Multi-agent Optimization Design for Multi-resource Job Shop Scheduling Problems

Fan Xue and Wei Fan

College of Computer Science and Technology, Civil Aviation University of China, Tianjin 300300, P.R. China
wfan@cauc.edu.cn

Abstract. As a practical generalization of the job shop scheduling problem, multi-resource job shop scheduling problem (MRJSSP) is discussed in this paper. In this problem, operations may be processed by a type of resources and jobs have individual deadlines. How to design and optimize this problem with DSAFO, a novel multi-agent algorithm, is introduced in detail by a case study, including problem analysis, agent role specification, and parameter selection. Experimental results show the effectiveness and efficiency of designing and optimizing MRJSSPs with multi-agent.

1 Introduction

A practical generalization of the job shop scheduling problem (JSSP), which we call the multi-resource job shop scheduling problem (MRJSSP), is concerned in this paper. Informally, the problem can be stated as follows. There are a set of jobs and a set of resources. Each job consists of a lattice of operations that must be processed in a given order, and has, individually, a job ready time and a job deadline. Each operation is given an integeral processing time, and a longer resource usage time (plan time) for extra traffic (spatial distribution), preparation, and reset actions. Each operation needs one resource to process, and the processing is uninterruptible. Each resource can process only one operation simultaneously. The objective of MRJSSP is to find the best scheduling solution with minimal resource consumption, i.e. maximal resource utility.

JSSP has been studied by both academic and industrial society for decades [1], however in many practical situations, (i) an operation can be processed by any one resource (or machine) from a group; (ii) jobs have individual deadlines; (iii) the requirement of no tardiness for any jobs is more important than makespan; and (iv) to reduce consumption as much as possible in order to maximize the machine utilities. Those are, fitly, the cases of MRJSSPs.

M. Perregaard (1995) proposed multi-processor job shop scheduling problem (MPJSSP), which concerned multiple processing capacity as well, and A. Cesta, A. Oddi, and S. F. Smith (2000) developed an iterative improvement search approach for it. W. P. M. Nuijten and E. H. L. Aarts (1996) [4] presented another problem: multiple capacitated job shop scheduling problem (MCJSSP), which

D.-S. Huang, L. Heutte, and M. Loog (Eds.): ICIC 2007, LNAI 4682, pp. 1193–1204, 2007.

extended MPJSSP by allowing each operation has a size. Nevertheless, both MPJSSP and MCJSSP ignored the spatial conditions and supporting handling in real-world engineering processes. Furthermore most of scheduling works are, practically, pre-scheduled by domain experts, so what we need to optimize is, usually, to maximize the resource utility to meet a timetable, not the makespan. All of these are well presented in MRJSSP, which is detailed in Section 2.

The remainder of the paper is structured as follows: Section 2 represents a definition of the multi-resource job shop scheduling problem. Section 3 reviews the DSAFO algorithm briefly. Section 4 demonstrates the design procedure via a case study. Experimental results appear in Section 5 and a brief conclusion is given in Section 6.

2 The Multi-resource Job Shop Scheduling Problem

Definition 1 (MRJSSP). *An instance of multi-resource job shop scheduling problem is a tuple* $\langle \mathcal{F}, \mathcal{O}, \mathcal{R}, \mathcal{T}, \preceq, D, F, rt, st, ut, et, tt, \Omega, \gamma, s \rangle$ *where*

$\mathcal{J} = \{j_1, j_2, \ldots, j_n\}$ - *a set of n jobs;*

$\mathcal{O} = \{o_1, o_2, \ldots, o_p\} = \mathcal{O}_1 \cup \mathcal{O}_2 \cup \ldots \cup \mathcal{O}_m$, *where* $\underset{m_i \neq m_j}{\forall} \mathcal{O}_{m_i} \cap \mathcal{O}_{m_j} = \emptyset$ - *a set of p operations in m types (partitions);*

$\mathcal{R} = \{r_1, r_2, \ldots, r_q\} = \mathcal{R}_1 \cup \mathcal{R}_2 \cup \ldots \cup \mathcal{R}_t$, *where* $\underset{t_i \neq t_j}{\forall} \mathcal{R}_{t_i} \cap \mathcal{R}_{t_j} = \emptyset$ - *a set of q resources in t types (partitions);*

