The Distributed Weighing Problem: A Lesson in Cooperation Without Communication

Tibor Bosse[1], Mark Hoogendoorn[1], and Catholijn M. Jonker[2]

[1] Vrije Universiteit Amsterdam, Department of Artificial Intelligence,
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
{tbosse, mhoogen}@cs.vu.nl
[2] Radboud Universiteit Nijmegen, Nijmegen Institute for Cognition and Information,
Montessorilaan 3, 6525 HR Nijmegen, The Netherlands
C.Jonker@nici.ru.nl

Abstract. Cooperative problem solving without communication is an often-studied field within multi-agent research. Realistic problems investigated in this particular field are complex and difficult to model, and therefore not suitable for education. This paper presents the distributed weighing problem as a novel problem to be used for educational purposes within the domain of cooperation without communication. An example agent-based architecture is developed of which parts can be provided to students as a starting-point for practical exercises in cooperative problem solving without communication. Two example strategies are discussed and implemented using this example architecture. Moreover, it is shown how such strategies can be tested and formally validated against a number of desired properties. The educational benefits of the distributed weighing problem are presented as observed in a course for 6 groups of each 3 students.

1 Introduction

Coordination and cooperation between agents has been a topic of research for many years (see for example [5] and [9]), and is a part of everyday life. For instance, within the port of Rotterdam coordination is essential [14]. The port has approximately 24 terminals and about 120 sea vessels and barges are continuously loading and unloading containers. Coordination is needed to determine routes for the vessels and barges in such a way that they do not have to wait too long for other ships when they arrive at a certain terminal.

In the above example, the port authorities communicate with the ships to determine a schedule that is satisfactory for all parties. However, in a number of cases of cooperation between agents, no communication takes place (see also [7]). For instance, without communication people often coordinate their actions so that they do not bump in to each other on the street. Management games such as the broken squares problem [1] have been developed to train people in cooperation without communication. The broken squares problem requires the players to each construct a square out of a set of given parts. The parts, however, are distributed randomly across the individuals, therefore they need to forward those broken parts to other players for

T. Eymann et al. (Eds.): MATES 2005, LNAI 3550, pp. 191 – 203, 2005.

which those parts are useful. The problem is solved when all individuals have formed their individual goal square. Also in the domain of software agents cooperation with limited or no communication plays a role. Agents might not have all the information or abilities they need to reach a certain goal. They may need to cooperate with other agents to be able to reach that goal. The problem solving capacity of the overall system increases with the cooperation capacity of the agents in the system. However, the communication load increases with the number of agents, which makes it attractive to solve the cooperation problem using limited communication.

As the examples show, in a number of cooperation problems communication is not possible or unnecessary. Students in the fields of artificial intelligence, computer science, information science, and management need to be familiar with solving such problems. Especially, the IT-related students need to be trained in the development of cooperative software agents. Suitable problems for educational purposes are problems that can easily be modelled. Scalability of the problem allows for testing the generality of the solution. Problems identified in the literature for cooperation without communication are often very hard to model. The broken squares problem for example entails modelling the different shapes of the pieces of the puzzle, the shapes that can be created when combining pieces of puzzles, and so on. Therefore it is hard to get students to study this type of problem in depth.

To improve the quality of education in cooperation without communication, this paper introduces a problem that is purely cooperative and can be solved without communication. The problem is derived from the *twelve balls problem* (see [11]), also known as the twelve coins problem. It involves twelve balls, each with the same appearance, of which one has a deviating weight which can either be lighter or heavier compared to the other balls. The balls can be put on a balance and weighed against each other. The goal of the problem is to find the deviating ball, and determine how it deviates from the rest (whether it is heavier or lighter). An additional restriction applies: The maximum number of weighings allowed is three.

The twelve balls problem, which is initially a centralized problem, can easily be modelled as a distributed problem without communication. To this end, each ball is represented by one agent that can decide to jump on the balance or not. The common goal is to derive the solution for the problem. The agents representing the balls are not allowed to communicate with each other. As the emphasis is on cooperation and not on efficiency of the solution, the requirement of solving the problem within three weighings is dropped. To ensure the eventual solution of the problem, no repetition of weighings is allowed, so the combined strategy of the agents should always result in performing a different weighing than before. In the rest of the paper this distributed problem is addressed as the *distributed weighing problem*. The number of balls can easily be scaled to any preferred number.

