TRANSFER LEARNING METHOD APPLIED TO ROBOCUP 2D SIMULATION AGENTS

Luiz A. Celiberto Jr.*, Reinaldo A. C. Bianchi*

*Centro Universitário da FEI
Av. Humberto de Alencar Castelo Branco, 3972.
09850-901 São Bernardo do Campo - SP, Brazil.

Email: celibertojr@fei.edu.br, rbianchi@fei.edu.br

Abstract— This paper describes the design and implementation of robotic agents for the RoboCup Simulation 2D category that learns using a new Transfer Learning algorithm, the SARSA(λ) Transfer Learning with CMAC (SλTL). This algorithm allows the use of case-based reasoning (CBR) to speed up the well-known reinforcement learning algorithm SARSA(λ) with CMAC. Each case correspond to past experience from other agents in similar learning and will influence the choice of the action in a new soccer Simulation 2D game. A set of empirical evaluations was conducted in the RoboCup 2D Simulator, and experimental results show that even a simple case enhances significantly the performance of the agents.

Keywords— Reinforcement Learning, Cognitive Robotics, RoboCup Simulation 2D, Transfer Learning, CMAC.

1 Introduction

The goal of the RoboCup Simulation League is to provide an environment where teams can be created in order to compete against each other in a simulated soccer championship. However, the task is not trivial. Any soccer team must play in harmony and must act as a perfect and dynamic multi-agent system. In the RoboCup Simulation 2D one of the main problems is to find the best action to do when the player have the possession of the ball. Reinforcement Learning is one way to solve this problem, RL provides model-free learning of adequate control strategies, has strong theoretical guarantees on convergence (Szepesvári and Littman, 1996) and also has been successfully applied to solve a wide variety of control and planning problems.

However, one of the main problems with RL algorithms is that they typically suffers from very slow learning rates, requiring a huge number of iterations to converge on a good solution. This problem becomes worse in tasks with high dimensional or continuous state spaces and when the system is given sparse rewards. One of the reasons for the slow learning rates is that most RL algorithms assumes that neither an analytical model nor a sampling model of the problem is available *a priori*, when, in some cases, there is a domain knowledge that could be used to speed up the learning process.

This paper describes a solution to the problem using RL algorithms in high dimensional or continuous state spaces. Describing the design and implementation of robotic agents for the RoboCup Simulation 2D category that learns using a Transfer Learning algorithm, the SARSA(λ) Transfer Learning with CMAC (SλTL). The use of transfer learning can increase significantly the RL algorithm in difficult tasks by allowing agents to generalize their experience across learning problems (Taylor, 2008). The SλTL is a case-based heuristically accelerated extension of the traditional RL algorithm, SARSA (Rummery and Niranjan, 1994).

The paper is organized as follows: Section 2 describes the briefly reviews the RL problem, describes the SARSA(λ) and CMAC algorithm. Section 3 describes the proposed algorithm and section 4 describes the experiment and result. Section 5 concludes this work.

2 Reinforcement Learning, SARSA(λ) algorithm and CMAC

Reinforcement Learning (RL) algorithms have been applied successfully to the on-line learning of optimal control policies in Markov Decision Processes (MDPs). In RL, this policy is learned through trial-and-error interactions of the agent with its environment: on each interaction step the agent senses the current state s of the environment, chooses an action a to perform, executes this action, altering the state s of the environment, and receives a scalar reinforcement signal r (a reward or penalty).

The RL problem can be formulated as a discrete time, finite state, finite action Markov Decision Process (MDP). The learning environment can be modeled by a 4-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, where:

- $\mathcal{S}$: is a finite set of states.
- $\mathcal{A}$: is a finite set of actions that the agent can perform.
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \Pi(\mathcal{S})$: is a state transition function, where $\Pi(\mathcal{S})$ is a probability distribution over $\mathcal{S}$. $T(s, a, s')$ represents the probability of moving from state s to s' by performing action a.

- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \Re$: is a scalar reward function.

The goal of the agent in a RL problem is to learn an optimal policy $\pi^* : \mathcal{S} \rightarrow \mathcal{A}$ that maps the current state s into the most desirable action a to be performed in s.

