

Coordinate Multi-agent with Organization in Distributed Scheduling System

A Dissertation Submitted to
Civil Aviation University of China
For the Academic Degree of Master of Science

BY
XUE Fan

Supervised by
Prof. FAN Wei

College of Computer Science and Technology
Civil Aviation University of China
25th February 2007

分类号: TP18 密 级: 公开
UDC: 004.89 学 号: 048030307

中国民航大学

硕 士 学 位 论 文

结合组织模型的多Agent分布式调度研究

研究生姓名: 薛 帆
导师姓名: 樊 玮 教授

申请学位级别: 工学硕士 学科专业名称: 计算机应用技术
所在院系: 计算机科学与技术学院 论文答辩日期: 2007 年 3 月 17 日

2007 年 2 月 25 日

中国民航大学学位论文独创性声明

本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得中国民用航空学院或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

研究生签名：_____ 日 期：_____

中国民航大学学位论文使用授权声明

中国民航大学、中国科学技术信息研究所、国家图书馆有权保留本人所送交学位论文的复印件和电子文档，可以采用影印、缩印或其他复制手段保存论文。本人电子文档的内容和纸质论文的内容相一致。除在保密期内的保密论文外，允许论文被查阅和借阅，可以公布（包括刊登）论文的全部或部分内容。论文的公布（包括刊登）授权中国民航大学研究生部办理。

研究生签名：_____ 导师签名：_____ 日 期：_____

To My Beloved Parents

Acknowledgements

First and foremost, I thank my advisors Prof. Fan Wei. He is a first-rate mentor and great friend, from whom I learnt more than academic skills, moral characters and social adaptability. It is difficult to fully outline all the ways in which he has contributed to my study and growth. It is impossible to fully express my gratitude. I would like to thank Prof. Zhu Shi-Xing and Dr. Gu Zhao-Jun for their delightful collaborations and fatherly enlightenments.

I have been fortunate to have such great colleagues and friends, Zhang Guang-Cai, Wang Xing-Yun, Wang Yuan-Kun, Zhang Jie, Guo Qi-Ming, Fan Jing-De over the past years as well as the research group of Software Technology Research Center, and I got huge support from them in all-round. I thank Mr. Huang Xiao-Rong, Ms. Huang Cui-Wei and Ms. Pan Hai-Ying in Xiamen Airline, Mr. Liu Yun-Lei in Center of Aviation Safety Technology CAAC, Ms. Kang Li-Ping in AMECO, it was a pleasure to collaborate with them. A very special thanks goes to my friends for their support on this thesis and past three-year academic life, especially Ellen Zhang and Anne Cao.

I also thank the foundation support by National Natural Science Foundation of China (NSFC) (Grant No. 60472123, Jan. 2005 – Mar. 2006) and Doctoral Startup Foundation of Civil Aviation University of China (Grant No. QD13X04, Jan. 2004 – Jan. 2006).

Finally, I thank from the bottom of my heart, my parents, for giving me life, culturing me, offering me unconditional support and encouraging me to pursue my dreams. I would also like to thank my younger brother, with whom I had a cherished golden childhood.

摘 要

许多工程领域中的调度和规划问题都相当地困难，尤其是大规模调度和规划优化问题。飞机地面作业调度（AGSS）就是这样的一个问题。

本文在回顾了飞机地面作业调度相关领域的研究后，首先对飞机地面作业调度问题在约束满足问题框架下进行了形式化。并证明了该问题是 \mathcal{NP} -完全（在某些情况下是 PSPACE-完全）的困难问题。

随后提出了一个面向飞机地面作业调度问题的动态分布式调度模型、一个动态动态收集并融合飞机地面作业相关的数据的调度环境 run-and-schedule 和一个 DSAFO（Dynamic Scheduling Agents with Federation Organization，具有联邦结构的动态调度 Agent）算法。DSAFO 算法是一个专为飞机地面作业调度问题开发的新颖的多 Agent 算法，该算法引入了两种策略来满足飞机地面作业调度中的约束：局部启发式和基于 Agent 角色和联邦组织实现的全局协作。

DSAFO 进行飞机地面作业调度的主要步骤是：实时地从 run-and-schedule 接收航班数据，将航班需求分解为许多子作业；利用多 Agent 动态地将每套子作业的解空间分割为合理的划分；在每个划分（Agent）内进行局部启发式求解；利用划分间的协作进行全局解的优化；并同时结果分发到飞机服务资源上。

DSAFO 算法具有不错的时间复杂度：介于平方和三次方之间。虽然实验证实 DSAFO 是不稳定算法，而且受到几个参数的影响，但是该算法能够很好地满足全部约束、跳出局部极小值、寻找资源耗费和人力分配的近优解。在对参数造成的影响进行了深入的实验和理论分析后，文章将 DSAFO 算法同 MMAS 蚂蚁算法和传统启发式算法进行了实验对比。

最后给出了 DSAFO 的研究总结和飞机地面作业调度问题的未来研究方向。

关键词 多 Agent 算法，飞机地面作业， \mathcal{NP} -完全，分布式约束满足问题，多价 π -演算，分布式调度

Abstract

Numerous scheduling and planning problems in various industrial environments are known to be extremely challenging, especially large scale scheduling and planning optimization problems. Airport Ground Service Scheduling (AGSS) problem is such a problem.

After a brief review of researches on AGSS related areas, formulations of AGSS problem are presented from constraint satisfaction view. Furthermore, AGSS problem is classified as a \mathcal{NP} -complete (PSPACE-complete in some cases) scheduling problem.

A dynamic distributed scheduling model is structured for AGSS problem then, and a dynamic distributed scheduling environment *run-and-schedule* is put forward to collect and uniform AGSS related data. DSAFO (Dynamic Scheduling Agents with Federation Organization) is a novel multi-agent algorithm for AGSS problem. To fulfill constraint satisfactions and optimizations in AGSS, DSAFO employs two strategies: local heuristics and global coordination, based on roles of agents in a federation organization.

In a typical AGSS solving process, DSAFO accepts real-time flights data from *run-and-schedule* environment; decomposes flight service goals into operations, according to gathered data; divides the solution space dynamically into rational partitions with multi-agents; conquers each partition with local heuristics within an agent; optimizes the solution simultaneously via coordination among partitions from global view; and dispatches the solution to real world aircraft service resources simultaneously.

The complexity of DSAFO is bounded between quadratic and cubic polynomial time. Though experiments show that DSAFO is unstable and influenced by several parameters, this algorithm is good at satisfying all constraints, jumping out of local minimum, and finding near optimal solutions for consumption of resources and man-days. After careful experiments and theoretical analysis on parameters in DSAFO, a comparison is presented with three opponent algorithms, including a \mathcal{MMAS} approach and two traditional heuristics.

Finally a brief conclusion of DSAFO and the future research directions in AGSS are given at the end of this thesis.

Keywords Airport ground service, Distributed constraint satisfaction problem, Distributed scheduling system, Multi-agent algorithm, \mathcal{NP} -complete, Polyadic π -calculus

Contents

Dedication	i
Acknowledgements	ii
Chinese Abstract and Keywords	iii
Abstract and Keywords	iv
List of Tables	ix
List of Figures	xi
List of Abbreviations	xv
1 Introduction and Motivation	1
1.1 What is AGSS	1
1.2 Motivation	5
1.2.1 Economic importance	5
1.2.2 Existing management problems	5
1.2.3 Benefits of effective AGSS	6
1.3 Thesis contributions	6
1.4 Thesis outline	7
2 Background and Related works	9
2.1 AGSS investigations	9
2.2 Job-Shop Scheduling Problem	10
2.3 Distributed Constraint Satisfaction Problem	11
2.4 Coordinative multi-agent system	13
2.4.1 Multi-agent system	13
2.4.2 Coordinative multi-agent system	14
2.4.3 Agent organization paradigms	19

2.5	Polyadic π -calculus	23
3	AGSS: Formulation and Characteristics	25
3.1	Assumptions	25
3.2	Formulation	26
3.2.1	Airport ground service	26
3.2.2	AGSS satisfaction	27
3.2.3	AGSS problem	30
3.2.4	Uncertain AGSS satisfaction	31
3.2.5	Uncertain AGSS problem	32
3.3	Notes in practical AGSS programming	32
3.4	Characteristics	33
4	Dynamic Distributed Scheduling Modeling	34
4.1	Motivation	34
4.2	Model overview	35
4.3	Run-and-scheduling	36
5	DSAFO: Overview, Design and Implementation	38
5.1	An overview	38
5.2	DSAFO strategies	40
5.2.1	Local heuristics	40
5.2.2	Global coordination	41
5.3	Agent roles in DSAFO	41
5.3.1	Role <i>Blackboard</i>	42
5.3.2	Role <i>ResourceAdmin</i>	44
5.3.3	Role <i>Member</i>	45
5.3.4	Role <i>Coordinator</i>	48
5.4	A formalized summary	49
5.4.1	Blackboard	49
5.4.2	ResourceAdmin	50

5.4.3	Member	51
5.4.4	Coordinator	52
5.5	Complexity	53
5.5.1	UC's Complexity	53
5.5.1.1	Complexity from agent behaviors	53
5.5.1.2	Complexity from agent communications	55
5.5.1.3	UC's complexity summary	56
5.5.2	Complexity of DSAFO	56
5.6	Implementation	56
6	Experiments and Analysis on Diverse Parameters	60
6.1	The experiment	60
6.2	The factors	61
6.2.1	<i>Member</i> agent number and Reqcycle	61
6.2.2	Blockfactor	65
6.2.3	Delayfactor	68
6.2.4	Syncycle	71
6.3	Experimental summary	71
7	Comparison	75
7.1	Opponents	75
7.1.1	<i>MMAS</i>	75
7.1.2	EDD* and ERT*	77
7.2	Comparison	77
8	Conclusion and Future Works	80
8.1	Conclusion	80
8.2	Future works	80
	Appendix A: Agent Communication Grammar in DSAFO	81
	Publications During M.Sc. Study	97
	Curriculum Vitæ	98
	References	97

List of Tables

1–1	Resource requirements in airport ground service	4
2–1	A taxonomy of MAS organizational paradigms	21
7–1	Mapping BT operations to TSP points	77
7–2	Optimization algorithm comparison	79
A–1	ACL protocols and agent communication grammar in DSAFO	81

List of Figures

1-1	Typical operations for a transfer flight	2
1-2	Typical operations for a departure flight	3
1-3	Typical operations for an incoming flight	3
2-1	A taxonomy of agent interaction behavior	15
4-1	Dynamic distributed model for AGSS	37
5-1	Algorithm category of DSAFO	39
5-2	Channel organization for agent communication	42
5-3	Behaviors and channel utilization of agent role <i>Blackboard</i>	44
5-4	Behaviors and channel utilization of agent role <i>ResourceAdmin</i>	45
5-5	Behaviors and channel utilization of agent role <i>Member</i>	48
5-6	Behaviors and channel utilization of agent role <i>Coordinator</i>	49
5-7	Gantt chart of typical scheduled plans for a flight (the <i>Blackboard</i>)	57
5-8	Gantt chart of typical scheduled plans for resources (a BT <i>Member</i> agent)	58
5-9	DSAFO on JADE RMA GUI	58
5-10	Communication observed by JADE sniffer	59
5-11	Agent inner states observed by JADE introspector	59
6-1	BT solution distributions with respect to agent number	62
6-2	BT solution distributions with respect to agent number (more)	63
6-3	BT solution marginal distributions with respect to agent number	64

6-4	BT solution distributions with respect to Blockfactor	66
6-5	BT solution marginal distributions with respect to Blockfactor	67
6-6	BT solution distributions with respect to Delayfactor	69
6-7	BT solution marginal distributions with respect to Delayfactor	70
6-8	BT solution distributions with respect to Syncycle	72
6-9	BT solution marginal distributions with respect to Syncycle	73
7-1	A simple illustration of EDD and ERT	78

List of Abbreviations

Abbreviation	Description	
A-SMGCS	Advanced-Surface Movement Guidance and Control Systems	10
ACL	Agent Communication Language	16, 56, 68
ACO	Ant Colony	11, 18, 74
ADOPT	Asynchronous Distributed OPTimization	13
AGSS	Airport Ground Service Scheduling	4, 38, 60
AI	Artificial Intelligence	7
ANN	artificial neural networks	11
AODB	Airport Operation Data Base	36
ARCHON	ARchitecture for Cooperative Heterogeneous ONline system	17
ARM	Airline Resource Management system	9, 17
AS	Ant System	75
BCIA	Beijing Capital International Airport	10, 60
BDI	Belief-Desire-Intention	18
BOID	Beliefs-Obligations-Intentions-Desires	18
BT	Baggage Tractor	4, 53
CMDP	single-agent Constrained MDP	19
CNET	Contract NET	16
COM-MTDP	COMmunicative Multiagent Team Decision Problem	19
Coo-BDI	Cooperative BDI	18
CSP	Constraint Satisfaction Problem	11

Abbreviation	Description	
DAI	Distributed AI	11, 14
DEC-MDP	DECentralized MDP	19
DEC-POMDP	DECentralized POMDP	19
Dis-CSP	Distributed Constraint Satisfaction Problem	12, 30, 33
Dis-DSS	Distributed Dynamic Scheduling System	9, 17
Dis-HTN	Distributed HTN	17
DJSSP	Dynamic Job-Shop Scheduling Problem	31
dMARS	distributed Multi-Agent Reasoning System	18
DSAFO	Dynamic Scheduling Agents with Federation Organization	38, 60, 77
DSIPE	Distributed System for Interactive Planning and Execution	17
DSS	Distributed Scheduling System	9
EDD	Earliest Due Date	11, 40, 77
EMTDP	Extended Multiagent Team Decision Problem	19
ERT	Earliest Ready Time	11, 77
FA/C	Functionally Accurate, Cooperative system	17
FCFS	First-Come-First-Serve	11
FIDS	Flight Information Display System	36
FIFO	First-In-First-Out	11
FIPA	Foundation for Intelligent Physical Agents	16, 56
FOC	Flight Operations Control system	36
FSTS	Fuzzy Subjective Task Structure	18
GA	Genetic Algorithm	9, 11
GBB	Generic Blackboard	9
GIS	Geographical Information System	10
GPGP	Generalized Partial Global Planning	17
HPS	Hospital Patient Scheduling	17
HTN	Hierarchical Task Network	17

Abbreviation	Description	
JADE	Java Agent DEvelopment framework	56, 60, 68
JSSP	Job-Shop Scheduling Problem	10, 31, 33, 76
KQML	Knowledge Query Manipulation Language	16
LB	Load Baggage	4, 60
LC	Load Cargo and mail	4, 60
LGPL	Lesser General Public License Version	56
MAS	Multi-Agent System	11, 13, 16, 20
MBO	Model-Based Optimization	11
MDP	Markov Decision Process	18
MIS	Management Information System	9
\mathcal{MMAS}	$\mathcal{MAX-MIN}$ Ant System	76
MMDP	Multiagent MDP	19
MSF	minimum slack	11
NEXP-complete	Non-EXponential complete	19
\mathcal{NP}-complete	Non-Polynomial complete	11, 28, 30
PGP	Partial Global Planning	17
POMDP	Partially Observable Markov Decision Process	13, 19
poset	partial ordered set	26
PSO	Particle Swarm Optimization	11, 74
PSPACE-complete	Polynomial SPACE complete	31
QBF	Quantified Boolean Formula	32
RCS	Rolegraph Coordination Strategy	18
SA	simulated annealing	11
SIPE	System for Interactive Planning and Execution	17
SPT	shortest processing time	11

Abbreviation	Description	
TAAM	Total Airport & Airspace Modeller system	9
TÆMS	a framework for Task Analysis, Environment Modeling, and Simulation	17
TM	Turing Machine	18, 31
TSP	Travel Salesman Problem	75
UB	Unload Baggage	4, 60
UC	Unload Cargo and mail	4, 53, 60

Chapter 1 Introduction and Motivation

Numerous scheduling and planning problems in various industrial environments are known to be extremely challenging. Because from computational complexity view, they are \mathcal{NP} -hard or harder [1, 2, 3]. It is extremely difficult for traditional optimization algorithms to solve this type of problems in their general forms. AGSS (*airport ground service scheduling*), which is focused in this thesis, is such a very difficult scheduling problem.

1.1 What is AGSS

AGSS (*airport ground service scheduling*), briefly, is the scheduling activities for *airport ground service*.

In this thesis, *airport ground service* is the series of service processes from flight landing to takeoff, including baggage handling, catering, fueling, cleaning, etc. In this thesis, we called the service processes as “*operations*”. It should be noticed that H. Hartmann (2001) divided this definition into three partitions: *Ground traffic service* (follow-me), *Airport ground service* (towing, de-icing, refuel, etc.) and *Airline ground service* (ground handling: load, unload, catering, water, lavatory, etc.) [4]. His partitions are mainly from individual enterprise view, oppositely our definition is mainly from systematic collaboration view.

From flight landing on to takeoff, there are many service operations to be fulfilled for an aircraft in different work flow orders, according to different flights and situations [5]. Service operations and their orders may vary from one flight to another. For example, typical operation flows for a typical transfer flight, a departure flight, and an incoming flight are respectively shown in Fig. 1-1, Fig. 1-2, and Fig. 1-3.

In Fig. 1-1, operations which are marked with a light pink background color are some critical operations which usually delay flights. And those connected with dotted

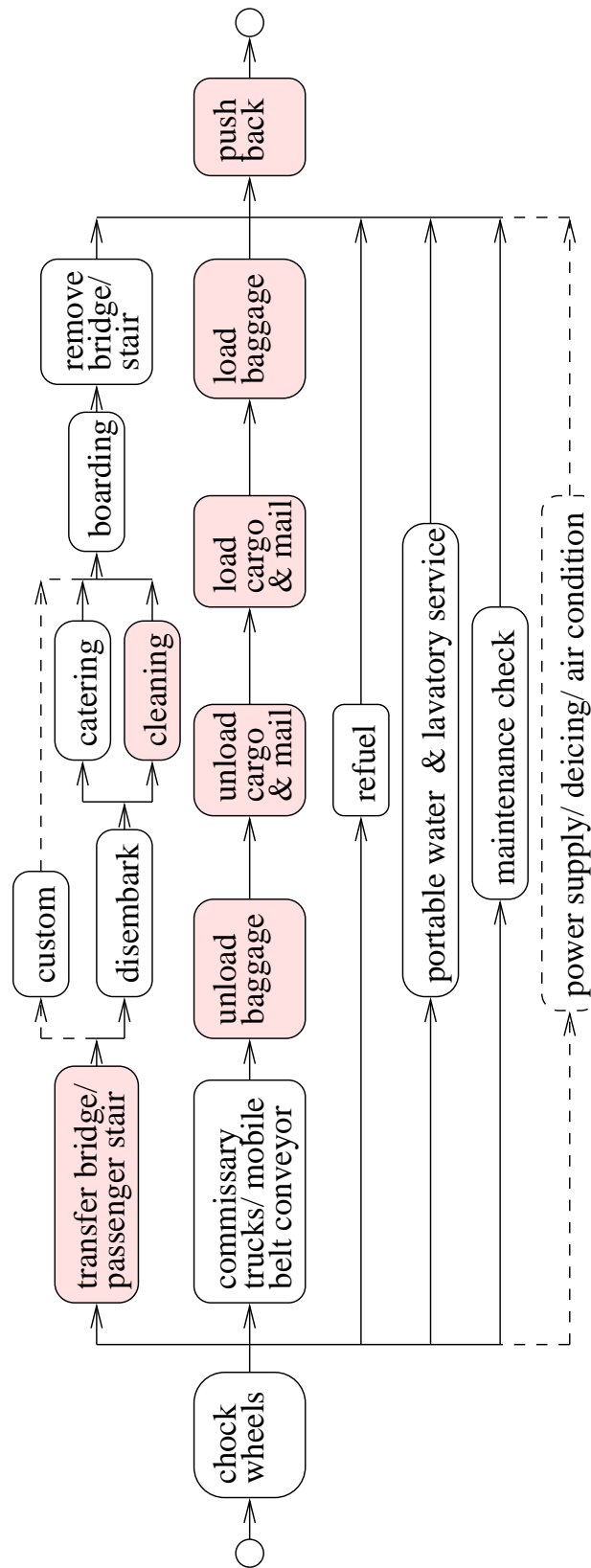


Figure 1-1: Typical operations for a transfer flight

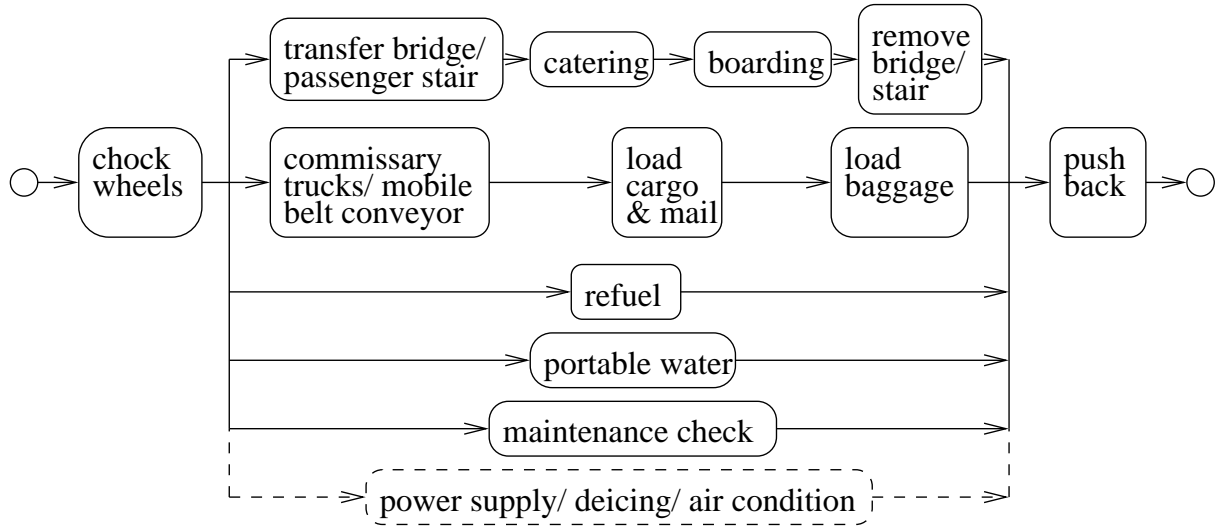


Figure 1-2: Typical operations for a departure flight

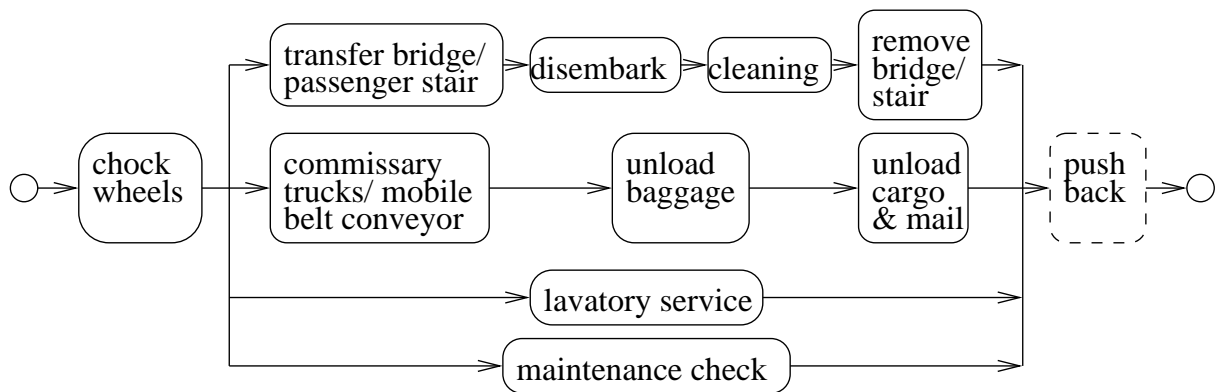


Figure 1-3: Typical operations for an incoming flight

arrow are optional operations for different kinds of flights, they depend on specific flight's internationality, real-time aircraft maintenance requirement, and so on.

For each type of operations shown in Fig. 1-1, Fig. 1-2, and Fig. 1-3, certain type(s) of materials, productions, workforce, engineers, aviation ground support facilities and equipments, and transportation equipments are required. In this thesis, we call these requirements as “*resources*”, and the typical resource requirements in airport ground service are listed in Table 1-1, categorized by different operation types.

Table 1-1: Resource requirements in airport ground service

Operation Type	Resource Type
chock wheels	wheel chock
handling preparation	commissary truck/ mobile belt conveyor
unload baggage (UB)	baggage tractor (BT) + carts
unload cargo and mail (UC)	baggage tractor (BT) + carts
load cargo and mail (LC)	baggage tractor (BT) + carts
load baggage (LB)	baggage tractor (BT) + carts
deplaning preparation	transfer bridge/ passenger stairs
cleaning	cleaners (+ cleaners' bus)
catering	catering truck
refueling	refuel truck
portable watering	portable water cart
lavatory service	lavatory truck/ cart
rubbish disposal	rubbish truck
maintenance check	maintenance engineer (+ stand)
push back	push-back tractor (+ towing bar)
power supply (optional)	ground power supply
air supply (optional)	air conditioner
deicing (optional)	deicer

The AGSS (*airport ground service scheduling*) problem is the process to schedule *airport ground service* effectively and efficiently [5]. P. Baptiste, C. Le Pape and W. Nuijten (1995) stated that “scheduling is the process of assigning *activities* to *resources* in time” [6]. Following this definition, the AGSS (*airport ground service scheduling*) problem is the process of assigning all constrained operations (aircraft service processes) to many kinds of dynamic resources (such as engineers, aviation ground support facilities and equipments, material, transportation equipments, etc.), in order to meet every flight's arrival and departure deadlines timely.

Moreover, formalized definitions on AGSS problem are discussed further in Chapter 3, where more theoretical and practical characteristics of AGSS are introduced as well.

1.2 Motivation

The AGSS problem is economically very important to airline industry, but currently there are some management problems in AGSS process in China.

1.2.1 Economic importance

On one hand, AGSS is important to economy of airlines and airports. In 2006 first season, average flight normal rate of all China flights is 81.07%, and the lowest three company Shenzhen Airline, Xiamen Airline, and Shanghai Airline has respectively 75.86%, 78.65%, and 75.87% [7]. Such a high rate of flight delay would bring enormous economic loss to airlines and airports. According to L. Shi (2005), loss caused by abnormal flights counts for 2–3% of total flight operation cost which is usually about *tens of billion* RMB dollars for large airline in China [8]. That means, abnormal flights bring *hundreds of million* RMB dollars to China airline. Among all kinds of reasons responsible for flight delay, AGSS is one of the most important factors.

On the other hand, AGSS is important to services quality of airlines and airports. Convenient departure and arrival time and timeliness, which is provided by AGSS, are very important factors for airline professionals and passengers to evaluate airline services quality: they count for 12.8% of all factors in Taiwan reported by S.-H. Tsaur, T.-Y. Chang and C.-H. Yen (2002) [9].

In conclusion, AGSS is very important for airline industry.

1.2.2 Existing management problems

However there currently exist two major problems in AGSS process in China:

During tens of years enterprise management, most airlines and airports have built kinds of information systems. But these systems were constructed for different targets in different time, they lack of uniformed blueprints. They become isolated information islands. Additionally, some fundamental infrastructures are in short. So in some airports, information for AGSS now cannot be systematically gathered, and emergency strategies

cannot be applied efficiently. For example, once a typhoon went ashore, airport in Macao computed that flights did not need grounding, but the Airport in Shenzhen could not compute result in time, so all the flights grounded, and the economic loss should be counted in hundreds of millions [10].

Secondly, in current airlines and airports, real AGSS operational mechanisms are mainly with single resource type, i.e., scheduling different type of vehicles with isolated different methods. Systematic scheduling has not been considered. Additionally, most of scheduling methods are personally experiential, so the manpower and vehicles cannot be fully utilized. AGSS should be finished via many types of resources and coordination among them, because there might be spatial collision in limited job space around aircraft body. However there still lacks of systematic and scientific decision approaches to process AGSS rationally.

1.2.3 Benefits of effective AGSS

Effective AGSS, which this thesis is expected to achieve, is greatly profitable for airline industry. It is because:

1. effective AGSS solution results in efficient time-sequenced aircraft service plans and yet minimizes aircraft transfer time;
2. it identifies the preferential requirements that should be produced properly and yet minimizes cost on man-days, equipment and inventory;
3. it also improves airline's ground scheduling capability under uncertain situations, and identifies possible delayed aircraft services due to lack of availability in uncertain flight arrivals.

1.3 Thesis contributions

The main contributions of this thesis are:

1. to formalize the AGSS problem from constraint satisfaction view, and reveal some characteristics of AGSS;
2. to establish a dynamic distributed scheduling model and a dynamic scheduling environment run-and-schedule for AGSS problem;

3. to develop a novel multi-agent approach DSAFO, practically and formally, satisfy and optimize AGSS problem;
4. to study parameters of DSAFO, and give advice on how to incorporate DSAFO with suitable parameters into AGSS problems;
5. to represent that the nature of DSAFO is a coordination optimization model based on a traditional Artificial Intelligence (AI) concept Divide-and-Conquer, and point out that the optimization of DSAFO comes from dynamic message transmission and organization-based coordination.
6. to develop a $\mathcal{M}\mathcal{M}\mathcal{A}\mathcal{S}$ ($\mathcal{M}\mathcal{A}\mathcal{X} - \mathcal{M}\mathcal{I}\mathcal{N}$ Ant System, in Subsection 7.1.1) approach for AGSS problem, for algorithm comparison.

1.4 Thesis outline

The remainder of this thesis is organized into seven chapters.

Chapter 2 offers a general overview of former AGSS researches and applications, and also reviews two research areas related to AGSS: *job-shop scheduling problem* (JSSP, in Section 2.2) and Distributed Constraint Satisfaction Problem (Dis-CSP, in Section 2.3). Coordinative *multi-agent system* (MAS) and its organizational paradigms are surveyed in Section 2.4 as well. In addition, polyadic π -calculus is also reviewed in Section 2.5.

Chapter 3 discusses the formalization of AGSS problem, after making some ideal assumptions for computational availability. Practical scheduling features and characteristics are also described, in order to adapt the formal analysis to real problem.

Chapter 4 states the advantage and necessity of modeling AGSS within distributed scheduling model in Section 4.1, and presents a distributed scheduling model in Section 4.2 and a dynamic scheduling environment *run-and-scheduling* for AGSS problem in the last section.