The distributed weighing problem is explained in more detail in section 2. Section 3 describes an example design of a component-based multi-agent system that models the problem. This design can serve as a starting-point in practical assignments in cooperation without communication. Example strategies that can be used by the agents are shown in more detail in section 4. Section 5 shows how such strategies can be tested and formally validated against a number of desired properties. Experiences in using the problem in education are discussed in Section 6. Section 7 compares the work with related literature and presents conclusions.

2 The Distributed Weighing Problem

This section describes the distributed weighing problem at a more detailed level, including the assumptions that agents are allowed to make on the behaviour of other players in the puzzle. Two possible protocols for the problem are introduced, and their differences discussed. Thereafter, a theoretical overview is presented of the possible types of agents one can encounter when solving the problem.

2.1 Puzzle Design

The distributed weighing problem is a derivative of the twelve balls problem, explained in the introduction, in which an arbitrary number of balls is used. Each ball is represented by an agent. All balls but one have exactly the same weight. The deviating ball is either heavier or lighter than the others. The goal of the game is to determine which ball is the deviating ball and to determine the deviation (heavier of lighter). The puzzle is solved when at least one ball knows the answer. Actions of a ball are observable to others. A ball has the following options:

1. **join_left.** If not already on the balance, the ball agent can select this option when it wants to join the left scale.
2. **join_right.** If not already on the balance, the ball agent can perform this action in case it wants to join the right scale.
3. **do_nothing.** This option can be used in case the ball agent is satisfied with the current balance configuration. If the agent is already on the balance, this is its only course of action. This design choice simplifies the problem of detecting that an agent decides not to join any scale.

A balance is available to perform the weighings determined by the balls. The game begins with an empty balance. After a weighing is performed, all agents are automatically removed from the balance.

The weighing process is performed according to a *sequential* protocol. In this protocol all balls get turns in a predetermined order. At the beginning an empty balance is observed by the first ball in line. After a ball has performed the chosen response action (possibly the observable "do_nothing"), the next ball observes the new situation. For example, ball B observes a situation with ball A on the left scale and an empty right scale. After all agents have performed the "do_nothing" action after each other, the balance performs the weighing and the agents observe the result of the weighing (i.e., one scale heavier than the other, or both scales equal). When playing this protocol, each ball knows after which other agent it has the turn. The first ball knows it has the turn at the beginning of the game.

In fact, the sequential protocol reflects a specific type of cooperation without communication. In such cooperation problems, where the parties involved contribute to the problem sequentially, their main concern is *what* action they have to perform. In cooperation problems where the parties involved contribute to the problem in parallel, an additional concern is *when* to perform the action. In future work, the distributed weighing problem will be used to study parallel cooperation as well.

2.2 Types of Agents

In order to ensure success, the agents are allowed to assume that the other agents will also behave with some intelligence and with the same goal in mind. In this section (and only this section) this assumption is dropped for the purpose of identifying all possible kinds of agents. In principle, three different types of agents can be identified:

A. Nasty. This agent tries to cause loops. A loop means that the same weighing is done over and over again. Nasty agents do not meet the benevolent requirement presented in [13] which roughly states that agents want to help each other whenever possible.

B. Dummy. A dummy agent performs arbitrary moves without any notion of previous weighings, or the possible consequences of his moves. Therefore the strategy can non-intentionally cause loops. Such an agent might also confuse agents that try to solve the problem in a more intelligent fashion.

C. Progressive. Progressive agents are those who always move towards the solution. Three basic strategies can be distinguished in the behaviour of the agent:

C-a. Non-repetitive. An agent that follows a non-repetitive strategy tries to prevent a weighing that has already been done. If the other agents are also non-repetitive, no loops will emerge during the problem solving process. Success is ensured if the agents keep performing weighings until they know the answer.

