Scheduling in a Multi-Agent Environment

Milan Schmotzer, Jan Paralic, Julius Csonto,
Dept. of Cybernetics and AI, Technical University of Kosice,
TU Kosice, Letna 9, 042 00 Kosice, SLOVAKIA
tel.n.: (++421 95) 6253574, fax: (++421 95) 6330115,
e-mail: schmotze@neuron-ai.tuke.sk

Abstract

A new scheduling agent for existing CIM multi-agent system is being currently developed at the Technical University of Kosice. The basic idea is to create an agent based on Scheduling Program developed at the Technical University of Kosice. The Scheduling Program was developed in the ECL^{i}PS^e programming language using its constraint logic programming capabilities. It is used for solving Job-Shop problems. This paper gives the reader some basic information about prepared software.

Keywords
Constraint logic programming, Scheduling, Agent

1 INTRODUCTION

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35390-6_58

L. M. Camarinha-Matos et al. (eds.), *Intelligent Systems for Manufacturing*

The basic idea of CIM (Computer Integrated Manufacturing) is to integrate all company activities into a unified management structure exploring a large scale hierarchy of computers. Considering all possible activities (planning, scheduling, product design, manufacturing, etc.) and the fact that some of these tasks are running in parallel on geographically distributed sites, the integration is a nontrivial problem.

DISCIM - geographically DIStributed decision making system for CIM (Marik,1994). DISCIM was developed in a tight co-operation of CTU Prague and FAW Linz. This system can be viewed as a group of co-operating autonomous processes which enables processing of geographically distributed data and knowledge.

This paper should inform about developing a new scheduling agent for DISCIM. This agent is based on the Scheduling Program developed at the Technical university in Kosice using the constraint logic programming (CLP) technology provided by the ECL^{i}PS^e language.

CLP is a programming paradigm which extends logic programming represented e.g. by Prolog in two different ways. On the one hand, incorporating of new semantic objects gives the possibility to use more effective algorithms and, on the other hand, the search process is speeded up using constraint propagation in an interleaved fashion. CLP potential lies in its power to tackle difficult combinatorial problems, such as job scheduling in factories, resource allocation, placement problems, production planning and many more.

In addition, we have used some features of the ECL^{i}PS^e language which enabled us to add some new specialised constraints there as well as to control the search process using some specialised heuristics. The most important improvements related to our scheduling program are described in this paper.

2 DISCIM

Distributed systems offer the potential of significant rising the memory capacity as well as the problem–solving speed. DISCIM 1.0 is a distributed, multi-agent system which should be used for CIM (Computer Integrated Manufacturing) purposes. Its implementation has been aimed at using on SUN-SPARC UNIX-based workstations interconnected via the INTERNET network. It was created as an open multi–agent environment suitable for efficient creating complex expert or decision support systems and therefore it is easily extensible.

Currently, there are two different types of system components (agents). One is the ***graphical interface component*** (GIC) and the other is the ***expert system component***. During one session the system involves one graphical interface component and several expert system components whose number is defined by the user. Each expert system component of the system is an independent expert system. One of all expert system components plays the most important role during the

session. It is called *Main Expert Component* (MEC). Inference engines of all other expert systems called *Remote Expert Components* (REC) are started only when their knowledge is requested. Inference engine of the Main Expert Component controls the session. It solves goal assertions from Main Expert Component knowledge base using information from the user and Remote Expert Components. Remote Expert Component can be contacted not only by the Main Expert Component but by other Remote Expert Components too.

3 AGENTS

For the purposes of this paper, let us define that an agent is an autonomous entity which can actively communicate with it's environment, for example with some other agents, and to process obtained informations. An agent can be a part of one or more multi-agent systems.

The following principles have been used to design the DISCIM system:

- Agents are independent, autonomous entities communicating in the peer-to-peer way among themselves. The asynchronous message passing in UNIX/INTERNET environment is used to perform this communication.
- Each agent consists of a functional body (usually a stand-alone program) and a wrapper (which is responsible for involvement of the agent into the community of agents).
- There is no central part of memory or control in the agents' community. The corresponding pieces of the control strategy are "owned" by the individual agents.
- There is a library of standard classes/messages (written in the OO-oriented programming language Eiffel) available for the agent designer.
- User interface has been designed as a separate agent in the community (Hazdra,1995). It is the only agent which can initialise the activity of the agents' community. It is a mobile agent - it can be run on each of the computers in the considered network.