One strategy to learn the optimal policy π^* is by the SARSA algorithm. This algorithm was proposed by Rummery and Niranjan (Rummery and Niranjan, 1994) and allow the agent to learn the evaluation function $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{R}$. Each action value $Q(s, a)$ represents the expected cost incurred by the agent when taking action a at state s and following an optimal policy thereafter, to help the learning we used the eligibility traces (λ).

The basic mechanism of reinforcement learning are the eligibility traces (Sutton and Barto, 1998). The eligibility define how many times in a recent past one state was visited. The equation 1 allow the reinforcement received update all states recently visited and make the states near more influenced by them.

$$e(s,a) = \sum_{k=1}^{t} (\lambda\gamma)^{(t-k)} \delta_{s,s_k}, \tag{1}$$

where:

- s state being updated,
- a action,
- t current instant,
- λ discount factor ($0 \leq \lambda \leq 1$),
- γ discount factor for future reward ($0 \leq \gamma < 1$),
- δ_{s,s_k} is 1 if $s = s_k$ and 0 otherwise.

One characteristic that helps to compute eligibility is calculating each step running the following algorithm:

$$e_t(s,a) = \begin{cases} \gamma\lambda e_{t-1}(s,a) + 1 & \text{if } s = s_t \text{ and } a = a_t \\ \gamma\lambda e_{t-1}(s,a) & \text{otherwise} \end{cases} \tag{2}$$

One strategy to choose policy of actions widely used is the $\epsilon - Greedy$ exploration. In this kind of exploration the agent will choose the action with the best value of Q with probability $1-\epsilon$ and choose a random action with probability ϵ. The state transition will occur following by the rule:

$$\pi(s_t) = \begin{cases} a_{random} & \text{if } q \leq \epsilon, \\ \arg\max_{a_t} \hat{Q}_t(s_t, a_t) & \text{otherwise,} \end{cases} \tag{3}$$

where:

- q is a random value with uniform probability in [0,1] and ϵ ($0 \leq \epsilon \leq 1$) is the parameter which defines the exploration/exploitation trade-off
- a_{random} is an action randomly chosen among those available in state s_t.

Table 1: The SARSA(λ) algorithm

```
Initialize $\hat{Q}_t(s,a)$ arbitrarily and e(s,a)=0 .
Repeat (for each episode):
  Initialize $s, a$.
  Repeat (for each step of episode):
    Select an action $a$,, observe $r(s,a)$, $s'$.
    Choose $a'$ from $s'$ using policy derived from Q
    $\delta = r + \gamma Q(s',a') - Q(s,a)$
    $e(s,a) = \gamma\lambda e(s,a)$
    For all s,a
      $Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$
      $e(s,a) \leftarrow \gamma\lambda e(s,a)$
    $s \leftarrow s'; a \leftarrow a'$
  Until $s$ is terminal.
Until some stopping criterion is reached.
```

The algorithm is presented in Table 1, using the following update rule is:

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\delta_t e_t(s,a) \tag{4}$$

where:

$$\delta_t = r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t) \tag{5}$$

RL algorithms are very useful for solving a wide variety problems when the model is not known in advance, when the space states is small it can be represented in a table form, but when the space states grows, and became impossible to continue with tabular form. One method to describe the state when have a huge space is by Cerebellar Model Articulation Controller (CMAC)(Albus, 1975). The CMAC was proposed as a functional model of the brain and a function approximation and have been applied since the 80's especially in automatic control.

The CMAC consists of a set of inputs with multidimensional limits finite overlapped called tiles (figure 1). An input vector is contained in a number of tiles overlapped less than the total number of tiles. The tiles are layered axis-parallel tilings over then. More tiles allows better generalization and more tilling allows more accurate representation of smaller details (Taylor, 2008) but the number of the tiles and tillings are generally hand coded.

