Using the Max-Plus Algorithm for Multiagent Decision Making in Coordination Graphs

Jelle R. Kok and Nikos Vlassis

Informatics Institute, University of Amsterdam, The Netherlands
{jellekok, vlassis}@science.uva.nl

Abstract. Coordination graphs offer a tractable framework for cooperative multiagent decision making by decomposing the global payoff function into a sum of local terms. Each agent can in principle select an optimal individual action based on a variable elimination algorithm performed on this graph. This results in optimal behavior for the group, but its worst-case time complexity is exponential in the number of agents, and it can be slow in densely connected graphs. Moreover, variable elimination is not appropriate for real-time systems as it requires that the complete algorithm terminates before a solution can be reported. In this paper, we investigate the max-plus algorithm, an instance of the belief propagation algorithm in Bayesian networks, as an approximate alternative to variable elimination. In this method the agents exchange appropriate payoff messages over the coordination graph, and based on these messages compute their individual actions. We provide empirical evidence that this method converges to the optimal solution for tree-structured graphs (as shown by theory), and that it finds near optimal solutions in graphs with cycles, while being much faster than variable elimination.

1 Introduction

A multiagent system (MAS) consists of a group of agents that interact which each other [1, 2]. In such systems agents act individually, but the outcome can differ based on the behavior of the other agents. In this paper, we concentrate on cooperative MASs in which the agents try to optimize a shared performance measure and have to ensure that their selected individual actions result in good team behavior. RoboCup [3] is a good example of a cooperative (or team) MAS in which the agents also have to deal with time constraints since the soccer-playing robots have to coordinate their actions in real-time in order to win.

A recently introduced framework for multiagent coordination is the concept of coordination graphs (CG) [4], which allows for a tractable representation of the coordination problem. In this framework, payoff functions between subsets of the agents are specified that represent local coordination dependencies between the agents. In order to determine the optimal joint action that maximizes the sum of the local payoffs, a variable elimination (VE) algorithm was proposed in [4]. We applied CGs and VE to our UvA Trilearn RoboCup Simulation team,

A. Bredenfeld et al. (Eds.): RoboCup 2005, LNAI 4020, pp. 1–12, 2006.

which won the RoboCup-2003 World Championship [5]. However, although VE is exact and always produces the optimal joint action, it can be slow in certain cases and in the worst case scales exponentially in the number of agents for densely connected graphs. In previous work [6], we compared two different alternatives to VE. In this paper we will concentrate further on the max-plus algorithm, which is analogous to the belief propagation algorithm [7, 8, 9] for Bayesian networks, as an approximate alternative to VE. In this method, the agents repeatedly exchange payoff messages on which they base their individual action selection. In this paper, we provide empirical evidence that this method converges to the optimal solution for tree-structured graphs and also show that it finds near optimal solutions in densely connected graphs with cycles, while being much faster than VE. These results make this framework interesting for domains as RoboCup where real-time decision making in a group of distributed cooperative agents is of great importance.

2 Coordination Graphs and Variable Elimination

In this section we will review the problem of computing a coordinated action for a group of n agents using the *variable elimination* (VE) algorithm [4]. Each agent chooses an individual action a_i from a set A_i, and the resulting *joint* action $a = (a_1, \ldots, a_n)$ generates a payoff $u(a)$ for the team. The coordination problem is to find the optimal joint action a^* that maximizes $u(a)$, i.e., $a^* = \arg\max_a u(a)$. An obvious approach is to enumerate all possible joint actions and select the one that maximizes $u(a)$. However, this approach quickly becomes impractical, since the joint action space $\times_i A_i$ grows exponentially with the number of agents n.

Fortunately, many problems exhibit the property that the payoff matrix $u(a)$ is sparse. Each agent only depends on a small subset of the other agents, e.g., in robotic soccer only robots that are close to each other have to coordinate their actions. A recent approach to exploit such dependencies involves the use of a coordination graph [4], which decomposes the global payoff function $u(a)$ into a linear combination of local payoff functions, each involving only a few agents. For example, a payoff function involving four agents can be decomposed as follows:

$$u(a) = f_{12}(a_1, a_2) + f_{13}(a_1, a_3) + f_{34}(a_3, a_4). \tag{1}$$

The functions f_{ij} specify the local coordination dependencies between the actions of the agents and can be mapped to a graph $G = (V, E)$ in which each node in V represents an agent, while an edge in E defines a coordination dependency between two agents. Only interconnected agents have to coordinate their actions at any particular instance. The global coordination problem is thus replaced by a number of local coordination problems each involving fewer agents.

