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：_____

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；并同时将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 $\mathcal{MM}$AS 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 $\mathcal{MM}$AS 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

目 录

表 格

插 图

专业缩写索引

缩写	全称	
A-SMGCS	Advanced-Surface Movement Guidance and Control Systems	9
ACL	Agent Communication Language	14, 53, 62
ACO	Ant Colony Optimization	10
ACO	Ant Colony	16, 70
ADOPT	Asynchronous Distributed OPTimization	12
AGSS	Airport Ground Service Scheduling	1
AI	Artificial Intelligence	6
ANN	artificial neural networks	10
AODB	Airport Operation Data Base	33
ARCHON	ARchitecture for Cooperative Heterogeneous ONline system	15
ARM	Airline Resource Management system	8, 15
AS	Ant System	71
BCIA	Beijing Capital International Airport	9
BDI	Belief-Desire-Intention	16
BOID	Beliefs-Obligations-Intentions-Desires	16
BT	Baggage Tractor	1, 50
CMDP	single-agent Constrained MDP	17
CNET	Contract NET	14
COM-MTDP	COMmunicative Multiagent Team Decision Problem	17
Coo-BDI	Cooperative BDI	16
CSP	Constraint Satisfaction Problem	10

缩写	全称	
DAI	Distributed AI	10, 13
DEC-MDP	DECentralized MDP	17
DEC-POMDP	DECentralized POMDP	17
Dis-CSP	Distributed Constraint Satisfaction Problem	11, 28
Dis-DSS	Distributed Dynamic Scheduling System	8, 15
Dis-HTN	Distributed HTN	15
DJSSP	Dynamic Job-Shop Scheduling Problem	29
dMARS	distributed Multi-Agent Reasoning System	16
DSAFO	Dynamic Scheduling Agents with Federation Organization	36, 57, 73
DSIPE	Distributed System for Interactive Planning and Execution	15
EDD	Earliest Due Date	9, 38, 73
EMTDP	Extended Multiagent Team Decision Problem	17
ERT	Earliest Ready Time	9, 73
FA/C	Functionally Accurate, Cooperative system	15
FCFS	First-Come-First-Serve	9
FIDS	Flight Information Display System	33
FIFO	First-In-First-Out	9
FIPA	Foundation for Intelligent Physical Agents	15, 53
FOC	Flight Operations Control system	33
FSTS	Fuzzy Subjective Task Structure	16
GA	Genetic Algorithm	10
GBB	Generic Blackboard	8
GPGP	Generalized Partial Global Planning	15
HPS	Hospital Patient Scheduling	16
HTN	Hierarchical Task Network	15
JADE	Java Agent DEvelopment framework	53, 57, 62

缩写	全称	
JSSP	Job-Shop Scheduling Problem	9, 29, 72
KQML	Knowledge Query Manipulation Language	14
LB	Load Baggage	1, 57
LC	Load Cargo and mail	1, 57
LGPL	Lesser General Public License Version	53
MAS	Multi-Agent System	10, 12
MBO	Model-Based Optimization	9
MDP	Markov Decision Process	16
$\mathcal{MM}$**AS**	$\mathcal{MAX}$-$\mathcal{MIN}$ Ant System	72
MMDP	Multiagent MDP	17
MSF	minimum slack	9
NEXP-complete	Non-EXPonential complete	17
$\mathcal{NP}$**-complete**	Non-Polynomial complete	9, 26
PGP	Partial Global Planning	15
POMDP	Partially Observable Markov Decision Process	12, 17
poset	partial ordered set	24
PSO	Particle Swarm Optimization	10, 70
PSPACE-complete	Polynomial SPACE complete	29
QBF	Quantified Boolean Formula	29
RCS	Rolegraph Coordination Strategy	16
SA	simulated annealing	10
SIPE	System for Interactive Planning and Execution	15
SPT	shortest processing time	9
TAAM	Total Airport & Airspace Modeller system	8

缩写	全称	
TÆMS	a framework for Task Analysis, Environment Modeling, and Simulation	15
TM	Turing Machine	16, 29
TSP	Travel Salesman Problem	71
UB	Unload Baggage	1, 57
UC	Unload Cargo and mail	1, 50, 57

第一章 绪 论

在各种工业环境中，有许许多多调度和规划问题都是非常具有挑战性的。因为从计算复杂性的角度来看，它们是$\mathcal{NP}$-困难或者更困难的问题[1, 2, 3]。要在这些问题的通用形式下用传统优化算法来解决这些问题是极其困难的。本文所关注的飞机地面作业调度（AGSS，Airport Ground Service Scheduling）问题就是这样的一个困难的调度问题。

1.1 什么是飞机地面作业调度？

飞机地面作业调度，简单地说，就是为“飞机地面作业”而进行的一系列计划调动活动。

在本文中，飞机地面作业是从航班落地到起飞之间的一系列航班服务操作，包括行李装卸、供餐、补充燃油、清洁等。并且在本文中，我们将这些服务操作分别称为“子作业”，在不引起误解的情况的也称“作业”。值得一提的是 H. Hartmann (2001) 将该定义分为三部分：地面交通服务（地面引导服务等）、机场地面服务（飞机牵引、除冰、补充燃油等）和航空公司地面服务（地面装卸、供餐、加清水、除污水等）[4]。他的划分主要是表现了个体的企业的视角，相对地，本文的定义主要是体现系统协作的角度。

从航班的落地到起飞，其中存在许多飞机服务作业需要完成。这些服务作业之间还存在根据不同的航班和环境而变化的工作流顺序[5]。从一个航班到另外一个航班，众多服务作业和它们的顺序都可能发生变化。例如，图 1–1 、图 1–2 和图 1–3 分别展示了中转航班、离港航班和进港航班的典型服务工作流。在图 1–1 中，用浅红色标记的服务作业为经常影响航班正点起飞的关键作业，而在这三幅图中用虚线框表示的服务作业为可选作业，是否进行这些服务取决于飞机型号、是否国际航线、环境以及现场的飞机机务维修需要等等。

进行图 1–1 、图 1–2 和图 1–3 中的每一种作业时，都需要特定种类的材料、产品、人力和（或）工程师、航空地面支持设备以及交通设备等。在本文中，我们将这些被需求物称为“资源”。飞机地面服务作业中的典型资源需求依作业种类不同而不同，详细情况见表 1–1 所示。

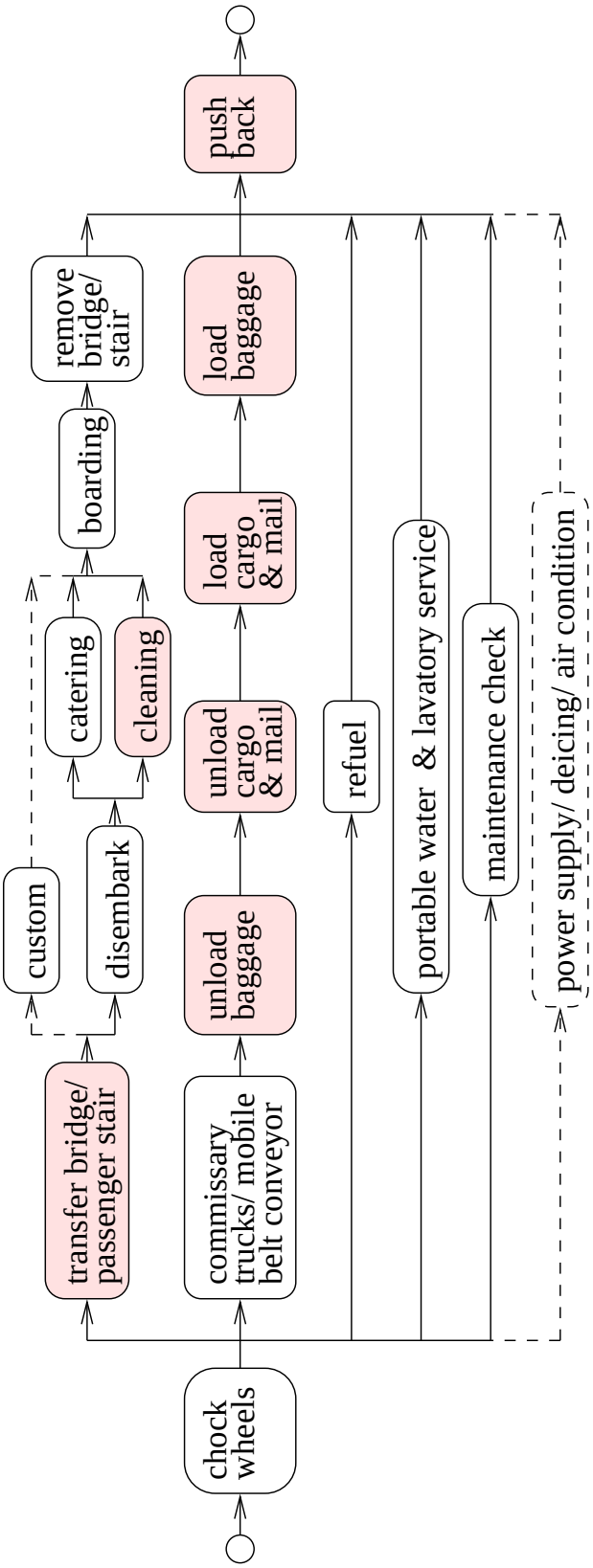


图 1–1: 典型中转航班的服务作业

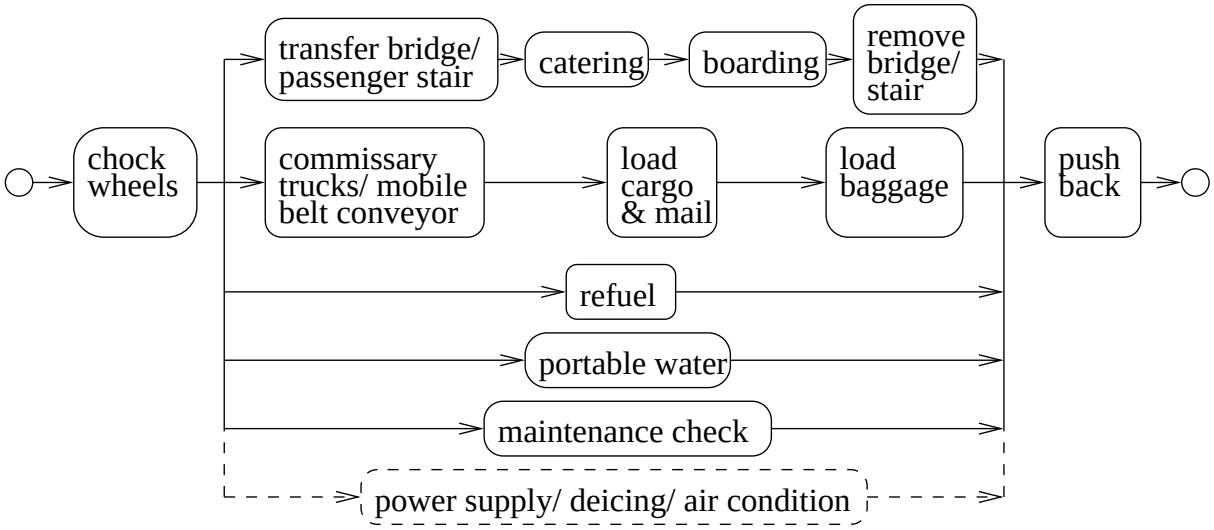


图 1–2: 典型离港航班的服务作业

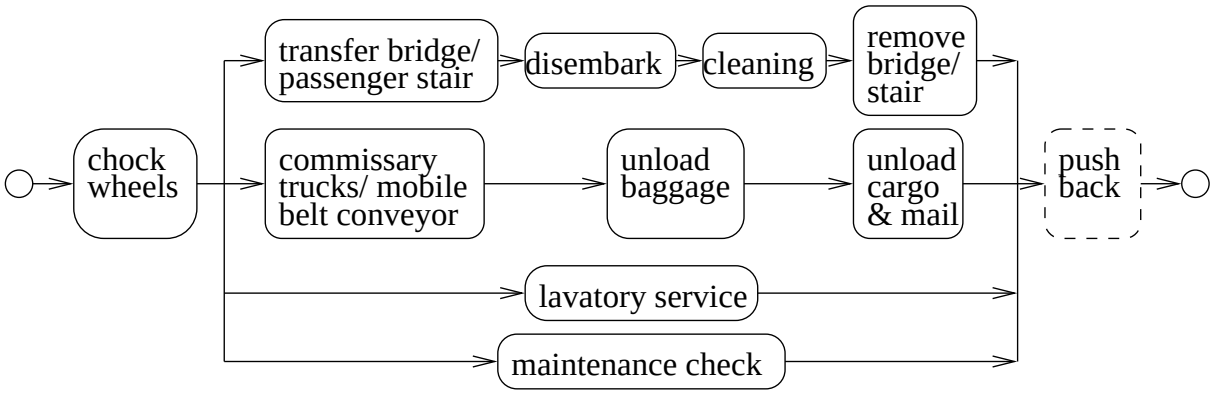


图 1–3: 典型进港航班的服务作业

表 1–1: 飞机地面服务作业中的典型资源需求

作业类型	资源需求
放置轮挡	轮挡
装卸准备	平台车 / 移动传送带
卸行李（UB, unload baggage）	行李牵引车（BT, baggage tractor）+ 小车板
卸货邮（UC, unload cargo and mail）	行李牵引车（BT）+ 小车板
装货邮（LC, load cargo and mail）	行李牵引车（BT）+ 小车板
装行李（LB, load baggage）	行李牵引车（BT）+ 小车板
旅客下机准备	廊桥 / 客梯车
清洁机舱	清洁工(+运载小巴)
供餐	餐车
补充燃料	加油车 / 油罐车
补充清水	清水车
处理污水	污水卡车 / 污水小车
处理垃圾	垃圾车
机务检查	机务工程师（+检察平台）
推出	拖车（+拖把）
电源服务（可选）	地面电源车
气源服务（可选）	空调车
除冰（可选）	除冰车

飞机地面作业调度（AGSS）问题是迅速高效地将飞机地面作业进行规划和安排的过程[5]。P. Baptiste, C. Le Pape 和 W. Nuijten (1995) 说：“调度就是及时地将活动分配到资源上的过程”[6]。依照这个定义，飞机地面作业调度问题就是将所有受到约束的作业（飞机服务操作）分配到各种各样的动态资源（例如工程师、航空地面支持设备、原料、交通设备等）上，来及时地满足航班降落和起飞的时间需求。

文章第 3 章对该问题进行了详细的形式化定义，同时也介绍了该问题的理论和实践中的特性。

1.2 研究意义

在经济意义上，飞机地面作业调度问题是对于航空工业非常重要的，但是目前国内在该问题上尚存在一些管理上的问题。

1.2.1 经济上的重要性

首先，飞机地面作业调度问题对于航空公司和机场的经济和收益十分重要。在 2006 年第一季度，国内平均航班正常率为 81.07%，最低的三个航空公司深圳航空、厦门航空和上海航空的航班正常率分别只有 75.86%，78.65% 和 75.87% [7]。大量的航班延误会给航空公司和机场带来巨大的经济损失。根据 L. Shi (2005) 的统计，非正常航班给航空公司带来的损失大约占航空公司总运营成本的 2–3%，这对于国内运营成本在上百亿人民币的大型航空公司来说就是上亿元的损失[8]。在各种导致航班延误的因素中，飞机地面作业调度是其中最重要的一个因素之一。

另一方面，飞机地面作业调度对航空公司和机场的服务质量也十分重要。由飞机地面作业调度所保障的准时的合适的航班起降时间，是航空业界专门人员和旅客评价航班服务质量的最要因素。通过 S.-H. Tsaur, T.-Y. Chang 和 C.-H. Yen (2002) 的调查，台湾航空界该因素所占的比重为所有因素的 12.8%[9]。

总之，飞机地面作业调度问题对航空工业是十分重要的。

1.2.2 存在的管理问题

但是目前在国内飞机地面作业调度中，仍然存在两方面的管理问题：

首先，航空公司和航空港在多年的实践中建立了各种各样的信息系统，这些信息系统为民航事业的科学管理和高速发展作出了重要的贡献，但是，这些信息系统由于建设初衷和建设时间的不同，缺乏统一全面的规划。对同一个信息目标，各信息系统之间存在着信息不完全、信息不一致、甚至信息相悖的问题。对飞机地面作业相关信息系统建设来说，还存在着各种信息目标的实时移动问题，即静态信息和动态信息的不一致情形。再加上部分信息系统基础建设的不足，使得目前飞机地面作业系统的信息不通畅，各种资源利用的适时情况不明，缺少对信息目标的全面、准确把握，缺少对异常事件的及时应变能力。如去年的一场台风来临，由于澳门机场具有良好的信息系统基础建设，能很快运算出飞机不必停飞，而深圳机场由于缺乏统一的信息基础建设，无法在有限的时间内得出数据，因此只能停飞，损失将以亿计，这无疑为深圳航空造成不小的损失[10]。

同时，目前的实际运行机制只有单资源（如：不同车辆由不同的调度指挥）调度功能，没有综合调度功能，且这些调度大都以人工经验为主，使得各种资源无法得到充分利用。并且，飞机地面作业是由若干种资源共同完成，在执行过程中，由于作业空间有限，存在着相互协调的问题，但还没有一套切实可行的科学方法做为合理进行飞机地面作业安排的决策依据。

1.2.3 高效飞机地面作业调度的优点

文本所期望的高效的飞机地面作业调度，将可能大幅提高航空工业的运营利润，这是因为：

1. 高效的飞机地面作业调度方案会导致高效的飞机服务时间序列规化，从而最小化航班中转时间；
2. 高效的飞机地面作业调度能够辨别优先作业需求并进行合理的服务，从而最小化人力、资源和设备成本；
3. 高效的飞机地面作业调度也能够提升航空公司和机场面对不确定情况下的规划调度能力，并能鉴别在带有延误的不确定航班时刻表下的因缺少可用资源而导致的潜在航班延误。

1.3 论文主要工作

本文的主要工作是：

1. 从约束满足的视角对飞机地面作业调度问题进行了形式化定义，揭示了飞机地面作业调度的一些特性；
2. 为飞机地面作业调度问题建立了一个动态分布式调度模型和一个动态分布式调度环境 run-and-schedule；
3. 开发了一个多 Agent 算法 DSAFO，从理论上和实践上来满足和优化飞机地面作业调度问题；
4. 详细研究了 DSAFO 算法中的参数选取，并给出了应用 DSAFO 算法到实际飞机地面作业调度问题中的合适的参数选择的建议；
5. 指出了 DSAFO 算法的本质是基于传统人工智能（AI） 的分治（Divide-and-Conquer）概念的多 Agent 协作优化算法，并且指出了 DSAFO 算法的优化本质来源于组织协作和动态消息传输；
6. 为了进行算法对比，为飞机地面作业调度问题设计了一个 $\mathcal{MM}$AS（$\mathcal{MAX}-\mathcal{MIN}$ Ant System，最大－最小蚂蚁系统，见第 7.1.1 小节）。

1.4 论文组织

本文的剩余部分被分别组织为七个章节。

第 2 章给出了过去的飞机地面作业调度问题的研究概况，并回顾了该问题相关的两个研究领域的研究进展和现状：Job-Shop 调度问题（JSSP，Job-Shop Scheduling Problem，见第 2.2 节）和分布式约束满足问题（Dis-CSP，Distributed Constraint Satisfaction Problem，见第 2.3 节）。协作多Agent系统（Coordinative multi-agent system）和它的组织结构模型总结在第 2.4 节。另外，多价 π-演算（polyadic π-calculus）也在第 2.5 节有所介绍。

第 3 章在进行了为计算可行性而作的一些假设后，讨论了飞机地面作业调度问题的形式化。同时为了更好地将形式化分析应用到实际问题中，还介绍了该问题的实际调度特性。

第 4 章在第 4.1 节说明了将飞机地面作业调度问题进行分布式建模的优点和必要性，在第 4.2 节介绍了一个分布式调度模型，在最后一节为飞机地面作业调度问题给出了一个动态调度环境 run-and-scheduling。

第 5 章详细描述了为优化飞机地面作业调度问题而开发的多 Agent 优化算法 DSAFO。DSAFO 算法的本质和策略介绍在第 5.2 节；Agent 角色和对应的形式化分别介绍在第 5.3 节和第 5.4 节；第 5.5 节分析了 DSAFO 的时间复杂度；最后第 5.6 节将该多 Agent 优化算法在 JADE 和 FIPA ACL 平台下进行了实现。

第 6 章介绍了采用实际数据的飞机地面作业调度性能优化实验。第 6.2 节演示了 DSAFO 的优化结果和在不同参数影响下的优化结果的变动；第 6.3 节给出了 DSAFO 的实验结果小结和算法特性。

算法对比位于第 7 章。将 DSAFO 同另外三个算法进行了对比：$\mathcal{MM}$AS，EDD* 和 ERT*。为飞机地面作业调度问题而开发的 $\mathcal{MM}$AS 算法在第 7.1.1 小节有详细的介绍。

最后，第 8 章归纳了一个 DSAFO 的研究总结，并且也给出了飞机地面作业调度的可望的未来研究方向。

第二章　研究现状和相关工作

在飞机地面作业调度领域，已经有了一些理论研究和工业应用；在飞机地面作业调度的相关的 JSSP 和 Dis-CSP 领域，则已经有了大量从数学规划到人工智能（AI）的各种方法；作为本文所提出的算法的重要基础，多 Agent 协作和组织，以及多价 π-演算，在最近的一段时间内也有长足的发展。

2.1　飞机地面作业调度研究现状

在“航空公司资源管理系统”（Airline Resource Management, ARM）中，D. E. Neiman, D. W. Hildum, V. R. Lesser 等人为飞机地面作业调度开发了一个基于多 Agent的 Dis-DSS 系统（Distributed Dynamic Scheduling System，分布式动态调度系统）[11, 12]。该系统是从较早的 DSS（Distributed Scheduling System，分布式调度系统）演化而来，并引用了一套基础黑板（Generic Blackboard, GBB）机制。Dis-DSS 还具有资源纹理测量、最紧约束优先（most-tightly-constrained-first）、协商和同步计划修正机制。D. E. Neiman 和 V. R. Lesser 后来使用了一种协作修正机制来处理约束不可满足的情况[13]。并且 M. Chia, D. E. Neiman, V. R. Lesser 使用了两种协作行为（poaching）和（distraction）来强化 Dis-DSS[14]。

A. Cheung, W. H. Ip, D. Liu and C. L. Lai 则研究了一个使用遗传算法（Genetic Algorithm, GA）来解决飞机地面作业调度的方法[15]。他们改进的而且基于操作的染色体表达方法来代表不同的车辆，从而可以得出合适的服务调度。

在工业领域，一个波音的子公司 Preston[1] 为机场地面资源管理研制了一套 TAAM 系统（Total Airport & Airspace Modeller），其中包含廊桥和行李传送带的安排功能。德国汉莎（Lufthansa）研发了 PERSEUS 项目[2]，其中包括与汉莎旅客服务相关的员工的排班、排班调整和实时的任务分配。ASCENT 公司[3]研发了 ARIS/AR 航路规划系统来调度行李装卸、清洁工作业、航空食品作业、机坪工人和其他地面设备等。中国民航总局

[1] 见 http://www.preston.net
[2] 见 http://www.groundstar.de
[3] 见 http://www.ascent.com

第二研究所[4]生产了一个名为“指挥调度系统”的管理信息系统，作为“机场综合信息管理系统”的子系统来分配地面资源和收集作业结果。该“指挥调度系统”已经成功地应用在深圳、昆明、成都、杭州、郑州、西安、厦门、天津等许多国内机场。

另外，Northrop Grumman 公司[5]的 Park Air Systems[6] 及其子系统 A-SMGCS（Advanced-Surface Movement Guidance and Control Systems）在世界范围内广泛地被用来控制机场交通。另外北京航空航天大学也曾为首都机场（Beijing Capital International Airport, BCIA）开发基于 GIS（Geographical Information System，地理信息系统）的机场地面交通辅助工具[16]。

2.2 Job-Shop 调度问题

定义 1 (JSSP) A. S. Manne 于 1960 年首先给出了 Job-Shop 调度问题（Job-Shop Scheduling Problem, JSSP）[17]:

> ……该排序问题涉及到 n 个任务的性能— —定义任务为需要一个单独的机器连续工作整数单位时间。一个最终生产，在一般意义上，会需要若干几个前序任务的完成。该调度问题包含若干个独立的时间片工作，而且需要让它们满足:
>
> - 前后顺序要求，
> - 设备占用问题。
>
> 整数未知变量 x_j 用来表示任务 j 的开始时间（$x = 0, 1, \ldots,$ T）。总共的排序会被准备来最小化“总耗时”（make-span）。

另外，A. S. Jain 和 S. Meeran 也给出了一个约束满足角度的定义[18]。

有很多方法来处理 JSSP 问题。一类方法是数学策略，有时候也被叫作精确算法，这其中包括动态规划（dynamic programming）、分解策略（decomposition strategies）、枚举技术（enumerative techniques）和基于模型的优化（Model-Based Optimization, MBO）等[19]。不过，因为 JSSP 是众所周知的 $\mathcal{NP}$-完全问题[1]，所以应用数学策略是很有局限性的。

另一种方法,相对应的，就是近似算法，其中包括分配规则（dispatching rules，也称为序列规则 sequencing rule 或调度规则 scheduling rule）、启发式[7]（heuristics）和