The value of the CMAC is determined by the tiles excited by a given input and the output is the computed sum:

$$f(x) = \sum^{i} w_i f_i(x) \tag{6}$$

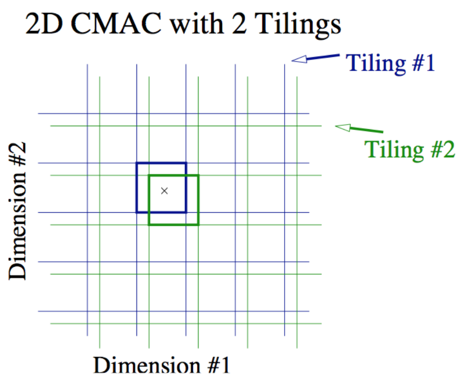

Figure 1: The value of the CMAC is computed summing the weights w_i from each tiles. To determine which tile is activated in each different tilings are used static variable.

where:

- w is the weight of each tilling i,
- $f_i(x)$ is 1 if tile i is active and 0 otherwise

3 The SARSA(λ) Transfer Learning

The SARSA(λ) transfer learning works in two steps. The first step is retrieval of cases and the second step is using this cases to accelerate the RL algorithm. Case Based Reasoning (de Mántaras et al., 2005) is an AI technique that has been shown to be useful in a multitude of domains. CBR uses knowledge of previous situations (cases) to solve new problems, by finding a similar past case and reusing it in the new problem situation.

3.1 Retrieving of Cases

We uses knowledge of previous situations (cases) to solve new problems, by finding a similar past case and reusing it in the new problem situation. Every case is composed of a problem description (P) and the corresponding description of the solution (A). Therefore, the case definition is formally described as a tuple:

$$case = (P, A). \tag{7}$$

This model was proposed by Ros (Ros, 2008), the problem description P corresponds to the situation in which the case can be used. For example, for a Soccer problem the description of a case is the agent position and opponent distance and the solution description A is composed by the action that must perform to solve the problem to pass a ball to another agent.

In theory, more cases in a cases-base means better performance, because will cover all solution, but it is a false assumption. The case based needs the best cases, the cases that solve the problem and this problem is not trivial. To find a partial solution we use RL algorithm in the domains and acquires the cases when the learning stabilizes, for a Q-Learning for example the learning stabilizes when $\hat{Q}(s', a') - \hat{Q}(s, a) \sim 0$, but the number of the cases continues a problem, in practice show the best solution is to acquire sparse (or random) example from the best actions.

3.2 Using this cases to accelerate the RL algorithm - The TL algorithm

Transfer Learning is a very important tool to speed up RL algorithms because, in RL, even a small change on the configuration of a problem may requires a complete new training. With TL, what an agent has learned can be transferred to a new situation, helping it to learn faster. Drummond (Drummond, 2002) was probably the first to use CBR to speed up RL, proposing to accelerate by transferring parts of previously learned solutions to a new problem, exploiting the results of prior learning to speed up the process.

To transfer the cases between learning agents in between domains, we propose a algorithm that expands the SARSA(λ) with CMAC by making use of Transfer Learning, Case-based reasoning and heuristics, the SARSA(λ) Transfer Learning algorithm.

The cases in this algorithm will be the heuristics functions that help to speed up the SARSA(λ) algorithm. The main motivation of using cases as heuristics to transfer the learning is that the heuristic function is an action policy modifier which does not interfere with the standard bootstrap like update mechanism of RL the algorithm: the SλTL differs from the SARSA only in the way exploration is carried out, which allows many theoretical conclusions obtained for the SARSA(λ) to remain valid for the SλTL.

Case retrieval is in general driven by a similarity measure between the new problem and the solved problems in the case base. In this work we use the case retrieval method proposed by Ros (Ros, 2008), which considers the similarity between the problem and the case (the similarity is computed using by Gaussian distance between the case and the problem), the cost of adapting the problem to the case, and the applicability of the solution of the case. The cost of adapting the problem to the case is computed as a function of the euclidean distances between the features (positions of players and ball) in the problem and the ones specified in the case. The complete case retrieval algorithm is described in detail in Ros (Ros, 2008).

