 1, 2, 3, 4, 0, 5], fitness=0.16591653097657477]
, Schedule
[mnfcTypeArray=[11, 8, 2, 2, 9, 10, 11, 7, 11, 3, 5, 9], sequenceSchedule=[3, 0, 4, 5, 1,
3, 2, 1, 5, 0, 4, 2], fitness=0.20882182629019713]
, Schedule
[mnfcTypeArray=[5, 6, 2, 2, 10, 9, 11, 5, 7, 13, 5, 10], sequenceSchedule=[4, 0, 5, 4, 3,
0, 2, 1, 3, 2, 5, 1], fitness=0.22502520268966394]
, Schedule
[mnfcTypeArray=[11, 6, 2, 14, 9, 9, 7, 11, 7, 15, 5, 9], sequenceSchedule=[2, 2, 0, 4, 5,
1, 3, 0, 1, 4, 3, 5], fitness=0.19841300013801495]
]]
```

EGA Initial fittest population EGA: Schedule
[mnfcTypeArray=[5, 6, 2, 2, 10, 9, 11, 5, 7, 13, 5, 10], sequenceSchedule=[4, 0, 5, 4, 3, 0, 2, 1, 3, 2, 5, 1], fitness=0.22502520268966394]

Parameter: [Total Process: 12, Total Scheduling Time: 8.13802277515496, Total Execution Time for all process: 26.977654695564976, Total Execution Cost for all process: 1481.8839307929745, Average Processing Time: 2.2481378912970813, Average Processing Cost: 123.4903275660812, Average Quality: 86.99773914964811, Average Reliability: 93.77026042098277, Fitness: 0.22502520268966394]

---------- START SIMULATING ----------

EGA Fittest Schedule 100 : Schedule
[mnfcTypeArray=[11, 8, 14, 14, 10, 10, 11, 5, 5, 15, 5, 10], sequenceSchedule=[2, 4, 2, 0, 3, 5, 3, 1, 4, 0, 5, 1], fitness=0.30724294073992076]

Parameter: [Total Process: 12, Total Scheduling Time: 6.31713770387964, Total Execution Time for all process: 25.145510594456695, Total Execution Cost for all process: 1335.2584317001715, Average Processing Time: 2.0954592162047248, Average Processing Cost: 111.27153597501429, Average Quality: 89.00214783260218, Average Reliability: 93.0888417444905, Fitness: 0.30724294073992076]

EGA Fittest Schedule 200 : Schedule
[mnfcTypeArray=[11, 8, 14, 14, 10, 10, 11, 5, 5, 15, 5, 10], sequenceSchedule=[0, 1, 2, 3, 3, 1, 5, 5, 4, 0, 4, 2], fitness=0.3107739116830898]

Parameter: [Total Process: 12, Total Scheduling Time: 12.324541359199921, Total Execution Time for all process: 24.974236925278618, Total Execution Cost for all process: 1331.3662914805311, Average Processing Time: 2.081186410439885, Average Processing Cost: 110.94719095671093, Average Quality: 89.00214783260219, Average Reliability: 93.0888417444905, Fitness: 0.3107739116830898]

EGA Fittest Schedule 500 : Schedule
[mnfcTypeArray=[11, 8, 14, 14, 10, 10, 11, 5, 5, 15, 5, 10], sequenceSchedule=[1, 5, 0, 4, 4, 3, 2, 3, 2, 1, 5, 0], fitness=0.31572788570513766]

Parameter: [Total Process: 12, Total Scheduling Time: 6.236359784051436, Total Execution Time for all process: 24.61866759896738, Total Execution Cost for all process: 1329.9400906396895, Average Processing Time: 2.0515556332472817, Average Processing Cost: 110.82834088664079, Average Quality: 89.00214783260218, Average Reliability: 93.08884174449048, Fitness: 0.31572788570513766]

EGA Fittest Schedule 1000 : Schedule
[mnfcTypeArray=[11, 8, 14, 14, 10, 10, 11, 5, 5, 15, 5, 10], sequenceSchedule=[5, 2, 4, 0, 3, 4, 2, 1, 5, 0, 3, 1], fitness=0.30724294073992087]

Parameter: [Total Process: 12, Total Scheduling Time: 6.236359784051436, Total Execution Time for all process: 25.14551059445669, Total Execution Cost for all process: 1335.2584317001713, Average Processing Time: 2.0954592162047243, Average Processing Cost: 111.27153597501427, Average Quality: 89.00214783260218, Average Reliability: 93.0888417444905, Fitness: 0.30724294073992087]

EGA Fittest Schedule 2000 : Schedule
[mnfcTypeArray=[11, 8, 14, 14, 10, 10, 11, 5, 5, 15, 5, 10], sequenceSchedule=[1, 5, 5, 4, 3, 1, 2, 2, 0, 3, 0, 4], fitness=0.3123019025394146]

Parameter: [Total Process: 12, Total Scheduling Time: 7.358661861994611, Total Execution Time for all process: 24.821164780873616, Total Execution Cost for all process: 1332.4454421369967, Average Processing Time: 2.0684303984061345, Average Processing Cost: 111.03712017808306, Average Quality: 89.00214783260218, Average Reliability: 93.0888417444905, Fitness: 0.3123019025394146]

EGA Fittest Schedule 5000 : Schedule

[mnfcTypeArray=[11, 8, 14, 14, 10, 10, 11, 5, 5, 15, 5, 10], sequenceSchedule=[3, 1, 2, 4, 3, 1, 5, 5, 0, 0, 2, 4], fitness=0.3123019025394146]

Parameter: [Total Process: 12, Total Scheduling Time: 8.656246859011077, Total Execution Time for all process: 24.821164780873616, Total Execution Cost for all process: 1332.4454421369967, Average Processing Time: 2.0684303984061345, Average Processing Cost: 111.03712017808306, Average Quality: 89.00214783260218, Average Reliability: 93.0888417444905, Fitness: 0.3123019025394146]

EGA Fittest Schedule 10000 : Schedule
[mnfcTypeArray=[11, 8, 14, 14, 10, 10, 11, 5, 5, 15, 5, 10], sequenceSchedule=[5, 2, 3, 5, 0, 1, 3, 4, 4, 2, 1, 0], fitness=0.3157278857051379]

Parameter: [Total Process: 12, Total Scheduling Time: 7.708916497396623, Total Execution Time for all process: 24.618667598967377, Total Execution Cost for all process: 1329.9400906396895, Average Processing Time: 2.0515556332472813, Average Processing Cost: 110.82834088664079, Average Quality: 89.00214783260219, Average Reliability: 93.08884174449048, Fitness: 0.3157278857051379]

EGA Fittest Schedule 20000 : Schedule
[mnfcTypeArray=[11, 8, 14, 14, 10, 10, 11, 5, 5, 15, 5, 10], sequenceSchedule=[1, 2, 1, 0, 3, 5, 2, 5, 4, 3, 4, 0], fitness=0.31572788570513816]

Parameter: [Total Process: 12, Total Scheduling Time: 6.236359784051436, Total Execution Time for all process: 24.618667598967377, Total Execution Cost for all process: 1329.9400906396893, Average Processing Time: 2.0515556332472813, Average Processing Cost: 110.82834088664077, Average Quality: 89.00214783260219, Average Reliability: 93.0888417444905, Fitness: 0.31572788570513816]

EGA Fittest Schedule 50000 : Schedule
[mnfcTypeArray=[11, 8, 14, 14, 10, 10, 11, 5, 5, 15, 5, 10], sequenceSchedule=[1, 2, 1, 0, 3, 5, 2, 5, 4, 3, 4, 0], fitness=0.31572788570513816]

Parameter: [Total Process: 12, Total Scheduling Time: 6.236359784051436, Total Execution Time for all process: 24.618667598967377, Total Execution Cost for all process: 1329.9400906396893, Average Processing Time: 2.0515556332472813, Average Processing Cost: 110.82834088664077, Average Quality: 89.00214783260219, Average Reliability: 93.0888417444905, Fitness: 0.31572788570513816]

EGA Fittest Schedule 100000 : Schedule
[mnfcTypeArray=[11, 8, 14, 14, 10, 10, 11, 5, 5, 15, 5, 10], sequenceSchedule=[1, 2, 1, 0, 3, 5, 2, 5, 4, 3, 4, 0], fitness=0.31572788570513816]

Parameter: [Total Process: 12, Total Scheduling Time: 6.236359784051436, Total Execution Time for all process: 24.618667598967377, Total Execution Cost for all process: 1329.9400906396893, Average Processing Time: 2.0515556332472813, Average Processing Cost: 110.82834088664077, Average Quality: 89.00214783260219, Average Reliability: 93.0888417444905, Fitness: 0.31572788570513816]

---------- Elistist Ant Colony Optimization (EACO) ----------

Initial fittest population EACO: Schedule
[mnfcTypeArray=[5, 6, 2, 2, 10, 9, 11, 5, 7, 13, 5, 10], sequenceSchedule=[4, 0, 5, 4, 3, 0, 2, 1, 3, 2, 5, 1], fitness=0.22502520268966394]

Parameter: [Total Process: 12, Total Scheduling Time: 8.13802277515496, Total Execution Time for all process: 26.977654695564976, Total Execution Cost for all process: 1481.8839307929745, Average Processing Time: 2.2481378912970813, Average Processing Cost: 123.4903275660812, Average Quality: 86.99773914964811, Average Reliability: 93.77026042098277, Fitness: 0.22502520268966394]

---------- START SIMULATING ----------

Elitist Ant Colony Fittest Schedule 100 iteration : Schedule
[mnfcTypeArray=[11, 8, 14, 2, 10, 9, 11, 5, 5, 15, 7, 10], sequenceSchedule=[1, 2, 3, 2, 3, 4, 4, 1, 5, 0, 5, 0], fitness=0.2825547215627184]

Parameter: [Total Process: 12, Total Scheduling Time: 7.708916497396623, Total Execution Time for all process: 24.952590080958345, Total Execution Cost for all process: 1383.4419446929248, Average Processing Time: 2.0793825067465286, Average Processing Cost: 115.2868287244104, Average Quality: 87.20938497309278, Average Reliability: 93.42264985280714, Fitness: 0.2825547215627184]

Elitist Ant Colony Fittest Schedule 200 iteration : Schedule
[mnfcTypeArray=[11, 8, 14, 14, 10, 9, 5, 5, 5, 15, 5, 10], sequenceSchedule=[2, 0, 1, 0, 3, 4, 3, 4, 5, 5, 2, 1], fitness=0.29439567150171597]

Parameter: [Total Process: 12, Total Scheduling Time: 10.882729017972776, Total Execution Time for all process: 26.30798263840495, Total Execution Cost for all process: 1350.434432262796, Average Processing Time: 2.1923318865337458, Average Processing Cost: 112.53620268856633, Average Quality: 89.32876277197879, Average Reliability: 93.42564514427096, Fitness: 0.29439567150171597]

Elitist Ant Colony Fittest Schedule 500 iteration : Schedule
[mnfcTypeArray=[11, 8, 14, 14, 10, 10, 11, 5, 5, 15, 11, 9], sequenceSchedule=[2, 2, 3, 1, 1, 3, 4, 4, 5, 5, 0, 0], fitness=0.29492810533339625]

Parameter: [Total Process: 12, Total Scheduling Time: 7.708916497396623, Total Execution Time for all process: 24.525359810491633, Total Execution Cost for all process: 1362.1459315871384, Average Processing Time: 2.043779984207636, Average Processing Cost: 113.51216096559487, Average Quality: 87.12901657661367, Average Reliability: 93.42791737245649, Fitness: 0.29492810533339625]

Elitist Ant Colony Fittest Schedule 1000 iteration : Schedule
[mnfcTypeArray=[11, 8, 14, 14, 10, 10, 5, 5, 5, 15, 5, 10], sequenceSchedule=[1, 2, 0, 2, 3, 4, 3, 5, 1, 5, 4, 0], fitness=0.310822096490861]

Parameter: [Total Process: 12, Total Scheduling Time: 10.882729017972776, Total Execution Time for all process: 25.78113964291564, Total Execution Cost for all process: 1327.5845213634352, Average Processing Time: 2.148428303576303, Average Processing Cost: 110.63204344695293, Average Quality: 90.10202093028472, Average Reliability: 93.08770563039774, Fitness: 0.310822096490861]

Elitist Ant Colony Fittest Schedule 2000 iteration : Schedule
[mnfcTypeArray=[11, 8, 14, 14, 9, 10, 11, 5, 5, 15, 5, 10], sequenceSchedule=[2, 2, 1, 3, 3, 4, 4, 5, 1, 0, 5, 0], fitness=0.3088022989151687]

Parameter: [Total Process: 12, Total Scheduling Time: 9.720256974024519, Total Execution Time for all process: 24.47505459174003, Total Execution Cost for all process: 1349.6536146764404, Average Processing Time: 2.0395878826450025, Average Processing Cost: 112.47113455637003, Average Quality: 88.22888967429623, Average Reliability: 93.4267812583637, Fitness: 0.3088022989151687]

Elitist Ant Colony Fittest Schedule 5000 iteration : Schedule
[mnfcTypeArray=[11, 8, 14, 14, 10, 10, 5, 5, 5, 15, 5, 10], sequenceSchedule=[2, 2, 1, 0, 3, 1, 4, 4, 5, 5, 3, 0], fitness=0.3108220964908607]

Parameter: [Total Process: 12, Total Scheduling Time: 9.870539707750138, Total Execution Time for all process: 25.781139642915633, Total Execution Cost for all process: 1327.5845213634352, Average Processing Time: 2.1484283035763028, Average Processing Cost: 110.63204344695293, Average Quality: 90.10202093028472, Average Reliability: 93.0877056303977, Fitness: 0.3108220964908607]

Elitist Ant Colony Fittest Schedule 10000 iteration : Schedule
[mnfcTypeArray=[11, 8, 14, 14, 10, 10, 11, 5, 5, 15, 5, 10], sequenceSchedule=[1, 0, 1, 3, 3, 2, 4, 0, 5, 2, 5, 4], fitness=0.3123019025394145]

Parameter: [Total Process: 12, Total Scheduling Time: 12.405319279028125, Total Execution Time for all process: 24.821164780873612, Total Execution Cost for all process: 1332.445442136997, Average Processing Time: 2.0684303984061345, Average Processing Cost: 111.03712017808307, Average Quality: 89.00214783260218, Average Reliability: 93.0888417444905, Fitness: 0.3123019025394145]

Elitist Ant Colony Fittest Schedule 20000 iteration : Schedule [mnfcTypeArray=[11, 8, 14, 14, 10, 10, 11, 5, 5, 15, 5, 10], sequenceSchedule=[1, 0, 1, 3, 3, 2, 4, 0, 5, 2, 5, 4], fitness=0.3123019025394145]

Parameter: [Total Process: 12, Total Scheduling Time: 12.405319279028125, Total Execution Time for all process: 24.821164780873612, Total Execution Cost for all process: 1332.445442136997, Average Processing Time: 2.0684303984061345, Average Processing Cost: 111.03712017808307, Average Quality: 89.00214783260218, Average Reliability: 93.0888417444905, Fitness: 0.3123019025394145]

Elitist Ant Colony Fittest Schedule 50000 iteration : Schedule [mnfcTypeArray=[11, 8, 14, 14, 10, 10, 11, 5, 5, 15, 5, 10], sequenceSchedule=[1, 0, 1, 3, 3, 2, 4, 0, 5, 2, 5, 4], fitness=0.3123019025394145]

Parameter: [Total Process: 12, Total Scheduling Time: 12.405319279028125, Total Execution Time for all process: 24.821164780873612, Total Execution Cost for all process: 1332.445442136997, Average Processing Time: 2.0684303984061345, Average Processing Cost: 111.03712017808307, Average Quality: 89.00214783260218, Average Reliability: 93.0888417444905, Fitness: 0.3123019025394145]

Elitist Ant Colony Fittest Schedule 100000 iteration : Schedule [mnfcTypeArray=[11, 8, 14, 14, 10, 10, 11, 5, 5, 15, 5, 10], sequenceSchedule=[1, 2, 2, 3, 3, 4, 1, 5, 4, 0, 5, 0], fitness=0.31572788570513766]

Parameter: [Total Process: 12, Total Scheduling Time: 9.720256974024519, Total Execution Time for all process: 24.61866759896738, Total Execution Cost for all process: 1329.9400906396895, Average Processing Time: 2.0515556332472817, Average Processing Cost: 110.82834088664079, Average Quality: 89.00214783260218, Average Reliability: 93.08884174449048, Fitness: 0.31572788570513766]

EGA Time 100 = 0.00304061
EGA Time 200 = 0.020041103
EGA Time 500 = 0.012132776
EGA Time 1000 = 0.025090606
EGA Time 2000 = 0.049099745
EGA Time 5000 = 0.122409231
EGA Time 10000 = 0.24333005
EGA Time 20000 = 0.491442666
EGA Time 50000 = 1.220402378
EGA Time 100000 = 2.453102735
EACO Time 100 = 0.030383766
EACO Time 200 = 0.060768489
EACO Time 500 = 0.151451289
EACO Time 1000 = 0.300296981
EACO Time 2000 = 0.634507245
EACO Time 5000 = 1.941905041
EACO Time 10000 = 4.777559068
EACO Time 20000 = 7.17637191
EACO Time 50000 = 14.995619278
EACO Time 100000 = 31.359122912

Lampiran 2. JAVA *source code* simulasi

AntColonyOptimization.java

