2007

Decision theoretic agent design for personal rapid transit systems

Iheanyi Umez-Eronini

Ferat Sahin

Decision Theoretic Agent Design for Personal Rapid Transit Systems

Iheanyi C. Umez-Eronini, Ferat Sahin

***Abstract*—This paper details a learning decision-theoretic intelligent agent designed to solve the problem of guiding vehicles in the context of *Personal Rapid Transit* (PRT). The intelligent agents are designed using Bayesian Decision Networks. The agents are designed to utilize the known methods of machine learning with *Bayesian Networks* (BN): parameter learning and structure learning. In addition, a new method of machine learning with BNs, termed utility learning in this paper, is introduced. BN software for Matlab is used to realize the proposed agent. Additional software is written to simulate the PRT problem using various intelligent agents that utilize one or more learning methods.**

***Index Terms*—Bayesian Decision Networks, Machine Learning, Multi-agent systems, Personal Rapid Transit.**

I. Introduction

This paper's goal is to study the design of intelligent agents that operate in a multi-agent environment. An intelligent agent is a machine that is capable of perceiving its environment and acting upon that environment to change it from its current state to a desired state. The ability to learn is important for an agent to change its structure, program, or responses in a manner that is expected to improve its future performance [1]. As the complexity of intelligent agents increase, the agents will need the ability to perform *unsupervised learning*, which is learning without input from human controllers or observers.

The decision theoretic agents (DTA) are designed using Bayesian Decision Networks (BDN). Bayesian Decision Networks are formed by two network structures, Bayesian Networks (BN) and Influence Diagrams (ID) [2]. A BN also known as a *Belief Network* is a probabilistic model encoded by a graphical structure that allows designers to represent variables of interest and the conditional dependencies between them. The strength of BNs over some other probabilistic models is that a BN allows the incorporation of prior knowledge or belief in the model. Influence Diagrams add decision making capabilities to the Bayesian Networks because it enables the agent to decide which action to take. Influence Diagrams apply utility theory to decision networks by allowing a designer to specify how much weight each state of a set of variables has on the available actions that can be taken. Section 3 contains a detailed description of the components of a BDN and the methodology used to arrive at decisions.

There are two methods of learning with BNs; structure learning that allows determining network structure from data and parameter learning that allows determining the conditional probability relationships between network nodes. In this paper, we introduce a method of learning with IDs termed utility learning, developed in [3, 8], where an agent changes the utility function or weighting given to influence variables in order to change the way it makes its decisions.

The decision theoretic agent designed in this paper is applied to the problem of vehicle control in a Personal Rapid Transit (PRT) system. The nature of PRT makes the problem multi-agent in nature since the actions an agent takes have definitive effect on those that other agents can make. A simulator is created to enable testing of the decision theoretic agents and comparison of the performance of utility learning to that of parameter and structure learning. The DTA will be built using the Bayesian Network Toolbox (BNT) for Matlab by Kevin Murphy and the Structure Learning Package (SLP) by Philippe Leray which adds additional structure learning functions including ones that handle learning with incomplete data to the BNT [4].

II. Personal Rapid Transit

Personal Rapid Transit (PRT) is a proposed public transport idea that offers automated on-demand non-stop point-to-point transportation between one's start and destination [5]. Passengers enter and exit the vehicles at stops that are not along the main guide-way allowing the vehicles to always travel at top speed between destinations. The issues involved in implementation of PRT are those of design, construction and cost of guide-ways, spacing between vehicles when traveling at speed, other safety considerations, and the control system of the vehicles (which are assumed to be autonomous) as well as balancing the cost of running all the vehicles on the system versus the actual demand. This can become complex as demand can shift to different parts of the system, as opposed to being balanced across the entire system, and ridership of public transportation tends to fluctuate depending on the time of day, season, or special occasions.

The autonomous vehicles that implement a PRT system must simultaneously perform numerous tasks. The vehicles need to

Manuscript received March 4, 2007.

I. C. Umez-Eronini is with Rochester Institute of Technology, Rochester, NY 14623 USA (phone: 850-590-2232; e-mail: iheanyi.umez.eronini@gmail.com).

