Optimize Cooperative Agents with Organization in Distributed Scheduling System

Wei Fan and Fan Xue

Software Technology Research Center, Civil Aviation University of China,
Tianjin 300300, China P.R.
fanxinhui@eyou.com

Abstract. DSAFO (Dynamic Scheduling Agents with Federation Organization) is a novel multi-agent constraint satisfaction algorithm for AGSS problem (a NP-hard scheduling problem). This paper improves on DSAFO by employing a resource requisition strategy, and models this parallel multi-agent algorithm in polyadic π-calculus. The time complexity of the improved DSAFO is $O(n^3) + O(n^2) \times t_{\text{trans}}$. Experiments show improved DSAFO performs well in AGSS consumptions optimization of resources and man-days. Though it is unstable, improved DSAFO makes good probability to find better solutions than classical heuristics and its distributed and parallel agents viewpoint is potential to deal with distributed dynamic troubles in real applications.

1 Introduction

Airport ground service is a service process from flight landing to takeoff, including gate assignment, baggage handling, catering, fueling, cleaning, etc. AGSS (Airport ground service scheduling) is to schedule many kinds of dynamic ground resources (baggage trucks, fuel trucks, etc.), to fulfill all constrained service subtasks of flights timely to meet their arrival and departure deadlines [1].

AGSS problem is a NP-hard problem. Moreover, it could be viewed as either a JSSP (Job-Shop Scheduling Problem) [2] $J_m|r_j, prmp|\Sigma w_j U_j$ [3] or a Dis-CSP [4], because of the ways that real subtasks and resources are organized. In real AGSS, most resources are ample except some prepared for accidents. Consequently the scheduling target $\Sigma w_j U_j = 0$ is not difficult to achieve, however how many resources and man-days consumed are important to airports and airlines. The algorithm in this paper optimizes the resources and man-days consumptions.

DSAFO (Dynamic Scheduling Agents with Federation Organization) [1] is a novel multi-agent algorithm for Dis-CSP (Distributed Constraint Satisfaction Problem), especially for AGSS problem under high constraint. DSAFO employs blackboard mechanism, federation organization, meta-level guided resource borrowing and domain knowledge guided plan backtrack. The principle of DSAFO is coordinating distributed resources with cooperative agents.

The remainder of the paper is structured as follows: Chap. 2 represents the improved DSAFO in form of polyadic π-calculus to precisely describe the parallel essence of improved DSAFO. Chap. 3 analyzes the time complexity. Experiment results appear in Chap. 4 and a brief conclusion is given in Chap. 5.

D.-S. Huang, K. Li, and G.W. Irwin (Eds.): ICIC 2006, LNAI 4114, pp. 502–509, 2006.

2 Improved DSAFO

2.1 Polyadic π-Calculus

The polyadic π-calculus developed by Milner [5] is a very powerful tool to model processes in parallel systems such as multi-agent systems and mobile systems. The most primitive entity in polyadic π-calculus is a *name*. Names, infinitely many, are $x, y, \ldots \in \chi$; they have no structure. The other kind of entity, a *process*, is built from names by syntax

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

In this paper, we model DSAFO in π-calculus and usually abbreviate name sequences in π-calculus to vector names, e.g. a subtask tuple structure may be abbreviated to "$\overrightarrow{task}$". A name in vector name could be accessed using "$\triangleright$", e.g. the LFT (Latest Finish Time) of a task is $task \triangleright LFT$.

2.2 DSAFO: An Overview

DSAFO is based on several parameters such as $Heartbeat$. And it implements a unique Blackboard, a unique ResAdmin, some Coordinators and more Members.

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

Channels among agents are organized as Fig. 1.

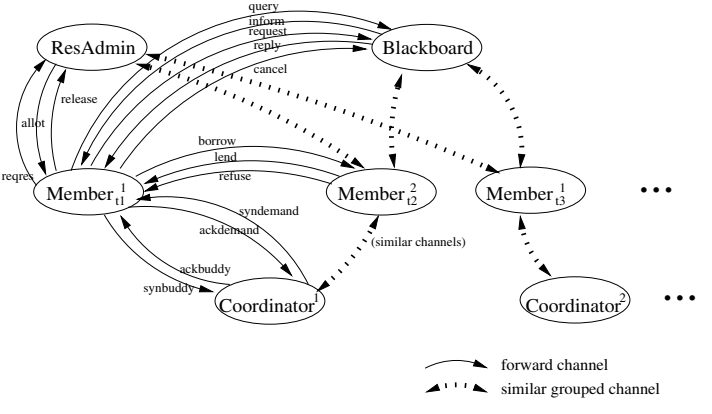

Fig. 1. channel organization of DSAFO

2.3 Blackboard Agent

The unique agent BLACKBOARD perceives information of flights coming in half an hour, decomposes them into subtasks and assigns subtasks with plans from member agents.