In order to find the optimal joint action a^* we can apply VE, which is essentially identical to variable elimination in a Bayesian network [10]. The algorithm operates as follows. One agent is selected and collects all payoff functions in which it is involved from its neighbors. Next, it optimizes its decision conditionally on the possible action combinations of its neighbors and communicates the resulting ‘conditional’ payoff function to one of its neighbors. Thereafter, this agent is eliminated from the graph. When only one agent remains, this agent selects an action that maximizes the final conditional strategy. A second pass in the reverse order is then performed in which every agent computes its optimal action based on its conditional strategy and the fixed actions of its neighbors.

We will illustrate VE on the aforementioned decomposition (1). We first eliminate agent 1. This agent depends on the local payoff functions f_{12} and f_{13} and therefore the maximization of $u(a)$ in (1) can be written as

$$\max_a u(a) = \max_{a_2,a_3,a_4} \Big\{ f_{34}(a_3,a_4) + \max_{a_1}[f_{12}(a_1,a_2) + f_{13}(a_1,a_3)] \Big\}. \qquad (2)$$

Agent 1 defines the function $\phi_{23}(a_2,a_3) = \max_{a_1}[f_{12}(a_1,a_2) + f_{13}(a_1,a_3)]$ and the best-response (conditional strategy) function $B_1(a_2,a_3) = \arg\max_{a_1} [f_{12}(a_1,a_2) + f_{13}(a_1,a_3)]$ which respectively return the maximal value and the associated best action for agent 1 given the actions of agent 2 and 3. The function $\phi_{23}(a_2,a_3)$ is independent of agent 1, which can now be eliminated from the graph, simplifying (2) to $\max_a u(a) = \max_{a_2,a_3,a_4}[f_{34}(a_3,a_4) + \phi_{23}(a_2,a_3)]$. Note that the elimination of agent 1 induces a new dependency between agent 2 and 3 and thus a change in the graph’s topology.

Next, we apply the same procedure to eliminate agent 2. Since only ϕ_{23} depends on agent 2, we define $B_2(a_3) = \arg\max_{a_2} \phi_{23}(a_2,a_3)$ and replace ϕ_{23} by $\phi_3(a_3) = \max_{a_2} \phi_{23}(a_2,a_3)$ producing $\max_a u(a) = \max_{a_3,a_4}[f_{34}(a_3,a_4) + \phi_3(a_3)]$, which is independent of a_2. Next, we eliminate agent 3 giving $\max_a u(a) = \max_{a_4} \phi_4(a_4)$ with $\phi_4(a_4) = \max_{a_3}[f_{34}(a_3,a_4) + \phi_3(a_3)]$. Agent 4 is the last remaining agent and fixes its optimal action $a_4^* = \arg\max_{a_4} \phi_4(a_4)$. Thereafter, a second pass in the reverse elimination order is performed in which each agent computes its optimal (unconditional) action from its best-response function and the fixed actions from its neighbors. In our example, agent 3 first selects $a_3^* = B_3(a_4^*)$. Similarly, we get $a_2^* = B_2(a_3^*)$ and $a_1^* = B_1(a_2^*, a_3^*)$. In the case that one agent has more than one maximizing best-response action, it can select one randomly, since it always communicates its choice to its neighbors.

The outcome of VE is independent of the elimination order and always results in the optimal joint action. On the other hand, the execution time of the algorithm does depend on the elimination order[1] and is exponential in the induced width of the graph (the size of the largest clique computed during node elimination). This can be slow in certain cases and in the worst case scales exponentially in n. Furthermore, VE will only produce its final result after the end of the second pass. This is not always appropriate for real-time multiagent systems where decision making must be done under time constraints. For example, in the RoboCup 2D Simulation League, each agent has to sent an action to the server within 100ms. In these cases, an anytime algorithm that improves the quality of the solution over time would be more appropriate.

[1] Computing the optimal order for minimizing the runtime costs is known to be NP-complete, but good heuristics exists, e.g., minimum deficiency search which first eliminates the agent with the minimum number of neighbors [11].

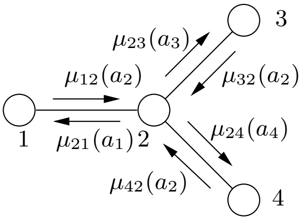


Fig. 1. Graphical representation of different messages μ_{ij} in a graph with four agents

3 The Max-Plus Algorithm

In this section, we describe the *max-plus algorithm* [7, 8, 9, 6] as an approximate alternative to VE. The max-plus algorithm is a popular method for computing the *maximum a posteriori* (MAP) configuration in an (unnormalized) undirected graphical model. This method, analogous to the belief propagation (BP) or sum-product algorithm in Bayesian networks, operates by iteratively sending messages $\mu_{ij}(a_j)$, which can be regarded as locally optimized payoff functions, between agent i and j over the edges of the graph. For tree-structured graphs, the message updates converge to a fixed point after a finite number of iterations [7].