C-b. Information-eager. Agents following an information-eager strategy only aim for weighings from which new information can be derived. For example, a first weighing of ball A and B on the left scale against ball C and D on the right scale results in a balance. Therefore, it is known that all the balls are non-deviant, and a weighing of ball A against ball B would not add any information and is not accepted in the information-eager strategy.

C-c. Efficient. For each number of balls there is a minimal number of weighings that is always enough to find a solution. For the general problem (from [10]) with a maximum of n weighings the maximum number of balls for which the solution can be determined, m, is defined as: $m \leq (3^n - 3) / 2$. In case of the twelve balls problem that is always three weighings. An agent that follows an efficient strategy aims at finding the solution in that amount of weighings.

For educational purposes the focus is on agents of type C. Agents of type A and B are considered less interesting, because they do not cooperate. When strictly following the strategies of type C, the robustness relationships depicted in Table 1 hold. Robust means that the agents find a solution. As the table shows, strict C-a agents are robust against all other C agents, since these also try to prevent repetition. Strict C-b agents are robust against other strict C-b agents and strict C-c agents, but not against strict C-a agents. This is because a strict C-b agent assumes that the others are also information-eager. If this is not the case, a situation might occur which the agent cannot handle. For example, consider the situation sketched above when it is known that ball A and B are non-deviant. Then, a strict C-a agent might still propose a weighing of ball A against B, whilst a strict C-b agent will not allow this. As a result, the C-b agent will not be able to derive any appropriate action for the situation. For similar reasons, strict C-c agents are only robust against other strict C-c agents.

Section 4 presents the strategies of a type C-a and of a type C-b agent. A type C-c agent is not considered, since the focus is on finding a solution using a simple strategy, not on finding it in an efficient manner. It is however easy to incorporate such a strategy in the design presented in the next section.

Table 1. Robustness of C-strategies

	C-a	C-b	C-c
C-a	+	+	+
C-b	-	+	+
C-c	-	-	+

3 Example Design of a Multi-agent System for the Distributed Weighing Problem

This Section presents an example design of a multi-agent system for the distributed twelve balls problem. Students can, based on their experience in modelling and designing multi-agent systems, be provided with parts of this model as a starting-point for the exercise.

3.1 Top Level

The multi-agent system is designed using the component-based agent modelling approach DESIRE; cf. [3]. The highest level of abstraction consists of the agents representing the balls (called *ball_A*, *ball_B*, and so on) and the *External World*. The agents can perform actions in the external world, and observe the external world.

The execution of actions generated by the agents is modelled as follows. After an agent generates a certain action to be performed (e.g., jump on the left scale of the balance), this action is transferred to the external world, where the result of the action will occur (e.g., ball A is currently on the left scale of the balance). Thus, the execution of physical actions by the agents is modelled as part of the component external world. The action *do_nothing* represents the fact that an agent does not move for a certain period of time. Introducing this as an action makes the problem of knowing when an agent finished his turn easy.

Besides performing actions, the agents can observe the world. In the simplest model, these observations are not modelled explicitly (i.e., the agents do not have to determine pro-actively when to observe what). Instead, every relevant aspect of the world is transferred automatically from the external world to the agents in the form of *observation results*. These observation results include the current position of the balls on the balance, the results of weighings, and the actions performed by others.

3.2 Agent Level

The composition of the agents is based on the generic agent model as described in [3]. In the current model, four of the generic agent components are used, namely *Own Process Control*, *World Interaction Management*, *Maintenance of Agent Information*, and *Agent Specific Task*.

According to the generic agent model, the task of the component *Own Process Control* is to control the agent's own activities (e.g., determining, monitoring and evaluating its own goals, plans and characteristics). In the current domain, this is done by maintaining the following information: the agent's own name, the name of the

current protocol, and other information associated with the protocols. For example, for the sequential protocol, the agent needs to know either the order in which the agents are allowed to perform actions (e.g., A-B-C-D-E-F-G-H-I-J-K-L-A), or the name of the agent ahead of it.

The component *World Interaction Management* takes care of the processes involved in interaction with the external world, i.e., observation interpretation, directed observation initiation, and action performance. The component passes actions and observation results (e.g., concerning the current position of the balls on the balance) from the relevant other components to the world and vice versa.

