Using the Simulated Annealing Algorithm for Multiagent Decision Making

Jiang Dawei and Wang Shiyuan

Department of Computer Science and Technology,
Southeast University, P.R.China
jiang203@jlonline.com, desiree_wsy@yahoo.com.cn

Abstract. Coordination, as a key issue in fully cooperative multiagent systems, raises a number of challenges. A crucial one among them is to efficiently find the optimal joint action in an exponential joint action space. Variable elimination offers a viable solution to this problem. Using their algorithm, each agent can choose an optimal individual action resulting in the optimal behavior for the whole agents. However, the worst-case time complexity of this algorithm grows exponentially with the number of agents. Moreover, variable elimination can only report an answer when the whole algorithm terminates. Therefore, it is unsuitable in real-time systems. In this paper, we propose an anytime algorithm, called the simulated annealing algorithm, as an approximation alternative to variable elimination. We empirically show that our algorithm can compute nearly optimal results with a small fraction of the time that variable elimination takes to find the solution to the same coordination problem.

1 Introduction

A multiagent system (MAS) consists of a group of agents that interact with each other [1,2]. Research in MAS aims to provide theories and techniques for agents' behavior management. In this paper, we focus on the fully cooperative MASs in which the agents share a common goal. Examples are a team of robots who play football against another team or a group of agents who plan to build a house. A key aspect in such systems is *Coordination*: the procedure to ensure the individual actions of the agents generate optimal joint decisions for the whole group. RoboCup [3] provides a good platform for comparing and testing different coordination techniques.

To solve the above problem, previous research focuses on the use of game theoretic techniques [4], communication [5,6], social conventions or social lows [7], learning [8,9]. However, all these approaches need to exhaust the whole joint action space whose size grows exponentially with the number of agents. Thus, even in very small settings, they are infeasible.

A recent work to decrease the size of the joint action space uses a *coordination graph* (CG) [10,11,12]. The idea of CG is that in many situations, only a small number of agents need to coordinate their actions while the rest of others can act individually. For example, in robotic soccer, only the ball owner and his surrounding players need to coordinate their actions to perform a pass while others can act individually. So the global joint payoff function, the representation of the global joint coordination dependencies between all agents, can be decomposed into a linear combination of local terms, each of which represents the local coordination dependencies between a small subgroup of the agents. Then each agent employs *variable elimination* (VE) algorithm to select an optimal individual action. The outcome results in optimal behavior for the whole group. However, the worst case time complexity of VE is the same with the aforementioned methods of exhausting all possibilities [13,14]. Moreover, although VE is an exact method which always reports the optimal joint action, it does not return any results until the entire algorithm terminates, which is not suitable for real-time systems. In [14], max-plus (MP) algorithm, which is analogous to the belief propagation algorithm [15] for Bayesian networks, was proposed as an approximate alternative to VE. MP can find optimal solutions for tree-structured coordination graphs and also the near optimal solutions in graphs with cycles, but it restricts each local payoff function involved at most two agents [14,15,16].

In this paper, we propose the *simulated annealing* (SA) algorithm as another approximation to VE. In our algorithm, agents repeatedly start independent tries. In an independent try, each agent tries to maximize the global payoff using his own action, while the actions of the other agents stay the same. If a better solution is found, accept it; otherwise, accept it with a certain probability.

We make the following contributions.

- The time complexity of our algorithm grows polynomially with the number of agents.
- Our algorithm is an anytime algorithm that reports result at any time.
- Our algorithm has no restrictions on the number of agents involved in local payoff functions.
- Experiments show that our algorithm can also find near optimal solution within only a small fraction of the time that VE takes to find the solution of the same coordination problem.

The paper is organized as follows. In section 2, we briefly describe the basic concepts of multiagent coordination problem and the process of finding the optimal joint action by VE and CG. Then we describe our proposed algorithm in section 3. Section 4 experimentally validate the correctness and efficiency of our algorithm, followed by conclusion and future work in section 5.

2 Variable Elimination and Coordination Graphs

In this section, we review the variable elimination (VE) algorithm. In a multiagent system, we have a collection of agents $\boldsymbol{G} = \{G_1, \ldots, G_n\}$[1]. Each agent G_i selects

[1] In this paper, we use upper case letters (*e.g.*, X) to denote random variables, and lower case (*e.g.*, x) to denote their values. We also use boldface to denote vectors of variables (*e.g.*, $\boldsymbol{X}$) or their values ($\boldsymbol{x}$).