```
1 import java.util.ArrayList;2

3 public class AntColonyOptimization {
4       // ACO parameters
5       private static final double alpha = 2;
6       private static final double evaporation = 0.5;
7       private static final double initialPheromone = 0.1;8
9       // Solution Space variables
10      private static ArrayList<MnfcTypePheromoneTrack>[] mnfcTypePTArray;
11      private static SequenceSchedulePheromoneTrack[] seqSchPTArray;12
13      // Variable to store ant population
14      private static ArrayList<Schedule> antPopulationList = new ArrayList<Schedule>();15
16      //nested class of pheromone Track for mnfcTypeSchedule
17      static class MnfcTypePheromoneTrack {
18              int mnfcServiceId;
19              double pheromone;20
21              public MnfcTypePheromoneTrack(int mnfcServiceId, double pheromone) {
22                      this.mnfcServiceId = mnfcServiceId;
23              this.pheromone = pheromone;24 }
25
26              public int getMnfcServiceId() {
27              return mnfcServiceId;28             }
29
30              public void setMnfcServiceId(int mnfcServiceId) {
31              this.mnfcServiceId = mnfcServiceId;32         }
33
34              public double getPheromone() {
35              return pheromone;36 }
37
38              public void setPheromone(double pheromone) {
39              this.pheromone = pheromone;40}
41
42      }
43
44      //nested class of pheromone Track for sequenceSchedule
45      static class SequenceSchedulePheromoneTrack {
46              int scheduleId;
47              int taskId;
48              int subTaskId;
49              double pheromone;50
51              public SequenceSchedulePheromoneTrack(int scheduleId, int taskId,
52                              int subTaskId, double pheromone) {
53                      this.scheduleId = scheduleId;
54                      this.taskId = taskId;
55                      this.subTaskId = subTaskId;
56              this.pheromone = pheromone;57}
58
59              public int getScheduleId() {
60              return scheduleId;61  }
62
```

```
63              public void setScheduleId(int scheduleId) {
64              this.scheduleId = scheduleId;65   }
66
67              public int getTaskId() {
68              return taskId;69         }
70
71              public void setTaskId(int taskId) {
72              this.taskId = taskId;73 }
74
75              public int getSubTaskId() {
76              return subTaskId;77   }
78
79              public void setSubTaskId(int subTaskId) {
80              this.subTaskId = subTaskId;81     }
82
83              public double getPheromone() {
84              return pheromone;85 }
86
87              public void setPheromone(double pheromone) {
88              this.pheromone = pheromone;89}
90
91              @Override
92              public String toString() {
93                      return "SequenceSchedulePheromoneTrack [scheduleId=" + scheduleId
94                                      + ", taskId=" + taskId + ", subTaskId=" + subTaskId
95                                      + ", pheromone=" + pheromone + "]";96      }
97
98
99      }
100
101     //pheromone update
102     public static double pheromoneFormula(double initialPheromone, double fitness) {
103             return ((1-evaporation)*initialPheromone)+(fitness);
104     }
105
106     public static Schedule getFittestAnt() {
107             Schedule fittestSchedules;108
109             fittestSchedules = antPopulationList.get(0);110

111             for (int i = 0; i < antPopulationList.size(); i++) {
112                     if (fittestSchedules.getFitness() <= antPopulationList.get(i).getFitness()) {
113                             fittestSchedules = antPopulationList.get(i);
114                     }
115             }
116
117             return fittestSchedules;

118     }
119
120     public static Population antColonyOptimization(Population antPopulations, int iteration, Calculate calculate, boolean elitism) throws
    CloneNotSupportedException {
121             Population newAntPopulation = new Population(antPopulations.populationSize());
122             int antPopulationSize = newAntPopulation.populationSize();
123             //int antPopulation = calculate.getSubTaskPoolArray().size()*antFactor;
```

```
124             initializeSolutionSpace(calculate);
125             initializeAntPopulation(calculate, antPopulationSize);126
127             if (elitism) {
128                     Schedule EliteAnt = (Schedule) getFittestAnt().clone();129
130                     for (int i = 0; i < iteration; i++) {
131                             calculatePheromone(calculate);
132                             antPopulationList = (ArrayList<Schedule>) regenerateAntPopulation(calculate,
antPopulationSize).clone();
133
134                             if (getFittestAnt().getFitness() >= EliteAnt.getFitness()) {
135                                     EliteAnt = (Schedule) getFittestAnt().clone();
136                             }
137
138                             antPopulationList.set(0, EliteAnt);

139                     }
140             } else {
141                     for (int i = 0; i < iteration; i++) {
142                             calculatePheromone(calculate);
143                             antPopulationList = (ArrayList<Schedule>) regenerateAntPopulation(calculate,
antPopulationSize).clone();
144                     }
145             }
146
147             for (int i = 0; i < antPopulationList.size(); i++) {
148                     newAntPopulation.saveSchedules(i, antPopulationList.get(i));
149             }
150
151             return newAntPopulation;

152     }
153
154     public static Population antColonyOptimization(Population antPopulations, intiteration, Calculate calculate) throws
CloneNotSupportedException {
155             return antColonyOptimization(antPopulations, iteration, calculate, true);
156     }
157
158     //initialize solution space with default initial pheromone
159     public static void initializeSolutionSpace(Calculate calculate) {
160             mnfcTypePTArray = new ArrayList[calculate.getSubTaskPoolArray().size()];
161             seqSchPTArray = new SequenceSchedulePheromoneTrack[calculate.getSubTaskPoolArray().size()];
162
163             for (int i = 0; i < calculate.getSubTaskPoolArray().size(); i++) {
164                     ArrayList<MnfcTypePheromoneTrack> mnfcTypePTList = new
ArrayList<AntColonyOptimization.MnfcTypePheromoneTrack>();
165                     int mnfcType = calculate.getSubTaskPoolArray().get(i).getSubTaskManufacturingType();
166
167                     for (int j = 0; j < calculate.getServicePoolArray().size(); j++) {
168                             if (mnfcType == calculate.getServicePoolArray().get(j).getServiceManufacturingType()) {
169                                     int mnfcServiceId =
calculate.getServicePoolArray().get(j).getServiceId();
170                                     double pheromone = initialPheromone;
171                                     MnfcTypePheromoneTrack mnfcTypePheromoneTrack = new
MnfcTypePheromoneTrack(mnfcServiceId, pheromone);
172                                     mnfcTypePTList.add(mnfcTypePheromoneTrack);
173                             }
174                     }
175                     mnfcTypePTArray[i] = mnfcTypePTList;

176             }
```

```
177
178           for  (int i = 0; i < calculate.getSubTaskPoolArray().size(); i++) {
179                int subTaskId = calculate.getSubTaskPoolArray().get(i).getSubTaskId();
180                int taskId = calculate.getTaskIdfromSubTaskId(subTaskId);
181                double pheromone = initialPheromone;
182                SequenceSchedulePheromoneTrack seqSchPheromoneTrack = new
    SequenceSchedulePheromoneTrack(i, taskId, subTaskId, pheromone);
183
184                seqSchPTArray[i] = seqSchPheromoneTrack;
185           }
186      }
187
188      public static void initializeAntPopulation(Calculate calculate, int antPopulation) {
189           antPopulationList = new ArrayList<Schedule>();
190
191           for  (int i = 0; i < antPopulation; i++) {
192                ArrayList<Integer> mnfcTypeArray = calculate.generateMnfcTypeSchedule();
193                ArrayList<Integer> scheduleSequence = calculate.generateScheduleSequence();
194                double fitness = calculate.calculateFitness(mnfcTypeArray, scheduleSequence);
195                Schedule newAnt = new Schedule(mnfcTypeArray, scheduleSequence, fitness);
196                antPopulationList.add(newAnt);
197           }
198
199      }
200
201
202      //update pheromones in solution space
203      public static void calculatePheromone(Calculate calculate){
204           for (int i = 0; i < antPopulationList.size(); i ++) {
205                Schedule ant = antPopulationList.get(i);
206                double fitness = ant.getFitness();207
208                for (int j = 0; j < ant.getMnfcTypeArray().size(); j++) {
209                     int mnfcServiceId = ant.getMnfcTypeArray().get(j);
210                     ArrayList<MnfcTypePheromoneTrack> mnfcTypePTList = mnfcTypePTArray[j];211
212                     for (int k = 0; k < mnfcTypePTList.size(); k++) {
213                          if (mnfcTypePTList.get(k).getMnfcServiceId() == mnfcServiceId) {
214                               double modifiedPheromone =
    pheromoneFormula(mnfcTypePTArray[j].get(k).getPheromone(), fitness);
215                               mnfcTypePTArray[j].get(k).setPheromone(modifiedPheromone);
216                          }
217                     }
218
219                }
220
221                int[] taskCounter = new int[calculate.getTaskPoolArray().size()];
222                for (int j = 0 ; j < taskCounter.length; j++) {
223                     taskCounter[j] = 0;
224                }
225
226                for (int j = 0; j < ant.getSequenceSchedule().size(); j++) {
227                     int taskId = ant.getSequenceSchedule().get(j);
228                     int subTaskInTaskCounter = taskCounter[taskId];
229                     int subTaskId = calculate.getTaskPoolArray().get(taskId).getSubTaskArray().get(subTaskInTaskCounter).getSubTaskId();
230
231                     for (int k = 0; k < seqSchPTArray.length; k++) {
232                          if (seqSchPTArray[k].getSubTaskId() == subTaskId) {
233                               double modifiedPheromone =
    pheromoneFormula(seqSchPTArray[k].getPheromone(), fitness);
```