The basic types of agents currently tested are:

- an expert system agent,
- a database agent,
- a qualitative simulation agent,
- a user interface agent, and
- the neural network agent

To add an agent to a multi-agent system it is necessary to use a communication platform. Such platform must enable peer-to-peer communication connections among the agents. Each agent is an autonomous unit which communicates with other agents in the system by message passing. The agent must contain a set of communication operations for sending and receiving messages as well as operations

determining how to react on received messages (e.g., changing the internal state of an agent, forwarding messages, creating new agents, moving an agent, etc.).

The problem of attaching different agents to a system must be solved by creating different interfaces corresponding to the agents. Each agent is then attached to the whole system via the interface and communication modules.

If there exists a program developed as stand-alone, DISCIM specific methods allow to create a new agent from it. Therefore in many cases it is possible to add a new agent to the DISCIM environment very quickly. The communication model is supported by the Parallel Virtual Machine (PVM) system which is a suitable superstructure over the SUN RPC programming. The system implementation is based on the object-oriented design and is programmed in Eiffel.

4 CONSTRAINT LOGIC PROGRAMMING

CLP is a programming paradigm which extends logic programming (LP) in two different ways. Firstly, there were attempts to incorporate semantic objects inside LP, which would enable us to use more effective algorithms to manipulate with them. These attempts resulted in the so called CLP(X) scheme (incremental linear solving), which was theoretically formulated in (Jaffar,1987) and consequently implemented on the real arithmetic domain in the CLP(R) system (Jaffar,1992).

Secondly, there were attempts to overcome the well known performance problems of the *generate-and-test* strategy of the LP. The first representative of this approach was the finite domain part of the CHIP system (Dincbas,1988). The key aspect here is a tight integration between a deterministic process, constraint evaluation, and a non-deterministic process, search (domain technology).

Two different approaches resulted in two different technologies used inside existing CLP systems. In both cases, not only expressiveness of the resulted CLP language but also the efficiency is highly improved due to the underlying *constraint solving techniques.*

CHIP was a first and most innovative prototype implementing both CLP technologies mentioned above. It was initially developed in an industrial laboratory, the ECRC (European Computer-Industry Research Centre) based in Munich (Germany). This event triggered much of what has followed in this area. Since the beginning of nineties a couple of commercial CLP tools have been released. Besides CHIP V5 there are currently DECISIONPOWER, Prolog III, SNI Prolog, ILOG or BNR Prolog each of them being already successfully used for serious industrial applications. There are many more CLP systems (CLP(R), ECL^{i}PS^e, CAL, clp(FD) to name a few) used more less for research purposes (some of them going to be commercialised).

CLP potential lies in its power to tackle difficult combinatorial problems, such as job scheduling in factories, resource allocation, placement problems, production planning and many more. The applicability of the CLP approach has already been

proven being used by large corporations, including manufacturers as Dassault Aviation (scheduling and planning) or Renault (planning and car sequencing).

We used ECL^{i}PS^e Prolog system (developed at ECRC as one of the successors of the first CHIP system).

Typically, the CLP systems work in three steps as follows:

1. The variables which represent solved problem are defined and their finite domains are specified.
2. The constraints are imposed on these variables and propagated through the constraint net.
3. If stating and propagation of the constraints alone did not solve the problem, it is needed to try to assign values to the variables. Every instantiation immediately wakes all constraints associated to the variable and changes are propagated to the other variables. In such a way the search space is usually reduced very quickly by reducing the domains of other variables or directly instantiating them.

5 SCHEDULING AGENT FOR DISCIM

The functional body of the Scheduling Agent (SA) for DISCIM will be based on a stand-alone program, the Scheduling Program which was developed by Milan Schmotzer as his MSc dissertation (Schmotzer,1997) at the Technical University of Kosice in 1997. The Scheduling Program (SP) is implemented in ECL^{i}PS^e. It can solve Job-Shop scheduling problems (French,1987). In this section the main features of this program are presented.