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

BLACKBOARD has four sorts of behaviors responding messages from members. For example, $RespQuery_t^a$ receives signals from channel $Query_t^a$ and then responses with the top of ready subtasks and the most delayed subtask.

$$\begin{aligned}RespQuery_t^a \equiv & !query_t^a.(\nu\, c)(\overline{readytask_t}\langle c\rangle|c(\overrightarrow{tsk}).[\overrightarrow{tsk} \neq nil]\overline{inform_t^a}\langle\overrightarrow{tsk}\rangle) \\ & .(\nu\, uc)(\overline{delaytask_t}\langle uc\rangle|uc(\overrightarrow{utsk}).[\overrightarrow{utsk} \neq nil]\overline{inform_t^a}\langle\overrightarrow{utsk}\rangle) \\ RespReq_t^a \equiv & !request_t^a(\overrightarrow{plan}).(\nu\, ch)(\overline{getplan}\langle ch, plan \triangleright fno, plan \triangleright tsk\rangle \\ & |ch(\overrightarrow{myplan}).[\overrightarrow{myplan} = nil](\overline{assign}\langle\overrightarrow{plan}\rangle.\overline{reply_t^a}\langle syn, \overrightarrow{plan}\rangle)) \\ RespCancel_t^a \equiv & !cancel_t^a(\overrightarrow{plan}).(\nu\, ch)(\overline{getplan}\langle ch, plan \triangleright fno, plan \triangleright tsk\rangle \\ & |ch(\overrightarrow{myplan}).[\overrightarrow{myplan} = \overrightarrow{plan}]\overline{free}\langle plan \triangleright fno, plan \triangleright tsk\rangle)\end{aligned}$$

2.4 ResAdmin

Agent RESADMIN is a resource administrator. When received resources request, RESADMIN would search free resources and allot one if possible. After half a man-day (4-hour work), the resource would be free again. Additionally, one resource serves no more than one and a half man-days.

$$\begin{aligned}\text{RESADMIN} \stackrel{\text{def}}{=} & (reslist_r, historylist_r)(RaFunc|Respreq_t^a|Resprelease_t^a) \\ & (r \in Resources, t \in Schtasks, a \in Agents_t)\end{aligned}$$

Behaviors $Respreq_t^a$ and $Resprelease_t^a$ do resources allocation and recovery.

$$\begin{aligned}Respreq_t^a \equiv & !resreq_t^a(begintm).(\nu\, ch)(\overline{gettopfreeres_t}\langle ch, begintm\rangle|ch(\overrightarrow{res}) \\ & .[\overrightarrow{res} \neq nil]\overline{allot_t^a}\langle res \triangleright name, begintm\rangle) \\ Resprelease_t^a \equiv & !release_t^a(resname).(\nu\, ch)(\overline{getresfromhash}\langle ch, resname\rangle \\ & |ch(\overrightarrow{res}).[\overrightarrow{res} \neq nil]\overline{loghistory}\langle\overrightarrow{res}\rangle.\overline{releaseres}\langle\overrightarrow{res}\rangle)\end{aligned}$$

2.5 Member

A member agent MEMBER_t^a is in charge of subtask t. It always has a desire to handle subtasks. After MEMBER_t^a received available subtask goals from

BLACKBOARD, it intends to make suitable plans and request for them. Every member agent always has an open hand when others need help.

$$\begin{aligned}\text{MEMBER}_t^a \stackrel{\text{def}}{=} & (Resource_t^a, Remoteres_t^a)(MbFunc \\ & |ActQry_t^a|MkPlan_t^a|CallBd_t^a|Ackres_t^a|TryLend_t^a|AckLend_t^a) \\ & (t \in Schtasks, a \in Agents_t)\end{aligned}$$