```
234                                     seqSchPTArray[k].setPheromone(modifiedPheromone);
235                                 }
236                             }
237
238                             taskCounter[taskId] += 1;
239                         }
240
241                 }
242         }
243
244         //regenerate ant population
245 public static ArrayList<Schedule> regenerateAntPopulation(Calculate calculate, intantPopulation) {
246                 ArrayList<Schedule> newAntPopulationList = new ArrayList<Schedule>();247
248                 for (int i = 0; i < antPopulation; i++) {
249                         ArrayList<Integer> mnfcTypeArray = new ArrayList<Integer>();
250                         ArrayList<Integer> scheduleSequence = new ArrayList<Integer>();251
252
253                         // fill new mnfcTypeArray
254                         for (int j = 0; j < calculate.getSubTaskPoolArray().size(); j++) {
255                                 int subTaskId = calculate.getSubTaskPoolArray().get(j).getSubTaskId();
256                                 ArrayList<MnfcTypePheromoneTrack> mnfcTypePTList =
    mnfcTypePTArray[subTaskId];
257                                 double[] probability = new double[mnfcTypePTList.size()];
258                                 double r = Math.random();
259                                 double totalPheromone = 0;260
261                                 for (int k = 0; k < mnfcTypePTList.size(); k++) {
262                                         totalPheromone += Math.pow(mnfcTypePTList.get(k).getPheromone(),alpha);
263                                 }
264
265                                 for (int k = 0; k < mnfcTypePTList.size(); k++) {266
                                 if (k > 0) {
267                                         probability[k] = probability[k-1] +
    (Math.pow(mnfcTypePTList.get(k).getPheromone(), alpha)/totalPheromone);
268                                 } else {
269                                         probability[k] = Math.pow(mnfcTypePTList.get(k).getPheromone(),alpha)/totalPheromone;
270                                 }
271                                 }
272
273                                 int selectedServiceId = -1;
274                                 int counter = 0;
275
276                                 do {
277                                     if (r <= probability[counter]) {
278                                         selectedServiceId = mnfcTypePTList.get(counter).getMnfcServiceId();
279                                         mnfcTypeArray.add(selectedServiceId);
280                                     }
281                                     counter++;
282                                 } while (selectedServiceId < 0 && counter < probability.length);
283
284                         }
285
286                         //fill new scheduleSequence
287                         ArrayList<Integer> visited = new ArrayList<Integer>();
288                         for (int j = 0; j < seqSchPTArray.length; j++) {
289                             double[] probability = new double[seqSchPTArray.length];
290                             double r = Math.random();
```

```
291                         double totalPheromone = 0;
292                         double pheromoneCounter = 0;293
294                         for (int k = 0; k < seqSchPTArray.length; k++) {
295                             if (!visited.contains(seqSchPTArray[k].getScheduleId())){
296                                 totalPheromone += Math.pow(seqSchPTArray[k].getPheromone(), alpha);
297                             }
298                         }
299
300                         for (int k = 0; k < seqSchPTArray.length; k++) {
301                             if (!visited.contains(seqSchPTArray[k].getScheduleId())){302                         if (k > 0) {
303                                 probability[k] = pheromoneCounter +
    (Math.pow(seqSchPTArray[k].getPheromone(), alpha)/totalPheromone);
304                             } else {
305                                 probability[k] = Math.pow(seqSchPTArray[k].getPheromone(),alpha)/totalPheromone;
306
307                             }
308                         } else {
309                             probability[k] = 0;
310                         }
311                         pheromoneCounter += probability[k];
312                         }
313
314                         int selectedTaskId = -1;
315                         int counter = 0;
316                         do {
317                             if (r <= probability[counter]) {
318                                 selectedTaskId = seqSchPTArray[counter].getTaskId();
319                                 scheduleSequence.add(selectedTaskId);
320                                 visited.add(seqSchPTArray[counter].getScheduleId());
321                             }
322                             counter++;
323                         } while (selectedTaskId < 0 && counter < probability.length);
324                     }
325
326                     double fitness = calculate.calculateFitness(mnfcTypeArray, scheduleSequence);
327 Schedule regeneratedAnt = new Schedule(mnfcTypeArray, scheduleSequence,fitness);
328                     newAntPopulationList.add(regeneratedAnt);
329             }
330
331             return newAntPopulationList;
332
333         }
334
335 }
336
```

Calculate.java

```
1 import java.util.ArrayList;4

5
6 public class Calculate {
7       private ArrayList<Client> clientPoolArray = new ArrayList<Client>();
8       private ArrayList<Task> taskPoolArray = new ArrayList<Task>();
9       private ArrayList<SubTask> subTaskPoolArray = new ArrayList<SubTask>();
10      private ArrayList<MnfcServiceProvider> mnfcServiceProviderPoolArray = newArrayList<MnfcServiceProvider>();
11      private ArrayList<Service> servicePoolArray = new ArrayList<Service>();
12      private ArrayList<Location> locationArray = new ArrayList<Location>();13
14      public Calculate(ArrayList<Client> clientPoolArray,
15                      ArrayList<Task> taskPoolArray, ArrayList<SubTask> subTaskPoolArray,
16                      ArrayList<MnfcServiceProvider> mnfcServiceProviderPoolArray,
17                      ArrayList<Service> servicePoolArray,
18                      ArrayList<Location> locationArray) {
19              this.clientPoolArray = clientPoolArray;
20              this.taskPoolArray = taskPoolArray;
21              this.subTaskPoolArray = subTaskPoolArray;
22              this.mnfcServiceProviderPoolArray = mnfcServiceProviderPoolArray;
23              this.servicePoolArray = servicePoolArray;
24      this.locationArray = locationArray;25     }
26
27      public int getTaskIdfromSubTaskId(int subTaskId){
28              int taskId = -1;
29              for (int i = 0; i < taskPoolArray.size(); i++) {
30                      for (int j = 0; j < taskPoolArray.get(i).getSubTaskArray().size(); j++) {
31                              if (taskPoolArray.get(i).getSubTaskArray().get(j).getSubTaskId() ==subTaskId) {
32                              taskId = taskPoolArray.get(i).getTaskId();33              }
34                      }
35              }
36
37              return taskId;38        }

39
40      public int getAnotherRandomServiceId(int serviceId) {
41              ArrayList<Integer> newServiceIdArray = new ArrayList<Integer>();
42              int newServiceId = serviceId;
43              int mnfcType = servicePoolArray.get(serviceId).getServiceManufacturingType();44
45              for (int i = 0 ; i < servicePoolArray.size(); i++) {
46                      if (servicePoolArray.get(i).getServiceManufacturingType() == mnfcType &&servicePoolArray.get(i).getServiceId() !=
    serviceId) {
47                      newServiceIdArray.add(servicePoolArray.get(i).getServiceId());48        }
49              }
50
51              if (newServiceIdArray.size()> 0) {
52                      int rnd = new Random().nextInt(newServiceIdArray.size());
53              newServiceId = newServiceIdArray.get(rnd);54            }
55
56              return newServiceId;57
        }

58
59
60      public ArrayList<Integer> generateMnfcTypeSchedule() {
61              ArrayList<Integer> mnfcTypeArray = new ArrayList<Integer>();
```

```
62
63              for (int i = 0; i < subTaskPoolArray.size(); i++) {
64                      int mnfcType = subTaskPoolArray.get(i).getSubTaskManufacturingType();
65                      ArrayList<Integer> mnfcTypeServiceArray = new ArrayList<Integer>();
66                      for (int j = 0; j < servicePoolArray.size(); j++) {
67                              if (mnfcType == servicePoolArray.get(j).getServiceManufacturingType()) {
68                              mnfcTypeServiceArray.add(servicePoolArray.get(j).getServiceId());69          }
70                      }
71
72                      int size = mnfcTypeServiceArray.size();
73                      if (size > 0) {
74                              mnfcTypeArray.add(mnfcTypeServiceArray.get(new Random().nextInt(size)));
75                      } else {
76                              System.out.println("\n Error: no manufacturing service available forsubtaskid: " + i + ", with manufacturing type: "+
    mnfcType +" \n");
77                      System.exit(0);78              }
79
80
81              }
82
83              return mnfcTypeArray;84          }

85
86      public static ArrayList<Integer> generateMnfcTypeSchedule(ArrayList<SubTask>subTaskPoolArray, ArrayList<Service>
    servicePoolArray) {
87              ArrayList<Integer> mnfcTypeArray = new ArrayList<Integer>();88
89              for (int i = 0; i < subTaskPoolArray.size(); i++) {
90                      int mnfcType = subTaskPoolArray.get(i).getSubTaskManufacturingType();
91                      ArrayList<Integer> mnfcTypeServiceArray = new ArrayList<Integer>();
92                      for (int j = 0; j < servicePoolArray.size(); j++) {
93                              if (mnfcType == servicePoolArray.get(j).getServiceManufacturingType()) {
94                              mnfcTypeServiceArray.add(servicePoolArray.get(j).getServiceId());95          }
96                      }
97
98                      int size = mnfcTypeServiceArray.size();
99                      if (size > 0) {
100                             mnfcTypeArray.add(mnfcTypeServiceArray.get(new Random().nextInt(size)));
101                     } else {
102                             System.out.println("\n Error: no manufacturing service available forsubtaskid: " + i + ", with manufacturing type: "+
    mnfcType +" \n");
103                             System.exit(0);
104                     }
105
106
107             }
108
109             return mnfcTypeArray;

110     }
111
112     public ArrayList<Integer> generateScheduleSequence() {
113             ArrayList<Integer> scheduleSequence = new ArrayList<Integer>();114
115             for (int i = 0; i < subTaskPoolArray.size(); i++) {
116                     for (int j = 0; j < taskPoolArray.size(); j++) {
                                                    117                             if
    (taskPoolArray.get(j).getSubTaskArray().contains(subTaskPoolArray.get(i))) {
118                                     scheduleSequence.add(taskPoolArray.get(j).getTaskId());
119                     }
```

```
120                 }
121           }
122
123           Collections.shuffle(scheduleSequence);
124
125           return scheduleSequence;
126      }
127
128      public static ArrayList<Integer> generateScheduleSequence(ArrayList<Task>taskPoolArray,
    ArrayList<SubTask> subTaskPoolArray) {
129           ArrayList<Integer> scheduleSequence = new ArrayList<Integer>();
130
131           for (int i = 0; i < subTaskPoolArray.size(); i++) {
132                      for (int j = 0; j < taskPoolArray.size(); j++) {
133                     if
    (taskPoolArray.get(j).getSubTaskArray().contains(subTaskPoolArray.get(i))) {
134                          scheduleSequence.add(taskPoolArray.get(j).getTaskId());
135                     }
136                }
137           }
138
139           Collections.shuffle(scheduleSequence);140
141           return scheduleSequence;

142      }
143
144      public double calculateFitness(ArrayList<Integer> mnfcTypeArray, ArrayList<Integer>scheduleSequence) {
145           return calculateFitness(new Schedule(mnfcTypeArray, scheduleSequence, 0));
146      }
147
148      public double calculateFitness(Schedule schedule) {
149           return calculateFitnessWithParemeter(schedule, false);
150      }
151
152      public double calculateFitnessWithParemeter(Schedule schedule) {
153           return calculateFitnessWithParemeter(schedule, true);
154      }
155
156      public double calculateFitnessWithParemeter(Schedule schedule, boolean isWithParameter)
    {
157           ArrayList<Integer> mnfcTypeArray = schedule.getMnfcTypeArray();
158           ArrayList<Integer> scheduleSequence = schedule.getSequenceSchedule();159
160           double fitness;161

162           double totalProcessingTime = 0;
163           double totalLogisticalTime = 0;
164           double totalExecutionTime = 0;165
166           double totalProcessingCost = 0;
167           double totalLogisticalCost = 0;
168           double totalExecutionCost = 0;169
170           double totalQuality = 0;
171           double totalReliability = 0;172
173           int[] taskCounter = new int[taskPoolArray.size()];174
175           double totalScheduleTime = 0;
176           double[] taskTimeStamp = new double[taskPoolArray.size()];
177           double[] serviceTimeStamp = new double[servicePoolArray.size()];
```

```
178
179             for  (int i = 0 ; i < taskCounter.length; i++) {
180                  taskCounter[i] = 0;
181             }
182
183             for  (int i = 0; i < taskTimeStamp.length; i++) {
184                  taskTimeStamp[i] = 0;
185             }
186
187             for  (int i = 0; i < serviceTimeStamp.length; i++) {
188                  serviceTimeStamp[i] = 0;
189             }
190
191             ArrayList<Integer> mscScheduleTasksList = new ArrayList<Integer>();192
193             //arrange urutan mnfcTypeArray sesuai scheduleSequence
194             for (int i = 0; i < scheduleSequence.size(); i++) {
195                  int executedTaskId = (scheduleSequence.get(i));
196                  Task executedTask = taskPoolArray.get(executedTaskId);
197                  ArrayList<SubTask> executedTaskSubtasks = executedTask.getSubTaskArray();
198                  int counter = taskCounter[executedTaskId];199
200                  if (counter >= executedTaskSubtasks.size()){
201                       System.out.println("\n Error counter bigger than subtask schedule sequence:" + scheduleSequence.toString() + "\n");
202                       System.exit(0);
203                  }
204
205                  SubTask executedSubtask = executedTaskSubtasks.get(counter);
206                  int executedSubTaskId = executedSubtask.getSubTaskId();
207                  int executeSubTaskServiceId = mnfcTypeArray.get(executedSubTaskId);208
209                  mscScheduleTasksList.add(executeSubTaskServiceId);
210                  taskCounter[executedTaskId]++;
211             }
212
213             //reset taskcounter
214             for (int i = 0 ; i < taskCounter.length; i++) {
215                  taskCounter[i] = 0;
216             }
217
218             int[] serviceLastPosition = new int[servicePoolArray.size()];219
220             for (int i = 0 ; i < serviceLastPosition.length; i++) {
221                  serviceLastPosition[i] = 0;
222             }
223
224
225             double initialTime = 0;226

227             for (int i = 0; i < scheduleSequence.size(); i++) {
228                  double subTaskTotalProcessingTime = 0;
229                  int initialServiceLocation;
230                  int destinationLocId;
231                  int taskId = scheduleSequence.get(i);
232                  int serviceId = mscScheduleTasksList.get(i);233
234                  if (taskCounter[scheduleSequence.get(i)] == 0) {
235                       if (serviceId > servicePoolArray.size()) {
236                            System.out.println("\n Error serviceId > servicePoolArray.size()serviceId: " +
237                                      serviceId + ", mscScheduleTasksList: " +
```