[4]见 http://www.caacsri.com

[5]见 http://www.northropgrumman.com

[6]见 http://www.parkairsystems.com

[7]应该注意到分配规则一般情况下是启发式；但启发式的内涵比分配规则要大一些。

Lagrangian 松弛[8]（Lagrangian relaxation）。启发式可以基于处理时间，比如最短处理时间（shortest processing time, SPT）、最早完工时间（earliest due date, EDD））、松弛时间（比如最小松弛优先（minimum slack first, MSF））、最早就绪时间（earliest ready time, ERT，也被称为先进先出 first-in-first-out, FIFO 或者先来先服务 first-come-first-serve, FCFS）以及这些方法的各种组合[20]。这些方法都相对简单而且复杂度都不高，不过他们是近似算法俄且一般来说不能得出 JSSP 问题的最优解。

另一类方法是 AI 方法，通常也叫做智能算法（intelligent algorithm）或者智能优化算法（intelligent optimization algorithm）。从上世纪80年代开始，一系列新技术被应用到了 JSSP 问题中，比如基于专家 / 知识的系统[2, 21]（expert/knowledge-based systems）、分布式人工智能（Distributed AI, DAI, 比如多 Agent 系统[22, 23, 24, 25]）（MAS））人工神经网络[26]（artificial neural networks, ANN）、禁忌搜索[27, 28, 29]（tabu search）、模拟退火[30, 31]（simulated annealing, SA）、遗传算法[32, 33]（genetic algorithms, GA）、蚁群优化[34]（Ant Colony Optimization, ACO）、粒子群优化[35]（Particle Swarm Optimization, PSO）等。这些方法都有自己独特的寻找近优解的策略。

模糊逻辑（Fuzzy logic）也许可以作为最后一种方法，而且不是一个独立的方法。因为模糊集（fuzzy set）理论仅仅曾经被用来开发混合的调度方法[36, 37]。

2.3 分布式约束满足问题

定义 2 (CSP) 形式化地说，一个约束满足问题（Constraint Satisfaction Problem, CSP）包含 n 个变量 $x_1, x_2, \ldots, x_n$，这些变量的值分别来自于有限的、离散的值域 $D_1, D_2, \ldots, D_n$，并且在它们的取值上有一些约束组成的集合。一般来说，一个约束就是一个谓词。也就是说，约束 $p_k(x_{k_1}, \ldots, x_{k_j})$ 是一个定义在 Cartesian 积 $D_{k_1} \times \ldots \times D_{k_j}$ 上的一个谓词。这个谓词是真当且仅当所包含的所有变量都满足这个约束。因此，解决一个约束满足问题等效于找到这样一个赋值，该赋值使得对所有的变量的取值对所有的约束都满足。[38]

解决约束满足问题的算法可以被分为两类：即，搜索算法（search algorithms）和相容性算法（Consistency algorithms）。解决该问题的搜索算法可以进一步被分为两类：即，回溯算法（Backtracking algorithms）和迭代改良算法（Iterative improvement algorithms）。

回溯算法 回溯算法是面向约束满足问题的最基础的、系统的搜索算法，例如最小冲突启发式（min-conflict heuristic）[39]。例如，弱承诺搜索算法（weak-commitment

[8]有时候 Lagrangian 松弛被分类为一种数学策略，不过终究它是一种近似算法。

search algorithm）[40]就是基于最小冲突回溯的；

迭代改良算法 在迭代改良算法中，同最小冲突启发式一样，所有变量都有试验性质的初始值。不过，并不构造一致的局部解。一个包含全部变量的有瑕疵的（flawed）的解，并使用爬山法（hill-climbing search）来不断修订所有变量；

相容性算法 相容性算法是一些预处理算法，来减少无意义的额外的回溯[41]。

定义 3 (Dis-CSP) 一个分布式约束满足问题（Distributed Constraint Satisfaction Problem, Dis-CSP）定义在一个多元组 $< \mathcal{A}, X, D, \mathcal{C} >$ 上，其中

$\mathcal{A} = \{\alpha_1, \ldots, \alpha_p\}$ - 一个含有 p 个 Agent 的集合;

$X = \{X_{\alpha_1}, \ldots, X_{\alpha_p}\}$ - 一个含有 p 各变量的集合,

对于每个 $\alpha \in \mathcal{A}$, $X_\alpha = \{x_\alpha^1, \ldots, x_\alpha^{q_\alpha}\}$ - 每个变量集合 α 包含 q_α 个变量;

$D = \{D_{\alpha_1}, \ldots, D_{\alpha_p}\}$ - 每个 Agent 的值域的集合;

对于每个 $\alpha \in \mathcal{A}$ and $\chi \in X_\alpha$, $\chi \in D_\alpha(\chi)$ - 每个变量 χ 的值域，并且改值能（且仅能）被 Agent α 所赋值;

$\mathcal{C} = \{c_1, \ldots, c_r\} = \mathcal{C}_{\alpha_1} \cup \ldots \cup \mathcal{C}_{\alpha_p}$ - 定义所有变量上的约束的集合,

对于每个 $i \in \{1, \ldots, r\}$,

$c_i : D_{\alpha_1}(x_{\alpha_1}^1) \times \ldots \times D_{\alpha_1}(x_{\alpha_1}^{q_{\alpha_1}}) \times \ldots \times D_{\alpha_p}(x_{\alpha_p}^{q_{\alpha_p}}) \mapsto \{\text{true}, \text{false}\}$

- 其中每个谓词可能和任何变量相关;

对于每个 Agent $\alpha \in \mathcal{A}$, $\mathcal{C}_\alpha = \{c_i | c_i \in \mathcal{C} \wedge \underset{x \in \mathcal{X}_\alpha}{\exists} c_i \text{ is relative to } x\}$,

- Agent α 的所有相关约束的集合，该集合必须被 Agent α 完全知道。

则，Dis-CSP 问题即找到这样一个赋值:

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

使得

$$\underset{1 \le i \le r}{\forall} c_i(a) \equiv \text{true}.$$

另外，M. Yokoo, E. H. Durfee, T. Ishida 和 K. Kuwabara 也给出了一个类似的形式化[42]。用来处理 Dis-CSP 问题的算法则可以列表如下：

异步回溯 异步回溯（asynchronous backtracking）算法是回溯算法的一个分布式的、异步的版本[42, 43]；

异步弱承诺搜索 M. Yokoo 提出的异步弱承诺搜索（asynchronous weak-commitment search）算法则是引入了最小冲突启发式来减少进行“坏”决策的风险[42, 44, 45]。另外，Agent 的顺序也是动态可变的，因此一个“坏”的决策可以通过详尽的搜索就能得到修正。该算法对于处理多 Agent 系统本身的冲突也十分有用；

分布式突围算法 在这个算法中，在各个邻居间互相传递两种消息“ok?”和“improve”来跳出局部极小；

分布式一致性算法 通过多 Agent 来实现 2-一致性是相对直接的，因为该算法可以通过迭代局部处理而完成[46]；

ADOPT P. J. Modi, W.-M. Shen, M. Tambe 和 M. Yokoo (2005) 开发了一个名为 ADOPT（Asynchronous Distributed OPTimization, ADOPT，异步分布式优化），来处理更一般的分布式约束满足优化问题[47]；

多 Agent POMDP 即，多 Agent 部分可见 Markov 决策过程（Partially Observable Markov Decision Processes, POMDP）[48]。

2.4 协作多 Agent 系统

2.4.1 多 Agent 系统

多 Agent 系统（Multi-agent system, MAS）是一个非常有前途的新型计算模式。一般地来说，一个多 Agent 系统是一些交互的并追求一些目标或任务集合的智能 Agent 所组成的系统。一个 Agent 是一个计算实体，比如一个软件程序或一个机器人，它可以被看作感知并作动于它的环境，并且它的行为的自治性（至少部分地）来源于它自己的经验[49]。M. Wooldridge 和 N. Jennings (1995) 给出了 Agent 的四个特性[50]：

- *自治性（autonomy）*：Agent 的行动和计算不需要人和其它设备的直接干预，而且具有一些种类的关于它们的行为和内部状态的控制；
- *社会性（social ability）*：Agent 和其它 Agent（有可能是人）通过一定种类的 Agent 通信语言（agent-communication language）进行交互；
- *反应性（reactivity）*：Agent 感知周围环境（这可能是物理世界、通过图形用户界面的用户、其它 Agent 的集合、国际互联网 Internet 或者是它们的组合），并对环境中所发生的改变进行立刻的响应；
- *预动性（pro-activeness）*：Agent 并不总是简单地对环境进行响应，它可以带有决心地执行目标驱动的行为。

同时，在不同的应用领域还需要 Agent 具有不同的特性[51]。在本文中，一般的 Agent 都被定义为具有实时性（real-time）、倾向性（intending）、持续性（continuous）和资源有限性（resource limited）的实体。

尽管多 Agent 系统已经被广泛接受，并且已经在很多领域实现了许多年，一些学者仍然对 Agent 和对象（Object）之间的区别争论不休。我们在这里采用 M. Wooldridge (2001) 的建议[51]，将本文所提出的算法从对象（尤其是主动对象 active Objects）中区别开来：

- Agent 比对象体现出了更强的自治性概念；
- Agent 的能力是柔性的（包括反应的、预动的和社会的）行为；
- 多 Agent 系统天然是多线程的。

2.4.2 协作多 Agent 系统

协作是管理不同动作（在多 Agent 系统内也就是 Agent）间交互和依赖关系的行为[52, 53]。一个协作模型可以被认为包含三个要素：协作主体（coordinables）、协作媒介（coordination media）和协作规则（coordination laws）. H. S. Nwana (1996) 指出[54]，具有协作的 Agent 系统的假设、基本原理和其目标就是 M. N. Huhns and M. P. Singh (1994) 所说的分布式人工智能（Distributed AI, DAI）[55]的详细说明。通过理解这些作者的观点，协作多 Agent 系统可以被叙述为：构造一个将独立构建的协作 Agent 进行相互接连的系统，使得整体的效果比任何一个成员 Agent 的能力都要高。形式化地来看，即

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

其中函数 V 表示‘价值增量’。它可以是任意涉及一些属性的定义，像速度、最坏情况下的性能、可靠度、适应性、精确程度等或它们的组合。协作是基于 Agent 和它们之间的关系的。这些关系有很多种，比如等效（equality）、启用（enablement）、禁止（inhibit）、干扰（interference）等等[56, 57]，如图 2–1 所示[58]。因此协作的不同类型的定义可以简要地如下给出：

- 协作（coordination）：协作是一些 Agent 在共同的环境中进行活动的系统的特性[52]；
- 合作（cooperation[9]）：合作是非对抗的（nonantagonistic）Agent 之间的协作[58]；
- 协商（negotiation）：协商是竞争的或者自私的 Agent 之间的协作[58]。

那么我们就可以总结出协作 Agent 的特性，即就是自治性（autonomy）、交互性（interaction）、任务性（task）和仁慈性（benevolence）：

[9]由于本文主要关注分布式动态调度，因此在不引起误解的情况下，在本文中我们将协作与合作不加区分地使用。

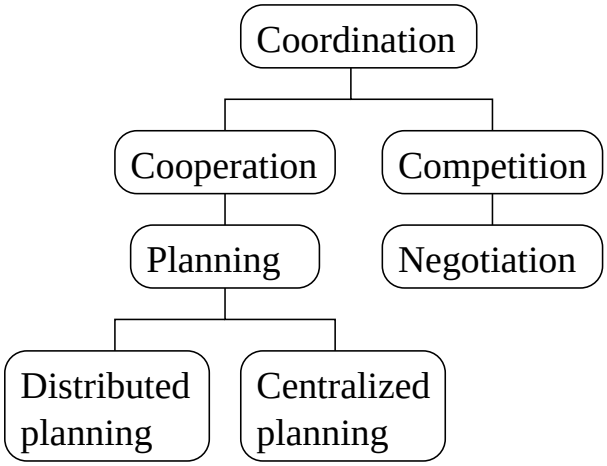


图 2–1: Agent 交互行为的分类

- *自治性*：Agent 的行动和计算不需要人和其它设备的直接干预，而且具有一些种类的关于它们的行为和内部状态的控制；
- *交互性*：Agents 可以同其它 Agent 交互；
- *任务性*：Agent 总是拥有一些任务作为自己的目标；
- *仁慈性*：当其他 Agent 需要帮助的时候，它会尽力去提供支援。

也就是说，协作 Agent 是一类动态地与环境进行交互、仁慈地与其它 Agent 进行合作来完成它自己的任务的的自治实体。因此协作多 Agent 系统就是给定合适的任务的协作 Agent 的群体。自从协作多 Agent 系统被提出以来，已经被广泛地应用在多种分布式工业领域和其它领域[11, 13, 14, 59, 60, 61, 62, 63, 64, 65, 66, 67]。

在本文对协作多 Agent 系统的主流历史的回顾中，过去的研究被近似地分为四类，即协作协议（coordination protocol）、协作应用系统（coordination application system）、强数学协作模型（strong mathematical coordination model）和协作扩展（coordination extension）：

1. 协作协议：从 D. Gelernter 和 N. Carriero (1992) [52] 我们知道协作协议对于正确和高效的协作是至关重要的。多 Agent 系统中主要的协作协议是 CNET（Contract NET，合同网）[68, 69]、KQML（Knowledge Query Manipulation Language，知识查询操作语言）[70, 71]和 FIPA ACL（Agent Communication Language，Agent 通信语言）。
   (a) CNET：合同网是最著名和最高效的协作协议之一，它通过 Agent 之间的合同来进行协作；

(b) KQML：KQML 是 Agent 交互中最著名的知识表达的标准。它是一个在 Agent 之间结表示和共享知识，以便于获得其它 Agent 的知识和目标情报的构化语言；

(c) FIPA ACL：由 FIPA（Foundation for Intelligent Physical Agents, 智能实际 Agent 组织） 提出的 ACL [72, 73]则是一个广泛使用的 Agent 交互的语言标准。ACL 精确地完整地定义了通信语法和通信的实现。

2. 协作应用系统: 利用协作或协作多 Agent 系统已经构建了许多应用系统，其中很多都含有组织或者隐含组织：

(a) FA/C 系统：功能精确/协作系统（Functionally accurate, cooperative system, FA/C）是一个将由分布性导致的不确定和错误作为一个整合的网络问题进行解决的分布式处理系统[74, 75]；

(b) HTN：层次任务网络（Hierarchical Task Network, HTN）通常将程序上的解释给推导规则，例如 SIPE（System for Interactive Planning and Execution，交互规划和执行系统）[76]；

(c) PGP：部分全局规划（Partial Global Planning, PGP）通过组合很多种协作技术形成一个统一的框架，提供了一个在分布式感知网络（distributed sensor network）中协作多个人工智能系统的框架[77]；

(d) ARCHON 系统：即 ARchitecture for Cooperative Heterogeneous ONline system（协作异构在线系统结构），改项目致力于构件一个软件结构使得已经存在的多个专家系统在给定的复杂环境中处理不同方面的决策问题，或者是提供一个相对更有效的友好地互助的协作方法[78, 79]；

(e) Dis-DSS：即 Distributed Dynamic Scheduling System（分布式动态调度系统），它是一个对不同的航空公司和机场的不同资源进行协作的一个多 Agent 系统，同时也是机场资源管理系统（Airline Resource Management, ARM）的一部分[11, 13]；

(f) GPGP：通用部分全局规划（Generalized Partial Global Planning）是 PGP 之上的基于 TÆMS（Task Analysis, Environment Modeling, and Simulation，任务分析环境建模和模拟）的一族扩展。相对于 PGP，GPGP 采用完工期限来调度任务，它允许 Agent 异构。它交换少量的全局信息，并采用多层抽象来进行通信[80]；

(g) Dis-HTN：分布式 HTN 是 HTN 的一个分布式版本，比如 DSIPE（Distributed System for Interactive Planning and Execution，分布式 SIPE）。DSIPE 首先

基于约束，然后引用分布式 Agent 来自动识别并共享信息，最后从部分规划中生成全局规划[81]；

(h) 资源约束 GPGP：K. Decker 和 J. J. Li 扩展了任务结构化表示语言 TÆMS [82, 83]，使它拥有了表示资源约束的能力，然后将 GPGP 扩展成资源约束 GPGP 来处理一个天然的资源约束的分布式问题：医院病人调度（Hospital Patient Scheduling, HPS）[84]；

(i) 类黄蜂：类黄蜂（wasp-like）是一个基于天然黄蜂群体的多 Agent 协作算法，该算法与蚁群优化（ACO）有些相似，但是并不完全相同[64, 85]。

3. 强数学协作模型：在本文中我们将这些模型称为“强”是因为这些模型通常需要比图灵机（Turing Machine, TM）更强的机器来进行求解，现在的计算机想要在合理的时间内完整地计算这些算法或模型是不可能的。M. d'Inverno (1997) 说这些数学方法主要是面向博弈理论和模态或时态逻辑[86]：

(a) 联合意图：联合意图（Joint Intention）是协助 Agent 构造联合承诺（joint commitment）、拥有联合责任（joint responsibility）和执行联合行动（joint action）的协作方法[87, 88]；

(b) BOID: 由 J. Broersen, M. Dastani, J. Hulstijn, Z. Huang 和 L. van der Torre (2001) 所提出的 BOID（Beliefs-Obligations-Intentions-Desires，信念-义务-意图-愿望）结构[89]扩展于传统的 BDI（Belief-Desire-Intention，信念-愿望-意图）结构[51, 90]，并带有更多的社会行为的构型；

(c) Coo-BDI：Coo-BDI（Cooperative BDI，协作 BDI）是基于 dMARS（Distributed Multi-Agent Reasoning System，分布式多 Agent 推理系统）的详细规格[86]，并且对传统的 BDI 结构在很多方面进行了扩展，比如主要愿望和外部事件的分离、引入 Agent 之间的协作、引入缺省计划等[91, 92]；

(d) RCS: RCS（Rolegraph Coordination Strategy，角色图协作策略）是面向在运行环境中每个 Agent 只能处理部分信息的 Agent 团队工作而提出的[93]。它通过图匹配规则来表达团队意图的层次关系；

(e) FSTS：FSTS（Fuzzy Subjective Task Structure，模糊主观任务结构）被提出来用必要的概念（比如方法、任务、方法间的关系等）来抽象协作问题[94]。模糊逻辑（Fuzzy logic）[95] 被引用到该模型中来处理面向任务的环境的描述上的不确定性；

(f) MDP: MDP（Markov Decision Process，Markov 决策过程）是一个通过探索可能世界状态（possible world-states）和每个可能世界状态的行动执行后产生的到新的世界状态的所有的概率，然后构建规定了每个世界状态中最优行动

的最优策略，从而进行决策的模型。协作领域方面有一些 MDP 方向的研究包括：

i. MMDP：多 Agent MDP（Multiagent MDP）[96]，

ii. DEC-MDP 和 DEC-POMDP：非集中式 MDP（Decentralized MDP）和非集中式部分可见 MDP（Decentralized partially observable Markov decision process, Decentralized POMDP）[97, 98]，

iii. COM-MTDP：通信多 Agent 团队决策问题（COMmunicative Multiagent Team Decision Problem）[99]，

iv. CMDP 和 EMTDP：单 Agent 约束 MDP（constrained MDP）和扩展多 Agent 团队决策问题（Extended Multiagent Team Decision Problem）[100]。

4. 协作扩展：协作扩展主要关注除了 Agent-Agent 协作关系之外的尚待开发的领域：

(a) P. Scerri, L. Johnson, D. V. Pynadath, P. Rosenbloom, N. Schurr, M. Si 和 M. Tambe (2003) 讨论了在“人-Agent”和“人-Agent-人”中的协作[101]；

(b) A. Omicini, A. Ricci, M. Viroli, C. Castelfranchi 和 L. Tummolini (2004) 给出了一个全新的概念“协作人工品”（Coordination Artifact），看起来是一个可视化的、便于使用的、面向工业开发的类似于智能 Agent 的概念[102]。

简要回顾了协作技术的历史之后，我们对协作 Agent 和第 5 所给出的 DSAFO 算法就会有一个更好的理解。

在上述这些进展中，有一些值得我们注意的研究。其中一个就是 GPGP 方法，它是解决调度问题的一个通用方法，但是它是一个 NEXP-完全（Non-EXPonential-complete，非指数-完全）复杂度的算法[82, 98]。另外，K. Sycara, S. Roth, N. Sadeh 和 M. Fox (1991) 给出了一个在 Agent 之间传输抽象资源需求（纹理）的一个机制[24]。每个 Agent 使用这些纹理来形成一个总体的系统资源需求的模型。这个模型通过很多种启发式来分配资源。此外，A. Garland 和 R. Alterman (2004) 指出没有自治性的 Agent 可能会通过公共知识和协作工作得比自治 Agent 更好[103]。

2.4.3 Agent 组织模型

关于组织的定义，目前并没有一个一致认可的定义，但是大部分定义都基本上拥有一些相同的原则。一般的来说，组织具有以下特点[49]：

- 大尺度问题解决技术
- 包含多个主体（人、人工 Agent、或兼有）

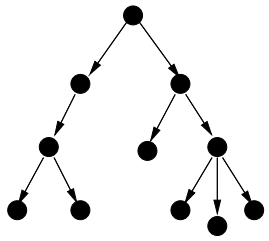
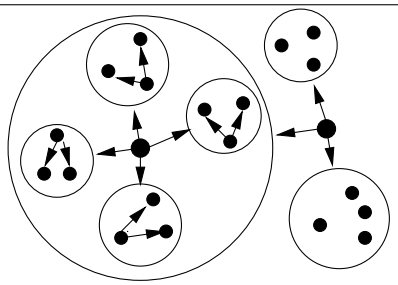
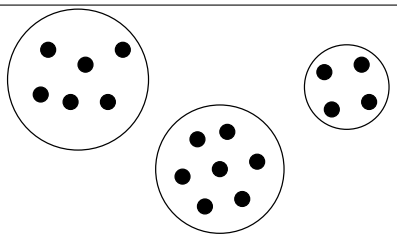
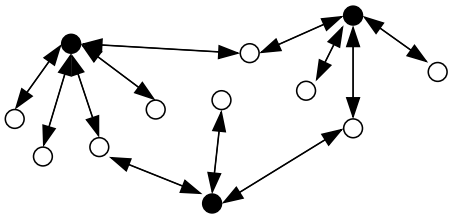
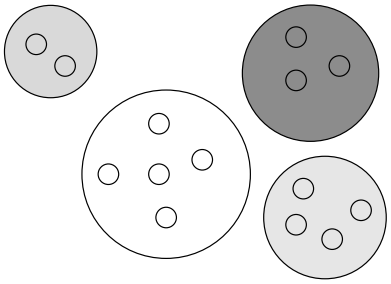
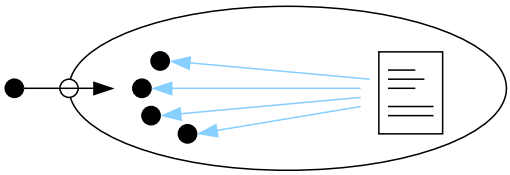
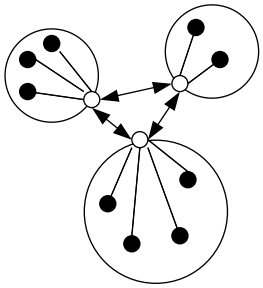
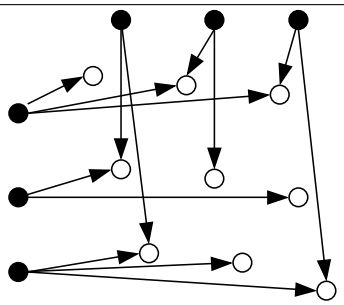
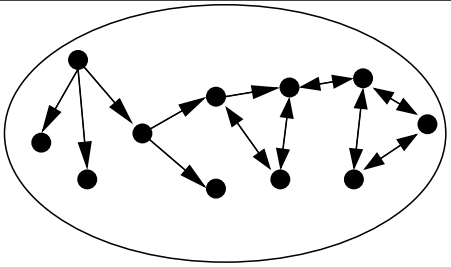
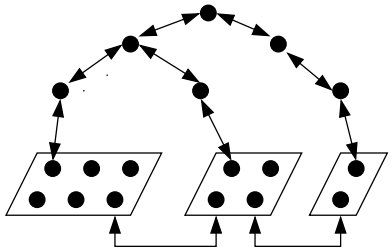
- 从事一个或更多的任务；组织是进行活动的系统
- 目标驱动（goal directed）（不过，目标可能会改变，也可能不是模糊的也不是清晰的，也可能不能被所有的组织成员所共享）
- 能够影响环境，也能够被环境所影响
- 拥有知识、文化、记忆、历史和明显不同于任何单独 Agent 的特别能力
- 拥有不同于单个 Agent 的合理的地位。

自从 M. S. Fox (1979, 1981) 首次提出了多 Agent 系统中的组织[104, 105]，许多种组织已经被研究得越来越越来越成熟。其中一些已经被良好地形式化和在单独的应用中量化研究[106, 107, 108, 109, 110]。如表 2–1 所示，B. Horling (2004) 给出了十种多 Agent 系统组织模型的总结[111]：层次（Hierarchy）、合弄（Holarchy）、联合（Coalition）、市场（Marketplace）、集合（Congregation）、社会（Society）、联邦（Federation）、矩阵（Matrix）、团队（Team）和复合组织（Compound organization）。每一种都有自己独特的优点和缺点：

