Decision Making in Uncertain Real-World Domains Using DT-Golog [1]

Mikhail Soutchanski and Huy Pham [2] and **John Mylopoulos**[3]

Abstract.

DTGolog, a decision-theoretic agent programming language based on the situation calculus, was proposed to ease some of the computational difficulties associated with Markov Decision Processes (MDPs) by using natural ordering constraints on execution of actions. Using DTGolog, domain specific constraints on a set of policies can be expressed in a high-level program to reduce significantly computations required to find a policy optimal in this set. We explore whether the DTGolog framework can be used to evaluate different designs of a decision making agent in a large real-world domain. Each design is understood as combination of a template (expressed as a Golog program) for available policies and a reward function. To evaluate and compare alternative designs we estimate the probability of goal satisfaction for each design. As a domain, we choose the London Ambulance Service (LAS) case study that is well known in software engineering, but remains unknown in AI. We demonstrate that DTGolog can be applied successfully to quantitative evaluation of alternative designs in terms of their ability to satisfy a system goal with a high probability. We provide a detailed axiomatization of the domain in the temporal situation calculus with stochastic actions. The main advantage of this representation is that neither actions, nor states require explicit enumeration. We do an experimental analysis using an on-line implementation of DTGolog coupled with a simulator that models real time actions of many external agents.

1 Introduction and Motivation

There are many practical domains where the task of designing a decision making agent (that guarantees goal satisfaction with a sufficiently high probability) is difficult due to a very large number of the state features and (ground) actions with uncertain effects. In these domains, the main problem is that state of the art planners cannot scale up to compute (or approximate) an optimal policy due to extremely large size of the state space. Even the task of computing the value of a single policy can be prohibitively difficult in these domains. The second common problem is that in some domains the goal of interest is characterized in terms of quality of an on-going process driven by external agents. To deal with the first problem (scalability), one can try to use a logical representation that avoids explicit state and action enumeration. In addition, one can try to elaborate alternative designs of a decision making agent by goal-means analysis, e.g., using goal regression, a mechanism well studied in AI [11]. By careful refining a goal that must be (at least partially) satisfied into sub-goals, and then by identifying sub-tasks to solve and primitive actions that must be executed to solve these sub-tasks, it is possible to ease to some degree the computational burden of designing such an agent. Indeed, a gradual refinement process can identify useful sequences, loops, conditional or recursive structures of actions that provide together important constraints on the set of policies that need to be considered, and as a consequence, significantly reduce the number of potential policies that ever need to be analyzed. One can imagine also that such analysis can identify where search between alternative actions must concentrate: this can be indicated by nondeterministic choices between alternatives. In realistic domains, this refinement process can lead to different designs depending on how stakeholder goals will be captured in this process. Because this can lead to a variety of designs that need precise evaluation, the problem of designing a decision making agent can be reduced to quantitative evaluation of those different designs of an agent which have been elaborated during the goals-means refinement process. To deal with the second problem (representation of an ongoing interaction with external agents), one can build a simulator of exogenous actions and evaluate all identified alternative designs with respect to the same simulator.

A decision-theoretic extension of Golog (DTGolog) is an expressive framework that is convenient for providing domain specific constraints. It was first introduced in the context of designing efficient controllers for mobile robots [2, 9]. Later, it was extended to the multi-person games [4], was successfully adapted to program robots playing soccer [3], and was also extended to incorporate qualitative preferences to personalize Web services [5].