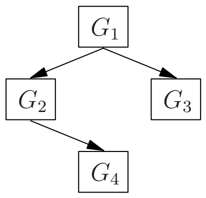


Fig. 1. Initial coordination graph

an individual action a_i from his own action set A_i. Their joint action space thus can be represented as $\boldsymbol{A} = \times_i A_i$. The global payoff function of the agents $\upsilon(\boldsymbol{a})$ maps each joint action $\boldsymbol{a}$ to a real value: $\upsilon(\boldsymbol{a}) \rightarrow \mathbb{R}$. The coordination problem is to find the optimal joint action $\boldsymbol{a}^*$ that maximizes $\upsilon(\boldsymbol{a})$, i.e., $\boldsymbol{a}^* = \text{argmax}_{\boldsymbol{a}} \upsilon(\boldsymbol{a})$. In a naive way, we may consider all possible joint actions and select the one that maximizes $\upsilon(\boldsymbol{a})$. Unfortunately, this approach is infeasible in even the simplest settings, for the number of joint actions grows exponentially with the number of agents (It is called "*curse of dimensionality*" [13]).

This "curse of dimensionality" may be solved by exploiting the structure of the problem to define a compact representation for the global joint payoff function [11,12]. In this way, the global joint payoff function is decomposed into a linear combination of a set of local payoff functions, each of which is only related to a part of system controlled by a small number of agents. For example, in RoboCup, only players that are close to each other have to coordinate their actions to perform a pass or a defend, thus we can use the sum of local payoff functions of subgroup agents to approximate the whole team's payoff. In some situations, this approach can get a very compact representation for coordination dependencies among agents. Furthermore, such representation can be mapped onto a coordination graph $G = (V, E)$ according to the following rules [11]: each agent is mapped to a node in V, and each coordination dependency is mapped to an edge in E. Then the agents can use VE which is identical to variable elimination (or bucket elimination) [17] in a Bayesian network on such CG to determine the optimal joint action.

We show how VE works as follows. Suppose we have 4 agents with each one having 4 different actions, then the number of joint actions is $4^4 = 256$, and global joint payoff function can be decomposed as:

$$\upsilon(\boldsymbol{a}) = \upsilon_1(a_1, a_2) + \upsilon_2(a_2, a_4) + \upsilon_3(a_1, a_3) \tag{1}$$

Fig. 1 shows the initial corresponding coordination graph. The key idea in VE is that, rather than enumerating all possible joint actions and summing up all functions to do maximization, we maximize over variables once at a time. Let us begin with optimization for agent 1. Agent 1 collects all local payoff functions including himself, i.e., υ_1 and υ_3 then does maximization. Hence, we obtain:

$$\text{max}_{\boldsymbol{a}} \upsilon(\boldsymbol{a}) = \text{max}_{a_2,a_3,a_4} \{\upsilon_2(a_2, a_4) + \text{max}_{a_1}[\upsilon_1(a_1, a_2) + \upsilon_3(a_1, a_3)]\} \tag{2}$$

After enumeration of possible action combinations of his neighbors, i.e., agent 2 and agent 3, agent 1 conditionally returns his best response and yield a new function $e_1(a_2, a_3) = \max_{a_1}[v_1(a_1, a_2) + v_3(a_1, a_3)]$ whose value at the point a_2, a_3 is the value of the internal max expression in equation (2). At this time, agent 1 is eliminated from CG. The global joint payoff function is rewritten as:

$$\max_{\boldsymbol{a}} v(\boldsymbol{a}) = \max_{a_2,a_3,a_4}\{v_2(a_2, a_4) + e_1(a_2, a_3)\} \tag{3}$$

Now fewer agents remain. Next, agent 2 does the same procedure. After collecting $v_2(a_2, a_4)$ and $e_1(a_2, a_3)$, agent 2 produces a conditional strategy based on the possible actions of agent 3 and agent 4, and returns his choice, i.e., $e_2(a_3, a_4) = \max_{a_2}[v_2(a_2, a_4)+e_1(a_2, a_3)]$ to the system, then is eliminated. The global payoff function only contains 2 agents now:

$$\max_{\boldsymbol{a}} v(\boldsymbol{a}) = \max_{a_3,a_4}\{e_2(a_3, a_4)\} \tag{4}$$

Agent 3 begins to do optimization. Enumerating actions of agent 4, he reports his own choice and gives a conditional payoff $e_3(a_4) = \max_{a_3} e_2(a_3, a_4)$. Finally, the only remaining agent 4 can simply choose his optimal action: $a_4^* = \operatorname{argmax}_{a_4} e_3(a_4)$.

