# Appendix A

**A1. Procedure of the proposed KEDAS**

|  |
| --- |
| **Algorithm** The proposed KEDAS for the considered AGVDP |
| **Input**: Instance data for the AGVDP; initial parameters of KEDAS (*N*: the population size, : the maximum number of iterations,: the minimum inheritance ratio, :the maximum inheritance ratio, : the initial learning rate, *M*: the offspring population size).  **Output**: Pareto front solutions. |
| **Begin**:   1. Generate initial population ***P*** via 12 heuristic dispatching rules and *N*-12 random permutations, and initialize the probability model ***A***, a matrix of AGV transition probability; 2. Sort ***P*** based on the Pareto dominance relationship, then initialize non-dominated solutions and the *HV* metric; 3. Set ; 4. **While**  **do** 5. Update the inheritance ratio according to the equation (20); 6. Select *M* individuals randomly from current population ***P*** into the sampled set; 7. **For** each individual **in** the sampled set: 8. Generate an offspring individual of parent individual by sampling current matrix ***A***; 9. Update the offspring population ***Q***. 10. **End** **for** 11. Evaluate the offspring population ***Q*** based on the delivery satisfaction constraint; 12. Form a united population , then update the non-dominated solutions based on the Pareto dominance; 13. Select a set of candidate solutions from current non-dominated solutions based on the delivery satisfaction; 14. Implement the VNS for the selected non-dominated solutions according to the procedure of VNS in Section4.2.5; 15. Update population ***U*** by uniting the result of VNS, then sort ***U*** based on the concept of Pareto dominance and crowding distance; 16. Calculate new *HV* of current pareto front solution set; 17. Select the best *N* individuals from ***U*** to form new population ***P***; 18. Update the learning rate  according to the equation (22); 19. Update the probability matrix ***A*** according to the equation (21); 20. Set ; 21. **End** **while**   **End**. |

**A2. The discretization approaches on the update of particles’ velocities and positions of MGPSO**

The velocity update approach of MGPSO for our problem is formally transformed as follows:









Where  is the velocity of particle *i* at iteration *t*+1, while , and  are the swap operator sequences derived from local guides, global guides and archive guides, respectively. Moreover, *w* is the inertia weight within (0,1), is the position of particle *i* at iteration *t*, is the personal best position of particle *i* at iteration *t*, is the neighborhood best position of particle *i* at iteration *t*, and  is selected from the archive for particle *i* at iteration *t*. These , ,  and  are the number sequences with identical dimension, and they are all the material orders’ delivery sequences in our problem.

The formula  of equation (A.1) indicates all the swap operators generated by comparing the differences between the two sequences ,  from front to back, where  is the target sequence, that is, if there exists a element difference between these two sequences at same index, it would notice the corresponding element of the target sequence and record its different indexes both in these two sequences to generate a swap operator of the sequences . For instance, there are three differences between these two sequences (1,2,3,4,5) and (2,3,1,4,5), if the index of sequences starts at zero, it would generate three swap operators and they can be indicated as SO(1,0), SO(2,1), SO(0,2), respectively. For a sequence, the swap operator SO(*i*,*j*) can swap its element in index *i* with its element in index *j*. Take the sequence (1,3,2,5,4) as an example, use an swap operator SO(0,2) to update it, and then its new sequence is (2,3,1,5,4).

While this formula  of the equation (A.1) indicates that 100(1-*w*) % of all the swap operators  are selected randomly to form the swap operator sequence , and in the same way the swap operator sequences and  can be constructed, where a swap operator sequence is the set of swap operators with sequential order. As equation (A.4) shows, the updated velocity  of particle *i* at iteration *t*+1 is actually a set of swap operator sequences, and it can be used to change the position  of particle *i* at iteration *t* into the updated position  of particle *i* at iteration *t*+1, which can be expressed as equation (A.5).

 Specifically, the position update operation includes the following three steps.

