Finding a Feasible Solution for a Simple LP Problem using Agents

Arie de Bruin[1], Gerard Kindervater[1], Tjark Vredeveld[2⋆], and Albert Wagelmans[3]

[1] Department of Computer Science, Erasmus University Rotterdam
{adebruin, kindervater}@few.eur.nl
[2] Department of Mathematics and Computer Science, Eindhoven University of Technology
tjark@win.tue.nl
[3] Econometric Institute, Erasmus University Rotterdam
wagelmans@few.eur.nl

Abstract. In this paper we will describe a Multi-Agent System which is capable of finding a feasible solution of a specially structured linear programming problem. Emphasis is given to correctness issues and termination detection.

1 Introduction

In this paper we will describe a Multi-Agent System which is capable of finding a feasible solution of a specially structured linear programming problem. As our objective is to solve a simple LP problem using agents, first, we will indicate what we mean by the notion of an agent. A number of frameworks have been proposed, examples of these are Agent0 [4], KQML [3] and Agent-K [2]. Although different frameworks have quite different assumptions about the nature of agents, the following features seem fundamental: (i) an agent is an active and autonomous object with state, goals, knowledge and actions; an agent takes actions based on its own state and goals, and communicates with the environment by sending messages to other agents; (ii) agents communicate with each other by exchanging knowledge, i.e., agents do not only pass service requests and replies, they also pass knowledge in some declarative form; (iii) an agent is associated with a process, it can interact with other agents and its knowledge state may change; other agents can query the agent to obtain information on a state in the process; (iv) a group of agents typically exhibits some forms of human-like behavior, such as commitments, negotiations and mediations.

In this paper, we will describe an agent based method for finding a feasible solution for a simple linear programming problem. The agents will try to solve, that is, try to find a feasible solution to, the following problem.

$$
(\boldsymbol{P}) \qquad
\begin{array}{rcccl}
\boldsymbol{c_1}^T\boldsymbol{x_1} + \boldsymbol{c_2}^T\boldsymbol{x_2} + \ldots + \boldsymbol{c_K}^T\boldsymbol{x_K} & & & & = b_0 \\
D_1\boldsymbol{x_1} & & & & = \boldsymbol{b_1} \\
& D_2\boldsymbol{x_2} & & & = \boldsymbol{b_2} \\
& & \ddots & & \vdots \\
& & & D_K\boldsymbol{x_K} & = \boldsymbol{b_K},
\end{array}
$$

where
$\boldsymbol{x_i} \in \mathbb{R}_+^{n_i},\ i = 1, \ldots, K$
$b_0 \in \mathbb{R}$
$\boldsymbol{b_i} \in \mathbb{R}^{m_i},\ i = 1, \ldots, K$
$\boldsymbol{c_i} \in \mathbb{R}^{n_i},\ i = 1, \ldots, K$
$\boldsymbol{D_i} \in \mathbb{R}^{m_i \times n_i},\ i = 1, \ldots, K$

⋆ This research was done while the author was with the Department of Computer Science of the Erasmus University Rotterdam and was partially supported by TNO-FEL

The agents are trying to satisfy a single global constraint ($\sum_{i=1}^{K} \boldsymbol{c_i}^T \boldsymbol{x_i} = b_0$), and each of them has to satisfy certain local constraints ($D_i \boldsymbol{x_i} = b_i$). In the remainder of this paper it is assumed that each equation $D_i \boldsymbol{x_i} = b_i$ has a solution.

The set of equations can be seen as a mathematical representation of a planning problem in a decentralized organization. The blocks $D_i \boldsymbol{x_i} = \boldsymbol{b_i}$ ($i = 1, \ldots, K$) could be associated with K divisions of the organization. The vector $\boldsymbol{x_i}$ represents the activities of the division. The local constraints $D_i \boldsymbol{x_i} = \boldsymbol{b_i}$ refer to the constraints of the division, e.g. capacity constraints. The global constraint $\sum_{i=1}^{K} \boldsymbol{c_i}^T \boldsymbol{x_i} = b_0$ refers to the interdependencies between the divisions, e.g. common use of a scarce resource. $\sum_i \boldsymbol{c_i}^T \boldsymbol{x_i}$ can be seen as the value of the resource, and b_0 can be seen as the available amount of the resource.