II. Ferat Sahin is with Rochester Institute of Technology, Rochester, NY 14623 (phone: 585 475 2175, email: feseee@rit.edu)

be able to plan the most efficient routes along the transit system, maintain occupant safety and comfort by avoiding collisions with other vehicles as well as sudden accelerations or decelerations. In order for the system to function in an on-demand manner throughout any given day, vehicles need to learn ridership patterns and adjust their actions accordingly. A centralized control system that directs each individual vehicle is likely to be of great computational complexity and vulnerable in that failure of the central system could lead to a system-wide shutdown. The individual vehicles will need to be able to pilot themselves as well as plan routes that take into account the movements of other vehicles.

Testing of the intelligent agent is done using a simulator developed to model a simplified version of the PRT problem. This work makes the assumption that the guide-ways allow for travel of only one vehicle at a time. This is similar to having a train system with only one set of tracks that must be used for travel in either direction. As in the PRT concept, the stops will be off the main guide-ways. The vehicles (intelligent agents) will respond to calls and attempt to service them as quickly as possible. The overall control problem is simplified by allowing the vehicles to stop along a guide-way instead of always traveling at full speed. The agents have a finite-range 360-degree sensor capable of detecting other agents and their current direction of travel. Additionally, the agents have a full map of the transit system and always know their precise location on it. The vehicle actions are limited to traveling in one of four directions (north, south, east, or west), or not moving at all. The agent's overall goals are to service calls as quickly as possible without any collisions.

The simulator maintains a static map of the guide-ways and destinations. This map holds information indicating which locations are guide-ways, intersection points, or destination points. The simulator provides functions that generate goals for each agent, a route planner, and data collection from an agent's simulated sensors.

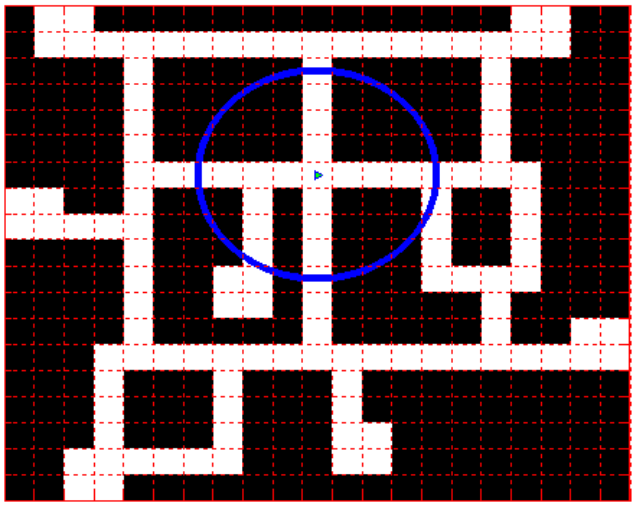

Figure 1: PRT Map Showing Sensor Radius of an Agent

The simulator can also graphically display the map overlaid with a symbol representing an agent at its current location.

This provides a means of qualitatively analyzing the performance of an agent planner as well as rapid development and debugging of simulator code. Fig. 1 shows the PRT system map with a single agent facing east. The circle represents the limits of the agent's sensors. The sensors are modeled such that each intelligent agent can determine the exact locations of any other vehicles detected directly in front of the agent and a sector in the sensor circle where any other agents are detected.

III. Bayesian Decision Networks

A BN is a directed acyclic graph (DAG) that is constructed by a set of variables coupled with a set of directed edges between variables where each set of variables contain a finite set of mutually exclusive states [6]. These variables are known as the chance nodes of the graph. These nodes are not limited to being random variables; they can be known elements or observed entities. Fig 2 shows an example BN-DAG. In a BN, directed edges exist between two variables that are directly related. In terms of probability, there is a conditional dependency between the two variables. For example, in Fig 2, there are conditional dependencies between *A* and *C* and between *B* and *C*. The absence of a directed edge between two variables denotes a conditional independency. So, *A* and *B* are conditionally independent given *C*. So, while it is possible that the events *A* and *B* are not independent, once *C* is known, knowledge about *A* cannot affect the probability of event B.