$C = \mathcal{O}_1 \times \mathcal{R}_{t_1} \cup \mathcal{O}_2 \times \mathcal{R}_{t_2} \cup \ldots \cup \mathcal{O}_m \times \mathcal{R}_{t_m}$ - *processing capabilities;*

$\preceq : \mathcal{O} \mapsto \mathcal{O}$ - *precedence or equality, decomposing* $\mathcal{O}$ *into lattices (specially, chains) of jobs;*

$D : \mathcal{J} \mapsto \mathbb{Z}_0^+$ - *deadline of a job;*

$J : \mathcal{O} \mapsto \mathcal{J}$ - *job belonging to;*

$rt : \mathcal{O} \mapsto \mathbb{Z}_0^+$ - *operation ready time;*

$st : \mathcal{O} \mapsto \mathbb{Z}^+$ - *non-zero operation service time;*

$ut : \mathcal{O} \mapsto \mathbb{Z}_0^+$ - *operation setup time;*

$et : \mathcal{O} \mapsto \mathbb{Z}_0^+$ - *operation reset time;*

$tt : \mathcal{R} \times \mathcal{O} \mapsto \mathbb{Z}_0^+$ - *resource traffic time for an operation;*

$\Omega : \mathcal{R} \times \mathbb{Z}_0^+ \mapsto \mathcal{O} \cup \{\emptyset\}$ - *which operation is in process at a certain time, returns* $\emptyset$ *when non-single operations assigned.*

The objective is to find two functions: s and γ*, where*

$\gamma : \mathcal{O} \mapsto \mathcal{R}$ - *assign resources;*

$s : \mathcal{O} \mapsto \mathbb{Z}_0^+$ - *assign service start time,*

$$\begin{aligned}
\textbf{s.t.} \ \underset{o \in \mathcal{O}}{\forall} [\ & \langle o, \gamma(o) \rangle \in C \\
& \wedge rt(o) \leq s(o) \\
& \wedge s(o) + st(o) \leq D(F(o)) \\
& \wedge \underset{o \prec o'}{\forall} s(o) + st(o) \leq rt(o') \\
& \wedge \underset{s(o)-ut(o)-tt(\gamma(o),o) \leq \tau < s(o)+st(o)+et(o)}{\forall} \Omega(\gamma(o), \tau) = o] \\
\wedge |ran(\gamma)| &= \min_{\gamma_i \in \Gamma} |ran(\gamma_i)|,
\end{aligned}$$

where $o \prec o'$ *is abbreviated from* $o \preceq o' \wedge o \neq o'$, Γ *is the set of all valid schedules,* $|ran(\gamma)|$ *is the range size of function* γ.

MRJSSP is a special case of distributed constraint satisfaction problem (Dis-CSP), according to the definition by [6]. Remark that the decision variant of the MRJSSP is NP-complete, as (i) it is in NP, because for a given schedule s all constraints can be checked in polynomial time, and (ii) the decision variant of the JSSP is an NP-complete special case of the decision variant of MRJSSP [5].

3 DSAFO: Overview and Strategies

DSAFO, i.e. Dynamic Scheduling Agents with Federation Organization, which had been formalized in [7], is a novel multi-agent approach to Dis-CSP, especially for MRJSSPs. DSAFO employs blackboard mechanism, federation organization, meta-level guided resource borrowing and domain knowledge guided plan backtrack. This algorithm is based on *run-and-schedule*, a dynamic distributed scheduling environment, which uninterruptedly observes real-time job data from enterprise systems. As shown in Fig. 1, DSAFO runs as follow:

1. to read real-time job data from *run-and-schedule* environment;
2. to decompose jobs into operations;
3. to divide the solution space dynamically into rational partitions with multi-agents;
4. to conquer each partition with local heuristics;
5. to optimize the solution simultaneously via coordination among partitions from global view;
6. to dispatch the solution to real world environment simultaneously.

Internally, DSAFO algorithm can be viewed as a multi-agent approach with strategies of local heuristics and global coordination.