In the second pass, all agents do the entire process in reverse elimination order. To fulfill agent 4's optimal action a_4^*, agent 3 must select $a_3^* = \operatorname{argmax}_{a_3} e_3(a_4^*)$. Then agent 2 can make a decision $a_2^* = \operatorname{argmax}_{a_2} e_2(a_3^*, a_4^*)$. Finally, agent 1 does $a_1^* = \operatorname{argmax}_{a_1} e_1(a_2^*, a_3^*)$ to choose his optimal action appropriately. The whole procedure needs only $4 \times 4 + 4 \times 4 + 4 = 36$ iterations which is much smaller than 256 iterations of the whole joint action space.

The outcome of VE is independent of the elimination order and always gives the optimal joint action [13]. However, the running speed of VE is depended on the elimination order and exponential in the *induced width* of the coordination graph [11,17]. Finding the optimal elimination order for VE is a well known NP-complete problem [18,19]. Thus, in some cases and especially in the worse case, the time consumed by VE grows exponentially with the number of agents. Furthermore, VE can not give any useful results until the termination of the complete algorithm, therefore it is not suitable for RoboCup 2D simulation league for the robot player has to send actions to server every 100ms. We aim to find an alternative approache that can circumvent such limitations.

3 The Simulated Annealing Algorithm

In many real-world applications, especially in limited computing time cases such as RoboCup, we should make tradeoff between the optimality of the actions and running time. Thus a sub-optimal or nearly optimal solution would be sufficient. In [14], max-plus (MP) algorithm was proposed as an approximation to VE. MP is essentially an instance of Perl's *belief propagation* (BP) algorithm [15] in Bayesian network. It can converge to optimal joint action in tree-structured CGs and find nearly optimal result in graphs with cycles [14,15]. However, MP limits the number of agents in local coordination dependencies not exceeding two.

In this section, we describe the *simulated annealing* (SA) algorithm proposed as an approximate alternative to VE without MP's limitation. The simulated annealing algorithm [2], inspired by statistical mechanics, is very popular for combinatorial optimization [20,21,22]. In this area, efficient techniques are developed to find minimum or maximum values for a function of a number of independent variables [22]. The simulated annealing process executes by "melting" the system being optimized at a high effective temperature at first, and then lowering the temperature by slow stages until the system "freezes" and no further change occurs.

We decide to apply SA to our multiagent decision making problem, since our problem also needs to optimize the global joint payoff function via a number of independent action variables of the agents. The key idea in our approach is rather similar to CG. We decompose the global joint payoff function into a sum of local terms, and then do optimization. Given n agents (defined in section 2), the global joint payoff function can be decomposed as follows:

$$v(\boldsymbol{a}) = \sum_{i \in \boldsymbol{G}} v_i(a_i) + \sum_{i,j \in \boldsymbol{G}} v_{ij}(a_i, a_j) + \sum_{i,j,k \in \boldsymbol{G}} v_{ijk}(a_i, a_j, a_k) + \cdots \qquad (5)$$

Here, $v_i(a_i)$ represents the payoff that an agent contributes to the system when acting individually, e.g., dribbling with the ball. $v_{ij}(a_i, a_j)$ denotes the payoff of a coordination action, e.g., a coordination pass between agent i and agent j, and $v_{ijk}(a_i, a_j, a_k)$, depicts another coordination action involving three agents, e.g., pass from i to j, then j to k. Coordination dependencies with more players can be added if needed. Our decomposition is different from MP in that there is no limitation on the number of robot players involved in local terms. In MP algorithm, the global joint payoff function can only be decomposed into $\sum_{i \in \boldsymbol{G}} v_i(a_i) + \sum_{i,j \in \boldsymbol{G}} v_{ij}(a_i, a_j)$.