Associated with each node is a probability table that models the chance of a node being in one of its states. Nodes that have parents are conditionally dependent on at least one other node and thus have associated conditional probability tables (CPT). These tables allow designers to encode prior knowledge or belief. If an agent is unable to observe the state of a variable, it can estimate its belief that the node is in a particular state using the known probability tables and relationships between variables by a process called BN inference. This is an algorithmic process that utilizes Bayes Rule, the Law of Total Probability, and other probability rules to estimate the probable state of unobserved variables.

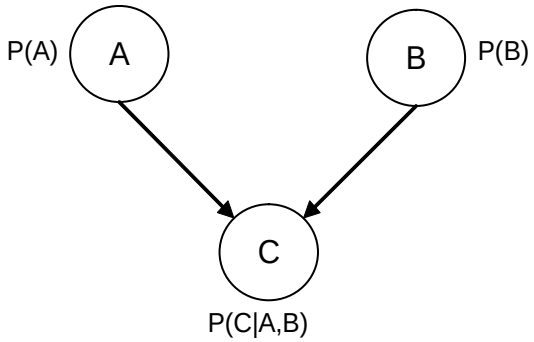

Figure 2: Example Bayesian Network

It is this process that enables an agent to evaluate its belief about its environment and other agents.

Bayesian decision networks extend BNs with IDs. Influence diagrams introduce decision nodes to BNs. Decision nodes are a set of mutually exclusive choices available to the decision maker. Influence Diagrams are a network structure that graphically represents the relationships between decisions nodes, chance nodes, and utility nodes. The utility node is a

function that maps all the possible combinations of decisions and the chance nodes the utility node depends upon to a value representing the desirability of those combinations. The objective of an ID is to choose the decision that maximizes (or minimizes) the value of the utility node.

If the state of one of the chance nodes that the utility node depends upon is unknown or unobserved, then the IDs decision is indeterminate. By combining IDs with BNs, an agent can make decisions in spite of uncertainty by estimating the states of unobserved variables. This is a very powerful tool for designing intelligent agents since it is impossible for an agent to have complete knowledge of its environment's state at any given time. In [20], expected utility is given by (1).

$$EU(A \mid E) = \sum_i P(O_i \mid E, A)U(O_i \mid A) \qquad (1)$$

In (1), E is the available evidence, A is an action with possible outcome state Oi, $U(Oi \mid A)$ is the utility of each of the outcome states, given that action A is taken and $P(Oi \mid E,A)$ is the conditional probability distribution of the possible outcome states in light of the observation of evidence E and that action A is taken [7].

IV. Design of the Intelligent Agents

In this work, the simulator determines, ahead of time, the current route each agent will take in the PRT system. Based on the sensor data, the agent decides whether to move along its route, halt, that is to remain on the route but don't move, or to take a temporary detour from its current route to a temporary new goal location. Agents can also prematurely return to their original path from temporary route. The simulator handles generating the alternate route goal point (at random) and uses its path planning function to reroute the agent.

The first agent developed is the rule-based planner. It serves as a baseline for comparing the performance of the BN based planners for several reasons:

1. It would be the simplest to develop (once a set of rules governing agent behavior have been established).
2. Rule based planners are typically the types of controllers most system designers initial use baring experience or knowledge of more advanced methods.
3. The planner itself is simple, its just made up of a series of if-then statements (with one exception); if the planners developed in this research cannot meet or exceed its performance, then they're not worth the extra time and development effort necessary for their implementation.

To aid in developing the rules for the planner, it is necessary to first establish the system goals. Keeping in line with the real-world concerns of PRT, the absolute goal is that the system maintains safety – agents cannot ever *crash* into each other. In terms of the simulator, that means that the agents cannot occupy the same block at the same time. The second PRT goal is that agents must travel to their goal points as quickly as possible without violating the first. Since the simulator plans the route for each agent to its goal points, the main job of the planner is to ensure that an agent does not have a collision and direct it along its path in such a way that it can travel to its target position as fast as possible. As a result, when developing the rules that govern agent actions in situations where another agent is nearby and could impede its travel, the rules were designed such that as much as possible agents will remain in motion towards their target. A full discussion of the rules can be found in [3, 8].