After a case is retrieved, a heuristic is computed using Equation 8 and the action suggested by the case is selected using Equation 9 and exe-

Table 2: The SλTL algorithm

```
Initialize $\hat{Q}_t(s,a)$arbitrarily and $H_t(s,a) = 0$
Repeat (for each episode):
   Initialize $s$.
   Repeat (for each step of episode):
     Compute similarity and cost.
     If there is a case that can be reused:
         Retrieve and Adapt if necessary.
         Compute $H_t(s,a)$ using Equation 8
             with the actions suggested
             by the case selected.
         Select an action $a$ using equation 9.
     If don't have a case that can be reused:
      Choose $a'$ from $s'$ using
                  policy derived from Q
     Execute the action $a$, observe $r(s,a)$, $s'$.
     $\delta = r + \gamma Q(s',a') - Q(s,a)$
     $e(s,a) = \gamma\lambda e(s,a)$
     For all s,a
         $Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$
         $e(s,a) \leftarrow \gamma\lambda e(s,a)$
     $s \leftarrow s'; a \leftarrow a'$
    Until $s$ is terminal.
Until some stopping criterion is reached.
```

cuted. If the case base does not contain a case that can be used in the current situation, the SλTL algorithm will behave as the traditional SARSA(λ) algorithm. The complete SλTL algorithm is presented in Table 2.

$$H(s,a) = \begin{cases} \max_i \hat{Q}(s,i) - \hat{Q}(s,a) + \eta \text{ if } a = \pi^H(s), \\ 0 \text{ otherwise.} \end{cases} \tag{8}$$

where η is a small real value (usually 1) and $\pi^H(s)$ is the action suggested by the heuristic policy.

$$\pi(s) = \begin{cases} \arg\max_a \left[\hat{Q}(s,a) + \xi H(s,a)^\beta\right] & \text{if } q \leq p, \\ a_{random} & \text{otherwise,} \end{cases} \tag{9}$$

where $H(s,a)$ is the heuristic function that plays a role in the action choice, ξ and β are design parameters that control the influence of the heuristic function, q and p are parameters that define the exploration/exploitation trade-off and a_{random} is an action randomly chosen among those available in state s.

As a general rule, the value of $H(s,a)$ used should be higher than the variation among the $\hat{Q}(s,a)$ values for the same $s \in \mathcal{S}$, in such a way that it can influence the choice of actions, and it should be as low as possible in order to minimize the error.

4 Experiment in the RoboCup 2D Simulation domain

We used in this work two domains to acquire the cases: the modified Littman's Soccer and domain information. These cases will be used in the Robocup 2D soccer game.

The modified Littman's Soccer domain used in this work is a game played by two teams, A and B, of two players each, which compete in a 10 x 10 grid. Each team is composed by the defender (d) and the attacker (a). Each cell can be occupied by one of the players, which can take an action at a turn. The actions that are allowed are: keep the agent still, move – north, south, east and west – or pass the ball to another agent. The action "pass the ball" from agent a_i to a_j is successful if there is no opponent in between them. If there is an opponent, it will catch the ball and the action will fail.

Actions are taken in turns: all actions from one team's agents are executed at the same instant, and then the opponents actions are executed. The ball is always with one of the players. When a player executes an action that would finish in a cell occupied by the opponent, it looses the ball and stays in the same cell. If an action taken by the agent leads it out the board, the agent stands still. When a player with the ball gets into the opponent's goal, the move ends and its team scores one point. At the beginning of each game, the agents are positioned in a random position and the possession of the ball is randomly determined, with the player that holds the ball making the first move.

In this domain the Q-Learning algorithm is used, with agents continuously playing soccer games. Acquiring cases starts when the learning stabilizes ($\hat{Q}(s',a') - \hat{Q}(s,a) \sim 0$). A cases are acquired by sampling the action-state set randomly every time a player passes the ball successfully to another teammate. At the end of this training, 300 cases are stored in the case base, and every case represents a situation in which a player passes the ball successfully. Each case is described by: The problem description (P), which is composed by the distance of the agent to teammate and opponent near of the player; the action (A), that describes the action the player was doing when passes the ball.

The domain information with use is human knowledge of past knowledge to describe possible cases that can the agents use to score much as possible. This knowledge consists with rule that the agents must follow or should follow to score. This domain information case based is made with two single rules:

- If the agent position is until a 25 meters from the teammate goal, try to passes the ball to another teammate,

- otherwise try to kick the ball to opponent goal if the player position is less than 25 meters from opponent goal.