```
mscScheduleTasksList.toString() + ", servicePoolArray.size(): " + servicePoolArray.size() +"\n");
238                             System.exit(0);
239                         }
240                         initialServiceLocation =
servicePoolArray.get(serviceId).getServiceLocationIndex();
241                     } else {
242                         initialServiceLocation = taskPoolArray.get(taskId).getSubTaskArray().get(taskCounter[taskId]-1).getSubTaskLocationId
);
243                     }
244
245                     destinationLocId = taskPoolArray.get(taskId).getSubTaskArray().get(taskCounter[taskId]).getSubTaskLocationId()
246                     serviceLastPosition[serviceId] = destinationLocId;247
248                     double distancePassed = locationArray.get(destinationLocId).distanceTo(locationArray.get(initialServiceLocation));
249                     double serviceCostPerTime =
servicePoolArray.get(serviceId).getServiceCostPerTime();
250                     double serviceLogisticalCostPerDistance =
servicePoolArray.get(serviceId).getServiceTravelCostPerDistance();
251                     double serviceTravelVelocity =
servicePoolArray.get(serviceId).getServiceTravelVelocity();
252
253                     double travelCost = serviceLogisticalCostPerDistance*distancePassed;
254                     double travelTime = distancePassed/serviceTravelVelocity;255
256                     double executionTime =
taskPoolArray.get(taskId).getSubTaskArray().get(taskCounter[taskId]).getSubTaskExecutionTime();
257
258                     double executionCost = executionTime*serviceCostPerTime;259
260                     double serviceQuality = servicePoolArray.get(serviceId).getServiceQuality();
261 double serviceReliability = servicePoolArray.get(serviceId).getServiceReliability();
262
263
264                     totalProcessingTime += executionTime;
265                     totalLogisticalTime += travelTime;266
267                     totalProcessingCost += executionCost;
268                     totalLogisticalCost += travelCost;
269                     totalQuality                    += serviceQuality;
270                     totalReliability                += serviceReliability;271
272                     subTaskTotalProcessingTime = executionTime+travelTime;273
274                     if ( serviceTimeStamp[serviceId] >= taskTimeStamp[taskId]) {
275                         initialTime = serviceTimeStamp[serviceId];
276                     } else {
277                         initialTime = taskTimeStamp[taskId];
278                     }
279
280                     taskTimeStamp[taskId] = initialTime + subTaskTotalProcessingTime;
281                     serviceTimeStamp[serviceId] = initialTime + subTaskTotalProcessingTime;282
283                     taskCounter[taskId]++;284
285                     if (i == (scheduleSequence.size()-1)) {
286                         int LastLocation = destinationLocId;
```

```
287                         int EndLocation = servicePoolArray.get(serviceId).getServiceLocationIndex();
288                         double lastDistancePassed = locationArray.get(EndLocation).distanceTo(locationArray.get(LastLocation));
289
290                         double lastTravelCost = serviceLogisticalCostPerDistance*lastDistancePassed;
291                         double lastTravelTime = lastDistancePassed/serviceTravelVelocity;292
293                         totalLogisticalTime += lastTravelTime;
294                         totalLogisticalCost += lastTravelCost;295
296                         taskTimeStamp[taskId] = initialTime + lastTravelTime;
297                         serviceTimeStamp[serviceId] = initialTime + lastTravelTime;298
299                 }
300
301             }
302
303             //find actual schedule time
304             double maxTaskTime = 0;
305             for (int i = 0; i < taskTimeStamp.length; i++) {
306                 if (maxTaskTime < taskTimeStamp[i]) {
307                     maxTaskTime = taskTimeStamp[i];
308                 }
309             }
310
311             double maxServiceTime = 0;
312             for (int i = 0; i < serviceTimeStamp.length; i++) {
313                 if (maxServiceTime < serviceTimeStamp[i]) {
314                     maxServiceTime = serviceTimeStamp[i];
315                 }
316             }
317
318                     if (maxTaskTime > maxServiceTime) {
319                         totalScheduleTime = maxTaskTime;
320             } else {
321                 totalScheduleTime = maxServiceTime;
322             }
323
324             totalExecutionTime = totalProcessingTime + totalLogisticalTime;
325             totalExecutionCost = totalProcessingCost + totalLogisticalCost;
326
327             double   avgProcessingTime = totalExecutionTime/mnfcTypeArray.size();
328             double   avgProcessingCost = totalExecutionCost/mnfcTypeArray.size();
329             double   avgQuality = totalQuality/mnfcTypeArray.size();
330             double   avgReliability = totalReliability/mnfcTypeArray.size();
331
332             fitness = 0.3*((3-avgProcessingTime)/(3-1)) + 0.3*((100-avgProcessingCost)/(100-30)) + 0.2*((avgQuality-75)/(100-75)) + 0.2*((avgReliability-85)/(100-85));
333
334             if (isWithParameter) {
335                 System.out.println("Parameter: [Total Process: "+ mnfcTypeArray.size() +", " +"Total Scheduling Time: "+ totalScheduleTime +", "
336                         + "Total Execution Time for all process: "+ totalExecutionTime +", "
337                                 + "Total Execution Cost for all process: "+ totalExecutionCost +", "
338                                         + "Average Processing Time: "+ avgProcessingTime +", "
339                                                 + "Average Processing Cost: "+ avgProcessingCost +", "
340                                                         + "Average Quality: "+ avgQuality +", "
341                                                                 + "Average Reliability: "+
```

```
avgReliability +", "
342                                                                                 + "Fitness: "+ fitness
+"]\n");
343           }
344
345           return fitness;

346      }
347
348      public ArrayList<Client> getClientPoolArray() {
349           return clientPoolArray;
350      }
351
352      public void setClientPoolArray(ArrayList<Client> clientPoolArray) {
353           this.clientPoolArray = clientPoolArray;
354      }
355
356      public ArrayList<Task> getTaskPoolArray() {
357           return taskPoolArray;
358      }
359
360      public void setTaskPoolArray(ArrayList<Task> taskPoolArray) {
361           this.taskPoolArray = taskPoolArray;
362      }
363
364      public ArrayList<SubTask> getSubTaskPoolArray() {
365           return subTaskPoolArray;
366      }
367
368      public void setSubTaskPoolArray(ArrayList<SubTask> subTaskPoolArray) {
369           this.subTaskPoolArray = subTaskPoolArray;
370      }
371
372                        public ArrayList<MnfcServiceProvider> getMnfcServiceProviderPoolArray() {
373           return mnfcServiceProviderPoolArray;
374      }
375
376      public void setMnfcServiceProviderPoolArray(
377                                ArrayList<MnfcServiceProvider> mnfcServiceProviderPoolArray) {
378           this.mnfcServiceProviderPoolArray = mnfcServiceProviderPoolArray;
379      }
380
381      public ArrayList<Service> getServicePoolArray() {
382           return servicePoolArray;
383      }
384
385      public void setServicePoolArray(ArrayList<Service> servicePoolArray) {
386           this.servicePoolArray = servicePoolArray;
387      }
388
389      public ArrayList<Location> getLocationArray() {
390           return locationArray;
391      }
392
393      public void setLocationArray(ArrayList<Location> locationArray) {
394           this.locationArray = locationArray;
395      }
396
397
398 }
399
```

Client.java

```
1 import java.util.ArrayList;3

4 public class Client {
5       private static AtomicInteger count = new AtomicInteger(-1);
6       private int clientId;
7       private String clientName;
8       private int clientLocationId;
9       private ArrayList<Task> taskArray = new ArrayList<Task>();10
11      public Client(String clientName, int clientLocationId) {
12              this.clientId = count.incrementAndGet();
13              this.clientName = clientName;
14      this.clientLocationId = clientLocationId;15 }
16
17      public Client(String clientName, int clientLocationId, ArrayList<Task> taskArray) {
18              this.clientId = count.incrementAndGet();
19              this.clientName = clientName;
20              this.clientLocationId = clientLocationId;
21      this.taskArray = taskArray;22   }
23
24      public void addTask(Task task){
25      taskArray.add(task);26          }
27
28      public void removeTask(Task task){
29      taskArray.remove(task);30       }
31
32      public void clearTask(){
33      taskArray.clear();34 }
35
36      public int getClientId() {
37      return clientId;38   }
39
40      public String getClientName() {
41      return clientName;42            }
43
44      public void setClientName(String clientName) {
45      this.clientName = clientName;46             }
47
48      public int getClientLocationId() {
49      return clientLocationId;50      }
51
52      public void setClientLocationId(int clientLocationId) {
53      this.clientLocationId = clientLocationId;54 }
55
56      public ArrayList<Task> getTaskArray() {
57      return taskArray;58 }
59
60      public void setTaskArray(ArrayList<Task> taskArray) {
61      this.taskArray = taskArray;62   }
63
```

```
64      @Override
65      public String toString() {
66              return "Client [clientId=" + clientId + ", clientName=" + clientName
67                              + ", clientLocationId=" + clientLocationId
68                              /*+ ", taskArray="+ taskArray*/ + "]";69 }
70
71 }
72
```

GA.java

```
1 import java.util.ArrayList;2

3
4
5 public class GA {
6       /* GA parameters */
7       private static final double crossoverRate = 0.5;
8       private static final double mutationRate = 0.015;
9       private static final int tournamentSize = 5;10
11      public static Population evolvePopulation(Population pop, Calculate calculate, booleanelitism) {
12              Population newPopulation = new Population(pop.populationSize());13
14              int elitismOffset = 0;
15              if (elitism) {
16                      newPopulation.saveSchedules(0, pop.getFittest());
17              elitismOffset = 1;18    }
19
20              for (int i = elitismOffset; i < newPopulation.populationSize(); i++) {
21                      Schedule parent1 = tournamentSelection(pop);
22                      Schedule parent2 = tournamentSelection(pop);
23                      Schedule child = crossover(parent1, parent2, calculate);
24              newPopulation.saveSchedules(i, child);25    }
26
27              for (int i = elitismOffset; i < newPopulation.populationSize(); i++) {
28              mutate(newPopulation.getSchedules(i), calculate);29    }
30
31              return newPopulation;32         }

33
34
35      public static Population evolvePopulation(Population pop, Calculate calculate) {
36      return evolvePopulation(pop, calculate, true);37    }
38
39      private static Schedule tournamentSelection(Population pop) {40

41              Population tournament = new Population(tournamentSize);42

43              for (int i = 0; i < tournamentSize; i++) {
44                      int randomId = (int) (Math.random() * pop.populationSize());
45              tournament.saveSchedules(i, pop.getSchedules(randomId));46      }
47
48              Schedule fittest = tournament.getFittest();
49      return fittest;50   }
51
52      public static Schedule crossover(Schedule parent1, Schedule parent2, Calculatecalculate) {
53
54              Schedule child = new Schedule(parent1.scheduleSize());
55              ArrayList<Integer> childMnfcTypeArray = new ArrayList<Integer>();
56              ArrayList<Integer> childSequenceSchedule = new ArrayList<Integer>();57
58              int crossoverPoint = (int) (Math.random() * parent1.scheduleSize());59
60              int temp = -1;
```

```
61          int swapValue = -1;
62          boolean swapped = false;63
64          for(int i=0; i < child.scheduleSize(); i++){
65                  int taskIdSchedule = parent1.getSequenceSchedule().get(i);
66                  int mnfcType = parent1.getMnfcTypeArray().get(i);67
68                  if (i == crossoverPoint) {
69                          temp = taskIdSchedule;
70                          taskIdSchedule = parent2.getSequenceSchedule().get(i);
71                          swapValue = taskIdSchedule;
72                          mnfcType = parent2.getMnfcTypeArray().get(i);
73                  } else if (swapValue == taskIdSchedule && !swapped) {
74                          taskIdSchedule = temp;
75                  swapped = true;76           }
77
78                  childMnfcTypeArray.add(mnfcType);
79                  childSequenceSchedule.add(taskIdSchedule);80
81                  while (i == child.scheduleSize()-1 && !swapped) {
82                          for (int j = crossoverPoint-1; j >= 0 ; j--) {
83                                  if (swapValue == parent1.getSequenceSchedule().get(j)) {
84                                          childSequenceSchedule.set(j, temp);
85                                          swapped = true;
                                                86                                  break;
87                                  }
88                          }
89                  }
90          }
91
92
93          child.setMnfcTypeArray(childMnfcTypeArray);
94          child.setSequenceSchedule(childSequenceSchedule);
95          child.setFitness(calculate.calculateFitness(child));96
97          return child;98
        }

99
100     public static Schedule crossover2(Schedule parent1, Schedule parent2, Calculatecalculate) {
101
102         Schedule child = new Schedule(parent1.scheduleSize());
103         ArrayList<Integer> childMnfcTypeArray = new ArrayList<Integer>();
104         ArrayList<Integer> childSequenceSchedule = new ArrayList<Integer>();105
106         int crossoverPoint = (int) (Math.random() * parent1.scheduleSize());107
108         int temp = -1;
109         int swapValue = -1;
110         boolean swapped = false;111
112         for(int i=0; i < child.scheduleSize(); i++){113
114                 if(Math.random() < crossoverRate){
115                         crossoverPoint = i;
116                 }
117
118                 int taskIdSchedule = parent1.getSequenceSchedule().get(i);
119                 int mnfcType = parent1.getMnfcTypeArray().get(i);120
121                 if (i == crossoverPoint) {
```

```
122                             temp = taskIdSchedule;
123                             taskIdSchedule = parent2.getSequenceSchedule().get(i);
124                             swapValue = taskIdSchedule;
125                             mnfcType = parent2.getMnfcTypeArray().get(i);
126                     } else if (swapValue == taskIdSchedule && !swapped) {
127                             taskIdSchedule = temp;
128                             swapped = true;
129                     }
130
131                     childMnfcTypeArray.add(mnfcType);
132                     childSequenceSchedule.add(taskIdSchedule);133
134                     while (i == child.scheduleSize()-1 && !swapped) {
135                             for (int j = crossoverPoint-1; j >= 0 ; j--) {
136                                     if (swapValue == parent1.getSequenceSchedule().get(j)) {
137                                             childSequenceSchedule.set(j, temp);
138                                             swapped = true;
                                                139                                                 break;
140                                     }
141                             }
142                     }
143             }
144
145
146             child.setMnfcTypeArray(childMnfcTypeArray);
147             child.setSequenceSchedule(childSequenceSchedule);
148             child.setFitness(calculate.calculateFitness(child));149
150             return child;