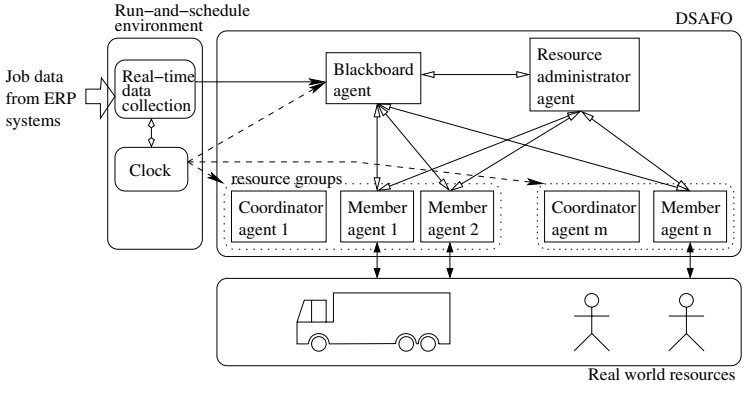


Fig. 1. An overview of DSAFO

There are two parts of local heuristics in DSAFO, both are EDD (Earliest Due Date first). One is to select the first operation to do according to weighted *latest finish time*, and the other is to assign an operation to a resource from some committed resources according to the earliest *committed service start time*. However, when the problem scale increases, EDD falls, generally, into local minimum. So DSAFO employs local heuristics (EDD) in dynamic cooperative agents to jump out of local minimum.

Global coordination is a mechanism that exchanges resource textures of agents and realizes a complete resource borrow procedure for agents. And there are two kinds of relationships between two *Member* agents with the same type of resources: *buddy* and *competitor*. If two *Member* agents share same type of resources and same type of operations, they are competitors; if they share same type of resources but different type of operations, they are buddies. A group of buddies always help each other, by lending its own resource friendly.

4 Design with DSAFO: A Case Study

Airport ground service is the service process from flight landing to takeoff, including gate assignment, baggage handling, catering, fueling, cleaning, etc. Airport ground service scheduling (AGSS) is to schedule many kinds of dynamic ground resources (baggage trucks, fuel trucks, etc.), to fulfill all constrained service operations of flights timely to meet their arrival and departure deadlines [8]. The typical operations for a job, transfer flight service, is shown as Fig. 2.

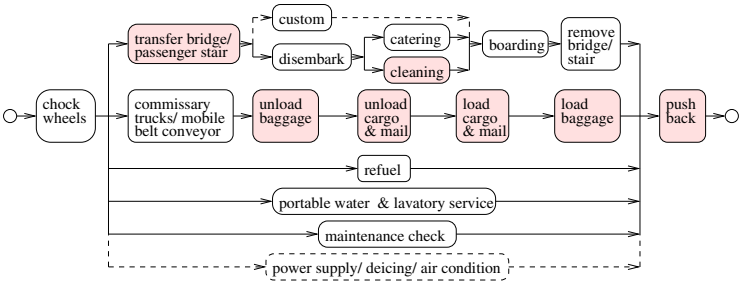


Fig. 2. Typical operations for a transfer flight

The AGSS problem well meets the qualifications of the MRJSSP definition. For each type of operations shown in Fig. 2, one (or a group) of certain type(s) of materials, engineers, aviation ground support facilities and equipments, and transportation equipments are required. For instance, process "*cleaning*" requires cleaners (and cleaners' bus), "*catering*" requires catering truck, etc. However, AGSS problem has one more constraint: resources, i.e. engineers and equipments, have limited job time, e.g. 8 hours for an engineer per day.

Then we demonstratively design AGSS problem optimization with DSAFO in three phases: entity and extra constraint analysis, algorithm specification, and parameter decision. Similar design procedures could be applied to other MRJSSPs.

In the analysis phase, in AGSS problem the pre-determined flight schedule and real-time operation systems (such as Flight Operation Control system, Airport Operation Data Base) provide efficient input data (ready time and due date) for jobs (flights). The jobs come, commonly, one by one, and peak not too much in busy hours. In airports, service support vehicles and engineers must travel along specific roads, load materials, and unload at job-specific points. These geographic distance and the maximal airport ground travel speed (5km/h) could provide us efficient traffic time estimation on service support actions. Some resources are always bounded together, such as a cleaners' group, they can be considered as one resource.