1. **层次**[104, 105, 112, 113]将 Agent 安排成一个树型，这可能是最早的结构化的例子。它善于分治（divide-and-conquer）、映射到许多领域，并且在大尺度问题上比较有用。但是它不稳定，而且容易引起瓶颈效应和延迟。
2. **合弄**[114, 115]是来自于 Koestler 的 holon[10] 的意义，它是一个层次化的、嵌套的和自相似的结构，可应用来精确地描述很多系统。它充分利用了个体的自治性。但是对于很多问题几乎不可能它来进行建模，而且它的性能也难以预测。
3. **联合**[112, 116]已经被博弈理论界研究了多年，而且被证明在现实世界的经济和多 Agent 系统中都是非分有用的策略。联合利用了众多 Agent 的力强，但是它的短期益处可能还赶不上组织构建的耗费。
4. **市场**[117, 118]是一个基于市场的组织。买入 Agent（显示为白色）可能向一个公共的物件集合请求或者发出竞标。这类物件可能包括共享资源、任务、服务或货物。市场在分配和竞价方面的优势被充分发挥，但是它可能引起恶意的（malevolent）竞争且它的分配过程十分复杂。
5. **集合**[119, 120]是一群绑定在一起的 Agent 个体，从而形成一个平坦的组织，来产生额外的好处。不同于其它组织模型，集合被假定为长期存在的（long-lived），而且

[10] 术语合弄（holon）首次出现于作家 Koestler 的书《机器中的鬼魂》（The Ghost In The Machine, Koestler, 1967）。在这本书中，Koestler 尝试表达一个统一的、描述性的基于嵌套和自相似所组织的物理系统理论，很多系统都是合弄，比如天文学的星系。

表 2–1: 多 Agent 系统组织模型的分类

 层次	 合弄
 联合	 市场
 集合	 社会
 联邦	 矩阵
 团队	 复合组织

并不是由一个单独的明确的目标所形成的。长期存在的集合能够强化 Agent 的发现，但是有时候预先设定的条件可能过分约束群体。

6. **社会**[121, 122]直觉地就是长时期存在的、具有社会结构的、从我们的社会经验而派生出来的组织模型。不同于其它组织模型，Agent 社会天然就是一个开放式系统。社会提供良好的公共服务而且具有良好定义的约定和传统。但是它潜在地就是很复杂的，而且 Agent 可能需要额外的社会相关的能力。

7. **联邦**[123, 124]就大大不同了。所有的 Agent 都拥有一些共同特性，但是部分 Agent 让出了它们自己的部分的自治性，让给了一个仲裁人来代表一个群体。联邦擅长于匹配、代理服务和转换服务，同时它也有利于构建动态 Agent 池。但是这些仲裁人成为了组织的瓶颈。

8. **矩阵**[80]放松了“一个 Agent，一个管理者”的约束，允许许多个管理者或者同等级 Agent 来影响一个 Agent 的活动。如此一来，Agent 的能力就被共享了，Agent 的行为（可望）被影响得对大家更好。但是共享的 Agent 成了一个潜在的争夺焦点，而且这样的 Agent 可能十分复杂。

9. **团队**[105, 124, 125, 126]由一些愿意共同工作、拥有一致目标的协作 Agent 所组成。团队面向更大粒度的问题，而且它是以任务为中心的（task-centric）。但是它在面对大尺度问题的时候，通信量增长得非常快。

10. **复合组织**[77]允许一个系统包含若干个不同的组织模型。一个系统可能用一个组织来进行控制，另一个来进行数据交流，第三个来进行发现，等等。优点和缺点分别由所组合成分的组织模型的所赋予。

2.5 多价 π-演算

R. Milner (1991) 所开发的多价 π-演算（polyadic π-calculus）[127]是对并行系统（比如移动系统）进行建模的强有力的工具。它是基于 π-演算的[128, 129]。因为多 Agent 系统的天然的并发性[51]，多价 π-演算也就自然地被引入到多 Agent 系统的形式化中来，该工作分别由 T. Rorie (1998) [130] 和 W. Jiao 和他的同事(1999) [131, 132]所完成。接下来我们对 π-演算和多价 π-演算的定义进行简要的回顾。

定义 4 (π-演算[127]) π-演算中最基础的实体是“名字”(name)。名字，可以是无限多个，表示为 $x, y, \ldots \in \mathcal{X}$; 它们没有结构。在开始的基本版本的 π-演算中，另外只有一

类实体“进程”(process)。进程表示为 $P, Q, \ldots \in \mathcal{P}$，而且是通过名字按照以下文法所构造的:

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

这里 I 是一个有限的索引集合; 在 $I = \emptyset$ 的时候，我们将所求的和简写为 $\mathbf{0}$。在被加数 π.P 中，前缀 π 表示一个原子动作 (atomic action)，π.P 所执行的第一个动作。有两种类型的基础前缀:

$x(y)$, 将 y 绑定在前缀进程上，意思是“通过名为 x 的连接输入名字 y”;
$\overline{x}y$, 不绑定 y，意思是“通过名为 x 的连接输出名字 y”。

在每种我们都称 x 为动作的主语 (subject)，而称 y 为动作的宾语 (object)；主语是对于输入是正的 (positive)，对于输出是负的 (negative)。

一个名字涉及到一个连接或者一个通道。它有时候可以被认为是在通道的“另外一端”命名一个进程；名字是有两极性的，而且当 x 被用来输入的时候，$\overline{x}$——也就是 x 的补名（co-name）——就被用来输出。

$P|Q$（“P 并 Q”）就是意味着 P 和 Q 是同时处于活动状态的，因此它们可以独立地进行行动。$!P$（“爆炸 P”）意思是 $P|P|\ldots$; 随你希望地有无数多的拷贝。$(\nu x)P$（“在 P 中新的 x”）将名字 x 的使用限制到 P。最后，像 $x(y).\mathbf{0}$ 和 $\overline{x}y.\mathbf{0}$ 的进程太普遍了，所以我们省略末尾的“$.\mathbf{0}$”，仅仅分别写作 $x(y)$ 和 $\overline{x}y$。

多价输入和多价输出是通过一些缩写被引入的：

$$\begin{array}{lll} x(y_1 \cdots y_n) & \text{表示} & x(w).w(y_1).\cdots.w(y_n) \\ \overline{x}y_1 \cdots y_n & \text{表示} & (\nu w)\overline{x}w.\overline{w}y_1.\cdots.\overline{w}y_n \end{array}$$

随着这些缩写的引入，我们就能够继续定义多价 π-演算。

定义 5 (多价 π-演算[127]) 从单价到多价演算，我们增加了一些形式: 抽象(abstraction)$F, G, \ldots$ 和具体(concretion)$C, D, \ldots$，分别称他们为集体主体 $A, B, \ldots$。我们使用 $\alpha, \beta, \ldots$ 来包括名字和补名，并用 $\overrightarrow{x}, \overrightarrow{y}, \ldots$ 来表示名字向量，它们的大小分别为 $|\overrightarrow{x}|, |\overrightarrow{y}|, \ldots$。

$$\begin{array}{rcl} \text{标准进程 (Normal process)} & : & N ::= \alpha.A \mid \mathbf{0} \mid M + N \\ \text{进程 (Process)} & : & P ::= N \mid P|Q \mid !P \mid (\nu x)P \\ \text{抽象 (Abstraction)} & : & F ::= P \mid (\lambda x)F \mid (\nu x)F \\ \text{具体 (Concretion)} & : & C ::= P \mid [x]C \mid (\nu x)C \\ \text{主体 (Agent)} & : & A ::= F \mid C \end{array}$$

$M+N$ 表示的是不是 M 就是 N 被随机选择并执行，有且仅有一个进程能够被执行。$(\lambda x)P$ 是一个表示参数定义的本质的抽象；也就是说

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

同时 $[x]C$ 是一个具体，“[ ]”来源于输出前缀的形式：

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

另外为了阅读方便，在本文中我们将前缀形式 $\overline{x}y_1 \cdots y_n$ 一般加上括号写作 $\overline{x}\langle y_1 \cdots y_n \rangle$。

第三章　飞机地面作业调度：形式化和特性

在本章中，我们列出了面向飞机地面作业调度问题的形式化和建模的假设条件，并以约束满足问题的视角对飞机地面作业调度问题进行了形式化，然后讨论了飞机地面作业调度的特性和从形式化到实际编程应用中的必要的改变。

3.1　假设

在现实飞机地面作业调度问题中，有相当的多不确定干扰。但是在本文中，我们为了建立一个可计算的模型，对该问题和相关事项进行了一些假设。这些假设列表如下：

- 机场地面交通是理想的（没有交通阻塞）；
- 各种资源和它们的服务作业是理想的（没有机械故障等）；
- 需要的航班数据都能够被采集到；
- 通过航班数据，作业情况能够被近似精确的估计出来；
- 同类资源之间在服务能力上没有差别；
- 每个作业仅仅需要一个资源来完成；
- 每个资源每次只能处理一个作业；
- 每个资源只能被分配不多于 3 个 4-小时工作；
- 每个工作人员只能被分配 2 个 4-小时工作，而且工作时间区间可以任意指定；
- Agent 之间的消息传送是耗费时间的，但是消息一定会按发送顺序到达（可靠性）；

3.2 形式化

飞机地面作业调度问题可以简要地描述如下。存在一个航班集合 $\mathcal{F}$ 和一个资源的集合 $\mathcal{R}$。每个航班都有一个固定的完工期限（起飞），并且包含按照一定顺序排列的一系列作业需要被完成。每个作业给定了一个整数的服务时间，和一个额外包含了交通时间和准备时间的长一点的资源使用时间（资源计划时间）。每个作业需要一个资源来处理，而且是不间断地处理。每个资源同时只能处理一个作业。一个可行的调度将每个作业分配个一个资源，并给出满足航班起飞要求的每个作业的服务开始时间。则飞机地面作业调度问题就是寻找可满足所有航班要求的的最小资源耗费的调度计划。

接下来我们将给出一些形式化的函数来表示将飞机地面作业调度的服务作业分配到飞机地面作业资源上的过程。如此一来，飞机地面作业调度相关的概念就可以从一种约束满足的观点来进行形式化描述了。

3.2.1 飞机地面作业

引理 1 *一个航班的服务作业（$\mathcal{O}$, 其中包括降落和起飞）以及它们的时间上的“优先或相等”（不迟于）顺序（$\preceq$）组成了一个格 $<\mathcal{O}, \preceq>$。*

证明. 一个格是一个特殊的偏序集（partial ordered set, poset）$< S, R >$，其中对于任何两个元素的子集 $\{a, b\}$ 都有一个最大下确界（infimum），记为 $\inf\{a, b\}$，和一个最小上确界（supremum），记作 $\sup\{a, b\}$[133]。

首先，$<\mathcal{O}, \preceq>$ 是一个偏序集，因为：

- 封闭性（closure）：对于每个 $< o_1, o_2 > \in \preceq$，有 $o_1, o_2 \in \mathcal{O}$。也就是说，如果有两个作业满足“优先或相等”关系，它们肯定在一个航班的服务作业集合中；
- 自反性（reflexivity）：对于每个 $o \in \mathcal{O}$，有 $o \preceq o$。也就是说，所有作业都不会比自己更迟；
- 传递性（transitivity）：对于所有 $o_1, o_2, o_3 \in \mathcal{O}$，如果 $o_1 \preceq o_2$ 而且 $o_2 \preceq o_3$，那么 $o_1 \preceq o_3$。这就是说，如果 o_1 不迟于 o_2 而且 o_2 不迟于 o_3，那么 o_1 不迟于o_3。

其次，$<\mathcal{O}, \preceq>$ 中确实存在一个最大下确界和一个最小上确界:

- 最大下确界：对于所有 $o \in \mathcal{O}$，有 $landing \preceq o$，也就是说，一个航班的所有的作业都不能在航班降落 $landing$ 之前开始；

- 最小上确界：对于所有 $o \in \mathcal{O}$，有 $o \preceq takeoff$，也就是说，一个航班的所有的作业都不能迟于航班起飞 $takeoff$。

因此，一个航班的所有服务作业和它们的“优先或相等”关系形成的 $< \mathcal{O}, \preceq >$ 是一个格。 □

定义 6 (飞机地面作业) 一个飞机地面作业（airport ground service）的实例是一个多元组$\langle \mathcal{F}, \mathcal{O}, \mathcal{R}, \mathcal{T}, \preceq, D, F, rt, st, ut, et, tt, \Omega, \gamma, s \rangle$，其中

$\mathcal{F}$	$=$	$\{\varphi_1, \varphi_2, \ldots, \varphi_n\}$	- n 个航班组成的集合;
$\mathcal{O}$	$=$	$\{o_1, o_2, \ldots, o_p\}$	- p 个作业组成的集合;
$\mathcal{R}$	$=$	$\{r_1, r_2, \ldots, r_q\}$	- q 个资源组成的集合
$\mathcal{T}$	$=$	$\{0, 1, 2, \ldots, \Delta\}$	- 离散时间的非负整数集合，并附带有典型的整数运算，比如 $+, -, =, <, \leq$ 等等;
$\preceq$	:	$\mathcal{O} \mapsto \mathcal{O}$	- 优先或相等关系，将 $\mathcal{O}$ 按照航班分解成许多个格，如引理 1 所示;
D	:	$\mathcal{F} \mapsto \mathcal{T}$	- 航班服务完工期限函数;
F	:	$\mathcal{O} \mapsto \mathcal{F}$	- 作业属于哪个航班;
rt	:	$\mathcal{O} \mapsto \mathcal{T}$	- 作业就绪时间;
st	:	$\mathcal{O} \mapsto \mathcal{T} - \{0\}$	- 非零作业服务时间;
ut	:	$\mathcal{O} \mapsto \mathcal{T}$	- 作业准备时间;
et	:	$\mathcal{O} \mapsto \mathcal{T}$	- 作业重置时间;
tt	:	$\mathcal{R} \times \mathcal{O} \mapsto \mathcal{T}$	- 一个作业的交通时间;
Ω	:	$\mathcal{R} \times \mathcal{T} \mapsto \mathcal{O} \cup \{\emptyset\}$	- 当前正在处理哪个作业，当没有执行作业或者给定了多于一个作业的时候返回 $\emptyset$。

3.2.2 飞机地面作业调度满足

定义 7 (飞机地面作业调度满足) 飞机地面作业调度满足是对于一个飞机地面作业实例 $\langle \mathcal{F}, \mathcal{O}, \mathcal{R}, \mathcal{T}, \preceq, D, F, rt, st, ut, et, tt, \Omega, \gamma, s \rangle$ 寻找下面两个函数的运算:

γ	:	$\mathcal{O} \mapsto \mathcal{R}$	- 分配资源;
s	:	$\mathcal{O} \mapsto \mathcal{T}$	- 分配服务开始时间，

同时满足以下要求:

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

其中，$o\prec o'$ 是 $o\preceq o'\wedge o\neq o'$ 的缩写。

换句话说，飞机地面作业调度满足就是寻找一个有效的调度方案，来利用有限的资源及时地满足服务作业的过程。

引理 2 $p\sim \mathrm{O}(n)$，也就是说，总作业数量 p 的复杂度是 $\mathrm{O}(n)$。

证明. 对于每个航班，存在有限个需要满足的作业，也就是说，$\forall\varphi\in\mathcal{F}$，存在

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

表示的是属于 φ 作业的集合。而且这样的集合必然是非空的，而且大小具有一个数量上界，因此很明显有 $\forall\varphi\in\mathcal{F},|OP_\varphi|=\mathrm{O}(1)$。因此，

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

□

引理 3 $q\sim \mathrm{O}(n)$，也就是说，总资源数量 q 的复杂度是 $\mathrm{O}(n)$。

证明. 考虑所有作业的资源使用需求，一定存在一个全局最小资源使用时间：

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

而且如果一个资源总共只服务一个资源，资源的（最大的）数量就等于 p。将数量为 p 的资源使用安排到时间段 $\Delta+1$（$\mathcal{T}$ 的大小）上，总共需要的资源的最小值肯定不小于 $\frac{p\times\mathrm{rut}_{\min}}{\Delta+1}$。那么，

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

因此，$q\sim\mathrm{O}(n)$。 □

定理 1 飞机地面作业调度满足属于 $\mathcal{NP}$-完全（$\mathcal{NP}$-complete）类。

证明. 首先，很明显地任何飞机地面作业调度满足都可以在多项式时间内进行解的验证，因此飞机地面作业调度满足一定属于$\mathcal{NP}$。

在接下来的证明中，我们将一步一步地从划分问题（Partition）把飞机地面作业调度满足推演出来。其中，划分问题是最知名的基础$\mathcal{NP}$-完全的问题之一[1]。

一个划分问题的实例包含一个有限的集合 A 和对于每个元素 $a \in A$ 的“大小” $s(a) \in \mathbb{Z}^+$。该问题的目标是寻找一个子集 $A' \subseteq A$，使得

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

然后，我们来讨论一个名为“多处理器调度”（Multi-processor scheduling）的问题。该问题的一个实例包含一个优先的“任务”集合 A，和每个任务 $l(a) \in \mathbb{Z}^+$ 的任务的“长度” $l(a) \in \mathbb{Z}^+$，以及一组 m 个“处理器”，和一个“完工期限” $D \in \mathbb{Z}^+$。该问题的目标是寻找一个 A 的划分 $A = A_1 \cup A_2 \cup \ldots \cup A_m$，使得

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

考虑多处理器调度问题的这样一个约束（restriction）：仅仅允许 $m = 2$ 而且 $D = \frac{1}{2} \sum_{a \in A} l(a)$。很清楚地，我们可以发现该问题确切地被约束成了一个划分问题。也就是说，每一个划分问题都可以被映射到一个 $m = 2$ 且 $D = \frac{1}{2} \sum_{a \in A} l(a)$ 的多处理器调度问题。所以，

$$\text{划分问题} \leq_{\text{p}} \text{多处理器调度问题}$$

因此，多处理器调度是是一个$\mathcal{NP}$-完全问题。M. R. Garey, E. G. Coffman Jr., 和 D. S. Johnson (1978, 1979) 曾给出了该问题的介绍和一个类似的证明[134, 1]。

接下来考虑飞机地面作业调度满足的这样一个约束：仅仅允许对于所有作业 $o \in \mathcal{O}$ 有 $ut(o) = et(o) = tt(o) = 0$ ，$rt(landing) = 0$，且对于所有 $f \in \mathcal{F}$ 航班 $D(f) = D_F$ 的实例。该约束代表着一些飞机地面作业调度满足的特例（暂时不考虑其现实可能）：没有作业准备时间、重置时间、和交通时间，所有航班具有统一的降落和起飞时间。

那么通过这个约束，每个多处理器调度的实例都可以同态映射到一个飞机地面作业调度满足中，其中 $q = m$，$p = |A|$，$D_F = D$，且一个作业 $o \in \mathcal{O}$ 表示一个任务 $a \in A$。因此，

$$\text{多处理器调度问题} \leq_{\text{p}} \text{飞机地面作业调度满足.}$$

所以，

$$\text{划分问题} \leq_{\text{p}} \text{飞机地面作业调度满足.}$$

所以，飞机地面作业调度满足属于$\mathcal{NP}$-完全类。 □

幸运的是，实际国内机场和航空公司的这类资源通常都是富余的，因此寻找到一个可行的飞机地面作业调度满足方案并不难。不过，花费多少资源和多少人力对于航空公司和机场却是非常重要的。所以我们需要将可行的飞机地面作业调度满足方案进行优化。

命题 1 飞机地面作业调度满足是一类特殊的分布式约束满足问题（Distributed Constraint Satisfaction Problem, Dis-CSP）。

证明.（简要证明）根据定义 3，如果认为每个资源是一个 Agent，对于它所能做的所有作业是它的变量，同时依赖关系、就绪时间和完工期限是约束，那么飞机地面作业调度满足就能够在分布式约束满足问题中被形式化。 □

3.2.3 飞机地面作业调度问题

定义 8 (飞机地面作业调度问题) 飞机地面作业调度问题是寻找一个有效的飞机地面作业调度满足计划 γ_0, s_0，使得

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

其中 Γ 是所有可满足的解决方案的集合，$|ran(\gamma)|$ 是 γ 的值域的大小。

也就是说，飞机地面作业调度问题是寻找最优的（最低耗费的）调度的过程。

定理 2 飞机地面作业调度问题是一类 $\mathcal{NP}$-完全问题。

证明. 从定义 7、定义 8 和定理 1 中，我们可以清晰地得出：飞机地面作业调度问题的复杂度不低于 $\mathcal{NP}$-完全类。接下来我们将证明飞机地面作业调度问题的复杂度上限也处于 $\mathcal{NP}$-完全水平。

让我们考虑一个资源约束最紧的情况：只有非常少的资源，以至于只可能存在唯一一个飞机地面作业调度满足方案。在这个极端情况下，飞机地面作业调度问题的解决过程就变成了寻找这个唯一的飞机地面作业调度满足方案的过程，也就是说，这个飞机地面作业调度问题是 $\mathcal{NP}$-完全问题。在这个计算中，二进制数组 B 的所有可能的赋值假设都可能被尝试，因此这个暴力搜索（brute-force search）过程是 $\mathcal{NP}$-完全的。

一般地来说，飞机地面作业调度问题的解决过程可以被变形为如下两个过程：进行如上的暴力搜索，和在找到一个方案的时候对比调度资源耗费。尽管在不同情况下对比的计算可能差别很大，但是由于一般来说存在 $\mathrm{O}(2^{|B|}) = \mathrm{O}(2^b)$ 个可行解，所以这个对比就是 $\mathcal{NP}$-完全时间耗费的。

因此，综上两个过程，飞机地面作业调度问题是一类 $\mathcal{NP}$-完全问题。 □

命题 2 飞机地面作业调度问题是一类特殊的分布式约束满足问题。

证明.（简要证明）根据定义 8，飞机地面作业调度问题是一类特殊的飞机地面作业调度满足，但是带有更多约束。而飞机地面作业调度满足是一类特殊的分布式约束满足问题（命题 1）。因此，飞机地面作业调度问题是一类特殊的分布式约束满足问题。 □

命题 3 飞机地面作业调度问题是一类特殊的 Job-Shop 调度问题（Job-Shop Scheduling Problem, JSSP）。

证明.（简要证明）根据定义 1 和定义 8，作业前后关系代表的就是先后顺序要求，而作业的唯一的资源占有代表的就是设备冲突问题。因此，飞机地面作业调度问题是一类特殊的 Job-Shop 调度问题。 □

3.2.4 不确定性飞机地面作业调度满足

定义 9 (不确定性飞机地面作业调度满足) 不确定性飞机地面作业调度满足是飞机地面作业调度满足的一个特例，其中对于每个作业 $o \in \mathcal{O}$，它的就绪时间 $tr(o)$ 都是是个概率函数。

例如，对于下个月的航班计划，考虑各种原因可能造成的航班延误和其它干扰的情况下，生成一个总是可行的飞机地面作业调度满足方案，这就是一个不确定性飞机地面作业调度满足。

命题 4 不确定性飞机地面作业调度满足是一类特殊的动态 Job-Shop 调度问题（Dynamic Job-Shop Scheduling Problem, DJSSP）。

证明.（简要证明）类似定理 3，不确定性飞机地面作业调度满足首先是一类特殊的 Job-shop 调度问题，不过，带有不确定的作业就绪时间。这正是动态 Job-shop 调度问题（DJSSP）[135]。 □

还有一个复杂类 PSPACE-完全（PSPACE-complete），表示由图灵机（Turing machine, TM）在多项式空间可解的一类问题[1]。众所周知，PSPACE-完全比 $\mathcal{NP}$-完全更加复杂。

定理 3 不确定性飞机地面作业调度满足是一类 PSPACE-完全问题。

证明.（简要证明）根据命题 4，不确定性飞机地面作业调度满足是一类特殊的动态 Job-Shop 调度问题。而动态 Job-Shop 调度问题可以在多项式时间内约简为带量词的布尔表达式（Quantified Boolean Formula, QBF）[135]。而带量词的布尔表达式 QBF 是

一个众所周知的 PSPACE-完全问题[1]。因此不确定性飞机地面作业调度满足一定属于 PSPACE-完全类。 □

3.2.5 不确定性飞机地面作业调度问题

定义 10 (不确定性飞机地面作业调度问题) 不确定性飞机地面作业调度问题是寻找一个可行的不确定性飞机地面作业调度满足方案 γ_0, s_0，使得

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

其中 Γ 是所有可行方案的集合。

换句话说，不确定性飞机地面作业调度问题就是寻找不确定作业状态下最优的（最低耗费的）调度的过程。

3.3 实际飞机地面作业调度编程

在实际的飞机地面作业调度编程中，我们将 $\mathcal{R}\times\mathcal{T}$ 缩写为 $\mathcal{P}$ 来表达服务计划，那么一个调度就成为 $sch:\mathcal{O}\mapsto\mathcal{P}$。一个作业是一个包含航班号、作业类型、停机位、就绪时间（RT）、最晚完成时间（LFT）、期望服务时间（ST）、期望准备时间（UT）和期望重置时间（ET）的元组。

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

另外一个计划通常是一个包含待服务的作业、所使用资源、交通时间（TT）和承诺服务开始时间（CSST）的元组。这些计划往往是多步骤的。

$$\text{plan} \stackrel{\text{def}}{=} < \text{op}, \text{res}, \text{TT}, \text{CSST} >$$