The most natural way for decomposing the problem $(\boldsymbol{P})$ is to represent every division by an agent. The value of the scarce resource used by an agent is denoted by λ_i. The simplicity of the problem lies in the fact that this value can be represented by a scalar. The problem $(\boldsymbol{P})$ can now be reformulated as:

$$(\boldsymbol{P'}) \qquad \begin{array}{l} \sum_i^K \lambda_i = b_0 \\ \lambda_i \in S_i = \{\lambda : \boldsymbol{c_i}^T \boldsymbol{x_i} = \lambda, D_i \boldsymbol{x_i} = \boldsymbol{b_i}\} \end{array}$$

The region S_i is called the feasible region for λ_i, i.e., the values of the scarce resource ($\boldsymbol{c_i}^T \boldsymbol{x_i}$) for feasible activities $\boldsymbol{x_i}$.

The problem is solved by letting the agents exchange parts of their scarce resource until each agent has a feasible value for it.

Before we start discussing how the agents will communicate and solve the problem, we first describe the model of agents used in this Multi-Agent System.

2 Model of an Agent

As stated in the previous section, the most natural way for decomposing the problem $(\boldsymbol{P})$ is to represent each block $\boldsymbol{x_i}$ by an agent. We also stated that an agent is some object with goals, state, knowledge and actions. Furthermore, an agent is associated with a process and it has some human-like behavior. We will now describe the agents used in this paper in terms of these features.

Goals Agents are trying to satisfy a single global constraint ($\boldsymbol{c_1}^T \boldsymbol{x_1} + \ldots + \boldsymbol{c_K}^T \boldsymbol{x_K} = b_0$), but each of them has to satisfy certain local constraints ($D \boldsymbol{x_i} = \boldsymbol{b_i}$).

State The essential part of the state of an agent can be either of the two values below:

1. *feasible*: if the agent has found a feasible solution to its subproblem ($D_i \boldsymbol{x_i} = \boldsymbol{b_i}$).
2. *not-yet-feasible*: if the agent has not (yet) found a feasible solution to its subproblem.

Knowledge The idea is to let an agent know as little about the other agents as possible. So it should not know, how much the total amount of the scarce resource is and it should not know what the states of the other agents are. What an agent does know, is how much it uses of the scarce resource, so what its value for λ_i is and it also should know what the feasible region, S_i is for its λ_i. It is easy to see that S_i equals the closed interval $[min_i, max_i]$ [1], with

$$min_i = \min\{\boldsymbol{c_i}^T \boldsymbol{x_i} | D_i \boldsymbol{x_i} = \boldsymbol{b_i}\} \tag{1}$$

$$max_i = \max\{\boldsymbol{c_i}^T \boldsymbol{x_i} | D_i \boldsymbol{x_i} = \boldsymbol{b_i}\} \tag{2}$$

[1] Let $\boldsymbol{x_i} = \arg\max\{\boldsymbol{c_i}^T \boldsymbol{x_i} | D_i \boldsymbol{x_i} = \boldsymbol{b_i}\}$ and let $\boldsymbol{y_i} = \arg\min\{\boldsymbol{c_i}^T \boldsymbol{x_i} | D_i \boldsymbol{x_i} = \boldsymbol{b_i}\}$. Choose a $\lambda_i \in [min_i, max_i]$ and define μ as $\lambda_i = min_i + \mu(max_i - min_i)$ and let $\boldsymbol{z_i} = \boldsymbol{x_i} + \mu(\boldsymbol{y_i} - \boldsymbol{x_i})$. Obviously $\boldsymbol{c_i}^T \boldsymbol{z_i} = \lambda_i$ and $D_i \boldsymbol{z_i} = \boldsymbol{b_i}$, so $\lambda_i \in S_i$ and $S_i = [min_i, max_i]$. The reader is invited to adapt this line of reasoning to the cases $min_i = -\infty$ and $max_i = \infty$.