We can translate the above result to our multiagent decision making problem, since computing the MAP configuration is equivalent to finding the optimal joint action in a CG [6]. Suppose that we have a coordination graph $G = (V, E)$ with $|V|$ vertexes and $|E|$ edges. Instead of variables, the nodes in V represent agents, while the global payoff function can be decomposed as follows[2]

$$u(a) = \sum_{i \in V} f_i(a_i) + \sum_{(i,j) \in E} f_{ij}(a_i, a_j). \tag{3}$$

Here f_i denotes a local payoff function for agent i and is only based on its individual action a_i. Furthermore, $(i, j) \in E$ denotes a pair of neighboring agents (an edge in G), and f_{ij} is a local payoff function that maps two actions (a_i, a_j) to a real number $f_{ij}(a_i, a_j)$. A function f_i thus represents the payoff an agent contributes to the system when it acts individually, e.g., dribbling with the ball, and f_{ij} represents the payoff of a coordinated action, e.g., a coordinated pass.

Again, the goal is to find the optimal joint action a^* that maximizes (3). Each agent i (node in G) repeatedly sends a message μ_{ij} to its neighbors $j \in \Gamma(i)$, where μ_{ij} maps an action a_j of agent j to a real number as follows:

$$\mu_{ij}(a_j) = \max_{a_i} \big\{ f_i(a_i) + f_{ij}(a_i, a_j) + \sum_{k \in \Gamma(i) \setminus j} \mu_{ki}(a_i) \big\} + c_{ij}, \tag{4}$$

where $\Gamma(i) \setminus j$ represents all neighbors of i except j and c_{ij} is a normalization vector. This message can be understood as an approximation of the maximum

[2] Note that this function $u(a)$ is analogous to the log-transform of an (unnormalized) factorized probability distribution for which the MAP state is sought.

Algorithm 1. Pseudo-code of the centralized max-plus algorithm

```
centralized max-plus algorithm for $CG = (V, E)$
intialize $\mu_{ji} = \mu_{ij} = 0$ for $(i, j) \in E$, $g_i = 0$ for $i \in V$ and $m = -\infty$
while fixed-point = false and deadline to sent action has not yet arrived do
  // run one iteration
  fixed-point = true
  for every agent $i$ do
    for all neighbors $j = \Gamma(i)$ do
      send $j$ message $\mu_{ij}(a_j) = \max_{a_i} \big\{ f_i(a_i) + f_{ij}(a_i, a_j) + \sum_{k \in \Gamma(i) \setminus j} \mu_{ki}(a_i) \big\} + c_{ij}$
      if $\mu_{ij}(a_j)$ differs from previous message by a small threshold then
        fixed-point = false
    determine $g_i(a_i) = f_i(a_i) + \sum_{j \in \Gamma(i)} \mu_{ji}(a_i)$ and $a'_i = \arg\max_{a_i} g_i(a_i)$
  if use anytime extension then
    if $u((a'_i)) > m$ then
      $(a^*_i) = (a'_i)$ and $m = u((a'_i))$
  else
    $(a^*_i) = (a'_i)$
return $(a^*_i)$
```

payoff i can achieve for a given action of j, and is computed by maximizing (over the actions of i) the sum of the payoff functions f_i and f_{ij} and all incoming messages to i except that from j. The agents keep exchanging messages until they converge. Fig. 1 shows a CG with four agents and the corresponding messages.

A message μ_{ij} in max-plus has three important differences with respect to the conditional strategies in VE. First, before convergence each message is an approximation of the exact value (conditional payoff) since it depends on the incoming (still unconverged) messages. Second, an agent i only has to sum over the received messages from its neighbors defined over its individual actions, instead of enumerating over all possible action combinations of its neighbors (this makes the algorithm tractable). Finally, in VE the elimination of an agent often causes a change in the graph topology. In the max-plus algorithm, messages are always sent over the edges of the original graph.

For trees the messages converge to a fixed-point within a finite number of steps [7, 9]. A message $\mu_{ij}(a_j)$ then equals the payoff the subtree with agent i as root can produce when agent j performs action a_j. If, at convergence, we define

$$g_i(a_i) = f_i(a_i) + \sum_{j \in \Gamma(i)} \mu_{ji}(a_i), \tag{5}$$

then we can show that $g_i(a_i) = \max_{\{a' | a'_i = a_i\}} u(a')$ holds [6]. Each agent i now individually selects its optimal action $a^*_i = \arg\max_{a_i} g_i(a_i)$. If there is only one maximizing action for every agent i, the globally optimal joint action $a^* = \arg\max_a u(a)$ is unique and has elements $a^* = (a^*_i)$. Note that this global optimal joint action is computed by only local optimizations (each node maximizes $g_i(a_i)$ separately). In case the local maximizers are not unique, an optimal joint action can be computed by a dynamic programming technique [9, sec. 3.1].