实际上并不存在现成的 TT (即$tt(\gamma,o)$)。对于一个计划，一个 Agent 可以通过上一个作业所在位置和当前服务目标所在位置来计算地面交通距离。在一些国际机场，可能存在若干条通往目标位置的地面交通路线，因此 Agent 应该具有简单的空间推理能力，以计算出最短的（或者最合理的）路径。然后将这个地面交通距离除以 5km/h，机场车辆最大地面移动速度，来取得合适的地面交通时间。

3.4 特性

根据命题 1 和命题 2，飞机地面作业调度满足和飞机地面作业调度问题是两类特殊的分布式约束满足问题。但它不是一个单纯的分布式约束满足问题。因为仅仅取得飞机

地面作业调度满足方案对于航空工业来说是不够的，需要计算得出最少的资源调度方案，也就是最少的耗费。

通过命题 3，飞机地面作业调度问题是一类特殊的 Job-Shop 调度问题。但是飞机地面作业调度问题也不是一个单纯的 Job-Shop 调度问题。飞机地面作业调度问题本身也不是以最小化航班服务总时间（makespan）为目标的，而是以寻找一个满足所有航班和作业的可行调度为目标的，继而寻找最小资源耗费的可行解决方案。而且一些作业的服务完成并不代表着资源就空闲了，比如，在典型作业卸行李（unload baggage）的过程中，对于一个行李牵引车可能有四个行动：

- 开车到停靠在目标航班的传送带旁边；
- 用传送带将货舱里的行李装载到行李小车上；
- 开车到旅客行李接收传送带的一端；
- 将行李卸到旅客行李传送带上。

第四章　分布式动态调度模型

在本章中，根据飞机地面作业调度问题的动态本质，给出了一个分布式动态调度模型。另外还描述了一个面向飞机地面作业调度问题的动态调度环境（运行时）run-and-schedule。

4.1　必要性

飞机地面作业调度问题中的一个主要问题是航班服务的（时间）动态本质。一个航班可能被各种原因所延误，而且特殊的飞机、旅客或天气都可能为为航班带来特殊服务。因此，使用一个固定的航班预测序列来进行飞机地面作业调度问题或者不确定性飞机地面作业调度问题都是不合适的。对于飞机地面作业调度问题，动态调度框架十分必要。

飞机地面作业调度问题中的另一个主要问题是飞机服务需求中的变量，也就是说，资源或者作业通常都有自己的需求和约束。例如，分配航空地面支持车辆必须满足作业的各自的需要、车队的空间分布、车辆作业准备和（某些情况下的）由飞机移动优先引起的地面交通阻塞。这样一来，会有很多约束被带进集中式的优化模型中。这些约束的数量使得这样的模型十分困难（通常是$\mathcal{NP}$-困难或者是更难）。因此，分布式调度模型对飞机地面作业调度问题也是十分重要的。

M. E. Aydin 和 E. Öztemel (2000) 开发过带有学习 Agent 系统的动态调度环境[136]。他们采用实时的（也就是动态的）信息搜集机制来处理不确定性干扰。分布式模型方法也已经被自然地提出来处理局部资源/作业范围中的约束，比如 Dis-CSP 方法[42]、面向市场的计算（market-oriented programming）[137]以及G. İnalhan, D. M. Stipanović 和 C. J. Tomlin (2002) 所做的分析[138]等等。因而，分布计算使得优化问题更可行。而且 G. İnalhan 等人 (2002) 指出，各个解局部之间的关联或者局部凸面比较弱，那么集中式的 Pareto 最优解在分布式方法中也可以被保证[138]。很幸运的是，从定义 8 可以看出，飞机地面作业调度问题中的局部关联和全局约束都不强。

总之，研究面向飞机地面作业调度问题的分布式动态调度环境是非常有益的。

4.2 模型概览

如图 4–1 所示，我们开发了一个分布式动态调度模型，并由四个模块组成：

民航输入系统： 机场与航空公司的与飞机地面作业调度相关的那些航空支持信息系统的集合。这些系统能够用它们的实时的精确的数据来支持飞机地面作业调度问题的优化和求解，例如航班计划时间（估计就绪时间和完工期限）、航班飞行距离（估计清洁和加油）、旅客流量（估计下飞机和登机）、货物量（估计货物装卸）等等。没有这些数据，就不能得到精确的和有效的服务调度计划。不过，这些系统是空间上分离的，而且从不同系统的数据也有可能不一致。因此也需要一个数据整合层；

Run-and-schedule 环境： 是面向调度算法的运行时[139]。这个环境收集并整合航班数据，并提供一个时钟，为下一个模块（分布式调度算法）提供支持。Run-and-schedule 在接下来的一节中会有详细介绍；

分布式调度算法： 就是优化调度的算法。这里可以应用很多算法，在本文中，我们在这里开发了一个新的算法 DSAFO，详情参考第 5 章；

现实世界的资源： 就是飞机地面服务的人力和设备，比如工程师、航空地面支持设备等等。关于现实世界资源的实时信息和作业分配情况也会被调度算法和资源实体只见同步地进行交换。该模块式调度的最后阶段——将优化调度应用到实际生产中。

4.3 Run-and-scheduling

Run-and-schedule 是一个动态分布式调度环境[139]。它不停地观察半小时之内将降落或者起飞的航班信息，从一些航空支持系统——例如运行控制系统（Flight Operations Control system, FOC）、机场运行数据库（Airport Operation Data Base, AODB）、航班信息显示系统（Flight Information Display System, FIDS）、订票系统（Billing system）和其它运行和管理信息系统——中搜集对飞机地面作业调度有用的航班信息数据，例如 VIP 标识、航程、装卸行李量、货邮信息、旅客信息、特别服务需求等等。在成功地采集数据后，run-and-schedule 对数据进行一致的和可靠的数值和格式整合。

这样一来，分布式调度算法中的作业服务目标和计划就可以根据这些信息合理地、动态地、而且比较精确地计算出来。

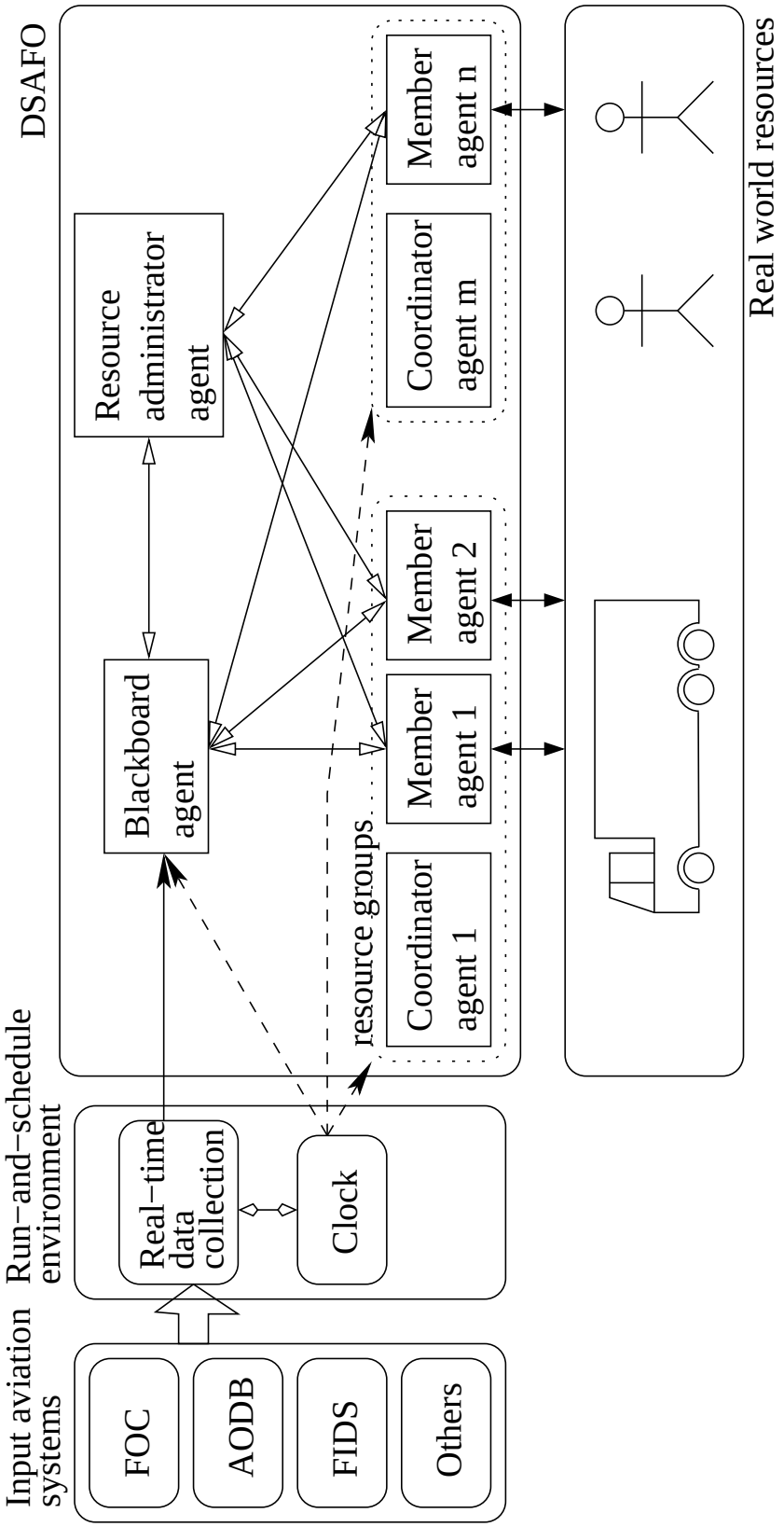


图 4–1: 飞机地面作业调度的动态分布式模型

同时，通过一致化一个心跳周期 Heartbeat，并给定一个算法基准时间，它就可以给出一个公用的时钟。该时钟为分布式算法中的所有主体、航班任务、服务计划和资源都都提供了一个同步的时间，从而保证了分布式调度运算过程中的作业分配的同步性。

根据 S. J. Russell 和 P. Norvig (1995) 的观点[140]，run-and-schedule 和现实世界资源组合在一起形成了一个可访问的、确定性的、动态的和离散的多 Agent 运行时：

- 可访问的（accessible）：一个 Agent 的感知可以使它获得 run-and-schedule 和资源的全部状态；
- 确定性的（deterministic）：资源的下一个状态完全取决于当前状态和 Agent 所选择的对应行为；
- 动态的（dynamic）：run-and-schedule 所提供的信息在 Agent 思考的时候可能会改变；
- 离散的（discrete）：只有数量有限的、明确定义的感知和行为。

第五章 DSAFO：概述、设计和实现

在本章中，提出了一个多 Agent 算法 DSAFO（Dynamic Scheduling Agents with Federation Organization，带有联邦组织的动态调度 Agent），来解决飞机地面作业调度满足和飞机地面作业调度问题。同时也给出了算法的形式化和复杂度分析。

5.1 概述

DSAFO，也就是 Dynamic Scheduling Agents with Federation Organization，是一个新颖的优化飞机地面作业调度满足和问题的多 Agent 方法[139]。相对于第 2.2 节给出的其它调度技术，我们可以给出 DSAFO 的归类，如图 5–1 所示。

如图 4–1 所示，DSAFO 的运行过程可以如下列出：

1. 从 run-and-schedule 环境读取实时的有用的航班数据；
2. 根据收集来的数据，将航班服务目标分解为许多作业；
3. 利用多 Agent，将解空间动态地分为许多合理的划分；
4. 采用局部启发式来处理每个划分；
5. 同步地采用各个划分之间的协作来达到全局意义上的解的优化；
6. 同步地将解分配到现实世界中去。

而且从图 4–1 中可以观察到，有一个 Agent Blackboard 负责作业分配的决策制定，有一个 Agent ResourceAdmin 负责资源分配的决策制定，有许多 Member Agent 负责动态地将资源和作业进行匹配。Coordinator Agent 则被设计来提高 Member 之间的协作效率。

另外，DSAFO 和 run-and-schedule 还与一些已有的研究享有一些共同点：

1. 与 K. Sycara, S. Roth, N. Sadeh 和 M. Fox. (1991) [24]所提出的方案享有相似的空闲资源因子的同步机制，

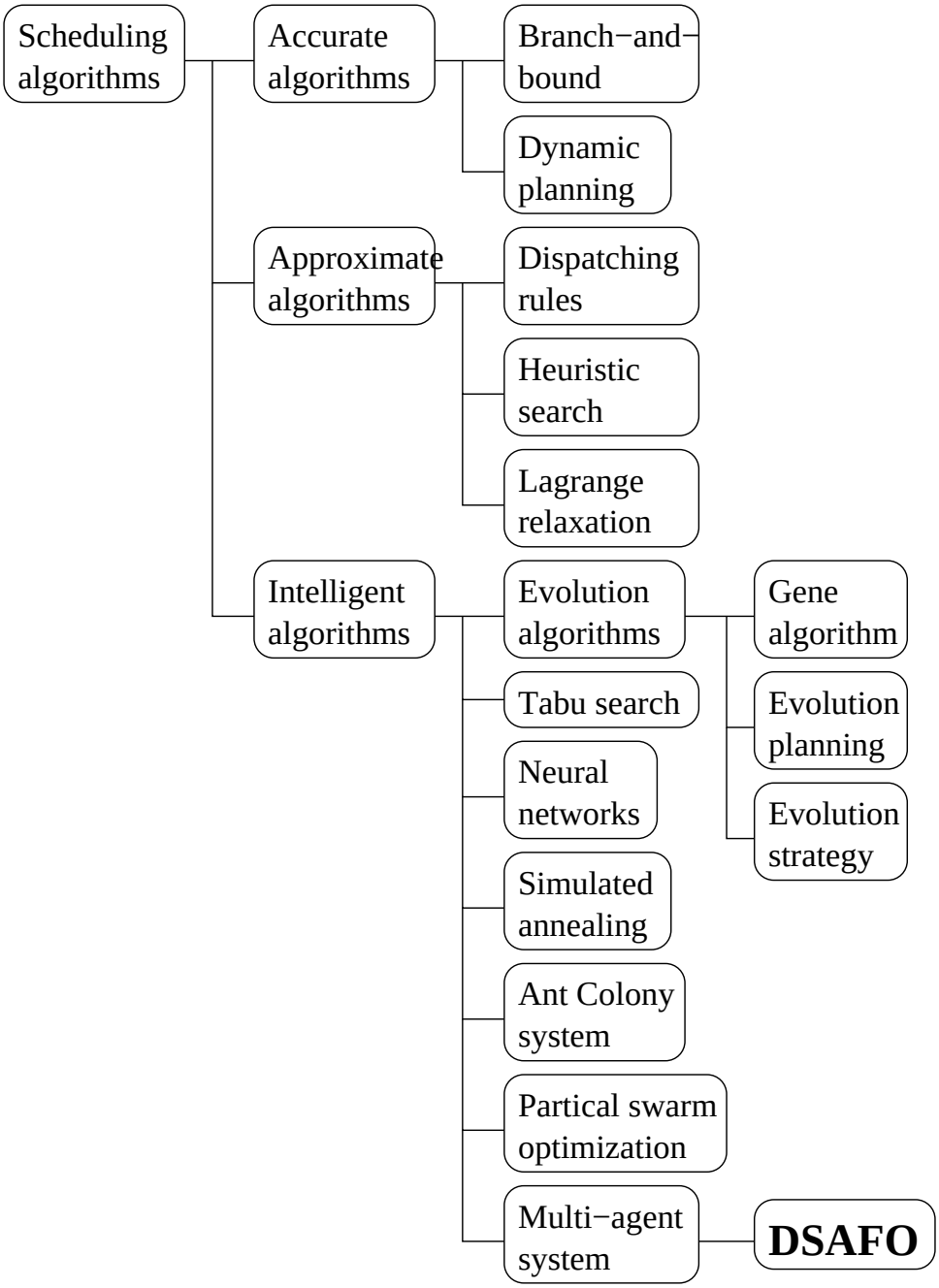


图 5–1: DSAFO 的算法归类

2. 与 Dis-DSS [11, 12]享有相似的资源借用机制和黑板机制，

3. 与 M. E. Aydin 和 E. Öztemel (2000) [136]所提出的模拟环境（simulated environment）享有相似的动态环境特性，

4. 与异步弱承诺搜索（asynchronous weak-commitment search）[42, 44, 45] 享有相似的启发式回溯机制。

5.2 DSAFO 中的策略

从内部来看，DSAFO 算法可以被认为是一个带有局部启发式和全局协作的多 Agent 方法。

5.2.1 局部启发式

DSAFO 中有两种局部启发式，两个都是 EDD（Earliest Due Date first，最早完工期限优先）。其中一个用来根据加权最晚完成时间（Latest Finish Time, LFT）来选择出最优先的待分配作业，另外一个根据最早承诺服务开始时间（Committed Service Start Time）来为某个作业指定资源。

EDD 是一个根据领域知识解决小尺度问题可靠的和高效的方法。它可以迅速地在低阶多项式时间内为 $\mathcal{NP}$-完全问题甚至 PSPACE-完全问题计算出一个解。但是当尺度增大的时候，EDD（和其它启发式）算法会陷入局部极小解。鉴于这个原因，DSAFO 将局部的启发式（EDD）引入了动态 Agent 来跳出局部极小解。

该 EDD 策略最终实现于 Blackboard 和 Member 角色中，如第 5.3.1 小节和第 5.3.3小节所述。

5.2.2 全局协作

全局协作是一个通过 Agent 之间的资源空闲程度的信息交换从而实现一个完整的 Agent 间资源借用过程的一种机制。

通常情况下在多 Agent 调度中，一个 Agent 控制一部分的资源，因此最好的决策可能无法从局部的不完整的资源信息中所制定，比如，当一个 Agent 在不知道另一个 Agent 所拥有的资源能给出一个更好的方案的时候，就尝试用自己的资源来处理作业。Agent 之间的全局协作就是主要用来解决这个问题。

在 DSAFO 中，全局协作最终实现于 Member Agent 和进行辅助的 Coordinator Agent。而且在拥有相同类型资源的众多 Member Agent 之间有两类关系：伙伴（buddy）和竞争（competitor）。如果两个 Member Agent 共享同样类型的资源和同样的作业，那么它们就是竞争关系；如果它们共享同样类型的资源但是面向的作业不同，那么它们就是伙伴关系。一群伙伴总是愿意通过友善地资源借用来进行互相帮助的。

5.3 DSAFO 中的 Agent 角色

DSAFO 总共包含四种 Agent 角色：Blackboard、ResourceAdmin 、Coordinator 和 Member。角色 Blackboard 主动地获取航班数据，角色 Member 主动地完成作业，检查自己的资源是否过期，并且同步伙伴资源列表，角色 Coordinator 主动地同步它的所有成员 Member 的资源空闲程度列表。其余的行为都是被动发生的，也就是说，其它的任何行为都是直接或者间接被这四个主动行为中的一个（或多个）驱动着的。

从实际实现上来看，有一个唯一的 Blackboard，一个唯一的 ResourceAdmin，一些 Coordinator 和更多的 Member。这些 Agent 通过如图 5–2 所示的通信信道来组织在一起。

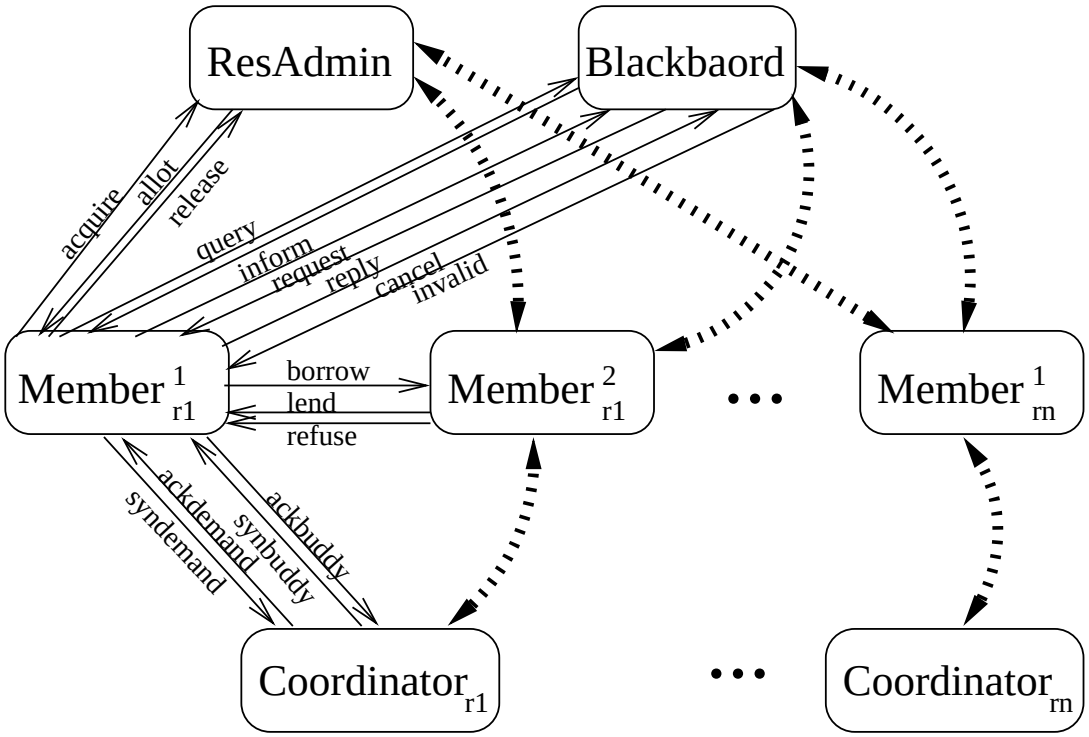


图 5–2: Agent 通信信道

这些通信信道组成了几个联邦（Federation）组织模型[111]。联邦组织由部分让出了它们自己的部分的自治性给一个仲裁人的 Agent 和仲裁人组成。联邦擅长于匹配、代理

服务和转换服务，同时它也有利于构建动态 Agent 池[107, 111]。

在接下来的小节中，一个角色一个角色地给出了程序风格的伪码，伪码的精细级别恰恰足够精确表示所有通信信道的使用情况。

5.3.1 Blackboard

角色 Blackboard 被设计来感知航班数据、将航班任务分解成许多作业、并且为作业指定由 Member 承诺的服务计划。Blackboard 有一个主动行为：

- 读取航班数据：每个 Heartbeat（心跳周期）都从 run-and-schedule 环境中主动地获取航班数据；

和三个被动行为：

- 响应作业查询：答复消息发送人所指定的作业类型的作业队列里面最优先的作业信息；
- 响应作业承诺：如果服务计划可行，答复消息发送者“成功”，否则答复“失败”；
- 响应作业取消：移除一个已经承诺的服务计划。

这些异步的作业指派其实是弱承诺。因为一旦由 Blackboard 所认可的服务计划同相关的 Member 的角度的实际资源使用相冲突，Blackboard 就会接收到一个回溯消息，然后服务计划就会被取消。这与异步弱承诺搜索[44, 45]很相似。

根据图 5–2，只有一个 Agent 扮演这个角色。通过这个 Agent，一个可靠的、完整的和一致的全局调度才可能从许多 Member 的局部视点中综合出来。而且所有的 Member 的最终服务计划都不会产生冲突，因为它们最终都和 Blackboard 保持一致。

Blackboard 的行为和通信信道使用情况如图 5–3 所示:

```
procedure Blackboard_thread
  operations←Ø;
  flightdata←Ø;
  while(true)
    flightdata←get_data();
    operations←operations∪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×Blockfactor);
      end switch
      if (end_of_time)
        terminate_algorithm();
        output_result ();
      end if
    end while
end procedure

function weighted_EDD
  input taskType;
  p←+∞;
  op←NIL;
  for t in unsolved_op_in_30min(taskType)
    if (t.priv == VIP)
      return t;
    else if (t.LFT − now < p)
      op←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
```

图 5–3: Blackboard 的行为和信道使用

5.3.2 ResourceAdmin

角色 ResourceAdmin 是拥有两个被动行为的资源管理者：

- 响应资源请求：搜索某个类型的空闲资源，并在可能的情况下向消息发送者分配一个；
- 响应资源释放：重新标记一个资源可用，通常情况下这需要在分配之后经过 4 个小时，也就是半个“一人一天工作量”（man-day），即4-小时工作。

这个角色也由一个 Agent 扮演，它的通信信道使用和行为如图 5–4 所示：

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

图 5–4: ResourceAdmin 的行为和信道使用

5.3.3 Member

角色 Member 负责一个类型（或几个类型）的作业，它有三个主动行为：

- 查询作业：主动地给 Blackboard 发送作业查询指令，因为 Member 总是以处理作业为它的愿望；
- 资源过期检查：发送资源释放指令给 ResourceAdmin，当一个资源已经被分配了 4 小时后；