Notice that this only holds if $D_i \boldsymbol{x_i} = \boldsymbol{b_i}$ has a solution. Notice furthermore that $min_i = -\infty$ and $max_i = \infty$ are possible.

Actions Agents will be able to perform two types of actions. The first type is an "internal" action, which does not influence other agents. This type of action includes computing the above defined values min_i and max_i. For any value of λ_i agents will be able to compute a feasible solution for $\boldsymbol{x_i}$ in $\begin{cases} \boldsymbol{c_i}^T \boldsymbol{x_i} = \lambda_i \\ D_i \boldsymbol{x_i} = \boldsymbol{b_i} \end{cases}$, if this solution exists, or to determine that there does not exist a feasible solution.

The second type of actions consists of actions that will influence other agents. These actions are communicative actions and will be discussed in subsequent sections.

Behavior As said in the previous section, a group of agents exhibits some human-like behavior. In our system, every agent is the same, so every agent exhibits the same human-like behavior.

The first is negotiating: agents will negotiate with each other about how much of the scarce resource they may use. Secondly, agents are benevolent: when they receive a request, they will help the other agent within their ability. A third human-like behavior is their commitment: if an agent offers a part of its part of the scarce resource, it will reserve this part until the other agent has accepted or rejected the offer (see, e.g. [5]).

With this model of an agent in mind, we show how the agents will find a feasible solution to the problem, if there exists one. In the following section we describe the base algorithm. In Section 4 we will turn our attention to termination detection, more specifically we will discuss how to detect feasibility/infeasibility. The last section gives conclusions and indicates further research.

3 The base algorithm

In this section we will describe the method the agents use to arrive at a feasible solution if there exists one. The discussion of the previous sections shows that it is sufficient to solve the problem

$$(\boldsymbol{P''}) \qquad \begin{array}{l} \sum_i^K \lambda_i = b_0 \\ \lambda_i \in S_i = [min_i, max_i], \end{array}$$

where $min_i = -\infty$ and $max_i = \infty$ might be possible.

3.1 Transactions

The problem will be solved by the agents through performing (in parallel) a sequence of transactions. A transaction *transaction(i,j)* between agent i and agent j is set up if agent i wants to change its value of λ_i. First of all agent i sends a message to agent j specifying by which amount it wants to change this value.

In the reply to such a request, agent j specifies whether or not it can accept it. As we assume that agents are benevolent, agent j will always (partially) accept the request, if this will make its own situation better (that is, the distance between λ_j and S_j will decrease) or if its own situation stays the same (that is, λ_j remains in S_j). If a request is accepted, the agent will also specify how much of the request is granted.

Notice, that both the request from agent i to agent j and the reply back are part of one transaction, i.e. they constitute a single, uninterruptable action.

We now give the code of the request part of *transaction(i,j)*, i.e. the algorithm executed by agent i. The agent will always ask for the minimum amount it needs to get into its

feasible region. After sending the request, it will wait for the answer and if it receives a positive reply, the agent updates its internal values. See Figure 1.

I1. (a) **if** $\lambda_i < min_i$ **then** $\Delta = min_i - \lambda_i$
(b) **if** $\lambda_i > max_i$ **then** $\Delta = max_i - \lambda_i$
I2. **send** *request* for Δ to j
I3. **receive** *reply*, Δ'
I4. **update** *internal values*:
if *request not rejected* **then**
$\lambda_i = \lambda_i + \Delta'$
if $\lambda_i \in S_i$ **then** state$_i$ = *feasible*

Fig. 1. The request part of *transaction(i,j)*

We continue with the code of the reply part of *transaction(i,j)*, i.e. the algorithm executed by agent j. When agent j receives the request, it will evaluate this request and send a reply. A request will either be rejected or (partially) accepted. It is rejected only if accepting the request will increase the distance between λ_j and S_j. If the request is accepted, the return value will be as good as possible, that is, if the request is to decrease λ_j, the decrease is not more than $\lambda_j - min_j$, and if the request is to increase λ_j, the increase is not more than $max_j - \lambda_j$. See Figure 2.