As to the job time limitation of resources, we should modify the resource allocation mechanism in DSAFO. Another constraint is that in minor cases several operations around a physical airplane may conflict with each other, we should conclude the cases and properly relax these operations in operation dispatching process.

In the specification phase, agent roles in DSAFO are assigned with capabilities and meanings. We design the *Blackboard* role to be in charge of decision-making of operation dispatch, the *ResourceAdmin* role to be in charge of decision-making of resource allocation, and *Member* roles to be in charge of making operation-resource match-up dynamically. The *Coordinator* role is, invariably, assigned to facilitate cooperations among *Member* agents.

The real-time job data are gathered and decomposed into operations by the *Blackboard* role, with considering the possible spatial collision among operations. the job time limitation of resources is monitored by the *ResourceAdmin* role, for it controls the allocation of resource. Proper processing plans for resources are made by the *Member* role, according to airport geographic information.

In the last phase, the parameters are designed, experimented, and analyzed one by one. In DSAFO, there are some main parameters: (*Member*) AgentNumber Blockfactor, Delayfactor, and Syncycle. We choose baggage tractor (BT) consumption and related drivers' 4-hour job optimization to test the parameters, and have spontaneous 250 runs to get the distributions of the solutions in each test.

Parameter *Member* agent number (accompanied with Reqcycle) stands for how many *Member* agents in DSAFO are maintained for a certain type of resources. And in all the parameters, this number should be notably influential. When Blockfactor=1/12, Delayfactor=1/6, and Reqcycle is from 1/6 to 2 respectively, Fig. 3 and Fig. 4 show the solution distributions with different BT *Member* agents in 3D map and contour map, as well as marginal distributions. When the number increases from 1 to 4, the solutions are going better, because the coordination strategy makes more insufficient effect when the number increases from only one (no coordination). On the contrary, when the number increases from 4