The task of the component *Maintenance of Agent Information* is to maintain (partial) agent models, containing relevant information about the state of the surrounding agents. In most applications, this information is obtained in two different ways: by observing the other agents and by communicating with them. Obviously, in the distributed weighing domain only the first approach occurs. For this domain, the agent models are restricted to the assumed weights of the agents (including itself). At any time, to each agent exactly one of the following values is assigned: {unknown, neutral, heavier_or_neutral, lighter_or_neutral, heavier, lighter}. Initially, each agent gets the value unknown. In later stages of the process, these values are updated in accordance with the observed weighing results. A number of knowledge rules are used to perform this modification:

- each ball occurring in a *balanced* weighing gets the value neutral
- each ball *not* occurring in an *unbalanced* weighing gets the value neutral
- each ball occurring on the *lower* scale in one weighing, and occurring on the *higher* scale in another weighing, gets the value neutral
- each unknown ball occurring on the *lower* scale in a weighing gets the value heavier_or_neutral
- each unknown ball occurring on the *higher* scale in a weighing gets the value lighter_or_neutral
- if one ball is lighter_or_neutral and all other balls are neutral, then this ball gets the value lighter
- if one ball is heavier_or_neutral and all other balls are neutral, then this ball gets the value heavier

Moreover, it is assumed that all agents have perfect recall.

Within the generic agent model specific tasks (e.g., design, diagnosis, information retrieval) can be modelled in the component *Agent Specific Task*. For the current domain, the specific task can be described as the determination of actions to be performed, based on the current situation of the balance. Thus, the output of this component is a proposal of the form join_left, join_right, or do_nothing. The exact knowledge used within Agent Specific Task depends on the strategy used by the agent, as described in the next section.

4 Example Strategies

This section describes a concrete example of a strict non-repetitive and a strict information-eager strategy. These examples show that the problem is relatively easy to solve. Moreover, they illustrate what comes into play when designing such

strategies. The current strategies were used to construct strict C-a and strict C-b type agents that were tested in different combinations (see Section 5).

4.1 Non-repetitive Strategy

Non-repetitive strategies require looking ahead at the possible options that can still be performed without resulting in repetition. Without these calculations it is impossible to determine whether jumping on a scale can or cannot result in a new weighing of m to m balls, for some m. Two solutions are considered here: (1) Generate all possible weighings; (2) Use a mathematical formula to calculate the amount of options left.

A first option is to generate all possible weighings that might result after jumping on one of the scales, until you find one that has not been done in the past. If such a weighing cannot be found, try the same for jumping on the other scale. If that does not work either, then don't jump on any scale. However, its exponential character makes this option unsuitable for scaling to larger numbers.

A second option is to mathematically calculate how many possible weighings can be constructed in total, considering the current balance after an action of the agent (i.e. join_left, join_right, do_nothing), and the amount of balls still not on the balance. Thereafter, sum up the amount of past weighings of which the current proposal combined with the action of the agent is a subset. If there are more possible weighings than past weighings, the action for which the calculation was done is allowed to be performed. The number of possible weighings can be calculated as follows: Choose a type of weighing: m:m, where m varies from 1 to half the number of balls in the game. Let l denote the amount of available places on the left side of the balance with respect to your choice m, and r the amount of available places on the right side again with respect to m. For example, if you aim for a 3:3 weighing, and there is already one ball on the right scale, then r is 2. The amount of balls not on the balance is n. A formula to calculate the number of possible weighings w given these parameters is:

$$w = \frac{\left(\prod_{i=0}^{l-1}(n-i)\right)}{l!} \times \frac{\left(\prod_{j=l}^{l+(r-1)}(n-j)\right)}{r!}$$

This number can be calculated for every possible value of m. The specific strategy determines which value is attempted first. The agent has the following arbitrary preference: (1) join_left; (2) join_right; (3) do_nothing.