/* a *request* for Δ has been received */
P1. **evaluate:**
(a) **if** $\Delta > 0$ **and** $\lambda_j \leq min_j$ **then reject**
(b) **if** $\Delta < 0$ **and** $\lambda_j \geq max_j$ **then reject**
(c) **if** $\Delta > 0$ **and** $\lambda_j > min_j$ **then** $\Delta' = \min(\Delta, \lambda_j - min_j)$
(d) **if** $\Delta < 0$ **and** $\lambda_j < max_j$ **then** $\Delta' = \max(\Delta, \lambda_j - max_j)$
P2. **send** *answer*:
if *request rejected* **then** *reply rejected*
else *reply* Δ'
P3. **update** *internal values*:
if *request not rejected* **then**
$\lambda_j = \lambda_j - \Delta'$
if $\lambda_j \in S_j$ **then** state$_j$ = *feasible*

Fig. 2. The reply part of *transaction(i,j)*

Next, we describe the main algorithm. As stated before, this algorithm will be realized by pairs of agents executing the transactions described above. These transactions will be executed in parallel, i.e. different pairs of agents will be active at the same time.

However, because transactions are uninterruptible, and because in a transaction only local values of the participating agents are being used, any execution of this parallel algorithm is equivalent with an interleaving. This enables us to give a much more perspicuous proof of its correctness. The possible interleavings are defined by the algorithm in Figure 3.

A short explanation is in order. The notation is inspired by the UNITY-approach, cf. [1]. The algorithm consists of two parts. First, the agents are initialised. They acquire a λ_i, such that all these values sum to b_0. Furthermore the agents decide whether this value makes their state *feasible* or *not-yet-feasible*, by determining whether $\lambda_i \in [min_i, max_i]$.

The main part of the algorithm is a loop. Each iteration of this loop consists of nondeterministically choosing a pair (i, j) where the state of agent i is *not-yet-feasible*,

$$\textbf{init} < \textbf{for all } i :: \begin{cases} \lambda_i = \textit{anything} | \sum_i \lambda_i = b_0 \\ \text{state}_i = \textit{computestate} \end{cases} >$$
$$\textbf{loop} < \textbf{choose } i, j : state_i = \textit{not-yet-feasible} :: \textit{transaction(i,j)} >$$

Fig. 3. Main algorithm

and executing *transaction(i,j)*. The nondeterminism is supposed to be fair, that is, it will not occur that a *transaction(i,j)* is infinitely long enabled (the state of agent i is *not-yet-feasible*) and never chosen. Notice that this loop will only terminate if all agents reach state *feasible*. In the next subsection we will prove that if there exists a feasible solution, it will be found. On the other hand, as it stands now, if there does not exist a feasible solution to the problem the algorithm will not terminate.

In this paper we will assume that our agents will be able to perform a parallel calculation that is adequately described by the interleavings given by Figure 3. We will not dwell on how the agents implement transactions or how the fairness restrictions are realized.

3.2 Correctness of the base algorithm

Now that we have defined how communication between agents proceeds, we will show that this Multi-Agent System will always find a feasible solution if there exists one. This is stated in the following theorem.

Theorem 1. *If there exists a feasible solution to problem (P''), then the Multi-Agent System as defined above, will find one.*

To prove this theorem, we will first state and prove an invariant and prove some lemma's. The invariant is that the sum of the λ_i's will always be equal to b_0. To see this, first note that, when the system is initialized, the λ_i's are chosen such that $\sum_i^K \lambda_i = b_0$. The only occasions, when a λ_i changes, is when there has been an acceptance of a request. In this case λ_i will increase by some amount Δ and for one j, λ_j will decrease by the same amount, so the sum of these two λ's will stay the same, and $\sum_{i=1}^K \lambda_i$ will remain unchanged.