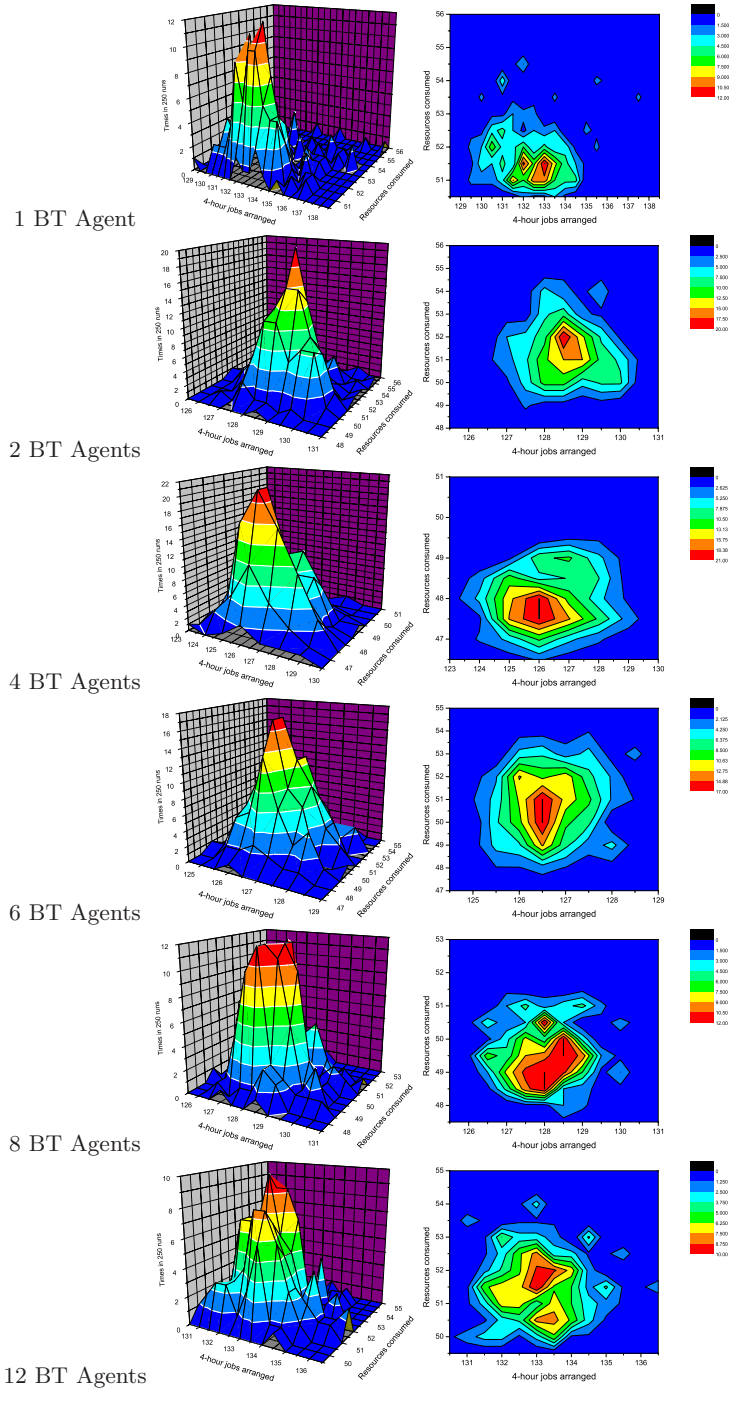


Fig. 3. BT solution distributions with respect to AgentNumber

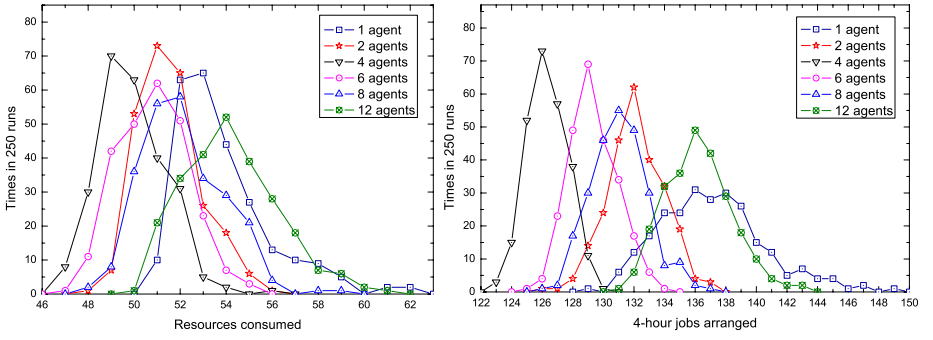
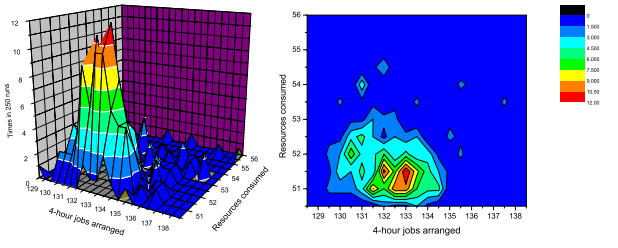
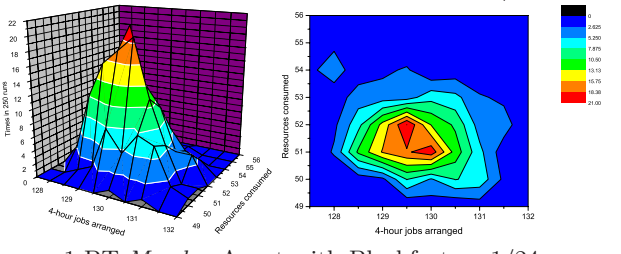


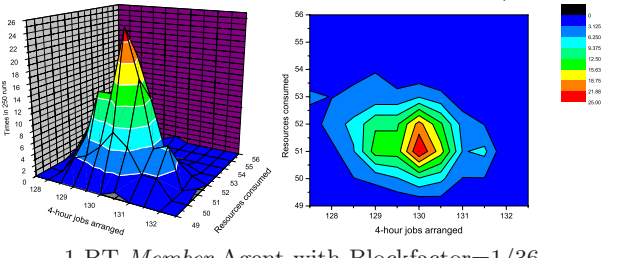
Fig. 4. BT solution marginal distributions with respect to AgentNumber