**Step 1:** the update operation by the swap operator sequences of local guides.

**For** each swap operator **in** :

If *c*1/(*c*1+*c*2+*c*3)≥*rand* and the position of corresponding element unchanged:

Make the element swap for  according to the swap operator.

**End for**.

Update the  as .

**Step 2:** the update operation by the swap operator sequences of global guides.

**For** each swap operator **in** :

If *c*2/(*c*1+*c*2+*c*3)≥*rand* and the position of corresponding element unchanged:

Make the element swap for  according to the swap operator.

**End for**.

Update the  as .

**Step 3:** the update operation by the swap operator sequences of archive guides.

**For** each swap operator **in** :

If *c*3/(*c*1+*c*2+*c*3)≥*rand* and the position of corresponding element unchanged:

Make the element swap for  according to the swap operator.

**End for**.

Update the  as .

Where *c*1, *c*2 and *c*3 are the acceleration coefficients, *rand* isthe random number within (0,1), and are the intermediate variables.

**A3**. **The computational complexity of KEDAS**

KEDAS includes three knowledge-guided strategies, namely dispatching rule-guided population initialization, pareto evolutionary performance-guided probability model update for offspring generation, and delivery satisfaction-guided variable neighbourhood search (VNS). KEDAS is used to solve an optimization problem with *m* objectives and *n*-dimension decision variables, and its population size is equal to *PS*. In each evolutionary iteration of this algorithm, the computational complexity of these three strategies is ,  and  respectively, where *l* is the number of the selected individuals(solutions) for the VNS. It should be noted that the hypervolume (HV) values of the KEDAS can be computed via calling the external high-performance HV function provided by some independent computing tools such as the Geatpy, a high-performance algorithm framework for genetic and evolutionary algorithms, so the computing time of HV values is not incorporated into the runtime of KEDAS. Therefore, the overall computational complexity of KEDAS is .

Figure 8(d) in the manuscript provides a comparison of the average runtime (millisecond) per 100 iterations of the compared algorithms. MEDA has the shortest execution time for all selected instances. KEDAS has higher computational complexity than MEDA mainly due to the complexity of the VNS adopted by KEDAS, but its average runtime per 100 iterations is still relatively shorter than those of the four other algorithms on all selected instances.

# Appendix B

To further visualize the impact of the order delivery satisfaction on the order delivery punctuality, we select the best solution of the instance N10S10 as an example and present the corresponding AGV dispatching scheme for two AGVs in Figure B(a)-(e) using a delivering orders Gantt chart. More specifically, the estimated AGV unload time for every order at its corresponding workstation is depicted after each order’s transportation stage, together with the specified delivery time window. The delivery satisfaction for every order is also labelled on the Gantt charts.

**Figure B.** AGVs dispatching schemes of different methods for instance N10S10.

Figure B Alt Text: It visualizes the solution results of different methods for instance N10S10 using a set of delivering orders Gantt charts.

|  |  |
| --- | --- |
| (a) FCFS: [0,2,6,9,0,5,4,7,3,8,1,10] | (b) LMQ: [0,7,2,4,8,0,5,6,9,1,3,10] |
| (c) SDTDW: [0,4,3,5,6,0,2,7,9,8,10] | (d) KEDA: [0,6,9,4,3,8,1,10,0,2,7,5] |
| (e) KEDAS: [0,9,6,5,4,0,7,2,3,8,1,10] | |
| **Figure B.** AGVs dispatching schemes of different methods for instance N10S10. | |

From the Gantt charts in Figure B(a)-(e), we can observe that the delivery satisfaction value of the order is zero if an AGV reaches the demand workstation before the specified delivery time window, resulting in a penalty cost for earliness. The delivery satisfaction of the order improves when the AGV arrives at the order’s delivery destination closer to the scheduled delivery time. Comparing the AGVs dispatching schemes depicted in Figure B(a)-(e), we observe that KEDAS provides a more competitive material replenishment plan, and a relative better mean delivery satisfaction, guarantying the punctual delivery for most material orders of N10S10. The illustration further highlights that the AGVDP model with the delivery satisfaction constraint solved by KEDAS not only can give the better results for the optimization objectives, but also improves the satisfaction of material orders delivery.