Let P be an assertion. We call P a *semi-invariant* of a computation if its value cannot change from $true$ to $false$ during that computation. A semi-invariant is therefore an assertion that, once $true$, will remain so forever. The difference with a usual invariant is that it is not necessarily $true$ at the beginning of the computation.

Lemma 1. *The properties $\lambda_i \geq min_i$ and $\lambda_i \leq max_i$ are semi-invariants.*

Proof. λ_i only changes through an accepted request. Suppose that $\lambda_i \geq min_i$ before this request.

Case 1 Agent i is the sender of the request. Then the requested amount is $max_i - \lambda_i$ (Step I1) and by Step P1d we know that for the accepted amount Δ' it is true that $|\Delta'| \leq |\Delta| = \lambda_i - max_i$. Thus the new value of λ_i after the acceptance is $\bar{\lambda}_i = \lambda_i + \Delta' = \lambda_i - |\Delta'| \geq \lambda_i - (\lambda_i - max_i) = max_i \geq min_i$.

Case 2 Agent i is the receiver of the request. If the request was to decrease its λ_i (Step P1c) then the value after the update will be $\bar{\lambda}_i = \lambda_i - \min(\Delta, \lambda_i - min_i) \geq \lambda_i - (\lambda_i - min_i) = min_i$. If the request was to increase its λ_i (Step P1d) then for the new value we have $\bar{\lambda}_i > \lambda_i \geq min_i$.

The proof for the property $\lambda_i \leq max_i$ is analogous. ■

As a corollary of this Lemma, it follows that once an agent has state *feasible*, it will always remain feasible.

Corollary 1. *The property $min_i \leq \lambda_i \leq max_i$ is a semi-invariant.*

Lemma 2. *If there exists a feasible solution to (P''), then for each agent that has state* not-yet-feasible*, there exists an agent that will (partially) accept a request sent by the agent with state* not-yet-feasible*.*

Proof. Suppose agent i has state$_i$ = *not-yet-feasible*, i.e. $\lambda_i \notin S_i$. Suppose $\lambda_i < min_i$, then for some agent j to be able to accept a request from agent i, it must be true that $\lambda_j > min_j$ (Step I1 and Step P1c). Suppose there does not exist such an agent j, then for all agents j it must be true that $\lambda_j \leq min_j$. Thus $b_0 = \sum_i^K \lambda_i < \sum_i^K min_i$, and there does not exist a feasible solution for $\boldsymbol{(P'')}$. ■

Notice that, due to the fairness of the choice of pairs (i, j) in our main algorithm (Figure 3) we obtain the following

Corollary 2. *If there exists a feasible solution to (P''), then each agent that has state* not-yet-feasible*, will eventually execute a succesful transaction, i.e. a transaction with an accepted request.*

So, now we know that, if there exists a feasible solution, as long as not all agents are feasible, progress will be made. In the sequel we show that such progress will proceed in steps that are sufficiently large to guarantee termination. We will do this by a standard well-foundedness argument. To this end we define the following values:

$$\begin{array}{ll} n_{\text{feas}} & = \text{number of agents in state } \textit{feasible} \\ n_{\text{bound}} & = \text{number of agents with } \lambda_i \in \{min_i, max_i\} \\ d & = \sum_{i=1}^{K} \max(0, min_i - \lambda_i, \lambda_i - max_i) \end{array}$$

d can be seen as the total distance from finding a feasible solution, as for each agent i $\max(0, min_i - \lambda_i, \lambda_i - max_i)$ denotes the distance between λ_i and S_i. Obviously a feasible solution has been found if $d = 0$. Another way to know that a feasible solution is found is when $n_{\text{feas}} = K$.

The next lemma says that in each successful transaction the number of agents in state *feasible* will increase, or else the number of agents with $\lambda_i \in \{min_i, max_i\}$ will increase, or else the distance d will decrease with at least δ, where $\delta = \min\{max_i - min_i | max_i > min_i\}$ is the length of the smallest (non-degenerate) feasible region of the agents.