Chapter 5 details the multi-agent optimization algorithm DSAFO, which is proposed to optimize AGSS effectively. The nature and strategies of DSAFO are introduced in Section 5.2; Agent roles and their formalizations are presented in Section 5.3 and Section 5.4 respectively; Section 5.5 analyzes the time complexity of DSAFO; Section 5.6 lastly implements this multi-agent optimization algorithm in JADE and FIPA ACL.

Chapter 6 describes a real-data-driven AGSS experiment to prove the performance of DSAFO. Section 6.2 demonstrates the DSAFO's optimization results and optimization variation under variant parameters of DSAFO. Section 6.3 gives a summary of the experiment results and characteristics of DSAFO.

Algorithm comparison appears in Chapter 7. DSAFO is compared with three opponent algorithms: \mathcal{MMAS} , EDD^* and ERT^* . The \mathcal{MMAS} algorithm for AGSS problem is detailed in Subsection 7.1.1.

Finally, a conclusion is given in Chapter 8, where research summary of DSAFO and hopeful future research directions in AGSS are represented as well.

Chapter 2 Background and Related works

In AGSS domain, many researches and systems have been done from theoretical algorithms to industrial applications; The JSSP and DisCSP, which are related to AGSS problem, have been investigated largely from the mathematical programming to Artificial Intelligence (AI) areas; Multi-agent coordination and organizational paradigms, which are important foundation of algorithm proposed in this thesis, have also been developed for several decades.

2.1 AGSS investigations

In the *Airline Resource Management* (ARM) system, a multi-agent system *Distributed Dynamic Scheduling System* (Dis-DSS) for AGSS had been done by D. E. Neiman, D. W. Hildum, V. R. Lesser, *et al* [11, 12]. This Dis-DSS evolved from *Distributed Scheduling System* (DSS), by employing a *Generic Blackboard* (GBB). Dis-DSS employs resource texture measure, *most-tightly-constrained-first*, negotiation, and synchronous plan repair to generate schedules. D. E. Neiman and V. R. Lesser then improved Dis-DSS with a cooperative repair method to deal unsatisfiable situations [13]. And M. Chia, D. E. Neiman, *et al* enhanced Dis-DSS with two cooperative behaviors *poaching* and *distraction* [14].

Another approach, which schedules AGSS with *Genetic Algorithm* (GA), was also investigated by A. Cheung, W. H. Ip, D. Liu and C. L. Lai [15]. They proposed a modified operation-based chromosome representation to stand for different trucks and tractors in order to generate vehicle service schedules.

In industrial applications, a Boeing company Preston¹ presented a system called TAAM (Total Airport & Airspace Modeller) for airport resource management, which includes to allocate all gates and baggage belts trucks. Lufthansa built the PERSEUS project² and its support staff demanded planning, shift planning and the allocation of

¹See <http://www.preston.net>

²See <http://www.groundstar.de>

staff in real time for Lufthansa's passenger services. ASCENT technology inc.³ presented ARIS/AR routing system to schedule baggage handlers, cleaners, caterers, ramp workers, ground equipments, etc. The *Second Research Institute, General Administration of Civil Aviation of China*⁴ produced a *Management Information System* (MIS) named *Command Scheduling System* (指挥调度系统), as a subsystem of *Airport Integrated Information Management System* (机场综合信息管理系统), to do ground resource allocation and operation result collection. And the *Command Scheduling System* was widely used in many domestic airports such as Shenzhen, Kunming, Chengdu, Hangzhou, Zhengzhou, Xi'an, Xiamen, Tianjin, etc.

Moreover, A-SMGCS (*Advanced-Surface Movement Guidance and Control Systems*) within *Park Air Systems*⁵ by Northrop Grumman Corporation⁶ spread over the world for complete control of airport traffic movements. And there was another airport ground traffic assistant with GIS (Geographical Information System) technology which researched by *Beijing University of Aeronautics and Astronautics* for *Beijing Capital International Airport* (BCIA) [16].

2.2 Job-Shop Scheduling Problem

Definition 1 (JSSP). *A. S. Manne firstly gave a definition of Job-Shop Scheduling Problem (JSSP) in 1960 [17]:*

...the sequencing problems involves the performance of n tasks — each task being defined in such a way as to require the services of a single machine for an integral number of time units. Any one end product will, in general, necessitate the performance of several tasks in sequence. The scheduling problem consists of drawing up a plan for time-phasing the individual jobs so as to satisfy:

- *sequencing requirements,*
- *equipment interference problems.*

The integer-valued unknowns x_j are indicated the day on which task j is to begun ($x = 0, 1, \dots, T$). The schedule is to be drawn up so as to minimize the 'make-span'.

³See <http://www.ascent.com>

⁴See <http://www.caacsri.com>

⁵See <http://www.parkairsystems.com>

⁶See <http://www.northropgrumman.com>

Besides, a Constraint Satisfaction definition was also given by A. S. Jain and S. Meeran [18].

There are much many ways to solve general JSSPs. One type of the techniques is mathematical strategy, sometimes called accurate algorithm, including dynamic programming, decomposition strategies, enumerative techniques, Model-Based Optimization (MBO), etc. [19]. However JSSP is a well known \mathcal{NP} -complete problem [1], so the mathematical strategy has been limited.

Another type is approximate algorithm, including dispatching rules (sequencing rule or scheduling rule), heuristics⁷ and Lagrangian relaxation⁸. The heuristics could be based on processing times (such as shortest processing time (SPT)), due dates (such as earliest due date (EDD)), slack (such as minimum slack (MSF)), arrival times (such as earliest ready time (ERT, known as first-in-first-out FIFO or first-come-first-serve FCFS as well)), and kinds of combinations of them [20]. These approaches are quite simple and in low complexity, however they are approximate and generally cannot output the optimal solution for a JSSP.

Yet another type is AI technique, generally called intelligent algorithm. Starting in the early 1980s, a series of new technologies were applied to JSSPs, such as expert/knowledge-based systems [2, 21], Distributed AI (DAI, such as multi-agent system (MAS) [22, 23, 24, 25]), artificial neural networks (ANN) [26], tabu search [27, 28, 29], simulated annealing (SA) [30, 31], genetic algorithms (GA) [32, 33], Ant Colony (ACO) [34], Particle Swarm Optimization (PSO) [35], etc. Each of them has a special way to find near optimal solutions.

Fuzzy logic might be classified as the last type, and it is a dependent type. Because *fuzzy set* theory has only been utilized to develop hybrid scheduling approaches [36, 37].

2.3 Distributed Constraint Satisfaction Problem

Definition 2 (CSP). *Formally, a Constraint Satisfaction Problem (CSP) consists of n variables x_1, x_2, \dots, x_n , whose values are taken from finite, discrete domains D_1, D_2, \dots, D_n , respectively, and a set of constraints on their values. In general, a constraint is defined by a predicate. That is, the constraint $p_k(x_{k_1}, \dots, x_{k_j})$ is a predicate that is defined on the*

⁷It should be noticed that dispatching rules usually is heuristics; but heuristics includes more than dispatching rules.

⁸Sometimes Lagrangian relaxation is classified as a mathematical strategy, anyway, it is approximate.

Cartesian product $D_{k_1} \times \dots \times D_{k_j}$. This predicate is true iff the value assignment of these variables satisfies this constraint. Solving a CSP is equivalent to finding an assignment of values to all variables such that all constraints are satisfied [38].

Algorithms for solving Constraint Satisfaction Problems (CSPs) can be divided into two groups, i.e., search algorithms and consistency algorithms. The search algorithms for solving CSPs can be further divided into two groups, i.e., backtracking algorithms and iterative improvement algorithms.

Backtracking A backtracking algorithm is a basic, systematic search algorithm for solving CSPs, such as min-conflict heuristic [39]. For example, the *weak-commitment search algorithm* [40] is based on the min-conflict backtracking;

Iterative improvement In iterative improvement algorithms, as in the min-conflict backtracking, all variables have tentative initial values. However, no consistent partial solution is constructed. A *flawed* solution that contains all variables is revised by using hill-climbing search;

Consistency algorithms Consistency algorithms are preprocessing algorithms that reduce futile backtracking [41].

Definition 3 (Dis-CSP). *An instance of the Distributed Constraint Satisfaction Problem (Dis-CSP) is a tuple $\langle \mathcal{A}, X, D, \mathcal{C} \rangle$ where*

- $\mathcal{A} = \{\alpha_1, \dots, \alpha_p\}$ *-a set of p agents;*
- $X = \{X_{\alpha_1}, \dots, X_{\alpha_p}\}$ *-a set of p variables sets,*
- for each $\alpha \in \mathcal{A}$, $X_\alpha = \{x_\alpha^1, \dots, x_\alpha^{q_\alpha}\}$* *-each variables set α consists of q_α variables;*
- $D = \{D_{\alpha_1}, \dots, D_{\alpha_p}\}$ *-a set of value domains for each agent,*
- for each $\alpha \in \mathcal{A}$ and $\chi \in X_\alpha$, $\chi \in D_\alpha(\chi)$* *-the range of each variable χ , and the value may be assigned by (and only by) agent α ;*
- $\mathcal{C} = \{c_1, \dots, c_r\} = \mathcal{C}_{\alpha_1} \cup \dots \cup \mathcal{C}_{\alpha_p}$ *-a set of constraints on the variables,*
- for each $i \in \{1, \dots, r\}$, $c_i : D_{\alpha_1}(x_{\alpha_1}^1) \times \dots \times D_{\alpha_1}(x_{\alpha_1}^{q_{\alpha_1}}) \times \dots \times D_{\alpha_p}(x_{\alpha_p}^{q_{\alpha_p}}) \mapsto \{\text{true}, \text{false}\}$* *-each predict may be related to any variables;*
- for each agent $\alpha \in \mathcal{A}$, $\mathcal{C}_\alpha = \{c_i | c_i \in \mathcal{C} \wedge \exists_{x \in X_\alpha} c_i \text{ is relative to } x\}$,* *-a set of all constraints related to agent α , and it must be known by α .*

The Dis-CSP problem is to find an assignment

$$a \in D_{\alpha_1}(x_{\alpha_1}^1) \times \dots \times D_{\alpha_1}(x_{\alpha_1}^{q_{\alpha_1}}) \times \dots \times D_{\alpha_p}(x_{\alpha_p}^{q_{\alpha_p}})$$

such that

$$\forall_{1 \leq i \leq r} c_i(a) \equiv \text{true}.$$

Well, a similar formulation can also be seen at [42]. And algorithms for solving Dis-CSPs can be listed as follow:

Asynchronous backtracking The asynchronous backtracking algorithm is a distributed, asynchronous version of a backtracking algorithm [42, 43];

Asynchronous weak-commitment search Asynchronous weak-commitment search algorithm proposed by M. Yokoo introduces the min-conflict heuristic to reduce the risk of making bad decisions [42, 44, 45]. Furthermore, the agent ordering is dynamically changed so that a bad decision can be revised without performing an exhaustive search. This algorithm is also useful to deal with collisions in multi-agent systems;

Distributed breakout algorithm In this algorithm, two kinds of messages (*ok?* and *improve*) are communicated among neighbors to jump out of local minimum;

Distributed consistency algorithm Achieving 2-consistency by multiple agents is relatively straightforward, since the algorithm can be achieved by the iteration of local processes [46];

ADOPT P. J. Modi, W.-M. Shen, M. Tambe and M. Yokoo have developed an algorithm called Asynchronous Distributed OPTimization (ADOPT) that can solve more general distributed constraint optimization problems [47];

Multi-agent POMDP i.e. Multi-agent Partially Observable Markov Decision Processes [48].

2.4 Coordinative multi-agent system

2.4.1 Multi-agent system

Multi-agent system (MAS) is a promising new paradigm in computing. In general, a *multi-agent system* is a system in which several interacting and intelligent *agents* pursue some set of goals or tasks. An *agent* is a computational entity such as a software program or a robot that can be viewed as perceiving and acting upon its environment and that is

autonomous in that its behavior is at least partially depends on its own experience [49]. M. Wooldridge and N. Jennings (1995) presented four characteristics of agent [50]:

- *autonomy*: Agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state;
- *social ability* : Agents interact with other agents (and possibly humans) via some kind of agent-communication language;
- *reactivity*: Agents perceive their environment, (which may be the physical world, a user via a graphical user interface, a collection of other agents, the internet, or perhaps all of these combined), and respond in a timely fashion to changes that occur in it;
- *pro-activeness*: Agents do not simply act in response to their environment , they are able to exhibit goal-directed behavior by taking the initiative.

And different characteristics are required in different application areas [51]. In this thesis, a common agent is defined as an entity which has characteristics of *real-time*, *intending*, *continuous* and *resource limited* .

Though MAS have been widely accepted and implemented in many areas recent years, some researchers still argue about distinction between *agents* and *Objects*. We distinguish the algorithm presented in this thesis from *Objects* (especially *active Objects*) according to M. Wooldridge's advice [51]:

- *agents* embody a stronger notion of autonomy than *objects*;
- *agents* are capable of flexible (reactive, proactive, social) behavior;
- a *multi-agent system* is inherently multi-threaded.

2.4.2 Coordinative multi-agent system

Coordination is the art of managing interactions and dependencies among activities, that is, among agents, in the context of MASs [52, 53]. A coordination model can be thought as consisting of three elements: *coordinables*, *coordination media* and *coordination laws*. H. S. Nwana (1996) pointed out [54] that the hypothesis, rationale, or goal for having collaborative agent systems is a specification of the goal of DAI (Distributed AI) as noted

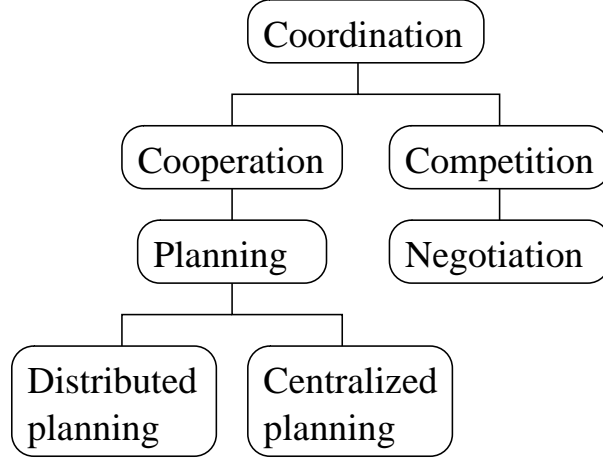


Figure 2-1: A taxonomy of agent interaction behavior

by M. N. Huhns and M. P. Singh (1994) [55]. Paraphrasing these authors, it may be stated as ‘creating a system that interconnects separately developed collaborative agents, thus enabling the ensemble to function beyond the capabilities of any of its members’. Formally,

$$V\left(\sum \text{agent}_i\right) > \max\left(\sum V(\text{agent}_i)\right)$$

where function V represents ‘value addedness’. This could have an arbitrary definition involving attributes such as speed, worst-case performance, reliability, adaptability, accuracy, etc., or some combination of these.

Coordination is based on relations among agents. And there are a number of relations among agent activities, such as equality, enablement, inhibit, interference, etc. [56, 57], as shown in Figure 2-1 [58]. Consequently we could briefly give definitions of different type of coordinations as follow:

- Coordination: Coordination is a property of a system of agents performing some activity in a shared environment [52];
- Cooperation⁹: Cooperation is coordination among nonantagonistic agents [58];
- Negotiation: Negotiation is coordination among competitive or simply self-interested agents [58].

Then we can conclude the characteristics of coordinative agent, i.e. *autonomy*, *interaction*, *task* and *benevolence*:

⁹Because this thesis mainly focuses on distributed scheduling, so we use the notion *coordination* as *cooperation* in this thesis, under conditions without misunderstanding.

- *autonomy*: Agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state;
- *interaction*: Agents are capable to interact with each other;
- *task*: An agent is always aimed to some tasks;
- *benevolence*: When other agents need help, an agent would do its best to facilitate others.

In other words, coordinative agent is such a kind of autonomous software entity, which interacts with environment dynamically and cooperate with other agents benevolently in order to accomplish the task of itself. Therefore, coordinative MAS is a group of coordinative agents with proper tasks. Since coordinative MAS was proposed, it has been widely used in kinds of distributed industrial applications and other areas [11, 13, 14, 59, 60, 61, 62, 63, 64, 65, 66, 67].

In this thesis, on the historical mainstream of coordination in MAS, the former investigations are divided into four categories approximately, i.e. coordination protocol, coordination application system, strong mathematical coordination model and coordination extension:

1. Coordination protocol: According to D. Gelernter and N. Carriero (1992) [52] we know that coordination protocol is very important for successful and effective coordination. Main coordination protocols in MAS are CNET, KQML and FIPA ACL.
 - (a) CNET: Contract NET [68, 69] is one of the most famous and most effective coordination protocols, which coordinates agents via contracts among agents;
 - (b) KQML: Knowledge Query Manipulation Language [70, 71] is a famous knowledge representational standard in agent interaction. It is a structured language to represent and share knowledge among agents in order to attempt on each other's knowledge and goal stores;
 - (c) FIPA ACL: Agent Communication Language proposed by Foundation for Intelligent Physical Agents (FIPA) [72, 73] is a widely used language standard in agent interaction. ACL precisely and completely defined the communication grammar and implementation.

2. Coordination application system: There are many systems implemented with coordination or coordinative MASs, most of which are in organization or implied organization:
 - (a) FA/C system: Functionally accurate, cooperative system is a distributed processing system to handle distribution-caused uncertainty and errors as an integral part of the network problem-solving process [74, 75];
 - (b) HTN: Hierarchical Task Network, often gives deductive rules a procedural interpretation, such as SIPE (System for Interactive Planning and Execution) [76];
 - (c) PGP: Partial Global Planning, provides a framework to coordinating multiple AI systems that cooperating in a distributed sensor network, by combining a variety of coordination techniques into a single, unified framework [77];
 - (d) ARCHON system: ARchitecture for Cooperative Heterogeneous Online system, this project was to build a software architecture that would allow pre-existing expert systems dealing with different aspects of decision making of a given complex environment or a system to cooperate in a mutually beneficial way [78, 79];
 - (e) Dis-DSS: Distributed Dynamic Scheduling System is a MAS to coordinate different resources among some airlines and airport, as well as a part of the *Airline Resource Management* (ARM) system [11, 13];
 - (f) GPGP: Generalized Partial Global Planning is an extendable family of PGP based on TÆMS (Task Analysis, Environment Modeling, and Simulation). In comparison to PGP, GPGP schedules tasks with deadlines, it allows agent heterogeneity, it exchanges less global information, and it communicates at multiple levels of abstraction [80];
 - (g) Dis-HTN: Distributed HTN is a distributed version of HTN, such as DSIPE (Distributed System for Interactive Planning and Execution). DSIPE firstly is based on constraints, secondly employs distributed agents to auto-identify and share information, finally generates global planning from partial plannings [81];
 - (h) Resource constraint GPGP: K. Decker and J. J. Li extended a task structure representation language TÆMS [82, 83] to have the capacity of representing resource constraints, then extended GPGP to resource constraint GPGP

to process a natural resource constraint distributed problem Hospital Patient Scheduling (HPS) [84];

- (i) Wasp-like: wasp-like is a multi-agent coordination model, which was based on the natural MAS of the wasp colony, an algorithm similar to ACO but they are different [64, 85].
3. Strong mathematical coordination model: These models are called “strong” in this thesis, because the models usually need machines much more superior than Turing Machine (TM) and current computer to completely solve problems in reasonable time. M. d’Inverno (1997) argues that these mathematical approaches mainly focus on game theory and modal or temporal logic [86]:
- (a) Joint Intention: Joint Intention is a coordinative function to coordinate agents to make joint commitment, to have joint responsibility and to perform joint action [87, 88];
 - (b) BOID: Beliefs-Obligations-Intentions-Desires architecture was proposed [89] by J. Broersen, M. Dastani, J. Hulstijn, Z. Huang and L. van der Torre to extend traditional BDI (Belief-Desire-Intention) architecture [51, 90] with more social behavior formation;
 - (c) Coo-BDI: Cooperative BDI is based on the dMARS (Distributed Multi-Agent Reasoning System) specification [86] and extends the traditional BDI architecture in many respects, such as separation of external events and main desires, introduction of *cooperations* among agents, the introduction of default plans, etc. [91, 92];
 - (d) RCS: Rolegraph Coordination Strategy is proposed for agent teamwork, which can operate with only partial information available to each agent at runtime [93]. This is achieved using graph matching principles to interpret hierarchical role relationships that represent team intentions;
 - (e) FSTS: Fuzzy Subjective Task Structure is proposed to abstract the coordination problems with the essential notions such as methods, tasks, and method relations [94]. Fuzzy logic techniques [95] are explored in the model to capture the uncertainties within the descriptions of the task-oriented environment;
 - (f) MDP: Markov Decision Process is a decision model, which makes decisions by exploring possible world-states and the probability that an action performed in any world-state will lead to a transition to any other world-state, and then

they construct optimal policies, which prescribe the optimal action to perform in each world state. Some of the MDP researches in coordination area are:

- i. MMDP : Multiagent MDP [96],
 - ii. DEC-MDP and DEC-POMDP: Decentralized MDP and Decentralized partially observable Markov decision process (POMDP) [97, 98],
 - iii. COM-MTDP: COMMunicative Multiagent Team Decision Problem [99],
 - iv. CMDP and EMTDP: single-agent constrained MDP and Extended Multi-agent Team Decision Problem [100].
4. Coordination extension: Coordination extension focuses on frontier besides agent-agent coordinative relations:
- (a) P. Scerri, L. Johnson, D. V. Pynadath, P. Rosenbloom, N. Schurr, M. Si and M. Tambe (2003) discussed the coordination between human-agent and human-agent-human [101];
 - (b) A. Omicini, A. Ricci, M. Viroli, C. Castelfranchi, and L. Tummolini presented a novel concept *Coordination Artifacts*, which seems like visual, easy-to-use, industrial development oriented intelligent agents [102].

Having a brief review of coordination technique history, we could have a better understand of coordinative agents and DSAFO in Chapter 5.

In these progresses, there are some approaches which it is valuable to pay our attention to. One is GPGP method, which is a general method to solve scheduling problems, however it is a NEXP-complete algorithm [82, 98]. Additionally, K. Sycara, S. Roth, N. Sadeh and M. Fox (1991) described a mechanism for transmitting abstractions of resource requirements (textures) between agents [24]. Each agent uses these texture measures to form a model of the aggregate system demand for resources. This model is used to allocate resources using various heuristics. And A. Garland and R. Alterman (2004) argued that agents without autonomy could achieve better coordination via common knowledge than autonomous agents [103].

2.4.3 Agent organization paradigms

While there is no single definition of organizations that is uniformly agreed to, there are general tenets that are more or less shared. In general, organizations are characterized as [49]:

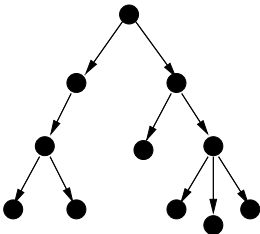
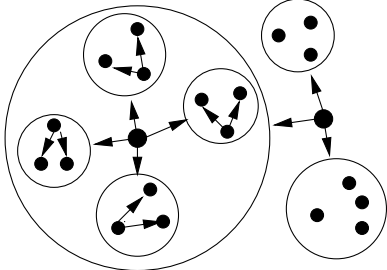
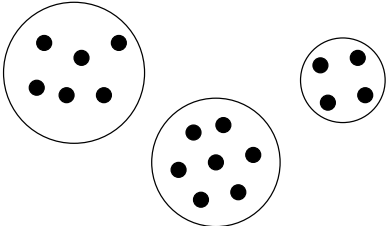
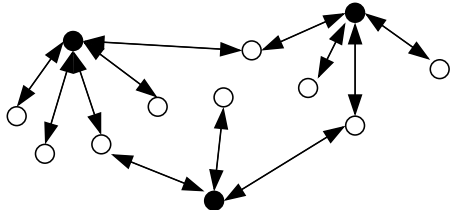
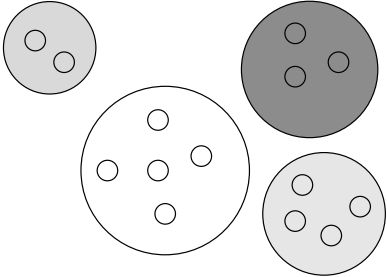
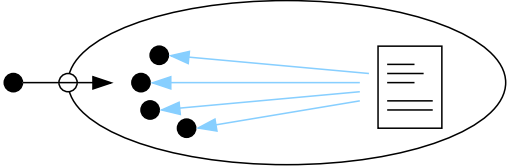
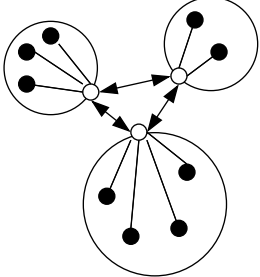
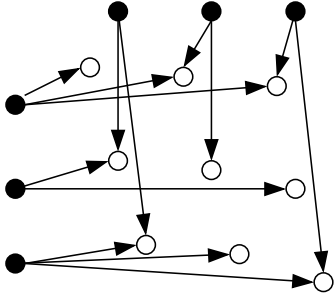
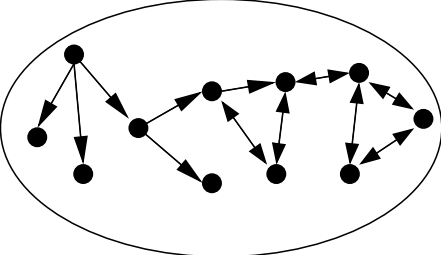
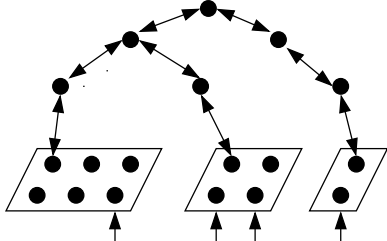
- large-scale problem solving technologies
- comprised of multiple agents (human, artificial, or both)
- engaged in one or more tasks; organizations are systems of activity
- goal directed (however, goals may change, may not be articulable or clear, and may not be shared by all organizational members)
- able to affect and be affected by their environment
- having knowledge, culture, memories, history, and capabilities distinct from any single agent
- having legal standing distinct from that of individual agents.

Since M. S. Fox (1979, 1981) presented organization in MAS firstly [104, 105], a number of organization models are studied and become maturer and maturer. Some of them are well formalized and possess the qualifications of stand-alone application [106, 107, 108, 109, 110]. As shown in Table 2–1, B. Horling (2004) gave a summary of ten MAS organizational paradigms [111]: Hierarchy, Holarchy, Coalition, Marketplace, Congregation, Society, Federation, Matrix, Team and Compound organization. Each of them has particular benefits and weaknesses:

1. **Hierarchy** [104, 105, 112, 113] arranges agents in a tree, which is perhaps the earliest example of structured. It is good at divide-and-conquer, mapping to many fields and useful in large scale problems. But it is not robust and causes bottle-neck effect and delay.
2. **Holarchy** [114, 115] is, through Koestler’s intent to *holon*¹⁰, a notion of a hierarchical, nested structure does accurately describing the organization of many systems. It makes fully use of individual autonomy. However it is impossible to model define problems in *Holon* and its performance is hard to predict.
3. **Coalition** [112, 116] has been studied by the game theory community for decades, and has proved to be a useful strategy in both real-world economic scenarios and MASs. Coalition exploits strength in agent numbers. However its short term benefits may not outweigh organization construction costs.

¹⁰The term holon was first coined by Arthur Koestler in his book *The Ghost In The Machine* (Koestler, 1967). In this work, Koestler attempts to present a unified, descriptive theory of physical systems based on the nested, self-similar organization that many such systems possess, such as astronomic galaxy.

Table 2-1: A taxonomy of MAS organizational paradigms

 <p>Hierarchy</p>	 <p>Holarchy</p>
 <p>Coalition</p>	 <p>Marketplace</p>
 <p>Congregation</p>	 <p>Society</p>
 <p>Federation</p>	 <p>Matrix</p>
 <p>Team</p>	 <p>Compound organization</p>

4. **Marketplace** [117, 118] is a market-based organization. Buying agents (shown in white) may request or place bids for a common set of items, such as shared resources, tasks, services, or goods. Markets excel at the processes of allocation and pricing. However it might cause malevolent competition and it is a complex allocation process.
5. **Congregation** [119, 120] are groups of individuals who have banded together into a typically flat organization in order to derive additional benefits. Unlike these other paradigms, congregations are assumed to be long-lived and yet not formed with a single specific goal in mind. This long-lived Congregation facilitates agent discovery. However the presumed sets may be over restrictive.
6. **Society** [121, 122] intuitively brings to mind a long-lived, social construct, drawing from our own experiences with biological societies. Unlike some other organizational paradigms, agent societies are inherently open systems. Society provides good public services and has well defined conventions. However it is potentially complex, and agents may require additional society-related capabilities.
7. **Federation** [123, 124] comes in many different varieties. All share the common characteristic of a group of agents which have ceded some amount of autonomy to a single delegate which represents the group. Federation is good at matchmaking, brokering, translation services and it facilitates dynamic agent pool. However the intermediaries become bottlenecks.
8. **Matrix** [80] relaxes the one-agent, one-manager restriction, by permitting many managers or peers to influence the activities of an agent. In this way, the agent's capabilities may be shared, and the agent's behaviors (hopefully) influenced so as to benefit all. However the shared agent becomes a potential point of contention and the agent might become very complex.
9. **Team** [105, 124, 125, 126] consists of a number of cooperative agents which have agreed to work together toward a common goal. Team is addressed to larger grained problems and it is task-centric. However its communication increases very quickly when the problem scale grows.
10. **Compound organization** [77] allow system to include characteristics of several different styles. A system may have one organization for control, another for data flow, a third for discovery, and so on. The positive and negative characteristics of a compound organization are derived primarily from its constituent parts.

2.5 Polyadic π -calculus

The polyadic π -calculus developed by R. Milner (1991) [127] is a very powerful tool to model parallel systems such as mobile systems, it is based on π -calculus [128, 129]. And because of the parallel nature of MAS [51], the polyadic π -calculus was naturally introduced into MAS formalization respectively by T. Rorie (1998) [130] and W. Jiao and company (1999) [131, 132]. Nextly we will have a brief review of definitions of π -calculus and polyadic π -calculus.