1 BT *Member* Agent with Blockfactor=1/12



1 BT *Member* Agent with Blockfactor=1/24



1 BT *Member* Agent with Blockfactor=1/36

Fig. 5. BT solution distributions with respect to Blockfactor

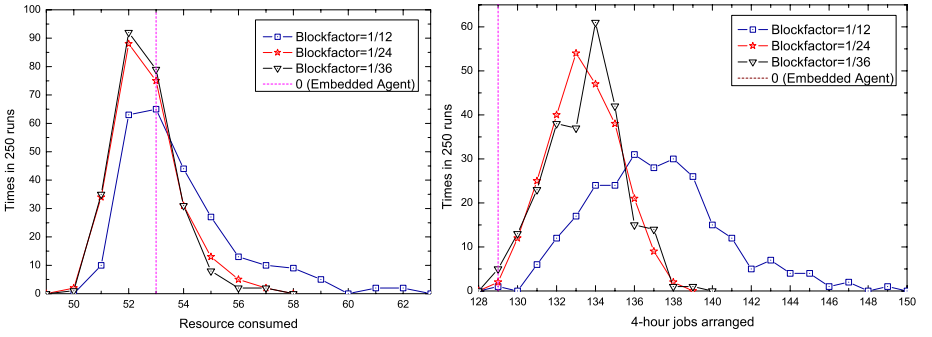
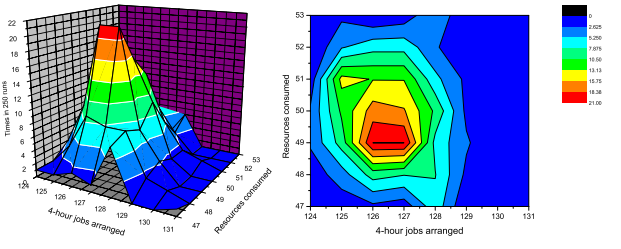
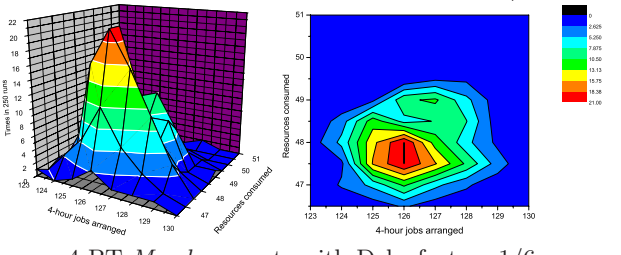


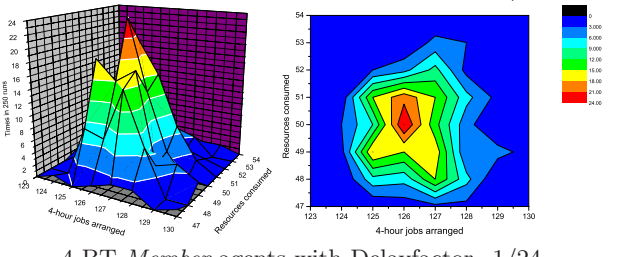
Fig. 6. BT solution marginal distributions with respect to Blockfactor



4 BT *Member* agents with Delayfactor=1/3



4 BT *Member* agents with Delayfactor=1/6



4 BT *Member* agents with Delayfactor=1/24

Fig. 7. BT solution distributions with respect to Delayfactor

to 12, the solutions are going worse, because too many agents divide the global solution into too many fragments, so that local heuristics in these fragments lack of enough global viewpoints, and coordination cannot optimize these fragments very well.

Blockfactor is another influential parameter. We set AgentNumber=1, Delayfactor=1/6, Syncycle=5 and Reqcycle = 1/6, and then we get solution distributions with respect to Blockfactor in 3D map and contour map, and marginal distributions, as shown in Fig. 5 and Fig. 6. From the figures we can conclude that when Blockfactor decreases, resource consumption stays the same approximately and 4-hour jobs arranged decrease a bit; and when the Blockfactor goes smaller, the solutions distributes turn more centralized, and the algorithm acts more similar as a stable algorithm. Specially, we embedded one single *Member* agent in *Blackboard* agent, i.e. without any unstable factors from communication, to eliminate the interference from network message communication. The new embedded algorithm degrades to a deterministic algorithm as shown in Fig. 6 (approximately the same as EDD* in Section 5).