151     }
152
153     private static void mutate(Schedule schedule, Calculate calculate) {
154             for(int schPos1=0; schPos1 < schedule.scheduleSize(); schPos1++){
155                     if(Math.random() < mutationRate){
156                             int schPos2 = (int) (schedule.scheduleSize() * Math.random());157
158                             while (schPos1 == schPos2) {
159                                     schPos2 = (int) (schedule.scheduleSize() * Math.random());
160                             }
161
162                             ArrayList<Integer> mutatedMnfcTypeArray = schedule.getMnfcTypeArray();
163                             ArrayList<Integer> mutatedSequenceSchedule =
    schedule.getSequenceSchedule();
164
165                             int mutatedMnfctypeServiceId =
    calculate.getAnotherRandomServiceId(mutatedMnfcTypeArray.get(schPos1));
166                             mutatedMnfcTypeArray.set(schPos1, mutatedMnfctypeServiceId);167
168                             int sequenceSchedule1 = mutatedSequenceSchedule.get(schPos1);
169                             int sequenceSchedule2 = mutatedSequenceSchedule.get(schPos2);
170                             mutatedSequenceSchedule.set(schPos1, sequenceSchedule2);
171                             mutatedSequenceSchedule.set(schPos2, sequenceSchedule1);172
173                             schedule.setMnfcTypeArray(mutatedMnfcTypeArray);
174                             schedule.setSequenceSchedule(mutatedSequenceSchedule);
175                             schedule.setFitness(calculate.calculateFitness(schedule));
176                     }
177             }
178     }
179
180 }
181
```

InitData.java

```
1 import java.util.ArrayList;3

4 public class InitData {
5       private int nClient;
6       private int maxTaskPerClient;
7       private int maxSubTaskPerTask;
8       private int nMnfcServiceProvider;
9       private int maxServicePerMServiceProvider;
10      private int maxManufacturingType;11
12      private double minServiceCostPerTime;
13      private double maxServiceCostPerTime;
14      private double minServiceQuality;
15      private double maxServiceQuality;
16      private double minServiceReliability;
17      private double maxServiceReliability;
18      private double minSubTaskExecutionTime;
19      private double maxSubTaskExecutionTime;
20      private double minserviceTravelVelocity;
21      private double maxserviceTravelVelocity;
22      private double minserviceTravelCostPerDistance;
23      private double maxserviceTravelCostPerDistance;24
25      private ArrayList<Client> clientPoolArray;
26      private ArrayList<Task> taskPoolArray;
27      private ArrayList<SubTask> subTaskPoolArray;
28      private ArrayList<MnfcServiceProvider> mnfcServiceProviderPoolArray;
29      private ArrayList<Service> servicePoolArray;
30      private ArrayList<Location> locationArray;31
32
33      public InitData(int nClient, int maxTaskPerClient, int maxSubTaskPerTask,
34                      int nMnfcServiceProvider, int maxServicePerMServiceProvider, intmaxManufacturingType,
35                      double minServiceCostPerTime, double maxServiceCostPerTime,
36                      double minServiceQuality, double maxServiceQuality,
37                      double minServiceReliability, double maxServiceReliability,
38                      double minSubTaskExecutionTime, double maxSubTaskExecutionTime,
39                      double minserviceTravelVelocity, double maxserviceTravelVelocity,
40                      double minserviceTravelCostPerDistance, double maxserviceTravelCostPerDistance,
41                      ArrayList<Client> clientPoolArray,
42                      ArrayList<Task> taskPoolArray,
43                      ArrayList<SubTask> subTaskPoolArray,
44                      ArrayList<MnfcServiceProvider> mnfcServiceProviderPoolArray,
45                      ArrayList<Service> servicePoolArray,
46                      ArrayList<Location> locationArray) {
47              super();
48              this.nClient = nClient;
49              this.maxTaskPerClient = maxTaskPerClient;
50              this.maxSubTaskPerTask = maxSubTaskPerTask;
51              this.nMnfcServiceProvider = nMnfcServiceProvider;
52              this.maxServicePerMServiceProvider = maxServicePerMServiceProvider;
53              this.maxManufacturingType = maxManufacturingType;54
55              this.minServiceCostPerTime = minServiceCostPerTime;
56              this.maxServiceCostPerTime = maxServiceCostPerTime;
57              this.minServiceQuality = minServiceQuality;
58              this.maxServiceQuality = maxServiceQuality;
59              this.minServiceReliability = minServiceReliability;
60              this.maxServiceReliability = maxServiceReliability;
61              this.minSubTaskExecutionTime = minSubTaskExecutionTime;
62              this.maxSubTaskExecutionTime = maxSubTaskExecutionTime;
```

```
63              this.minserviceTravelVelocity = minserviceTravelVelocity;
64              this.maxserviceTravelVelocity = maxserviceTravelVelocity;
65              this.minserviceTravelCostPerDistance = minserviceTravelCostPerDistance;
66              this.maxserviceTravelCostPerDistance = maxserviceTravelCostPerDistance;67
68              this.clientPoolArray = clientPoolArray;
69              this.taskPoolArray = taskPoolArray;
70              this.subTaskPoolArray = subTaskPoolArray;
71              this.mnfcServiceProviderPoolArray = mnfcServiceProviderPoolArray;
72              this.servicePoolArray = servicePoolArray;
73      this.locationArray = locationArray;74     }
75
76      // init data for random number
77      public void initDataRandom() {78
79              Random random = new Random();80
81              // generate client, order, task, subtask
82              for (int i = 1; i < nClient; i++) {
83                      String clientName = "Client "+i;
84                      Location clientLocation = new Location("Location "+clientName);
85                      int clientLocationId = clientLocation.getLocationId();
86                      ArrayList<Task> taskArray = new ArrayList<Task>();87
88                      for (int j = 1; j < random.nextInt(maxTaskPerClient); j++) {
89                              String taskName = "Task "+i+","+j;
90                              int taskLocationId = clientLocationId;
91                              ArrayList<SubTask> subTaskArray = new ArrayList<SubTask>();92
93                              for (int k = 1; k < random.nextInt(maxSubTaskPerTask); k++) {
94                                      String subTaskName = "SubTask "+i+","+j+","+k;
95                                      int subTaskManufacturingType = random.nextInt(maxManufacturingType);
96                                      int subTaskLocationId = taskLocationId;
97                                      double subTaskExecutionTime = minSubTaskExecutionTime +
    random.nextDouble()*(maxSubTaskExecutionTime - minSubTaskExecutionTime);
98
99                                      SubTask subTask = new SubTask(subTaskName, subTaskManufacturingType,subTaskLocationId,
    subTaskExecutionTime);
100                                     subTaskArray.add(k-1, subTask);
101                                     subTaskPoolArray.add(subTask);
102                             }
103
104                             Task task = new Task(taskName, taskLocationId, subTaskArray);
105                             taskArray.add(j-1, task);
106                             taskPoolArray.add(task);
107                     }
108
109                     Client client = new Client(clientName, clientLocationId, taskArray);
110                     clientPoolArray.add(client);
111                     locationArray.add(clientLocation);
112             }
113
114             // generate manufacturingServiceProvider, service
115             for (int i = 1; i < nMnfcServiceProvider; i++) {
116                     String mnfcServiceProviderName = "Mnfc Service Provider "+i;
117                     Location mnfcServiceProviderLocation = new Location("Location"+mnfcServiceProviderName);
118                     int mnfcServiceProviderLocId = mnfcServiceProviderLocation.getLocationId();
119                     ArrayList<Service> serviceArray = new ArrayList<Service>();120
120                     for (int j = 1; j < random.nextInt(maxServicePerMServiceProvider); j++) {
121
```

```
122                         String serviceName = "Service "+i+","+j;
123                         int serviceManufacturingType = random.nextInt(maxManufacturingType);
124                         double serviceCostPerTime = minServiceCostPerTime + random.nextDouble()*(maxServiceCostPerTime -
    minServiceCostPerTime);
125                         double serviceQuality = minServiceQuality + random.nextDouble()*(maxServiceQuality - minServiceQuality);
126                         double serviceReliability = minServiceReliability + random.nextDouble()*(maxServiceReliability - minServiceReliability);
127                         int serviceLocationIndex = mnfcServiceProviderLocId;
128                         double serviceTravelVelocity = minserviceTravelVelocity +
    random.nextDouble()*(maxserviceTravelVelocity - minserviceTravelVelocity);
129                         double serviceTravelCostPerDistance = minserviceTravelCostPerDistance +
    random.nextDouble()*(maxserviceTravelCostPerDistance - minserviceTravelCostPerDistance);
130
131                         Service service = new Service(serviceName, serviceManufacturingType,serviceCostPerTime, serviceQuality,
    serviceReliability, serviceLocationIndex,
132                                 serviceTravelVelocity, serviceTravelCostPerDistance);133
134                         serviceArray.add(j-1, service);
135                         servicePoolArray.add(service);
136                 }
137
138                 MnfcServiceProvider mnfcServiceProvider = new
    MnfcServiceProvider(mnfcServiceProviderName, mnfcServiceProviderLocId, serviceArray);
139                 mnfcServiceProviderPoolArray.add(mnfcServiceProvider);
140                 locationArray.add(mnfcServiceProviderLocation);
141             }
142
143         }
144
145         // init data for max number
146         public void initDataMax() {147
148             Random random = new Random();149
150             // generate client, order, task, subtask
151             for (int i = 1; i < nClient; i++) {
152                 String clientName = "Client "+i;
153                 Location clientLocation = new Location("Location "+clientName);
154                 int clientLocationId = clientLocation.getLocationId();155
156                 ArrayList<Task> taskArray = new ArrayList<Task>();157
158                 for (int j = 1; j < maxTaskPerClient; j++) {
159                     String taskName = "Task "+i+","+j;
160                     int taskLocationId = clientLocationId;
161                     ArrayList<SubTask> subTaskArray = new ArrayList<SubTask>();162
163                     for (int k = 1; k < maxSubTaskPerTask; k++) {
164                         String subTaskName = "SubTask "+i+","+j+","+k;
165                         int subTaskManufacturingType = random.nextInt(maxManufacturingType);
166                         int subTaskLocationId = taskLocationId;
167                         double subTaskExecutionTime = minSubTaskExecutionTime +
    random.nextDouble()*(maxSubTaskExecutionTime - minSubTaskExecutionTime);
168
169                         SubTask subTask = new SubTask(subTaskName, subTaskManufacturingType,subTaskLocationId,
    subTaskExecutionTime);
170                         subTaskArray.add(k-1, subTask);
171                         subTaskPoolArray.add(subTask);
172                     }
173
174                     Task task = new Task(taskName, taskLocationId, subTaskArray);
```

```
                    taskArray.add(j-1, task);
                    taskPoolArray.add(task);
                }

                Client client = new Client(clientName, clientLocationId, taskArray);
                clientPoolArray.add(client);
                locationArray.add(clientLocation);
            }

            // generate manufacturingServiceProvider, service
            for (int i = 1; i < nMnfcServiceProvider; i++) {
                String mnfcServiceProviderName = "Mnfc Service Provider "+i;
                Location mnfcServiceProviderLocation = new Location("Location"+mnfcServiceProviderName);
                int mnfcServiceProviderLocId = mnfcServiceProviderLocation.getLocationId();189
                ArrayList<Service> serviceArray = new ArrayList<Service>();191
                for (int j = 1; j < maxServicePerMServiceProvider; j++) {
                    String serviceName = "Service "+i+","+j;
                    int serviceManufacturingType = random.nextInt(maxManufacturingType);
                    double serviceCostPerTime = minServiceCostPerTime + random.nextDouble()*(maxServiceCostPerTime -
minServiceCostPerTime);
                    double serviceQuality = minServiceQuality + random.nextDouble()*(maxServiceQuality - minServiceQuality);
                    double serviceReliability = minServiceReliability + random.nextDouble()*(maxServiceReliability - minServiceReliability);
                    int serviceLocationIndex = mnfcServiceProviderLocId;
                    double serviceTravelVelocity = minserviceTravelVelocity +
random.nextDouble()*(maxserviceTravelVelocity - minserviceTravelVelocity);
                    double serviceTravelCostPerDistance = minserviceTravelCostPerDistance +
random.nextDouble()*(maxserviceTravelCostPerDistance - minserviceTravelCostPerDistance);

                    Service service = new Service(serviceName, serviceManufacturingType,serviceCostPerTime, serviceQuality,
serviceReliability, serviceLocationIndex,
                            serviceTravelVelocity, serviceTravelCostPerDistance);204
                    serviceArray.add(j-1, service);
                    servicePoolArray.add(service);
                }

                MnfcServiceProvider mnfcServiceProvider = new
MnfcServiceProvider(mnfcServiceProviderName, mnfcServiceProviderLocId, serviceArray);
                mnfcServiceProviderPoolArray.add(mnfcServiceProvider);
                locationArray.add(mnfcServiceProviderLocation);
            }

        }

        public ArrayList<Client> getClientPoolArray() {
            return clientPoolArray;
        }

        public ArrayList<Task> getTaskPoolArray() {
            return taskPoolArray;
        }

        public ArrayList<SubTask> getSubTaskPoolArray() {
            return subTaskPoolArray;
        }

        public ArrayList<MnfcServiceProvider> getMnfcServiceProviderPoolArray() {
```

```
229             return mnfcServiceProviderPoolArray;
230         }
231
232         public ArrayList<Service> getServicePoolArray() {
233             return servicePoolArray;
234         }
235
236         public ArrayList<Location> getLocationArray() {
237             return locationArray;
238         }
239
240 }
241
```

Location.java