Algorithm 2. Pseudo-code of a distributed max-plus implementation

```
distributed max-plus for agent $i$, $CG = (V, E)$, spanning tree $ST = (V, S)$
initialize $\mu_{ji} = \mu_{ij} = 0$ for $j \in \Gamma(i)$, $g_i = 0$, $p_i = 0$ and $m = -\infty$
while deadline to sent action has not yet arrived do
  wait for message $msg$
  if $msg = \mu_{ji}(a_i)$ // max-plus message then
    for all neighbors $j = \Gamma(i)$ do
      compute $\mu_{ij}(a_j) = \max_{a_i} \left\{ f_i(a_i) + f_{ij}(a_i, a_j) + \sum_{k \in \Gamma(i) \setminus j} \mu_{ki}(a_i) \right\} + c_{ij}$
      send message $\mu_{ij}(a_j)$ to agent $j$ (if it differs from last sent message)
    if use anytime extension then
      if heuristic indicates global payoff should be evaluated then
        send evaluate( $i$ ) to agent $i$ (me) // initiate computation global payoff
    else
      $a_i^* = \arg\max_{a_i} [f_i(a_i) + \sum_{j \in \Gamma(i)} \mu_{ji}(a_i)]$
  if $msg =$ evaluate( $j$ ) // receive request for evaluation from agent $j$ then
    lock $a_i' = \arg\max_{a_i} [f_i(a_i) + \sum_{j \in \Gamma(i)} \mu_{ji}(a_i)]$ and set $p_i = 0$ if $a_i'$ not locked
    send evaluate( $i$ ) to all neighbors (parent and children) in $ST \neq j$
    if $i =$ leaf in $ST$ then
      send accumulate_payoff( 0 ) to agent $i$ (me) // initiate accumulation payoffs
  if $msg =$ accumulate_payoff( $p_j$ ) from agent $j$ then
    $p_i = p_i + p_j$ // add payoff child $j$
    if received accumulated payoff from all children in $ST$ then
      get actions $a_j'$ from $j = \Gamma(i)$ in CG and set $g_i = f_i(a_i') + \frac{1}{2} \sum_{j \in \Gamma(i)} f_{ij}(a_i', a_j')$
      if $i =$ root of $ST$ then
        send global_payoff( $g_i + p_i$ ) to all children in $ST$
      else
        send accumulate_payoff( $g_i + p_i$ ) to parent in $ST$
  if $msg =$ global_payoff( $g$ ) then
    if $g > m$ then
      $a_i^* = a_i'$ and $m = g$
    send global_payoff( $g$ ) to all children in $ST$ and unlock action $a_i'$
return $a_i^*$
```

Unfortunately, in graphs with cycles there are no guarantees that either max-plus converges or that the local maximizers $a_i^* = \arg\max_{a_i} g_i(a_i)$ correspond to the global optimum. It has been shown that a fixed point of message passings exists [9], but there is no algorithm yet that can provably converge to such a solution. However, bounds are available that characterize the quality of the solution if the algorithm converges [12]. In practice, the max-product algorithm has been successfully applied in graphs with cycles [8, 13, 14].

The max-plus algorithm can both be implemented in a centralized and distributed version. In the distributed implementation, each agent computes and sends updated messages after it has received a new (and different) message from one of its neighbors. In this case, messages are sent in parallel, resulting in a computational advantage over the sequential execution of the centralized algorithm. However, an additional complexity arises since each agent has to individually determine whether the system has converged or when it should report its action. In general we can assume that each agent receives a 'deadline' signal (either from an external source or from an internal synchronized timing signal) after which it must report its action. This, in turn, necessitates the development of an anytime algorithm in which each (local) action is only updated when the corresponding global payoff improves upon the best one found so far. In the centralized version of Alg. 1, we therefore compute the global payoff after every iteration by inserting the current computed joint action into (3). For the distributed case in Alg. 2 the evaluation of the (distributed) joint action is much more complex and is only initiated by an agent when it believes it is worthwhile to do so, e.g., after a big increase in the values of the received messages. This agent starts the propagation of an 'evaluation' over a spanning tree ST of the nodes in G. This tree is fixed beforehand and common knowledge among all agents. An agent receiving an evaluation message fixes its individual action. When an agent is a leaf of ST it also computes its local contribution to the global payoff and sends it to its parent in ST. Each parent accumulates all payoffs of its children and after adding its own contribution sends the result to its parent. Finally, when the root of ST has received all accumulated payoffs from its children, the sum of these payoffs (global payoff) is distributed to all nodes in ST. An agent updates its best individual action a_i^* only when this payoff improves upon the best one found so far. When the 'deadline' signal arrives, each agent thus does not report the action corresponding to the current messages, but the action related to the highest found global payoff. Alg. 1 and Alg. 2 respectively show a centralized and a distributed version in pseudo-code.