Cyclic behavior $ActQuery_t^a$ is infinite (but finite in practical scheduling). It sends subtask requests to drive the DSAFO and releases expired resources. The behaviors $MakePlan_t^a$ perceives subtask information and try to make plans for them. If the subtask cannot be locally scheduled, $CallBd_t^a$, $TryLend_t^a$ and $AckLend_t^a$ cooperate to perform resource borrowing. After all resources borrowing failed, it would send resource request to RESADMIN, and behavior $Ackres_t^a$ would receive the coming resource.

$$\begin{aligned}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{tsk}).(\nu\, ch)(\overline{makenullplan_t}\langle ch,\ \overrightarrow{tsk}\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]\overline{synbuddy_t^a}(\overrightarrow{plan}))) \\ CallBd_t^a \equiv & !(ackbuddy_t^a(\overrightarrow{uplan}, \overrightarrow{lst})|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{reqres}\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) \\ 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}, res).([res \neq null]\overline{assign_t^a}\langle\overrightarrow{plan}\rangle.\overline{succ}\langle\overrightarrow{plan}\rangle \\ & + [res = null]\overline{fail}\langle\overrightarrow{uplan}, \overrightarrow{lst}\rangle)) \\ AckLend_t^a \equiv & !lend_t^a(\overrightarrow{plan}).\overline{assignremote}\langle\overrightarrow{plan}\rangle.\overline{request_t^a}\langle\overrightarrow{plan}\rangle\end{aligned}$$

2.6 Coordinator

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

$$\begin{aligned}\text{CORDINATOR}_r \stackrel{\text{def}}{=} & (Metainfo_r, Syncycle)(CooFunc|Synch_r|Store_t^a|RespBd_t^a) \\ & (r \in Resources, t \in TskinRes_r, a \in Agents_t)\end{aligned}$$

$Synch_r$ and $Store_t^a$ cyclically collect meta-level information of member agents which belong to it. Behavior $RespBd_t^a$ tells a member buddy list sorted by meta-level information.

$$\begin{aligned} Synch_r &\equiv (\overline{synall_r}.\overline{block_r}\langle Syncycle \times Heartbeat\rangle.)^{+\infty} \\ Store_t^a &\equiv !ackdemand_t^a(dm).(\nu\, ch)(\overline{setval}\langle ch, dm\rangle | ch(demands_t^a) \\ &\quad .\overline{removelist}\langle demands_t^a\rangle.\overline{insertsort}\langle demands_t^a\rangle) \\ RespBd_t^a &\equiv !synbuddy_t^a.(\nu\, ch)(\overline{genbuddylist_t}\langle ch\rangle | ch(\overrightarrow{blst}).\overline{ackbuddy_t^a}\langle\overrightarrow{blst}\rangle)) \end{aligned}$$

3 Complexity

First of all, we abbreviate the number of flights $\|Flights\|$ to "n" as well as $24 \times 60 \times Heartbeat$ to "WholeDay".

3.1 Preparation

For any subtask t, there are n jobs to do at most. And there must be a minimal completion time $\mathrm{ct}_t^{\min}$ for t. The resource expectation res_t^*

$$\mathrm{O}(n) = n \times \mathrm{ct}_t^{\min}/\mathrm{WholeDay} \leq res_t^* \leq n = \mathrm{O}(n)$$

Hence, $res_t^* = \mathrm{O}(n)$ $(t \in Schtasks)$

For each resource r, there is a constant set $Tskinres_r$. By summing it up one by one, we have expectation of each resource $r \in Resources$ is $res_r^* = \mathrm{O}(n)$. In fact, resources to allocate usually approximates the expectations $\mathrm{O}(n)$, so we assume the number of each resource $r \in Resources$ is $res_r = \mathrm{O}(n)$.

Because the agents number affects the complexity, we assume there are O(n) agents to compute the upper bound of complexity.

3.2 Time Complexity

Time complexity of improved DSAFO is mainly from two parts: resource borrowing and demand synchronization. The rest processes are less complex.

Resource Borrowing. In the worst case, there are $\mathrm{O}(n)$ searches and $\mathrm{O}(n)$ borrowing message transmissions to fulfill every subtask. All $\mathrm{O}(n)$ subtasks cost $t_{\mathrm{borrow}} = \mathrm{O}(n^2) \times t_{\mathrm{search}} + \mathrm{O}(n^2) \times t_{\mathrm{trans}} = \mathrm{O}(n^3) + \mathrm{O}(n^2) \times t_{\mathrm{trans}}$.

