ABSTRACT PLANNING IN DYNAMIC ENVIRONMENTS

DAVID CAMACHO, DANIEL BORRAJO, JOSE M. MOLINA, RICARDO ALER

Computer Science Department, Universidad Carlos III Madrid
Avda. Universidad nº 30, 28911 Leganés (Spain)
{dcamacho, dborrajo, molina}@ia.uc3m.es, aler@inf.uc3m.es

Abstract

Solving problems in dynamic and heterogeneous environments where information sources change its format representation and the data stored along the time is a very complex problem. In previous work we have presented a system called MAPWeb (MultiAgent Planning in the Web) that tries to solve those problems by integrating artificial intelligence planning techniques within the MultiAgent framework. Basically, MAPWeb allows cooperative work between planning agents and Web agents. The purpose of MAPWeb is to find solutions to travel problems. In order to give detailed solutions, MAPWeb uses information gathering techniques to retrieve travel information that is made available by many different companies. However, Web access to the information sources is quite time expensive. In this paper, we try to minimize the number of Web queries by using caching techniques based on relational databases. Experimental results show that the reduction in Web access time is quite important, while maintaining the number of solutions found.

Keywords

1 Introduction

Currently, there is a vast (and increasingly growing) amount of information stored in Internet, especially since the development of the Web. This information is difficult to handle because it is heterogeneous, dynamic and distributed in nature. However, there are very few approaches that try to integrate a set of different and specialized information sources and reuse the data retrieved to solve problems [1, 2]. This is especially true if the goal of the system is to solve complex problem solving tasks, like finding complete travel plans by gathering information available on the Web.

In order to use Web information to solve complex problems, a framework called MAPWeb (MultiAgent Planning in the Web) has been developed and applied to solve travelling problems [6]. The key aspects of MAPWeb are:

- Solving complex problems requires using intelligent software components. MAPWeb integrates AI planning techniques with information gathering techniques. It actually divides the planning problem into two processes: abstract planning (performed by an AI planning system) and plan completion and validation (performed by Web gathering agents). This makes the planning problem more tractable, at least in some domains. On the other hand, using AI planning makes our approach very flexible.
- MAPWeb is a Multiagent System (MAS) [3, 9, 11]. This is very appropriate for our purposes because of MAS flexibility. First, it is easy to divide the task into different agents, each one containing the most appropriate skills. Thus, MAPWeb has PlannerAgents and WebAgents. Second, it faces the heterogeneity of Web sources by having different specialized Web agents. Third, the Web is a dynamic environment (Web sources are added/removed daily). To face this problem, MAPWeb takes advantage of the flexibility of MAS approaches for adding/removing agents to the agent society (besides the flexibility offered by classical AI planning).

However, MAPWeb (and any Web gathering system) must send many Web queries in order to access the desired information. This is usually costly in terms of time. Besides, in some cases a Web source might be temporally unavailable. In order to improve MAPWeb performance, learning techniques can be used in two ways: adding CBR skills to the PlannerAgents to store abstract plans [8] and adding caching skills to the WebAgents to store previous Web queries. In this paper we focus on the second aspect and show that the time cost can be decreased without significantly reducing the number of solutions found.

This paper is divided into 5 sections. Section 2 describes the MAPWeb architecture from a Multiagent perspective. Section 3 describes the caching skills of the WebAgents. Section 4 evaluates MAPWeb caching skills. And finally, section 5 summarizes the conclusions of the paper.
2 MAPWeb Architecture

MAPWeb is structured into several layers whose purpose is to isolate the user from the details of problem solving and Web access. Each of these layers is implemented by a set of heterogeneous agents, which have different skills to solve the user problems. This multi-layer architecture can be seen in Fig. 1.

![Fig. 1: MAPWeb system architecture.](image)

MAPWeb deploys this architecture using a set of heterogeneous agents. Next, each of these types of agents will be described:

- **UserAgents**: They pay attention to user queries and display to the users the solution(s) found by the system. When an UserAgent receives problem queries from the users, it gives them to the PlannerAgent and when it answers back with the plans, the UserAgent provides them to the user.