In developing the rules for the rule-based planner, decisions were made which directly linked a specific agent-environment state to a set of observations. One of the reasons for using BNs is the realization that the observation-state linkages may not always be correct since these determinations were made with limited knowledge of actual system dynamics. A BN, which models the system, allows the designer to over time, generate a statistically accurate set of observation-state linkages. Additionally, BNs allows the designer to incorporate the original belief of the system behavior – effectively what is encoded by the rules developed in the previous section. This means that a BN can capture all the rules developed for the rule based planner and through parameter updating, develop a more accurate association of observations to agent-system states.

The BN agents are formed from the rules used to create the rule-based planner. Nodes are variables that model possible sensor readings to which a PRT agent would have access. The arcs denoting conditional probability relationships were determined by grouping together all of the variables used to implement each rule. The utility table, a weighting of each combination of decision values and influence node states was manually specified to select actions the designer anticipated would achieve the objectives of guiding agents to their target locations as quickly as possible while minimizing collisions.

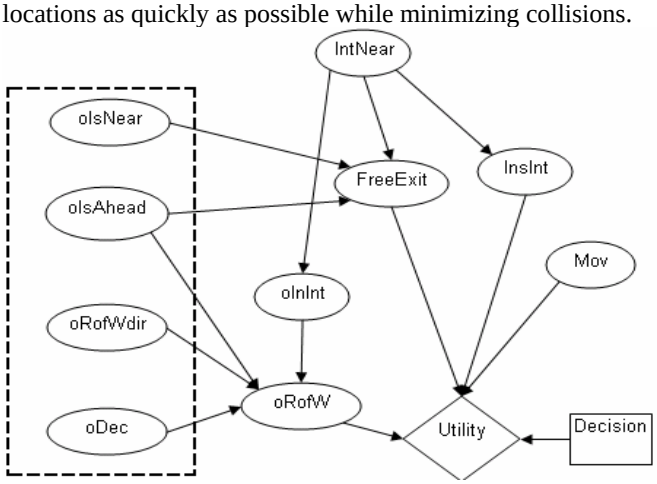

Figure 3: Internal Agent Model of Itself and Its Environment

Each agent maintains a separate BN to model the behavior of other agents in the system, and a single internal BN that models the relationships between the actions that the other agents take, the environment, and agent goals. Fig. 3 shows the internal agent BN. The actual networks created with the BNT do not incorporate the utility and decision nodes. The operation of those nodes is instead carried out in software with functions specifically written for the purpose of this work.

The names shown in Fig. 3 for each node are the ones used in the code written to implement the agents. To simplify the design, each node, with the exception of utility and decision nodes, is a binary node taking on either a true or false value.

The probability tables describe the chances of a node being in either the true or false state given the possible states of the other variables. The node names prefixed with an 'o' indicate variables which depend upon observations of, or predictions about, other agents.

IntNear is a node indicating whether an intersection was detected within an agent's scan (or field of view). *InsInt* indicates whether a particular agent is inside of an intersection. As can be seen from the BDN in Fig. 3, knowledge of being inside an intersection depends upon whether or not there is even one detected in an agent's field of view. *FreeExit* indicates whether there is a potential direction by which an agent that is in a detected intersection could leave it. The value of this node logically depends upon whether there is an intersection detected by an agent and whether other agents are also detected. The *oIsNear* indicates if another agent was detected within a particular scan while *oIsAhd* indicates that not only is an agent detected, but also the detect agent is directly ahead of the one doing the scan. The *Mov* indicates whether the current agent is moving or standing still while *oDec* gives the move or stay decision predicted by BDN modeling the other agent. The *oRofWDir* indicates whether or not the other agent is detected in the sector of the scan corresponding to an agent that would have right-of-way at an intersection. The *oInInt* indicates whether the other agent is in the intersection and *oRofW* indicates whether the other agent actually has right-of-way.