# Appendix C

Supplemental result tables.

**Table C1.** Results of each sub-objective for instances with 40 orders.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | *F*1 | |  | *F*2 | |
| Instance | Method | Avg | Min |  | Avg | Min |
| N40S0 | FCFS | 11361 | 11361 |  | 3472 | 3472 |
| NSGAⅡ | 8062.77 | 7086 |  | 2256.87 | 1952 |
| NSGAⅢ | 8102.31 | 7669 |  | 2578.0 | 2173 |
| RVEA | 8443.31 | 7157 |  | 2309.23 | 1923 |
| MGPSO | 9125.34 | 8533 |  | 2797.33 | 2585 |
| MEDA | 8332.51 | 7621 |  | 2705.57 | 2343 |
| KEDAS | **7472.02** | **5804** |  | **2007.5** | **1895** |
| N40S20 | FCFS | 8859 | 8859 |  | 3424 | 3424 |
| NSGAⅡ | **6428.70** | 6030 |  | 2676.66 | **2084** |
| NSGAⅢ | 8004.04 | 6616 |  | 2888.5 | 2257 |
| RVEA | 8051.10 | 6547 |  | 2768.16 | 2196 |
| MGPSO | 8323.42 | 7025 |  | 2992.8 | 2617 |
| MEDA | 7908.64 | 6978 |  | 3234.18 | 2515 |
| KEDAS | 6667.99 | **5814** |  | **2279.0** | 2098 |
| N40S40 | FCFS | 8709 | 8709 |  | 3880 | 3880 |
| NSGAⅡ | 7144.67 | 6690 |  | 3241.42 | 2713 |
| NSGAⅢ | 7372.53 | 6855 |  | 3575.76 | 2887 |
| RVEA | 7446.41 | 6987 |  | **2944.21** | **2440** |
| MGPSO | 8509.09 | 7291 |  | 3250.0 | 3198 |
| MEDA | 7776.68 | 7225 |  | 3360.0 | 3010 |
| KEDAS | **7020.44** | **6450** |  | 3169.14 | 2591 |

**Table C2.** Results of each sub-objective for instances with 50 orders.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | *F*1 | |  | *F*2 | |
| Instance | Method | Avg | Min |  | Avg | Min |
| N50S0 | FCFS | 12789 | 12789 |  | 3756 | 3756 |
| NSGAⅡ | 10043.24 | 9374 |  | **2719.37** | 2537 |
| NSGAⅢ | 10224.77 | 9625 |  | 2837.66 | 2459 |
| RVEA | 10412.38 | 9390 |  | 2880.5 | 2583 |
| MGPSO | 10841.53 | 10644 |  | 3169.0 | 3090 |
| MEDA | 10799.64 | 9885 |  | 3176.5 | 2615 |
| KEDAS | **9065.73** | **8267** |  | 2765.5 | **1963** |
| N50S25 | FCFS | 12907 | 12907 |  | 4071 | 4071 |
| NSGAⅡ | 9354.06 | 8294 |  | 3703.57 | 2872 |
| NSGAⅢ | 9763.31 | 8357 |  | 3494.14 | 3116 |
| RVEA | 10082.63 | 8326 |  | 3557.12 | 2997 |
| MGPSO | 10746.98 | 9468 |  | 3894.4 | 3374 |
| MEDA | 9631.58 | 8986 |  | 3768.66 | 3346 |
| KEDAS | **8908.30** | **7670** |  | **3186.0** | **2769** |
| N50S50 | FCFS | 13667 | 13667 |  | 3889 | 3889 |
| NSGAⅡ | 9633.69 | 8845 |  | 3244.69 | 2789 |
| NSGAⅢ | 10771.40 | 9216 |  | 3206.0 | 2757 |
| RVEA | 10396.41 | 9087 |  | 3340.3 | 2990 |
| MGPSO | 11551.56 | 10436 |  | 3844.71 | 3424 |
| MEDA | 10481.68 | 9602 |  | 3863.71 | 3452 |
| KEDAS | **9103.56** | **8761** |  | **2906.0** | **2714** |