SP uses a good deal of typically AI algorithms - constraint logic programming, branch and bound, heuristic algorithms, Carlier and Pinson's algorithm and task intervals (Caseau,1995). For the SP new optimal search algorithms (*logarithmic min_max* and *logarithmic minimize*) as well as algorithms to deterministically and heuristically find lower and upper bounds were developed. Some new specialised constraints were developed as well.

There exist two versions of SP - TSP (text version) and GSP (graphic version). Graphic interface for the GSP was created in the Tcl/Tk script language. The GSP uses a special kind of pseudomultitasking. For SA, there will be used a modified text version of SP.

5.1 The General Job-Shop scheduling problem

The terminology of scheduling theory arose in the processing and manufacturing industries. Its structure fits many scheduling problems arising in business, computing, government, the social services and industry.

Suppose that there exist n **jobs** $\{J_1, J_2, \ldots, J_n\}$. Each job is divided into m tasks from which every task has to be processed on different machine. So there are m **machines** $\{M_1, M_2, \ldots, M_m\}$ to consider. Technological constraints demand

that there is an ordering between the tasks in particular jobs. Each job has it's own ordering of tasks. No two tasks from the same job can be processed simultaneously. The processing times (durations of tasks' execution) are independent on the schedule.

5.2 The logarithmic min_max algorithm

If the programmer needs to find an optimal schedule, he (or she) can use the branch and bound algorithm. There are two standard optimisation predicates in ECL^{i}PS^e - `min_max` and `minimize`. We have developed some new optimisation algorithms. The most important one - the *logarithmic min_max* algorithm - will be explained here (we published the results of our benchmark tests comparing the new algorithms with the traditional ones in (Paralic,1997)).

In every iteration it divides the interval to half size owing to it needs about $\log_2(Max - Min)$ proves of solution existence. The `min_max` algorithm needs $Max - Min + 1$ proves of solution existence (in the worst case). The proving of the solution existence is done by setting a constraint which says that the cost of the solution cannot be greater than *Limit* and by finding a solution limited by that constraint (condition).

In the following the *logarithmic min_max* algorithm will be presented. Variable called *AllowedDeviation* denotes the required distance between the cost of the found schedule and the cost of a optimal one. The first step of the algorithm uses the "fix" operation. This mathematical operation cuts the decimal part of a real number.

1. Let $Limit = Max - fix\left(\frac{Max - Min}{2}\right)$.
2. If $Limit = Max$, then let $Limit = Max - 1$.
3. Can it be proven that there exists a solution with a cost smaller than *Limit* ? If yes, then let $Max = it's\_cost$[1], if not, let $Min = Limit + 1$.
4. Let $Deviation = 100 * \frac{Max - Min}{Min}$.
5. If (it is true, that) $Max = Min$ or (that) *Deviation* <= *AllowedDeviaton*, find a solution with cost equal or less than *Max* (we know that there exists a solution with such limitation) and stop, otherwise go to step 1.

Step 5 can be saved sometimes if the solution found in the step 3 can be remembered.

---

[1] The cost of the found solution can never be greater then *Limit*.

5.3 Approximate version of logarithmic min_max algorithm

While finding an optimal result, it sometimes happens that the *logarithmic min_max* algorithm quickly finds the optimal result (to be used as an upper bound) but the proof of its optimality takes too much time by decreasing the < *Min*, *Max*) intervals.

Because of this, we have developed a modification of the *logarithmic min_max* algorithm. If the *approximate version of logarithmic min_max* detects *N* proofs with the same upper bound, it directly sets *Limit* = *Max* – 1 for the next iteration and if there cannot be found a result the cost of which is lower or equal to Limit, the *approximate logarithmic min_max* algorithm returns the last but one schedule as the optimal one.

The question is, which number should be set *N* to. We tried 30 randomly generated Job-Shop problems (for 5, 8 and 10 machines) and tried to set *N* to 2, 3, 4, 5 and 6. In benchmark tests the best results gave the *approximate logarithmic min_max* with *N=3*.