Now the goal is to find the optimal joint action, i.e., $\boldsymbol{a}^* = \text{argmax}_{\boldsymbol{a}} v(\boldsymbol{a})$. The pseudo-code of SA algorithm is presented in Alg.1.. The SA algorithm is implemented in a centralized version and performed by the agents in parallel, without assuming the availability of communication. The idea behind it is very straightforward. In each iteration (called an independent try), the algorithm starts with a random choice of joint action for the agents, then loop over all agents. Each agent optimizes the global payoff function with his own action while the actions of all the others stay the same. If the agent's local optimization can yield a better joint action than the initial one, we accept it, otherwise accept the solution with a probability of $\epsilon = \frac{1}{1+e^{-(\Delta/T)}}$. The looping continues until the temperature T decayed from T_{max} to a predefined threshold T_{min}. Then we select a new random starting position and repeat the whole process. When an agent should send action to the server, he returns his own action from the optimal joint action found so far.

Basically, what the SA algorithm does is to seek the global maximum of the global joint payoff function. The SA algorithm has some important differences

[2] The simulated annealing algorithm is also called monte carlo annealing or probabilistic hill-climber.

from VE. Firstly, SA is an anytime algorithm that can report an answer at any time, while VE reports until the whole algorithm terminates. Secondly, in each independent try, agent i only has to iterate his own actions instead of all combinatorial actions of his neighbors, thus makes the algorithm tractable. Finally, the SA is essentially a stochastic algorithm that can not guarantee to find the optimal joint action, [3] while VE is an exact and deterministic algorithm that always report the optimal result. As an approximation algorithm, SA is also different from MP in that SA has no limitation on the form of decomposed local functions while the latter has.

SA has a feature of stochastic movement from one solution to another, which helps it jump away from local maxima and improve the answer's quality [21,22,23]. Although SA can not guarantee the convergence to optimal joint action, we shall see that it can find an approximately optimal solution in a rather short time.

4 Experiments

In this section, we evaluate the simulated annealing algorithm by comparing it with other algorithms, especially with variable elimination. The experiments run in two stages. In the first stage, we fix the number of agents and the number of different actions per agent to test the scalability of the two algorithms when the number of neighbors per agent grows. In the second stage, we compare the relative payoff SA returned with the optimal payoff produced by VE.

Since multiagent system is such a large field that there is no standard problem one can test against, it is important to generate the proper test sets. In this paper, we use a random generator (RG) to produce all test sets. The inputs of the random generator are values of the number of agents $|\boldsymbol{G}|$, the number of different actions per agent $|A|$, maximum number of neighbors per agent Nr_{ne}, and the number of value rules each agent has Nr_{ρ}. We believe that these aspects are sufficient to show the difficulty of the coordination problem. The output of the random generator is a set of value rules, each of which is in the form $\langle \rho : v \rangle$. The value rule is introduced in [11] and proved suitable for plenty of real-world applications such as RoboCup. The global joint payoff function is thus represented by the sum of value rules of all agents. Table 1 depicts a sample output of the random generator (RG) with $|G| = 4$, $|A| = 4$, $Nr_{ne} = 3$, $Nr_{\rho} = 1$.

Here, the integer value of a_i is an action index. In a real RoboCup 2D simulation program, such an action index is finally mapped to a real predefined action (or skill, i.e., dribbling, pass, etc.) and sent to the server. We ignore the details of the specific applications and only focus on the performance of the decision algorithm.

In the first experiment, we generate 120 coordination problems and assign them to 4 test sets based on different actions of each agent. For the problem of each test set, the settings are as follows. The number of the agents is fixed

[3] Technically the SA can also find the optimal solution if the annealing process is very very slow [22]. However this will cause the algorithm to run too long time so that it has no practical use.