Definition 4 (π -calculus [127]). *The most primitive entity in π -calculus is a name. Names, infinitely many, are $x, y, \dots \in \mathcal{X}$; they have no structure. In the basic version of π -calculus which we begin with, there is only one other kind of entity; a process. Processes are $P, Q, \dots \in \mathcal{P}$ and are built from names by this syntax*

$$P ::= \sum_{i \in I} \pi_i.P_i \mid P|Q \mid !P \mid (\nu x)P$$

Here I is a finite indexing set; in the case $I = \emptyset$, we write the sum as $\mathbf{0}$. In a summand $\pi.P$ the prefix π represents an atomic action, the first action performed by $\pi.P$. There are two basic forms of prefix:

$$\begin{aligned} x(y), & \text{ which binds } y \text{ in the prefixed process, means} \\ & \text{“input some name — call it } y \text{ — along the link named } x\text{”}; \\ \bar{x}y, & \text{ which does not bind } y, \text{ means “output the name } y \text{ along} \\ & \text{the link named } x\text{”}. \end{aligned}$$

In each case we call x the subject and y the object of the action; the subject is positive for input, negative for output.

A name refers to a link or a channel. It can sometimes be thought of as naming a process at “the other end” of a channel; there is a polarity of names, and \bar{x} — the *co-name* of x — is used for output, while x itself is used for input.

$P|Q$ (“ P par Q ”) simply means that P and Q are concurrently active, so they can act independently. $!P$ (“bang P ”) means $P|P|\dots$; as many copies as you wish. And $(\nu x)P$ (“new x in P ”) restricts the use of the name x to P . Finally, Processes like $x(y).\mathbf{0}$ and $\bar{x}y.\mathbf{0}$ are so common that we prefer to omit the trailing “ $\mathbf{0}$ ” and write, respectively, just $x(y)$ and $\bar{x}y$.

And polyadic input and polyadic output are introduced from abbreviations:

$$\begin{array}{ll} x(y_1 \cdots y_n) & \text{for } x(w).w(y_1) \cdots w(y_n) \\ \bar{x}y_1 \cdots y_n & \text{for } (\nu w)\bar{x}w.\bar{w}y_1 \cdots \bar{w}y_n \end{array}$$

With these abbreviations, we could go on with definition of polyadic π -calculus.

Definition 5 (polyadic π -calculus [127]). *From the monadic to the polyadic calculus, we add the forms for abstractions F, G, \dots and concretions C, D, \dots , calling them collectively agents A, B, \dots . We use α, β, \dots to range over names and co-names, and \vec{x}, \vec{y}, \dots to stand for vectors of names, with length $|\vec{x}|, |\vec{y}|, \dots$*

$$\begin{array}{ll} \text{Normal processes} & : N ::= \alpha.A \mid \mathbf{0} \mid M + N \\ \text{Processes} & : P ::= N \mid P|Q \mid !P \mid (\nu x)P \\ \text{Abstractions} & : F ::= P \mid (\lambda x)F \mid (\nu x)F \\ \text{Concretions} & : C ::= P \mid [x]C \mid (\nu x)C \\ \text{Agents} & : A ::= F \mid C \end{array}$$

$M + N$ means that either M or N is randomly selected and run, one and only one process could be executed. $(\lambda x)P$ is a *abstraction*, which denotes the essence of parametric definition; i.e.

$$x(y_1 \cdots y_n).P \stackrel{\text{def}}{=} x.(\lambda y_1 \cdots y_n)P$$

And $[x]C$ is a *concretion*, “[]” is derived from output prefix form:

$$\bar{x}y_1 \cdots y_n.P \stackrel{\text{def}}{=} \bar{x}.[y_1 \cdots y_n]P$$

Additionally, we usually, for reading convenience, write prefixed form $\bar{x}y_1 \cdots y_n$ as $\bar{x}\langle y_1 \cdots y_n \rangle$, by adding a pair of brackets.

Chapter 3 AGSS: Formulation and Characteristics

In this chapter, we list assumptions for AGSS formulation and modeling, and formalize AGSS problem in constraint satisfaction view, and discuss AGSS characteristics and adaptation to practical programming.

3.1 Assumptions

In real-life AGSS problem, there are many indeterministic interferences. But in this thesis, some assumptions are made, in order to establish a computable model. The assumptions are listed as follow:

- traffic condition in airport is optimal (no block);
- resources and their services are optimal (no breakdown);
- relative flight data needed could always be accessed;
- operations could be near precisely estimated according to the flight data above;
- there are no differences in same type of resources;
- each operation needs only one resource to fulfill;
- each resource can be possessed by only one operation at a time;
- each resource serves no more than three 4-hour jobs per day;
- each workforce has two 4-hour jobs per day. And job could be assigned to any time;
- message transmission between agents is costly, but messages arrive in sequence.

3.2 Formulation

Informally, the AGSS problem can be stated as follows. There are a set of flights \mathcal{F} and a set of resources \mathcal{R} . Each flight has a fixed deadline (Takeoff) and consists of a set of operations that must be processed in a given order. Each operation is given an integer service time, and a longer resource usage time (plan time) including traffic and preparation. Each operation needs one resource to process, and the processing is uninterruptible. Each resource can process only one operations simultaneously. A valid schedule assigns each operation with a resource and a service start time in order to meet all the deadlines of flights. And AGSS problem is the process to find the schedule with minimal resource consumption.

Nextly we would give some formal functions to represent the processes which assign AGSS resources with service operations. As a result, the AGSS related concepts could be formally described in constraint satisfaction view.

3.2.1 Airport ground service

Lemma 1. *The service operations (including landing and takeoff) for one flight (\mathcal{O}) and their temporal “precedence or equality” (or no later than) relation (\preceq) forms a lattice $\langle \mathcal{O}, \preceq \rangle$.*

Proof. A lattice is a poset (partial ordered set) $\langle S, R \rangle$ in which each two-element subset $\{a, b\}$ has an infimum, denoted $\inf\{a, b\}$, and a supremum, denoted $\sup\{a, b\}$ [133].

Firstly, $\langle \mathcal{O}, \preceq \rangle$ is a poset, because:

- closure: for each $\langle o_1, o_2 \rangle \in \preceq$, $o_1, o_2 \in \mathcal{O}$. It stands for that if two operations have “precedence or equality” relation, they must be in a set of service operations for one flight;
- reflexivity: for each $o \in \mathcal{O}$, $o \preceq o$, which shows each operation is no later than itself;
- transitivity: for each $o_1, o_2, o_3 \in \mathcal{O}$, if $o_1 \preceq o_2$ and $o_2 \preceq o_3$, then $o_1 \preceq o_3$, which represents if o_1 is no later than o_2 and o_2 is no later than o_3 , then o_1 is no later than o_3 .

Secondly, there do exists a supremum and an infimum in $\langle \mathcal{O}, \preceq \rangle$:

- supremum: for each $o \in \mathcal{O}$, $landing \preceq o$, i.e., all operations for one flight can not begin before the flight *landing*;
- infimum: for each $o \in \mathcal{O}$, $o \preceq takeoff$, that is to say, operations for one flight are no later than the flight *takeoff*.

Hence, what the service operations for one flight and their “precedence or equality” relation forms, $\langle \mathcal{O}, \preceq \rangle$, is a lattice. \square

Definition 6 (Airport ground service). *An instance of airport ground service 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{F}	$= \{\varphi_1, \varphi_2, \dots, \varphi_n\}$	- a set of n flights;
\mathcal{O}	$= \{o_1, o_2, \dots, o_p\}$	- a set of p operations;
\mathcal{R}	$= \{r_1, r_2, \dots, r_q\}$	- a set of q resources;
\mathcal{T}	$= \{0, 1, 2, \dots, \Delta\}$	- a set of discrete integral time, with typical algebraic operators such as $+$, $-$, $=$, $<$, \leq , etc;
\preceq	$: \mathcal{O} \mapsto \mathcal{O}$	- precedence or equality, decomposing \mathcal{O} into lattices of flights, according to Lemma 1;
D	$: \mathcal{F} \mapsto \mathcal{T}$	- deadline of a flight;
F	$: \mathcal{O} \mapsto \mathcal{F}$	- flight belonging to;
rt	$: \mathcal{O} \mapsto \mathcal{T}$	- operation ready time;
st	$: \mathcal{O} \mapsto \mathcal{T} - \{0\}$	- non-zero operation service time;
ut	$: \mathcal{O} \mapsto \mathcal{T}$	- operation setup time;
et	$: \mathcal{O} \mapsto \mathcal{T}$	- operation reset time;
tt	$: \mathcal{R} \times \mathcal{O} \mapsto \mathcal{T}$	- resource traffic time for an operation;
Ω	$: \mathcal{R} \times \mathcal{T} \mapsto \mathcal{O} \cup \{\emptyset\}$	- which operation is in process at a certain time, returns \emptyset when non-single operations assigned.

3.2.2 AGSS satisfaction

Definition 7 (AGSS satisfaction). *An AGSS satisfaction is a process to find two functions:*

- $\gamma : \mathcal{O} \mapsto \mathcal{R}$ - assign resources;
- $s : \mathcal{O} \mapsto \mathcal{T}$ - assign service starts,

for an instance of airport ground service $\langle \mathcal{F}, \mathcal{O}, \mathcal{R}, \mathcal{T}, \preceq, D, F, rt, st, ut, et, tt, \Omega, \gamma, s \rangle$,

$$\begin{aligned} \text{s.t. } & \forall_{o \in \mathcal{O}} [rt(o) \leq s(o) \\ & \wedge s(o) + st(o) \leq D(F(o)) \\ & \wedge \forall_{o \prec o'} s(o) + st(o) \leq rt(o') \\ & \wedge \forall_{s(o) - ut(o) - tt(\gamma(o), o) \leq \tau < s(o) + st(o) + et(o)} \Omega(\gamma(o), \tau) = o], \end{aligned}$$

where $o \prec o'$ is abbreviated from $o \preceq o' \wedge o \neq o'$.

In other words, AGSS satisfaction is the process to find a valid scheduling within limited resources to fulfill service operations timely.

Lemma 2. $p \sim O(n)$, i.e. the number of operations p is bounded at $O(n)$.

Proof. For each flight there are a limited number of operations to fulfill, i.e. $\forall \varphi \in \mathcal{F}$,

$$OP_\varphi = \{o \mid o \in \mathcal{O} \wedge F(o) = \varphi\}$$

denotes the set of operations belonging to φ . And such a set must be non-empty and with a limited number of upper bound, so it is clear that $\forall \varphi \in \mathcal{F}, |OP_\varphi| = O(1)$. Hence,

$$p = |\mathcal{O}| = \sum_{\varphi \in \mathcal{F}} |OP_\varphi| \sim O(n).$$

□

Lemma 3. $q \sim O(n)$, i.e. the number of resources q is bounded at $O(n)$.

Proof. Through all operations' resource usage demand, there must be a global minimal resource usage time:

$$\text{rut}_{\min} \stackrel{\text{def}}{=} \min_{o \in \mathcal{O}} [tt(\gamma(o), o) + ut(o) + st(o) + et(o)]$$

And if one resource serve only one operation totally, the maximum number of resources is p . To arrange a number p of resource usages in a period $\Delta + 1$ (the size of \mathcal{T}), the minimal number of resources must be no less than $\frac{p \times \text{rut}_{\min}}{\Delta + 1}$. Hence,

$$O(n) \sim \frac{p \times \text{rut}_{\min}}{\Delta + 1} \leq q \leq p \sim O(n)$$

Hence, $q \sim O(n)$.

□

Theorem 1. AGSS satisfaction is in \mathcal{NP} -complete class.

Proof. Firstly, any AGSS satisfaction, obviously, could be verified in polynomial time, so AGSS satisfaction must be in \mathcal{NP} .

In the following proof section, we will, step by step, derive AGSS satisfaction from Partition, which is one of the well known basic \mathcal{NP} -complete problems [1].

An instance of Partition problem consists of a finite set A and a “size” $s(a) \in \mathbb{Z}^+$ for each $a \in A$. The Partition problem is to find a subset $A' \subseteq A$ such that

$$\sum_{a \in A'} s(a) = \sum_{a \in A - A'} s(a).$$

Then, a problem named *Multi-processor scheduling* is discussed. An instance of this problem consists of a finite set A of “tasks”, a “length” $l(a) \in \mathbb{Z}^+$ for each $a \in A$, a number of m “processors”, and a “deadline” $D \in \mathbb{Z}^+$. The problem is to find a partition $A = A_1 \cup A_2 \cup \dots \cup A_m$ of A into

$$\max \left\{ \sum_{a \in A_i} l(a) : 1 \leq i \leq m \right\} \leq D.$$

Consider such a restriction of *Multi-processor scheduling* problem: allowing only instances in which $m = 2$ and $D = \frac{1}{2} \sum_{a \in A} l(a)$. We can clearly see that this problem could be restricted to a Partition problem, exactly. That is to say, one Partition problem could be mapped to one *Multi-processor scheduling* problem, where $m = 2$ and $D = \frac{1}{2} \sum_{a \in A} l(a)$. Hence,

$$\text{Partition} \leq_p \text{Multi-processor scheduling}$$

Therefore *Multi-processor scheduling* is, generally, a \mathcal{NP} -complete problem. Yet its introduction and another similar proof was done by M. R. Garey, E. G. Coffman Jr., and D. S. Johnson (1978, 1979) [134, 1].

Nextly, consider such a restriction of AGSS satisfaction: allowing only instances in which $ut(o) = et(o) = tt(o) = 0$ for each operation $o \in \mathcal{O}$, $rt(\text{landing}) = 0$, and $D(f) = D_F$ for each flight $f \in \mathcal{F}$. This restriction means an amount of special cases (temporarily leave the feasibility aside) of AGSS satisfaction: no variable setup time, reset time, and traffic time for operations, as well as a uniform landing and takeoff time for all flights.

Then, every instance of *Multi-processor scheduling* problem, according to this restriction, could have a homologous AGSS satisfaction, under $q = m$, $p = |A|$, $D_F = D$, and one operation $o \in \mathcal{O}$ denotes one task $a \in A$. Hence,

$$\text{Multi-processor scheduling} \leq_p \text{AGSS satisfaction}.$$

Therefore,

$$\text{Partition} \leq_p \text{AGSS satisfaction}.$$

Hence, AGSS satisfaction is in \mathcal{NP} -complete class. \square

Fortunately, this kind of resources are usually sufficient in real-life airports and airlines in China, so it is not difficult to find a valid schedule in AGSS. However, how many resources and man-days consumed are important to airports and airlines. So we need an optimization on valid AGSS solutions.

Proposition 1. *The AGSS satisfaction is a special case of Distributed Constraint Satisfaction Problem (Dis-CSP).*

Proof. (Sketch) According to Definition 3, when considering each resource as an agent, all operations what one agent could do as its variables, and precedent relation, ready time and deadline to be constraints. Hence AGSS satisfaction could be formalized in Distributed Constraint Satisfaction Problem (Dis-CSP). \square

3.2.3 AGSS problem

Definition 8 (AGSS problem). *An AGSS problem is to find a valid AGSS satisfaction schedule γ_0 , s_0 , so that*

$$\text{s.t. } |\text{ran}(\gamma_0)| = \min_{\gamma \in \Gamma} |\text{ran}(\gamma)|,$$

where Γ is the set of all valid schedules, and $|\text{ran}(\gamma)|$ is the range size of function γ .

That is to say, AGSS problem is the process to find the optimal scheduling (lowest cost).

Theorem 2. *AGSS problem is in \mathcal{NP} -complete class.*

Proof. From Definition 7, Definition 8 and Theorem 1, we can clearly reason out that AGSS problem is no simpler than \mathcal{NP} -complete class. Nextly we should bound AGSS problem up to \mathcal{NP} -complete class.

And let us consider a most tightly resource bounded situation: the resources are so few that there could be only one possible schedule for the AGSS satisfaction. In this extreme case, the AGSS problem solving process is the very one AGSS satisfaction process, i.e. this AGSS problem is \mathcal{NP} -complete. In this process, all possible value assertions of binary set B are possibly attempted, hence this brute-force search is \mathcal{NP} -complete.

Generally, AGSS problem solving can be transformed to two processes: doing such a brute-force search and scheduling resource consumption comparisons in case of finding a new AGSS satisfaction. Though comparison computation varies large in different cases, number of valid solutions is usually $O(2^{|B|}) = O(2^b)$, hence it must be in \mathcal{NP} -complete time.

Hence, by summing up these two processes, the AGSS problem is in \mathcal{NP} -complete class. \square

Proposition 2. *The AGSS problem is a special case of Distributed Constraint Satisfaction Problem (Dis-CSP).*

Proof. (Sketch) According to Definition 8, AGSS problem is a special case of AGSS satisfaction, however with more constraints. And AGSS satisfaction is a special case of Dis-CSP (Proposition 1). Hence the AGSS problem is a special case of Dis-CSP. \square

Proposition 3. *The AGSS problem is a special case of Job-Shop Scheduling Problem.*

Proof. (Sketch) According to Definition 1 and Definition 8, the precedent relation stands for sequencing requirement, and excessive possession of resources stands for equipment interference problem. Hence AGSS problem is a special case of JSSP. \square

3.2.4 Uncertain AGSS satisfaction

Definition 9 (Uncertain AGSS satisfaction). *An uncertain AGSS satisfaction is a special case of AGSS satisfaction, where for each operation $o \in \mathcal{O}$, the ready time $tr(o)$ is a probabilistic function.*

For example, it is an uncertain AGSS satisfaction to generate an always valid AGSS satisfaction schedule for next month flight services within considering every combination of flight arrivals and departures which might be delayed by kinds of reasons.

Proposition 4. *The uncertain AGSS problem is a special case of Dynamic Job-Shop Scheduling Problem (DJSSP).*

Proof. (Sketch) Similar to Theorem 3, uncertain AGSS problem is a special case of Job-shop scheduling problem (JSSP), however, with uncertain ready time of operations. That is what exactly is called *Dynamic Job-shop scheduling problem* (DJSSP) [135]. \square

There is another complexity class PSPACE-complete, which means a class of decision problems solvable by a *Turing machine* (TM) in polynomial space [1]. As is well-known, PSPACE-complete is more difficult than \mathcal{NP} -complete.

Theorem 3. *Uncertain AGSS satisfaction is in PSPACE-complete class.*

Proof. (sketch) According to Proposition 4, uncertain AGSS satisfaction is a special case of DJSSP. DJSSP can be reduced in polynomial time to a problem of QBF (Quantified Boolean Formula) [135]. Furthermore, QBF is well-known in PSPACE-complete [1]. Hence uncertain AGSS satisfaction must be in PSPACE-complete. \square

3.2.5 Uncertain AGSS problem

Definition 10 (Uncertain AGSS problem). *An uncertain AGSS problem to find a valid uncertain AGSS satisfaction schedule γ_0, s_0 ,*

$$\text{s.t. } |ran(\gamma_0)| = \min_{\gamma \in \Gamma} |ran(\gamma)|,$$

where Γ is the set of all valid schedules.

In other words, uncertain AGSS problem is the process to find the optimal scheduling (lowest cost) for all of uncertain situations.

3.3 Notes in practical AGSS programming

Practically in programming for AGSS, we abbreviate $\mathcal{R} \times \mathcal{T}$ to \mathcal{P} to represent service plans, then a schedule becomes $sch : \mathcal{O} \mapsto \mathcal{P}$. And an operation is a tuple consisting of a flight number, type of operation, place to park, ready time (RT), latest finish time (LFT), expected service time (ST), expected setup time (UT), expected reset time (ET).

$$\text{operation} \stackrel{\text{def}}{=} \langle \text{fno}, \text{type}, \text{parkNo}, \text{RT}, \text{LFT}, \text{ST}, \text{UT}, \text{ET} \rangle$$

And a plan is usually a tuple consisting of a operation to serve, a resource to use, traffic time (TT), committed service start time(CSST). Plans are usually in multi-step.

$$\text{plan} \stackrel{\text{def}}{=} \langle \text{op}, \text{res}, \text{TT}, \text{CSST} \rangle$$

There is no ready-to-wear TT ($tt(\gamma, o)$) for a plan, one agent should calculate the ground traffic distance according to the location of prior operation and the location of current

service target. In some international airports, there would be more than one way to drive to destination, so agent should have simple spatial reasoning ability to calculate the shortest (or the most reasonable) path. And then the ground traffic distance should be divided by 5km/h, which is the maximal ground traffic velocity allowed in airport, to make proper ground traffic time.

3.4 Characteristics

AGSS satisfaction and AGSS problem is a special case of Dis-CSP according to Proposition 1 and Proposition 2. However it is nor a pure and simple Dis-CSP. Because an AGSS satisfaction solution usually is not sufficient for airline industry, a schedule with least resources, i.e. with lowest cost, is needed.

Though AGSS problem is a special case of JSSP, according to Proposition 3. However AGSS problem is nor a pure and simple JSSP. AGSS problem, itself, is not aimed to minimize the makespan of flights, but aimed to find a valid schedule for all flights and their operations, and secondly to find a valid schedule with least resource consumption. And the service finish of some operation does not means the resource is free then, e.g., in a typical service operation of *unload baggage* there might be four actions for a baggage tractor:

- to drive to belt conveyor aside of target flight;
- to load baggage from cargo cabin via belt conveyor;
- to drive to service point of passenger baggage claim belt;
- to dispatch baggage.

Chapter 4 Dynamic Distributed Scheduling Modeling

In this chapter, a dynamic distributed scheduling is offered for the dynamic nature of AGSS problem. And an dynamic scheduling run-time (environment) *run-and-schedule* for AGSS problem is also described.

4.1 Motivation

One major problems in AGSS problem is the dynamic nature of flight schedules. One flight might be delayed by kinds of reasons, and special service might appear as a result of special plane, passenger or weather. Therefore, it is incompatible to schedule an AGSS or uncertain AGSS problem with a fixed flight schedule forecast. So a dynamic scheduling framework is very essential for AGSS problem.

The other major problem in AGSS problem is variations in aircraft service requirements, i.e., a resource or an operation usually has individual requirements or constraints. For example, dispatching aviation ground support vehicles must fulfill individual operation requirement, spatially distributed fleet of vehicles, vehicular operation preparation, and (in some cases) ground traffic jam which is possibly blocked by aircrafts. As a result, many constraints are taken to centralized optimization models. The quantities of constraints make the model very difficult (usually \mathcal{NP} -hard or harder). Hence a distributed scheduling model is also very important for AGSS problem.

M. E. Aydin and E. Öztemel (2000) had employed a dynamic environment for a dynamic scheduling system with learning agents [136]. They dispose uncertain interference with real-time (dynamic) information gathering mechanism. Distributed models and methods were also naturally proposed to handle individual constraints in local resource/-operation scopes, such as Dis-CSP [42], *market-oriented programming* [137], analysis by G. İnalhan, D. M. Stipanović, and C. J. Tomlin (2002) [138], etc. Thereafter, distributed

computation makes optimization solving more feasible. And G. İnalhan *et al* (2002) pointed out that centralized *Pareto-optimality* can also be guaranteed in decentralized methods, if there is dominating local convexity in the solution or the interconnections are weak [138]. Fortunately interconnections and global constraints in AGSS problems are not strong, according to Definition 8.

In a word, it is beneficial to study dynamic distributed scheduling modeling for AGSS problem.

4.2 Model overview

As shown in Figure 4-1, we developed such a dynamic distributed scheduling model, which consists of four modules:

Input aviation systems: the collection of those aviation support information systems related to AGSS in airports and airlines. These systems could support AGSS problem optimization or solving timely and accurately with their data, such as flight schedule time (ready time and deadline), flight distance (cleaning and refueling), passenger flow (disembark and boarding), cargo inbound and outbound (cargo handling), etc. without these data, an accurate and effective service could not be scheduled. However, these input system are physically distributed and data from different systems may be not the same. So a data integration level is needed;

Run-and-schedule environment: the run-time for scheduling algorithm [139]. This environment collects and integrates flight data, as well as a clock, for the next module (distributed scheduling algorithm). Run-and-schedule is detailed in the following Section;

Distributed scheduling algorithm: the algorithm to optimize scheduling. Many algorithms could be adapted here, and in thesis we develop a novel algorithm DSAFO here, see more details in Chapter 5;

Real world resources: the aircraft service workforce and equipments, such as engineers, aviation ground support facilities and equipments, material, transportation equipments, etc. Real-time information and operation assignments about real world resources are also exchanges between distributed scheduling algorithm and real world entities simultaneously. This module is the final stage, where optimized scheduling is performed to as real manufacturing.

4.3 Run-and-scheduling

Run-and-schedule is a dynamic distributed scheduling environment [139], which uninterruptedly observes flights arriving and departing in half an hour. It collects flight data useful for AGSS from some aviation support systems, such as *Flight Operations Control system* (FOC), *Airport Operation Data Base* (AODB), *Flight Information Display System* (FIDS), *Billing system* and other operational and management information systems. The collected data includes VIP mark, length of flight course, baggage to load and to unload, cargo and mail information, passengers, special service requirement, etc. After successful collecting the data, *run-and-schedule* integrates the data into consistent and reliable formats and values.

As a result, service goals and plans in the distributed scheduling algorithm can be calculated rationally, dynamically and near precisely, according to these information.

Meanwhile, by unifying a Heartbeat period and an algorithm base time, it provides a public clock. This clock provides current time for all agents, flight tasks, service plans, and resources, in order to ensure operation assignment synchronization in distributed scheduling process.

According to viewpoints from S. J. Russell and P. Norvig [140], combining *run-and-schedule* and real-world resources forms an accessible, deterministic, dynamic and discrete runtime for multi-agent algorithm:

- accessible: an agent's sensory apparatus give it access to the complete state of *run-and-schedule* and resources;
- deterministic: the next state of resources is completely determined by the current state of resources and the actions selected by the agents;
- dynamic: what *run-and-schedule* provides can change while an agent is deliberating;
- discrete: there are limited number of distinct, clearly defined percepts and actions.

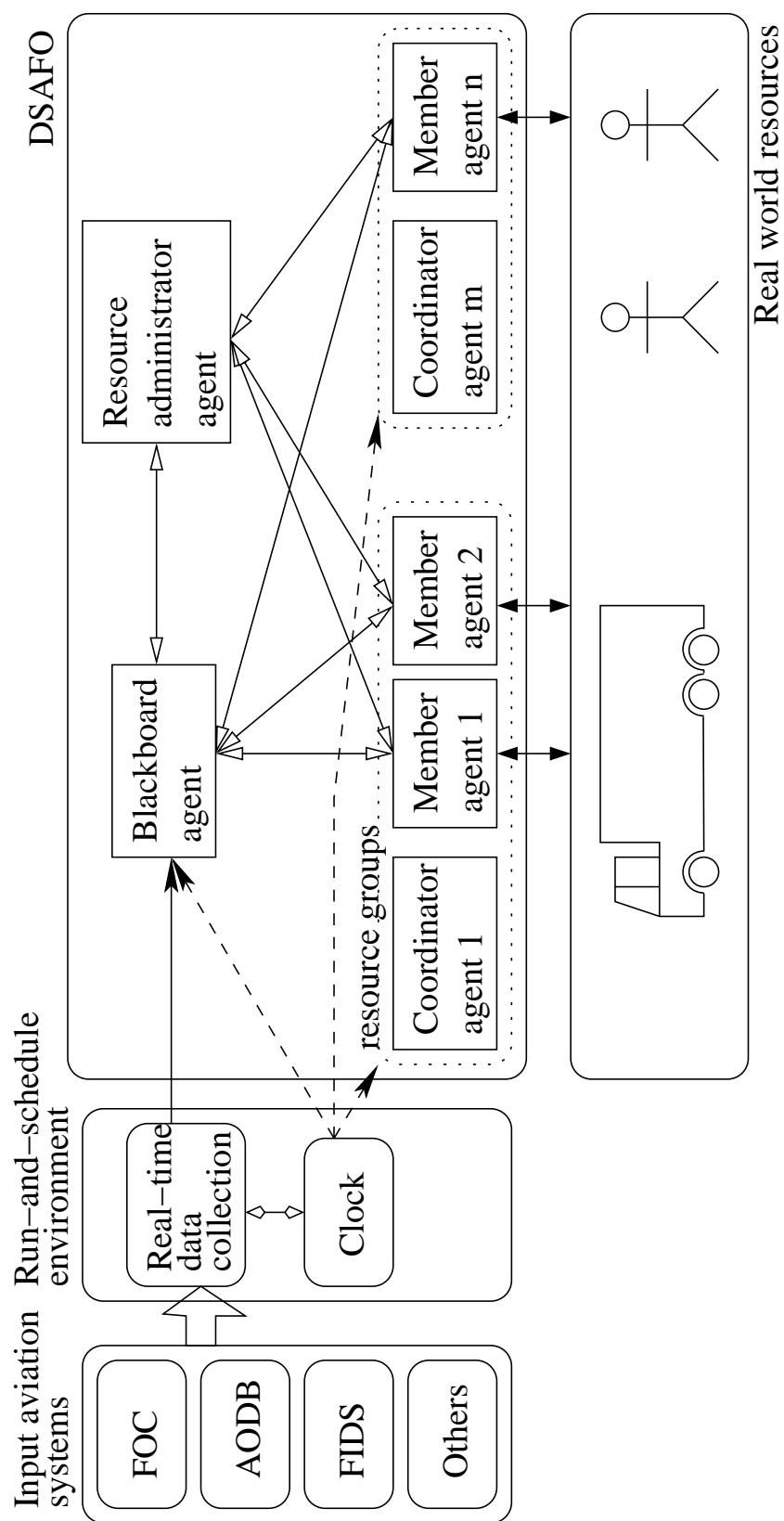


Figure 4-1: Dynamic distributed model for AGSS

Chapter 5 DSAFO: Overview, Design and Implementation

In this chapter, a multi-agent algorithm DSAFO (*Dynamic Scheduling Agents with Federation Organization*) is proposed to solve AGSS satisfaction and problem. Algorithm formalization and complexity analysis is presented as well.

5.1 An overview

DSAFO, i.e. Dynamic Scheduling Agents with Federation Organization, is a novel multi-agent approach to optimize AGSS satisfaction and problem [139]. Comparing to other scheduling techniques in Section 2.2, classification of DSAFO is shown in Figure 5–1.

As shown in Figure 4–1, DSAFO runs as follow:

1. to read real-time flights data from *run-and-schedule* environment;
2. to decompose flight service goals into operations, according to gathered data;
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.