- **PlannerAgents**: They receive a user query, build an abstract representation of it, and solve it by means of planning. Then, the PlannerAgent fills in the information details by querying the WebAgents. The planner that has been used by the PlannerAgent is Prodigy4.0 [10].

- **WebAgents**: Their main goal is to fill in the details of the abstract plans obtained by the PlannerAgents. They obtain that information from the Web.

The way these agents cooperate is as follows. First, the user interacts with the UserAgent to input his/her query. The query captures information like the departure and returns dates and cities, one way or return trip, maximum number of transfers, and some preference criteria. This information is sent to the PlannerAgent, which transforms it into a planning problem. This planning problem retains only those parts that are essential for the planning process, which is named the abstract representation of the user query. Prodigy4.0 provides several abstract solutions to the user query. The planning operators in the abstract solutions require to be completed with actual information that can be retrieved from the Web. To accomplish this, the PlannerAgent sends information queries to specialized WebAgents, which return several records for every information query. Then, the PlannerAgent integrates and validates the solutions and returns the data to the UserAgent, which in turn displays it to the user. MAPWeb agents use a subset of the KQML speech acts [7].

3 Caching techniques in the WebAgents

In this paper we have used caching techniques to reduce the number of actual Web queries. Caching has been implemented as a new skill for the WebAgents. MAPWeb caching follows the typical caching schema:

- Whenever an actual Web query is successful, a record is stored in the cache memory of the WebAgent. Cache memories have been implemented as local relational databases for each one of the WebAgents.
- Whenever a WebAgent is asked to carry out a query by the PlannerAgent, it first looks up its local database. If a record is found, then it is marked as if it were a new entry and it is returned to the PlannerAgent. If not successful, then it queries the Web directly.
- The cache memories have been limited in size. When the cache is full, old entries are removed by following a LRU policy (Least Recently Used: the oldest entries are deleted). As explained before, successful entries are automatically made young. This guarantees that useful entries will not be removed.

4 Experimental Evaluation

The purpose of this section is to compare the standard MAPWeb with a modified version where the WebAgents have extended caching skills.

Subsection 4.1 describes the domain that has been used to prove MAPWeb. Subsection 4.2 explains the topology that has been used in the experiments. Subsection 4.3 describes the experimental setup. And finally, Section 4.4 shows the actual results.

4.1 Problem Domain: e-Tourism

An e-tourism system must provide the user services such as:

- Inform how to go from the origin to the destination town using different means of transport.
- Lodging at destination.
Informing about possibilities for visiting around town (renting a car, local transport, etc). Returning to the initial (or other) town.

MAPWeb has the abilities enumerated above [4, 5]. However, in this paper, we will focus on the logistics problem of providing the user with plans to move from one place to another place.

Moving from place to place involves long range travels that can be achieved by means of airplanes, trains, or buses. It also involves taking local transport means (taxi, subway, bus, etc...) to move between airports, bus stations, or train stations. In order to represent and provide solutions to the user, we have defined an e-tourism domain that uses different planning operators like: [Travel-by-airplane, Travel-by-train, Travel-by-bus, Move-by-localbus, Book-hotel-room, etc...].

4.2 MAPWeb Topology

Due to the heterogeneous nature of the agents that implement MAPWeb, it is possible to build different topologies. Those topologies could be used to study the performance system within a particular problem. In this paper a very simple topology was used (see Fig. 2).

This topology includes a single PlannerAgent and four specialized WebAgents. Two of them offer information supplied by airplane companies (Iberia\(^1\) and Avianca\(^2\)) whereas the other two are meta-searchers (Amadeus\(^3\) and 4Airlines\(^4\)). A meta-searcher is a Web information source that is able to look in several information sources.

This topology employs only one reasoner agent that uses its planning skills to solve the problem given by the UserAgent and the WebAgents like softbots or searchers to validate and complete the solutions found. This topology should be analyzed like a monolithic planning application, which uses a set of distributed WebAgents in parallel.