4.2 Information-Eager Strategy

An agent that follows an Information-Eager strategy aims at weighings that provide some new information. Thus, if all agents are of this type, such weighings will indeed be performed until one of the agents solves the problem. In other words, after an Information-Eager agent has performed an action, the possibility to obtain a weighing that provides new information is still open (unless this was already impossible before the agent's action). A strategy of this type does not need to consider all remaining possibilities, because it can make use of its knowledge about the weights of the existing balls. For example, if a certain agent has the value lighter_or_neutral, and there is a ball with value heavier_or_neutral on the left scale, then it may be wise to join this

ball on the left scale, because the resulting weighing is guaranteed to change the value of one of these balls (as long as other balls "complete" the weighing to ensure that both scales contain an equal amount). A number of different strategies of this type can be implemented. The strategy that is described in this section uses a two-step algorithm. In the first step the current situation is classified. In the second step an appropriate action is selected, based on the current situation.

In the first step, a number of different situations can be distinguished. The main distinction is between *advantageous* weighings and *non-advantageous* weighings. A weighing is advantageous if it is guaranteed to provide new information, no matter what the other balls do (as long as they "complete" the weighing, which is assumed). Advantageous weighings are weighings that (1) contain an unknown ball, (2) contain a heavier_or_neutral and a lighter_or_neutral ball on one scale, (3) contain a heavier_or_neutral ball on both scales, or (4) contain a lighter_or_neutral ball on both scales. Examples of the non-advantageous weighings are the case that all balls are neutral, the case that one ball is heavier_or_neutral and the rest is neutral, and so on.

When the current situation is classified, an appropriate action can be determined. In order to do this, the current strategy uses the algorithm depicted in Figure 1. As can be seen in the figure, first an agent has to verify whether it is already on the balance, because then the only possible action is do_nothing. When the agent is not on the balance, it checks whether one of the scales contains the *maximally allowed* number of balls, which is exactly half of the total number of balls (e.g. for the twelve balls problem, it is 6). If this is the case, the agent has to jump on the other scale in order to complete the weighing. The next step is to check whether an action exists that immediately results in an advantageous weighing. For example, if a certain agent has the value lighter_or_neutral, and there is already a ball with value lighter_or_neutral on the left scale, then join_right is such an action.

However, if such an action cannot be found, then the action to be performed depends on the specific situation. For reasons of space, the knowledge used in this last step is not represented completely in Figure 1. However, an example sketch of such a knowledge rule is the following: "if I am neutral, and the left scale contains more balls than the right scale, and there are still two lighter_or_neutral balls **not** on the balance, then I will join_left". The idea of this rule (and of many similar rules) is that the agent leaves empty spaces for the balls of which it is known that they will contribute to an advantageous weighing. Note that in general, this strategy has a preference for jumping on the balance when possible. An advantage of this approach is that the action do_nothing is often avoided, which minimizes the risk of accidentally accepting a non-advantageous weighing.

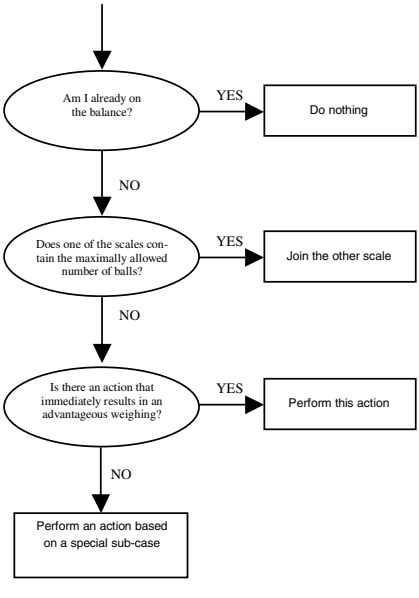

Fig. 1. Action selection algorithm

5 Testing Strategy Performance

The system described in the previous sections has been used to run a complete set of simulation experiments for six balls. The different parameters used in the experiments were the strategy used by each agent (i.e., either non-repetitive or information-eager), the name of the deviating ball, and the type of the deviating ball (i.e., either lighter or heavier). Hence, 768 (= 2^6 * 6 * 2) experiments have been performed in total. In all experiments, the sequential protocol was used for performance of actions. Similar experiments can be performed with student implementations to determine how well their strategies perform.