In this paper, experiment were applied in the RoboCup 2D Soccer Server (Noda, 1995). The Soccer Server is a system that enables autonomous agents programs to play a match of soccer against each other. A match is carried out in a client/server style: a server provides a virtual field and calculates movements of players and a ball, according to several rules. Each client is an autonomous agent that controls movements of one player. Communication between the server and each client is done via TCP/IP sockets. The Soccer Server system works in real-time with discrete time intervals, called cycles. Usually, each cycle takes 100 milliseconds and in the end of cycle, the server executes the actions requested by the clients and update the state of world. We modified a couple of RoboCup simulation soccer default to remove hidden state. We allow players to "see" with a 360° all the field players.

In the Robocup 2D domain we build a team with the SλTL algorithm, this team can take an action at a turn. The actions that are allowed to learn are: hold the ball, pass the ball and kick to goal. Other action allowed, but without learning is: if fastest intercept the ball and hold position otherwise. In the experiments, the implemented teams have to learn while playing against a team composed of agents from the UvA Trilearn 2003 Team (de Boer and Kok, 2002), this team is very simple and only kick the ball to forward.

The CMAC was build with 25 x 25 and made of 16 tilings (the setup utilized was detailed in Singh and Sutton (Singh et al., 1996)). The state representation was defined using a set of variables involving distances and angles between players. This state is derived from information about the position of the players. We use the following 12 states variable:

- D_1: Distance in meters, between the player and the first close teammate,
- D_2: Distance in meters between the player and the second close teammate,
- O_1: Distance in meters between the player and the first close opponent,
- O_2: Distance in meters, between the player and the second close opponent,
- θ_0: Angle in degrees of the player,
- θ_1: Angle in degrees between the player and ball,
- θ_2: Angle in degrees between the player and the first close teammate,
- θ_3: Angle in degrees between the player and the first close opponent,
- X_d: Position discretized of the player in x,
- Y_d: Position discretized of the player in y,
- G_t: Distance discretized between the player and the teammate goal,
- G_o: Distance discretized between the player and the opponent goal.

Table 3: Distance discretized between the player and the goal

Distance	Discretization
5 m and less	0
10 m	1
20 m	2
30 m	3
40 m	4
50 m and more	5

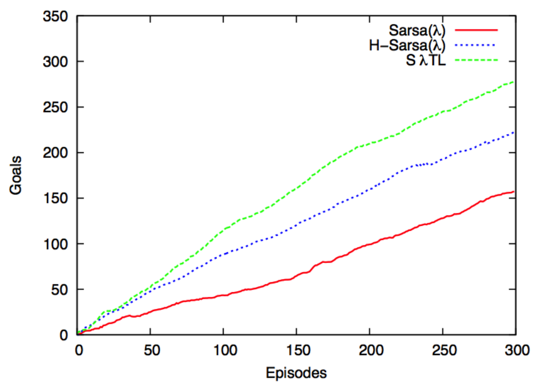

Figure 2: The learning curves for the SARSA(λ), H-SARSA(λ) and SλTL algorithms

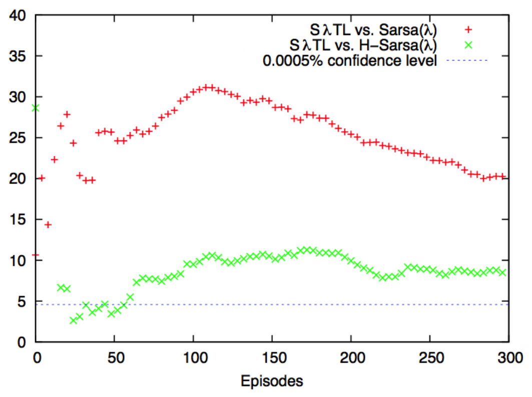

Figure 3: Results from Student's t-Test between SλTL versus SARSA(λ) and SλTL versus H-SARSA(λ) algorithms.