Yet it could be observed from Figure 4–1 that there is one agent *Blackboard* in charge of decision-making of operation dispatch, one agent *ResourceAdmin* in charge of decision-making of resource allocation, and many *Member* agents in charge of making operation-

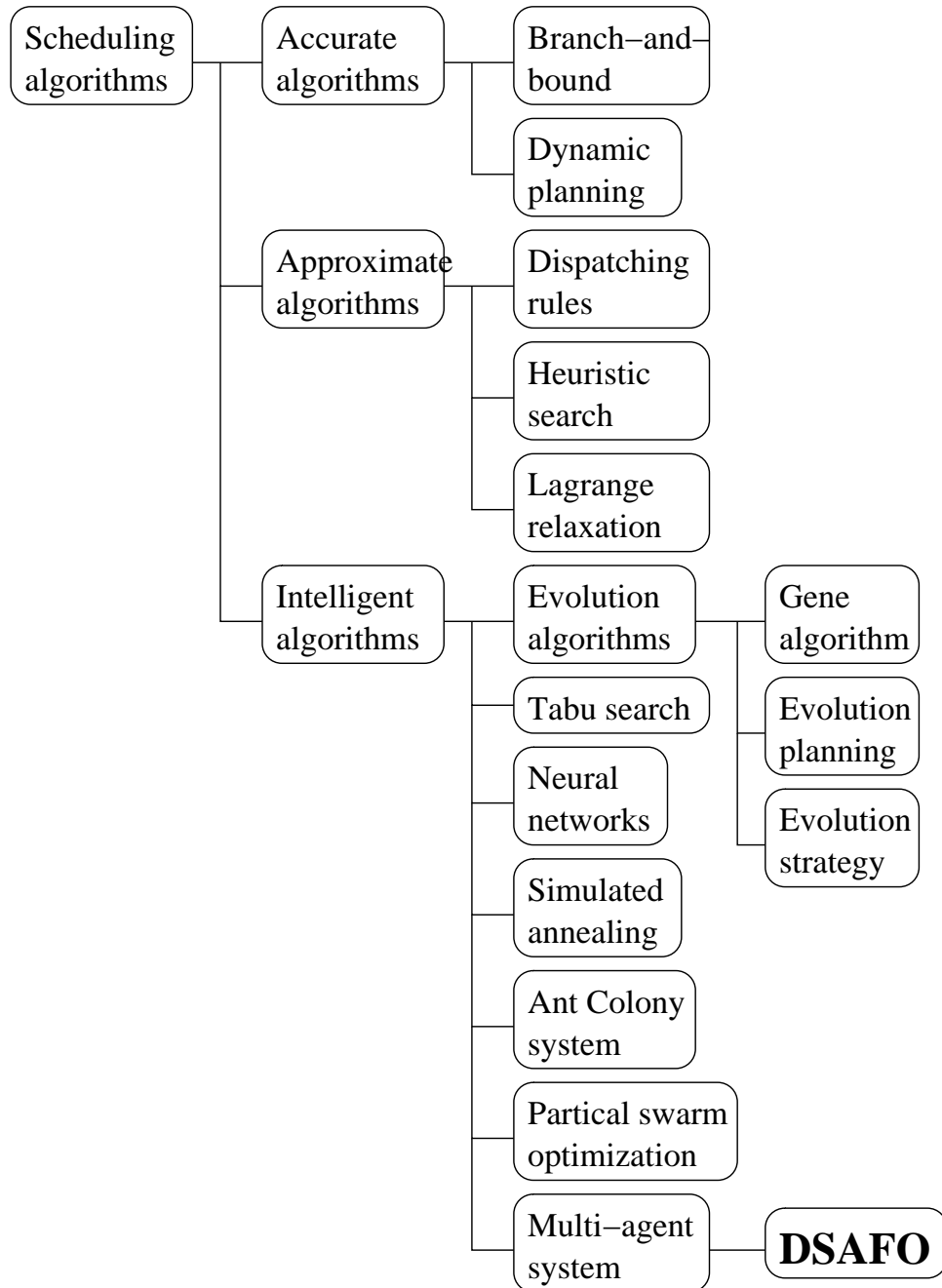


Figure 5-1: Algorithm category of DSAFO

resource match-up dynamically. The *Coordinator* agents are designed to facilitate cooperations among *Member* agents.

in addition, DSAFO shares many features with other forerunners:

1. similar spare resource factor synchronization mechanism with approach presented by K. Sycara, S. Roth, N. Sadeh, and M. Fox (1991) [24],
2. similar resource borrowing mechanism and blackboard system with Dis-DSS [11, 12],
3. similar dynamic environment with *simulated environment* presented by M. E. Aydin and E. Öztemel (2000) [136],
4. similar heuristic backtrack with *asynchronous weak-commitment search*[42, 44, 45].

5.2 DSAFO strategies

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

5.2.1 Local heuristics

There are two sorts of local heuristics in DSAFO, both are EDD. One is to select the first operation to do from many operations according to weighted Latest Finish Time, and the other is to assign an operation to a resource from some resources according to earliest Committed Service Start Time.

EDD is a sound and efficient way to solve small scale problems with suitable domain knowledge. It can quickly work out a solution for a \mathcal{NP} -complete or even PSPACE-complete problem in very low polynomial time. But when the scale increases, EDD and other heuristic algorithms will fall into local minimum. Consequently, DSAFO employs local heuristics (EDD) in dynamic agents to jump out of local minimum.

This EDD strategy is implemented in the *Blackboard* and *Member* roles in Subsect. 5.3.1 and 5.3.3.

5.2.2 Global coordination

Global coordination is a mechanism that exchanges resource textures of agents and realizes a complete resource borrow procedure for agents.

Usually an agent controls a portion of resources in multi-agent scheduling, so the best decision may not be made from incomplete information of all resources, e.g. an agent try to assign an operation to its resource, without knowing a better resource owned by another agent. As a result, global coordination among agents could solve this trouble prevailingly.

In DSAFO, global coordination is implemented in *Member* agents with help of *Coordinator* agents. And there are two kinds of relationships between two *Member* agents with 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.

5.3 Agent roles in DSAFO

DSAFO consists of four roles of agents in all: *Blackboard*, *ResourceAdmin*, *Coordinator* and *Member*. Role *Blackboard* is active to fetch real-time flight data, role *Member* is active to achieve operations, check resource's out-of-date, and synchronize buddy list, and role *Coordinator* is active to synchronize resource textures of *Members*. Rest actions of all roles are passive, i.e. each of rest actions is driven by one (or some) of these four action directly or indirectly.

Practically, there are one unique *Blackboard*, one Unique *ResourceAdmin*, some *Coordinators* and more *Members*. And these agents are organized via message channels structured as Figure 5-2.

The channels form several organization models of *Federation* [111]. *Federation* organization is formed of a group of agents which have ceded some amount of autonomy to a single delegate which represents the group. This organization is good at reducing communications, matchmaking, brokering, translation services and facilitating dynamic agent pool [107, 111].

In the following subsections, program-like models are also given, role by role, in

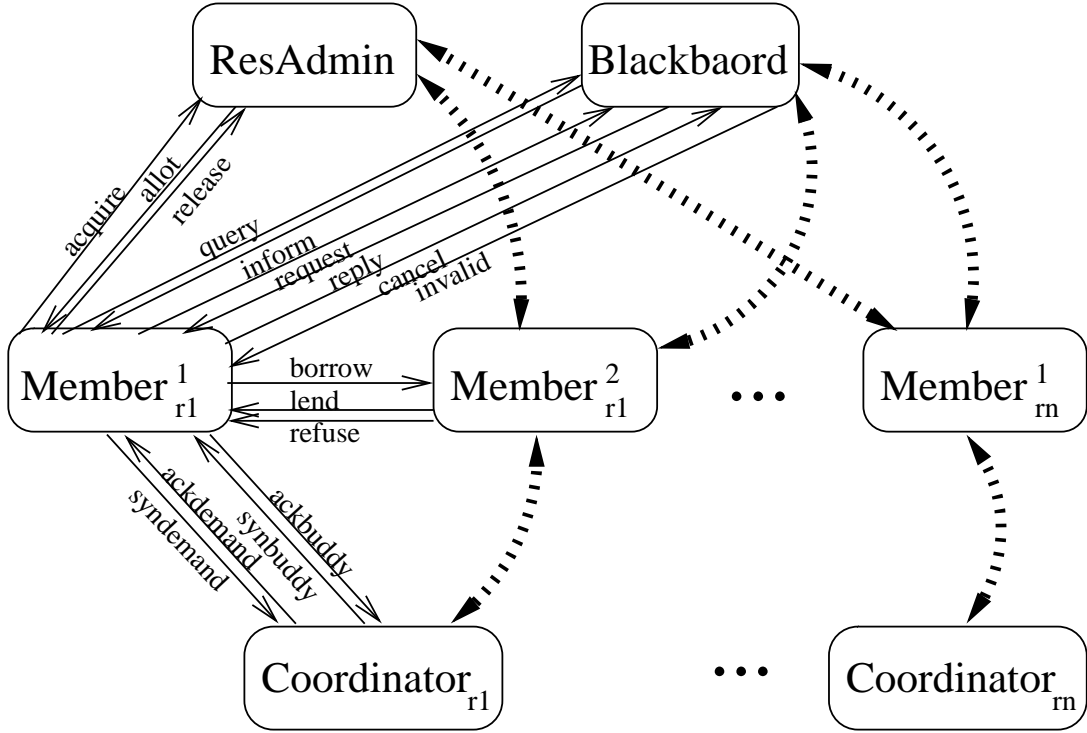


Figure 5-2: Channel organization for agent communication

pseudo codes at proper level to represent usages of all channels precisely.

5.3.1 Role *Blackboard*

Role *Blackboard* is designed to perceive flight data, to decompose flight service tasks into operations, and to assign operations with plans committed by *Member* agents. *Blackboard* has one active behavior

- reading flight data: to fetch date from run-and-scheduling actively in every Heart-beat;

and three passive behaviors:

- responding operation query: to answer message sender with the top operation in the type which is uniquely declared in the message;
- responding operation commitment: to answer message sender with “yes” if its plan is feasible, otherwise “no”;
- responding operation cancellation: to remove a committed service plan.

These asynchronous assignments are weak commitments, in fact. Because once the plan assigned by *Blackboard* is conflicted with another plan in the scope of related *Member* agent, a backtrack message will be received by *Blackboard*, then the plan will be dismissed. This type of actions is similar to *asynchronous weak-commitment search* [44, 45].

According to Figure 5–2, only one unique agent plays this role. With this agent, a sound, complete and consistent global schedule could be made from many *Member* agents' local viewpoints. And all final local plans in these *Member* agents are always consistent, because they are always consistent with the *Blackboard* finally.

The channel utilization and behaviors of *Blackboard* are shown in Figure 5–3:

```

procedure Blackboard_thread
  operations  $\leftarrow \emptyset$ ;
  flightdata  $\leftarrow \emptyset$ ;
  while(true)
    flightdata  $\leftarrow$  get_data ();
    operations  $\leftarrow$  operations  $\cup$  decomposite(flightdata);
    dequeue_message(sender, channel, content);
    switch (channel)
      case QUERY :
        sendto(sender, INFORM, weighted_EDD(content.opType));
      case REQUEST :
        if ( isfree (content.op))
          sendto(sender, REPLY, content);
          commit(content.op, sender, content.plan);
          adjust_succ(content.op, content.plan);
        else
          sendto(sender, INVALID, content);
        end if
      case CANCEL :
        recursive_free (content.op, false);
      case NIL :
        block(Heartbeat  $\times$  Blockfactor);
    end switch
    if (end_of_time)
      terminate_algorithm();
      output_result ();
    end if
  end while
end procedure

function weighted_EDD
  input taskType;
  p  $\leftarrow +\infty$ ;
  op  $\leftarrow$  NIL;
  for t in unsolved_op_in_30min(taskType)
    if (t.priv == VIP)
      return t;
    else if (t.LFT - now < p)
      op  $\leftarrow$  t;
  
```

```

        p ← t.LFT - now;
    end if
end for
return op;
end function

procedure recursive_free
input op
input notify
if (op.plan! = NIL and op.fixed == false)
    unassign(op);
    if (notify)
        sendto(op.plan.resp, INVALID, op.plan);
    end if
end if
for t in Succeed_operation(op)
    recursive_free (t, true);
end for
end procedure

```

Figure 5–3: Behaviors and channel utilization of agent role *Blackboard*

5.3.2 Role *ResourceAdmin*

Role *ResourceAdmin* is a resource administrator with two passive behaviors:

- responding resource acquirement: to search free resources in certain type and allot one to message sender if possible;
- responding resource release: to mark one resource available, which is usually four hours after allocation, i.e. half a man-day (4-hour work).

This role is also played by only one agent, and its channel utilization and behaviors are shown in Figure 5–4:

```

procedure ResourceAdmin_thread
lastAcq ← -∞;
lastOne ← NIL;
while(true)
    dequeue_message(sender, channel, content);
    switch (channel)
        case ACQUIRE :
            r ← available_res (content.resType);
            if (r! = NIL and (lastAcq < now - 1 or lastOne! = sender))
                r.busy ← true;
                sendto(sender, ALLOT, r);
                lastOne ← sender;
                lastAcq ← now;
            end if
        end case
    end switch
end while

```

```

    end if
  case RELEASE :
    content.resource.busy←false;
    content.resource.used++;
  case NIL :
    block(Heartbeat×Blockfactor);
  end switch
end while
end procedure

```

Figure 5–4: Behaviors and channel utilization of agent role *ResourceAdmin*

5.3.3 Role *Member*

Role *Member* is in charge of a type (or several types) of operations, it has three active behaviors

- query for operation: to send operation query to *Blackboard* actively, because a *Member* always has a desire to handle operations;
- out-of-date check: to send resource release to *ResourceAdmin* if a resource is dispatched after 4 hours;
- renewing buddy list: to send buddy list to *Coordinator* once in a certain time;

and seven passive behaviors:

- preparing for operation plan: after receiving goals of operations from *Blackboard*, it tries to make a local plan and send plan request back; after it fails to make a local plan, *Member* agent tries to borrow resource from its buddies;
- confirming plan:
 - success: after receiving “yes” from *Blackboard*, it tries to confirm plan locally. If confirmation is failed, then it sends back plan cancellation. Then, if the resource in service plan is held by another agent (buddy), then it copies current message to the holder;
 - failure: after receiving “no” from *Blackboard*, if the resource in service plan is held by another agent, then it copies current message to the holder;

- lending resource: after receiving resource borrow message from one agent, host *Member* tries to make plan locally, and then sends plan result (maybe OK or not) back. *Member* agent always has an open hand when other buddy needs help;
- confirming borrowing:
 - success: to copy plan to *Blackboard*;
 - failure: to send borrow message to next buddy; if no buddy left, send a resource request to *ResourceAdmin*;
- adding new resource: to set a new resource with properties declared in the message from *ResourceAdmin*;
- responding with resource texture: to send resource texture back to *Coordinator*;
- renewing buddy list: to renew buddy list with data from *Coordinator*.

Member agent also has a strategy of *asynchronous weak-commitment search* in resource selection practically, i.e. when *Blackboard* meets collision of two plans, one plan will be sent back by channel *invalid* to dismiss the plan.

And *Member*'s channel utilization and behaviors are shown in Figure 5–5:

```

procedure Member_thread
  buddies ← ∅;
  lastSyn ← -∞;
  while(true)
    for r in localResources
      if (is_expired(r))
        release(r);
        sendto(ResourceAdmin, RELEASE, r);
      end if
    end for
    dequeue_message(sender, channel, content);
    switch (channel)
      case INFORM :
        p ← earliest_finish_plan (content.op);
        if (p ≠ NIL)
          content.plan ← p;
          sendto(sender, REQUEST, content);
        else if (buddies.size > 0)
          content.buddylist ← buddies;
          sendto(buddies.first, BORROW, content);
        else
          sendto(ResourceAdmin, ACQUIRE, content);
        end if
      case REPLY :
        r ← try_assign (content.plan.res, content.plan);

```



```

if (r==false)
    sendto(Blackboard, CANCEL, content);
end if
if (content.plan.resOwner==self)
    if (r==true)
        lastReq←lastReq+Heartbeat×Delayfactor;
        opType←next_type();
    else if (content.plan.res!=self)
        sendto(content.plan.res, INVALID, content);
    end if
else
    sendto(content.plan.resOwner, REPLY, content);
end if
case INVALID :
    free_assignment(content.plan.res, content.plan);
    if (sender==Blackboard and content.plan.resOwner≠self)
        sendto(content.plan.resOwner, INVALID, content);
    end if
case BORROW :
    p←gen_plan_locally(content.op);
    if (p!=NIL)
        content.resource←p.res;
        content.plan←p;
        sendto(sender, LEND, content);
    else
        content.buddylist.remove(self);
        sendto(sender, REFUSE, content);
    end if
case LEND :
    sendto(Blackboard, REQUEST, content);
case REFUSE :
    if (content.buddylist.size>0)
        sendto(content.buddylist.first, BORROW, content);
    else
        sendto(ResourceAdmin, ACQUIRE, content);
    end if
case ALLOT :
    addres(content.res);
case REQDEMAND :
    sendto(sender, SYNDEMAND, res_job_free());
case SYNBUDDY :
    buddies←content;
case NIL :
    if (now−lastReq>Heartbeat×Reqcycle)
        sendto(Blackboard, QUERY, opType);
    end if
    if (now−lastSyn>Heartbeat×Syncycle)
        sendto(CoordinatorresType, REQBUDDY, self);
        lastSyn←now;
    end if
end switch
    block(Heartbeat×Blockfactor);
end while

```

end procedure

Figure 5–5: Behaviors and channel utilization of agent role *Member*

5.3.4 Role *Coordinator*

Role *Coordinator* coordinates all *Member* agents having same resource type as its. It has one active behaviors on resource texture synchronization

- resource texture synchronization: send resource texture query message to each of its *Member*;

and two passive behaviors on resource demand information:

- renewing resource texture: to renew local resource texture with data from every *Member*;
- responding buddy list synchronization: to send buddy list sorted by resource texture.

Channel utilization and behaviors of *Coordinator* are shown in Figure 5–6:

```
procedure Coordinator_thread
  nextSyn ← -∞;
  while(true)
    dequeue_message(sender, channel, content);
    switch (channel)
      case SYNDEMAND :
        memberDmd[sender] ← content.value;
      case REQBUDDY :
        sendto(sender, SYNBUDDY, buddies_in_order(sender));
      case NIL :
        if (memberHash.size > 0 and now > nextSyn)
          nextSyn ← nextSyn + Heartbeat × Syncycle;
          for m in memberDmd.candidates
            sendto(m, REQDEMAND, NIL);
          end for
        end if
      end switch
    block(Heartbeat × Blockfactor);
  end while
end procedure

function buddies_in_order
  input mem;
  bd ← ∅;
  for m in members
```

```

    if (m.opType!=mem.opType)
        bd += n;
    end if
end for
bd.sortBy(DESCENDING);
return bd;
end function

```

Figure 5–6: Behaviors and channel utilization of agent role *Coordinator*

5.4 A formalized summary

Besides former polyadic π -calculus definitions in Section 2.5, in this thesis we usually abbreviate name sequences (such as a tuple) in polyadic π -calculus to vector names, e.g. in AGSS problem a operation tuple structure

$$op \stackrel{\text{def}}{=} \langle \text{fno}, \text{type}, \text{parkNo}, \text{RT}, \text{LFT}, \text{ST}, \text{UT}, \text{ET} \rangle$$

may be abbreviated to “ \vec{op} ”. And a name in vector name could be accessed using “ \triangleright ”, e.g. the LFT (Latest Finish Time) of an operation \vec{op} is “ $op \triangleright LFT$ ”.

In the form of polyadic π -calculus, there are several parameters for DSAFO [139]: *SchOts* is a set representing operation types to schedule, for each type $t \in \text{SchOts}$, *Agent_t* is a set representing agent candidates who can work for t , *Resource* is a set representing resource types, for each resource type $r \in \text{Resource}$, *Otinres_r* is a set representing what operation types could be served by r , *Heartbeat* is a time duration representing how much local machine time represents one real minute, *Clockzero* is a moment representing when clock starts. Then we have

$$\begin{aligned}
 \text{DSAFO} \stackrel{\text{def}}{=} & (\text{SchOts}, \text{Agents}_t, \text{Resources}, \text{Otinres}_r, \text{Heartbeat}, \text{Clockzero}) \\
 & (\nu \text{query}_t^a, \text{inform}_t^a, \text{request}_t^a, \text{reply}_t^a, \text{cancel}_t^a, \text{invalid}_t^a, \text{reqbuddy}_t^a, \text{synbuddy}_t^a \\
 & , \text{borrow}_t^a, \text{lend}_t^a, \text{refuse}_t^a, \text{reqdemand}_t^a, \text{syndemand}_t^a, \text{acquire}_t^a, \text{allot}_t^a, \text{release}_t^a) \\
 & (\text{BLACKBOARD}|\text{RESADMIN}|\text{MEMBER}_t^a|\text{COORDINATOR}_r) \\
 & (t \in \text{SchOts}, a \in \text{Agents}_t, r \in \text{Resources})
 \end{aligned}$$

5.4.1 Blackboard

BLACKBOARD perceives information of flights coming in half an hour, decomposes them into operations and assigns these operations with plans from *Member* agents.

Parameter *Flights* is a set of all known flights, and for each $f \in Flights, t \in SchOts$, Ops_f^t denotes a unique operation (in certain flight's certain type), which is a structure representing operation tuple.

$$\begin{aligned} \text{BLACKBOARD} &\stackrel{\text{def}}{=} (Flights, Ops_f^t) \\ &\quad (BbFunc | RespQuery_t^a | RespReq_t^a | RespCancel_t^a) \\ &\quad (t \in SchOts, a \in Agents_t, f \in Flights) \end{aligned}$$

BbFunc is a set of useful functions supporting other behaviors of BLACKBOARD. For example, channel *clock* is watched at any time. When two parameters (output channel *tch* and time *time*) are received from *clock*, value $(time - Clockzero)/Heartbeat$ will be outputted through *tch*. In other words, function *clock(tch, systime)* converts time *time* from local system form to clock form and returns the result through *tch*.

$$\begin{aligned} BbFunc &\equiv !clock(tch, time). \overline{tch} \langle (time - Clockzero)/Heartbeat \rangle \\ &\quad | !readyop_t(rchret). (\nu chn) (\overline{clock} \langle chn, systime \rangle | chn(t) \\ &\quad \cdot (\nu c) (\overline{mostpreferredop_t} \langle c, t + 5, t + 30 \rangle | c(\overrightarrow{opr}). \overline{rchret} \langle \overrightarrow{opr} \rangle)) \\ &\quad | \dots \end{aligned}$$

Obviously the π -calculus model is too verbose to cover details of agent roles' behaviors. So nextly some behaviors of agent would not be modeled as precisely as in Section 5.3. Nevertheless, the main characteristics and portrait of DSAFO is clear in the next model.

$$\begin{aligned} RespQuery_t^a &\equiv !query_t^a. (\nu c) (\overline{weighted_edd_t} \langle c \rangle | c(\overrightarrow{op}). [\overrightarrow{op} \neq nil] \overline{inform_t^a} \langle \overrightarrow{op} \rangle)) \\ RespReq_t^a &\equiv !request_t^a(\overrightarrow{plan}). (\nu ch) (\overline{getplan} \langle ch, plan \triangleright fno, plan \triangleright ot \rangle \\ &\quad | ch(\overrightarrow{myplan}). ([\overrightarrow{myplan} = nil] (\overline{assign} \langle \overrightarrow{plan} \rangle. \overline{reply_t^a} \langle syn, \overrightarrow{plan} \rangle \\ &\quad \cdot \overline{enable_succ} \langle plan \triangleright fno, plan \triangleright ot \rangle) + [\overrightarrow{myplan} \neq nil] \overline{invalid_t^a} \langle \overrightarrow{plan} \rangle))) \\ RespCancel_t^a &\equiv !cancel_t^a(\overrightarrow{plan}). (\nu ch) (\overline{getplan} \langle ch, plan \triangleright fno, plan \triangleright ot \rangle | ch(\overrightarrow{myplan}) \\ &\quad \cdot [\overrightarrow{myplan} = \overrightarrow{plan}] \overline{recursive_free} \langle plan \triangleright fno, plan \triangleright ot \rangle, false) \end{aligned}$$

5.4.2 ResourceAdmin

RESADMIN is a resource administrator. When received resources request, RESADMIN would search free resources and allot one if possible. After 4-hour work, the resource

would be free again. Additionally, one resource serves no more than 12 hours, i.e. one and a half man-days.

Parameter $Reslist_r$ is a set of all known resources in type r , and $Historylist_t$ is a historical record of resource usage which aims at ensuring the number of usage of each resource in one day is no more than three, which is pre-assumed (12 hours/4 hours) before.

$$\begin{aligned}
 RESADMIN &\stackrel{\text{def}}{=} (Reslist_r, Historylist_r)(RaFunc|Resreq_t^a|Resrelease_t^a) \\
 &\quad (r \in Resources, t \in SchOts, a \in Agents_t) \\
 Resreq_t^a &\equiv !resreq_t^a(begin tm).(\nu ch)(\overline{available_res_t}\langle ch, begin tm \rangle | ch(\overrightarrow{res}) \\
 &\quad .[\overrightarrow{res} \neq nil]\overline{allot_t^a}\langle res \triangleright name, begin tm \rangle . \overline{set_busy}\langle \overrightarrow{res}, true \rangle \\
 &\quad . \overline{log_allot}\langle \overrightarrow{res}, t, a \rangle)) \\
 Resrelease_t^a &\equiv !release_t^a(resname).(\nu ch)(\overline{getresfromhash}\langle ch, resname \rangle \\
 &\quad | ch(\overrightarrow{res}).[\overrightarrow{res} \neq nil]\overline{log_release}\langle \overrightarrow{res} \rangle . \overline{set_busy}\langle \overrightarrow{res}, false \rangle)
 \end{aligned}$$

5.4.3 Member

A *Member* agent $MEMBER_t^a$ is an agent named a and in charge of operation type t . It always has a desire to handle operations. After $MEMBER_t^a$ received available operation goals from BLACKBOARD, it intends to make suitable plans and request for them. Every member agent always has an open hand when its buddies need help.

For each $t \in SchOts, a \in Agents_t$, $MEMBER_t^a$ has some parameters: *Syncycle* is a preset integer representing demand synchronization cycle, *Resource_t^a* is a set of structure holding local resources' history and current commitment plans, and *Remoterest_t^a* represents the remote resources lent to itself.

$$\begin{aligned}
 MEMBER_t^a &\stackrel{\text{def}}{=} (Syncycle, Resource_t^a, Remoterest_t^a) \\
 &\quad (MbFunc|ActQry_t^a|MkPlan_t^a|AckReply_t^a|AckInvl_d_t^a|TryLend_t^a \\
 &\quad |AckLend_t^a|CallBd_t^a|Ackres_t^a|AckRDmd_t^a|DumSyn_t^a|AckSyBd_t^a) \\
 &\quad (t \in SchOts, a \in Agents_t) \\
 ActQry_t^a &\equiv (\overline{query_t^a}.\overline{block_t^a}\langle Heartbeat \rangle).(\nu ch)(\overline{getexpiredres_t^a}\langle ch \rangle | ch(\overrightarrow{reslist}) \\
 &\quad .[\overrightarrow{reslist} \neq nil]\overline{releaseall_t^a}\langle \overrightarrow{reslist} \rangle).)^{+\infty}
 \end{aligned}$$

$$\begin{aligned}
 MkPlan_t^a &\equiv !inform_t^a(\vec{op}).(\nu ch)(\overline{makenullplan}_t\langle ch, \vec{op} \rangle | ch(\vec{n})) \\
 &\quad .(\nu p)(\overline{locallyplan}_t\langle p, \vec{n} \rangle | p(\overline{plan}, res).([res \neq null]\overline{request}_t^a\langle \overline{plan} \rangle \\
 &\quad + [res = null](\nu c)(\overline{getbuddy}_t\langle c \rangle | c(\overline{bud} \\
 &\quad .([bud \neq nil]\overline{try_borrow}_t\langle bud \triangleright top, \overline{plan} \rangle \\
 &\quad + [bud = nil]\overline{acquire}_t^a\langle plan \triangleright EST - 1 \rangle)))))) \\
 AckReply_t^a &\equiv (!reply_t^a(syn, \overline{pln}).(\nu ch)(\overline{try_assign}_t^a\langle ch, pln \triangleright res, \overline{pln} \rangle | ch(ret) \\
 &\quad .([ret \neq true]\overline{cancel}_t^a\langle \overline{pln} \rangle | ([pln \triangleright resOwner \neq self]\overline{reply}_t^{pln \triangleright resOwner} \\
 &\quad + [pln \triangleright resOwner = self]([ret = true](\overline{setlreq}_t\langle now \rangle.\overline{enum_restype}) \\
 &\quad + [ret \neq true][pln \triangleright resp \neq self]\overline{invalid}_t^{pln \triangleright resp}\langle \overline{pln} \rangle)))))) \\
 AckInvl_t^a &\equiv !invalid_t^a(\overline{pln}).\overline{freeres}_t^a\langle plan \triangleright res \rangle.[pln \triangleright sender = BLACKBOARD] \\
 &\quad [pln \triangleright resOwner \neq self]\overline{invalid}_t^{pln \triangleright resOwner}\langle \overline{pln} \rangle \\
 TryLend_t^a &\equiv !borrow_t^a(\overline{uplan}, \vec{lst}, succ, fail).(\nu ch)(\overline{locallyplan}_t\langle ch, \overline{uplan} \rangle \\
 &\quad | ch(\overline{plan}).([plan \triangleright res \neq null]\overline{assign}_t^a\langle \overline{plan} \rangle.\overline{succ}\langle \overline{plan} \rangle \\
 &\quad + [plan \triangleright res = null]\overline{fail}_t\langle \overline{uplan}, \vec{lst} \rangle)) \\
 AckLend_t^a &\equiv !lend_t^a(\overline{plan}).\overline{request}_t^a\langle \overline{plan} \rangle \\
 CallBd_t^a &\equiv !refuse_t^a(\overline{uplan}, \vec{lst}).(\nu c)(\overline{topof}_t\langle c, \vec{lst} \rangle \\
 &\quad | c(next, \overline{nlist}).([next \neq null]\overline{next}_t\langle \overline{uplan}, \overline{nlist}, lend_t^a, refuse_t^a \rangle \\
 &\quad + [next = null]\overline{acquire}_t^a\langle uplan \triangleright EST - 1 \rangle)) \\
 Ackres_t^a &\equiv !allot_t^a(name, begintm).(\nu ch)(\overline{genres}_t\langle ch, name, begintm, 239 \rangle \\
 &\quad | ch(\vec{res}).[\vec{res} \neq nil]\overline{addres2locallist}_t\langle \vec{res} \rangle) \\
 AckRDmd_t^a &\equiv !reqdemand_t^a.(\nu ch)(\overline{res_freedom}_t^a\langle ch \rangle | ch(ret).\overline{syndemand}_t^a\langle ret \rangle) \\
 DumSyn_t^a &\equiv (\overline{reqbuddy}_t.\overline{block}_t^a\langle Syncycle \times Heartbeat \rangle.)^{+\infty} \\
 AckSyBd_t^a &\equiv !synbuddy_t^a(\vec{lst}).\overline{setbuddy}_t\langle \vec{lst} \rangle
 \end{aligned}$$

5.4.4 Coordinator

A coordinator agent $COORDINATOR_r$ coordinates all member agents having resource r . After elimination of plan backup mechanism, it has several behaviors on information synchronization and buddy list making up.