After performing the experiments, the resulting traces were automatically translated to a format that is suitable for the LEADSTO environment [2]. This environment has an automated checker, which offers the possibility to formally verify dynamic properties against traces. This checker takes a formally specified property and a set of traces as input, and verifies whether the property holds for the traces. For formal specification of the properties, the Temporal Trace Language TTL was used, cf. [12]. This language is based on sorted first-order predicate logic, and allows making explicit references to time points and traces. Using the automated checker, relevant properties can be checked against traces generated by a particular implementation of the distributed weighing problem. Hence, using such tools enables automatic performance measurement of the strategies that have been implemented (e.g. by students). Some examples of such properties are the following (both in informal and in formal TTL notation):

reasoning_successfulness(γ:trace)
In trace γ, eventually there will be a time point t on which some ball knows the solution. Formalisation:

∃t:time ∃b1,b2:ball ∃ v:value
[state(γ, t) |= belief(b1, has_value(b2, v), pos) ∧ state(γ, t) |= deviating_ball(b2, v)]

no_repetition(γ:trace)
In trace γ, no weighing w will be performed twice. Formalisation:

∀t1:time ∀w:weighing
[state(γ, t1) |= to_be_performed(w) ⇒ ¬∃t2:time [t2>t1 ∧ state(γ, t2) |= to_be_performed(w)]]

reasoning_continuation(γ:trace)
In trace γ, as long as there is no ball that knows the solution, a new weighing w will be performed. Formalisation:

∀t1:time ∀b1,b2:ball
[[state(γ, t1) |=/= belief(b1, has_value(b2, heavier), pos) ∧
state(γ, t1) |=/= belief(b1, has_value(b2, lighter), pos)] ⇒
∃t2:time ∃w:weighing [t2>t1 ∧ state(γ, t2) |= to_be_performed(w)]]

strong_new_information(γ:trace)
In trace γ, after each observation result, each ball will update the agent model of at least one of the balls before the next observation result is available. Formalisation:

∀t1:time ∀o1:obs_result ∀b1:ball
[state(γ, t1) |= observation_result(o1, pos) ⇒
∃t2,t3:time ∃b2:ball ∃v1,v2:value
[t2<t1<t3 ∧ v1≠v2 ∧ state(γ, t2) |= belief(b1, has_value(b2, v1), pos)
∧ state(γ, t3) |= belief(b1, has_value(b2, v2), pos)
∧ ¬[∃t4:time ∃o2:obs_result
t1<t4<t3 ∧ o2≠o1 ∧ state(γ, t4) |= observation_result(o2, pos)]]]

efficiency(γ:trace)
In trace γ a solution is found within 3 weighings. Formalisation:
reasoning_successfulness(γ)∧
∃ t1,t2,t3:time ∃ w1,w2,w3:weighing
[state(γ, t1) |= to_be_performed(w1) ∧ state(γ, t2) |= to_be_performed(w2) ∧
state(γ, t3) |= to_be_performed(w3) ∧ t1<t2<t3 ∧
([(t4:time (w4:weighing
state((, t4) |= to_be_performed(w4) (t4 (t1 (t4 (t2 (t4 (t3]]

A summary of the results of the evaluation of the example strategies introduced in Section 4 can be found in Table 2. The properties are on the vertical axis, whereas three different categories of traces are on the horizontal axis. The cells indicate the percentages of generated traces for which a particular property holds. As can be seen in this table, the experiments in which all agents use the non-repetitive calculation strategy (C-a) of Section 4 were always successful (100%). Moreover, in these traces no repetition of weighings occurs, and the reasoning continues until the solution is known. As could be expected, not all of these traces (66.7%) satisfy the property strong_new_information. The reason for this is that these agents do not care whether they always derive new information, as long as there is no repetition of weighings. As a result, these traces do not always satisfy the property efficiency either. However, remember that the efficiency of the process is not considered as a measure of successful cooperation. On the other hand, the traces where all agents use the information-eager strategy (C-b) as given in Section 4 always derive new information. Of course, these traces are still not always efficient. Furthermore, these traces always satisfy the properties reasoning_successfulness, no_repetition, and reasoning_continuation.