4 Experiments

In this section, we describe our experiments with the max-plus algorithm on differently shaped graphs. Since our focus in this paper is on the resulting policies and the corresponding (global) payoff, we used the centralized algorithm from Alg. 1. We first tested it on trees with $|V| = 100$ agents, each having $|A_i| = 4$ actions and a fixed number of neighbors. We created 24 trees in which the number of neighbors per node ranged between $[2, 25]$. Since we fixed the number of agents, each tree had 99 edges but a different depth. Each edge $(i, j) \in E$ was associated with a payoff function f_{ij} where each action combination was assigned a payoff $f_{ij}(a_i, a_j)$ generated from a normal distribution $\mathcal{N}(0, 1)$.

We applied both the VE and max-plus algorithm to compute the joint action. In VE we always eliminated the agent with the minimum number of neighbors, such that each local maximization step involved at most two agents. For the max-plus algorithm, we applied both a random order and the same order as VE to select the agent to sent its messages. Note that in the latter case, the second iteration is the reverse order of the first (comparable to the reversed pass in VE).

Fig. 2 shows the relative payoff found with max-plus with respect to the optimal payoff after each iteration. Results are averaged over all 24 graphs. The policy converges to the optimal solution after a few iterations. When using the elimination order of VE to select the next agent, it always converges after two iterations. For this order, each message only has to be computed once

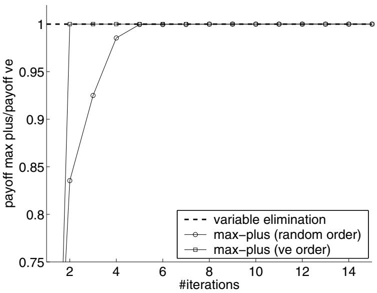


Fig. 2. Average payoff for the max-plus algorithm after each iteration

(see [15]) and the two methods become equivalent. When using a random order, it takes a few iterations before the same information is propagated through the graph.

We also tested max-plus on graphs with cycles. Now, because an outgoing message from agent i can eventually become part of its incoming messages, the values of the messages can become extremely large. Therefore, as in [9], we normalize each sent message by subtracting the average of all values in μ_{ik} using $c_{ij} = \frac{1}{|A_k|}\sum_k \mu_{ik}(a_k)$ in (4). Furthermore, we stopped max-plus after 100 iterations when the messages did not converge (as we will see later in Fig. 4 the policy has stabilized at this point).