Delayfactor stands for how much time *Member* agent should delay extra after a successful operation commitment, and it is not a very influential parameter. We set AgentNumber=4, Blockfactor=1/12, Syncycle=5 and Reqcycle = 1/2. Then we get distributions in 3D map and contour map, and marginal distributions, as shown in Fig. 7 and Fig. 8. No significant changes are there for 4-hour jobs, however the peaks of 3D map and contour map move up-down a bit, which means resource consumption is influenced, faintly.

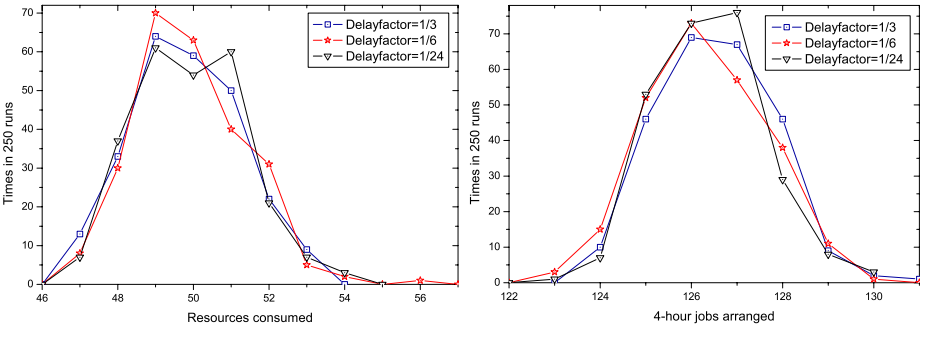


Fig. 8. BT solution marginal distributions with respect to Delayfactor

5 Experimental Comparison

Ant system have succeeded in many optimization and scheduling areas, and $\mathcal{MM}$AS ($\mathcal{MAX}$-$\mathcal{MIN}$ ant system) is an improved ant system which gives an upper and a lower bound to intensity quantity of trails in ant system [9,10]. We implemented a $\mathcal{MM}$AS for BT arrangement to compare. m BT related

operations in AGSS problem are transformed into m nodes in trade salesman problem (TSP). A distance between two nodes (operations) r_i and r_j is represented as

$$d_{r_i r_j} \stackrel{\text{def}}{=} \begin{cases} (DueDate_{r_j} - DueDate_{r_i})/10, & DueDate_{r_j} > DueDate_{r_i}, \\ 0.01, & DueDate_{r_j} = DueDate_{r_i}, \\ (DueDate_{r_i} - DueDate_{r_j})/30, & DueDate_{r_j} < DueDate_{r_i}. \end{cases}$$

And rest parameters of $\mathcal{MM}$AS are: $\alpha = 1.5$, $\beta = 2$, $\rho = 0.05$, $\tau_{\text{init}} = 1$, $\tau_{\max} = 100$, $\tau_{\min} = 0.01$, $\text{N}_{\text{ant}} = \lceil n/5 \rceil$ (upper integer), $\text{NC}_{\max} = 150$. And for each successful ant run R done by ant i, all of edges in Hamilton circle of R get a positive feedback

$$\Delta\tau^i_{r_i r_j} = \begin{cases} \frac{10}{(\text{res}_R + \text{job}_R/3)^2}, & < r_i, r_j > \in \text{circle of } R; \\ 0, & otherwise. \end{cases}$$

to reinforce the whole algorithm for fewer resources and jobs.

Then DSAFO (with parameters: $\text{AgentNumber}_{\text{BT}}$=4, Blockfactor=1/12, Delayfactor=1/6, Syncycle=5), EDD* (EDD in *run-and-schedule*), ERT* (earliest ready time first in *run-and-schedule*) and $\mathcal{MM}$AS were tested with real-world AGSS test data with 252 transfer flights. We choose BT related operations (1,008 activities in total) to test performances of these algorithms. A comparison in BT consumption and 4-hour BT jobs arrangement is shown in Table 1. Additionally we put time cost and average CPU rate in the table. The best value in each group is bolded.