---

1 Iberia airlines: http://www.iberia.com
2 Avianca airlines: http://www.avianca.com
3 Amadeus: http://www.amadeus.net
4 4Airlines: http://www.4airlines.com

\(^5\) A transfer is a point in the trip where the traveler can change plane, the means of transport, etc.

---
that the number of solutions for the non-caching and caching configurations is about the same (not many solutions are lost by using caching).

Fig. 3: Number of Abstract and Specific solutions found by MAPWeb without caching.

Fig. 4: Number of Abstract and Specific solutions found by MAPWeb with caching.

Fig. 5 shows the number of queries with and without caching. The number of queries is the same for the two experiments; the only difference between the two configurations is the number of Web accesses that the WebAgents (with caching skills) will finally perform.

Fig. 5: Number of queries between the PlannerAgent and the WebAgents.

Figs 6 and 7 display the response time with and without caching, respectively. The reduction of the request time in MAPWeb is related with the successful queries that the WebAgents have stored in their local databases. By comparing Fig. 7 with Fig. 6, we can see that request time grows more slowly with caching. This is because it is possible to find more stored information in the agent.

Fig. 6: Request time by MAPWeb without caching.

Fig. 7: Request time by MAPWeb with caching.

Request times are related to the hit ratio obtained by the WebAgents when they find the requested information in their own local databases. Table 1 shows the total number of actual Web queries with and without caching, and the hit ratio when using caching. Only 1 transfer problems are considered. On average, the number of Web queries was reduced by 25%.

Table 1: Average of Web queries cached by WebAgents.

<table>
<thead>
<tr>
<th>Nº Web Access</th>
<th>Iberia WebAgent</th>
<th>Avianca WebAgent</th>
<th>4Airlines WebAgent</th>
<th>Amadeus WebAgent</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Caching</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>60</td>
</tr>
<tr>
<td>With Caching</td>
<td>39</td>
<td>53</td>
<td>47</td>
<td>40</td>
</tr>
<tr>
<td>Hit ratio</td>
<td>35%</td>
<td>12%</td>
<td>22%</td>
<td>33%</td>
</tr>
</tbody>
</table>

Table 2 shows the different answer time by the WebAgents when the caching techniques are used. This table shows how the performance of the slower agents (Amadeus-WebAgent) could be improved through the caching technique, and how the agents that have a lower hit ratio of success (Avianca-WebAgent) do not improve very much their request time.

Table 2: Average of request time for different WebAgents in the MAPWeb Topology.

<table>
<thead>
<tr>
<th>Request Time</th>
<th>Iberia WebAgent</th>
<th>Avianca WebAgent</th>
<th>4Airlines WebAgent</th>
<th>Amadeus WebAgent</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Caching</td>
<td>88.4 sec</td>
<td>96.4 sec</td>
<td>156.4 sec</td>
<td>1440.5 sec</td>
</tr>
<tr>
<td>With Caching</td>
<td>39 sec</td>
<td>84.6sec</td>
<td>50.1sec</td>
<td>56.3 sec</td>
</tr>
</tbody>
</table>

Figs. 8, 9, 10, and 11 show the number of solutions found for each of the 35 user queries (on the x-axis). There are no important
differences between the two configurations of MAPWeb. These figures show that the caching techniques allow the system to gain efficiency without losing possible solutions.

5 Conclusions

Planning in real domains is a complex task. We present here a solution that separates abstract planning from the actual data, that is obtained from the Web. In order to give detailed solutions, for a particular travel domain, MAPWeb implements two different steps. First, it uses abstract planning to build a skeletal solution for the problem. And second, it uses information gathering techniques to retrieve travel information from Web sources. However, Web access to the information sources is quite time and computational expensive. In this paper, we show how it is possible to minimize the number of Web queries by using caching techniques based on relational databases, and therefore, to find solutions in a more efficient way. Experimental results show that the reduction in Web access time is quite important, while maintaining the number of solutions found.

Acknowledgments

The research reported here was carried out as part of the research project funded by CICYT TAP-99-0535-C02. (http://decsai.ugr.es/~lcv/SEPINtap99-0535-c02-01.html)

References