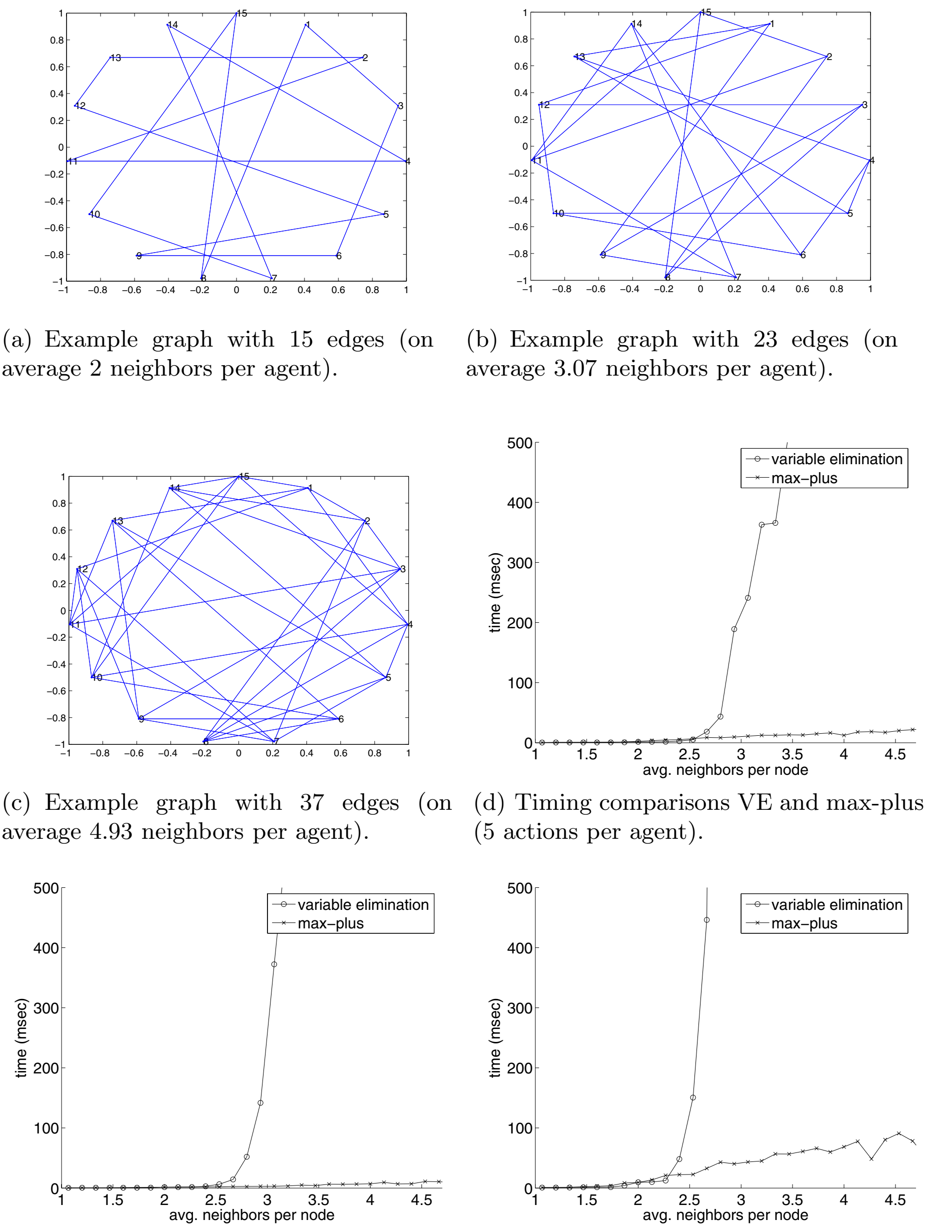
For our experiments, we created graphs with 15 agents, and a varying number of edges. In order to get a balanced graph in which each agent approximately had the same number of neighbors, we randomly added edges between the agents with the minimum number of edges. We generated 100 graphs for each $|E| \in [8, 37]$, resulting in 3000 graphs. Fig. 3(a)-3(c) depict example graphs with respectively 15, 23 and 37 edges (on average 2, 3.07 and 4.93 neighbors per node).

We applied the above procedure to create three test sets. In the first set, each edge $(i,j) \in E$ was associated with a payoff function f_{ij} defined over five actions per agent and each action combination was assigned a random payoff $f_{ij}(a_i, a_j) \in \mathcal{N}(0,1)$. In the second set, we added one outlier to each payoff function: for a randomly picked joint action, the corresponding payoff value was set to $10 * \mathcal{N}(0,1)$. For the third test set, we specified a payoff function using 10 actions per agent and the same method as in the first set to generate the values.

The timing results[3] for the three different test sets[4] are plotted in Fig. 3(d)-3(f). They show that the time for the max-plus algorithm grows linearly as the complexity of the graphs increases (the number of messages is related to the number of edges in the graph). The time for VE grows exponentially since it has to enumerate over an increasing number of neighboring agents in each local maximization step. Furthermore, the elimination of an agent often causes

[3] All results are generated on a 3.4GHz / 2GB machine using a C++ implementation.

[4] For the graphs with ten actions per agent and more than four neighbors per node, VE was not always able to compute a policy since the intermediate computed tables grew too large for the available memory. These graphs were removed from the set.



(a) Example graph with 15 edges (on average 2 neighbors per agent).

(b) Example graph with 23 edges (on average 3.07 neighbors per agent).