```
1 import    java.util.concurrent.atomic.AtomicInteger;2
3 public class Location {
4       private static AtomicInteger count = new AtomicInteger(-1);
5       private int locationId;
6       private String locaitonName;
                                                        7       private int x;
8       private int y;9
10      public Location(String locaitonName) {
11            this.locationId = count.incrementAndGet();
12            this.locaitonName = locaitonName;
13            this.x = (int)(Math.random()*100);
14      this.y = (int)(Math.random()*100);15      }
16
17      public Location(String locaitonName, int x, int y) {
18            this.locationId = count.incrementAndGet();
19            this.locaitonName = locaitonName;
20              this.x = x;
21              this.y = y;
22      }
23
24      public int    getLocationId() {
25            return   locationId;
26      }
27
28      /*public void setLocationId(int locationId) {
29            this.locationId = locationId;
30      }*/
31
32      public String getLocaitonName() {
33            return locaitonName;
34      }
35
36      public void setLocaitonName(String locaitonName) {
37            this.locaitonName = locaitonName;
38      }
39
40      public int getX() {
41            return x;
42      }
43
44      public void setX(int x) {
45            this.x = x;
46      }
47
48      public int getY() {
49            return y;
50      }
51
52      public void setY(int y) {
53            this.y = y;
54      }
55
56      public double distanceTo(Location location){
57            int xDistance = Math.abs(getX() - location.getX());
58            int yDistance = Math.abs(getY() - location.getY());
59            double distance = Math.sqrt( (xDistance*xDistance) + (yDistance*yDistance)                    );
60
61            return distance;
62      }
```

```
63
64          @Override
65          public String toString() {
66                  return "Location [locationId=" + locationId + ", locaitonName="67                  + locaitonName + ", x=" + x
+ ", y=" + y + "]";
68          }
69
70 }
71
```

MnfcServiceProvider.java

```
1 import java.util.ArrayList;3

4 public class MnfcServiceProvider {
5       private static AtomicInteger count = new AtomicInteger(-1);
6       private int mnfcServiceProviderId;
7       private String mnfcServiceProviderName;
8       private int mnfcServiceProviderLocId;
9       private ArrayList<Service> serviceArray = new ArrayList<Service>();10
11      public MnfcServiceProvider(String mnfcServiceProviderName, int mnfcServiceProviderLocId)
  {
12            this.mnfcServiceProviderId = count.incrementAndGet();
13            this.mnfcServiceProviderName = mnfcServiceProviderName;
14      this.mnfcServiceProviderLocId = mnfcServiceProviderLocId;15   }
16
17      public MnfcServiceProvider(String mnfcServiceProviderName, int mnfcServiceProviderLocId,ArrayList<Service>
  serviceArray) {
18          this.mnfcServiceProviderId = count.incrementAndGet();
19          this.mnfcServiceProviderName = mnfcServiceProviderName;
20          this.mnfcServiceProviderLocId = mnfcServiceProviderLocId;
21          this.serviceArray = serviceArray;
22      }
23
24      public void addService(Service service){
25          serviceArray.add(service);
26      }
27
28      public void removeService(Service service){
29          serviceArray.remove(service);
30      }
31
32      public void clearService(){
33          serviceArray.clear();
34      }
35
36      public int getMnfcServiceProviderId() {
37          return mnfcServiceProviderId;
43
38      }
39
40      public String getMnfcServiceProviderName() {
41          return mnfcServiceProviderName;
42      }

44      public void setMnfcServiceProviderName(String mnfcServiceProviderName) {
45          this.mnfcServiceProviderName = mnfcServiceProviderName;
46      }
47
48      public int getMnfcServiceProviderLocId() {
49          return mnfcServiceProviderLocId;
50      }
51
52      public void setMnfcServiceProviderLocId(int mnfcServiceProviderLocId) {
53          this.mnfcServiceProviderLocId = mnfcServiceProviderLocId;
54      }
55
56      public ArrayList<Service> getServiceArray() {
57          return serviceArray;
58      }
59
60      public void setServiceArray(ArrayList<Service> serviceArray) {
61          this.serviceArray = serviceArray;
```

```
62      }
63
64      @Override
65      public String toString() {
66              return "MnfcServiceProvider [mnfcServiceProviderId="
67                      + mnfcServiceProviderId + ", mnfcServiceProviderName="
68                      + mnfcServiceProviderName + ", mnfcServiceProviderLocId="
69                      + mnfcServiceProviderLocId /*+ ", serviceArray=" + serviceArray*/70      + "]";
71      }
72
73 }
74
```

Population.java

```
1 import java.util.ArrayList;3

4 public class Population implements Cloneable {
5       private Schedule[] schedules;6
7       public Population(int populationSize) {
8       this.schedules = new Schedule[populationSize];9      }
10
11      public Population(int populationSize, Calculate calculate) {
12              this.schedules = new Schedule[populationSize];13
14              for (int i = 0; i < populationSize(); i++) {
15                      ArrayList<Integer> mnfcTypeArray = calculate.generateMnfcTypeSchedule();
16                      ArrayList<Integer> scheduleSequence = calculate.generateScheduleSequence();
17                      double fitness = calculate.calculateFitness(mnfcTypeArray, scheduleSequence);18
19                      Schedule newSchedule = new Schedule(mnfcTypeArray, scheduleSequence, fitness);
20              saveSchedules(i, newSchedule);21  }
22      }
23
24      public Object clone() throws CloneNotSupportedException {
25      return super.clone();26          }
27
28      public void saveSchedules(int index, Schedule schedule) {
29      schedules[index] = schedule;30              }
31
32      public Schedule getSchedules(int index) {
33      return schedules[index];34      }
35
36      public Schedule getFittest() {
37              Schedule fittest = schedules[0];38
39              for (int i = 1; i < populationSize(); i++) {
40                      if (fittest.getFitness() <= getSchedules(i).getFitness()) {
41                      fittest = getSchedules(i);42   }
43              }
44
45              return fittest;46           }

47
48      public int populationSize() {
49      return schedules.length;50      }
51
52      @Override
53      public String toString() {
54      return "Population [schedules=" + Arrays.toString(schedules) + "]";55        }
56
57 }
58
```

Schedule.java

```
1 import java.util.ArrayList;2

3 public class Schedule implements Cloneable {
4       private ArrayList<Integer> mnfcTypeArray = new ArrayList<Integer>();
5       private ArrayList<Integer> sequenceSchedule = new ArrayList<Integer>();
6       private double fitness;7
8       public Schedule(ArrayList<Integer> mnfcTypeArray,
9                       ArrayList<Integer> sequenceSchedule, double fitness) {
10              this.mnfcTypeArray = mnfcTypeArray;
11              this.sequenceSchedule = sequenceSchedule;
12      this.fitness = fitness;13           }
14
15      public Schedule(int scheduleSize) {
16              ArrayList<Integer> tempMnfcTypeArray = new ArrayList<Integer>();
17              ArrayList<Integer> tempSequenceSchedule = new ArrayList<Integer>();18
19              for (int i = 0; i < scheduleSize; i++) {
20                      tempMnfcTypeArray.add(null);
21              tempSequenceSchedule.add(null);22             }
23
24              this.mnfcTypeArray = tempMnfcTypeArray;
25              this.sequenceSchedule = tempSequenceSchedule;
26      this.fitness = 0;27    }
28
29      public Object clone() throws CloneNotSupportedException {
30      return super.clone();31          }
32
33      public ArrayList<Integer> getMnfcTypeArray() {
34      return mnfcTypeArray;35       }
36
37      public void setMnfcTypeArray(ArrayList<Integer> mnfcTypeArray) {
38      this.mnfcTypeArray = mnfcTypeArray;39  }
40
41      public ArrayList<Integer> getSequenceSchedule() {
42      return sequenceSchedule;43    }
44
45      public void setSequenceSchedule(ArrayList<Integer> sequenceSchedule) {
46      this.sequenceSchedule = sequenceSchedule;47        }
48
49      public double getFitness() {
50      return fitness;51     }
52
53      public void setFitness(double fitness) {
54      this.fitness = fitness;55          }
56
57      public int scheduleSize() {
58      return sequenceSchedule.size();59          }
60
61      @Override
62      public String toString() {
```

```
63          return   "Schedule \n[mnfcTypeArray="
64                    + mnfcTypeArray + ", sequenceSchedule=" + sequenceSchedule
65                    + ", fitness=" + fitness + "]\n";
66      }
67
68  }
69
```

SchedulingSimulator.java – Main

```
import java.util.ArrayList;

public class SchedulingSimulator {

    public static double[][] setLogisticalWeight(
    ArrayList<Location> locationArray) {
    int totalSize = locationArray.size();
    double[][] logisticalWeight = new double[totalSize][totalSize];

                for (int i = 0; i < totalSize; i++) {
                for (int j = 0; j <= i; j++) {
                if (i == j)
                logisticalWeight[i][j] = 0;
                else {
                logisticalWeight[i][j] = locationArray.get(i).distanceTo(
                locationArray.get(j));
                logisticalWeight[j][i] = logisticalWeight[i][j];
                }
                }
                }

    return logisticalWeight;
    }

        public static void printAllArray(ArrayList<Client> clientPoolArray,
        ArrayList<Task> taskPoolArray, ArrayList<SubTask> subTaskPoolArray,
        ArrayList<MnfcServiceProvider> mnfcServiceProviderPoolArray,
        ArrayList<Service> servicePoolArray,
        ArrayList<Location> locationArray, double[][] logisticalWeight) {
        int totalLocation = clientPoolArray.size()
        + mnfcServiceProviderPoolArray.size();

    System.out.println("List Client: \n");
    for (int i = 0; i < clientPoolArray.size(); i++) {
    System.out.println(clientPoolArray.get(i).toString() + "\n");
    }

    System.out.println("List Task: \n");
    for (int i = 0; i < taskPoolArray.size(); i++) {
    System.out.println(taskPoolArray.get(i).toString() + "\n");
    }

    System.out.println("List subTask: \n");
    for (int i = 0; i < subTaskPoolArray.size(); i++) {
    System.out.println(subTaskPoolArray.get(i).toString() + "\n");
    }

            System.out.println("List MnfcServiceProvider: \n");
            for (int i = 0; i < mnfcServiceProviderPoolArray.size(); i++) {
            System.out.println(mnfcServiceProviderPoolArray.get(i).toString()
            + "\n");
            }

    System.out.println("List Service: \n");
    for (int i = 0; i < servicePoolArray.size(); i++) {
    System.out.println(servicePoolArray.get(i).toString() + "\n");
    }

    System.out.println("List Location: \n");
    for (int i = 0; i < locationArray.size(); i++) {
    System.out.println(locationArray.get(i).toString() + "\n");
    }

            System.out.println("Logistical weight: \n");
            for (int i = 0; i < totalLocation; i++) {
```

```
                    for (int j = 0; j < totalLocation; j++) {
                    System.out.println("Logistical weight ["
                    + locationArray.get(i).getLocaitonName() + ", "
                    + locationArray.get(j).getLocaitonName() + "] : "
                    + String.format("%.2f", logisticalWeight[i][j]) + "\n");
                    }

    }
    }

        public static void simulation(ArrayList<Long> timeStorageList)
        throws CloneNotSupportedException {
        // random input parameter (nilai minus 1)
        int nClient = 3;
        int maxTaskPerClient = 2;
        int maxSubTaskPerTask = 2;
        int nMnfcServiceProvider = 4;
        int maxServicePerMServiceProvider = 4;
        int maxManufacturingType = 5;

    // QOS parameter to determine scheduling scheme performance
    double minServiceCostPerTime = 30; // minimum Service cost per hour in Rp. 1000
    double maxServiceCostPerTime = 100; // maximum Service cost per hour in Rp. 1000
    double minServiceQuality = 75; // minimum Service Quality in percentage
    double maxServiceQuality = 100; // minimum Service Quality in percentage
    double minServiceReliability = 85; // minimum Service Reliability in percentage
    double maxServiceReliability = 100; // maximum Service Reliability in percentage
    double minSubTaskExecutionTime = 1; // minimum execution time in hour
    double maxSubTaskExecutionTime = 3; // maximum execution time in hour
    double minserviceTravelVelocity = 30; // minimum service travel velocity in kmph
    double maxserviceTravelVelocity = 80; // maximum service travel velocity in kmph
    double minserviceTravelCostPerDistance = 0.1; // minimum service travel cost per km in Rp. 1000
    double maxserviceTravelCostPerDistance = 0.3; // maximum service travel cost per km in Rp. 1000

    //offset data
    nClient++;
    nMnfcServiceProvider++;
    maxSubTaskPerTask++;
    nMnfcServiceProvider++;
    maxServicePerMServiceProvider++;

    // Array for storing initial data
    ArrayList<Client> clientPoolArray = new ArrayList<Client>();
    ArrayList<Task> taskPoolArray = new ArrayList<Task>();
    ArrayList<SubTask> subTaskPoolArray = new ArrayList<SubTask>();
    ArrayList<MnfcServiceProvider> mnfcServiceProviderPoolArray = new ArrayList<MnfcServiceProvider>();
    ArrayList<Service> servicePoolArray = new ArrayList<Service>(); 114 ArrayList<Location>
        locationArray = new ArrayList<Location>(); 115
                InitData initData = new InitData(nClient, maxTaskPerClient,
                maxSubTaskPerTask, nMnfcServiceProvider,
                maxServicePerMServiceProvider, maxManufacturingType,
                minServiceCostPerTime, maxServiceCostPerTime,
                minServiceQuality, maxServiceQuality, minServiceReliability,
                maxServiceReliability, minSubTaskExecutionTime,
                maxSubTaskExecutionTime, minserviceTravelVelocity,
                maxserviceTravelVelocity, minserviceTravelCostPerDistance,
                maxserviceTravelCostPerDistance, clientPoolArray,
                taskPoolArray, subTaskPoolArray, mnfcServiceProviderPoolArray,
                servicePoolArray, locationArray);