Parameter *Syncycle* is a preset integer representing demand synchronization cycle. For each $r \in Resources$, $MetaInfo_r$ is a set of meta-level information of members, such as resource demand and number of finished operations.

$$\begin{aligned}
 \text{COORDINATOR}_r &\stackrel{\text{def}}{=} (\text{Metainfo}_r, \text{Syncycle})(\text{CooFunc} | \text{Synch}_r | \text{Store}_t^a | \text{RespBd}_t^a) \\
 &\quad (r \in \text{Resources}, t \in \text{Otinres}_r, a \in \text{Agents}_t) \\
 \text{Synch}_r &\equiv (\overline{\text{synall}}_r. \overline{\text{block}}_r \langle \text{Syncycle} \times \text{Heartbeat} \rangle.)^{+\infty} \\
 \text{Store}_t^a &\equiv !\text{syndemand}_t^a(dm).(\nu ch)(\overline{\text{setval}} \langle ch, dm \rangle | ch(\text{demands}_t^a) \\
 &\quad \overline{\text{removelist}} \langle \text{demands}_t^a \rangle. \overline{\text{insertsort}} \langle \text{demands}_t^a, \text{DESC} \rangle) \\
 \text{RespBd}_t^a &\equiv !\text{reqbuddy}_t^a.(\nu ch)(\overline{\text{genbuddylist}}_t \langle ch \rangle | ch(\overrightarrow{\text{blst}}). \overline{\text{synbuddy}}_t^a \langle \overrightarrow{\text{blst}} \rangle)
 \end{aligned}$$

5.5 Complexity

There are constant types of different operations in AGSS satisfaction and problem, and all these operations share same scheduling strategy. In other words, all type of resources share homogeneous scheduling, and their solutions do look in same modes. Consequently total time cost is in direct proportion to a certain type of operations such as *unload cargo and mail* (UC). Hence,

$$t_{DSAFO} \sim \text{Const} \times t_{UC}.$$

5.5.1 UC's Complexity

UC is served by baggage tractors (BTs). Similarly to Lemma 3, the expectation of BTs

$$BT^* \sim O(n)$$

can be proved. It means the expectation of BTs is in direct ratio to the number of flights. And some parameters of the algorithm affect the complexity, so we assume there are $O(n)$ BT *Member* agents and the rest of parameters are reasonable const to compute the upper bound of complexity $t_{DSAFO_{UB}}$.

5.5.1.1 Complexity from agent behaviors

For the UC operations and related algorithm computation, there is one *Blackboard*, one *ResourceAdmin*, one *Coordinator*, and $O(n)$ BT *Member* agents. The *Blackboard* actively gets flight data in cycles, and response no more than $O(n)$ messages from channels

QUERY, REQUEST and CANCEL. All of these causes agent self-behavior running time:

$$\begin{aligned}
 t_{bb} &= t_{getInfo} + t_{QUERY} + t_{REQUEST} + t_{CANCEL} \\
 &\lesssim O(n^2) + O(n) \times [O(n) + O(n) + O(n)] \\
 &\sim O(n^2)
 \end{aligned}$$

The *ResourceAdmin* should response $O(n)$ messages from channels ACQUIRE and RELEASE, so the time complexity of agent self-behavior is:

$$t_{resAdmin} = t_{ACQUIRE} + t_{RELEASE} \lesssim O(n) \times [O(n) + O(1)] \sim O(n^2)$$

Every *Member* actively checks buddy resource expiration and actively synchronizes buddies' resource textures through *Coordinator* in cycles, and responses messages from channels IMFORM, REPLY, INVALID, BORROW, LEND, REFUSE, ALLOT, REQDEMAND and SYNBUDDY. With multiple *Member* agents — not single one — it is very hard to make sure how many operations requests are made through these channels for each agent. Nevertheless, from a global viewpoint, it is very clear that all *Member* agents process $O(n)$ times totally.

$$\begin{aligned}
 \sum t_{mem_i} &= \sum (t_{releaseRes} + t_{IMFORM} + t_{REPLY} + t_{INVALID} + t_{BORROW} + t_{LEND} \\
 &\quad + t_{REFUSE} + t_{ALLOT} + t_{REQDEMAND} + t_{SYNBUDY} + t_{activeSyn} + t_{activeReq}) \\
 &\lesssim O(n) + O(n) \times [O(n^2) + O(n) + O(1) + O(n^2) + O(1) + O(n^2) \\
 &\quad + O(n) + O(n^2) + O(n) + O(1) + O(1)] \\
 &\sim O(n^3)
 \end{aligned}$$

The *Coordinator*, which is in charge of BT, actively synchronizes its member's resource texture, and responses messages from channels SYNDEMAND and REQBUDDY.

$$\begin{aligned}
 t_{coord} &= t_{SYNDEMAND} + t_{REQBUDDY} + t_{activeSyn} \\
 &\lesssim O(n) + O(n^2 \times \log n) + O(n) \\
 &\sim O(n^2 \times \log n)
 \end{aligned}$$

Summing up the upper bounds of these time complexity, we could have the upper bound of complexity of UC operations caused by agent behaviors:

$$\begin{aligned}
 t_{UC_behav_{UB}} &= t_{bb_{UB}} + t_{resAdmin_{UB}} + \sum t_{mem_{i_{UB}}} + t_{coord_{UB}} \\
 &\lesssim O(n^2) + O(n^2) + O(n^3) + O(n^2 \times \log n) \\
 &\sim O(n^3)
 \end{aligned}$$

5.5.1.2 Complexity from agent communications

The communications among agents are simpler. Totally there are four types of communications in DSAFO: *Member-Blackboard*, *Member-Member*, *Member-Coordinator* and *Member-ResourceAdmin*. They should be considered one by one:

Member-Blackboard There are $O(n)$ UC operations in total, so channels INFORM, REQUEST and REPLY would be triggered by $O(n)$ times normally¹. Channels would be used no more than $O(n)$ times normally. Channel REQUEST would be used no more than $O(n)$ times, because there are no more than $O(n)$ *Member* agents in charge of UC operations in a limited long time, totally.

Member-Member Because each *Member* agent has no more than $O(n)$ buddies, hence each UC operation needs no more than $O(n)$ communications for a successful resource borrowing.

Member-Coordinator In a limited long scheduling time, there are $O(1)$ times of resource texture synchronization in each *Member-Coordinator* relation. And there no more than $O(n)$ *Member* agents, so there are no more than $O(n)$ times of resource texture synchronization.

Member-ResourceAdmin To solve $O(n)$ UC operations, it has been concisely proved previously that $O(n)$ resources (BT) are needed. Hence channels ACQUIRE, ALLOT and RELEASE would be used for $O(n)$ times normally.

As a summing-up from above analysis, the upper bound of UC processing complexity caused by communication is:

$$\begin{aligned}
 t_{UC_transUB} &= t_{resAllotUB} + t_{synDemandUB} + t_{synBuddyUB} + UC_op \times t_{opReqUB} + t_{borrowUB} \\
 &\sim \{O(n) + O(n) + O(n) + O(n) \times [O(1) + O(n)]\} \times t_{trans} \\
 &\sim O(n^2) \times t_{trans}
 \end{aligned}$$

where t_{trans} denotes the average message transmission time. The t_{trans} is usually several milliseconds in real-world local area networks, so it is too large to be ignored.

¹“Normally” stands for ignoring extreme cases such as being trapped with frequent requests failure caused by too little Blockfator or something else

5.5.1.3 UC's complexity summary

And the rest processes (such as agent initiates and data fetching done by *Blackboard*) are less complex, so the upper bound of time complexity of processing UC operations is:

$$t_{UC_{UB}} = t_{UC_behav_{UB}} + t_{UC_trans_{UB}} \sim O(n^3) + O(n^2) \times t_{trans}$$

5.5.2 Complexity of DSAFO

Accordingly, the upper bound of DSAFO complexity is

$$t_{DSAFO_{UB}} \sim \text{Const} \times t_{UC_{UB}} \sim O(n^3) + O(n^2) \times t_{trans}$$

Also, we can assume there are $O(1)$ *Member* agents in charge of UC operations, then have the lower bound of DSAFO complexity is:

$$t_{DSAFO_{LB}} \sim O(n^2) + O(n) \times t_{trans}$$

5.6 Implementation

JADE² (Java Agent DEvelopment Framework) is a software framework [141] to develop agent-based applications in compliance with the FIPA specifications³ for interoperable intelligent multi-agent systems. The goal is to simplify the development while ensuring standard compliance through a comprehensive set of system services and agents.

JADE can then be considered an agent middle-ware that implements an agent platform and development framework. It deals with all those aspects that are not peculiar of the agent internals and that are independent of the applications, such as message transport, encoding and parsing, or agent life-cycle.

JADE is also free software and is distributed by *Telecom Italia*⁴, the copyright holder, in open source software under the terms of the LGPL (Lesser General Public License Version 2).

We implemented DSAFO in JADE and implemented the channels on FIPA ACL [72, 73], with grammar of Appendix A. Some of the running user interfaces are shown in Figure 5–7 and Figure 5–8.

²See <http://jade.tilab.com/>

³See <http://www.fipa.org/>

⁴See <http://www.telecomitalia.com/>

For example, The plan of “BT35” for *unload cargo and mail* of flight *CA109* consists of two travel (marked with compact line), one service and one reset actions (marked with R), furthermore there are some small buffer gaps (1 minute usually) between actions. Once the service of operation *unload cargo and mail* is done, then operation *load cargo and mail* could start being served. Furthermore, in Figure 5–7 this *unload cargo and mail* operation plan is labeled with “CT-Agt0(CT+Agt1)”, which means the resource, i.e. the baggage tractor “BT35”, is held by CT-Agt0, however CT+Agt1 borrows it for this operation; in Figure 5–8 the target flight of this plan, i.e. *CA109*, is in red, and that color also stands for a resource borrowing.

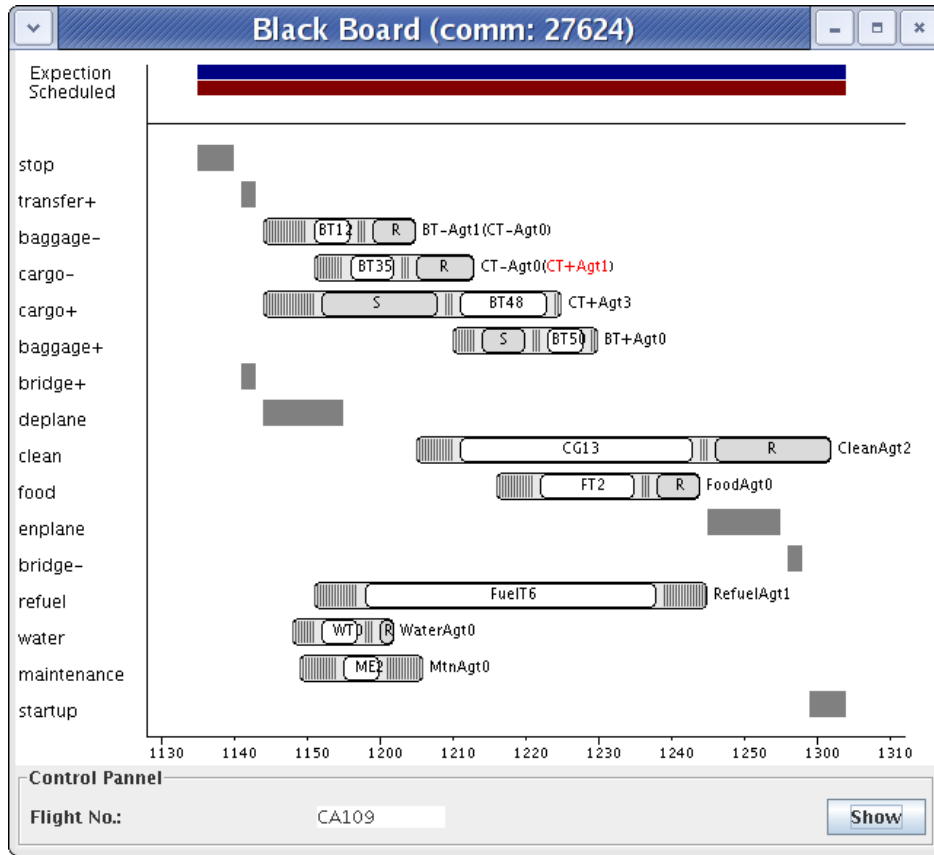


Figure 5–7: Gantt chart of typical scheduled plans for a flight (the *Blackboard*)

Because DSAFO is implemented in JADE and Java, and Java run-time is an OS-independent virtual machine environment, DSAFO could run in any mainstream operating systems, such as Windows, UNIX, Linux, MAC OS, etc. From JADE RMA GUI (Remote Monitoring Agent, Graphical User Interface), DSAFO is a group of agents running as Figure 5–9. JADE sniffer could also sniff then message transmission among agents in DSAFO, as shown in Figure 5–10. Also JADE introspector could show us the inner status of a certain agent in DSAFO, as shown in Figure 5–11.

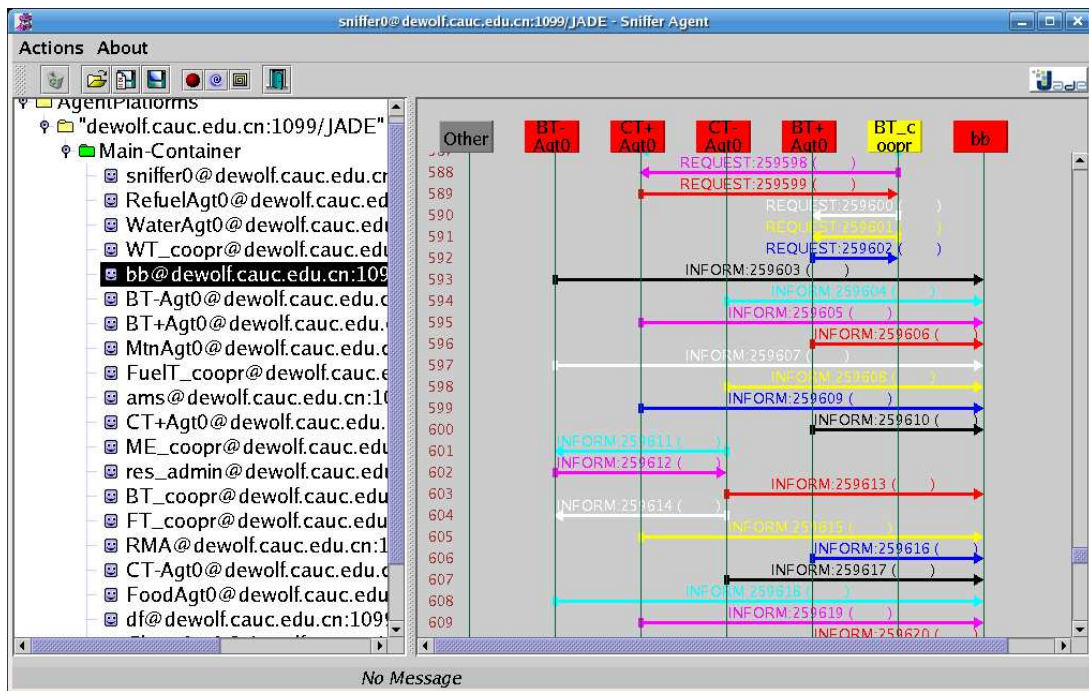


Figure 5-10: Communication observed by JADE sniffer

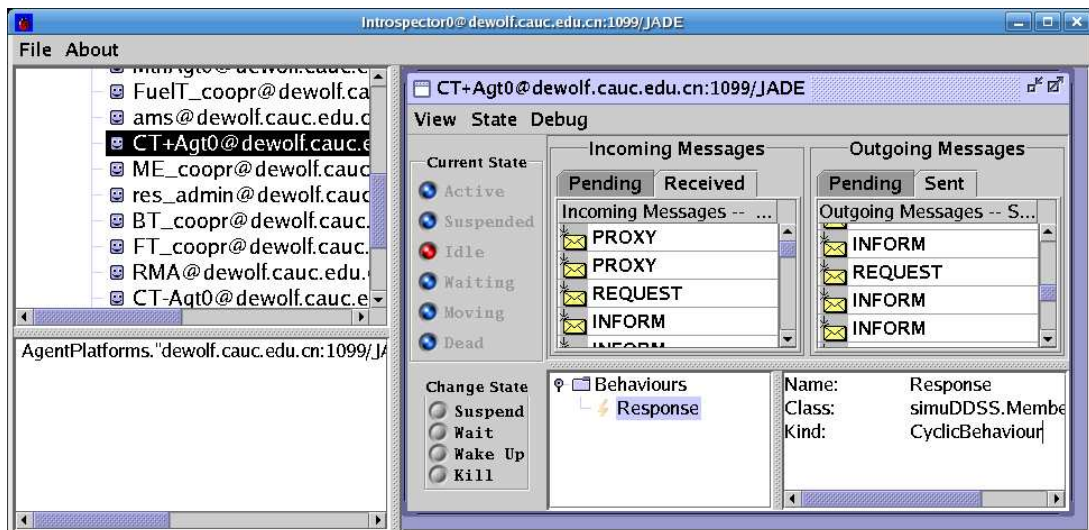


Figure 5-11: Agent inner states observed by JADE introspector

Chapter 6 Experiments and Analysis on Diverse Parameters

In this chapter, a kind of resources BTs are selected to test algorithm DSAFO; a series of experiments are made to test influences by algorithm factors: *Member* agent number, *Blockfactor*, *Delayfactor* and *Syncycle*; a following brief summary is presented.

6.1 The experiment

Our AGSS test data are 252 real-life transfer flights, which are collected from a whole day records of *Beijing Capital International Airport* (BCIA) historical data.

As show in Figure 5–7, each transfer flight is assumed to have nine typical operations: Unload baggage (UB), Unload cargo and mail (UC), Load cargo and mail (LC), Load baggage (LB), Cleaning, Catering, Watering, Refueling and maintenance. And the rest of operations are presumed to be fixed, i.e., they would be done perfectly without program assistance. As a result, there are 2,268 operations to be dispatched to hundreds of aircraft ground support resources for daily transfer flights.

There are so many types of operations, so it must be troublesome to completely analyze algorithm in all nine types of resources or to present detailed experiment results. Nevertheless it should be noticed that all type of resources and their operations share homogeneous scheduling mechanisms, and their solutions do look in same modes. That means when a good scheduling is done for one type of resources, all the resource schedules are good in general.

Among those operations and resources, BT (baggage tractor) related operations should be the most complex, because only BT can serve multiple kinds of operations as shown in Table 1–1. Hence one BT, which is different than other resources, could be assigned to different types of operations successively, e.g. UB–UC–LB–UC. So we choose BT consumption and relative man-days to algorithm quality test in the below sections.

6.2 The factors

As a dynamic coordination based multi-agent algorithm, DSAFO is naturally unstable in computing solutions, because of the parallel *Member* agents' coordination, unstable network communications and maybe even some tiny sensitive features from environment JADE and program language Java. Consequently we have spontaneous 250 runs to get the distributions of the solutions in each test.

In the description of DSAFO in Section 5.3, there are some parameters listed in pseudo codes: Blockfactor, Delayfactor, Syncycle and Reqcycle. Besides these four internal-agent parameters, *Member* agent number — such as *Member* agents in charge of BTs — is another important parameter. Algorithm DSAFO and its output is influenced largely or tinily respectively by different parameters. In following subsections, the parameters are experimented and analyzed one by one, both through solution comparison and theoretical analysis.

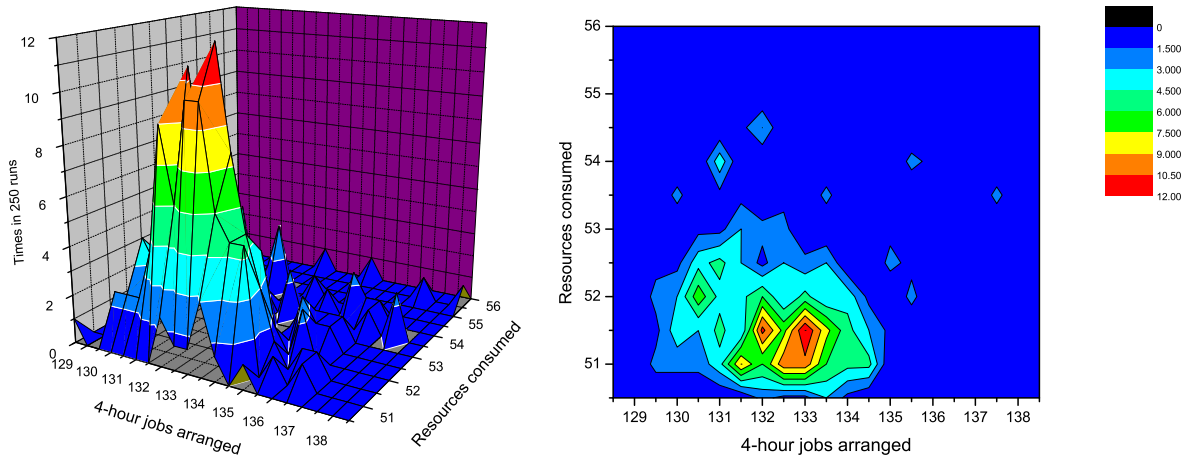
6.2.1 *Member* agent number and Reqcycle

Combining *Member* agent number and Reqcycle stands for how often *Member* agents in DSAFO try to request a certain type of operations. Keeping a suitable request frequency by suitable Reqcycle, number of *Member* agents affects DSAFO largely. And in all the parameters, number of *Member* agents should be notably influential. When Blockfactor=1/12, Delayfactor=1/6, Syncycle=5, and Reqcycle is from 1/6 to 2 respectively, Figure 6–1 and 6–2 show the solution distributions with different BT *Member* agents in 3D map and contour map, as well as Figure 6–3 shows the marginal distributions of resource (BT) and man-day consumptions.

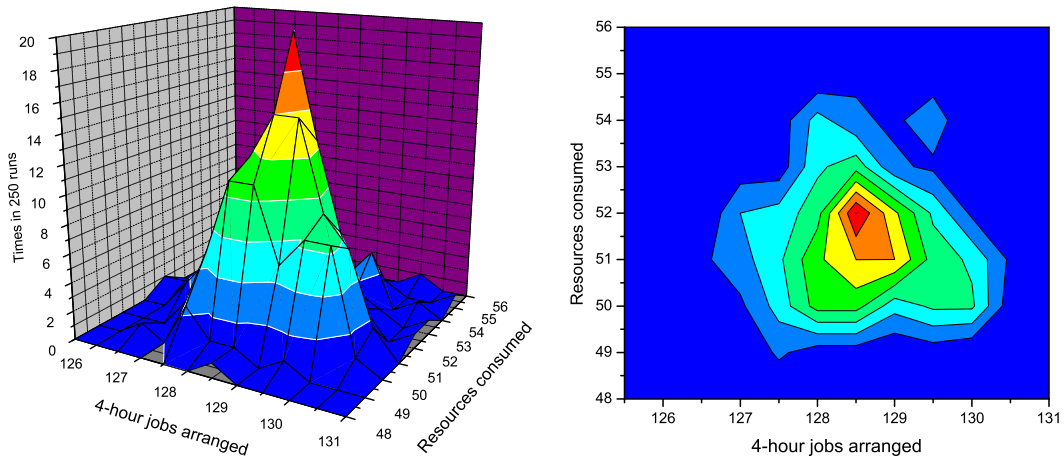
From Figure 6–1, 6–2 and 6–3, We can see an interesting phenomenon:

- When the BT *Member* agent number increases from 1 to 4, the solutions are going better: average BT consumption and 4-hour jobs arrangement both are decreasing. From charts we can see that in 3D maps the heaps of distribution are moving left-down; in the marginal chart, the distribution polylines are moving left both in resource consumptions and 4-hour jobs arrangement; meanwhile the shape of solution distributions are becoming “better¹ and better”.

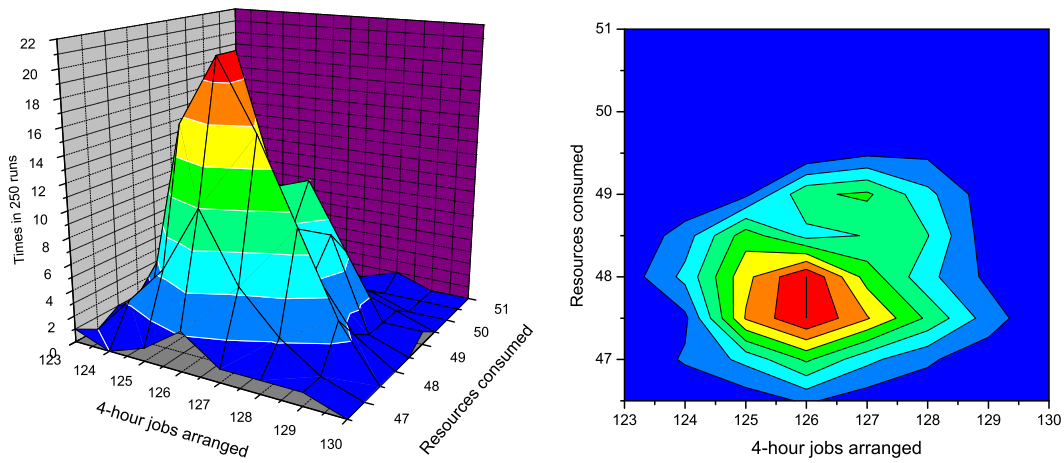
¹Ocular witness is not very strong proof, however the good shapes must be proved to fit normal distribution better, both in 3D charts and 2D polyline charts.