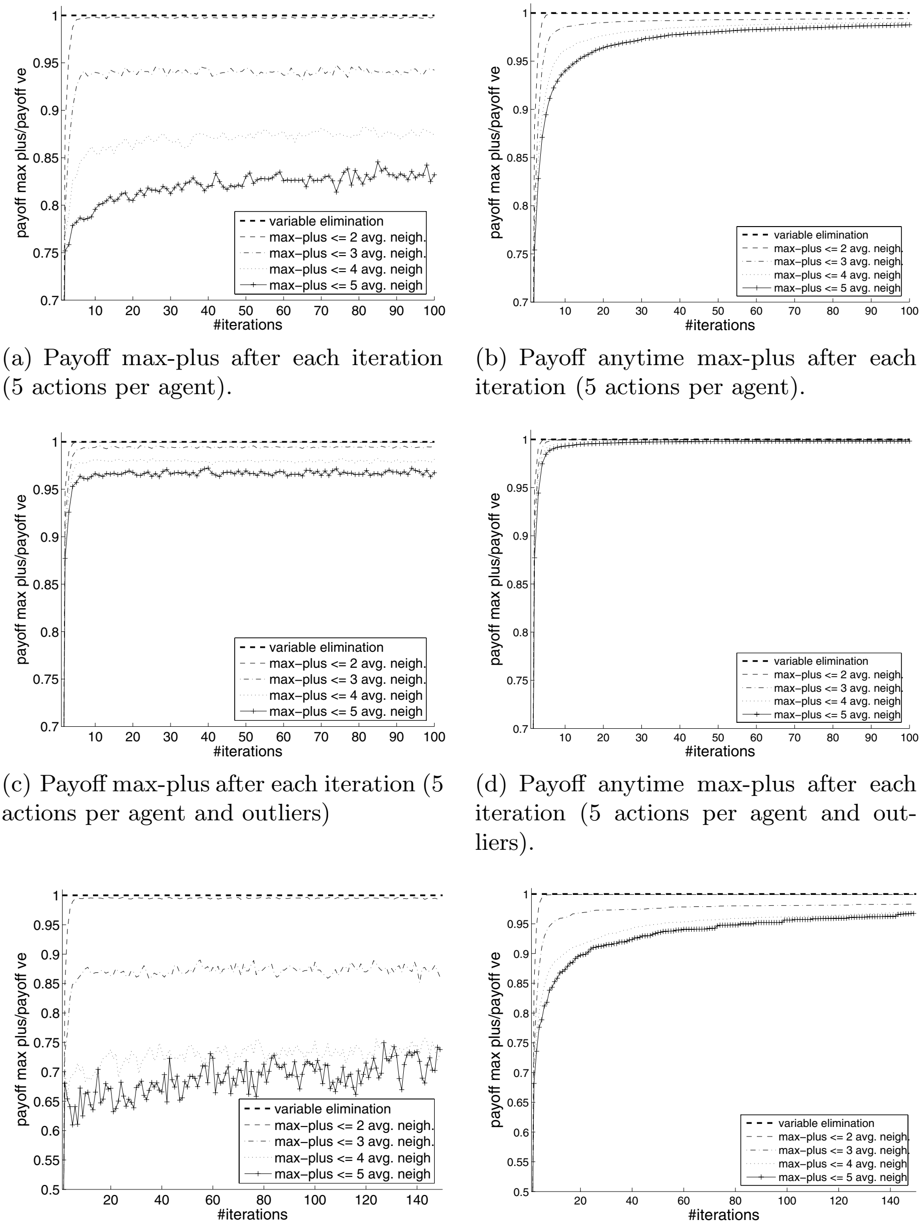
(c) Example graph with 37 edges (on average 4.93 neighbors per agent).

(d) Timing comparisons VE and max-plus (5 actions per agent).

(e) Timing comparisons VE and max-plus (5 actions per agent and outliers).

(f) Timing comparisons VE and max-plus (10 actions per agent).

Fig. 3. Example graphs and (average) timing results for both VE and max-plus for different graphs with 15 agents and cycles



(a) Payoff max-plus after each iteration (5 actions per agent).

(b) Payoff anytime max-plus after each iteration (5 actions per agent).

(c) Payoff max-plus after each iteration (5 actions per agent and outliers)

(d) Payoff anytime max-plus after each iteration (5 actions per agent and outliers).

(e) Payoff max-plus after each iteration (10 actions per agent).

(f) Payoff anytime max-plus after each iteration (10 actions per agent).

Fig. 4. Relative payoff with respect to VE for both max-plus with (graphs on the right) and without (graphs on the left) the anytime extension on different graphs with 15 agents and cycles

a neighboring agent to receive a conditional strategy involving agents it did not had to coordinate before, changing the graph topology to an even denser graph.

Fig. 4 shows the relative payoff found with the max-plus algorithm with respect to the optimal payoff after each iteration for graphs with different average numbers of neighbors. For the loosely connected graphs (less than two neighbors) the result is similar to the optimal result after a few iterations only. As the number of neighbors increases, the resulting policy becomes worse. This effect is less evident in the graphs with outliers (Fig. 4(c)) since certain action combinations are clearly preferred lowering the number of oscillations. Increasing the number of actions per agents (Fig. 4(e)) has a negative influence on the result because of the increase in the total number of action combinations.

Applying the anytime version as discussed in Section 3, improves the results for all graphs indicating that the failing convergence of the messages causes the algorithm to oscillate between different joint actions and 'forget' good joint actions. Fig. 4 shows that for all sets near-optimal policies are found, although it takes more iterations for the graphs with ten actions per agent to find them.