Table 1. Optimization algorithm comparison

Algorithm	Time	CPU	Resources			4-hour jobs		
			MIN	MAX	AVG	MIN	MAX	AVG
DSAFO	≈144 sec	<1%	**47**	56	**49.9**	**123**	130	**126.3**
$\mathcal{MM}$AS	≈ 12 hours	≈100%	**47**	—	—	141	—	—
EDD*	**≈43 sec**	<1%	53	53	53	129	**129**	129
ERT*	**≈43 sec**	<1%	52	**52**	52	130	130	130

From Table 1, it can be concluded that DSAFO and $\mathcal{MM}$AS both do well in resource consumption for MRJSSPs, and DSAFO do better in BT jobs arrangement. Furthermore, DSAFO cost not too much time (several minutes) and very low CPU rate, in fact most time is used to maintain effective message transmission. On the contrary, $\mathcal{MM}$AS costs very much time and near 100% CPU rate.

6 Conclusion and Future Works

In this paper, we present a practical general model of the job shop scheduling problem, i.e. multi-resource job shop scheduling problem, and demonstrates

a design and optimization process on this problem with a novel multi-agent algorithm DSAFO. We have shown that this design and optimization process is comprehensive for extra scheduling constraints and the experimental results shows its effectiveness and efficiency.

One of the future works is to put forward an easy-to-use schedule software to simplify the design process. A preliminary development environment AGSAP has been developed to apply DSAFO to aid common AGSS optimization in [11]. In future, more easy-to-use development environments should be put forward for general MRJSSPs.

Acknowledgement

The authors acknowledge the support by National Natural Science Foundation of China (NSFC) under Grant No. 60472123.

References

1. Jain, A.S., Meeran,S.: Deterministic Job-Shop Scheduling: Past, Present and Future. European Journal of Operational Research, **113**(2) (1999) 390-434
2. Perregaard, M.: Branch and Bound Method for the Multiprocessor Jobshop and Flowshop Scheduling Problem. Master Thesis, Departement of Computer Science, University of Copenhagen, Danemark. (1995)
3. Cesta, A., Oddi, A., and Smith, S. F: Iterative flattening: a scalable method for solving multi-capacity scheduling problems. In Proceedings of the Seventeenth National Conference on Artificial intelligence and Twelfth Conference on innovative Applications of Artificial intelligence (July 30 - August 03, 2000). AAAI Press / The MIT Press (2000) 742-747
4. Nuijten, W.P.M., Aarts, E.H.L.: A Computational Study of Constraint Satisfaction for Multiple Capacitated Job Shop Scheduling. European Journal of Operational Research, **90**(2) (1996) 269-284
5. Garey, M.R., Johnson, D.S.: Computers and Intractability - a Guide to the Theory of NP-completeness. W.H. Freeman and Company, New York. (1979)
6. Yokoo, M., Durfee, E.H., Ishida, T., Kuwabara K.: The Distributed Constraint Satisfaction Problem: Formalization and Algorithms. IEEE Transactions on Knowledge and Data Engineering **10**(5) (1998) 673-685
7. Fan, W., Xue, F.: Optimize Cooperative Agents with Organization in Distributed Scheduling System. in *Second International Conference on Intelligent Computing* (ICIC 2006), Kunming, China, 2006. D.-S. Huang, K. Li, and G.W. Irwin (Eds.): Lecture Notes in Artificial Intelligence **4114** (2006) 502-509
8. Xing, J., Liu, S., Fan, W., Ji, L.: Design of Airport Ground Service System Based on Multi-Agent. *Journal of Civil Aviation University of China*. **24**(3) (2006) 24–27 (in Chinese)
9. Stützle, T., Hoos, H.: The $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System and Local Search for The Traveling Salesman Problem. In *Proceedings of the Fourth International Conference on Evolutionary Computation* (ICEC'97), IEEE Press. (1997) 308-313

10. Stützle, T., Hoos, H.: Improvements on the Ant System: Introducing $\mathcal{MAX}$–$\mathcal{MIN}$ Ant system. In *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms*, Springer Verlag, Wien. (1997) 245-249
11. Fan, W., Zhang, G., Xue, F.: Design and Implementation of Airline Ground Services Mas Development Platform. In *First Conference on Multi-agent Theory and Application*, Yantai, China. C. Y. Shi, Z. Z. Shi, *et al* (Eds.): Journal of Computer Research and Development **43** (s1) (2006) 414-419.