Table 2. Results of the automated checks - percentage of traces for which the property holds

	all agents C-a	all agents C-b	some C-a, some C-b
reasoning successfulness	100	100	74.6
no repetition	100	100	78.8
reasoning continuation	100	100	95.8
strong new information	66.7	100	70.4
efficiency	50.0	33.3	41.7

The most interesting category is the set of "mixed" traces (where some agents used strategy C-a, and some agents used C-b). Table 2 shows that none of the properties succeeded for all of these traces. To be specific, 25.4% of the mixed traces was not successful. In fact, there were two reasons for failure: in 21.2% of the cases the same weighing was repeated forever (i.e., the property no_repetition failed), in 4.2% of the cases the reasoning stopped because an agent could not derive an action at all (i.e., the property reasoning_continuation failed). Closer examination of the unsuccessful traces led to the conclusion that the agent causing the failure was always of type C-b. In addition, the reason of failure had always to do with the agent's assumption that the other agents were also of type C-b. Based on this assumption, a strict information-eager agent can only deal with those situations where it is still possible to derive new information. In case a strict information-eager agent encounters another situation, it can show unpredictable behaviour (e.g., leading to repetition of weighings, or to termination of the reasoning).

Based on the above, it may be concluded that an agent using the non-repetitive strategy of Section 4.1 can successfully cooperate with other agents that use this strategy (although they do not always derive new information) and with agents that use the information-eager strategy of Section 4.2. On the contrary, agents using the information-eager strategy can cooperate with other agents of this strategy, but not always with agents of the non-repetitive strategy. This confirms the predictions about robustness made in Section 2.2. This is an important finding, because it has consequences for the requirements that may be defined when using the problem for educational purposes. For example, when students implement an agent of type C-b, it will not always have to be successful when cooperating with an agent of type C-a.

6 Educational Results

Six groups of each three 3rd year Bachelor students in Artificial Intelligence were given the assignment to implement the distributed weighing problem within 4 weeks time. They were each provided with an implemented external world and had to design and implement the agents representing the balls, including their strategies. The assignment required that their agent should be of type C (i.e., either C-a, C-b or C-c).

The solutions of the groups were tested in different settings, and evaluated using the properties described in Section 5. First the solutions were tested when all agents in the system used the same solution strategy (i.e., implemented by the same group). In this test the systems of three of the six groups only solved the problem in some settings (i.e., not for all possibilities of deviation). This indicates that they must have made some mistake, since it follows from Table 1 that agents should always be robust against agents using the same strategy. The other three groups succeeded in all settings. The second test consisted of using agents designed by different groups in one multi-agent system. Here, again in some cases no solution was found. In a subset of these cases this was to be expected. For example, when a C-b agent tried to cooperate with a C-a agent (see again Table 1). However, there were also some failures in cases where agents of the same type tried to cooperate. The most common types of failure in these cases were: (1) Derivation of multiple actions (e.g., an agent trying to jump on both scales at the same time); (2) No derivation of an action at all. The experiments were shown to all groups in a joint session. The students found it difficult to believe that the others would not follow the same line of reasoning as they did. After letting them explain to one another what kind of strategy they incorporated into their agent, the students understood that the assumptions they had made regarding the strategy of other agents were too strong. This gave them an important insight into the difficulties accompanying cooperation without communications.

Besides evaluating the performance of the strategies, students were also graded for the documentation they had written regarding their agent design and strategy. A standard evaluation form has been developed for this purpose, which comprises elements such as analysis, conceptual design, detailed design and rationale.

7 Discussion and Conclusion

The distributed weighing problem introduced in this paper has been designed with the goal of creating a cooperation problem without communication that is scalable, that is

relatively easy with respect to meta-reasoning required of the agents, and for which it is easy to create a simulated environment.