5 Conclusion and Future Directions

In this paper, we continued the work started in [6] and investigated further the usage of the max-plus algorithm as an alternative action selection method to variable elimination (VE) in coordination graphs (CG). VE is an exact method that will always report the optimal joint action, but is slow for densely connected graphs with cycles as its worst-case complexity is exponential in the number of agents. Furthermore, this method is only able to report a solution after the complete algorithm has ended. The max-plus algorithm operates by repeatedly sending local payoff messages over the edges in the CG. By performing a local computation based on its incoming messages, each agent is able to select its individual action. We provided empirical evidence that this method converges to the optimal joint action for tree-structured graphs (as shown by theory), and that it finds near optimal solutions in large, highly connected graphs with cycles an order of magnitude faster than VE. Another advantage of the max-plus algorithm is that it can be implemented fully distributed using asynchronous and parallel message passing. For these reasons, we believe max-plus is an appropriate action selection technique for cooperative real-time systems such as used in RoboCup.

As future research, we are planning to implement the max-plus algorithm in our UvA Trilearn 2D simulation team. In previous years, we used VE for cooperative action selection[5], but computational constraints restricted us in the number of coordination dependencies (see [5]). Using the max-plus algorithm we hope to be able to introduce more specialized fine-grained coordination.

Finally, we like to apply max-plus to sequential decision making. In [16, 4] CGs are used in combination with VE to learn coordinated policies of the agents using

[5] Since the agents in the 2D simulator cannot communicate directly, each agent models the complete algorithm separately using common knowledge assumptions (see [5]).

reinforcement learning. We like to investigate whether the usage of max-plus can help to learn the coordinated behavior for larger groups of agents.

Acknowledgments

This research is supported by PROGRESS, the embedded systems research program of the Dutch organization for Scientific Research NWO, the Dutch Ministry of Economic Affairs and the Technology Foundation STW, project AES 5414.

References

1. Weiss, G., ed.: Multiagent Systems: a Modern Approach to Distributed Artificial Intelligence. MIT Press (1999)
2. Vlassis, N.: A concise introduction to multiagent systems and distributed AI, Informatics Institute, University of Amsterdam (2003)
3. Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawa, E.: RoboCup: The Robot World Cup Initiative. In: Proc. of the IJCAI-95 Workshop on Entertainment and AI/Alife. (1995)
4. Guestrin, C., Koller, D., Parr, R.: Multiagent planning with factored MDPs. In: Advances in Neural Information Processing Systems 14, The MIT Press (2002)
5. Kok, J.R., Spaan, M.T.J., Vlassis, N.: Non-communicative multi-robot coordination in dynamic environments. Robotics and Autonomous Systems **50** (2005) 99–114
6. Vlassis, N., Elhorst, R., Kok, J.R.: Anytime algorithms for multiagent decision making using coordination graphs. In: Proc. of the International Conference on Systems, Man and Cybernetics, The Hague, The Netherlands (2004)
7. Pearl, J.: Probabilistic Reasoning in Intelligent Systems. Morgan Kaufman (1988)
8. Yedidia, J., Freeman, W., Weiss, Y.: Understanding belief propagation and its generalizations. In: Exploring Artificial Intelligence in the New Millennium. Morgan Kaufmann Publishers Inc. (2003) 239–269
9. Wainwright, M., Jaakkola, T., Willsky, A.: Tree consistency and bounds on the performance of the max-product algorithm and its generalizations. Statistics and Computing **14** (2004) 143–166
10. Zhang, N.L., Poole, D.: Exploiting causal independence in bayesian network inference. Journal of Artificial Intelligence Research **5** (1996) 301–328
11. Bertelé, U., Brioschir, F.: Nonserial dynamic programming. Academic Press (1972)
12. Wainwright, M., Jaakkola, T., Willsky, A.: Tree consistency and bounds on the performance of the max-product algorithm and its generalizations. Technical report, P-2554, LIDS-MIT (2002)
13. Crick, C., Pfeffer, A.: Loopy belief propagation as a basis for communication in sensor networks. In: Proc. of the 19th Conference on Uncertainty in AI. (2003)
14. Murphy, K., Weiss, Y., Jordan, M.: Loopy belief propagation for approximate inference: An empirical study. In: Proc. 15th Conf. on Uncertainty in Artificial Intelligence, Stockholm, Sweden (1999)
15. Loeliger, H.A.: An introduction to factor graphs. In: IEEE Signal Proc. Mag. (2004) 28–41
16. Kok, J.R., Vlassis, N.: Sparse Cooperative Q-learning. In Greiner, R., Schuurmans, D., eds.: Proc. of the 21st Int. Conf. on Machine Learning, ACM (2004) 481–488