The BN that each agent uses to model the others is almost identical, lacking the *oDec*, and *oIsNear* nodes and replacing the *oIsAhd* node with *AgtNear*. While in the internal agent model, the node values are taken directly from sensor readings, in the other agent BDN, the node values are the predicted sensor readings of the other agent. *AgtNear* predicts whether or not the other agent would itself detect other agents.

Even though *oIsAhd* can be seen to depend on *oIsAhd* both are needed in determining whether or not there is a free exit from an intersection. For example, while *oIsNear* may point to a *potential* block from exiting an intersection, if the *InsInt* value is true, indicating that the current agent is inside an intersection and *oIsAhd* is also true, then there *definitely* is a block from an intersection.

V. Learning with the Intelligent Agents

A. Parameter Learning

Parameter learning is the process by which the CPTs that describe the relationship between a node, its parents and descendents, are updated by the observation of new data. Parameter learning enables an agent to improve its belief of the environment's states. It is desirable that the agent use new observations to refine its belief, rather than using the new observations to redefine its belief. To do this, sample observations are generated from the existing CPT and added to the new observations before executing the parameter learning function.

Parameter updating is done after 500 iterations of the simulator for each agent. The BNT function, learn_params_em is used because it is able to handle situations where data is missing. The function, learn_params_em uses 100 samples from the observed data and existing BN to infer the values of the unobserved parameters. This allows the function to update the CPTs even when data is not observed for a particular node. Prior distribution data is incorporating by generating 100 additional sample observations from the existing network. This ensures the prior distribution carries half as much weight as the recent observations. As a result, the system should be able to react to changes in system parameters while still taking into account prior behavior.

B. Structure Learning

Structure learning is the process where the BN structure, the arcs that denote conditional dependencies between nodes, is learned from data. Structure learning enables an agent to improve its model of the relationships between environment, the internal agent, and other agent states. As with parameter updating, it is desirable that agents use new observations to refine the network structure, rather than defining a new structure based solely on the new observations.

In the agent implementation that utilizes structure updating, the structure learning step is executed once every 100 simulator iterations. It uses the same dataset as the parameter update, the 100 most recent observations plus 100 sample observations from the currently existing network. In order to deal with cases with missing or unobserved data, the learn_struct_em function from the SLP is used.

C. Utility Learning

Parameter and structure learning are tools that have been designed and studied for BNs. These tools allow automated learning of all the components necessary to form a BN model. With a BDN, a couple additional network structures are added: decision and utility nodes. In [3, 8], it is proposed that an additional learning step can be performed for utility nodes. Utility learning is the process where the utility tables that weight the combinations of decisions and influence nodes are updated by the observation of whether or not a selected decision allowed the agent to successfully change the environment from its current state to the desired one.

The utility update function is given by (2). R_{const} is the reinforcement constant. This value is small and positive when the action taken leads to a desired outcome. When the agent's decision does not yield the desired outcome, R_{const} is a large and negative value. Thus, the utility update weakly reinforces *good* decisions and strongly penalizes *bad* ones.

$$U(O_i \mid A)_{new} = U(O_i \mid A)_{old} + R_{const} * P(O_i \mid E) \quad (2)$$

Equation (2) relies on the assumption that incorrect decisions or predictions are made when the expected utility calculated for an incorrectly valued influence node is highest. The expected utility will be highest when $P(Oi|E)$, the probability of outcome *Oi* given evidence *E, is* closest to one. Thus, there is a secondary assumption that incorrect decisions occur when outcome probability for the incorrectly valued influence node is closest to one and outcome probabilities for the other nodes are not. Then, the utility update ensures that when incorrect decisions are made, the utility values for the incorrectly valued

influence nodes decrease by a greater amount than that of the other influence nodes.

VI. Simulation Results

As mentioned earlier, the overall system goals are to maintain safety by not allowing agents to "collide" with each other and for agents to travel to their destinations in the fastest time (or least number of simulator iterations). Keeping in line with the PRT system being modeled, the travel time being recorded is that of an agent from when it has been answered a call to when it finishes servicing that call (picking up and dropping off a passenger). A two-agent system is simulated using the rule based planner, then iterating through each stage of the DTA. Data is collected at the end of 1000 iterations of the simulator. The variables tracking the number of iterations per trip and number of collisions are reset for use in the next simulation run. This set of simulations consisted of five runs for every agent.