    // initData.initDataRandom();
    initData.initDataMax();

                clientPoolArray = initData.getClientPoolArray();
                taskPoolArray = initData.getTaskPoolArray();
                subTaskPoolArray = initData.getSubTaskPoolArray();
```

```
134 mnfcServiceProviderPoolArray = initData
135 .getMnfcServiceProviderPoolArray();
136 servicePoolArray = initData.getServicePoolArray();
137 locationArray = initData.getLocationArray();
138
139 // print
140 double[][] logisticalWeight = setLogisticalWeight(locationArray);
141 printAllArray(clientPoolArray, taskPoolArray, subTaskPoolArray,
142 mnfcServiceProviderPoolArray, servicePoolArray, locationArray,
143 logisticalWeight);
144
145 final Calculate calculate = new Calculate(clientPoolArray,
146 taskPoolArray, subTaskPoolArray, mnfcServiceProviderPoolArray,
147 servicePoolArray, locationArray);
148
149     Population initPopulation  = new Population(10, calculate);
150     Population population = (Population) initPopulation.clone();
151     Population populationACO = (Population) initPopulation.clone(); 152
153                     System.out.println("\n----------- Elistist Genetic Algorithm (EGA)
    -----------\n");
154
155     System.out.println("\n Population : " + population.toString() + "\n");
156     System.out.println("\nInitial fittest population: "
157     + population.getFittest().toString());
158     calculate.calculateFitnessWithParemeter(population.getFittest()); 159
160     System.out.println("\n----------- START SIMULATING -----------\n"); 161
162     // Evolve population for 100 generations
163     population = (Population) initPopulation.clone();
164     long startTime = System.nanoTime();
165     for (int i = 0; i < 100; i++) {
166     population = GA.evolvePopulation(population, calculate);
167     }
168     // System.out.println("\n GA Evolved Population 100 : " +
169     // population.toString() + "\n");
170     System.out.println("\nEGA Fittest Schedule 100 : "
171     + population.getFittest().toString());
172     long stopTime = System.nanoTime();
173     long timeStamp = stopTime - startTime;
174     timeStorageList.add(timeStamp);
175     String timeStorage = "EGA Time 100= " + timeStamp + "\n";
176     calculate.calculateFitnessWithParemeter(population.getFittest()); 177
178     // Evolve population for 200 generations
179     population = (Population) initPopulation.clone();
180     startTime = System.nanoTime();
181     for (int i = 0; i < 200; i++) {
182     population = GA.evolvePopulation(population, calculate);
183     }
184     // System.out.println("\n GA Evolved Population 200 : " +
185     // population.toString() + "\n");
186     System.out.println("\nEGA Fittest Schedule 200 : "
187     + population.getFittest().toString());
188     stopTime = System.nanoTime();
189     timeStamp = stopTime - startTime;
190     timeStorageList.add(timeStamp);
191     timeStorage = timeStorage + "EGA Time 200= " + timeStamp + "\n";
192     calculate.calculateFitnessWithParemeter(population.getFittest()); 193
194     // Evolve population for 500 generations
195     population = (Population) initPopulation.clone();
196     startTime = System.nanoTime();
197     for (int i = 0; i < 500; i++) {
198     population = GA.evolvePopulation(population, calculate);
199     }
200     // System.out.println("\n GA Evolved Population 500 : " +
201     // population.toString() + "\n");
202     System.out.println("\nEGA Fittest Schedule 500 : "
203     + population.getFittest().toString());
```

```
204     stopTime = System.nanoTime();
205     timeStamp = stopTime - startTime;
206     timeStorageList.add(timeStamp);
207     timeStorage = timeStorage + "EGA Time 500= " + timeStamp + "\n";
208     calculate.calculateFitnessWithParemeter(population.getFittest()); 209
210     // Evolve population for 1000 generations
211     population = (Population) initPopulation.clone();
212     startTime = System.nanoTime();
213     for (int i = 0; i < 1000; i++) {
214     population = GA.evolvePopulation(population, calculate);
215     }
216     // System.out.println("\n GA Evolved Population 1000 : " +
217     // population.toString() + "\n");
218     System.out.println("\nEGA Fittest Schedule 1000 : "
219     + population.getFittest().toString());
220     stopTime = System.nanoTime();
221     timeStamp = stopTime - startTime;
222     timeStorageList.add(timeStamp);
223     timeStorage = timeStorage + "EGA Time 1000= " + timeStamp + "\n"; 224
        calculate.calculateFitnessWithParemeter(population.getFittest()); 225
226     // Evolve population for 2000 generations
227     population = (Population) initPopulation.clone();
228     startTime = System.nanoTime();
229     for (int i = 0; i < 2000; i++) {
230     population = GA.evolvePopulation(population, calculate);
231     }
232     // System.out.println("\n GA Evolved Population 2000 : " +
233     // population.toString() + "\n");
234     System.out.println("\nEGA Fittest Schedule 2000 : "
235     + population.getFittest().toString());
236     stopTime = System.nanoTime();
237     timeStamp = stopTime - startTime;
238     timeStorageList.add(timeStamp);
239     timeStorage = timeStorage + "EGA Time 2000= " + timeStamp + "\n"; 240
        calculate.calculateFitnessWithParemeter(population.getFittest()); 241
242     // Evolve population for 5000 generations
243     population = (Population) initPopulation.clone();
244     startTime = System.nanoTime();
245     for (int i = 0; i < 5000; i++) {
246     population = GA.evolvePopulation(population, calculate);
247     }
248     // System.out.println("\n GA Evolved Population 5000 : " +
249     // population.toString() + "\n");
250     System.out.println("\nEGA Fittest Schedule 5000 : "
251     + population.getFittest().toString());
252     stopTime = System.nanoTime();
253     timeStamp = stopTime - startTime;
254     timeStorageList.add(timeStamp);
255     timeStorage = timeStorage + "EGA Time 5000= " + timeStamp + "\n"; 256
        calculate.calculateFitnessWithParemeter(population.getFittest());
257
258     // Evolve population for 10000 generations
259     population = (Population) initPopulation.clone();
260     startTime = System.nanoTime();
261     for (int i = 0; i < 10000; i++) {
262     population = GA.evolvePopulation(population, calculate);
263     }
264     // System.out.println("\n GA Evolved Population 10000 : " +
265     // population.toString() + "\n");
266     System.out.println("\nEGA Fittest Schedule 10000 : "
267     + population.getFittest().toString());
268     stopTime = System.nanoTime();
269     timeStamp = stopTime - startTime;
270     timeStorageList.add(timeStamp);
271     timeStorage = timeStorage + "EGA Time 10000= " + timeStamp + "\n";
```

```
272     calculate.calculateFitnessWithParemeter(population.getFittest());
273
274     // Evolve population for 20000 generations
275     population = (Population) initPopulation.clone();
276     startTime = System.nanoTime();
277     for (int i = 0; i < 20000; i++) {
278     population = GA.evolvePopulation(population, calculate);
279     }
280     // System.out.println("\n GA Evolved Population 20000 : " +
281     // population.toString() + "\n");
282     System.out.println("\nEGA Fittest Schedule 20000 : "
283     + population.getFittest().toString());
284     stopTime = System.nanoTime();
285     timeStamp = stopTime - startTime;
286     timeStorageList.add(timeStamp);
287     timeStorage = timeStorage + "EGA Time 20000= " + timeStamp + "\n";
288     calculate.calculateFitnessWithParemeter(population.getFittest());
289
290     // Evolve population for 50000 generations
291     population = (Population) initPopulation.clone();
292     startTime = System.nanoTime();
293     for (int i = 0; i < 50000; i++) {
294     population = GA.evolvePopulation(population, calculate);
295     }
296     // System.out.println("\n GA Evolved Population 50000 : " +
297     // population.toString() + "\n");
298     System.out.println("\nEGA Fittest Schedule 50000 : "
299     + population.getFittest().toString());
300     stopTime = System.nanoTime();
301     timeStamp = stopTime - startTime;
302     timeStorageList.add(timeStamp);
303     timeStorage = timeStorage + "EGA Time 50000= " + timeStamp + "\n";
304     calculate.calculateFitnessWithParemeter(population.getFittest());
305
306     // Evolve population for 100000 generations
307     population = (Population) initPopulation.clone();
308     startTime = System.nanoTime();
309     for (int i = 0; i < 100000; i++) {
310     population = GA.evolvePopulation(population, calculate);
311     }
312     // System.out.println("\n GA Evolved Population 100000 : " +
313     // population.toString() + "\n");
314     System.out.println("\nEGA Fittest Schedule 100000 : "
315     + population.getFittest().toString());
316     stopTime = System.nanoTime();
317     timeStamp = stopTime - startTime;
318     timeStorageList.add(timeStamp);
319     timeStorage = timeStorage + "EGA Time 100000= " + timeStamp + "\n"; 320
        calculate.calculateFitnessWithParemeter(population.getFittest());
321
322
323         System.out.println("\n----------- Elistist Ant Colony Optimization (EACO)
    -----------\n");
324
325     Schedule antColonyFittest;
326     Population antPopulation = populationACO;
327     System.out.println("\n Initial fittest population: "
328     + antPopulation.getFittest().toString());
329     calculate.calculateFitnessWithParemeter(antPopulation.getFittest());
330
331     System.out.println("\n----------- START SIMULATING -----------\n"); 332
333         // Ant colony with 100 iteration
334         startTime = System.nanoTime();
335         antColonyFittest = AntColonyOptimization.antColonyOptimization(
336         antPopulation, 100, calculate).getFittest();
337         System.out.println("\n Elitist Ant Colony Fittest Schedule 100 iteration : "
338         + antColonyFittest.toString());
```

```
339         stopTime = System.nanoTime();
340         timeStamp = stopTime - startTime;
341         timeStorageList.add(timeStamp);
342         timeStorage = timeStorage + "EACO Time 100= " + timeStamp + "\n";
343         calculate.calculateFitnessWithParemeter(antColonyFittest);
344
345         // Ant colony with 200 iteration
346         startTime = System.nanoTime();
347         antColonyFittest = AntColonyOptimization.antColonyOptimization(
348         antPopulation, 200, calculate).getFittest();
349         System.out.println("\n Elitist Ant Colony Fittest Schedule 200 iteration : "
350         + antColonyFittest.toString());
351         stopTime = System.nanoTime();
352         timeStamp = stopTime - startTime;
353         timeStorageList.add(timeStamp);
354         timeStorage = timeStorage + "EACO Time 200= " + timeStamp + "\n";
355         calculate.calculateFitnessWithParemeter(antColonyFittest);
356
357       // Ant colony with 500 iteration
358       startTime = System.nanoTime();
359       antColonyFittest = AntColonyOptimization.antColonyOptimization(
360       antPopulation, 500, calculate).getFittest();
361       System.out.println("\n Elitist Ant Colony Fittest Schedule 500 iteration : "
362       + antColonyFittest.toString());
363       stopTime = System.nanoTime();
364       timeStamp = stopTime - startTime;
365       timeStorageList.add(timeStamp);
366       timeStorage = timeStorage + "EACO Time 500= " + timeStamp + "\n";
367       calculate.calculateFitnessWithParemeter(antColonyFittest);
368
369       // Ant colony with 1000 iteration
370       startTime = System.nanoTime();
371       antColonyFittest = AntColonyOptimization.antColonyOptimization(
372       antPopulation, 1000, calculate).getFittest();
373       System.out.println("\n Elitist Ant Colony Fittest Schedule 1000 iteration : "
374       + antColonyFittest.toString());
375       stopTime = System.nanoTime();
376       timeStamp = stopTime - startTime;
377       timeStorageList.add(timeStamp);
378       timeStorage = timeStorage + "EACO Time 1000= " + timeStamp + "\n"; 379
          calculate.calculateFitnessWithParemeter(antColonyFittest); 380
381       // Ant colony with 2000 iteration
382       startTime = System.nanoTime();
383       antColonyFittest = AntColonyOptimization.antColonyOptimization(
384       antPopulation, 2000, calculate).getFittest();
385       System.out.println("\n Elitist Ant Colony Fittest Schedule 2000 iteration : "
386       + antColonyFittest.toString());
387       stopTime = System.nanoTime();
388       timeStamp = stopTime - startTime;
389       timeStorageList.add(timeStamp);
390       timeStorage = timeStorage + "EACO Time 2000= " + timeStamp + "\n"; 391
          calculate.calculateFitnessWithParemeter(antColonyFittest); 392
393       // Ant colony with 5000 iteration
394       startTime = System.nanoTime();
395       antColonyFittest = AntColonyOptimization.antColonyOptimization(
396       antPopulation, 5000, calculate).getFittest();
397       System.out.println("\n Elitist Ant Colony Fittest Schedule 5000 iteration : "
398       + antColonyFittest.toString());
399       stopTime = System.nanoTime();
400       timeStamp = stopTime - startTime;
401       timeStorageList.add(timeStamp);
402       timeStorage = timeStorage + "EACO Time 5000= " + timeStamp + "\n"; 403
          calculate.calculateFitnessWithParemeter(antColonyFittest); 404
405       // Ant colony with 10000 iteration
406       startTime = System.nanoTime();
```