Algorithm 1. Pseudo-code of the simulated annealing algorithm

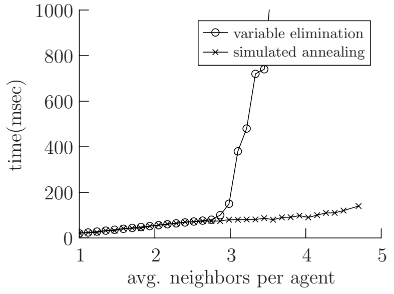
```
Define: $\boldsymbol{G} = \{G_1, \ldots, G_n\}$ the agents who want to coordinate their actions
Define: $v(\boldsymbol{a})$ the global joint payoff function defined by section 3
Define: $\boldsymbol{a}^*$ the optimal joint action found so far
Define: $a_i$ the action of agent $i$
Define: $a_i^*$ the optimal action of agent $i$ found so far
Define: $\boldsymbol{a}_{-i}$ the actions of all agents but agent $i$
  $g \leftarrow 0$
  $t \leftarrow 0$
  while $t < MaxTries$ do
    $\boldsymbol{a}$ = random joint action
    $T \leftarrow T_{max}$
    repeat
      for each agent $i$ in $\boldsymbol{G}$ do
        $\boldsymbol{a'} = \text{argmax}_{a_i} v(\boldsymbol{a}_{-i} \cup a_i)$
        $\Delta \leftarrow v(\boldsymbol{a'}) - v(\boldsymbol{a})$
        if $\Delta > 0$ then
          $\boldsymbol{a} \leftarrow \boldsymbol{a'}$
        else
          $\boldsymbol{a} \leftarrow \boldsymbol{a'}$ with probability $\frac{1}{1+e^{-(\Delta/T)}}$
        end if
        if $v(\boldsymbol{a}) > v(\boldsymbol{a}^*)$ then
          $\boldsymbol{a}^* \leftarrow \boldsymbol{a}$
          $g \leftarrow v(\boldsymbol{a}^*)$
          choose $a_i^*$ from $\boldsymbol{a}^*$
        end if
        if should send action to server then
          send $a_i^*$ to server
        end if
      end for
      $T \leftarrow T \cdot decay$
    until $T < T_{min}$
    $t \leftarrow t + 1$
  end while
```

to $|\boldsymbol{G}| = 15$, while each agent has $Nr_\rho = 8$ value rules with different number of neighbors. The payoff of each value rule is generated from a uniform random variable $v \sim U[1, 10]$. The number of neighbors k in each value rule is in the range $k \in [1,\ Nr_{ne}]$. Each value has a chance of $\binom{Nr_{ne}}{k}/2^{Nr_{ne}}$. All the programs are implemented in C++, and the results are generated on a 2.2GHz/512MB IBM notebook computer.

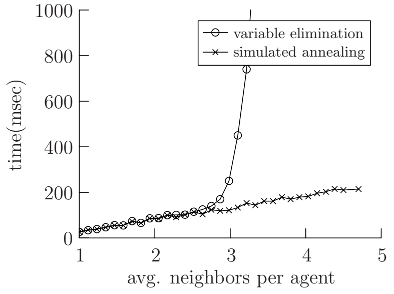
When applying variable elimination algorithm, we accelerate the running time by eliminating the agent with the minimum number of neighbors. When running simulated annealing algorithm, we set $MaxTries$ to 10, the highest temperature $T_{max} = 0.3$, and lowest temperature $T_{min} = 0.05$. The temperature *decay* of the algorithm is in proportion to Nr_{ne}, i.e., $decay \propto Nr_{ne}$. So if the coordination problem contains value rules involving large amounts of agents, we will do a

Table 1. Sample output of RG

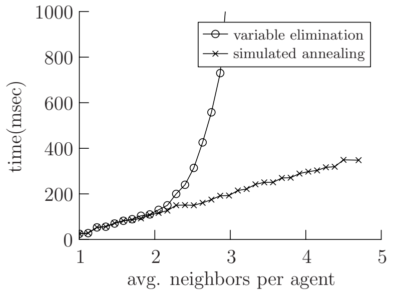
$\langle\, \rho : \upsilon \rangle$
$\langle\; a_1 = 3 \wedge a_3 = 3 \wedge a_4 = 4 : 7.19085 \rangle$
$\langle\; a_2 = 4 \wedge a_3 = 4 : 4.67774 \rangle$
$\langle\; a_1 = 1 \wedge a_2 = 1 \wedge a_3 = 2 \wedge a_4 = 2 : 4.67774 \rangle$
$\langle\; a_1 = 4 \wedge a_3 = 2 \wedge a_4 = 1 : 4.67774 \rangle$



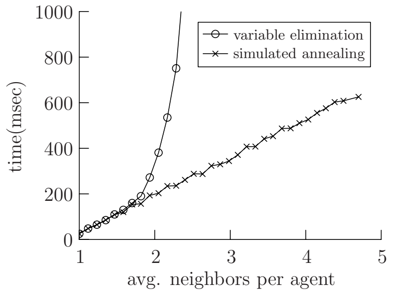
(a) Timing comparisons for VE and SA (4 actions per agent).