- 同步伙伴列表：每隔一段时间向 Coordinator 发送同步伙伴列表的指令；

和七个被动行为：

- 准备作业服务计划：从 Blackboard 接收到作业的目标后，它尝试在本地独立地制作一个服务计划并发送回该计划；如果本地制作计划失败，Member 会尝试向它的伙伴借用资源；
- 确认服务计划：
  - 成功：从 Blackboard 接收到计划“成功”消息，它会尝试在本地确认服务计划。如果确认失败，即资源已经被另外的作业占用，它会发送回取消指令。接下来，如果作业中的资源是被其它 Agent（伙伴）所占有，它会将当前的消息复制给该资源持有人；
  - 失败：从 Blackboard 接收到计划“失败”消息，如果作业服务计划中的资源被其它 Agent 所占有，会将当前的消息复制给该资源持有人；
- 借出资源：从一个 Agent 那里接收到资源借用消息后，Member 会尝试构建本地服务计划，然后发回本地的结果（可能是成功或是不是）。Member 对来自伙伴的帮助请求总是很慷慨的；
- 确认借出：
  - 成功：将服务计划复制并发送给 Blackboard；
  - 失败：给伙伴列表中的下一个伙伴发送资源借用请求；如果已经达到伙伴列表的末端，则向 ResourceAdmin 发送资源请求指令；
- 加入新资源：从 ResourceAdmin 接收到资源分配消息后在本地添加所指定的属性的新资源；
- 响应资源空闲情况：返回自己的资源空闲程度消息给 Coordinator；
- 更新伙伴列表：利用从 Coordinator 来的数据更新伙伴列表。

实际上，Member Agent 在资源选择的时候也有一个弱承诺搜索的策略，也就是说，当 Blackboard 碰到两个服务计划冲突的时候，一个服务计划就会被通过 invalid 信道发送回去，来取消那个计划。

Member 的通信信道使用和行为如图 5.3.3 所示：

```
procedure Member_thread
  buddies←Ø;
  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.resp!=self)
            sendto(content.plan.resp, 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(Coordinator_resType, REQBUDDY, self);
        lastSyn←now;
      end if
  end switch
  block(Heartbeat×Blockfactor);
 end while
end procedure
```

图 5–5: Member 的行为和信道使用

5.3.4 Coordinator

角色 Coordinator 用来协作所有的拥有同类型资源的 Member。它有一个资源空闲程度同步的主动行为：

- 资源空闲程度同步：发送资源空闲程度查询指令给它的每个成员 Member；

和两个资源信息的被动行为：

- 更新资源使用状况：通过来自每个 Member 的数据来更新对应成员的资源空闲状况；
- 响应伙伴列表同步：发送返回通过资源空闲状况排好序的伙伴列表。

Coordinator 的通信信道使用和行为如图 5–6 所示：

```
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←Ø;
  for m in members
    if (m.opType!=mem.opType)
      bd += n;
    end if
  end for
  bd.sortBy(DESCENDING);
  return bd;
end function
```

图 5–6: Coordinator 的行为和信道使用

5.4 一个形式化的小结

除过第 2.5 节中已有的多价 π-演算的定义之外，本文通常把元组一类的名字序列缩写为名字向量，比如，一个作业元组

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

可以被缩写为“$\vec{op}$”。同时名字向量中的一个名字可以通过“$\triangleright$”运算符来取得，例如，一个作业 $\vec{op}$ 的最晚完成时间（Latest Finish Time, LFT）就是“$op \triangleright LFT$”。

在多价 π-演算的形式下，DSAFO 有一些参数[139]：$SchOts$ 表示可调度的作业类型的集合，对于每个作业类型 $t \in SchOts$，$Agent_t$ 表示可以向 t 提供服务的 Agent 的集

合，$Resource$ 是一个表示资源类型的集合，对于每种资源类型 $r \in Resource$，$Otinres_r$ 表示 r 所能提供服务的资源类型的集合，$Heartbeat$ 表示一个多长的机器时间等效于实际时间 1 分钟的时间段，$Clockzero$ 表示算法时钟开始的时刻。那么我们有

$$\begin{aligned}\text{DSAFO} \stackrel{\text{def}}{=} & (SchOts, Agents_t, Resources, Otinres_r, Heartbeat, Clockzero) \\ & (\nu\, query_t^a, inform_t^a, request_t^a, reply_t^a, cancel_t^a, invalid_t^a, reqbuddy_t^a, synbuddy_t^a \\ & , borrow_t^a, lend_t^a, refuse_t^a, reqdemand_t^a, syndemand_t^a, acquire_t^a, allot_t^a, release_t^a) \\ & (\text{BLACKBOARD}|\text{RESADMIN}|\text{MEMBER}_t^a|\text{COORDINATOR}_r) \\ & (t \in SchOts, a \in Agents_t, r \in Resources)\end{aligned}$$

5.4.1 Blackboard

BLACKBOARD 动态地感知半小时内的航班信息，并将它们分解为许多作业，然后将这些作业分配上来自 Member 的服务计划。

参数 $Flights$ 是一个表示所有已知航班的集合，对于每个航班 $f \in Flights$ 和 作业类型 $t \in SchOts$，Ops_f^t 给出了一个唯一的（某航班的某种类型的）作业，而且它是一个表示作业元组 operation 的结构。

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

$BbFunc$ 是一个用来支持 BLACKBOARD 行为的函数的集合。例如，通道 $clock$ 总是被监控着。当从 $clock$ 接收到两个参数（输出通道 tch 和时间 $time$）的时候，值 $(time - Clockzero)/Heartbeat$ 就会被通过 tch 输出。也就是说，函数 $clock(tch, systime)$ 将时间 $time$ 从系统机器时间转换为算法的时钟，并输出于 tch。

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

很明显的，π-演算模型在涉及到角色的行为细节方面特别繁琐和冗长。因此接下来的内容中，我们对角色的行为的建模在有些地方可能并不会像第 5.3 节那么详尽。不过 DSAFO 的主要特性和概貌在接下来的模型中仍然很清晰。

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

5.4.2 ResourceAdmin

RESADMIN 是资源管理者。当接手到资源请求的时候，RESADMIN 就会对应的空闲资源并在可能情况下进行分配。每经过 4 个小时的工作，资源就会被再次重新释放。另外，一个资源最多每天只能服务 12 小时，也就是一个半的一人一天工作。

参数 $Reslist_r$ 是 r 类型的所有已知资源的集合，$Historylist_t$ 则是资源使用的历史纪录，目的在于确保每个资源每天的使用不超过预定的 3 次（即 12 小时/ 4 小时）。

$$
\begin{aligned}
\text{RESADMIN} \stackrel{\text{def}}{=} & (Reslist_r, Historylist_r)(RaFunc|Respreq_t^a|Resprelease_t^a)\\
& (r \in Resources, t \in SchOts, a \in Agents_t)\\
Respreq_t^a \equiv\ & !resreq_t^a(begintm).(\nu\, ch)(\overline{available\_res_t}\langle ch, begintm\rangle|ch(\overrightarrow{res})\\
& .[\overrightarrow{res} \neq nil])\overline{allot_t^a}\langle res \triangleright name, begintm\rangle.\overline{set\_busy}\langle\overrightarrow{res}, \text{true}\rangle\\
& .\overline{log\_allot}\langle\overrightarrow{res}, t, a\rangle))\\
Resprelease_t^a \equiv\ & !release_t^a(resname).(\nu\, ch)(\overline{getresfromhash}\langle ch, resname\rangle\\
& |ch(\overrightarrow{res}).[\overrightarrow{res} \neq nil]\overline{log\_release}\langle\overrightarrow{res}\rangle.\overline{set\_busy}\langle\overrightarrow{res}, \text{false}\rangle)
\end{aligned}
$$

5.4.3 Member

一个 Member Agent MEMBER_t^a 是名为 a 且负责作业类型 t 的。它总是渴望处理作业。在 MEMBER_t^a 从 BLACKBOARD 接收到待处理的作业目标后，它就会构建合适的服务计划并进行申请。每个 Member 同时也乐意对他的伙伴给予帮助。

对于每个作业类型 $t \in SchOts$，每个 Agent $a \in Agents_t$，MEMBER_t^a 有一些参数：$Syncycle$ 一个预设的伙伴资源列表的整数的同步周期，$Resource_t^a$ 是一个保存局部资源使用历史和当前承诺的集合，$Remoteres_t^a$ 是一个保存伙伴借用给自己的远程资源

及其使用历史的集合。

$$
\begin{aligned}
\mathrm{MEMBER}_t^a \stackrel{\mathrm{def}}{=}\ & (Syncycle, Resource_t^a, Remoteres_t^a) \\
& (MbFunc|ActQry_t^a|MkPlan_t^a|AckReply_t^a|AckInvld_t^a|TryLend_t^a \\
& |AckLend_t^a|CallBd_t^a|Ackres_t^a|AckRDmd_t^a|DumSyn_t^a|AckSyBd_t^a) \\
& (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}) \\
& .[\overrightarrow{reslist} \neq nil]\overline{releaseall_t^a}\langle\overrightarrow{reslist}\rangle).)^{+\infty} \\
MkPlan_t^a \equiv\ & !inform_t^a(\overrightarrow{op}).(\nu\, ch)(\overline{makenullplan_t}\langle ch,\ \overrightarrow{op}\rangle|ch(\overrightarrow{n})) \\
& .(\nu\, p)(\overline{locallyplan}\langle p,\ \overrightarrow{n}\rangle|p(\overrightarrow{plan}, res).([res \neq null]\overline{request_t^a}\langle\overrightarrow{plan}\rangle \\
& + [res = null]((\nu c)(\overline{getbuddy}\langle c\rangle|c(\overrightarrow{bud}) \\
& .([\overrightarrow{bud} \neq nil]\overline{try\_borrow}\langle bud \triangleright \mathrm{top}, \overrightarrow{plan}\rangle \\
& + [\overrightarrow{bud} = nil]\overline{acquire_t^a}\langle plan \triangleright EST - 1\rangle)))))) \\
AckReply_t^a \equiv\ & (!reply_t^a(syn, \overrightarrow{pln}).(\nu ch)(\overline{try\_assign_t^a}\langle ch, pln \triangleright res, \overrightarrow{pln}\rangle|ch(ret) \\
& .([ret \neq \mathrm{true}]cancel_t^a\langle\overrightarrow{pln}\rangle|([pln \triangleright resOwner \neq self]\overline{reply_t^{pln \triangleright resOwner}} \\
& + [pln \triangleright resOwner = self]([ret = \mathrm{true}](\overline{setlreq}\langle now\rangle.\overline{enum\_restype}) \\
& + [ret \neq \mathrm{true}][pln \triangleright resp \neq self]\overline{invalid_t^{pln \triangleright resp}}\langle\overrightarrow{pln}\rangle))))) \\
AckInvld_t^a \equiv\ & !invalid_t^a(\overrightarrow{pln}).\overline{freeres_t^a}\langle plan \triangleright res\rangle.[pln \triangleright sender = \mathrm{BLACKBOARD}] \\
& [pln \triangleright resOwner \neq self]\overline{invalid_t^{pln \triangleright resOwner}}\langle\overrightarrow{pln}\rangle \\
TryLend_t^a \equiv\ & !borrow_t^a(\overrightarrow{uplan}, \overrightarrow{lst}, succ, fail).(\nu\, ch)(\overline{locallyplan}\langle ch,\ \overrightarrow{uplan}\rangle \\
& |ch(\overrightarrow{plan}).([plan \triangleright res \neq null]\overline{assign_t^a}\langle\overrightarrow{plan}\rangle.\overline{succ}\langle\overrightarrow{plan}\rangle \\
& + [plan \triangleright res = null]\overline{fail}\langle\overrightarrow{uplan}, \overrightarrow{lst}\rangle)) \\
AckLend_t^a \equiv\ & !lend_t^a(\overrightarrow{plan}).\overline{request_t^a}\langle\overrightarrow{plan}\rangle \\
CallBd_t^a \equiv\ & !refuse_t^a(\overrightarrow{uplan}, \overrightarrow{lst}).(\nu\, c)(\overline{topof}\langle c, \overrightarrow{lst}\rangle \\
& |c(next, \overrightarrow{nlist}).([next \neq null]\overline{next}\langle\overrightarrow{uplan}, \overrightarrow{nlist}, lend_t^a, refuse_t^a\rangle \\
& + [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 \\
& |ch(\overrightarrow{res}).[\overrightarrow{res} \neq nil]\overline{addres2locallist}\langle\overrightarrow{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^a}.\overline{block_t^a}\langle Syncycle \times Heartbeat\rangle.)^{+\infty} \\
AckSyBd_t^a \equiv\ & !synbuddy_t^a(\overrightarrow{lst}).\overline{setbuddy}\langle\overrightarrow{lst}\rangle
\end{aligned}
$$

5.4.4 Coordinator

COORDINATOR$_r$ 用来协作共同拥有资源类型 r 的所有 Member。在取消了资源后备机制后，它就仅仅有信息同步和伙伴列表构建的几个行为了。

参数 $Syncycle$ 是预设的整数的信息同步周期。对于每种资源类型 $r \in Resources$，$Metainfo_r$ 是一个包含它的所有成员 Member 的元级信息的集合，例如资源需求状况和已经完成的作业。

$$\begin{aligned}
\text{CORDINATOR}_r &\stackrel{\text{def}}{=} (Metainfo_r, Syncycle)(CooFunc|Synch_r|Store_t^a|RespBd_t^a) \\
&\quad (r \in Resources, t \in Otinres_r, a \in Agents_t) \\
Synch_r &\equiv (\overline{synall_r}.\overline{block_r}\langle Syncycle \times Heartbeat\rangle.)^{+\infty} \\
Store_t^a &\equiv !syndemand_t^a(dm).(\nu\, ch)(\overline{setval}\langle ch, dm\rangle|ch(demands_t^a) \\
&\quad .\overline{removelist}\langle demands_t^a\rangle.\overline{insertsort}\langle demands_t^a, \text{DESC}\rangle) \\
RespBd_t^a &\equiv !reqbuddy_t^a.(\nu\, ch)(\overline{genbuddylist_t}\langle ch\rangle|ch(\overrightarrow{blst}).\overline{synbuddy_t^a}\langle\overrightarrow{blst}\rangle))
\end{aligned}$$

5.5 复杂度

在飞机地面作业调度满足和飞机地面作业调度问题中，只有有限种类的作业，而且这些所有种类的作业都享有同样的调度策略。也就是说，所有类型的资源享有同态的调度，而且实际上它们的调度方案看起来确实是一个模式的。因此总共的时间耗费对于一个特定种类的作业的调度，比如卸货邮（unload cargo and mail, UC），是成正比例的。因此，

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

5.5.1 UC 的复杂度

UC 是由行李牵引车（baggage tractors, BTs）和小车板来服务的。与引理 3 类似，BT 的数量期望

$$BT^* \sim \text{O}(n)$$

也可以被简单地证明。也就是说 BT 的数量期望正比于航班的数量。由于有一些参数会影响到算法的时间复杂度，则我们假设有 O(n) 多的 BT Member Agent 和合适的其它参数来计算出复杂度上界 $t_{DSAFO_{\text{UB}}}$.

5.5.1.1 来自 Agent 行为的复杂度

涉及到 UC 作业和相关的算法计算的，一共有一个 Blackboard，一个 ResourceAdmin，一个 Coordinator，和 $O(n)$ 多个 BT Member Agent。Blackboard 主动地循环地获取航班数据并且响应不超过 O(n) 个来自 QUERY、REQUEST 和 CANCEL 通信信道的消息。所有这些导致的它自己的行为的时间是：

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

ResourceAdmin 应该响应 O(n) 多个来自 ACQUIRE 和 RELEASE 的消息，因此它的行为的时间复杂度是：

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

每个 Member 主动地检查伙伴资源列表的过期，而且主动地通过 Coordinator 同步伙伴资源列表，还响应来自 IMFORM、REPLY、INVALID、BORROW、LEND、REFUSE、ALLOT、REQDEMAND 和 SYNBUDDY 的消息。当有多个 Member Agent——不是单一的——的时候，很难一一确定究竟有多少消息通过这些通信信道进行了传送。不过，从一个全局的观点来看，很明显地所有 Member 总共处理了 O(n) 次。

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

负责 BT 的 Coordinator 主动地同步所有成员的资源空闲程度，并响应来自 SYNDEMAND 和 REQBUDDY 的通信信道的消息。

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

将上述这些事件复杂度的上界加起来，我们就可以得到由于 Agent 的行为而引起的

UC 作业的复杂度的上界：

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

5.5.1.2 来自 Agent 通信的复杂度

Agent 之间的通信就简单一些了。DSAFO 中总共只有四种类型的通信：Member–Blackboard、Member–Member、Member– Coordinator 和 Member–ResourceAdmin。可以对它们一一考虑如下：

Member–Blackboard 总共存在 $\mathrm{O}(n)$ 多个 UC 作业，因此 INFORM、REQUEST 和 REPLY 应该被触发 $\mathrm{O}(n)$ 次，一般情况下[1]。通信信道一般情况下应该使用不超过 $\mathrm{O}(n)$ 次。信道 REQUEST 应该被使用不超过 $\mathrm{O}(n)$ 次，因为在有限的时间内总共只有不超过 $\mathrm{O}(n)$ 多的 Member Agent 负责 UC 作业。

Member–Member 因为每个 Member Agent 只有不超过 $\mathrm{O}(n)$ 的伙伴，因此每个作业的成功的资源借用最多需要不超过 $\mathrm{O}(n)$ 次借用。

Member–Coordinator 在有限长的调度时间内，对于每 Member–Coordinator 关系，只有 O(1) 多次资源空闲程度的同步。然而由于拥有不超过 $\mathrm{O}(n)$ 多个的成员 Member Agent，总计也是有不超过 $\mathrm{O}(n)$ 次的资源空闲程度同步。

Member–ResourceAdmin 为了满足 $\mathrm{O}(n)$ 个 UC 作业，前文已经详细地证明了需要 $\mathrm{O}(n)$ 个资源（BT）。因此通信信道 ACQUIRE、ALLOT 和 RELEASE 在一般情况下会被使用 $\mathrm{O}(n)$ 次。

将上述分析加起来，由于通信而引起的处理 UC 作业的时间复杂度的上界是：

$$\begin{aligned} t_{\text{UC\_trans}_{\text{UB}}} &= t_{resAllot_{\text{UB}}} + t_{synDemand_{\text{UB}}} + t_{synBuddy_{\text{UB}}} + UC\_op \times t_{opReq_{\text{UB}}} + t_{borrow_{\text{UB}}} \\ &\sim \{\mathrm{O}(n) + \mathrm{O}(n) + \mathrm{O}(n) + \mathrm{O}(n) \times [\mathrm{O}(1) + \mathrm{O}(n)]\} \times t_{\text{trans}} \\ &\sim \mathrm{O}(n^2) \times t_{\text{trans}} \end{aligned}$$

其中，t_{trans} 表示消息的平均传送时间。在现实网络情况下，t_{trans} 通常情况下是几毫秒，因此这个时间是不能够忽略的。

[1] “一般情况”指不考虑一些极端情况，像由于参数 Blockfator 太小等而引起的算法陷于过于频繁的请求失败等情况。

5.5.1.3 UC 的复杂度小结

其余的处理过程（例如 Agent 初始化和 Blackboard 的数据获取过程）就没有多少复杂度，那么 UC 作业处理的复杂度上界是：

$$t_{UC_{\mathrm{UB}}} = t_{\mathrm{UC\_behav_{UB}}} + t_{\mathrm{UC\_trans_{UB}}} \sim \mathrm{O}(n^3) + \mathrm{O}(n^2) \times t_{\mathrm{trans}}$$

5.5.2 DSAFO 的复杂度

那么，DSAFO 的复杂度上界是：

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

另外，我们通过设定 O(1) 多个 Member Agent 负责 UC 作业，可以得出DSAFO 的复杂度下界是：

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

5.6 实现

JADE[2]（Java Agent DEvelopment Framework, Java Agent 开发环境）是一个开发基于兼容 FIPA 标准[3]的 Agent 的内部可控的智能多 Agent 系统的应用程序的软件开发框架[141]。它的目标是通过又兼容性的系统服务和 Agent 来在保证兼容标准的前提下简化开发流程。

JADE 可以被认为是一个实现了 Agent 运行平台和开发框架的 Agent 中间件。它处理常见的独立于应用的 Agent 内部组件和机制，例如消息传送、编码和词法分析、以及 Agent 的生命周期。

JADE 同时也是一个由它的版权所有者 Telecom Italia[4] 所发布的自由软件，它采用开源软件的 LGPL（Lesser General Public License Version 2）协议来进行分发。

我们将 DSAFO 实现在了 JADE 环境中，并且通信信道实现在 FIPA ACL [72, 73] 之上，其语法见附录 A。部分的运行界面参见图 5–7 和图 5–8。

例如，“BT35” 对航班 CA109 的卸货邮作业的服务计划包含两个交通过程（标记为紧密的竖线）、一个服务过程和一个重置过程（标记为 R），另外，在邻接的动作之间都

[2]见 http://jade.tilab.com/
[3]见 http://www.fipa.org/
[4]见 http://www.telecomitalia.com/

留有一个很小的 1 分钟的缓冲时间。当卸货邮服务作业完成之后，装货邮作业才能就绪。另外，在图 5–7 中这个卸货邮作业服务被标记为“CT-Agt0(CT+Agt1)”，这表示资源，也就是“BT35”，是由 CT-Agt0 所有，但是被 CT+Agt1 借用过来完成服务的；在图 5–8 中，该服务计划的目标航班，也就是 CA109 ，是红色的，该颜色也表示这是一个资源借用。

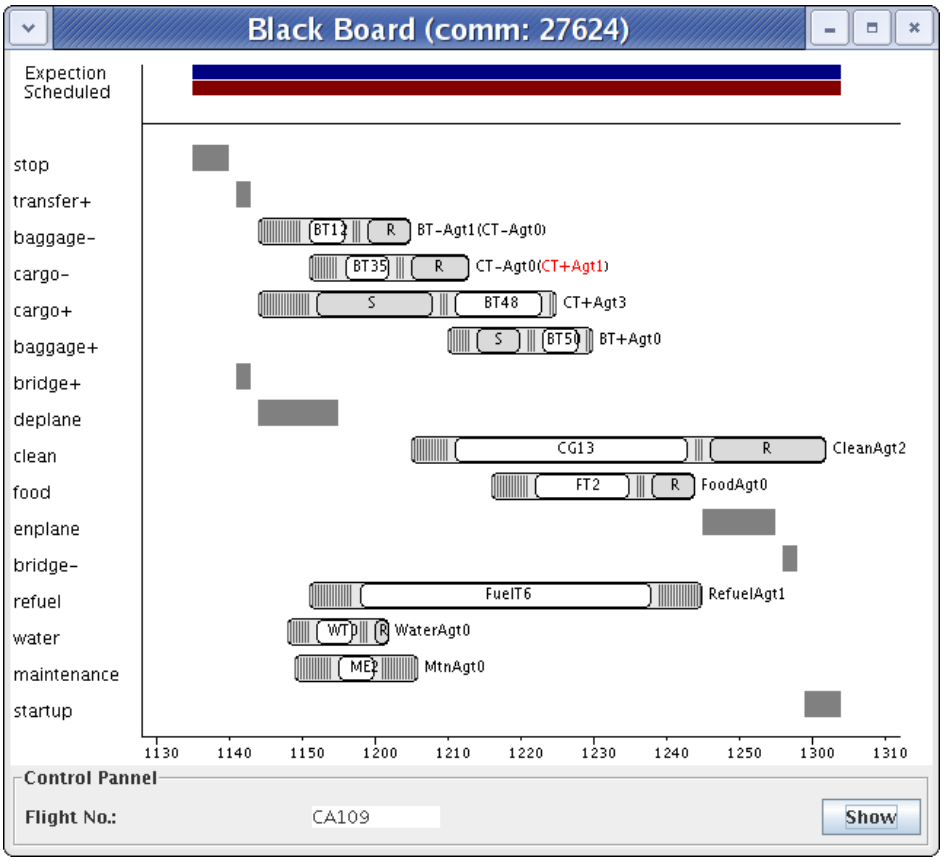


图 5–7: 一个航班的典型调度的甘特图（Blackboard 视点）

因为 DSAFO 实现于 JADE 和 Java 之上，而 Java 运行时是一个操作系统无关的（OS-independent）的虚拟机环境，因此 DSAFO 就可以在任何主流操作系统上运行，比如 Windows、UNIX、Linux、MAC OS 等等。从 JADE RMA GUI（Remote Monitoring Agent, Graphical User Interface，远程监控 Agent 的图形用户界面）可以看到，DSAFO 是一群如图 5–9 所示的（并发的）Agent。JADE sniffer（嗅探器）也可以侦听到 DSAFO 的 Agent 之间的消息传递，如图 5–10 所示。另外，JADE introspector（内窥器）也可以告诉我们 DSAFO 中某个 Agent 的内部状态，如图 5–11。