```
407     antColonyFittest = AntColonyOptimization.antColonyOptimization(
408     antPopulation, 10000, calculate).getFittest();
409     System.out.println("\n Elitist Ant Colony Fittest Schedule 10000 iteration : "
410     + antColonyFittest.toString());
411     stopTime = System.nanoTime();
412     timeStamp = stopTime - startTime;
413     timeStorageList.add(timeStamp);
414     timeStorage = timeStorage + "EACO Time 10000= " + timeStamp + "\n"; 415
        calculate.calculateFitnessWithParemeter(antColonyFittest); 416
417     // Ant colony with 20000 iteration
418     startTime = System.nanoTime();
419     antColonyFittest = AntColonyOptimization.antColonyOptimization(
420     antPopulation, 20000, calculate).getFittest();
421     System.out.println("\n Elitist Ant Colony Fittest Schedule 20000 iteration : "
422     + antColonyFittest.toString());
423     stopTime = System.nanoTime();
424     timeStamp = stopTime - startTime;
425     timeStorageList.add(timeStamp);
426     timeStorage = timeStorage + "EACO Time 20000= " + timeStamp + "\n"; 427
        calculate.calculateFitnessWithParemeter(antColonyFittest); 428
429     // Ant colony with 50000 iteration
430     startTime = System.nanoTime();
431     antColonyFittest = AntColonyOptimization.antColonyOptimization(
432     antPopulation, 50000, calculate).getFittest();
433     System.out.println("\n Elitist Ant Colony Fittest Schedule 50000 iteration : "
434     + antColonyFittest.toString());
435     stopTime = System.nanoTime();
436     timeStamp = stopTime - startTime;
437     timeStorageList.add(timeStamp);
438     timeStorage = timeStorage + "EACO Time 50000= " + timeStamp + "\n"; 439
        calculate.calculateFitnessWithParemeter(antColonyFittest); 440
441     // Ant colony with 100000 iteration
442     startTime = System.nanoTime();
443     antColonyFittest = AntColonyOptimization.antColonyOptimization(
444     antPopulation, 100000, calculate).getFittest();
445     System.out.println("\n Elitist Ant Colony Fittest Schedule 100000 iteration : "
446     + antColonyFittest.toString());
447     stopTime = System.nanoTime();
448     timeStamp = stopTime - startTime;
449     timeStorageList.add(timeStamp);
450     timeStorage = timeStorage + "EACO Time 100000= " + timeStamp + "\n"; 451
        calculate.calculateFitnessWithParemeter(antColonyFittest); 452
453     }
454
455     public static void main(String[] args) throws CloneNotSupportedException {
456     long[] timeCounter = new long[40];
457     int cycle = 1;
458
459         for (int i = 0; i < cycle; i++) {
460         ArrayList<Long> timeStorageList = new ArrayList<Long>();
461         simulation(timeStorageList);
462
463             for (int j = 0; j < timeStorageList.size(); j++) {
464             timeCounter[j] += timeStorageList.get(j);
465             }
466             }
467
468     long avgEGA100 = timeCounter[0] / cycle;
469     long avgEGA200 = timeCounter[1] / cycle;
470     long avgEGA500 = timeCounter[2] / cycle;
471     long avgEGA1000 = timeCounter[3] / cycle;
472     long avgEGA2000 = timeCounter[4] / cycle;
473     long avgEGA5000 = timeCounter[5] / cycle;
474     long avgEGA10000 = timeCounter[6] / cycle;
475     long avgEGA20000 = timeCounter[7] / cycle;
476     long avgEGA50000 = timeCounter[8] / cycle;
```

```
477     long avgEGA100000 = timeCounter[9] / cycle; 478
479     long avgEACO100 = timeCounter[10] / cycle;
480     long avgEACO200 = timeCounter[11] / cycle;
481     long avgEACO500 = timeCounter[12] / cycle;
482     long avgEACO1000 = timeCounter[13] / cycle;
483     long avgEACO2000 = timeCounter[14] / cycle;
484     long avgEACO5000 = timeCounter[15] / cycle;
485     long avgEACO10000 = timeCounter[16] / cycle;
486     long avgEACO20000 = timeCounter[17] / cycle;
487     long avgEACO50000 = timeCounter[18] / cycle;
488     long avgEACO100000 = timeCounter[19] / cycle;
489
490         System.out.println("EGA Time 100 = " + (double) avgEGA100
491         / 1_000_000_000);
492         System.out.println("EGA Time 200 = " + (double) avgEGA200
493         / 1_000_000_000);
494         System.out.println("EGA Time 500 = " + (double) avgEGA500
495         / 1_000_000_000);
496         System.out.println("EGA Time 1000 = " + (double) avgEGA1000
497         / 1_000_000_000);
498         System.out.println("EGA Time 2000 = " + (double) avgEGA2000
499         / 1_000_000_000);
500         System.out.println("EGA Time 5000 = " + (double) avgEGA5000
501         / 1_000_000_000);
502         System.out.println("EGA Time 10000 = " + (double) avgEGA10000
503         / 1_000_000_000);
504         System.out.println("EGA Time 20000 = " + (double) avgEGA20000
505         / 1_000_000_000);
506         System.out.println("EGA Time 50000 = " + (double) avgEGA50000
507         / 1_000_000_000);
508         System.out.println("EGA Time 100000 = " + (double) avgEGA100000
509         / 1_000_000_000);
510
511         System.out.println("EACO Time 100 = " + (double) avgEACO100
512         / 1_000_000_000);
513         System.out.println("EACO Time 200 = " + (double) avgEACO200
514         / 1_000_000_000);
515         System.out.println("EACO Time 500 = " + (double) avgEACO500
516         / 1_000_000_000);
517         System.out.println("EACO Time 1000 = " + (double) avgEACO1000
518         / 1_000_000_000);
519         System.out.println("EACO Time 2000 = " + (double) avgEACO2000
520         / 1_000_000_000);
521         System.out.println("EACO Time 5000 = " + (double) avgEACO5000
522         / 1_000_000_000);
523         System.out.println("EACO Time 10000 = " + (double) avgEACO10000
524         / 1_000_000_000);
525         System.out.println("EACO Time 20000 = " + (double) avgEACO20000
526         / 1_000_000_000);
527         System.out.println("EACO Time 50000 = " + (double) avgEACO50000
528         / 1_000_000_000);
529         System.out.println("EACO Time 100000 = " + (double) avgEACO100000
530         / 1_000_000_000);
531
532     }
533
534}
535
```

Service.java

```
1 import     java.util.concurrent.atomic.AtomicInteger;2
3 public class Service {
4       private static AtomicInteger count = new AtomicInteger(-1);
5       private int serviceId;
6       private String serviceName;
7       private int serviceManufacturingType;
8       private double serviceCostPerTime;
9       private double serviceQuality;
10      private double serviceReliability;
11      private int serviceLocationIndex;
12      private double serviceTravelVelocity;
13      private double serviceTravelCostPerDistance;14
15      public Service(String serviceName,
16                      int serviceManufacturingType, double serviceCostPerTime,
17                      double serviceQuality, double serviceReliability,
18                      int serviceLocationIndex, double serviceTravelVelocity,
19                      double serviceTravelCostPerDistance) {
20              super();
21              this.serviceId = count.incrementAndGet();
22              this.serviceName = serviceName;
23              this.serviceManufacturingType = serviceManufacturingType;
24              this.serviceCostPerTime = serviceCostPerTime;
25              this.serviceQuality = serviceQuality;
26              this.serviceReliability = serviceReliability;
27              this.serviceLocationIndex = serviceLocationIndex;
28              this.serviceTravelVelocity = serviceTravelVelocity;
29      this.serviceTravelCostPerDistance = serviceTravelCostPerDistance;30      }
31
32      public int getServiceId() {
33      return serviceId;34             }
35
36      /*public void setServiceId(int serviceId) {
37      this.serviceId = serviceId;38  }*/
39
40      public String getServiceName() {
41      return serviceName;42       }
43
44      public void setServiceName(String serviceName) {
45      this.serviceName = serviceName;46      }
47
48      public int getServiceManufacturingType() {
49      return serviceManufacturingType;50     }
51
52      public void setServiceManufacturingType(int serviceManufacturingType) {
53      this.serviceManufacturingType = serviceManufacturingType;54             }
55
56      public double getServiceCostPerTime() {
57      return serviceCostPerTime;58            }
59
60      public void setServiceCostPerTime(double serviceCostPerTime) {
61      this.serviceCostPerTime = serviceCostPerTime;62}
```

```
63
64      public double getServiceQuality() {
65      return serviceQuality;66       }
67
68      public void setServiceQuality(double serviceQuality) {
69      this.serviceQuality = serviceQuality;70   }
71
72      public double getServiceReliability() {
73      return serviceReliability;74   }
75
76      public void setServiceReliability(double serviceReliability) {
77      this.serviceReliability = serviceReliability;78          }
79
80      public int getServiceLocationIndex() {
81      return serviceLocationIndex;82             }
83
84      public void setServiceLocationIndex(int serviceLocationIndex) {
85      this.serviceLocationIndex = serviceLocationIndex;86              }
87
88      public double getServiceTravelVelocity() {
89      return serviceTravelVelocity;90            }
91
92      public void setServiceTravelVelocity(double serviceTravelVelocity) {
93      this.serviceTravelVelocity = serviceTravelVelocity;94              }
95
96      public double getServiceTravelCostPerDistance() {
97      return serviceTravelCostPerDistance;98 }
99
100     public void setServiceTravelCostPerDistance(double serviceTravelCostPerDistance) {
101             this.serviceTravelCostPerDistance = serviceTravelCostPerDistance;
102     }
103
104     public String toString() {
105             return "Service [serviceId=" + serviceId
106                     + ", serviceName=" + serviceName
107                     + ", serviceManufacturingType=" + serviceManufacturingType
108                     + ", serviceLocationIndex=" + serviceLocationIndex
109                     + ", serviceCostPerTime=" + String.format("%.2f", serviceCostPerTime)
110                     + ", serviceQuality=" + String.format("%.2f", serviceQuality)
111                     + ", serviceReliability=" + String.format("%.2f", serviceReliability)
112                     + ", serviceTravelVelocity=" + String.format("%.2f", serviceTravelVelocity)
113                     + ", serviceTravelCostPerDistance=" + String.format("%.2f",
    serviceTravelCostPerDistance) + "]";
114     }
115
116 }
117
```

Subtask.java

```
1 import    java.util.concurrent.atomic.AtomicInteger;2
3 public class SubTask {
4       private static AtomicInteger count = new AtomicInteger(-1);
5       private int subTaskId;
6       private String subTaskName;
7       private int subTaskManufacturingType;
8       private int subTaskLocationId;
9       private double subTaskExecutionTime;10
11      public SubTask(String subTaskName,
12              int subTaskManufacturingType, int subTaskLocationId,
13              double subTaskExecutionTime) {
14          this.subTaskId = count.incrementAndGet();
15          this.subTaskName = subTaskName;
16          this.subTaskManufacturingType = subTaskManufacturingType;
17          this.subTaskLocationId = subTaskLocationId;
18          this.subTaskExecutionTime = subTaskExecutionTime;
19      }
20
21      public int getSubTaskId() {
22          return subTaskId;
23      }
24
25      public String getSubTaskName() {
26          return subTaskName;
27      }
28
29      public void setSubTaskName(String subTaskName) {
30          this.subTaskName = subTaskName;
31      }
32
33      public int getSubTaskManufacturingType() {
34          return subTaskManufacturingType;
35      }

37      public void setSubTaskManufacturingType(int subTaskManufacturingType) {
38          this.subTaskManufacturingType = subTaskManufacturingType;
39      }
40
41      public int getSubTaskLocationId() {
42          return subTaskLocationId;
43      }
44
45      public void setSubTaskLocationId(int subTaskLocationId) {
46          this.subTaskLocationId = subTaskLocationId;
47      }
48
49      public double getSubTaskExecutionTime() {
50          return subTaskExecutionTime;
51      }
52
53      public void setSubTaskExecutionTime(double subTaskExecutionTime) {
54          this.subTaskExecutionTime = subTaskExecutionTime;
55      }
56
57      @Override
58      public String toString() {
59          return "SubTask [subTaskId=" + subTaskId + ", subTaskName="
60                  + subTaskName + ", subTaskManufacturingType="
61                  + subTaskManufacturingType + ", subTaskLocationId="
62                  + subTaskLocationId + ", subTaskExecutionTime="
```

```
63                      + String.format("%.2f", subTaskExecutionTime) + "]";64   }
65
66 }
67
```

Task.java

```
1 import java.util.ArrayList;3

4 public class Task {
5       private static AtomicInteger count = new AtomicInteger(-1);
6       private int taskId;
7       private String taskName;
8       private int taskLocationId;
9       private ArrayList<SubTask> subTaskArray = new ArrayList<SubTask>();10
11      public Task(String taskName, int taskLocationId) {
12              this.taskId = count.incrementAndGet();
13              this.taskName = taskName;
14      this.taskLocationId = taskLocationId;15     }
16
17      public Task(String taskName, int taskLocationId, ArrayList<SubTask> subTaskArray) {
18              this.taskId = count.incrementAndGet();
19              this.taskName = taskName;
20              this.taskLocationId = taskLocationId;
21      this.subTaskArray = subTaskArray;22         }
23
24      public void addSubTask(SubTask subTask){
25      subTaskArray.add(subTask);26 }
27
28      public void removeSubTask(SubTask subTask){
29      subTaskArray.remove(subTask);30             }
31
32      public void clearSubTask(){
33      subTaskArray.clear();34             }
35
36      public int getTaskId() {
37      return taskId;38      }
39
40      public String getTaskName() {
41      return taskName;42                  }
43
44      public void setTaskName(String taskName) {
45      this.taskName = taskName;46  }
47
48      public int getTaskLocationId() {
49      return taskLocationId;50          }
51
52      public void setTaskLocationId(int taskLocationId) {
53      this.taskLocationId = taskLocationId;54     }
55
56      public ArrayList<SubTask> getSubTaskArray() {
57      return subTaskArray;58              }
59
60      public void setSubTaskArray(ArrayList<SubTask> subTaskArray) {
61      this.subTaskArray = subTaskArray;62         }
63
```

```
64      @Override
65      public String toString() {
66              return "Task [taskId=" + taskId + ", taskName=" + taskName
67                              + ", taskLocationId=" + taskLocationId
68                              + "]";
69      }
70
71 }
72
```