1 BT Member Agent

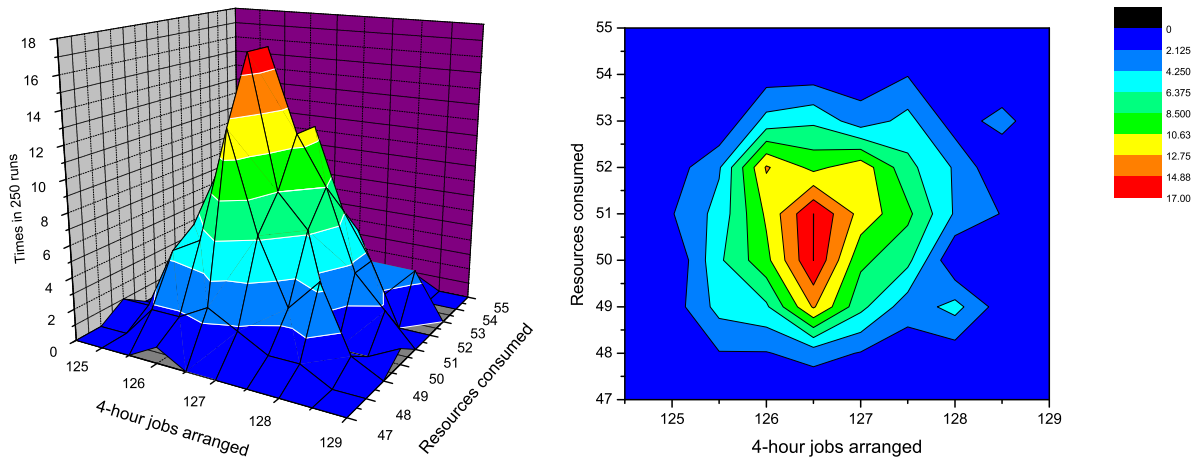


2 BT Member Agents

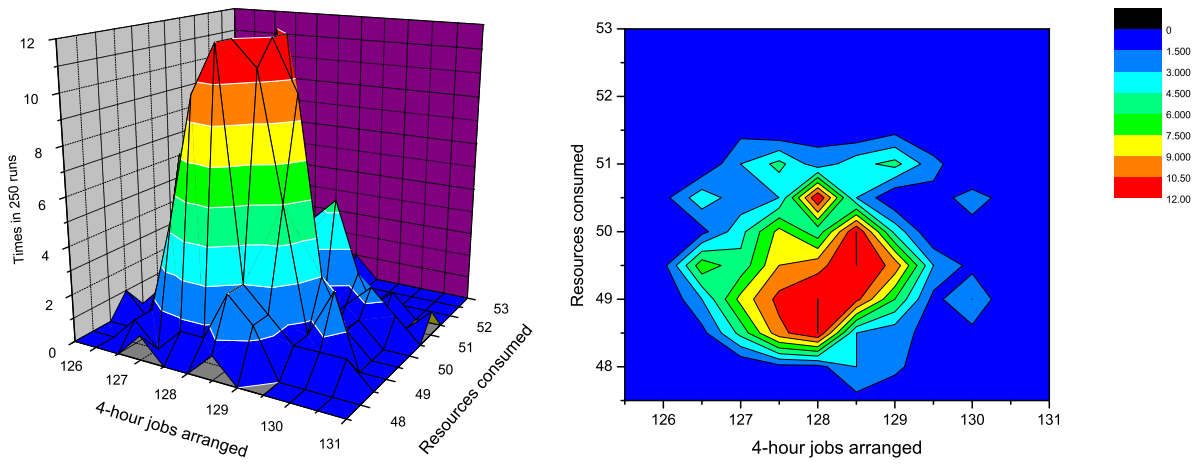


4 BT Member Agents

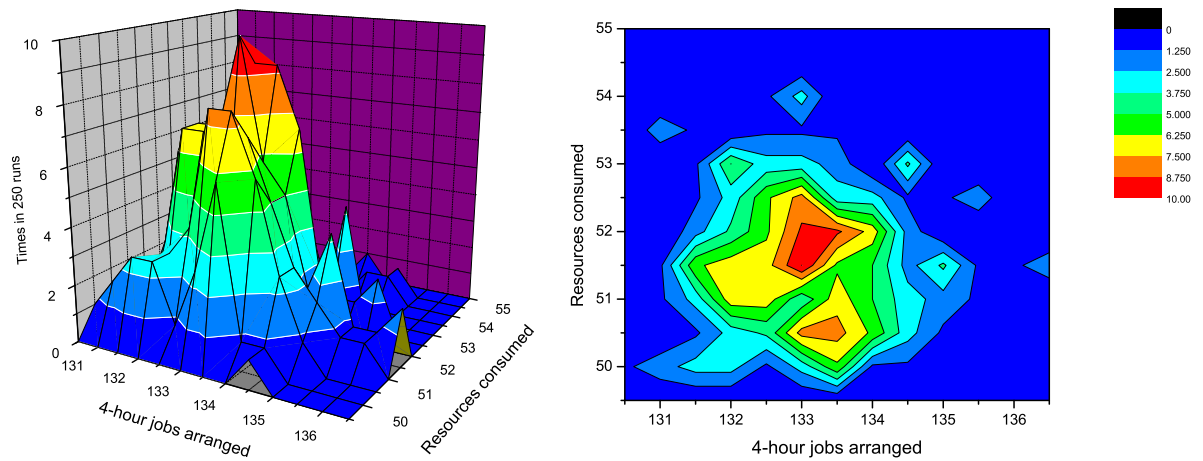
Figure 6-1: BT solution distributions with respect to agent number



6 BT *Member Agents*



8 BT *Member Agents*



12 BT *Member Agents*

Figure 6-2: BT solution distributions with respect to agent number (more)

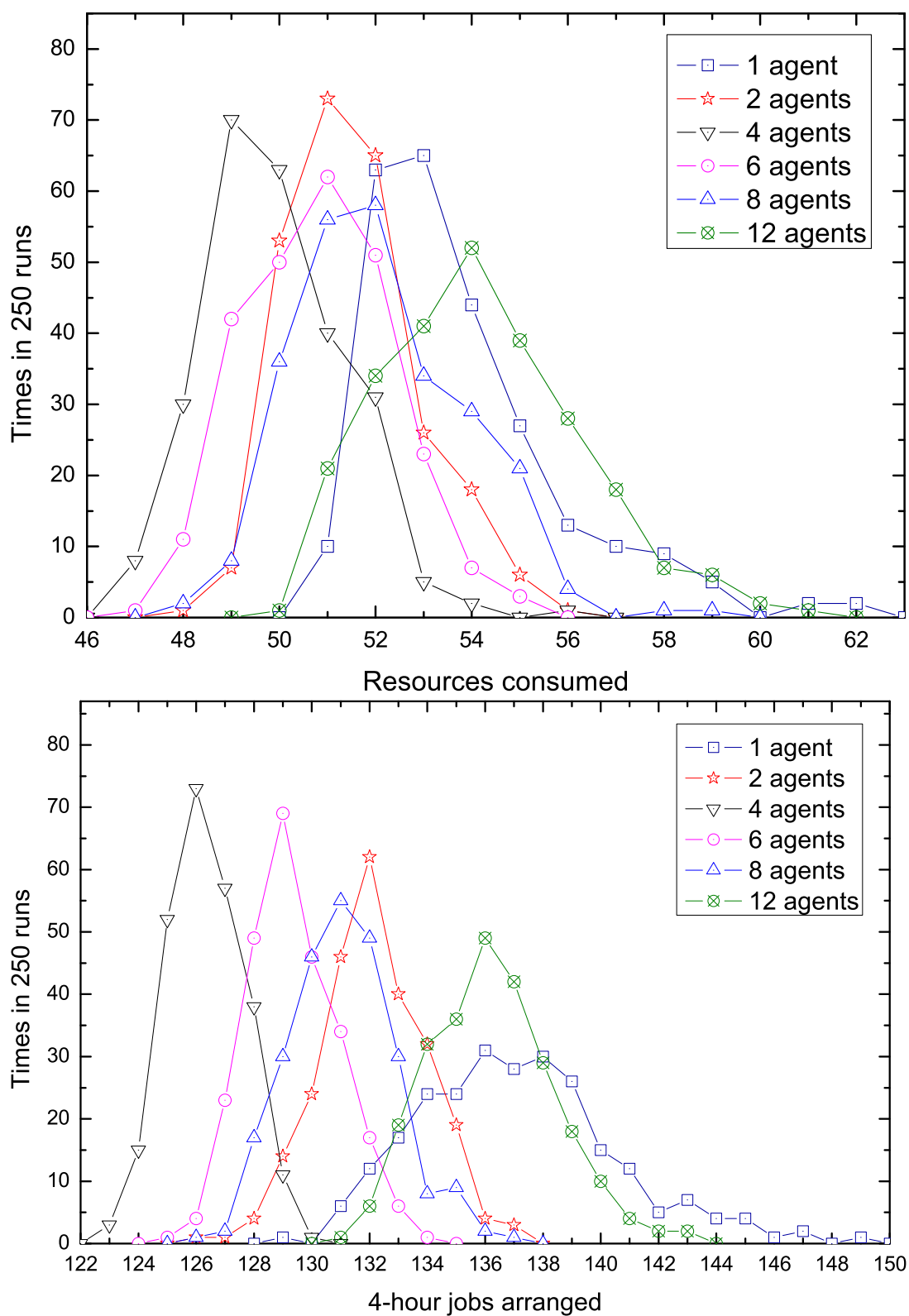


Figure 6-3: BT solution marginal distributions with respect to agent number

- On the contrary, when the number increases from 4 to 12, the solutions are going worse: average BT consumption and 4-hour jobs arrangement both are increasing. From charts we can see that in 3D maps the heaps of distribution are moving right-up; in the marginal chart, the distribution polylines are moving right both in resource consumptions and 4-hour jobs arrangement; meanwhile the shape of solution distributions are becoming “worse and worse”.

Why does it happen? Theoretical analysis could offer some reasons.

It is clear that multiple *Member* agents in DSAFO are designed to divide global solution space into local partitions, and the number of local partitions is equal to *Member* agent number. Then each local partition is handled with heuristics by single *Member* agent. Though such a Divide-and-Conquer — a very traditional AI technique — is the nature of some heuristics, DSAFO has a special mechanism to jump from local minimum of Divide-and-Conquer: some optimizations are processed via coordinations among partitions (agents). The optimizations provide DSAFO output better than Divide-and-Conquer heuristics.

Why the solutions are not good with too few agents? It should be because the coordination strategy takes insufficient effect, or sometimes no effect. If there is only one BT *Member* agent, then no coordinations will happen, so DSAFO with only one BT *Member* agent must have weak abilities to jump out of local minimum.

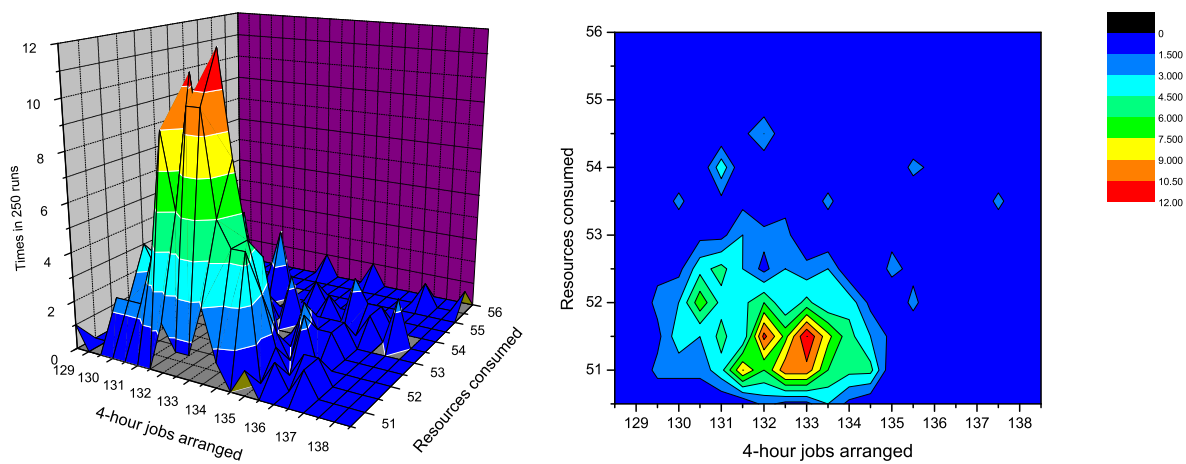
Why the solutions are not good with too many agents? It might be because too many agents divide the global solution into too many fragments, so that local heuristics in these fragments is lack of enough global viewpoints, and coordination cannot optimize these fragments very well.

6.2.2 Blockfactor

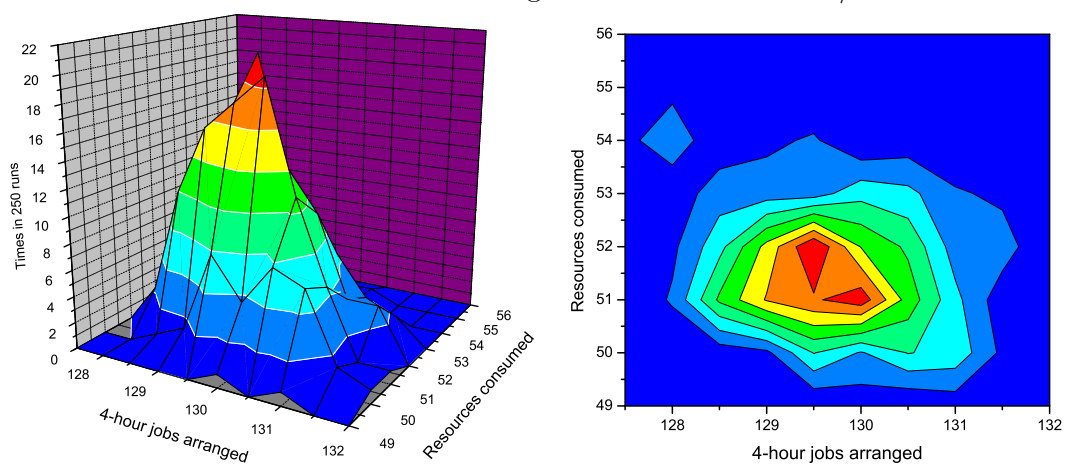
Blockfactor is another influential parameter. We set Agentnum=1, Delayfactor=1/6, Syncycle=5 and Reqcycle = 1/6, and then we get solution distribution with respect to Blockfactor comparison in 3D map and contour map, as shown in Figure 6–4, as well as marginal distribution comparison with respect to Blockfactor in Figure 6–5.

From Figure 6–4 and 6–5, We can see that:

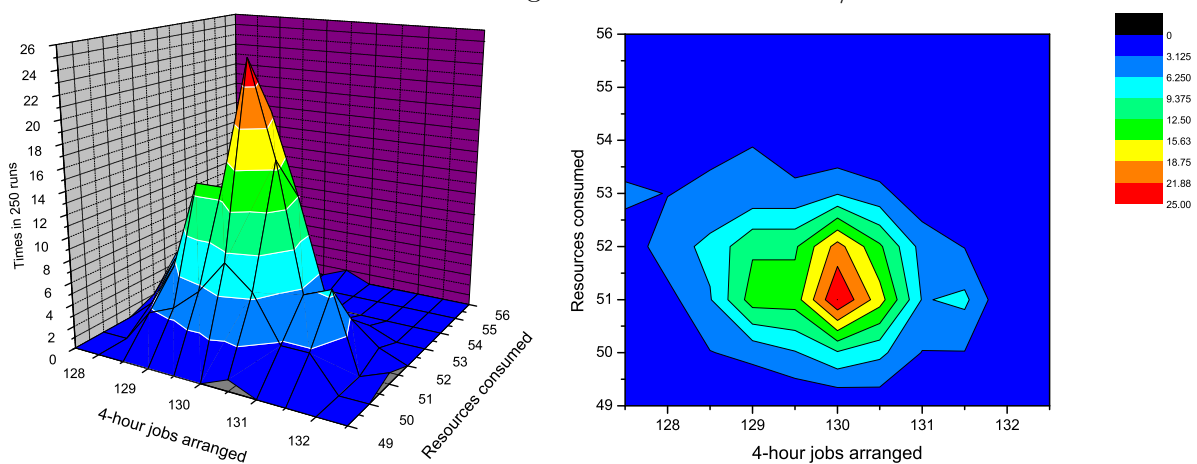
- The peak in the 3D map and contour map moves a bit downward when Blockfactor decreases, that means resource consumption stays the same approximately and 4-



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



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



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

Figure 6-4: BT solution distributions with respect to Blockfactor

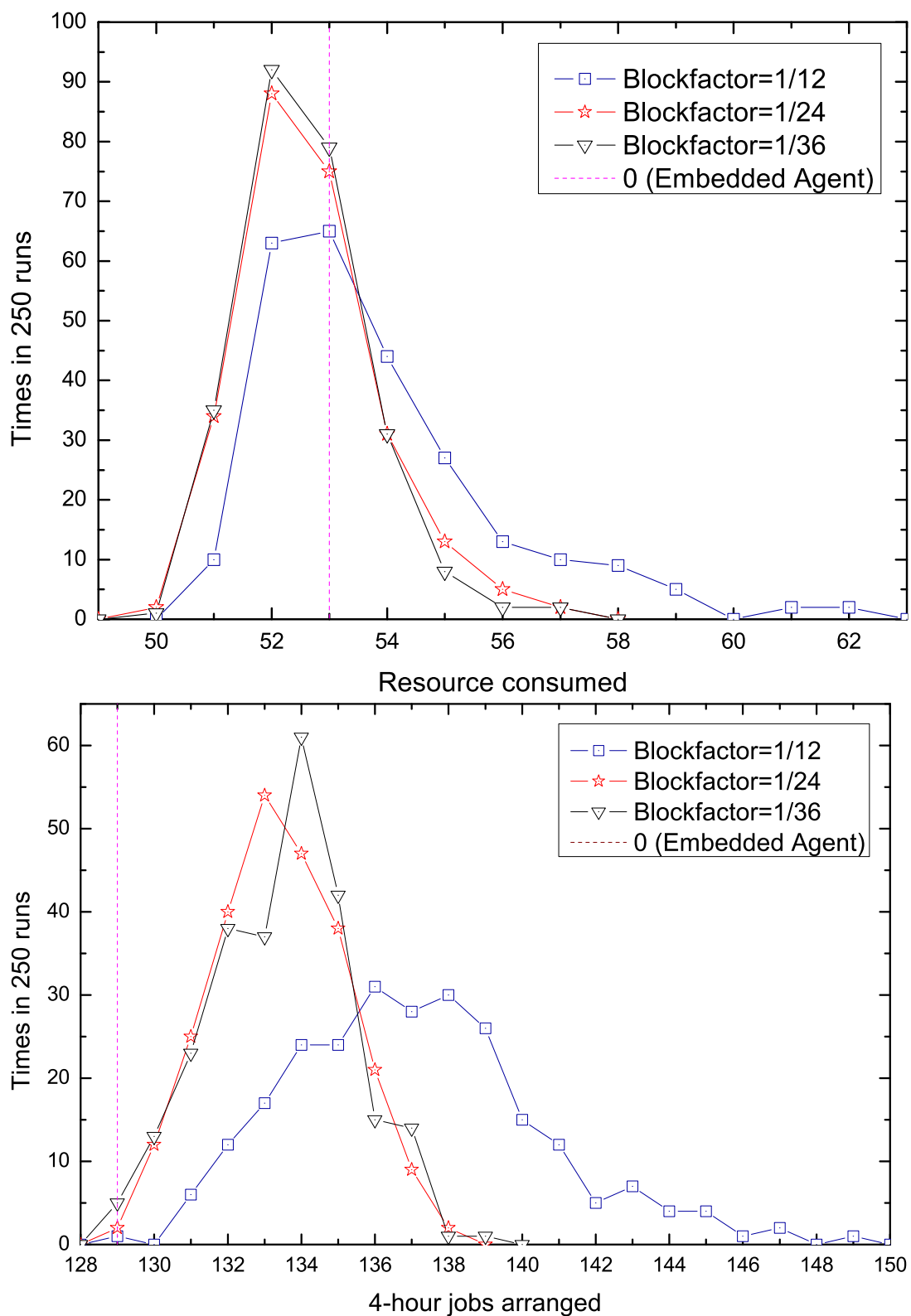


Figure 6-5: BT solution marginal distributions with respect to Blockfactor

hour jobs arranged decreases a bit.

- From 3D shapes and polyline shapes we could see that when the Blockfactor goes smaller, the solutions distributes more centralized, and the algorithm acts more similar as a stable algorithm.

In the description of DSAFO in Section 5.3, when an agent has nothing to do further, it blocks itself in time of $\text{Blockfactor} \times \text{Heartbeat}$. During this period, this agent will be awoken immediately when any message arrives. In other words, Blockfactor stands for how much time to block between agent activity cycles.

If there is only one BT *Member* agent, it has no buddies trying to to awake it. So it would keep idle until the time limit usually, unless *Coordinator*, *Blackboard* or *ResourceAdmin* sends message to it. Hence when Blockfactor is smaller, time to block is shorter, and the algorithm acts more like centralized heuristics. When Blockfactor is larger, the time to block is longer, the algorithm acts less like centralized heuristics.

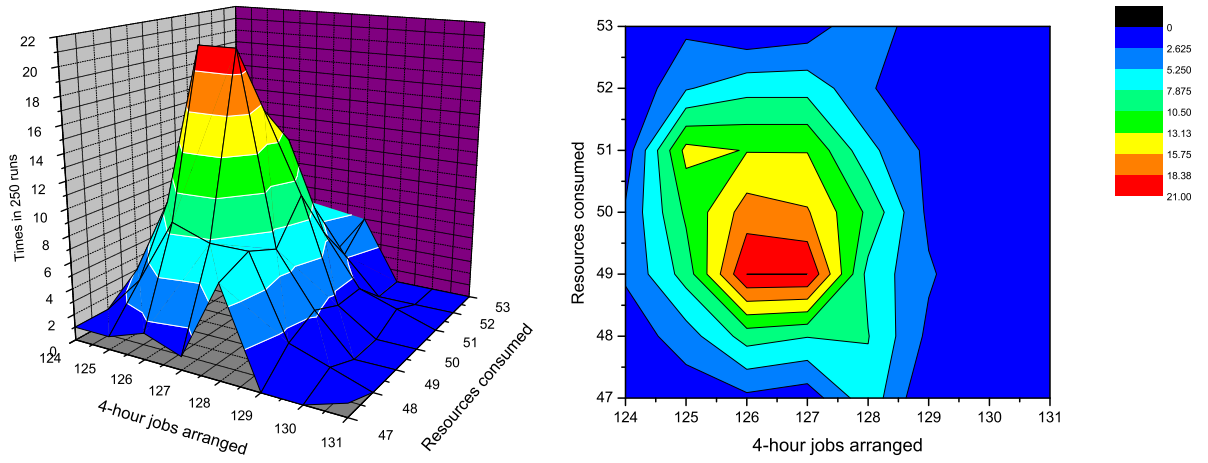
Why one *Member* agent with tiny Blockfactor (e.g. $1/36$) still outputs unstable solutions, anyway? The reason should be the network communications, which is the physical media of JADE ACL. Specially, we embedded one single *Member* agent in *Blackboard* agent, i.e. without any unstable factors from communication. The new embedded algorithm degraded to a deterministic algorithm as shown in Figure 6–5 (approximately same as EDD* in Chapter 7).

6.2.3 Delayfactor

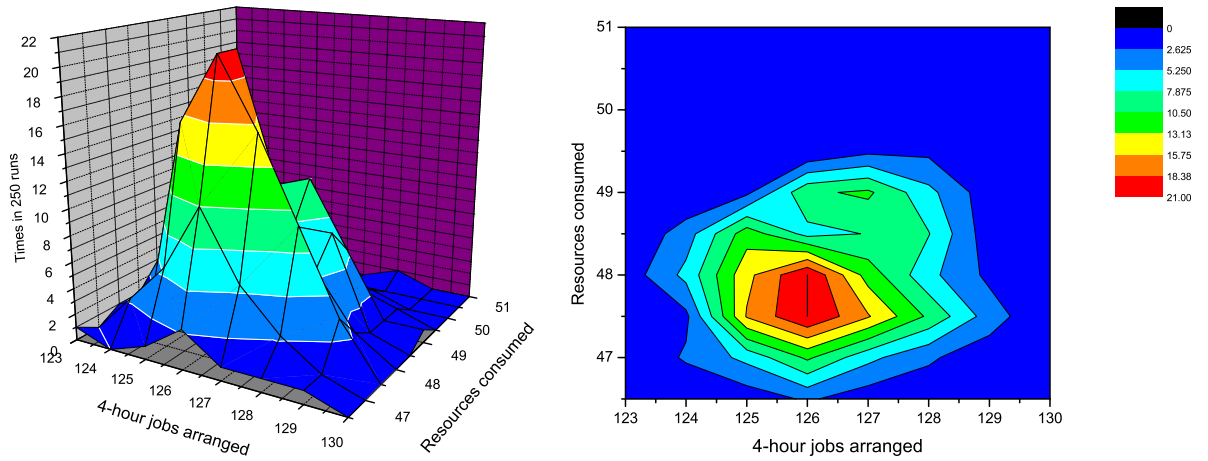
Delayfactor stands for how much time *Member* agent should delay extra after a successful operation commitment, which is not a very influential parameter. We set Agentnum=4, Blockfactor= $1/12$, Syncycle=5 and Reqcycle = $1/2$. Then we get distribution comparison in 3D map and contour map, as shown in Figure 6–6, as well as marginal distribution comparison with different Delayfactors in Figure 6–7.

No significant changes are there for 4-hour jobs, however the peaks of 3D map and contour map move up-down a bit, which usually means resource consumption is influenced. But the polylines of resource consumptions in Figure 6–7 deny this faint phenomenon: resources consumptions keep very close when Delayfactor changes.

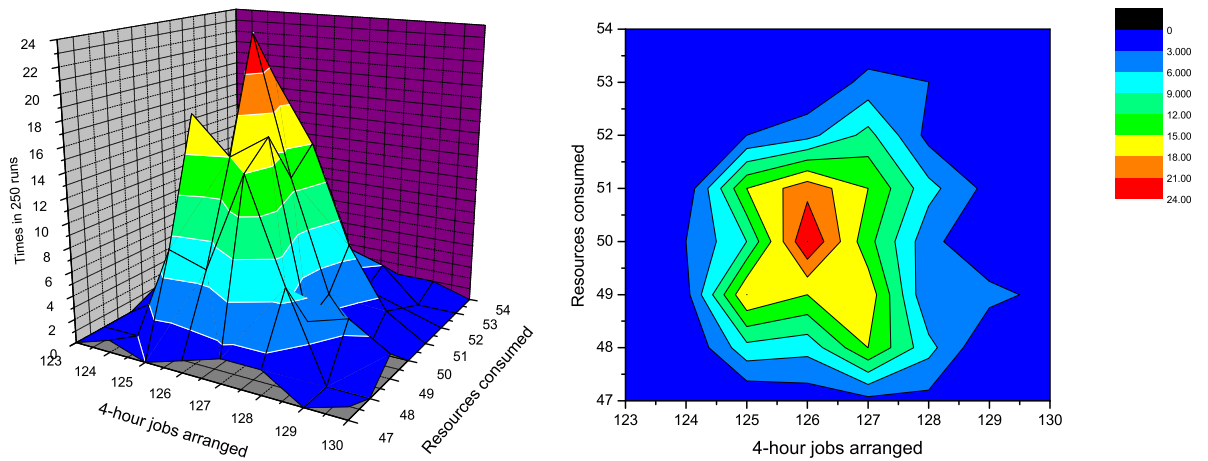
This delay is designed to prevent one *Member* agent in a group to request, to try and to commit operations exclusively, i.e. to keep load balance. If where is no extra delay



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



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



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

Figure 6-6: BT solution distributions with respect to Delayfactor

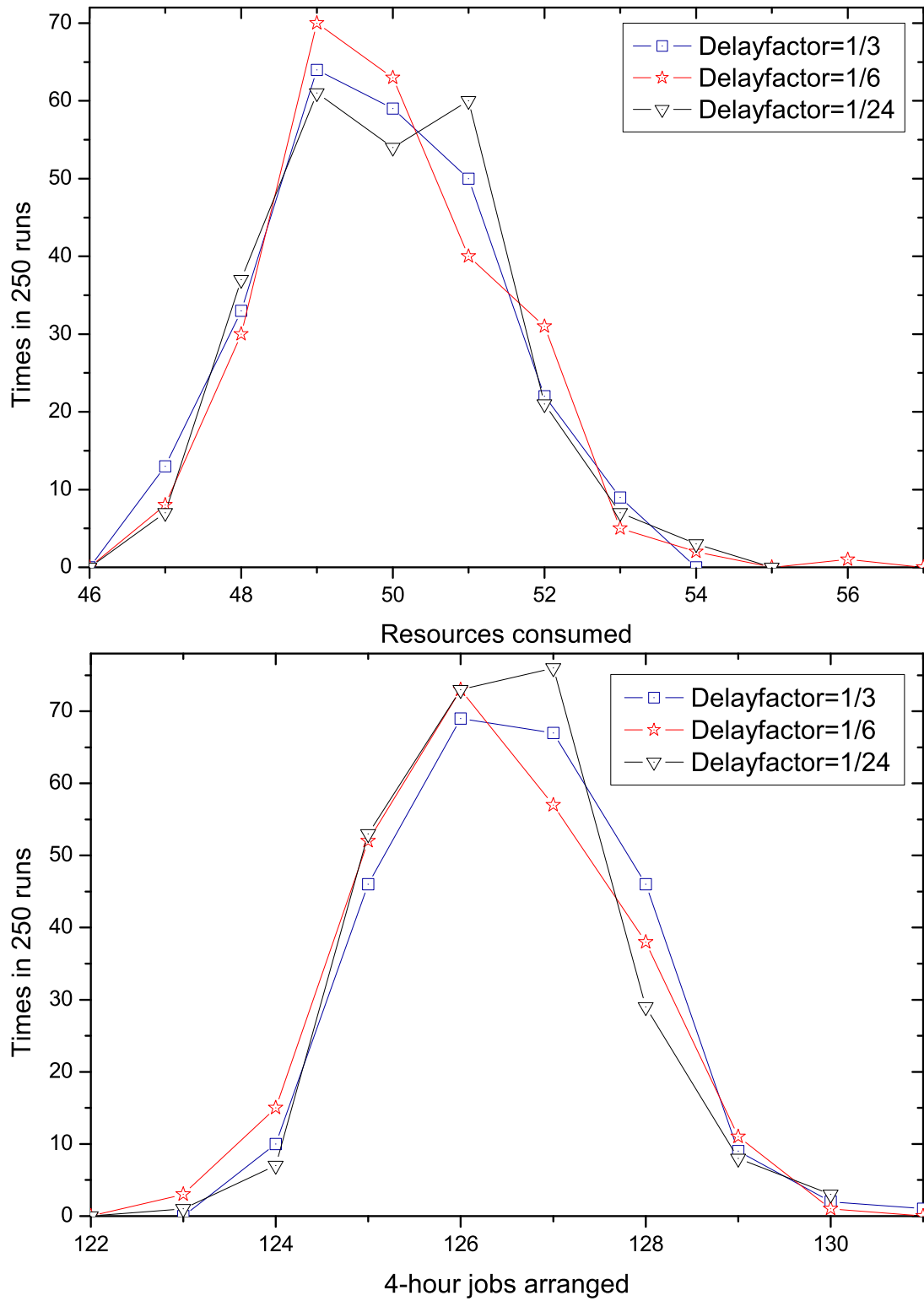


Figure 6-7: BT solution marginal distributions with respect to Delayfactor

and operation arrival is smooth, the *Member* agent, which successfully committed at the latest, must be the very agent which would gain the next operation in same type, before its resources running out. So, Delayfactor is designed to keep load balance, not to directly advance algorithm performance.

6.2.4 Syncycle

Syncycle stands for how much time *Member* and *Coordinator* should wait after successfully synchronizing remote resource textures, which is nor a very influential parameter. We set Agentnum=4, Blockfactor=1/12, Delayfactor=1/6 and Reqcycle = 1/2. Then we get distribution comparison in 3D map and contour map, as shown in Figure 6-8, as well as marginal distribution comparison with different Syncycles in Figure 6-9.

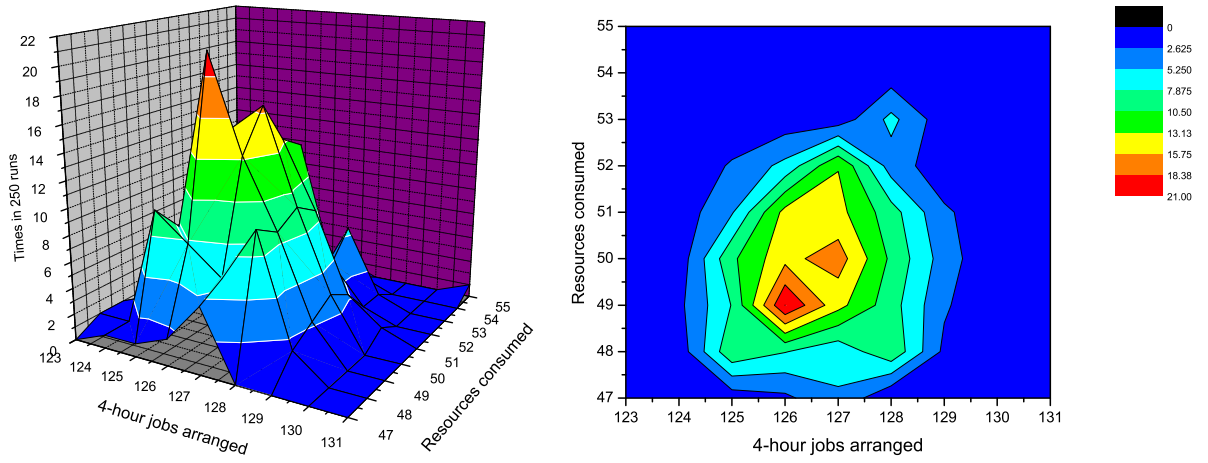
From these maps and charts, 4-hour jobs seems not significantly influenced by changing Syncycle; and nor does resource consumption.

The synchronization is designed to exchanging resource textures among *Member* agents, with help of *Coordinator* agent (*Coordinator* is very useful according to S. Abdallah, N. Darwish, and O. Hegazy (2002)[107]). So when the Syncycle is large, agents in DSAFO waits more time for next round of resource texture synchronization, i.e. the information kept by *Member* and *Coordinator* agents is more likely out-of-date. To call for help with a guidance of out-of-date information may not be a good idea. So theoretically, less Syncycle should aid algorithm better.