**Table C3.** Multiple comparison of HVs’ means with Tukey HSD (FWER=0.05).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Group1 | Group2 | Mean difference | P-adj | Lower | Upper | Reject |
| KEDAS | MEDA | -0.1218 | 0.001 | -0.1549 | -0.0887 | True |
| KEDAS | MGPSO | -0.1212 | 0.001 | -0.1543 | -0.0881 | True |
| KEDAS | NSGAⅡ | -0.0023 | 0.9 | -0.0354 | 0.0307 | False |
| KEDAS | NSGAⅢ | -0.0187 | 0.5718 | -0.0518 | 0.0144 | False |
| KEDAS | RVEA | -0.0366 | 0.0202 | -0.0697 | -0.0036 | True |
| MEDA | MGPSO | 0.0006 | 0.9 | -0.0325 | 0.0337 | False |
| MEDA | NSGAⅡ | 0.1195 | 0.001 | 0.0864 | 0.1525 | True |
| MEDA | NSGAⅢ | 0.1031 | 0.001 | 0.07 | 0.1362 | True |
| MEDA | RVEA | 0.0852 | 0.001 | 0.0521 | 0.1182 | True |
| MGPSO | NSGAⅡ | 0.1189 | 0.001 | 0.0858 | 0.1519 | True |
| MGPSO | NSGAⅢ | 0.1025 | 0.001 | 0.0694 | 0.1356 | True |
| MGPSO | RVEA | 0.0845 | 0.001 | 0.0515 | 0.1176 | True |
| NSGAⅡ | NSGAⅢ | -0.0164 | 0.6882 | -0.0495 | 0.0167 | False |
| NSGAⅡ | RVEA | -0.0343 | 0.0371 | -0.0674 | -0.0012 | True |
| NSGAⅢ | RVEA | -0.0179 | 0.6109 | -0.051 | 0.0151 | False |

**Table C4.** Multiple comparison of Spacings’ means with Tukey HSD (FWER=0.05).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Group1 | Group2 | Mean difference | P-adj | Lower | Upper | Reject |
| KEDAS | MEDA | 0.5186 | 0.001 | 0.4074 | 0.6298 | True |
| KEDAS | MGPSO | 0.5215 | 0.001 | 0.4103 | 0.6327 | True |
| KEDAS | NSGAⅡ | -0.1276 | 0.0142 | -0.2388 | -0.0164 | True |
| KEDAS | NSGAⅢ | -0.1228 | 0.0208 | -0.234 | -0.0116 | True |
| KEDAS | RVEA | -0.1196 | 0.0268 | -0.2308 | -0.0084 | True |
| MEDA | MGPSO | 0.0029 | 0.9 | -0.1083 | 0.1141 | False |
| MEDA | NSGAⅡ | -0.6462 | 0.001 | -0.7574 | -0.535 | True |
| MEDA | NSGAⅢ | -0.6414 | 0.001 | -0.7526 | -0.5303 | True |
| MEDA | RVEA | -0.6382 | 0.001 | -0.7494 | -0.527 | True |
| MGPSO | NSGAⅡ | -0.6491 | 0.001 | -0.7603 | -0.5379 | True |
| MGPSO | NSGAⅢ | -0.6444 | 0.001 | -0.7555 | -0.5332 | True |
| MGPSO | RVEA | -0.6411 | 0.001 | -0.7523 | -0.53 | True |
| NSGAⅡ | NSGAⅢ | 0.0047 | 0.9 | -0.1065 | 0.1159 | False |
| NSGAⅡ | RVEA | 0.008 | 0.9 | -0.1032 | 0.1191 | False |
| NSGAⅢ | RVEA | 0.0032 | 0.9 | -0.108 | 0.1144 | False |