Two sets of simulations are run. The initial results, which are show in Table I, are those from a set of 5 consecutive sets of 1000 simulator iterations. An extended simulation was done over 36 consecutive sets of 1000 simulator iterations. These results are summarized in Table II.

The simulation results show that in the area of trips completed and average ticks per trip, the BN based agents all outperform the rule-based agent. Also, the addition of utility updating to an agent with parameter updating shows that BN machine learning methods have a cumulative effect. It is notable that structure learning did not yield much improvement to these agents and in some cases, actually decreased their performance. This could be due to the small amount of data used for structure learning or the possibility that the original agent structures were already close to being optimum. Another result of note is that the rule-based agent had much lower collision numbers than the BN agents. This could arise from a bias by the designer towards intelligent agents that prioritize efficient trip completions over collisions as well as the simple method used to perform the utility update.

VII. Discussion

Utility updating enabled agents to better achieve the goal of increasing the PRT system's throughput. However, the second goal, minimizing collisions, was not achieved. The poor performance of the all BN-based agents with respect to collisions indicates that this is most likely attributable to the agent structure designed in this thesis. Since the implementation of structure updating did not readily lend itself to simulation and did not show significant promise for increasing agent performance in the initial simulations, it is unknown whether structure updating would improve the collision numbers for the BN-based agents.

Excluding agents that utilized structure updating, it is clear that utility updating had a positive albeit very small effect of agent performance. This could be due to the simple method used to perform the utility update as seen in (2), or the way in which R_{const} is chosen. This work indicates that while utility updating has promise as an additional learning tool for intelligent agents, more work is needed to define a methodology for creating utility update functions that can deliver significant results.

References

[1] N. Nilsson, "Introduction to Machine Learning", 2005. [Online]. http://robotics.stanford.edu/people/nilsson/MLDraftBook/mlbook.pdf

[2] F. Sahin and J. S. Bay, "A biological decision-theoretic intelligent agent solution to a herding problem in the context of distributed multi-agent systems", *SMC 2000*, Oct 2002, *IEEE International Conference of Systems, Man, and Cybernetics*, 2000.

[3] I. Umez-Eronini, "Online Structure, Parameter, and Utility Updating of Bayesian Decision Networks for Cooperative Decision Theoretic Agents", M.S. Thesis, Rochester Institute of Technology, Rochester NY, 2007.

[4] K. Murphy. How to Use the Bayes Net Toolbox, 2004. [Online] Available: http://bnt.sourceforge.net/usage.html

[5] Wikipedia, (2006, April). Personal Rapid Transit, [Online] Available: http://en.wikipedia.org/wiki/Personal_rapid_transit

[6] F. Sahin, J. S. Bay, "Learning from Experience Using a Decision-Theoretic Intelligent Agent in Multi-Agent Systems", in Proc. IEEE Mountain Workshop on Soft Computing in Industrial Applications, pp 109-114, 2001.

[7] K. B. Korb, A. E. Nicholson. Bayesian Artificial Intelligence. New York: Chapman & Hall/CRC, 2004

[8] I. Umez-Eronini and Ferat Sahin, "Design of Intelligent Agents for Personal Rapid Transit", presented in the IEEE International Conference on System of Systems Engineering, April 2007.

Iheanyi Umez-Eronini is a M.Sc. student in the Electrical Engineering department at Rochester Institute of Technology. Currently, he is an embedded systems engineer at Syn-Tech Systems in Florida. His research interests are in robotics, Bayesian Networks, personal transit systems, embedded systems, and multi-agent systems. He was an active member of RIT Multidisciplinary Robotics Club and led the team's first entry to the Intelligent Ground Vehicle Competition in 2006 and help design robots for other competitions such as the Trinity Firefighting Competition.