(b) Timing comparisons for VE and SA (6 actions per agent).



(c) Timing comparisons for VE and SA (8 actions per agent).

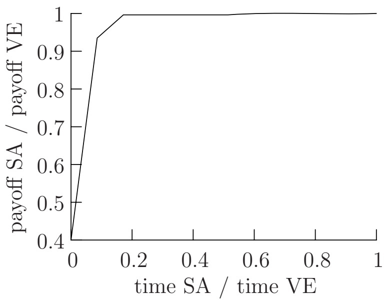


(d) Timing comparisons for VE and SA (10 actions per agent).

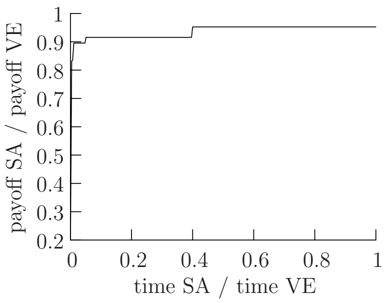
Fig. 2. Average timing comparisons for both VE and SA for testing the scalability when the number of neighbors per agent grows with 15 agents

deep search in an independent try, vice versa. The experiment repeats 10 times to weaken the effect of hardware and operation system.

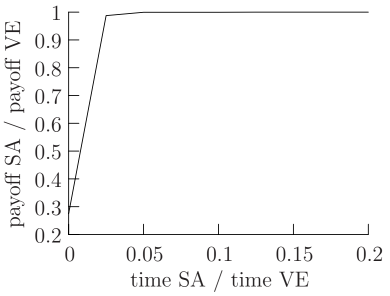
In the second experiment, we produce 6 coordination problems, each of which has its own settings such as number of agents, different actions per agent, etc. VE and SA are both evaluated. When applying SA, instead of starting from a random choice for all agents, in ith independent try, we let the agent select action according to the ith highest value rule if he is involved, otherwise select action randomly. We also set $MaxTries = 200$ to ensure sufficient time to run. Other settings are the same as in the first experiment.



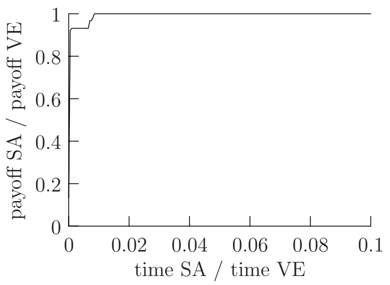
(a) $|\boldsymbol{G}| = 14$, $Nr_{ne} = 2$, $Nr_\rho = 10$, $|A| = 5$.



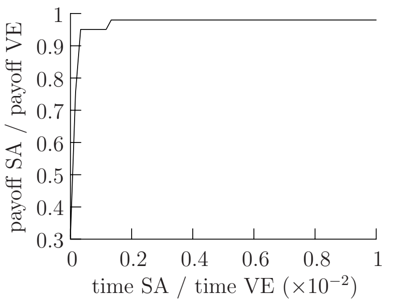
(b) $|\boldsymbol{G}| = 14$, $Nr_{ne} = 2$, $Nr_\rho = 10$, $|A| = 10$.



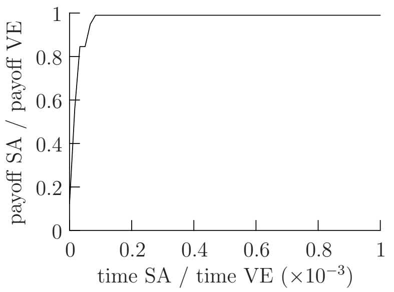
(c) $|\boldsymbol{G}| = 12$, $Nr_{ne} = 4$, $Nr_\rho = 10$, $|A| = 4$.



(d) $|\boldsymbol{G}| = 12$, $Nr_{ne} = 4$, $Nr_\rho = 10$, $|A| = 8$.



(e) $|\boldsymbol{G}| = 10$, $Nr_{ne} = 8$, $Nr_\rho = 10$, $|A| = 4$.