Lemma 3. *After each successful transaction $(n_{feas}, n_{bound}, -\lceil \frac{d}{\delta} \rceil)$ has lexicographically increased.*

Proof. Case 1 Suppose the transaction was between two agents in state *not-yet-feasible*. Then by Step I1 and Step P1, we know that at least one of the two agents will afterwards be in state *feasible*, so n_{feas} has increased.

Case 2 Suppose the transaction was between an agent i, with state$_i$ = *not-yet-feasible* and an agent j, with state$_j$ = *feasible*. Moreover, suppose that afterwards both agents are in state *feasible*. Then n_{feas} has increased (although it might be that, if $\lambda_j \in \{min_j, max_j\}$, n_{bound} decreases.)

Case 3 Suppose the transaction was between an agent, i, in state *not-yet-feasible*, and an agent, j, in state *feasible* and that afterwards agent i still is in state *not-yet-feasible*. Then, by Step P1, we know that afterwards it must be true that $\lambda_j \in \{min_j, max_j\}$. So, if at the time the transaction was initiated $\lambda_j \notin \{min_j, max_j\}$, then n_{bound} has increased. Otherwise, $d_i = \max(0, min_i - \lambda_i, \lambda_i - max_i)$ has decreased by at least δ, and thus $-\lceil \frac{d}{\delta} \rceil$ has increased. ■

Notice that $(n_{\text{feas}}, n_{\text{bound}}, -\lceil \frac{d}{\delta} \rceil)$ is a triple consisting of three integers. The set of all such triples is bounded from above by $(K, K, 0)$. Therefore this set is well founded. The Lemma above thus guarantees termination.

4 Termination

Earlier, we remarked that the algorithm will terminate if all agents reach state *feasible*. In the previous section we showed that, if the problem has a feasible solution, then this termination is guaranteed. Notice that in that case an agent in isolation does not yet know whether the problem has been solved. A standard algorithm for termination detection (cf. for instance [1], Chapter 9) can be applied to remedy this.

As it stands now, the algorithm will not terminate if there does not exist a feasible solution: at least one agent will not reach state *feasible*, and therefore it will be scheduled for unsuccessful transactions forever. In this section we will expand our algorithm to cater for this.

Suppose problem $(\boldsymbol{P''})$ cannot be solved. Then there are two possibilities. Either, at a certain moment one agent, say agent i, has $\lambda_i < min_i$ and all other agents j have $\lambda_j \leq min_j$, or the dual case holds 'on the other sides of the intervals $[min_i, max_i]$'. The reader is invited to check that these are the only possibilities and that such an overall state will always be reached (the argument is similar to the line of reasoning in Section 3.2.)

Without loss of generality we assume the first option. Because agent i will try all possible *transaction(i,j)*'s in a fair manner, at a certain moment it will have participated in an unsuccessful transaction with all other agents. Let us call this *the first round*. Now it is tempting to let agent i decide that the overall problem is infeasible. However, in general this would not be justified: it is quite well possible that after the unsuccessful *transaction(i,j)*, agent j might have participated in a successful *transaction(k,j)* with the effect that its λ has been incremented to a value $\lambda_j > min_j$ and that agent i is not aware of this.

The only conclusion that agent i can draw from an unsuccessful *transaction(i,j)* is that *at that moment* $\lambda_j \leq min_j$ but from this only the semi-invariant $\lambda_j \leq max_j$ can be inferred (cf. Lemma 1.)

Let us consider the case that agent i, having experienced unsuccessful *transaction(i,k)*'s for all k, now enters *the second round*: suppose it tries a second *transaction(i,j)* which again does not succeed. Again agent i knows that at this moment $\lambda_j \leq min_j$, but now the stronger conclusion can be derived that from now on forever $\lambda_j \leq min_j$. This can be seen as follows. Suppose λ_j changes due to a *transaction(j,k)* in which j is the requester. In such a case λ_j can never become greater than min_j (cf. Steps I1a, P1c and P3.) The value λ_j cannot change in a *transaction(k,j)* in which j is the receiver because such a transaction will always fail. Due to the fact that $\lambda_k \leq max_k$, agent k will request a $\Delta > 0$ and in Step P1a agent j will reject this.