Ferat Sahin received his B.Sc. in Electronics and Communications Engineering from Istanbul Technical University, Turkey, in 1992 and M.Sc. and Ph.D. degrees from Virginia Polytechnic Institute and State University in 1997 and 2000, respectively. In September 2000, he joined Rochester Institute of Technology, where he is an Associate Professor and the director of Multi Agent Bio-Robotics Laboratory. His current research interests are System of Systems, Swarm Intelligence, Robotics, MEMS Materials Modeling, MEMS-based Microrobots, Micro Actuators, Distributed Computing, Distributed Multi-agent Systems, and Structural Bayesian Network Learning. He has over 70 conferences and journals in these areas. He is also the co-author of a book called "Experimental and Practical Robotics". He is also an associate editor of IEEE Systems Journal and International Journal of Computers and Electrical Engineering. He is a member of the IEEE Systems, Man, and Cybernetics Society, Robotics and Automation Society, and Computational Intelligence Society. He has been the Secretary of the IEEE SMC society since 2003. He has received an "Outstanding Contribution Award" for his service as the SMC Society Secretary. He is the Publication Co-Chair for the IEEE SMC International Conference on System of Systems Engineering (SOSE 2007).

Table I: Initial Simulation Results for a System with Two Agents

		1	2	3	4	5	Overall
Rule Based Planner	Min	56	68	73	56	64	56
	Max	224	287	387	368	283	387
	Mean	108.267	127.188	159.538	117.529	123.667	127.238
	Trips	15	16	13	17	12	73
	Collisions	2	0	0	2	0	4
	Coll/trip	0.133333	0	0	0.117647	0	0.054795
		1	**2**	**3**	**4**	**5**	**Overall**
BN Without any Updating	Min	59	56	56	65	56	56
	Max	147	297	281	173	147	297
	Mean	95.059	128.733	104.500	108.211	104.579	108.216
	Trips	17	15	20	19	19	90
	Collisions	1	2	3	2	1	9
	Coll/trip	0.058824	0.133333	0.15	0.105263	0.052632	0.1
		1	**2**	**3**	**4**	**5**	**Overall**
BN with Parameter Updating only	Min	56	56	62	56	59	56
	Max	278	172	188	243	174	278
	Mean	112.143	97.227	97.550	119.833	110.389	107.428
	Trips	14	22	20	18	18	92
	Collisions	2	0	2	1	2	7
	Coll/trip	0.142857	0	0.1	0.055556	0.111111	0.076087
		1	**2**	**3**	**4**	**5**	**Overall**
BN w/ Parameter and Structure Updating	Min	61	56	56	62	62	56
	Max	145	303	202	290	334	334
	Mean	98.000	126.600	109.300	128.688	108.389	114.195
	Trips	17	15	20	16	18	86
	Collisions	2	6	1	1	4	14
	Coll/trip	0.117647	0.4	0.05	0.0625	0.222222	0.162791
		1	**2**	**3**	**4**	**5**	**Overall**
BN w/ Parameter and Utility Updating	Min	56	56	60	56	56	56
	Max	188	199	277	159	146	277
	Mean	103.059	109.421	134.643	95.682	91.476	106.856
	Trips	17	19	14	22	21	93
	Collisions	2	2	3	0	0	7
	Coll/trip	0.117647	0.105263	0.214286	0	0	0.075269
		1	**2**	**3**	**4**	**5**	**Overall**
BN w/ Parameter, Utility, and Structure Updating	Min	56	65	73	57	61	56
	Max	139	386	473	178	146	473
	Mean	100.941	142.917	167.643	108.947	93.429	122.775
	Trips	17	12	14	19	21	83
	Collisions	0	2	5	4	2	13
	Coll/trip	0	0.166667	0.357143	0.210526	0.095238	0.156627

TABLE II: SUMMARY OF EXTENDED SIMULATION RESULTS

Overall Performance	Rule Based Agent	DTA without Updating	DTA with Parameter Updating	DTA with Parameter and Utility Updating
Min Trip Length	56	56	56	56
Max Trip Length	1048	451	394	353
Mean Trip Length	132.2737	112.2108	116.3795	107.6513
Total Trips	558	647	623	671
Total Collisions	15	98	84	87
Collisions/Trip	0.0269	0.1515	0.1348	0.1297