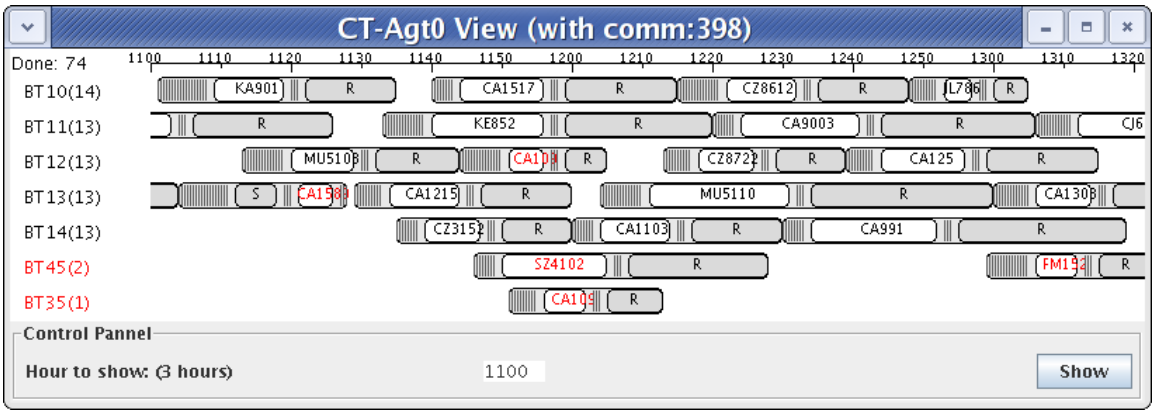


图 5–8: 资源的典型调度甘特图（BT Member Agent 视点）

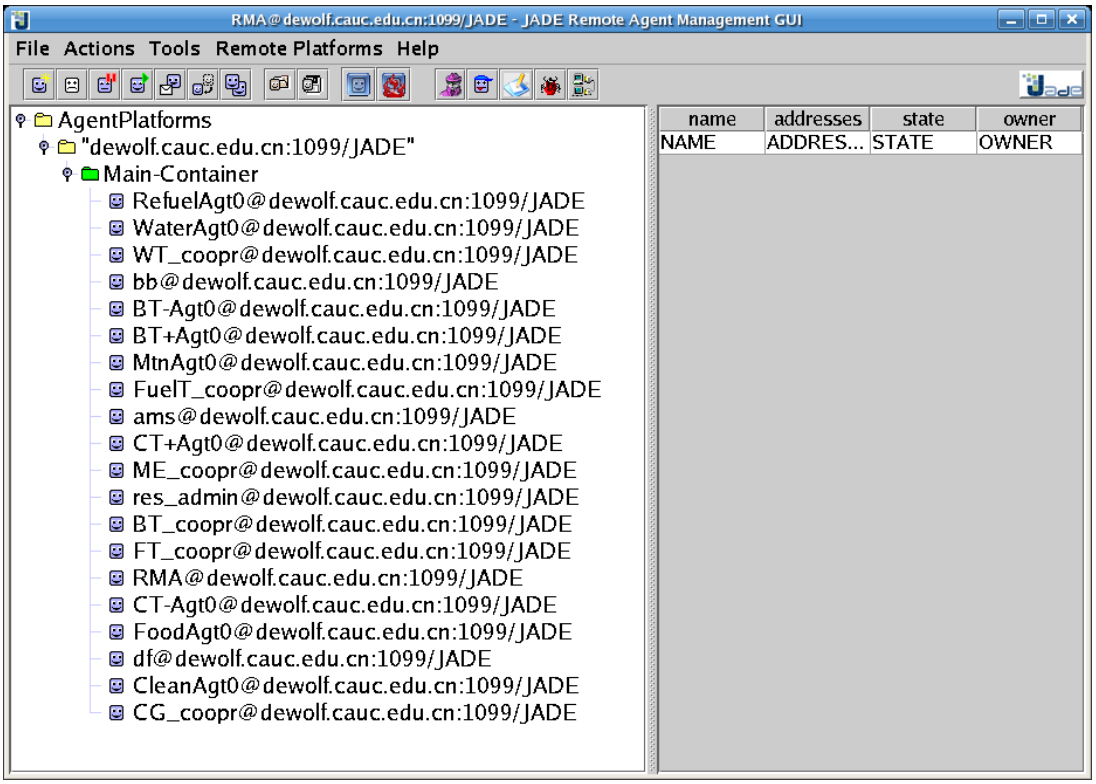


图 5–9: JADE RMA GUI 中的 DSAFO

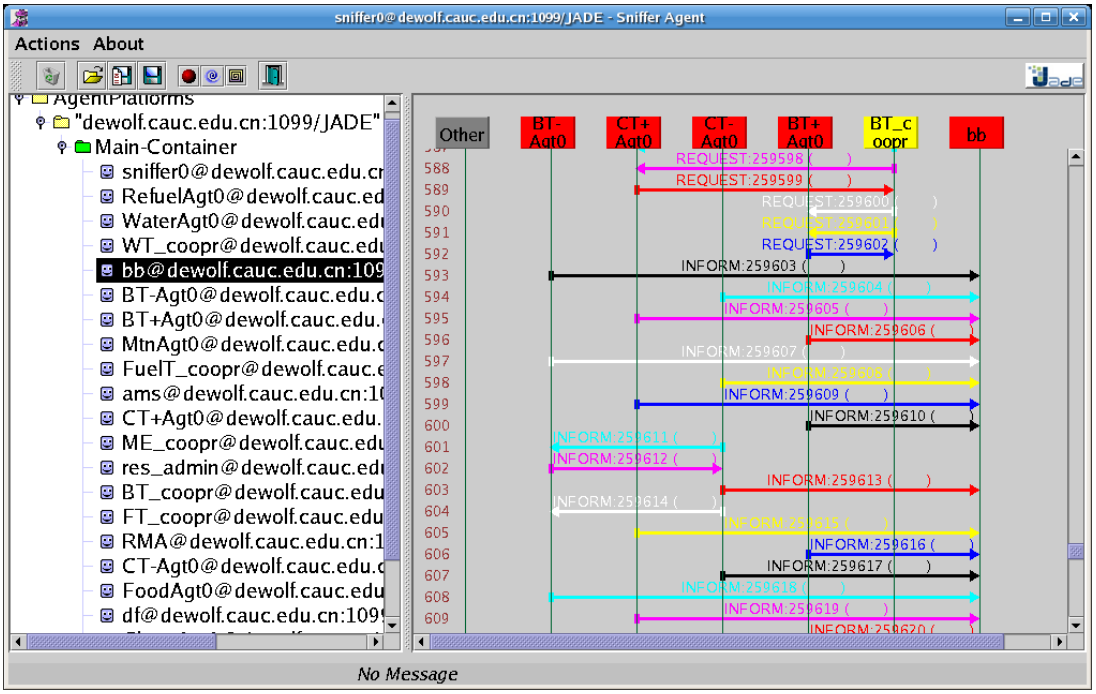


图 5–10: JADE sniffer 观察到的通信

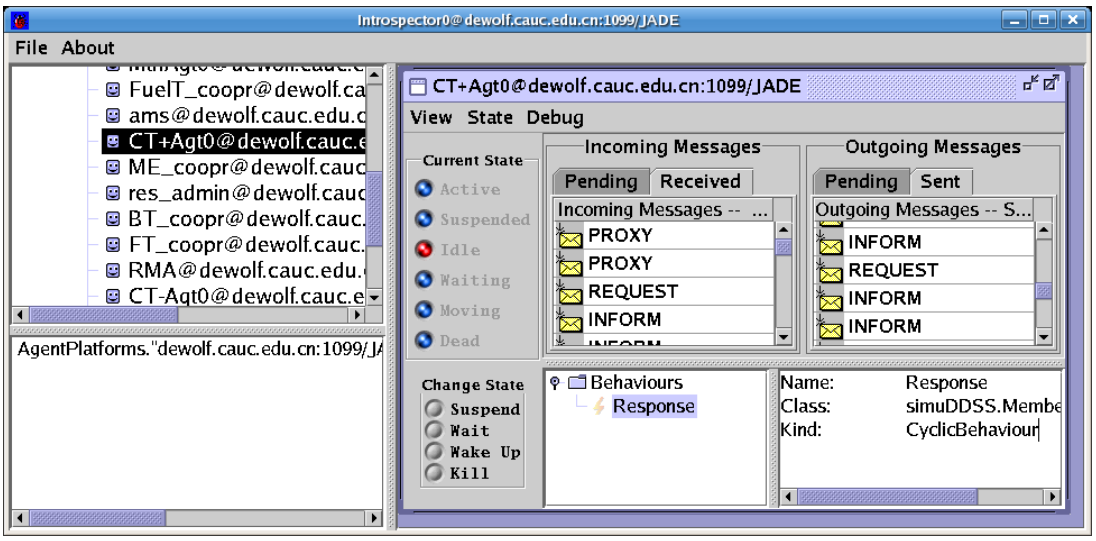


图 5–11: JADE introspector 观察到的 Agent 内部状态

第六章　实验和参数分析

在本章中，一类资源（BT）被选来测试算法 DSAFO；并进行了一系列实验来测试该算法中的四个参数，即 *Member* agent number, Blockfactor, Delayfactor 和 Syncycle，对解的影响；最后给出了一个小结。

6.1　实验设置

飞机地面作业调度实验测试数据来自于某机场某天历史记录中的 252 个实际航班。如图 5–7 所示，每个中转航班被假定具有九个典型作业：卸行李（Unload baggage, UB）、卸货邮（Unload cargo and mail, UC）、装货邮（Load cargo and mail, LC）、装行李（Load baggage, LB）、清洁客舱、配餐、加水排污、加油和机务维检。其他作业被假定为固定的，也就是说，其他作业不需要特别的调度优化就可以完成得很好。这样一来，（每天的）中转航班就总共有 2,268 个作业等待着被分配到几百个各种飞机地面服务资源上。

因为这些作业的种类太多，因此将所有九种作业都进行完整的算法分析或者将各种实验结果都详细地表示出来将会非常麻烦。而且，我们可以注意到所有类型的资源和相对的作业间都采用的是同态的调度技术，同时它们的调度结果看起来确实是处于同一个模式的。也就是说，当调度策略对于一种类型的资源是优秀的时候，该策略对所有的资源调度，一般来说，都会比较优秀。

在这些作业和资源中，BT（baggage tractor, 行李牵引车）相关作业应该是其中最复杂的。因为 BT 这种资源能服务多种资源，如图 1–1 所示。因此一辆 BT，不同于其它资源，可能被连续地分配到不同种类的作业中。例如，UB–UC–LB–UC。因此我们选择 BT 耗费和相关的人员工作量来进行下一节中的算法性能测试。

6.2　算法参数

作为一个多 Agent 算法，因为它的并行 *Member* Agent 协作、不稳定的网络通

信和（甚至）其它一些微小的但是重要的运行环境 JADE 和 Java 语言的敏感的特征，DSAFO 会天然地得到不稳定解。因此我们对每个测试都进行了连续的 250 次计算来得到每组测试的优化结果的分布和统计。

在第 5.3 节对 DSAFO 的介绍中，以伪码形式列出了几个参数：Blockfactor, Delayfactor, Syncycle 和 Reqcycle。除了这四个 Agent 内部的变量外，*Member* Agent 的数量——比如负责 BT 资源的 *Member* Agent 的个数——也是另外一个重要的参数。DSAFO 算法和它的输出在不同程度上受到这些参数的影响。在接下来的小节中，我们将对这些参数通过结果对比和理论分析逐个进行测试和分析。

6.2.1 *Member* Agent 数量和 Reqcycle

Member Agent 数量和 Reqcycle 的结合就是某种作业的申请频度。在以合适的 Reqcycle 保证作业申请频度能够满足测试要求的前提下，不同的 *Member* Agent 数量对算法的影响相当地大。甚至于在所有的参数中，该参数也是最有影响的。当 Blockfactor=1/12, Delayfactor=1/6, Syncycle=5, 和 Reqcycle 分别从 1/6 到 2 时，图 6–1 和图 6–2 显示了不同 BT *Member* Agent 数量下的 BT 解的 3D 分布图和高度分布图，同时图 6–3 给出了这些资源（BT）和人员工作量耗费量的边缘分布曲线。

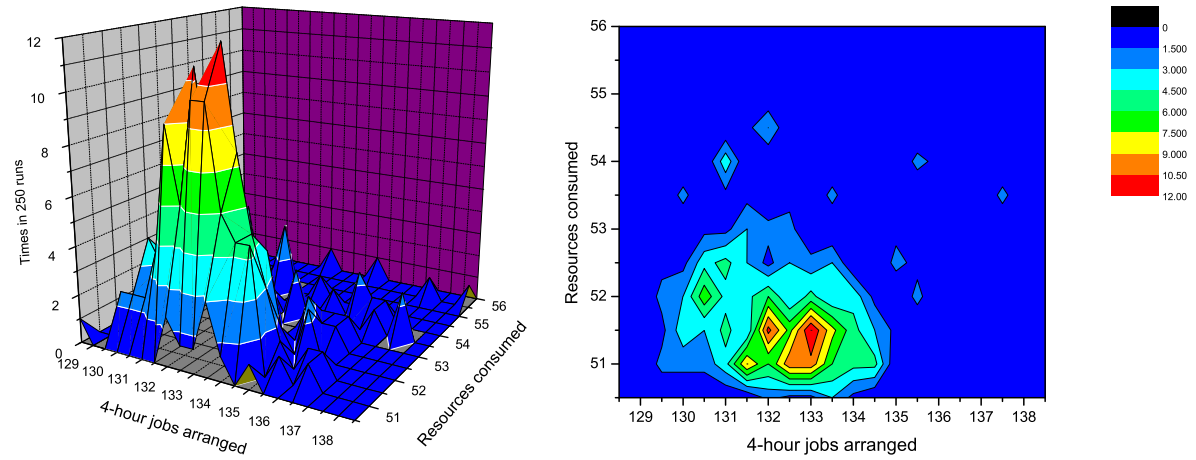
从图 6–1、图 6–2 和图 6–3 中，我们可以观察到一个非常有趣的现象：

- 当 BT *Member* Agent 数量从 1 增加到 4 的过程中，解是不断变好的：平均 BT 耗费和 4-小时工作安排量都在减少。从 3D 分布图中我们可以看到，峰值点不断往左下方移动；在两个边缘分布图中，资源耗费分布曲线和 4-小时工作分配量分布曲线都是不断往左方移动的；同时，分布的形状也在逐渐变得越来越“好”[1]；
- 相反的，当 BT *Member* Agent 数量从 4 增加到 12 的过程中，解是不断变坏的：平均 BT 耗费和 4-小时工作安排量都在增加。从 3D 分布图中我们可以看到，峰值点不断往右上方移动；在两个边缘分布图中，资源耗费分布曲线和 4-小时工作分配量分布曲线都是不断往右方移动的；同时，分布的形状也在逐渐变得越来越“坏”。

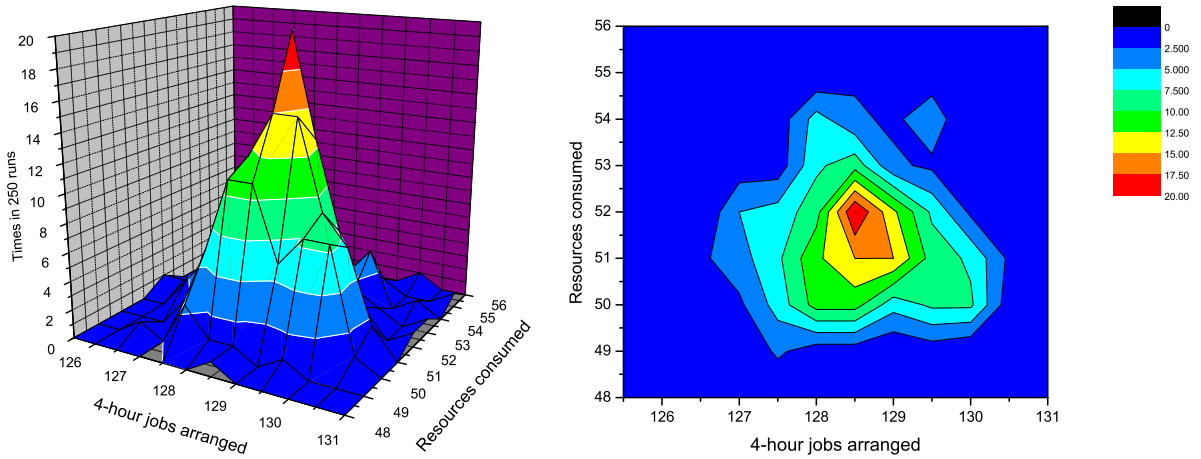
这种现象是怎么发生的呢？接下来的理论分析也许能提供部分原因。

很明显 DSAFO 中的多个 *Member* Agent 是用来将全局解空间分割为局部划分的，局部划分的数量就等于 *Member* Agent 的数量。然后每个局部划分优先使用单个 *Member* Agent 进行启发式求解。尽管分治法（Divide-and-Conquer）这样一类经典的人工智能方法的本质本身也是某些启发式的基础，DSAFO 拥有一些特殊的机制来跳出分治法的局

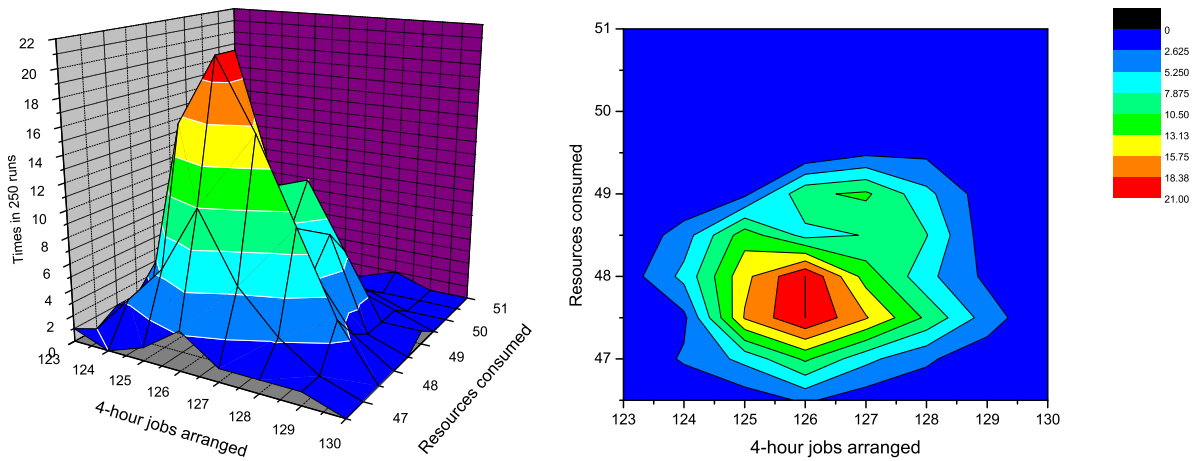
[1] 当然肉眼观测并不是强有力的证据，不过好的形状可以通过同正态分布的拟合程度来测定，在 3D 分布图和 2D 曲线图中都是这样。



1 BT *Member* Agent

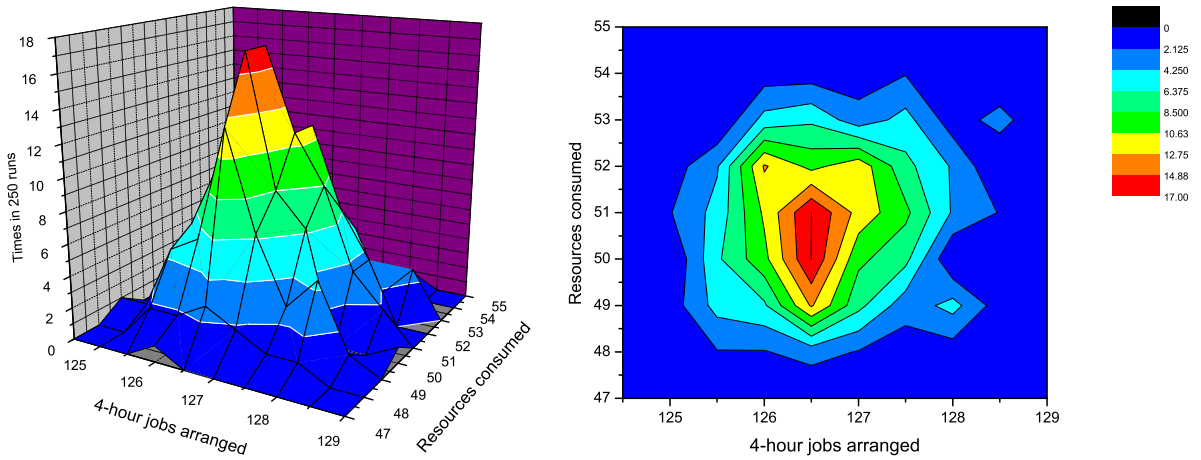


2 BT *Member* Agents

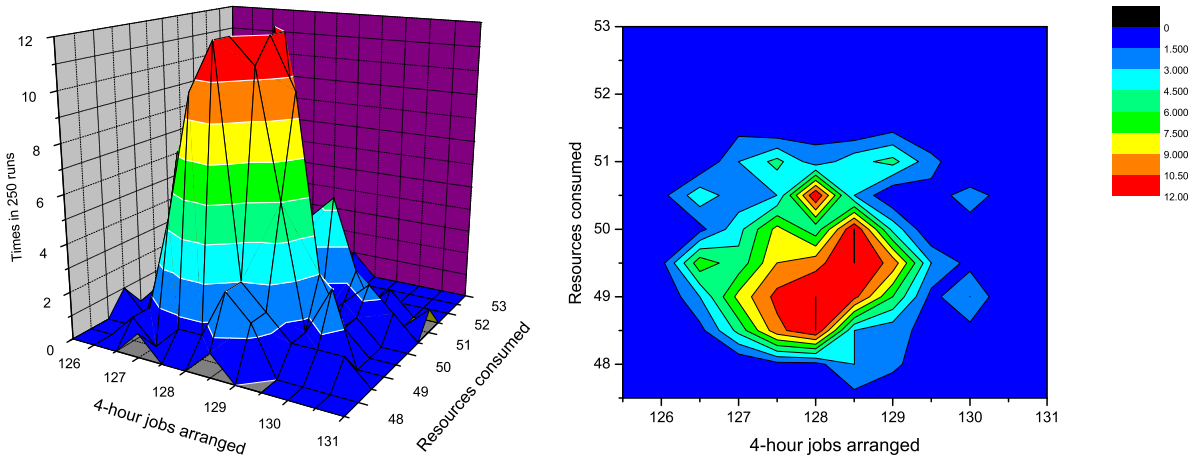


4 BT *Member* Agents

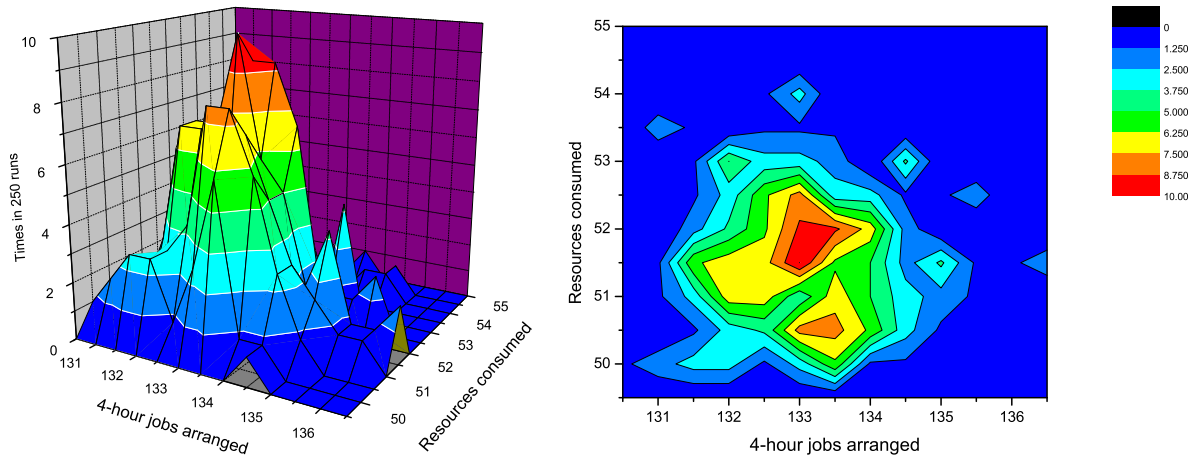
图 6–1: *Member* Agent 数量对 BT 解分布的影响



6 BT *Member* Agents



8 BT *Member* Agents



12 BT *Member* Agents

图 6–2: *Member* Agent 数量对 BT 解分布的影响（续）

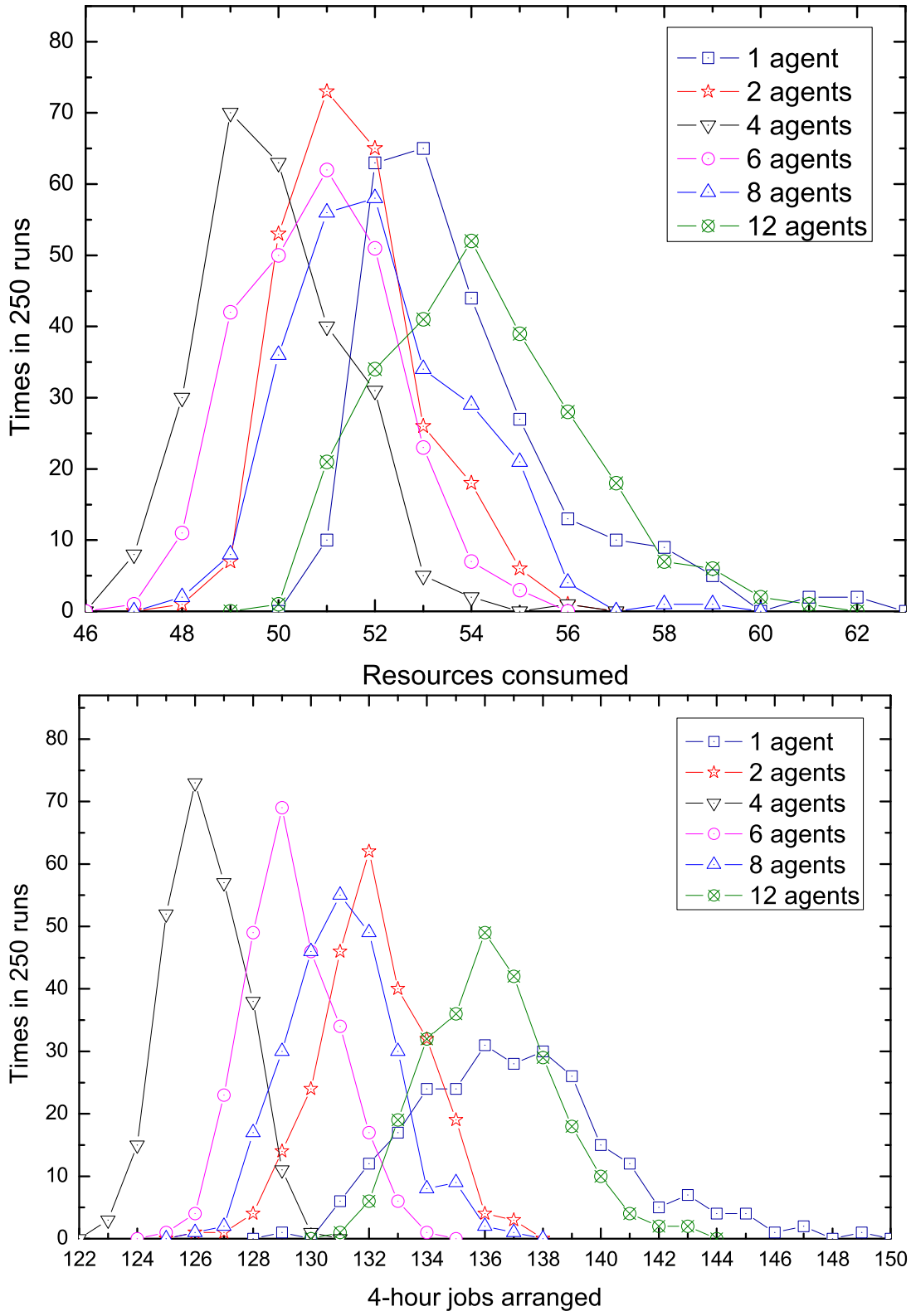


图 6–3: *Member* Agent 数量对 BT 解的边缘分布的影响

部极小值：通过划分（Agent）之间的协作可以得到一些优化。这样的协作为 DSAFO 提供了比传统的分治法启发式更好的解。

为什么太少的 *Member* Agent 不能得出好解呢？这应该是因为协作优化策略并没有发挥出足够的效果，或者没有效果。如果仅仅只有一个 BT *Member* Agent，就不可能发生协作，因此仅仅只有一个 BT *Member* Agent 的 DSAFO 肯定只有很弱的跳出局部极小的解性能的优化能力。