Ten training sessions were executed, with each session consisting of twenty four hours of learning. We used to compare the SARSA(λ), the

H-SARSA(λ) and the algorithm SλTL. The H-SARSA(λ) is the SARSA(λ) with simple heuristic only that we call from the domain information. Figure 2 show the learning curves for all algorithms. It can be seen that the performance of the SARSA(λ) is worst than that of the SλTL. It can also be seen that the SλTL performs better than the H-SARSA(λ), indicating that the speed up was not due only to the use of a simple heuristic. The reward used in the source domain was +100 a goal is made by the player (other teammate receives +10 by the action made) and -100 (to all players) when the team concedes a goal.

Student's t-Test was used to verify the hypothesis that the transfer of learning speeds up the learning process. For the experiments the absolute value of T was computed for each episode using the same data presented in Figure 2. The greater the absolute value of T, more significantly different is the result. The dotted line indicates the 0.0005% confidence limit, i.e. results above the line are different and the probability for this statement to be erroneous is 0.0005%. The result, presented in figures 3 shows that SλTL performs clearly better than SARSA(λ) and H-SARSA(λ). Is important to note in the figure of the t-Test between SλTL versus SARSA(λ) that initial variation happens by the dynamism of the game with only RL.

The parameters used in the experiments were the same for the algorithms: the learning rate is $\alpha = 0.125$, the exploration/ exploitation rate $p = 0.01$ the discount factor $\gamma = 0.9$, $\lambda = 1.0$ and $\eta = 10$. Values in the Q were randomly initiated, with $0 \leq Q(s, a, o) \leq 1$. The experiments were programmed in C++ and executed in a IMAC I7 2.8GHz, with 16GB of RAM in a MAC-*OS* 10.8.3 platform.

5 Conclusion and Future Works

This work proposed the algorithm SλTL. This algorithm combines the use of Case-Based Reasoning to speed up the SARSA(λ) algorithm using Transfer Learning. The contributions of this work is that combine two different cases in the same cased-base, transferring the learning to a multiple agents domain. The experiments showed that transferring the policy learned by the agents in two domain to agents in a different domain by means of the case-base speeds up the convergence time of the algorithm. Future works include incorporating case-based functions in other RL algorithms obtaining results for more complex domains.

Acknowledgments

Luiz Celiberto Jr. and Reinaldo Bianchi acknowledge the support of FAPESP (grants 2012/14010-5, 2011/19280-8, 2012/04089-3 and 2013/17286-4).

References

Albus, J. S. (1975). A new approach to manipulator control: The cerebellar model articulation controller (cmac), *Trans. of the ASME, J. Dynamic Systems, Measurement, and Control* **97**(3): 220–227.

de Boer, R. and Kok, J. (2002). *The Incremental Development of a Synthetic Multi-Agent System: The UvA Trilearn 2001 Robotic Soccer Simulation Team*, Master's Thesis, University of Amsterdam.

de Mántaras, R. L., McSherry, D., Bridge, D., Leake, D., Smyth, B., Craw, S., Faltings, B., Maher, M. L., Cox, M. T., Forbus, K., Keane, M., Aamodt, A. and Watson, I. (2005). Retrieval, reuse, revision and retention in case-based reasoning, *Knowl. Eng. Rev.* **20**(3): 215–240.

Drummond, C. (2002). Accelerating reinforcement learning by composing solutions of automatically identified subtasks, *Journal of Artificial Intelligence Research* **16**: 59–104.

Noda, I. (1995). Soccer server : a simulator of robocup, *Proceedings of AI symposium of the Japanese Society for Artificial Intelligence*, pp. 29–34.

Ros, R. (2008). *Action Selection in Cooperative Robot Soccer using Case-Based Reasoning*, PhD thesis, Universtitat Autònoma de Barcelona.

Rummery, G. and Niranjan, M. (1994). On-line q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166. Cambridge University, Engineering Department.

Singh, S., Sutton, R. S. and Kaelbling, P. (1996). Reinforcement learning with replacing eligibility traces, *Machine Learning*, pp. 123–158.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*, MIT Press.

Szepesvári, C. and Littman, M. L. (1996). Generalized markov decision processes: Dynamic-programming and reinforcement-learning algorithms, *Technical report*, Providence, RI, USA.

Taylor, M. E. (2008). *Autonomous Inter-Task Transfer in Reinforcement Learning Domains*, PhD thesis, Department of Computer Sciences, The University of Texas at Austin.