Demand Synchronization. Demand synchronization is a cyclic action. Each member has only one coordinator in federation organization, therefore, each synchronization transmits $\mathrm{O}(n)$ messages and do $\mathrm{O}(n \times n)$ sorts. The synchronization costs

$$t_{\mathrm{syn}} = \big(\mathrm{O}(n) \times t_{\mathrm{trans}} + \mathrm{O}(n^2)\big) \times \frac{\mathrm{WholeDay}}{Heartbeat \times Syncycle} = \mathrm{O}(n^2) + \mathrm{O}(n) \times t_{\mathrm{trans}}$$

To sum up, the upper bound of complexity $t_{\mathrm{DSAFO}} = \mathrm{O}(n^3) + \mathrm{O}(n^2) \times t_{\mathrm{trans}}$.

4 Experiment Result

We implemented DSAFO in JADE [6], a multi-agent development environment, and implemented the channels on FIPA ACL [7].

In classical JSSP, EDD (Earliest Due Date first) and FCFS (First Come Fist Serve, also well known as ERT, Earliest Ready Time) are two powerful heuristic algorithms for minimal uncompleted jobs scheduling [3]. We tested these algorithms with a 282 transfer flights problem under normal constraint.

As a multi-agent algorithm, DSAFO is an unstable algorithm. Consequently we gave spontaneous 250 runs to get the distribution of the solutions. Figure 2 shows the baggage truck solution distribution for trucks and man-days consumed by DSAFO with 8 baggage truck agents.

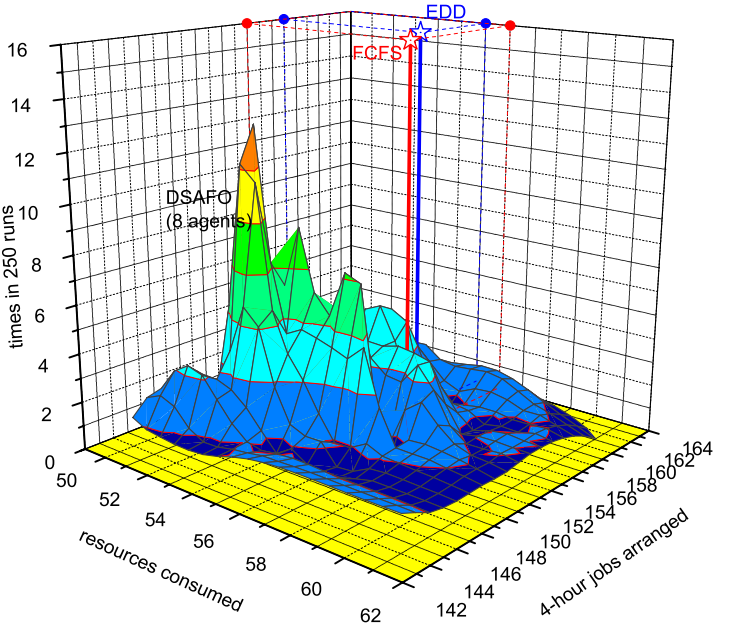

Fig. 2. Distribution of DSAFO (8 agents) solutions for baggage trucks

Furthermore, we tested DSAFO in different quantities of MEMBER agents. The results were accumulated in Fig. 3 and 4. From the marginal distributions in Fig. 3, we can see that 76.8% resource solutions of DSAFO with 8 agents are not worse (59.6% better) than EDD and FCFS, the probabilities are 54% and 32% with 12 agents and 16 agents. But the best solutions of the three are nearly the same. And so is in Fig. 4. It means when the number of agents increases, probabilities of good solutions decrease, but the best solution seems stable.

Figure 5 shows the simultaneous serving resources of solutions done by different optimization algorithms.