Other (educational) problems from the field of logic (e.g., the muddy children problem [8]) and from the field of distributed problem solving (e.g., the mutual exclusion problem [6]) do not have all these advantages. The muddy children problem is scalable, and simulating the environment is easy, but the problem is heavy in terms of reasoning. The mutual exclusion problem, on the contrary, is a too easy in terms of reasoning. In the mutual exclusion problem, the parties involved do not have to reason at all about the consequences of their actions. The distributed weighing problem offers a nice alternative, since it requires a bit of reasoning about consequences of actions. However, explicit meta-reasoning (see for example [4]) is unnecessary, because of the assumption that all agents will aim for non-repetitive weighings, and the allowance of suboptimal solutions. Under these circumstances the problem can be solved by agents that operate according to the following: "my move aims for non-repetitive weighings, and I assume other agents do the same".

An example solution to the problem was implemented in an agent-based framework, and rigorously tested for two example strategies of levels believed suitable for educational purposes. The two strategies were discussed and compared. Moreover, a methodology was presented to evaluate the performance of strategies, based on formal validation of properties. Using this methodology, the example strategies put forward were formally validated with respect to a number of desired properties.

The educational use of the problem was promising. The students found the problem interesting and challenging, and were confronted with their own faulty assumptions on other students' reasoning. To be able to design correct strategies for the problem, it turned out to be essential to make adequate assumptions about other agents, and to maintain some model of future possibilities. Therefore, the distributed weighing problem showed to be an appropriate problem for the education of cooperation without communication.

References

1. Bavelas, A. The five squares problem - An instructional aid in group cooperation. *Studies in Personnel Psychology*, 5, 29-38.
2. Bosse, T., Jonker, C.M., Meij, L., van der, and Treur, J. LEADSTO: a Language and Environment for Analysis of Dynamics by SimulaTiOn. *Proc. of the Third German Conference on Multi-Agent System Technologies, MATES 2005*. Lecture Notes in AI, Springer Verlag, 2005 (this volume).
3. Brazier, F.M.T., Jonker, C.M., and Treur, J., Principles of Component-Based Design of Intelligent Agents. *Data and Knowledge Engineering*, vol. 41, 2002, pp. 1-28.
4. Corkill, D., Lesser, V., The use of meta-level control for coordination in a distributed problem solving network. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, Karlsruhe, Germany, August 1983, pp. 748 – 756.
5. Dignum, F., Agent Communication and Cooperative Information Agents. In M. Klusch and L. Kerschberg (eds.) *Cooperative Information Agents IV - The Future of Information Agents in Cyberspace (LNCS-1860)*, Springer-Verlag, 2000, pages 191-207.

6. Dijkstra, E.W. Co-operating Sequential Processes. In: Programming Languages, Genuys, F. (Ed), London, Academic Press, 1965.
7. Doran, J.E., Franklin, S., Jennings, N.R., Norman, T.J., On Cooperation in Multi-Agent Systems, *The Knowledge Engineering Review*, 1997 (3), pp. 309-314.
8. Fagin, R., Halpern, J.Y., Moses, Y., and Vardi, M.Y. *Reasoning About Knowledge*. The MIT Press: Cambridge, MA, 1995.
9. Genesereth, M.R., Ginsberg, M.L., and Rosenschein, J.S., Cooperation Without Communication, *The National Conf. on AI*, Philadelphia, PA., August 1986, pp. 51-57.
10. Goodstein, R.L., Find the penny, *Mathematical Gazette*, December 1945, pp. 227-229.
11. Grossman, H.D., The Twelve-Coin Problem, *Scripta Mathematica*, vol. 11, December 1945, pp. 360-363.
12. Jonker, C.M., Treur, J., and Wijngaards, W.C.A., A Temporal Modelling Environment for Internally Grounded Beliefs, Desires and Intentions. *Cognitive Systems Research Journal*, vol. 4(3), 2003, pp. 191-210.
13. Rosenschein, J., Genesereth, M. Deals among rational agents. In *Proc. of the Ninth Int. Joint Conference on Artificial Intelligence,* LA, California, August 1985, pp. 91-99.
14. Schut M.C., Kentrop M., Leenaarts M., Melis M., and Miller I., APPROACH: Decentralised Rotation Planning for Container Barges. In: Lopez de Mataras R. and Saitta L., editors, *Proceedings of the Sixteenth European Conference on Artificial Intelligence* (ECAI 2004), IOS Press, 2004, pp. 755-759.