为什么太多的 *Member* Agent 不能得出好解呢？这可能是因为太多的 Agent 将全局解空间划分成太多的碎片，以至于在这些碎片中的启发式太缺乏法全局观点，最后连协作也不能够进行非常好的优化。

6.2.2 Blockfactor

Blockfactor 是另外一个具有影响力的参数。我们将其他参数设置为：Agentnum=1, Delayfactor=1/6, Syncycle=5 和 Reqcycle = 1/6，将能够得到不同 Blockfactor 下的解的 3D 图和高度图的分布对比，如图 6–4所示；同时也能得到对应的边缘分布图，如图 6–5所示。

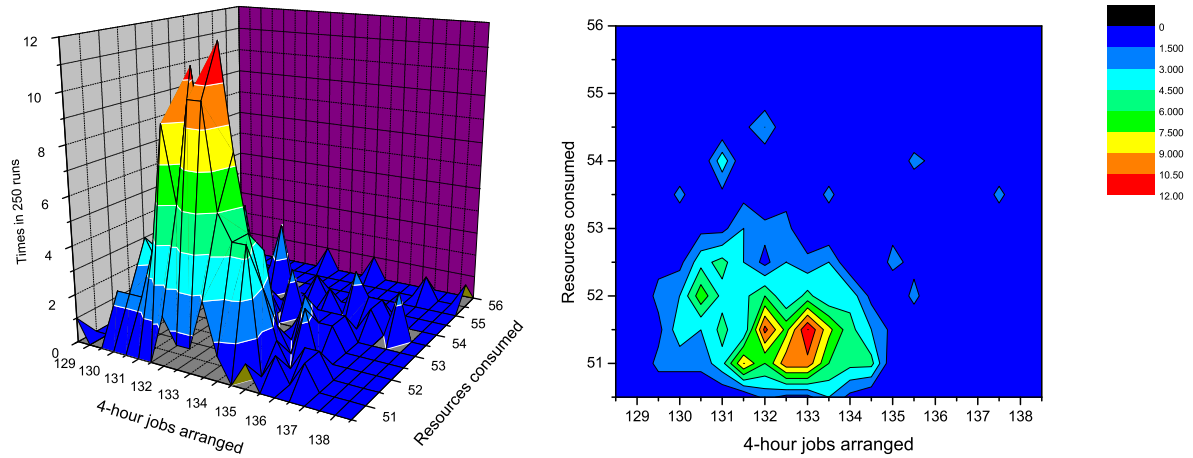
从图 6–4 和图 6–5中，我们可以看到：

- 在 Blockfactor 减少时，3D 图和高度图的峰值略微向下移动，也就是说，资源耗费基本保持不变，而 4-小时工作分配量略有减少。
- 从 3D 形状和折线形状我们可以看到，当 Blockfactor 变小，解的分布变得更加集中，同时算法看起来更接近一个稳定算法。

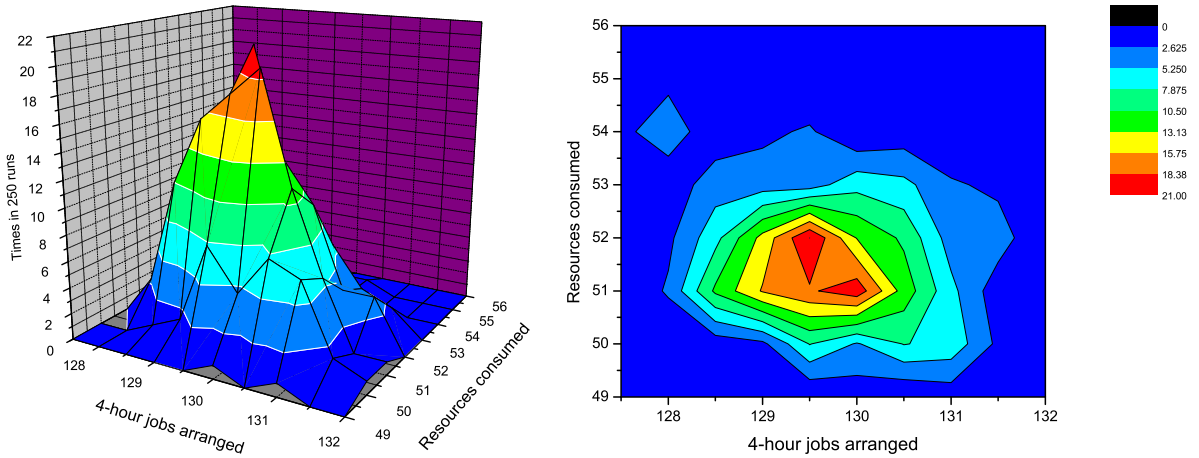
在第 5.3 节对 DSAFO 的描述中，当一个 Agent 没有事情可做的时候，它就将自己阻塞（block）Blockfactor × Heartbeat 长的时间。在这个时间内，该 Agent 可以被消息唤醒。也就是说，Blockfactor 代表着在 Agent 活动周期之间的阻塞时间的长短。

如果仅仅只有一个 BT *Member* Agent，就没有伙伴来尝试唤醒它。那么通常它就可能保持阻塞状态直到时间结束，除非 *Coordinator*, *Blackboard* 或 *ResourceAdmin* 向它发送消息。因此当 Blockfactor 小的时候，阻塞时间就短，DSAFO 算法就看起来更像集中式的启发式。当 Blockfactor 小的时候，阻塞时间就长，算法看起来就不像集中式的启发式。

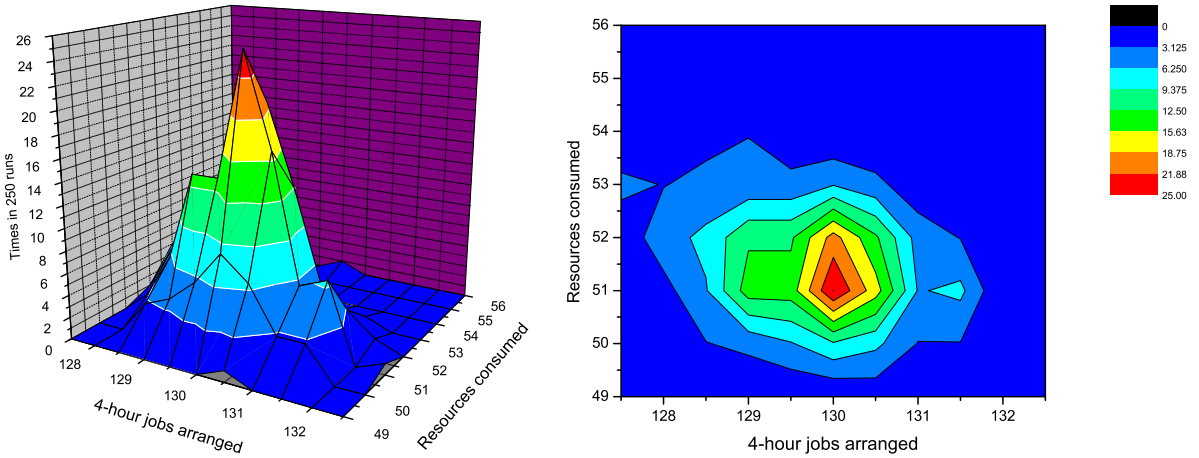
为什么带有很小的 Blockfactor（比如 1/36）的一个 *Member* Agent 仍然输出不稳定解？原因应该是作为 JADE ACL 物理媒介的网络通信本身。特别地，我们将单 *Member* Agent 嵌入 *Blackboard* Agent，也就是说，没有任何不稳定网络通信干扰因素。新的嵌



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



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



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

图 6–4: Blockfactor 对 BT 解分布的影响

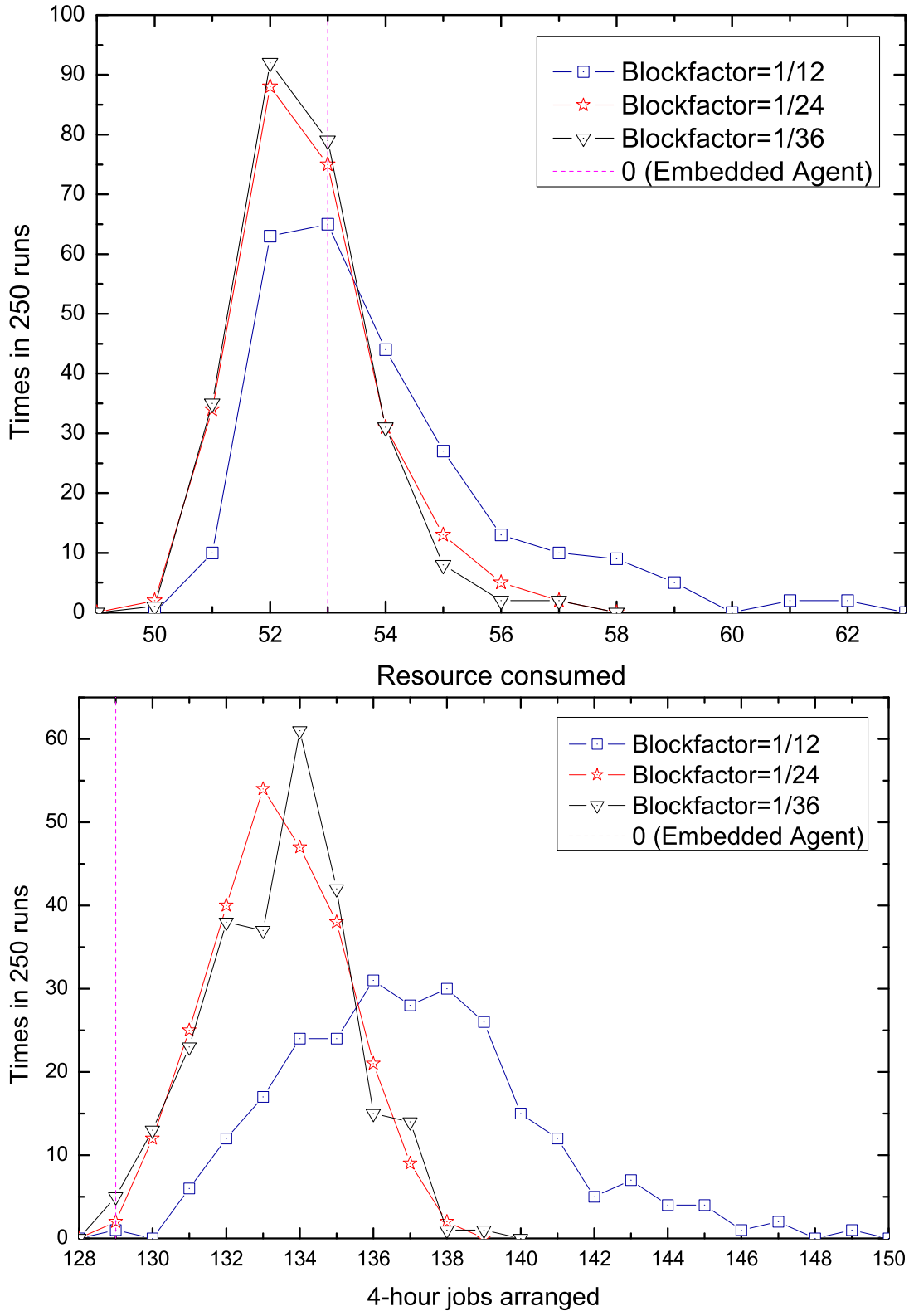


图 6–5: Blockfactor 对 BT 解的边缘分布的影响

入的算法回归成了一个确定性算法，如图 6–5 所示（非常接近第 7.1.2 小节的 EDD* 算法）。

6.2.3 Delayfactor

Delayfactor 表示的是 *Member* Agent 在承诺一个成功的作业后应该延迟多少时间，这并不是一个非常有影响力的参数。我们设定 Agentnum=4, Blockfactor=1/12, Syncycle=5 和 Reqcycle = 1/2，然后可以得到不同 Delayfactor 下的解的 3D 图和高度图的分布对比，如图 6–6所示；同时也能得到对应的边缘分布图，如图 6–7所示。

对 4-小时工作并没有显著的变化，不过 3D 图和高度图的峰值点有轻微的上下移动，也就意味着资源消耗受到了影响。但是图 6–7 中的资源耗费曲线并没有显示出这样的微弱现象：当 Delayfactor 变化的时候，资源耗费非常接近。

这个时间延迟用来阻止一个群体中的某个 *Member* Agent 独占地不停请求、尝试和承诺作业，也就是说，来保证负载平衡。如果没有额外的延迟，同时任务到达比较均匀，最后成功地承诺作业的 *Member* Agent 在自己的资源没有消耗殆尽的情况下，肯定也会是取得下一个同类作业的那个 Agent。因此，Delayfactor 被用来保证负载平衡，而不是直接改进算法性能。

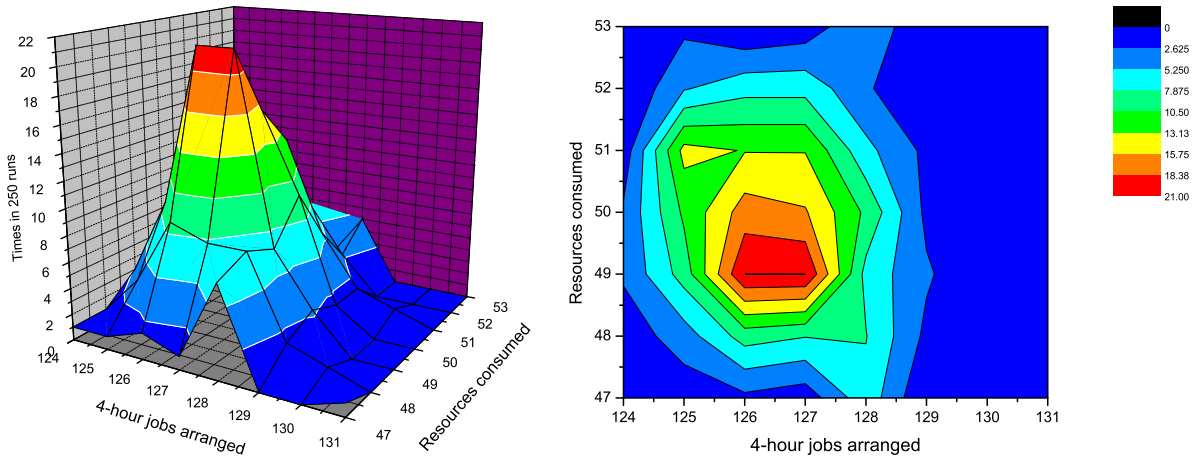
6.2.4 Syncycle

Syncycle 表示的是 *Member* 和 *Coordinator* 应该在每次远程资源利用率同步之间等待多少时间。这也不是一个有很大影响力的参数。我们设定 Agentnum=4, Blockfactor=1/12, Delayfactor=1/6 和 Reqcycle = 1/2，然后可以得到不同 Syncycle 下的解的 3D 图和高度图的分布对比，如图 6–8所示；同时也能得到对应的边缘分布图，如图 6–9所示。

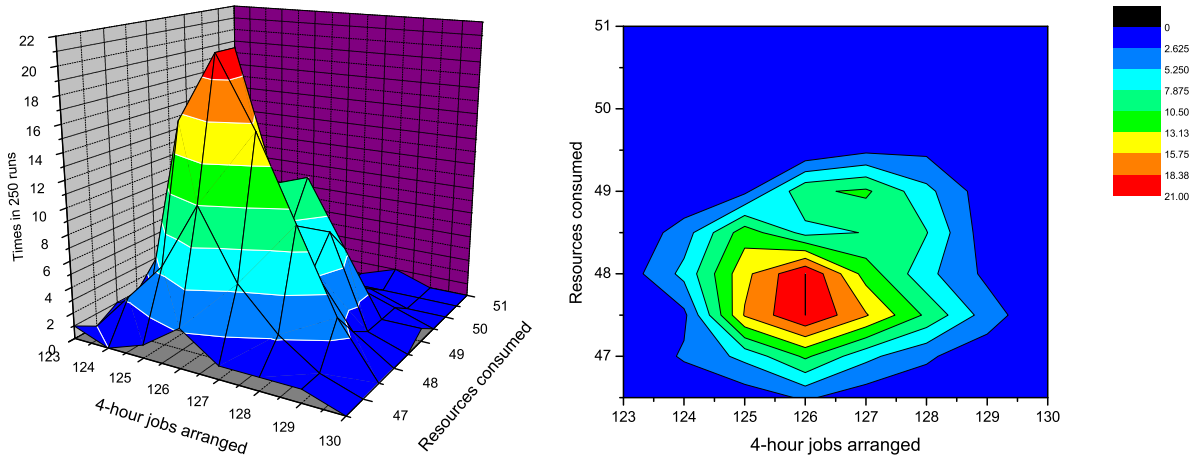
从这些图中，4-小时工作看起来并没有被变化的 Syncycle 所显著影响；同时资源耗费也没有被显著影响。

这个同步过程被设计来交换 *Member* Agent 之间的资源利用率，并且根据 S. Abdallah, N. Darwish 和 O. Hegazy (2002) 所说，*Coordinator* Agent 的设计也能辅助这个过程。因此当 Syncycle 比较大的时候，算法中的 *Member* 和 *Coordinator* Agent 所保存的信息就有比较大的可能陈旧和过期。因此使用一些陈旧的信息来指导协作肯定不是一个好点子，因此，理论上比较小的 Syncycle 应该能让算法运行得更好。

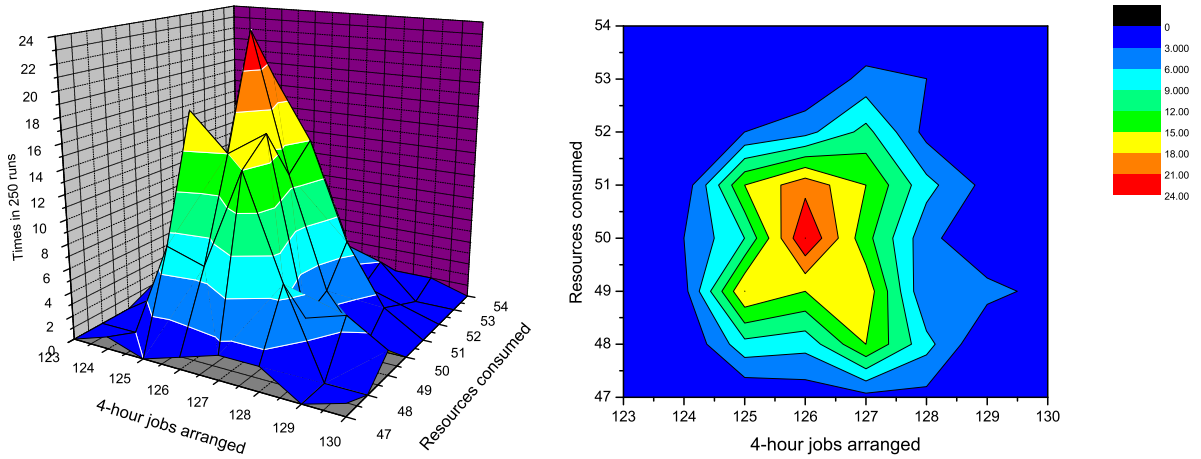
但是，在实际实验中，Agent 之间的协作比起所有的作业分配步骤来说是相当少的，而且有时候基于过期的信息协作的算法动态运行也许会得到不错的结果。不论如何，这