(f) $|\boldsymbol{G}| = 10$, $Nr_{ne} = 8$, $Nr_\rho = 10$, $|A| = 8$.

Fig. 3. Relative payoff found by SA with respect to VE

In order to give a clear image of VE and SA, we scale the payoff axis so that the global maximum payoff is 1. The time axis is also scaled so that the time it takes the whole VE to terminate is 1. Thus the points in the figure can be seen as the fraction of the payoff and the running time of VE. The results of SA will be scaled to its VE companion. Again, the experiment is also repeated 10 times to reduce hardware and software's side effects.

Fig. 2(a)–2(d) give the timing results for the four test sets in the first experiment. It can be seen that the running time of the SA algorithm grows linearly as the number of the neighbors per agent increases. The running time of VE grows exponentially, since it must enumerate all neighbor's possible combination actions in each iteration. Furthermore, when the average number of neighbors per agent was more than 3.5, VE can not always compute the optimal joint action, so these tests were removed from the test sets.

The relative payoff found by the SA with respect to VE are plotted in Fig. 3(a)–3(f). In all the plots, we see that the SA algorithm performed very well. It is obvious that we found approximately optimal results in all problems. In loosely connected coordination problem with few actions, i.e., Fig. 3(a), SA algorithm can converge to the maximum payoff while only using the 60% time of variable elimination [4]. However, if the number of actions is large (Fig. 3(b)), SA can not reach the optimal result, although it can find approximately optimal solution (96% payoff) quickly. Further experiments show that if the joint action space is huge (more than 15 agents, and each agent has more than 10 actions), we should increase the acceptable probability ϵ accordingly to speed up the convergence to optimal result. This is because in such situations, a little higher acceptable probability can increase the chance of stochastic solution movement for simulated annealing algorithm. This technique helps SA jump away from local optimizations and cover the joint action space as possible as it can. But the exact relationship between acceptable probability and the convergency speed are still not very clear. For the medium connected problems (Fig. 3(c)–3(d)), SA can compute the optimal policy with a little fraction of time (2%–6%) that variable elimination needs to solve the same problem. Fig. 3(e) and Fig. 3(f) give us a strong impression that SA can compute above 98% payoff within the time ranges between 0.015% to 0.2% of the time variable elimination takes in the densely connected problems. We also show that in these 2 experiments, although the SA can find near optimal solution very quickly, it still needs to take plenty of time to approximate the optimal result.

In our internal unpublished tests, we also compare SA with max-plus algorithm informally. The experiment shows that when reaching the same relative payoff, the time difference between the two algorithms is at most 5%. Although our algorithm is not faster than max-plus, we believe that our approach is more appropriate for complex coordination problems in which the coordination dependencies in the value rule is often more than two. Thus, max-plus can not be applied directly.

5 Conclusion

In this paper, we have described and investigated the use of the simulated annealing algorithm for cooperative action selection as an approximate alternative to variable elimination algorithm. As above-mentioned, Variable elimination is an exact approach that always reports the optimal joint action. It is also an

[4] Note that SA does not know even the maximum payoff has been found due to its stochastic property.

efficient algorithm in loosely connected coordination graphs. However, it is very slow in densely connected coordination graphs and unable to produce results at anytime. The simulated annealing algorithm repeats independent tries. In each try, each agent tries to maximize the global payoff using his own choice without influencing the actions of all other agents. Based on the result quality in each maximization, the algorithm accepts a solution with a certain probability. We have provided empirical evidences to show: 1) this method is almost optimal with a small fraction of the time that VE takes to compute the policy of the same coordination problem; 2) the running time of SA grows linearly with the increasing of the number of neighbors per agent; 3) it is an anytime algorithm to return result at any time. For above reasons, we believe that simulated annealing is an feasible approach for action selection in large complex cooperative autonomous systems such as RoboCup.

As for future research, we plan to implement the simulated annealing algorithm in our SEU_T 2D simulation team. Last year, we tried to use VE for our player's cooperative action selection framework, but the computational constraints made us only use a small set of value rules with each rule involving at most 3 agents [24]. Applying simulated annealing algorithm, we should produce more advanced coordination actions to involve much more agents.