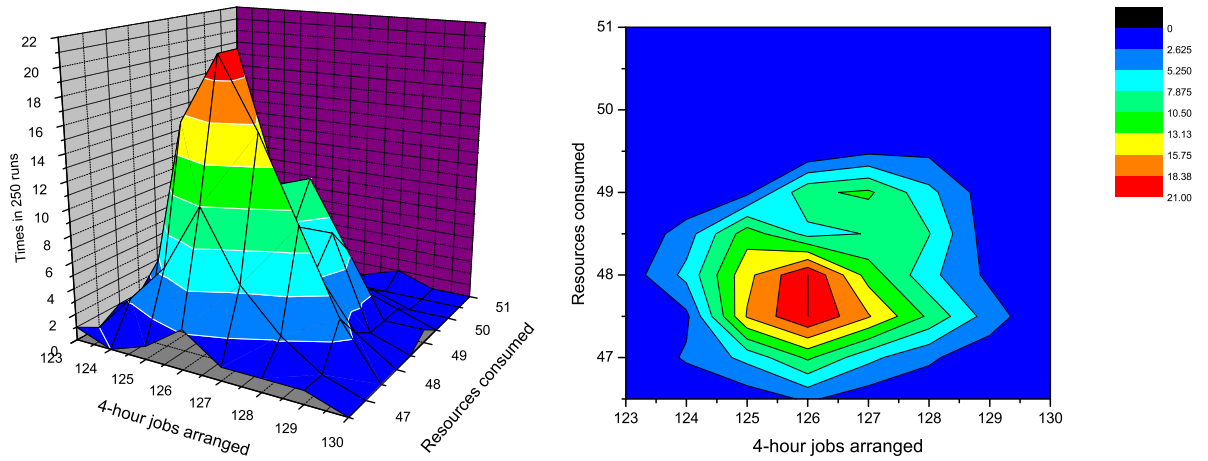
However, in real experiments, the cooperations among agents are far less than the number of total operations, and sometimes out-of-date information might guide the coordination better because of dynamic running. Whatever, this parameter affects algorithm practically far less than it is estimated theoretically.

6.3 Experimental summary

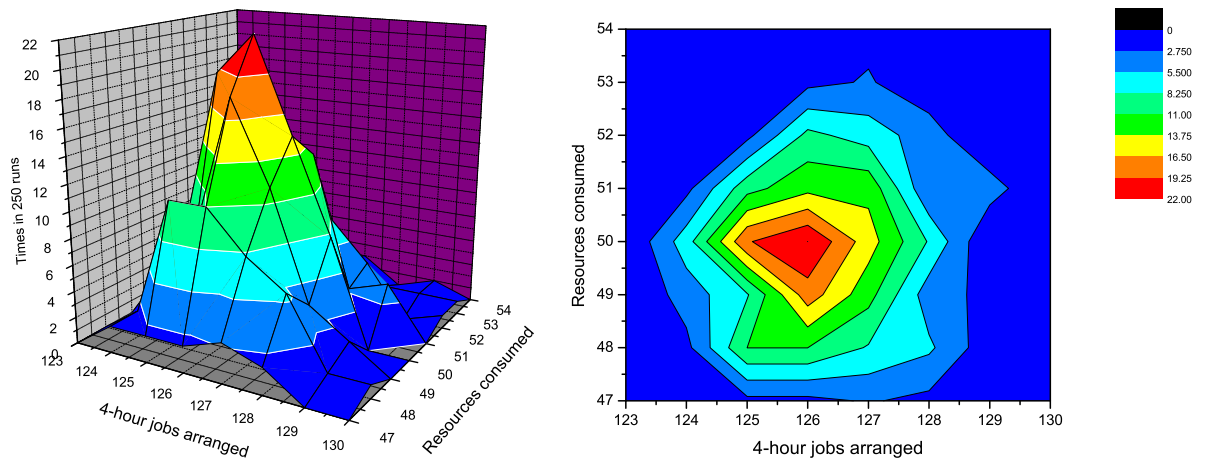
In this chapter, a number of experiments are presented. These results illustrate that DSAFO could successfully jump out of local minimum with dynamic agents. From another point of view, multi-agents split the global resource assignment into multi-partitions, each partition stands for an agent's view. To maximize the utilities of resources as much as possible, dynamic resource borrowing acts a key role after local heuristics. Each partition is dynamic and each resource in a partition is with near maximum utility, hence DSAFO



4 BT *Member* agents with Syncycle=3



4 BT *Member* agents with Syncycle=5



4 BT *Member* agents with Syncycle=15

Figure 6-8: BT solution distributions with respect to Syncycle

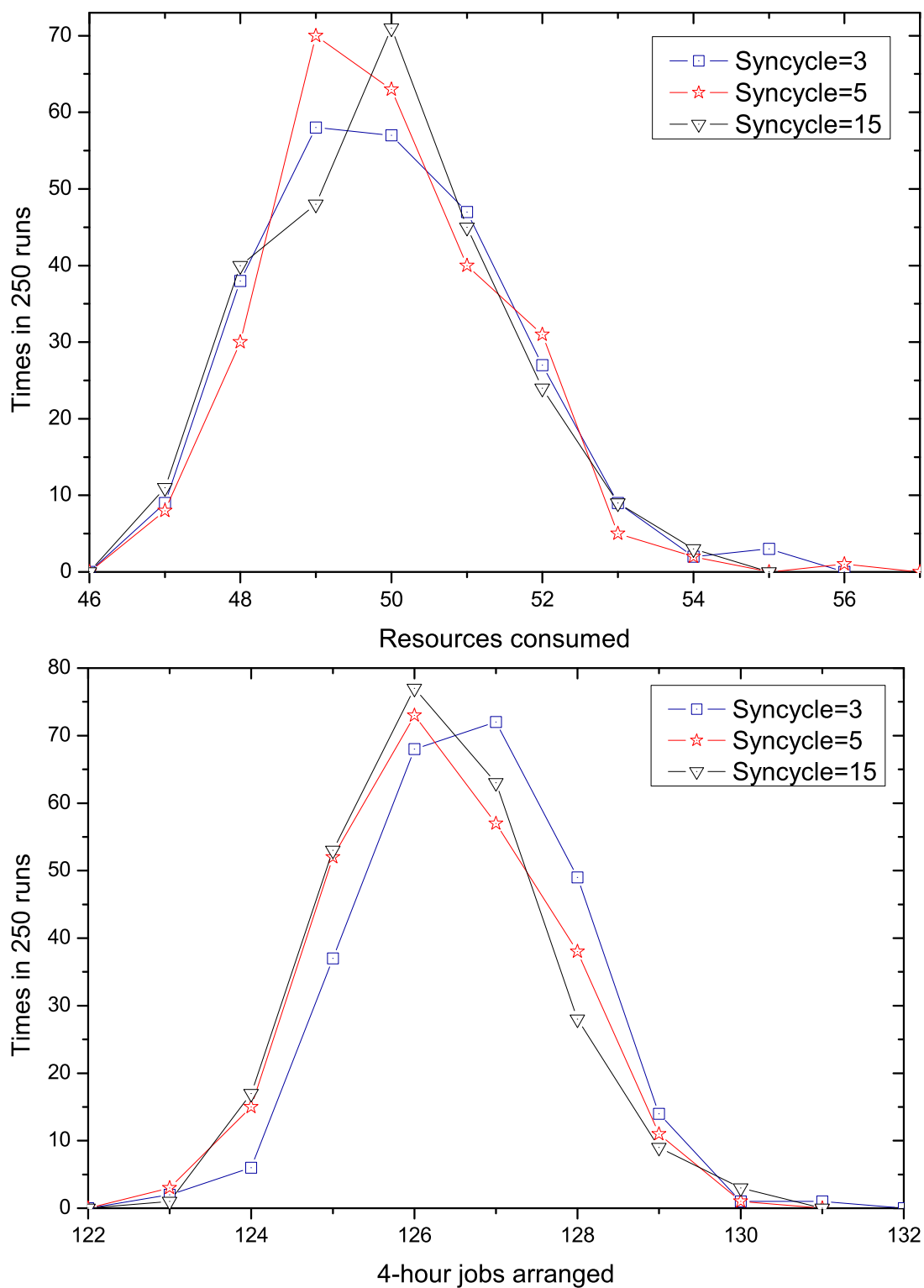


Figure 6-9: BT solution marginal distributions with respect to Syncycle

could successfully avoid being trapped in local minimum. And it is more natural to solve dynamic problem with dynamic agents. It is also able to adjust the indeterminacy of DSAFO by changing some parameters.

Many other difficult problems used to be troubled in local minimum, and they have been successfully optimized by many intelligent algorithms, such as Ant Colony, Particle Swarm Optimization, etc. DSAFO and other multi-agent algorithms may stand for another applicable direction to help these problems jump out of local minimum.

Chapter 7 Comparison

In this chapter, three opponent algorithms for AGSS problem are introduced in total: \mathcal{MMAS} , EDD* and ERT*; a solution comparison is given to compare DSAFO with them in solving AGSS problem.

7.1 Opponents

Three opponent algorithms for AGSS problem are introduced in the following subsections, they are \mathcal{MMAS} , EDD* and ERT*. \mathcal{MMAS} is an intelligent optimization algorithm which is implemented here to optimize AGSS problem statically, not dynamically; it is derived from ant system. And EDD* and ERT* are two traditional heuristics implemented in our dynamic scheduling environment *run-and-schedule*.

7.1.1 \mathcal{MMAS}

Ant system (AS) was proposed by M. Dorigo, V. Maniezzo and A. Coloni (1991) [142, 143]. Ant system was inspired by positive feedbacks in natural ant colony, in which each ant lays pheromone in its trail to help other company to choose correct path. Consequently ant systems are usually object to spatial problem such as Travel Salesman Problem (TSP).

Formally, an ant system for TSP consists of N_{ant} limited agents called “ant”, and a complete graph $\langle V, E \rangle$ and intensity of trails T (simulated pheromone) on whole E . In the $\langle V, E \rangle$ graph, V is a set of m vertices and E is the set of all edges between vertices. Every ant crawls from edge to edge to build a Hamilton circle¹ every time. To ensure a Hamilton circle, every ant has a tabu list to mark the visited vertices. And $\tau_{ij}(t)$ is the intensity of trail on edge $\langle i, j \rangle$,

$$\tau_{ij}(t+1) = \rho \cdot \tau_{ij}(t) + \Delta\tau_{ij}$$

¹A Hamilton circle is a close circle containing every vertex of a graph once and only once.

where ρ is a coefficient such that $(1 - \rho)$ represents the evaporation of trail,

$$\Delta\tau_{ij}(t, t+1) = \sum_{k=1}^{N_{\text{ant}}} \Delta\tau_{ij}^k$$

where $\Delta\tau_{ij}^k$ is the quantity per unit of length of trail substance (pheromone in real ants) laid on edge $\langle i, j \rangle$ by the k -th ant between time t and $t+1$; it is given by

$$\Delta\tau_{ij}^k = \begin{cases} Q/L_k & \text{if } k\text{-th ant uses edge } \langle i, j \rangle \text{ in its tour (between time } t \text{ and } t+1); \\ 0 & \text{otherwise.} \end{cases}$$

Visibility η_{ij} is defined as the quantity $1/d_{ij}$, where d_{ij} is the distance from vertex i to j . Finally, the transition probability from town i to town j for the k -th ant is:

$$p_{ij}^k = \begin{cases} \frac{[\tau_{ij}(t)]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{p \notin \text{tabu}_k} [\tau_{ip}(t)]^\alpha \cdot [\eta_{ip}]^\beta} & \text{if } j \notin \text{tabu}_k; \\ 0 & \text{otherwise.} \end{cases}$$

where α and β are parameters that control the relative importance of trail versus visibility.

Ant system have succeeded in many areas [143, 144] and \mathcal{MMAS} (\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 [145, 146]. \mathcal{MMAS} has been adopted to JSSPs since its proposal.

We implemented a \mathcal{MMAS} for BT arrangement to compare. We transform 1,008 BT related operations from 252 flights into 1,008 points. A distance between two operations r_i and r_j is represented as

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

This \mathcal{MMAS} is a little different from those for common TSP, because four types of BT related operations should be processed in sequential order. So we arrange every dependent UB-UC-LC-LB in order, as a result, the 1,008 points are arranged as Table 7.1.1:

And we define a boolean state $ready_i$ for each point $i \in V$. Apparently, only UB operations ($ready_{4k+1}, k \in \{0, \dots, 251\}$) are marked with *true* at first. Then we apply an extra recursive validate rule for ant after selecting edge $\langle i, j \rangle$:

$$\text{validate}(i, j) = \begin{cases} j, & \text{if } ready_j = \text{true}; \\ \text{validate}(i, j-1), & \text{otherwise.} \end{cases}$$

Table 7-1: Mapping BT operations to TSP points

Operations	UB ₁	UC ₁	LC ₁	LB ₁	...	UB _k	UC _k	LC _k	LB _k	...
Vertices	1	2	3	4	...	4k + 1	4k + 2	4k + 3	4k + 4	...

$(k \in \{0, \dots, 251\})$

Because $validate(i, j)$ is a recursive function, so the current ready operation in sequence UB-UC-LC-LB must be properly assigned before unready ones. And after choosing a point j , $ready_{j+1}$ becomes *true* unless j stands for a LB operation. As a result, the whole service scheduling could be performable.

And rest parameters of \mathcal{MMAS} are: $\alpha = 1.5$, $\beta = 2$, $\rho = 0.05$, $\tau_{init} = 1$, $\tau_{max} = 100$, $\tau_{min} = 0.01$, $N_{ant} = 200$, $NC_{max} = 150$. And for each successful ant run R done by ant i , all of edges in Hamilton circle of R get a positive feedback

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

to reinforce fewer resources and jobs.

7.1.2 EDD* and ERT*

EDD* here is simply to adapt Earliest Due Date first (EDD) to *run-and-schedule* environment. And ERT* is simply to adapt Earliest Ready Time first (ERT) to *run-and-schedule* environment as well. Main ideas of EDD and ERT are show in Figure 7-1.

7.2 Comparison

Then DSAFO (with parameters: agentNumber_{BT}=4, Blockfactor=1/12, Delayfactor=1/6, Syncycle=5), EDD*, ERT* and \mathcal{MMAS} were tested with 252 transfer flight data in Chapter 6. Once more we select BT related operations to test performances of these algorithms. A comparison in BT consumption and 4-hour BT jobs arrangement is shown in Table 7-2. Additionally we put time cost and average CPU rate in the table. The best value in each group is bolded.

From Table 7-2, it can be concluded that DSAFO and \mathcal{MMAS} both do well in resource consumption, 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

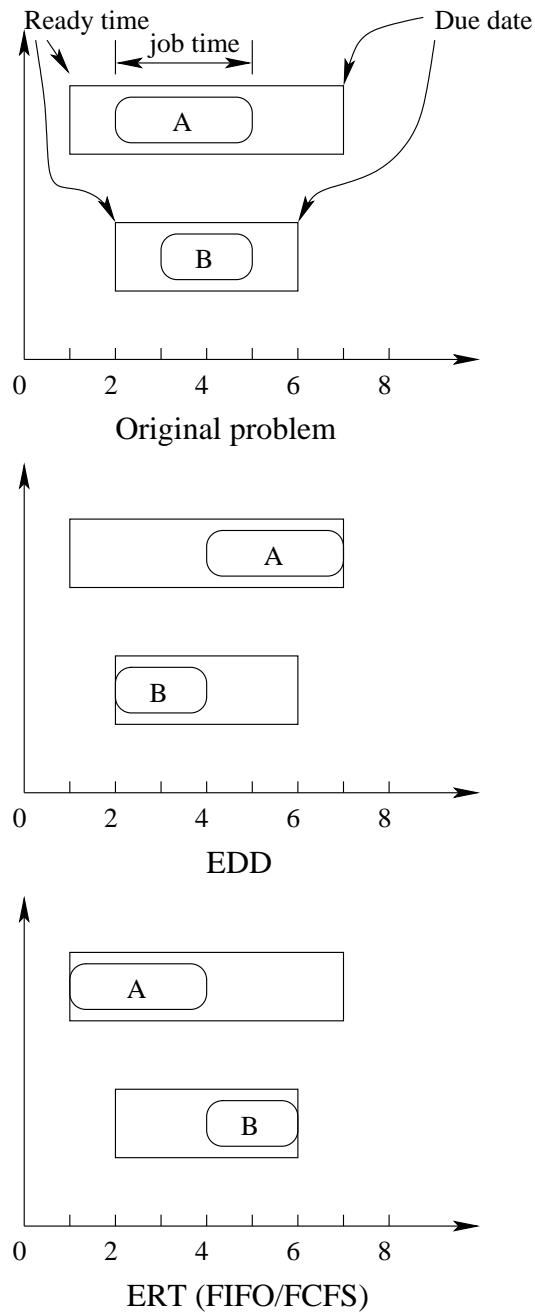


Figure 7-1: A simple illustration of EDD and ERT

Table 7-2: 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
$\mathcal{M}\mathcal{M}\mathcal{A}\mathcal{S}$	≈ 12 hours	$\approx 100\%$	47	—	—	141	—	—
EDD*	$\approx \mathbf{43}$ sec	$< 1\%$	53	53	53	129	129	129
ERT*	$\approx \mathbf{43}$ sec	$< 1\%$	52	52	52	130	130	130

time is used to maintain effective message transmission. On the contrary, $\mathcal{M}\mathcal{M}\mathcal{A}\mathcal{S}$ costs very much time and near 100% CPU rate.

Chapter 8 Conclusion and Future Works

8.1 Conclusion

AGSS problem is a typical difficult scheduling problem. Although it is \mathcal{NP} -hard to optimize consumptions of resources and man-days, AGSS problem is economically very important to airlines and airports, as well as passenger satisfaction.

DSAFO, a novel multi-agent constraint satisfaction algorithm for AGSS problems, is proposed in the thesis based on a dynamic distributed model and a dynamic scheduling environment *run-and-scheduling*. The complexity of DSAFO is bounded between quadratic and cubic polynomial time. So DSAFO is capable to solve large scale AGSS problem. Though experiments show that DSAFO is unstable and influenced by several parameters, this algorithm is good at satisfying all constraints, jumping out of local minimum, and finding near optimal solutions for consumption of resources and man-days.

Analysis shows that distributed heuristics with global coordination and dynamic multi-agent viewpoint in DSAFO is effective and promising in optimizing difficult problems in real applications.

8.2 Future works

One of the future works is to put forward an easy-to-use schedule software to make full use of algorithm DSAFO. A preliminary development environment AGSAP has been developed to apply DSAFO to aid common AGSS optimization by W. Fan, G. C. Zhang and F. Xue [147].

Furthermore, considering real-world management in airlines and airports, it is valuable to make scheduling with uncertainty of flights [148], such as uncertain AGSS satisfaction and problem. So how to generate robust (forecast) scheduling for uncertain flights (i.e. uncertain AGSS satisfaction and problem) should be investigated in the future.

Appendix A: Agent Communication Grammar in DSAFO

All agent messages in communications in DSAFO is based on strings which are sent via FIPA ACL. And each message consists of a channel mark, a formal comma, and message content, i.e.,

$$Message \stackrel{\text{def}}{=} ChannelMark, Content$$

When a *Message* is going to be sent, proper FIPA ACL Protocol should be chosen for it according to its *ChannelMark*, in order to avoid message collision. The mapping from *ChannelMark* to FIPA ACLs is pre-assumed as follow:

Table A–1: ACL protocols and agent communication grammar in DSAFO

<i>ChannelMark</i>	Protocol	<i>Content</i>
QUERY	INFORM	OpType
INFORM	INFORM	FlightNo, TaskType, ERT, LED, jobTime, Apron
REQUEST	INFORM	FlightNo, TaskType, MetaPlan(arg ₁ ; arg ₂ ; ...)
REPLY	INFORM	FlightNo, TaskType, MetaPlan(arg ₁ ; arg ₂ ; ...), ERT
CANCEL	INFORM	MetaPlan(arg ₁ ; arg ₂ ; ...)
INVALID	INFORM	MetaPlan(arg ₁ ; arg ₂ ; ...), isFromBB
BORROW	INFORM	ResType, MetaPlan(arg ₁ ; arg ₂ ; ...), BuddyList(Name ₁ ; ...)
LEND	INFORM	ResType, MetaPlan(arg ₁ ; arg ₂ ; ...)
REFUSE	INFORM	ResType, MetaPlan(arg ₁ ; arg ₂ ; ...), BuddyList(Name ₁ ; ...)
ACQUIRE	INFORM	ResType, StartTime
ALLOT	INFORM	ResName, StartTime
RELEASE	INFORM	ResType, ResName, ReleaseTime
SYNBUDDY	PROXY	< null >
ACKBUDDY	PROXY	BuddyList(Name ₁ ; Name ₂ ; ...)
SYNDEMAND	REQUEST	< null >
ACKDEMAND	REQUEST	myResFreedom

References

- [1] M. R. Garey, D. S. Johnson. (1979). Computers and intractability - a guide to the theory of NP-completeness[M]. W.H. Freeman and Company, New York.
- [2] M. S. Fox. (1983). Constraint-Directed Search: A case study of job-shop scheduling[D]. Doctoral dissertation, tech. report CMU-RI-TR-83-22, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, December, 1983.
- [3] D. Applegate, and W. Cook. (1991). A computational study of the job-shop scheduling problem[J]. ORSA Journal On Computing, 3:149 - 156, 1991.
- [4] H. Hartmann. (2001). CARE action innovation: final report of preliminary study total airport management[R]. Technical Report, German Aerospace Center, IB 112-2001/21, DLR, Institut für Flugführung, Germany November 2001.
- [5] J. Xing, S. Liu, W. Fan, L. Ji. (2006). Design of airport ground service system based on multi-agent[J]. *Journal of Civil Aviation University of China*. 24 (3) (2006) pp. 24–27 (邢建民, 刘绍华, 樊玮, 计雷. 基于多Agent的飞机地面作业系统设计[J]. 中国民航学院学报, 24(3): 24–27.)
- [6] P. Baptiste, C. Le Pape, and W. Nuijten. (1995). Incorporating efficient operations research algorithms in constraint-based scheduling[C]. In First International Joint Workshop on Artificial Intelligence and Operations Research, 1995.
- [7] General Administration of Civil Aviation of China. (2006). First season flight normal rate in China civil aviation[R/OL]. [http://www.caac.gov.cn, /E.PubWebApp/Doc/04/20060427154753.doc](http://www.caac.gov.cn/E.PubWebApp/Doc/04/20060427154753.doc). 27th, Apr. 2006. (中国民航总局. 民航第一季度航班正常率[R/OL]. <http://www.caac.gov.cn>. 2006年4月27日.)
- [8] L. Shi. (2005). Cost control in fleet planning. Chinese Civil Aviation Management, 2005 No 2, pp. 24–25. (石丽娜. 机队规划中的成本控制[J]. 民航管理, 2005(2): 24–25.)
- [9] S.-H. Tsaur, T.-Y. Chang and C.-H. Yen. (2002) The evaluation of airline service quality by fuzzy MCDM[J]. *Tourism Management*, Vol. 23, No.2, pp. 107–155, 2002.

- [10] Y. Hu. (2002). Academician Guojie Li: a talk on computer development strategy in China[N]. Guangming Daily, 4th, Jan. 2002. (胡永生. 李国杰院士: 我国计算机发展战略纵横谈[N]. 光明日报, 2002年1月4日.)
- [11] D. E. Neiman, D. W. Hildum, V. R. Lesser, T. W. Sandholm. (1994). Exploiting meta-level information in a distributed scheduling system[C]. In Proceedings of the Twelfth National Conference on Artificial intelligence (AAAI-94) Vol.1 Seattle, Washington, US. American Association for Artificial Intelligence, Menlo Park, CA (1994) 394–400.
- [12] D. W. Hildum. (1994). Flexibility in a knowledge-based system for solving dynamic resource-constrained scheduling problems[D]. PhD dissertation, Computer Science Dept., University of Massachusetts, Amherst, MA 01003, May 1994. UMI Order No. GAX95-10483.
- [13] D. E. Neiman, V. R. Lesser. (1996). A cooperative repair method for a distributed scheduling system[C]. In *Proceedings of the Third International Conference on Artificial Intelligence Planning System (AIPS-96)*, Edinburgh, Scotland. (1996) 174–181.
- [14] M. Chia, D. E. Neiman, V. R. Lesser. (1998). Coordinating asynchronous agent activities in a distributed scheduling system[C]. In *Proceedings of the Second International Conference on Autonomous Agents (Agents98)*, January, 1998.
- [15] A. Chenung, W. H. Ip, D. Lu, C. L. Lai. (2005). An aircraft service scheduling model using genetic algorithms[J]. Journal of manufacturing technology management. 16(1): 109–119
- [16] M. Xu and Z. X. Wang. (2003). A novel intelligent vehicle displaying and scheduling system[J]. Chinese Automation Information, 2003(1), No.31, pp. 29–31. (徐猛, 王宗学. 新型智能车辆监控调度系统[J]. 自动化信息, 2003(1), 总第 31 册: 29–31.)
- [17] A. S. Manne. (1960). On the job shop scheduling problem[J]. Operations Research, Vol. 8(2) March, 1960. pp. 219–223.
- [18] A. S. Jain and S. Meeran. (1999). Deterministic job-shop scheduling: past, present and future[J]. European Journal of Operational Research, Vol. 113(2), pp. 390–434.
- [19] A. Jones and L. C. Rabelo. (1998). Survey of job shop scheduling techniques[R]. Technical Reports. NISTIR, National Institute of Standards and Technology, Gaithersburg, MD, 1998.
- [20] D. Wu. (1987). An Expert Systems Approach for the Control and Scheduling of Flexible Manufacturing Systems[D]. Ph.D. Dissertation, Pennsylvania State University, PA, US.

- [21] S. F. Smith. (1994). OPIS: A methodology and architecture for reactive scheduling[M]. In M. Zweben and M. Fox (Eds.), *Intelligent scheduling*, San Francisco, CA: Morgan Kaufmann.
- [22] H. V. D. Parunak, B. W. Irish, J. Kindrick, and P. W. Lozo. (1985). Fractal actors for distributed manufacturing control[C], in *The Second Conference on Artificial Intelligence Applications*, Miami, December 1985, pp. 653–660.
- [23] P. S. Ow, S. F. Smith, R. Howie. (1988). A cooperative scheduling system[M], in: M.D. Oliff (ed.), *Expert System and Intelligent Manufacturing*, pp. 43–56.
- [24] K. Sycara, S. Roth, N. Sadeh, and M. Fox. (1991). Distributed constrained heuristic search[J]. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6):1446–1461, November/December 1991.
- [25] J. Butler, H. Ohtsubo. (1992). ADDYMS: architecture for distributed dynamic manufacturing scheduling[M], in: A. Famili, D. S. Nau, S. H. Kim, eds., *Artificial Intelligence Applications in Manufacturing*, AAAI Press/MIT Press, pp. 199–213.
- [26] L. C. Rabelo. (1990). A hybrid artificial neural networks and knowledge-based expert systems approach to flexible manufacturing system scheduling[D]. Doctoral Thesis. University of Missouri-Rolla.
- [27] F. Glover. (1989). Tabu search: part I[J]. *ORSA Journal on Computing*, Vol. 1(3), pp. 190–206.
- [28] F. Glover. (1990). Tabu search: part II[J]. *ORSA Journal on Computing* Vol. 2(1), pp. 4–32.
- [29] F. Glover. (1996). Tabu search and adaptive memory programming: advances, applications and challenges[M], in *Interfaces in Computer Science and Operations Research*, Barr, Helgason and Kennington (eds.) Kluwer Academic Publishers, pp. 1–75. 1996.
- [30] A. Vakharia and Y. Chang. (1990). A simulated annealing approach to scheduling a manufacturing cell.[J] *Naval Research Logistics*. Vol. 37, pp. 559–577.
- [31] P. J. van Laarhoven, E. H. Aarts, J. K. Lenstra. (1992). Job shop scheduling by simulated annealing[J]. *Operations Research*. 40(1) (Jan. 1992), pp. 113–125.
- [32] L. Davis. (1985). Job shop scheduling with genetic algorithms[C]. In *Proceedings of the 1st international Conference on Genetic Algorithms*. J. J. Grefenstette, Ed. Lawrence Erlbaum Associates, Mahwah, NJ, pp. 136–140.

- [33] T. Starkweather, D. Whitley and B. Cookson. (1993). A Genetic Algorithm for scheduling with resource consumption[C]. in *the Joint German/US Conference on Operations Research in Production Planning and Control*, G. Fandel, T. Gullledge and A. Jones (Eds.) Operation Research in Production Planning and Control, Springer-Verlag, Berlin, 1993, pp. 567–583.
- [34] A. Colorni, M. Dorigo, V. Maniezzo, M. Trubian. (1994). Ant system for job-shop scheduling[J]. *Belgian Journal of Operations Research, Statistics and Computer Science*, 34(1), pp. 39–53.
- [35] W. J. Xia, Z. M. Wu, W. Zhang, G. Yang. (2004). Applying particle swarm optimization to job-shop scheduling problem[J]. *Chinese Journal of Mechanical Engineering*, 17(3), pp.437–441.
- [36] Y. Tsujimura, S. H. Park, I. S. Chang, and M. Gen. (1993). An effective method for solving flow shop scheduling problems with fuzzy processing times[C]. In *Proceedings of the 15th Annual Conference on Computers and industrial Engineering*, Blacksburg, Virginia, United States. C. P. Koelling, Ed. Pergamon Press, Elmsford, NY, pp. 239–242.
- [37] W. Slany. (1996). Scheduling as a fuzzy multiple criteria optimization problem[J]. *Fuzzy Sets and Systems*. Vol 78(2), March 1996, pp. 197–222. Issue 2. Special Issue on Fuzzy Multiple Criteria Decision Making.
- [38] M. Yokoo and K. Hirayama. (2000). Algorithms for distributed constraint satisfaction: a review[J]. *Autonomous Agents and Multi-Agent Systems*, Vol 3(2), pp. 185–207, 2000.
- [39] S. Minton , M. D. Johnston , A. B. Philips , P. Laird. (1992). Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems[J], *Artificial Intelligence*, Vol.58(1-3), pp.161–205, Dec. 1992.
- [40] M. Yokoo. (1994). Weak-commitment search for solving constraint satisfaction problems[C]. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI '94)*, AAAI press, Vol. 1, pages 313–318, Seattle, WA, USA, July 31 - August 4 1994.
- [41] A. K. Mackworth. (1992). Constraint satisfaction[M]. In: S. C. Shapiro (ed.): *Encyclopedia of Artificial Intelligence*. New York: Wiley-Interscience Publication, pp. 285–293.
- [42] M. Yokoo, E. H. Durfee, T. Ishida, K. Kuwabara. (1998). The distributed constraint satisfaction problem: formalization and algorithms[J]. *IEEE Transactions on Knowledge and Data Engineering* 10(5) (Sep. 1998), pp. 673–685.