Therefore, if in the second round agent i gets a negative answer from all other agents as well,it knows that $\lambda_k \leq min_k$ for all k. Together with $\lambda_i < min_i$ (our starting point) agent i now knows that $\sum_i^K \lambda_i < \sum_i^K min_i$ and thus that the problem is infeasible.

This line of reasoning is captured in our extended algorithm given in Figure 4 (only the code of the request part of a transaction changes). The variables $state_i$ can now take a third value, viz. *infeasible*, indicating that agent i has detected infeasibility. Each agent needs some auxiliary variables $\text{rej}_i[j]$ for bookkeeping. Initially all $\text{rej}_i[j]$ are 0. After an unsuccessful *transaction(i,j)* agent i sets $\text{rej}_i[j] = 1$ (first round), unless agent i is in the second round, indicated by $\text{rej}_i[k] \geq 1$ for all k. In the latter case the assignment $\text{rej}_i[j] = 2$ is performed. As soon as $\text{rej}_i[k] = 2$ for all k, agent i sets $state_i$ to *infeasible*.

All in all we have the following

Theorem 2. *The Multi-Agent System as defined above will terminate. If there exists a feasible solution to the problem* $(\boldsymbol{P})$*, all agents will be in state* feasible. *If there does not exists a feasible solution, there will be at least one agent in state* infeasible.

I1. (a) **if** $\lambda_i < min_i$ **then** $\Delta = min_i - \lambda_i$
(b) **if** $\lambda_i > max_i$ **then** $\Delta = max_i - \lambda_i$
I2. **send** *request* for Δ to j
I3. **receive** *reply*, Δ'
I4. **update** *internal values*:
if *request not rejected* **then**
$\lambda_i = \lambda_i + \Delta'$
if $\lambda_i \in S_i$ **then** state$_i$ = *feasible*
else
(a) **if** $\forall k$: rej$_i[k] \geq 1$ **then**
i. rej$_i[j] = 2$
ii. **if** $\forall k$: rej$_i[k] = 2$ **then** state$_i$ = *infeasible*
(b) **else** rej$_i[j] = 1$

Fig. 4. The request part of a *transaction(i,j)*, second version

Proof. Follows directly from Theorem 1 and the arguments of this section.

5 Conclusions and Future Research

We have shown how to solve a simple linear constraint satisfaction problem using agents. The merit of our approach is that our agents have only limited overall knowledge: they need to know their local constraint ($D_i \boldsymbol{x_i} = b_i$) and their contribution to the global constraint ($\sum_{i=1}^{K} \boldsymbol{c_i}^T \boldsymbol{x_i} = b_0$).

Several extensions come to mind. First there are some technical issues to settle, fairness as well as transactions have to be implemented. More interesting are extensions to the problem, e.g. considering a many-dimensional global constraint (i.e. $b_0 \in \mathbb{R}^n$), or turning the problem into an optimization problem by adding a cost function to be minimized. Finally, adding dynamics to the problem seems interesting, e.g. one could allow b_0 or the number of participating agents to change during the computation.

References

1. K.M. Chandy and J. Misra, Parallel Program Design, Addison-Wesley, Reading, 1988.
2. W. Davies and P. Edwards. Agent-K: an integration of AOP and KQML. In Y. Labrou and T. Finin, editors, *Proceedings of the CIKM '94 Intelligent Information Agents Workshop*, 1994.
3. R. Neches, R. Fikes, T. Finin, T. Gruber, R. Patel, T. Senator, and W.R. Swartout. Enabling technology for knowledge sharing. *AI Magazine*, 12(3):36 – 56, 1991.
4. Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60:51 – 92, 1993.
5. M.J. Wooldridge and N.R. Jennings. Formalizing the cooperative problem solving process. In *Proceedings of the 13th International Workshop on Distributed Artificial Intelligence (IWDAI-94)*, pages 403 – 417, 1994.