5.4 Deterministic lower bound

The *logarithmic min_max* algorithm needs to know the lower bound. If we know that the cost of the result cannot be lower than some limit, this limit is called the lower bound. In the Scheduling Program we are trying to find a schedule of tasks and the cost of this schedule is equal to duration of all Job-Shop.[2]

Let us consider Job-Shop with *n* tasks, which have to be processed on *m* machines. We do not allow tasks with zero processing time. The total processing time of all jobs will surely not be shorter than processing time of all tasks in one job, i.e. $LB_J = \max_{i=1}^{n}\{d(J_i)\}$, where $\boldsymbol{LB_J}$ is the lower bound obtained through the tasks in jobs and $\boldsymbol{d(J_i)}$ is the processing time of all tasks in the job $\boldsymbol{J_i}$.

All tasks which are sharing the same resource (machine) must be processed sequentially. Therefore we can say that $LB_1 = \max\{LB_J, LB_M\}$, where $LB_J = \max_{i=1}^{n}\{d(J_i)\}$ and $LB_M = \max_{i=1}^{n}\{d(M_i)\}$. $\boldsymbol{d(M_i)}$ is the sum of processing times of all tasks which have to be processed on the machine $\boldsymbol{M_i}$ and $\boldsymbol{LB_M}$ is the lower bound obtained through the tasks which should be processed in one machine.

---

[2] We start performing of the Job-Shop in time 0. The duration of all Job-Shop is the time in which is each task of the Job-Shop completed. In this paper is the cost of Job-Shop equal to its duration.

Now, if we consider the technological constraints which must be satisfied, it sometimes happens that there exists a task that cannot be processed as the earliest (or as the latest) in the resulting schedule.
If before task T_i which has to be processed on the resource (machine) M_q must be sequentially processed group of tasks, sum of processing times of which is $d(P_i)$ then let $b(M_q) = \min_{Ti \in Mq} \{d(P_i)\}$.

Similarly, if after task T_i which has to be processed on the resource Mq must be sequentially processed group of tasks, sum of processing times of which is $d(Z_i)$ then let $a(M_q) = \min_{Ti \in Mq} \{d(Z_i)\}$.

If $LB_G = \max_{Mq \in M} \{b(M_q) + d(M_q) + a(M_q)\}$ where M is the set of all machines (resources), we can say that $LB = \max\{LB_G, LB_J\}$. It may be greater lover bound than LB_1.

5.5 Lower bound obtained through partial scheduling

It is possible to increase the LB (lower bound) if we do some partial scheduling. For that purpose, we can use the *logarithmic min_max* algorithm. For speeding up the lower bound finding, we use partial schedule of a small number of tasks with the largest processing times. We stop the *logarithmic min_max* algorithm when the lower bound of partial schedule is less than 10% closer to the cost of the optimal partial schedule. This algorithm allows us to find good lower bound very quickly.

5.6 Upper bound

A classical method for obtaining starting solution is to use priority dispatching rules (Caseau,1995): the schedule is constructed chronologically, tasks are selected one after the other and performed as soon as possible. The algorithm works as follows: At each step, a set of "selectable" tasks is kept. In the beginning, this set initialised to the set of all tasks that are first in a job (i.e. tasks that do not require any other task to be performed before them). One of the tasks in this set is selected and scheduled as soon as possible on its machine. It is then removed from the set of selectable tasks and replaced in this set by its direct successor in the job. This process is repeated until no more tasks are to be selected.

6 PROPLANT

SP can be used also within the new multi-agent system which is being developed at the CTU Prague called ProPlanT which differs in some aspects of the agents organisation and communication. The main features of the ProPlanT architecture are presented in the following.

Agents' communication is one of the most critical aspects of agents' activities. Communication usually causes much more delays than the computational processes themselves. To speed up the communication, there is used a acquaintance net protocol in ProPlanT. Each agent has a tri-base model which consists of the following three information bases:

a) *a cooperator-base* collects permanent information about the other agents (addresses, data formats, behaviour, capabilities etc.),
b) *a state base* contains the current information on the states of all the agents (their expected loads, global trust into their reliability etc.),
c) *a task-base* contains dynamically, very often up-dated and changed information on the community state.