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



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



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

图 6–6: Delayfactor 对 BT 解分布的影响

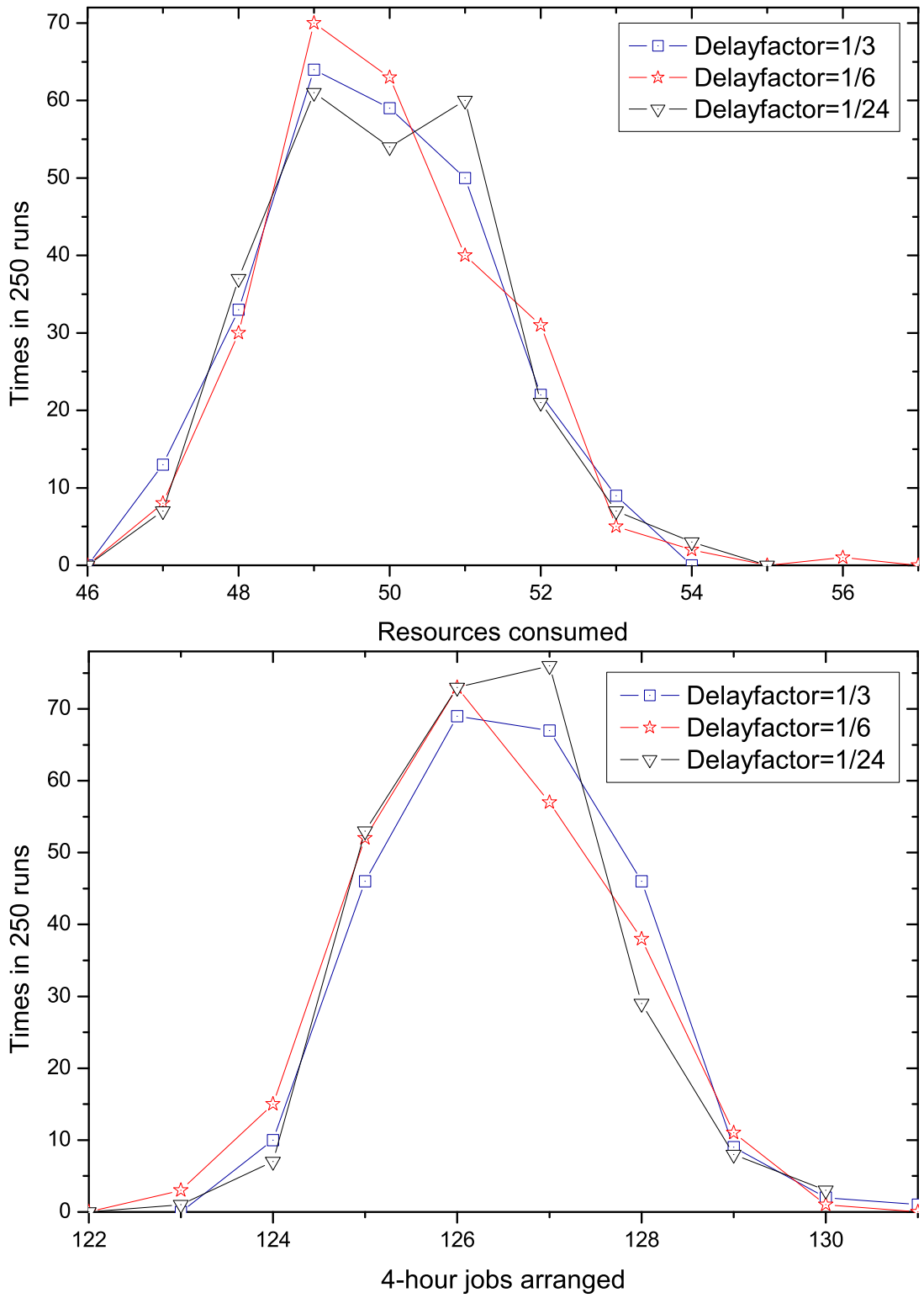
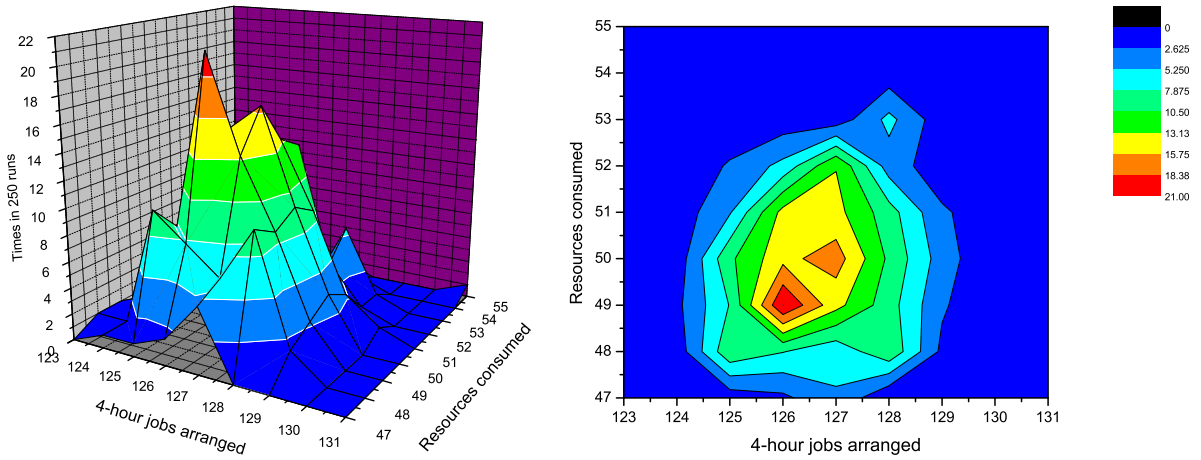
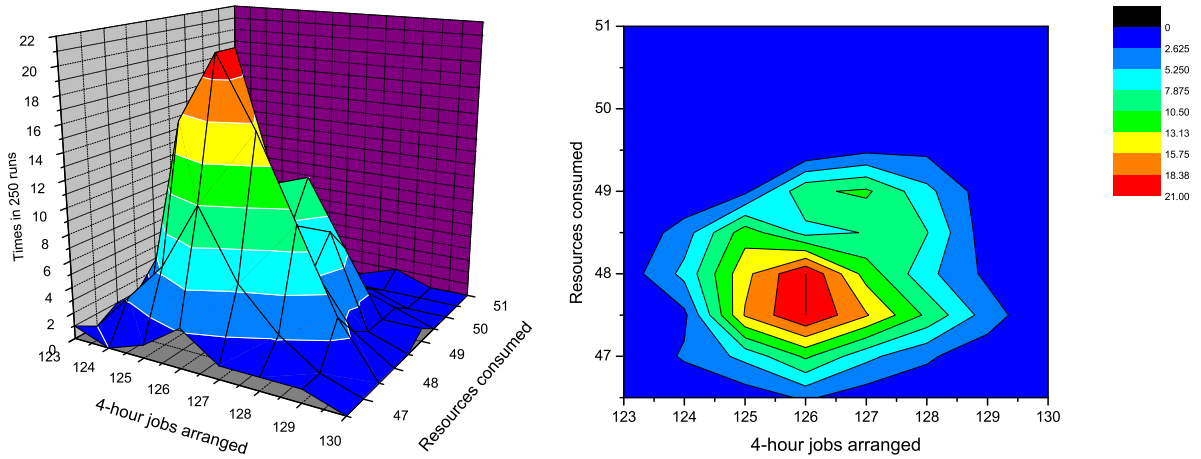


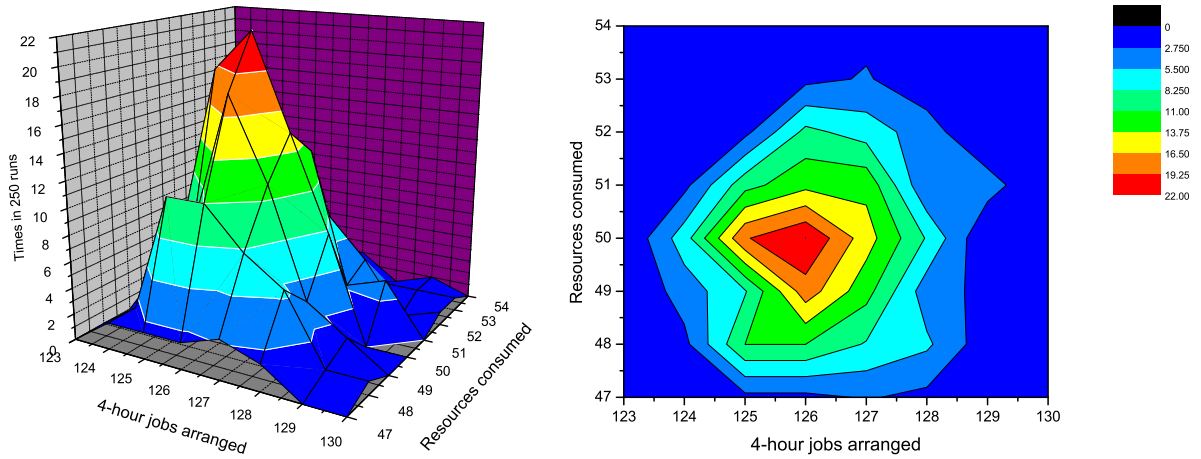
图 6–7: Delayfactor 对 BT 解的边缘分布的影响



4 BT *Member* agents with Syncycle=3



4 BT *Member* agents with Syncycle=5



4 BT *Member* agents with Syncycle=15

图 6–8: Syncycle 对 BT 解分布的影响

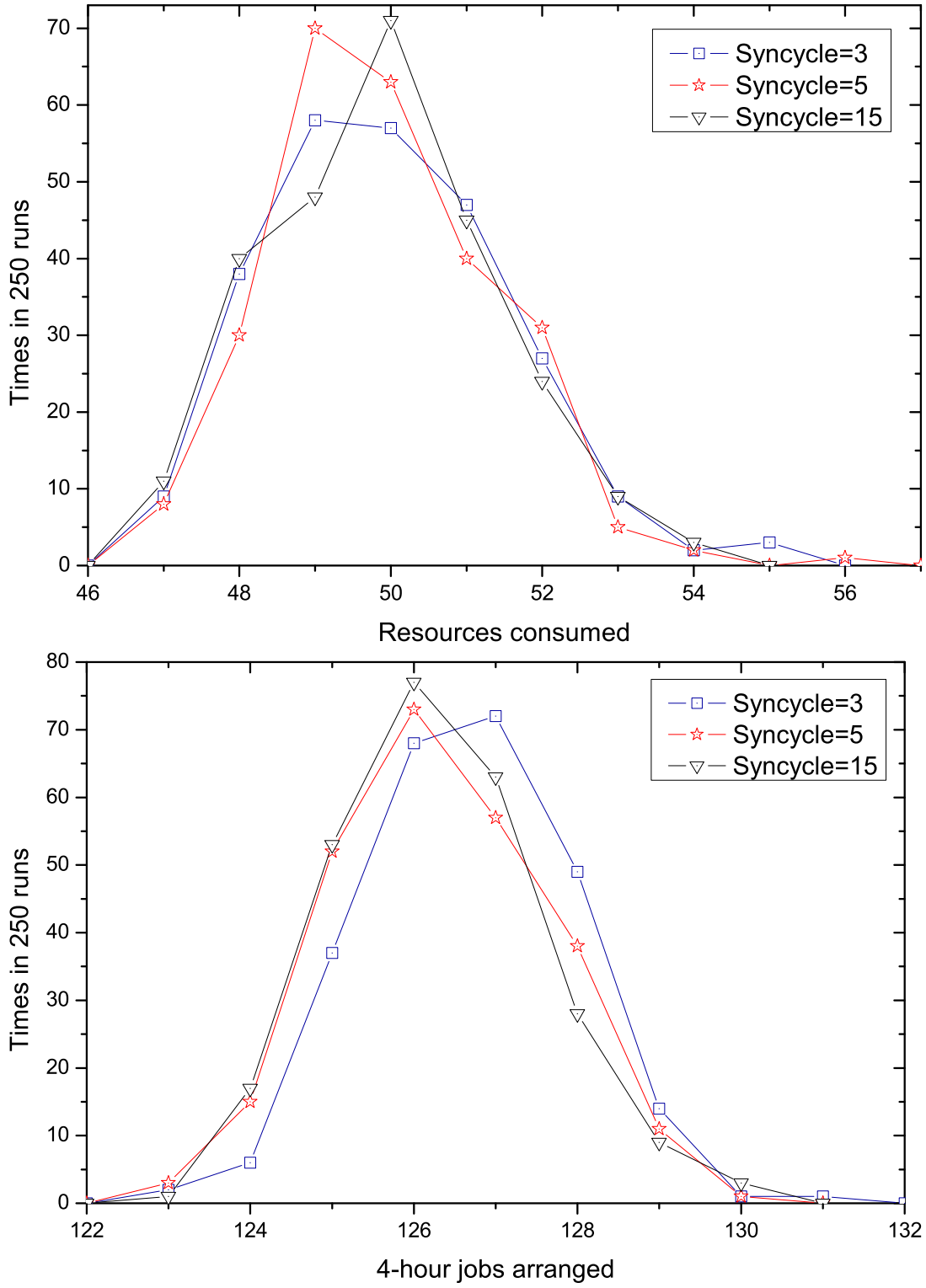


图 6–9: Syncycle 对 BT 解的边缘分布的影响

个参数对算法的实际影响比理论预期小很多。

6.3 实验小结

在本章中，进行了一系列实验。实验结果说明了 DSAFO 能够成功地利用动态 Agent 跳出局部极小点。从另一种视点来看，多 Agent 将全局资源分配分解为了多个划分，每个划分代表着一个 Agent 的观点。为了尽可能最大化资源使用效率，局部启发式之后的动态资源借用扮演了一个关键的角色。每个划分都是动态的，而且每个划分中的每个资源都得到了接近最大的利用，因此 DSAFO 能够成功地避免局部极小。另外，利用动态 Agent 来解决动态困难问题也是非常自然的；在改变某些参数的情况下，算法的不确定性也可以得到调整。

很多其它困难问题也总是被困于局部极小，而且也已经成功地被其他智能算法所优化，比如蚁群算法 ACO 、粒子群算法 PSO 等。DSAFO 和其它多 Agent 算法可能代表的是另一种帮助这些问题跳出局部极小的应用方向。

第七章　算法对比

本章分别介绍了面向飞机地面作业调度的 DSAFO 算法的三个对比算法：$\mathcal{MM}$AS，EDD* 和 ERT*；并给出了这四种算法在实际数据驱动下的飞机地面作业调度中的性能优化对比。

7.1　三种对比算法

本节介绍了面向飞机地面作业调度的三种对比算法，它们是 $\mathcal{MM}$AS，EDD* 和 ERT*。$\mathcal{MM}$AS 是一个衍生于蚂蚁优化算法的、静态地（并非动态地）优化飞机地面作业调度问题的智能优化算法。EDD* 和 ERT* 分别是实现到我们的动态调度环境 run-and-schedule 中的两个经典的传统的启发式算法。

7.1.1　$\mathcal{MM}$AS

蚂蚁算法（AS，Ant System）是由 M. Dorigo, V. Maniezzo 和 A. Colorni (1991) 所提出[142, 143]。蚂蚁算法是受到自然界蚂蚁社会的正反馈——每个蚂蚁都在自己的路径上释放信息素来帮助其他蚂蚁选择正确的路径——的启发而发明出来的。因此通常蚂蚁算法被用来进行空间问题的计算，比如旅行商问题（TSP，Travel Salesman Problem）。

形式化地说，一个面向旅行商问题的蚂蚁算法包含 N_{ant} 个有限的计算主体，叫做“蚂蚁”，以及一个完全图 <V, E> 和整个 E 上的痕迹浓度 T（模拟的信息素）。在图 <V, E> 中，V 是 m 个顶点的集合，E 是这些定点之间所有边的集合。每个蚂蚁从一个边爬行到另一个边，每个蚂蚁每次运行都形成一个 Hamilton 回路[1]。为了确保生成有效的 Hamilton 回路，每个蚂蚁都具有一个禁忌列表来标记已经访问过的顶点。另外，$\tau_{ij}(t)$ 指的是在边 <i,j> 上的访问痕迹浓度，

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

[1] Hamilton 回路是包含图中所有顶点一次且仅一次的闭环。

其中，ρ 是一个痕迹残留因子，而 $(1-\rho)$ 表示的是每个周期痕迹的挥发比率。

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

其中，$\Delta\tau_{\mathrm{ij}}^{k}$ 是边 <i,j> 上的由第 k 个蚂蚁在 t 到 t+1 时间上在每单位长度上的痕迹释放量（现实世界中就是蚂蚁的信息素），这个量由下式所决定：

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

“能见度”η_{ij} 被定义为 $1/d_{\mathrm{ij}}$，其中 d_{ij} 是顶点 i 到 j 之间的距离。最后，第 k 个蚂蚁从 i 移动到 j 点的概率是

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

其中，α 和 β 是控制痕迹权重和能见度权重的两个算法参数。

蚂蚁算法在很多领域都取得了成功[143, 144]。$\mathcal{MM}$AS（最大-最小蚂蚁算法，$\mathcal{MAX}$-$\mathcal{MIN}$ ant system）是蚂蚁算法的一个发展，它给痕迹浓度限定了一个上限和一个下限[145, 146]。从 $\mathcal{MM}$AS 刚开始被提出的时候，它就被应用到了 JSSP 问题中。

我们将 $\mathcal{MM}$AS 实现在了 BT 安排问题上来进行算法比较。我们将 252 个中转航班中的 1,008 BT 相关作业转换为 1,008 个点。两个作业 r_i 和 r_j 之间的距离表示为：

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

这样的 $\mathcal{MM}$AS 和面向 TSP 问题的稍有不同，因为 BT 相关的四种作业需要按顺序执行。因此我们将每个“UB–UC–LC–LB”作业组按顺序排列在一起，因此，1,008 个点看起来就如表 7.1.1 所示：

表 7–1: 将 BT 相关作业映射到 TSP 点上

作业	UB_1	UC_1	LC_1	LB_1	…	UB_k	UC_k	LC_k	LB_k	…
顶点	1	2	3	4	…	$4k+1$	$4k+2$	$4k+3$	$4k+4$	…

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

同时我们为每个点 $i\in$V 定义了一个布尔型变量 $ready_i$。很明显地，最初只有 UB 作业（$ready_{4k+1}, k\in\{0,\ldots,251\}$）才被标记为“$true$”。然后我们在蚂蚁决定选择边

$< i, j >$ 时采用一套递归验证规则来检验应该执行的作业：

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

因为 $validate(i,j)$ 是一个递归函数，所以在顺序“UB–UC–LC–LB”里的当前的就绪作业一定会在未就绪作业之前被分配。同时，选择了点 j 后，$ready_{j+1}$ 就会变成 $true$，除非 j 表示的是一个 LB 作业。结果，产生的整体服务作业调度方案保证是可执行的。

$\mathcal{MM}$AS 算法的其它的参数给定如下：$\alpha = 1.5, \beta = 2, \rho = 0.05, \tau_{\text{init}} = 1, \tau_{\max} = 100, \tau_{\min} = 0.01$, $\text{N}_{\text{ant}} = 200$, $\text{NC}_{\max} = 150$。并且对于第 i 个蚂蚁的成功的执行周期 R，对应的 Hamilton 回路上的所有边都会得到一定的正反馈：

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

来使得算法朝着较少资源耗费和人力耗费的方向发展。

7.1.2 EDD* 和 ERT*

这里的 EDD* 是简单地将传统启发式“最早完工时间优先”（EDD，Earliest Due Date first）应用到 run-and-schedule 动态调度环境中。类似地，ERT* 就是将“最早就绪时间优先”（ERT，Earliest Ready Time）应用到 run-and-schedule 环境中。EDD 和 ERT 的大意如图 7–1 所示。

7.2 优化对比

然后将 DSAFO 算法（参数：$\text{agentNumber}_{\text{BT}}$=4, Blockfactor=1/12, Delayfactor=1/6, Syncycle=5）、EDD*、ERT* 和 $\mathcal{MM}$AS 在第 6 章的 252 中转航班的数据下进行了实验测试。我们又一次选择了 BT 相关作业来测试这几个算法的优化性能。表 7–2 中显示的就是这几个算法在 BT 耗费和 4-小时 BT 工作安排上的优化对比结果。另外，算法的时间耗费和 CPU 占用也被纳入考虑范围。每组数据中的最有数值均采用黑体突出显示。

从表 7–2 可以看出，DSAFO 和 $\mathcal{MM}$AS 在资源优化上都有不错的结果，并且 DSAFO 在 BT 工作安排上要比 $\mathcal{MM}$AS 好。另外，DSAFO 花费的时间也不多（几分钟），占用的 CPU 也很低。实际上 DSAFO 把大部分的时间用来确保准确的消息传输。相反的，$\mathcal{MM}$AS 耗费很多时间，并且 CPU 占用率接近 100%。

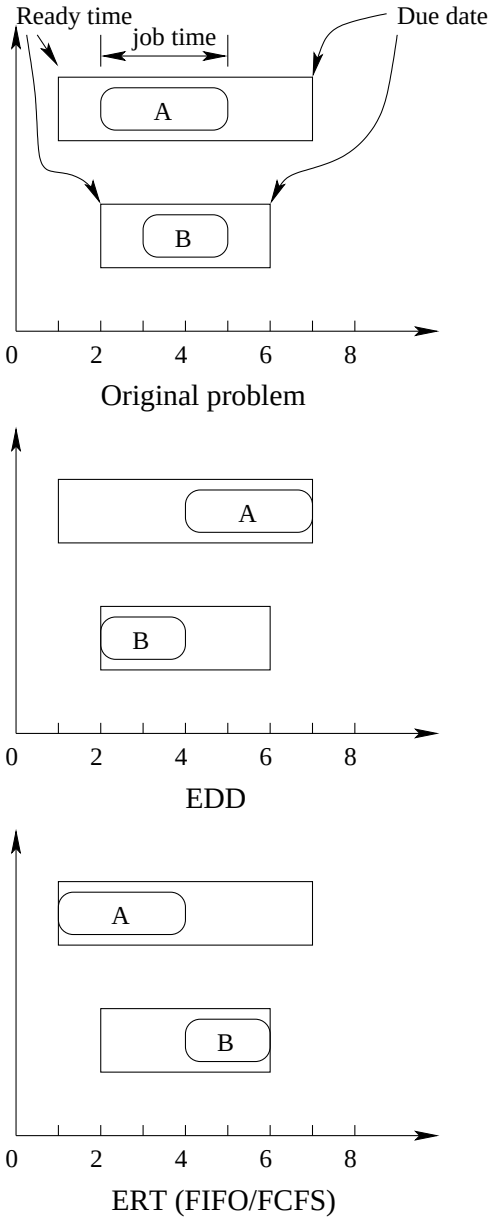


图 7–1: EDD 和 ERT 的简要示意图

表 7–2: 算法优化性能对比

算法	时间	CPU	资源			4-小时任务		
			MIN	MAX	AVG	MIN	MAX	AVG
DSAFO	≈144 秒	<1%	**47**	56	**49.9**	**123**	130	**126.3**
$\mathcal{MM}$AS	≈ 12 小时	≈100%	**47**	—	—	141	—	—
EDD*	**≈43 秒**	<1%	53	53	53	129	**129**	129
ERT*	**≈43 秒**	<1%	52	**52**	52	130	130	130

第八章　结论和未来研究方向

8.1　结论

飞机地面作业调度问题是一个典型的困难调度问题。尽管该问题是$\mathcal{NP}$-困难问题，但是它对航空公司和机场的正常运营和收益以及旅客满意度来说十分重要。

本文提出的 DSAFO，是一个基于动态分布式模型和动态调度环境 run-and-scheduling 的新型的多 Agent 约束满足优化算法。该算法的时间复杂度比较优秀：介于平方和三次方之间，因此能够处理大规模的飞机地面作业调度问题。实验表明 DSAFO 的解不是很稳定，而且解受到几个算法参数的影响，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。

分析表明带有全局协作的分布式启发式和动态多 Agent 视点在实际困难问题的优化中是有效的，同时也有很大的发展前景。

8.2　未来研究方向

未来工作之一是要开发针对飞机地面作业调度问题的、基于 DSAFO 算法的、便于工程师和调度人员使用的调度软件。一个比较基础的调度开发环境 AGSAP 已经被 W. Fan, G. C. Zhang 和 F. Xue 提出[147]，该环境是引用 DSAFO 算法进行了通用的飞机地面作业调度。

另外，考虑到航空公司和机场的现实管理需要，研究在不确定航班起降情况下的调度优化算法也是十分有价值的[148]，比如不确定飞机地面作业调度满足和最优化问题。因此，如何能够在不确定航班情况（即不确定飞机地面作业调度满足和最优化问题）下生成鲁棒的（稳定的）预测调度方案，应该是未来的一个研究要点。

致 谢

首先，我要感谢我的导师樊玮教授。他不但是一位良师，更是一位益友，从他那里我所学到的不仅仅是学术技能、道德修养和社会适应能力。他对我的学习和成长所做出的种种贡献难以只言片语加以概括，我的感激也同样难以完全表达。我也要感谢祝世兴教授和顾兆军博士，感谢那些愉快的合作和他们对我的慈父般的启迪和教育。

在这短短的几年里，我很有幸能结识很多伙伴和朋友：张广才、王兴云、王元崑、张劼、郭启铭、范敬德以及软件技术研究中心的研究团队的其他人员。我从他们那里得到了各个方面的巨大支持。我还要感谢厦门航空有限公司的黄小荣经理、黄翠薇女士和潘海莹女士、中国民航总局安全技术研究中心的刘云雷先生、AMECO（北京飞机机务维修公司）的康力平高级工程师，同他们的合作十分愉快。还要特别感谢过去的三个学年里大力支持我的生活、研究和论文工作的朋友们，特别是张晓燕和曹琳。

我也要感谢蓝天基金和国家自然基金委员会（NSFC）的基金支持（60472123，2005 年 1 月– 2005 年 12 月）和中国民航大学博士启动基金的资助（QD13X04，2004 年 1 月– 2005 年 12 月）对本论文不可估量的推动和促进。

最后，我要从心底里感谢我的父母，他们给了我生命，教育我长大，无条件地给我支持并鼓励我追求我的梦想。同时也要感谢我的弟弟，我们曾经共同度过了一个值得珍惜的金色童年。

附录 A：DSAFO 算法中的 Agent 通信语法

DSAFO 算法中所有的消息通信都是基于 FIPA ACL 的，并是一种字符串运载的通信。每条消息都包含一个信道标识，一个逗号“,”和消息本体，即

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

当某消息 $Message$ 被发送时，算法会根据它的 $ChannelMark$ 选取合适的 FIPA ACL 协议。这样做可以有效地避免消息冲突。从$ChannelMark$ 到 FIPA ACLs 的映射定义如下：

表 A–1: DSAFO 算法中的通信语法和 FIPA ACL 协议

$ChannelMark$	协议	$Content$
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

参考文献

[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. 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 noval 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. Gulledge 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 multiagent 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, Febrary 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/OC00003A.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: offline 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 $\mathcal{MAX}$–$\mathcal{MIN}$ 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 $\mathcal{MAX}$–$\mathcal{MIN}$ 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.

攻读硕士学位期间所发表的论文

2006 年 8 月 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))

2006 年 8 月 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**))

2006 年 8 月 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**)

2004 年 10 月 **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**))

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

简　历

简介

薛帆生于 1982 年 9 月，陕西白水人，他是薛信杰和景巧侠的大儿子。于 2000 年在陕西华县咸林中学完成了他的学习后，进入了天津市中国民用航空学院航空港工程系继续学习，并于 2004 年 7 月取得自动化专业的工学学士学位。2004 年 9 月，他进入中国民航大学计算机科学与技术学院继续攻读硕士学位。他的研究兴趣主要包括智能信息处理、多 Agent 系统和企业级应用智能。

硕士课程

在攻读硕士学位期间，他修习了 17 门理论课程和4门实践课程，共计取得 37 学分（获得该硕士学位需要 33 学分），这其中包括学位课21学分、选修课 12 学分和实践环节4学分。理论课程的平均成绩是 86.0 分，其中专业课平均成绩为 90.0 分。为了夯实专业基础，他还选修了 6 门计算机专业的本科生专业主干课程。

部分项目和研究经验

2005–2006 算法开发人员，“基于多Agent协作及信息融合的飞机地面作业运行控制优化算法”，排名第九，
由国家自然基金资助（60472123），与中国科学院和中国国际航空公司合作；

2004–2006 程序开发人员，“智能决策支持系统开发”，
由中国民航大学博士启动基金资助（QD13X04）；

2004–2005 程序开发人员，“厦门航空收益管理系统（二期）”，排名第三，
由厦门航空资助（043107011R），天津市科技攻关项目资助，与厦门航空合作；

2003–2006 系统设计开发人，“中国民航机队信息系统”，
由中国民航总局资助，与中国民航总局安全技术研究中心合作。