Finally, we will figure out a appropriate setting of the acceptable probability, especially the decay rate in simulated annealing algorithm. Some recent work shows that neural network algorithm can produce a good decay rate for larger problems [23]. We would like to try to employ such techniques in our multi-agent decision making problem. Furthermore, we want to investigate whether reinforcement learning algorithms can be applied to automatic learning of the payoff instead of hand tuning.

References

1. Weiss, G. (ed.): Multiagent Systems: a Modern Approach to Distributed Artificial Intelligence. MIT Press, Cambridge, MA, USA (1999)
2. Woolridge, M., Wooldridge, M.J.: Introduction to Multiagent Systems. John Wiley & Sons, Inc., New York, NY, USA (2001)
3. Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawa, E.: Robocup: The robot world cup initiative. In: AGENTS '97. Proceedings of the first international conference on Autonomous agents, Marina del Rey, California, United States, pp. 340–347. ACM Press, New York, NY, USA (1997)
4. Osborne, M.J., Rubinstein, A.: A Course in Game Theory. MIT Press, Cambridge (1999)
5. Carriero, N., Gelernter, D.: Linda in context. Communications of the ACM 32(4), 444–458 (1989)
6. Gelernter, D.: Generative communication in Linda. ACM Transactions on Programming Languages and Systems 7(1), 80–112 (1985)
7. Boutilier, C.: Planning, learning and coordination in multiagent decision processes. In: TARK '96. Proceedings of the 6th conference on Theoretical aspects of rationality and knowledge, The Netherlands, pp. 195–210. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1996)

8. Tan, M.: Multi-agent reinforcement learning: Independent vs. cooperative learning. In: Huhns, M.N., Singh, M.P. (eds.) Readings in Agents, pp. 487–494. Morgan Kaufmann, San Francisco, CA, USA (1997)
9. Claus, C., Boutilier, C.: The dynamics of reinforcement learning in cooperative multiagent systems. In: AAAI/IAAI, pp. 746–752 (1998)
10. Guestrin, C., Koller, D., Parr, R.: Multiagent planning with factored MDPs. In: 14th Neural Information Processing Systems (NIPS-14) (2001)
11. Guestrin, C., Venkataraman, S., Koller, D.: Context specific multiagent coordination and planning with factored MDPs. In: The Eighteenth National Conference on Artificial Intelligence (AAAI-2002), Edmonton, Canada, July 2002, pp. 253–259 (2002)
12. Guestrin, C., Koller, D., Parr, R., Venkataraman, S.: Efficient solution algorithms for factored MDPs. Accepted in Journal of Artificial Intelligence Research (JAIR) (2002)
13. Guestrin, C.: Planning Under Uncertainty in Complex Structured Environments. PhD thesis, Stanford University (2003)
14. Kok, J.R., Vlassis, N.: Using the max-plus algorithm for multiagent decision making in coordination graphs. In: Bredenfeld, A., Jacoff, A., Noda, I., Takahashi, Y. (eds.) RoboCup 2005. LNCS (LNAI), vol. 4020, Springer, Heidelberg (2006)
15. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1988)
16. 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, 143–166 (2004)
17. Dechter, R.: Bucket elimination: a unifying framework for reasoning. Artificial Intelligence 113(1-2), 41–85 (1999)
18. Arnborg, S., Corneil, D.G., Proskurowski, A.: Complexity of finding embeddings in a K-tree. SIAM J. Algebraic Discrete Methods 8(2), 277–284 (1987)
19. Bertelé, U., Brioschir, F.: Nonserial dynamic programming. Academic Press, London (1972)
20. Michalewicz, Z., Fogel, D.B.: How to solve it: modern heuristics. Springer, New York, NY, USA (2000)
21. Johnson, D.S., McGeoch, L.A.: The Traveling Salesman Problem: A Case Study in Local Optimization (Draft of November 20, 1995) In: Aarts, E.H.L., Lenstra, J.K. (eds.) To appear as a chapter in The book Local Search in Combinatorial Optimization, John Wiley & Sons, Inc., New York (1995)
22. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science 220(4598), 671–680 (1983)
23. Spears, W.M.: Simulated annealing for hard satisfiability problems. DIMACS Series in Discrete Mathematics and Theoretical Science 26, 533–558 (1996)
24. Dawei, J.: SEU_T 2005 team description (2D). In: Proceedings CD RoboCup 2005, Osaka, Japan, July 2005, Springer, Heidelberg (2005)