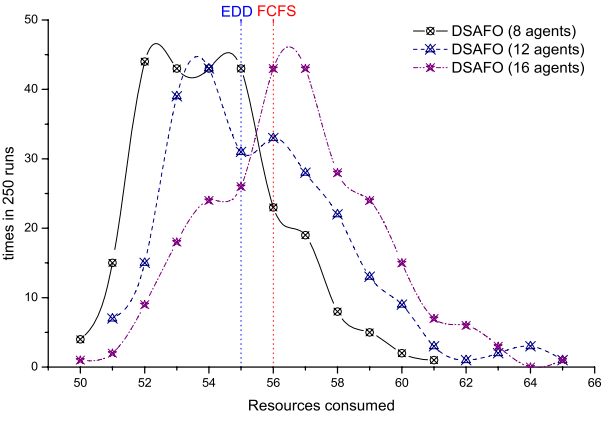

Fig. 3. Marginal distribution of baggage trucks consumed

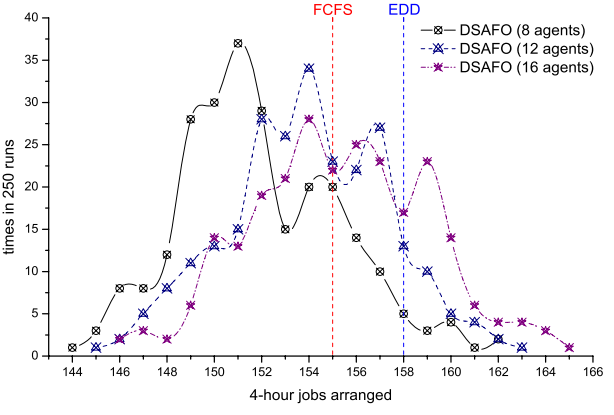

Fig. 4. Marginal distribution of 4-hour baggage truck works

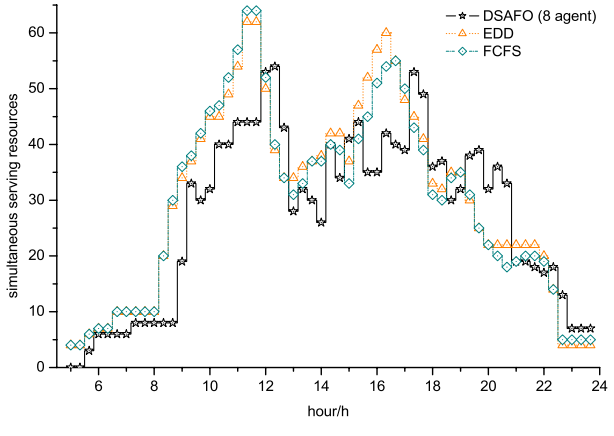

Fig. 5. Simultaneously serving baggage trucks by different optimization algorithms

5 Conclusion

AGSS problem is a typical dynamic distributed scheduling problem. To optimize AGSS consumptions of resources and man-days is difficult, but it is important to economy. DSAFO is a novel multi-agent constraint satisfaction algorithm for AGSS problems.

We improved DSAFO for optimizing consumptions of resources and man-days by incorporating a resource allocation strategy. Experiments show that improved DSAFO performs well in optimization. And when the number of agents increase, probabilities of good solutions would decrease but the best solution would be stable.

Though improved DSAFO is unstable, it makes good probability to find better solutions than classical heuristics, and its construction of distributed and parallel agents is potential to deal with distributed dynamic troubles in real applications. Furthermore, DSAFO may be allied to common dynamic distributed scheduling problems for its low complexity and sound effect.

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

References

1. Xing, J., Fan, W., Ji, L.: Design of Airport Ground Service System Based on Multi-agent. Journal of Civil Aviation University of China, 24 (2006) (in Chinese).
2. Modi, P. J.: Distributed Constraint Optimization for Multiagent Systems. Ph.D. Dissertation. University of Southern California, USA (2003)
3. Tang, H., Zhao, L.: Introduction to Scheduling Theory. Chapter 2. Chinese Acdemic press, Beijing (2002) (in Chinese).
4. Yokoo, M.: Distributed Constraint Satisfaction: Foundation of Cooperation of Multi-agent System. Springer Verlag, Heidelberg, Berlin (2001)
5. Milner, R.: The Polyadic π-calculus: A Tutorial. Tech. Report ECS-LFCS-91-180. University of Edinburgh, UK (1991)
6. Bellifemine, Poggi, F., Rimassa, A., Jade, G.: A FIPA2000 Compliant Agent Development Environment. In Proceedings of The Fifth International Conference on Autonomous Agents (AGENT01), Montreal, Canada, ACM Press, New York (2001) 216-217.
7. FIPA: FIPA ACL Message Structure Specification. Document No. 00061, Geneva, FIPA, http://www.fipa.org/specs/fipa00061/SC00061G.pdf (2002)