The most important role in ProPlanT plays a special meta-agent called *a co-operation trader*. The meta-agent permanently observes the message traffic, collects information in the activities of the other agents in the system, evaluates the data and tries to utilise the obtained results in order to increase efficiency of the overall system. It permanently creates, modifies and maintains the agents' internal qualitative model of the situation inside the multi-agent system. It performs periodical revisions of the content of tri-bases located in the agents' wrappers. Thus, the task-base always contains highly up-to-date information about the current capability of the others. This fact enables to direct the co-operation requests to the most suitable and applicable agents in the community. As a result, the communication traffic is significantly reduced and the reactions of the multi-agent system are becoming substantially faster than if the simple broadcasting used.

7 CONCLUSIONS

A new agent for DISCIM is being currently developed at the Technical University of Kosice. The basic idea is to create an agent based on Scheduling Program developed at the Technical University of Kosice. The agent will use a lot of artificial intelligence techniques and some new techniques like *approximate logarithmic min_max*, techniques for good lower and upper bounds finding and others. It will allow the other agents to choose optimal or sub-optimal solution (which can be found in seconds).

8 REFERENCES

Caseau Y., Laburthe F. (1995) Disjunctive Scheduling with Task Intervals, *LIENS Technical Report 95-25*, Laboratoire d'Informatique de I'Ecole Normale Superieure

Dincbas M., Hentenryck P. Van, Simonis H., Aggoun A., Graf T., Berthier F. (1988) The Constraint Logic Programming Language CHIP, Proc. of the *International Conference on fifth Generation Computer Systems, ICOT*

French S. (1987) *Sequencing and Scheduling*: An Introduction to the Mathematics of the Job-Shop Wiley & Sons, Chichester, England

Hazdra T., Koutnik J. (1995) Graphical Interface for Distributed Expert System, *Technical Report OAKBS-95/12*, FAW Linz

Jaffar J., Lassez J. L. (1987) Constraint Logic Programming, Proc. of the *14th Symp. POPL'87*, Munich, January, pp. 111-119

Jaffar J., Michaylov S., Stuckey, P., Yap R. (1992) The CLP(R) language and System, *ACM Transactions on Programming Languages*, Vol. 3, Nr. 14, pp. 339-395

Marik V., Lazansky J., Koutnik J., Hazdra T. (1994) Distributed System for CIM, Proceedings of 5th *International Conference DEXA'94*, Athens, Greece, pp. 785-594

Paralic J., Schmotzer M., Csonto J. (1997) Optimal Scheduling Using Constraint Logic Programming, Proceedings of 8th *Symposium on Information Systems IS'97*, Varazdin, October, pp. 65-72

Schmotzer M. (1997) *Analysis and design of new methods for time scheduling problems solving in constraint logic programming environment*, MSc Thesis, Technical University of Kosice, Kosice, Slovakia

9 BIOGRAPHY

Milan Schmotzer, M.Sc., the main author of this paper, studied Technical Cybernetics and Artificial Intelligence at the Technical University of Kosice. In 1997 received M.Sc. degree (Diploma work: Analysis and design of new methods for time scheduling problems solving in constraint logic programming environment.). Now is a Ph.D. student at the Technical University of Kosice. Research focused on CLP based scheduling, multi-agent intelligent systems, expert and knowledge systems. Developing the ProPlanT multiagent system in a close cooperation with the Gerstner laboratory of CTU Prague.

Jan Paralic, M.Sc. studied Technical Cybernetics and Artificial Intelligence at the Technical University of Kosice, received M.Sc. degree in 1992. Since 1992 to 1997 worked on Ph.D. thesis „Solving Scheduling Applications Using Constraint Logic Programming". Since 1996 works at the Dept. of Cybernetics and AI at the Technical University of Kosice. Research focused on constraint (logic) programming and operations research, mainly scheduling.

Assoc. Prof. Ing Julius Csonto, Ph.D., studied Control Engineering at the Czech Technical University, received M.S. degree in 1965. From 1973 to 1977 worked on Ph.D. thesis "Data compression algorithms" and received Ph.D. in 1978. In 1981 was promoted to associate professor. Research focused on biocybernetics (automatic ECG and EEG recognition), digital image preprocessing (data compression techniques). Recently has become increasingly involved in uncertainty processing in expert systems with special emphasis on use of multiple-valued logic, logic programming (and CLP) applications, biocybernetics and artificial life.