-
- [43] M. Yokoo, E. Durfee, T. Ishida, and K. Kuwabara. (1992). Distributed constraint satisfaction for formalizing distributed problem solving[C], in Proceedings of the Twelfth IEEE International Conference on Distributed Computing Systems (ICDCS-92), Yokohama, Japan, June 1992. pp. 614–621.
- [44] M. Yokoo. (1995). Asynchronous weak-commitment search for solving constraint satisfaction problems[C]. In *First International Conference on Principles and Practice of Constraint Programming* (CP-95) Cassis, France. U. Montanari, F. Rossi. Eds, Lecture Notes in Computer Science Vol.976 Springer, (1995) pp. 407–422.
- [45] M. Yokoo. (2001). Distributed constraint satisfaction: foundation of cooperation of multi-agent system[M]. Springer Verlag, Berlin.
- [46] P. Prosser, C. Conway, and C. Muller. (1992). A constraint maintenance system for the distributed allocation problem[J]. *Intelligent Systems Engineering* Vol.1(1), pp. 76–83.
- [47] P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo. (2005). An asynchronous complete method for distributed constraint optimization[J]. *Artificial Intelligence Journal*, vol.161(1-2), pp.149–180, 2005.
- [48] Yokoo, M. (2004). Protocol/mechanism design for cooperation/competition[C]. In Proceedings of the Third international Joint Conference on Autonomous Agents and Multiagent Systems - Vol. 1 (New York, New York, July 19 - 23, 2004). International Conference on Autonomous Agents. IEEE Computer Society, Washington, DC, pp. 3–7.
- [49] G. Weiss, (ed.). (1999). Multiagent systems: a modern approach to distributed artificial intelligence[M]. The MIT Press, Cambridge, Massachusetts, 1999
- [50] M. Wooldridge, and N. Jennings. (1995). Intelligent agents: theory and practice[J]. *Knowledge Engineering Review*, Vol.10, No. 2, 1995. Cambridge University Press, pp. 115–152
- [51] M. J. Wooldridge. (2001). Introduction to multiagent systems[M]. John Wiley & Sons, Inc. (See also: 多Agent系统引论[M]. 石纯一,等译. 北京:电子工业出版社, 2003.10.)
- [52] D. Gelernter and N. Carriero. (1992). Coordination languages and their significance[J]. *Communication of the ACM*, Vol.35(2) (Feb. 1992), pp. 97–107.
- [53] P. Ciancarini, A. Omicini, and F. Zambonelli. (2000). Multiagent system engineering: the coordination viewpoint[C]. In 6th international Workshop on intelligent Agents Vi, Agent theories, Architectures, and Languages (Atal), (July 15 - 17, 1999). N. R. Jennings and Y. Lespérance, Eds. Lecture Notes In Computer Science, vol. 1757. Springer-Verlag, London, 250–259.

-
- [54] H. S. Nwana. (1996). Software agents: an overview[J]. *The Knowledge Engineering Review*, 11(3): 205–244, October/November 1996.
- [55] M. N. Huhns, and M. P. Singh. (1994). Distributed Artificial Intelligence for Information Systems[R]. CKBS-94 Tutorial, June 15, University of Keele, UK.
- [56] K. S. Decker and V. R. Lesser. (1993). Analyzing a quantitative coordination relationship[J]. *Group Decision and Negotiation*, Vol. 2(3):195–217, 1993.
- [57] K. Decker, and J. Li. (1998). Coordinated Hospital Patient Scheduling[C]. In *Proceedings of the 3rd international Conference on Multi Agent Systems (July 03 - 07, 1998)*. ICMAS-98. IEEE Computer Society, Washington, DC, 104.
- [58] M. N. Huhns, and L. M. Stephens. (1999). Multiagent systems and societies of agents[M]. In *Multiagent Systems: A Modern Approach To Distributed Artificial intelligence*, G. Weiss, Ed. MIT Press, Cambridge, MA, 79–120.
- [59] J. S. Liu. (1996). Coordination of multiple agents in distributed manufacturing scheduling[D]. Doctoral Thesis , The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, April 1996.
- [60] N. R. Jennings. (1991). Cooperation in industrial systems[C]. In *Proceedings of ESPRIT Conference*, pp. 253–263, Brussels, Belgium.
- [61] N. R. Jennings. (1994). The ARCHON system and its applications[C]. In *Proceedings of 2nd Int. Conf. on Cooperating Knowledge Based Systems (CKBS-94)*, pages pp. 13–29, Keele, UK.
- [62] R. Nair, and M. Tambe. (2005). Hybrid BDI-POMDP framework for multiagent teaming[J]. *Journal of AI Research (JAIR)*, 23:367–413, 2005.
- [63] N. Schurr, P. Scerri, and M. Tambe. (2004). Coordination advice: a preliminary investigation of human advice to multiagent teams[C]. in *AAAI Spring Symposium on Interaction between Humans and Autonomous Systems over Extended Operation*, Invited Paper, 2004.
- [64] V. A. Cicirello and S. F. Smith. (2001). Wasp-like agents for distributed factory coordination[R]. Technical Report CMU-RI-TR-01-39, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, December 2001.
- [65] L. Panait, and S. Luke. (2005). Cooperative multi-agent learning: the state of the art[J]. *Autonomous Agents and Multi-Agent Systems*, Vol. 11, No. 3. (November 2005), pp. 387–434.

- [66] P. E. Utgoff, D. Jensen, and V. Lesser. (2000). Inferring Task Structure from Data[R]. Technical Report. UMI Order Number: UM-CS-2000-054., University of Massachusetts.
- [67] Reis, L. P., Lau, N., and Oliveira, E. (2001). Situation based strategic positioning for coordinating a team of homogeneous agents[J]. In *Balancing Reactivity and Social Deliberation in Multi-Agent Systems, From Robocup To Real-World Applications (Selected Papers From the ECAI 2000 Workshop and Additional Contributions)* M. Hannebauer, J. Wendler, and E. Pagello, Eds. Lecture Notes In Computer Science, vol. 2103. Springer-Verlag, London, pp. 175–197.
- [68] R. G. Smith. (1977). The CONTRACT NET: a formalism for the control of distributed problem solving[C]. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence (IJCAI-77)*, Cambridge, MA, February 1977. pp. 338–343.
- [69] R. G. Smith. (1980). The contract net protocol: high-level communication and control in a distributed problem solver[J]. *IEEE Transactions on Computers*, vol. C-29(12), pp. 1104–1113, Dec. 1980.
- [70] H. Chalupsky, T. Finin, R. Fritzson, D. McKay, S. Shapiro, and G. Wiederhold. (1992). An overview of KQML: a knowledge query and manipulation language[R]. Technical report, KQML Advisory Group, April 1992.
- [71] T. Finin, R. Fritzson, D. McKay, and R. McEntire. (1994). KQML as an agent communication language[C]. In *Proceedings of the Third international Conference on information and Knowledge Management (Gaithersburg, Maryland, United States, November 29 - December 02, 1994)*. N. R. Adam, B. K. Bhargava, and Y. Yesha, Eds. CIKM '94. ACM Press, New York, NY, pp. 456–463.
- [72] FIPA. (1997). FIPA 1997 specification part 2: agent communication language[S/OL]. Document No. 00003. Geneva:FIPA Foundation for Intelligent Physical Agents, October 1997. <http://www.fipa.org/specs/fipa00003/0C00003A.pdf>
- [73] FIPA. (2002). FIPA ACL message structure specification[S/OL]. Document No. 00061. Geneva: FIPA Foundation for Intelligent Physical Agents. <http://www.fipa.org/specs/fipa00061/SC00061G.pdf>
- [74] V. R. Lesser, and D. D. Corkill. (1981). Functionally accurate, cooperative distributed problem-solving systems[J]. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-11(1): 81–96, January 1981.

- [75] V. R. Lesser. (1991). A retrospective view of FA/C distributed problem solving[J]. IEEE Transactions on Systems, Man, and Cybernetics, Special Issue on Distributed Artificial Intelligence, 21(6):1347–1362, December 1991.
- [76] D. E. Wilkins. (1988). Practical planning: extending the classical AI planning paradigm[M]. Morgan Kaufmann Publishers Inc., San Mateo, CA, 1988.
- [77] Durfee, E. H. and Lesser, V. R. September. (1991). Partial global planning: a coordination framework for distributed hypothesis formation[J]. IEEE Transactions on Systems, Man, and Cybernetics, Special Issue on Distributed Sensor Networks, SMC-21(5):1167–1183.
- [78] T. Wittig, Ed. (1992). Archon: an architecture for multi-agent systems[M]. Ellis Horwood Series in Artificial Intelligence. Ellis Horwood.
- [79] N. R. Jennings, J. Corera, I. Laresgoiti, E. H. Mamdani, F. Perriolat, P. Skarek, and L. Z. Varga. (1996). Using ARCHON to develop real-world DAI applications for electricity transportation management and particle acceleration control[J]. IEEE Expert, Vol. 11(6) pp. 60–88, December 1996. Special Issue on Real World Applications of DAI systems.
- [80] K. Decker, and V. R. Lesser. (1995). Designing a family of coordination algorithms[C]. In Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), pages 73–80, San Francisco, CA, June 1995.
- [81] M. desJardins, M. Wolverton. (1999). Coordinating a Distributed Planning System, Artificial Intelligence Magazine, 20(4), pp.45–53, Winter, 1999.
- [82] K. Decker, and V. R. Lesser. (1993). Quantitative modeling of complex environments[C]. In International Journal of Intelligent Systems in Accounting, Finance and Management. Special Issue on Mathematical and Computational Models and Characteristics of Agent Behavior., Vol. 2, pp. 215–234, 1993.
- [83] T. Wagner, A. Garvey, and V. Lesser. (1997). Complex goal criteria and its application in design-to-criteria scheduling[R]. Technical Report. UMI Order Number: UM-CS-1997-010., University of Massachusetts.
- [84] K. Decker, and J. Li. (2000). Coordinating mutually exclusive resources using GPGP[J]. Autonomous Agents and Multi-Agent Systems, Vol. 3(2) (Jun. 2000), 133–157.
- [85] V. A. Cicirello, and S. F. Smith. (2004). Wasp-like agents for distributed factory coordination[J]. Autonomous Agents and Multi-Agent Systems 8(3): 237–266.
- [86] M. d’Inverno, D. Kinny, M. Luck, and M. Wooldridge. (1997). A formal specification of dMARS[C]. In Proceedings of the 4th international Workshop on intelligent Agents Iv,

- Agent theories, Architectures, and Languages (July 24 - 26, 1997). M. P. Singh, A. S. Rao, and M. Wooldridge, Eds. Lecture Notes In Computer Science, vol. 1365. Springer-Verlag, London, 155–176.
- [87] P. R. Cohen, and H. J. Levesque. (1991). Teamwork[J]. *Nous*, 25(4): 487–512. Special Issue on Cognitive Science and Artificial Intelligence.
- [88] N. R. Jennings. (1993). Controlling cooperative problem solving using joint intentions[J]. *AI Communications*, Vol. 6(3-4): 247–428.
- [89] J. Broersen, M. Dastani, J. Hulstijn Z. Huang and L. van der Torre. (2001). The BOID architecture: conflicts between beliefs, obligations, intentions and desires[C]. In *Proceedings of the Fifth international Conference on Autonomous Agents* (Montreal, Quebec, Canada). AGENTS '01. ACM Press, New York, NY, 9–16.
- [90] M. E. Bratman. (1987). Intention plans and practical reason[M]. Harvard University Press, Cambridge, MA, 1987.
- [91] D. Ancona, and V. Mascardi. (2003). Coo-BDI: Extending the BDI Model with Cooperativity, in Leite, J. Omicini, A.. L. Sterling, and P. Torroni editors, Declarative agent languages and techniques[C], First International Workshop, DALT 2003, Revised Selected and Invited Papers, Lecture Notes in Computer Science 2990, pages 109–134, Springer-Verlag, 2004.
- [92] D. Ancona, V. Mascardi, J. F. Hubner, and R. H. Bordini. (2004). Coo-AgentSpeak: cooperation in agentspeak through plan exchange[C]. In *Proceedings of the Third international Joint Conference on Autonomous Agents and Multiagent Systems - Volume 2* (New York, New York, July 19 - 23, 2004). International Conference on Autonomous Agents. IEEE Computer Society, Washington, DC, pp. 696–705.
- [93] S. Soon, A. Pearce, and M. Noble. (2004). Adaptive teamwork coordination using graph matching over hierarchical intentional structures[C]. In *Proceedings of the Third international Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS04) - Volume 1* (New York, New York, July 19 - 23, 2004). International Conference on Autonomous Agents. IEEE Computer Society, Washington, DC, pp. 294–301.
- [94] G. Chen, Z. Yang, H. He, K. M. Goh. (2005). Coordinating multiple agents via reinforcement learning[J]. In *Autonomous Agents and Multi-Agent Systems*, 10(2): 273–328, May, 2005.
- [95] L. A. Zadeh, R. R. Yage, and R. M. Ton (eds.). (1987). Fuzzy sets and applications: Selected Papers[M], John Wiley and Sons, New York, 1987.

- [96] C. Boutilier. (1999). Sequential optimality and coordination in multiagent systems[C]. In *Proceedings of the Sixteenth international Joint Conference on Artificial intelligence* (July 31 - August 06, 1999). T. Dean, (Ed.) Morgan Kaufmann Publishers, San Francisco, CA, pp. 478–485.
- [97] D. S. Bernstein, S. Zilberstein, and N. Immerman. (2000). The complexity of decentralized control of markov decision processes[C]. In *Proceedings of the 16th Conference on Uncertainty in Artificial intelligence* (June 30 - July 03, 2000). C. Boutilier and M. Goldszmidt, Eds. Morgan Kaufmann Publishers, San Francisco, CA, pp. 32–37.
- [98] R. Becker, S. Zilberstein, and V. Lesser. (2004). Decentralized Markov decision processes with event-driven interactions[C]. In *Proceedings of the Third international Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1* (New York, New York, July 19 - 23, 2004). International Conference on Autonomous Agents. IEEE Computer Society, Washington, DC, 302–309.
- [99] D. Pynadath and M. Tambe. (2002). The communicative multiagent team decision problem: analyzing teamwork theories and models[J]. *Journal of Artificial Intelligence Research*, Vol.16, pp. 389–4232.
- [100] P. Paruchuri, M. Tambe, F. Ordonez, S. and Kraus. (2004). Towards a formalization of teamwork with resource constraints[C]. In *Proceedings of the Third international Joint Conference on Autonomous Agents and Multiagent Systems - Volume 2* (New York, New York, July 19 - 23, 2004). International Conference on Autonomous Agents. IEEE Computer Society, Washington, DC, pp. 596–603.
- [101] P. Scerri, L. Johnson, D. V. Pynadath, P. Rosenbloom, N. Schurr, M. Si, and M. Tambe. (2003). Getting robots, agents and people to cooperate: an initial report[C/OL]. American Association Artificial Intelligence (AAAI) Spring Symposium on Human Interaction with Autonomous Systems in Complex Environments, 2003.
<http://www.cs.cmu.edu/~pscerri/papers/RAP-SS.pdf>
- [102] A. Omicini, A. Ricci, M. Viroli, C. Castelfranchi, and L. Tummolini. (2004). Coordination artifacts: environment-based coordination for intelligent agents[C]. In *Proceedings of the Third international Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1* (New York, New York, July 19 - 23, 2004). International Conference on Autonomous Agents. IEEE Computer Society, Washington, DC, 286–293.
- [103] A. Garland, and R. Alterman. (2004). Autonomous agents that learn to better coordinate[J]. *Autonomous Agents and Multi-Agent Systems*, Vol. 8(3) (May. 2004), 267–301.

- [104] M. S. Fox. (1979). Organization structuring: Designing large complex software[R]. Technical Report. CMU-CS-79-155, Computer Science, Carnegie-Mellon University, December 1979.
- [105] M. S. Fox. (1981). An organizational view of distributed systems[J]. In IEEE Transactions on systems, Man, and Cybernetics, 11(1): 70–80, January 1981.
- [106] W. Jiao, J. Debenham, B. Henderson-Sellers. (2005). Organizational models and interaction patterns for use in the analysis and design of multi-agent systems[J]. Web Intelligence and Agent Systems. Vol.3, No.2, 2005, pp. 67–83, IOS Press.
- [107] S. Abdallah, N. Darwish, O. Hegazy. (2002). Monitoring and synchronization for teamwork in GPGP[C]. In *Proceedings of the 2002 ACM symposium on Applied computing*(Madrid, Spain, March 11 - 14, 2002). SAC '02. ACM Press, New York, NY, 288–293.
- [108] W. Fan, X. Zuo. (2004). Instance of abstract performance of multi-agent organizational coordination[J]. Journal of Civil Aviation University of China. June, 2004, 22(3). pp.60–64. (樊玮. 左晓英. 一个多Agent组织协作的抽象推演实例[J]. 中国民航学院学报. 2004. 22(3): 60–64.)
- [109] W. Fan, H. Chi, and L. Ji. (2005). Multi-agent cooperation based on organizational structure[J]. Chinese Computer Applications. Vol. 25(5), pp. 1045–1048. (樊玮, 池宏, 计雷. 基于组织结构的多主体协作[J]. 计算机应用. 25(5): 1045–1048.)
- [110] B. Horling. (2006). Quantitative organizational modeling and design for multi-agent systems[D]. PhD disseration, University of Massachusetts at Amherst, February 2006.
- [111] B. Horling, and V. Lesser. (2004). A survey of multi-agent organizational paradigms[J]. The Knowledge Engineering Review, Vol. 19(4) (Dec. 2004), pp. 281–316.
- [112] M. Sims, C. Goldman, and V. Lesser. (2003). Self-organization through bottom-up coalition formation[C]. In Proceedings of Second International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2003), pages 867–874, Melbourne, AUS, July 2003. ACM Press.
- [113] K. Decker. (1996). TAEMS: a framework for environment centered analysis and design of coordination mechanisms[M]. In Foundations of Distributed Artificial Intelligence, Chapter 16, pages 429–448. G. O'Hare and N. Jennings (eds.), Wiley Inter-Science, January 1996.
- [114] K. Fischer. (1999). Agent-based design of holonic manufacturing systems[J]. Journal of Robotics and Autonomous Systems, 27(1-2): 3–13, 1999.

-
- [115] X. Zhang and D. Norrie. (1999). Holonic control at the production and controller levels[C]. In Valckenaers, P., Van Brussel, H. (Eds), Proceedings of the 2nd International Workshop on Intelligent Manufacturing Systems (IMS 99), pages 215–224, 1999.
- [116] T. Sandholm and V. Lesser. (1997). coalitions among computationally bounded agents[J]. Artificial Intelligence, Special Issue on Economic Principles of Multi-Agent Systems, 94(1):99–137, January 1997.
- [117] A. Chavez and P. Maes. (1996). Kasbah: an agent marketplace for buying and selling goods[C]. In First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'96), pages 75–90, London, UK, 1996. Practical Application Company.
- [118] M. Wellman. (2004). Online marketplaces[M]. In M. P. Singh (ed.), Practical Handbook of Internet Computing. Chapman Hall & CRC Press, Baton Rouge, 2004.
- [119] C. Brooks, E. Durfee, and A. Armstrong. (2000). An introduction to congregating in multiagent systems[C]. In Proceedings of the Fourth International Conference on Multiagent Systems, pp. 79–86, 2000.
- [120] C. Brooks and E. Durfee. (2003). Congregation formation in multiagent systems. Journal of Autonomous Agents and Multiagent Systems, 7(1-2):145–170, 2003.
- [121] Y. Shoham and M. Tennenholtz. (1995). On social laws for artificial agent societies: off-line design[J]. Artificial Intelligence, 73(1-2): 231–252, 1995.
- [122] M. Colombetti, N. Fornara, and M. Verdicchio. (2004). A social approach to communication in multiagent systems. In J. A. Leite, A. Omicini, L. Sterling, and P. Torroni, editors, Declarative Agent Languages and Technologies, volume 2990 of Lecture Notes in Artificial Intelligence, pages 191–220. Springer-Verlag, May 2004.
- [123] G. Wiederhold, P. Wegner, and S. Cefi. (1992). Toward megaprogramming[J]. Communications of the ACM, 33(11): 89–99, 1992.
- [124] K. Sycara, K. Decker, and M. Williamson. (1997). Middle-agents for the Internet[C]. In Proceedings of the 15th International Joint Conference on Artificial Intelligence, pages 578–583, January 1997.
- [125] G. Beavers and H. Hexmoor. (2001). Teams of agents[C]. In Proceedings of the IEEE Systems, Man, and Cybernetics Conference, pages 574–582, 2001.
- [126] N. R. Jennings. (1995). Controlling cooperative problem solving in industrial multi-agent systems using joint intentions[J]. Artificial Intelligence. 75(2) (Jun. 1995), 195–240.

- [127] R. Milner. (1991). The polyadic Pi-calculus: a tutorial[R]. Technical Report ECS-LFCS-91-180, Computer Science Department, University of Edinburgh, UK, October 1991. (See also: F. L. Bauer, W. Brauer, and H. Schwichtenberg, (eds.) Logic and Algebra of Specification, pages 203–246. Springer-Verlag, 1993.)
- [128] R. Milner, J. Parrow, and D. Walker. (1989). A calculus of mobile processes: part I[R]. Technical Report ECS-LFCS-89-85. Laboratory for Foundations of Computer Sciences, Department of Computer Science, University of Edinburgh, June 1989. (See also: Information and Computation, Vol. 100(1) pp. 1–40. September, 1992.)
- [129] R. Milner, J. Parrow, and D. Walker. (1989). A calculus of mobile processes: part II[R]. Technical Report ECS-LFCS-89-86. Laboratory for Foundations of Computer Sciences, Computer Science Department, University of Edinburgh, June 1989. (See also: Information and Computation, Vol. 100(1) pp. 41–77. September, 1992.)
- [130] T. Rorie. (1998). Formal modeling of multi-agent systems using the π -calculus[D]. Master thesis, Department of Computer Science, North Carolina A&T State University, Greensboro, NC.
- [131] W. Jiao, and Z. Shi. (1999). Formalizing agent’s attitudes with the polyadic π -calculus[C]. In *Proceedings of the 4th Workshop on Practical Reasoning and Rationality*, Stockholm, Sweden, 31st, July 1999. pp. 21–27.
- [132] W. Jiao, and Z. Shi. (2000). Modeling Dynamic Architectures for Multi-Agent Systems[J]. Chinese journal of computers, (焦文品, 史忠植. 构造MAS的动态体系结构的模型[J]. 计算机学报. 2000, 23(7): 732–737.)
- [133] B. Yin, Z. He, G. Xu, F. Tan, *et al.* (2004). Discrete mathematics[M]. 2nd Edition. Beijing, PR China: Higher Education Press, 2004. Chapter 19. (尹宝林, 何自强, 许光汉, 檀凤琴, 等. 离散数学[M]. 第二版. 北京: 高等教育出版社, 2004. 第19章.)
- [134] E. G. Coffman Jr., M. R. Garey, D. S. Johnson. (1978). An Application of Bin-Packing to Multiprocessor Scheduling[J]. SIAM Journal on Computing, Vol. 7, No. 1, February 1978. pp. 1–17.
- [135] A. D. Mali, (2005) On quantified weighted MAX-SAT[J]. Decision Support Systems 40(2) (Aug. 2005), pp. 257–268.
- [136] M. E. Aydin, E. Öztemel. (2000). Dynamic job-shop scheduling using reinforcement learning agents[J]. Robotics and Autonomous Systems, 33(3), pp. 169–178. Chap. 2.

-
- [137] M. P. Wellman, (1993). A market-oriented programming environment and its application to distributed multicommodity flow problems[J]. *Journal of Artificial Intelligence Research*. Vol. 1, pp. 1–23.
 - [138] G. İnalhan, D. M. Stipanović, and C. J. Tomlin. (2002). Decentralized optimization with application to multiple aircraft coordination[C]. In *Proc. IEEE Int. Conf. on Decision and Control*, Las Vegas, Nevada, 2002.
 - [139] W. Fan, F. Xue. (2006). Optimize cooperative agents with organization in distributed scheduling system[C]. 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* Vol.4114, pp. 502–509.
 - [140] S. J. Russell, P. Norvig. (1995). *Artificial intelligence: a modern approach*[M]. Prentice-Hall, Inc. Chap. 2. pp. 45–49.
 - [141] F. Bellifemine, A. Poggi, G. Rimassa. (2001). JADE: a FIPA2000 compliant agent development environment[C]. In *Proceedings of the Fifth international Conference on Autonomous Agents* (AGENT01), 216–217, Montreal, Canada. ACM Press, New York, NY.
 - [142] M. Dorigo, V. Maniezzo, and A. Colorni. (1991). Positive feedback as a search strategy[R]. Technical Report 91–016, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1991.
 - [143] E. Bonabeau, M. Dorigo, and G. Theraulaz. (2000). Inspiration for optimization from social insect behavior[J], *Nature*, Vol. 406 No 6, pp.39–42.
 - [144] M. Dorigo and T. Stützle. (2004). *Ant Colony Optimization*[M]. Bradford Books (MIT Press).
 - [145] T. Stützle, and H. Hoos. (1997). The $\mathcal{MAX-MIN}$ ant system and local search for the traveling salesman problem[C]. In *Proceedings of the Fourth International Conference on Evolutionary Computation (ICEC'97)*, pp. 308–313. IEEE Press.
 - [146] T. Stützle, and H. Hoos. (1997). Improvements on the ant system: Introducing $\mathcal{MAX-MIN}$ ant system[C]. In *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms*, pages 245–249. Springer Verlag, Wien, 1997.
 - [147] W. Fan, G. C. Zhang, and F. Xue. (2006). Design and implementation of airline ground services mas development platform[C]. In *First Conference on Multi-agent Theory and Application*, Yantai, China, 2006. C. Y. Shi, Z. Z. Shi, *et al* (Eds.): *Journal of Computer Research and Development* Vol 43(suppl.I), pp 414–419 (樊玮, 张广才, 薛帆. 飞机地面作业调度MAS开发平台的设计与实现[C]. 第一届Agent理论与应用学术会议. 2006年8月, 山东烟台. 石纯一, 史忠植, 等编. *计算机研究与发展*. 42(suppl.I): 414–419.)

- [148] A. J. Davenport, and J. C. Beck. (2000). A survey of techniques for scheduling with uncertainty[Z/OL]. <http://www.eil.utoronto.ca/profiles/chris/chris.papers.html>.

Publications During M.Sc. Study

Aug 2006 W. Fan, and **F. Xue**. *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 Vol 4114, pp 502–509. (SCI(BEY13), Ei(064210172483))

Aug 2006 S. X. Zhu ,and **F. Xue**. *Reinforced Circle Architecture and Implementation in Information System of Civil Aviation Fleet of China*. Computer Engineering and Design. Vol 27(16) pp 3076–3077 (祝世兴, 薛帆. 基于环状加强架构的中国民航机队信息系统. 计算机工程与设计: 27(16): 3076–3077.(中文核心))

Aug 2006 W. Fan, G. C. Zhang, and **F. Xue**. *Design and Implementation of Airline Ground Services MAS Development Platform*. In *First Conference on Multi-agent Theory and Application*, Yantai, China, 2006. CY Shi, ZZ Shi, *et al* (Eds.):Journal of Computer Research and Development Vol 43(suppl.I), pp 414–419 (樊玮, 张广才, 薛帆. 飞机地面作业调度MAS开发平台的设计与实现. 第一届Agent理论与应用学术会议. 2006年8月, 山东烟台. 石纯一, 史忠植, 等编. 计算机研究与发展. 42(suppl.I): 414–419) (中文核心)

Oct 2004 **F. Xue**, Z. J. Gu, J. Wang, and J. Zhang. *A Search Engine Applying to Campus Network: CAUCIIC*. in *First Postgraduate Seminar, Civil Aviation University of China*, XH Xu (eds.):Journal of Civil Aviation University of China Vol 23(suppl.), pp 134–136 (薛帆, 顾兆军, 等. 面向校园网的搜索引擎CAUCIIC. 中国民航学院第一届研究生论坛. 中国民航学院学报. 23(suppl.): 134–136.(科技核心))

Under view **F. Xue**, and W. Fan. *DSAFO: A Multi-agent Algorithm for Airport Ground Service Scheduling*. submitted to Journal of Information and Computational System. (Ei)

Curriculum Vitæ

Vita

Fan Xue (薛帆) was born in Baishui County, Shaanxi on September 28, 1982, the first son of Xinjie Xue and Qiaoxia Jing. After completing his study at Xianlin Senior Middle School, Huaxian County, Shaanxi, in 2000, he entered Civil Aviation University of China in Tianjin. He received the degree of Bachelor of Engineering in Automation from College of Traffic Engineering, Civil Aviation University of China in July, 2004. In September, 2004, he entered College of Computer Science and Technology, Civil Aviation University of China for a Master's degree study. He now is a M.Sc. student, interested in Intelligent Information Process, Multi-agent Systems and Artificial Intelligence.

M.Sc. Courses

During his M.Sc. study, 17 theoretical courses and 4 practices have been finished, and 37 credits have been gained totally (33 credits are required for M.Sc. degree), including 21 compulsory credits, 12 optional credits and 4 practical credits. The average score of the theoretical courses is 86.0, and the average score of major courses is 90.0. Six more undergraduate major courses in computer science are also studied as a complement.

Selected Project and Research Experiences

2005–2006 Algorithm developer of Airport Ground Service Scheduling Optimization Based on Multi-agent Coordination and Data Fusion (National)

collaborated with Air China and Chinese Academy of Sciences, granted by NSFC (No.60472123).

2004–2005 Developer of Xiamen Airline Revenue Management System 2nd Edition

collaborated with Xiamen Airline

2003–2006 Developer of Information System of Civil Aviation Fleet of China

collaborated with General Administration of Civil Aviation of China