All previous approaches applied DTGolog to the task of computing a policy in a finite horizon decision-theoretic planning problem. To the best of our knowledge, there were no attempts to use DTGolog as a tool for evaluation of alternative designs of a decision making agent functioning in a large-scale domain characterized by on-going interaction with many external agents. DTGolog is a natural choice for this type of problems because domain specific constraints elaborated during the refinement process can be easily expressed in Golog. Golog provides all standard programming constructs as well as several nondeterministic choice constructs that can be used to specify alternative decisions to be resolved at the moment of decision making. Goals of stakeholders can be modeled using rewards functions, but because mapping of qualitative goals to quantitative rewards is not unique this can lead to several reward functions. The semantics of DTGolog (based on directed value iteration) guarantees that the nondeterministic choices (if any), mentioned in a program, will be resolved to compute an optimal policy given a reward function and a finite horizon. We would like to do extensive analysis of applicability of DTGolog to evaluation of designs using a real case study (each design is represented as a combination of a Golog program and a reward function). In particular, we explore a well-known case study (LAS-CAD) that received a significant attention in the software engineering literature, but remains unknown to the AI community [10, 6]. It is an excellent example of a problem with probabilistic goals. This case study comes from an investigation into a failed software development project. We suggest this case study as a grand challenge for research on planning under uncertainty.

We try to keep our presentation as generic as possible to indicate how our modeling framework can be applied or extended to other decision-making environments. In particular, we clearly show the main features of DTGolog model that can be applied to other domains as well [8, 9]: actions, fluents, precondition axioms, successor-state axioms, initial database, transition probabilities, rewards, Golog procedures for expressing natural constraints on decision making. We illustrate all these main features of the specification framework using the case study.

The main contributions of our paper are the following. First, we developed an extensive logical formalization of a non-trivial domain.

[1] A preliminary version of this paper has been accepted (with another title) as a poster to the 17th European Conference on AI, Riva del Garda, Italy (Aug 28th - Sept 1st, 2006). A long version of this paper is available at http://www.cs.toronto.edu/~mes/papers/LAS/long.pdf

[2] Department of Computer Science, Ryerson University, 245 Church Street, Toronto, Ontario, M5B 2K3, Canada email: {mes, hpham}@scs.ryerson.ca

[3] Department of Computer Science, University of Toronto, Toronto, Ontario, M5S 3G4, Canada email: jm@cs.toronto.edu

Second, we demonstrated that DTGolog is well suited to the task of evaluation of alternative designs of a decision making agent. Third, we did experimental analysis of three different designs of a decision making agent using the same simulator for a fair comparison.

We cannot review all background required to understand this paper, but we ask the interested reader to consult [8] for background on the temporal situation calculus, stochastic actions and Golog, [1] for background on MDPs, and papers mentioned above for background on DTGolog.The full version of this paper includes all introductory material.

2 A Case Study: London Ambulance Service Computer Aided Dispatch (LAS-CAD) System

In this section, we would like to describe a very large real-world domain where there are many stakeholders and the system goal that must be satisfied with a high probability. Subsequently, we provide a detailed logical formalization of this domain, consider three alternative designs of a decision making agent and discus how they can be evaluated using DTGolog. Our formalization of LAS follows [10], because this is the only source of information available to us.

As described in [10], the job cycle of LAS comprises the following phases. (1) Call taking, reviewing and prioritization: a Call Taker gets a 999 emergency phone call requesting an ambulance service and writes down all necessary details including the location of a patient; subsequently, an Incident Reviewer (IR) removes any duplicated requests and assigns priorities. (2) Decision making: depending on the location of the request, the IR forwards each request to one of the three Resource Allocators (RA), each is in charge of one of the three London's city regions (north-west, north-east and south-west), who decides which available ambulance should be sent to serve the incident. (3) Dispatch: once the decision has been made, it will be passed on to a Dispatcher (DSP), who will communicate the mobilization instruction to the appropriate ambulance crew. (4) Mobilization: an ambulance vehicle can be mobilized either from its home base, or from a hospital, or on the road (e.g., using radio link) when a vehicle that completed a previous request is going back to its home base. (5) Travel to scene: an ambulance vehicle (from now on, we call it a car for brevity) travels as quickly as possible to the incident. (6) At scene: upon arrival, an ambulance crew should notify the DSP (e.g., by pressing buttons on the mobile terminal inside the ambulance); then, they perform on-site diagnosis. After doing a diagnosis, the crew decides whether take a patient to a hospital or not. If not, then the car returns back to its home base. (7) Travel to a hospital: if a patient needs hospitalization, then the car travels to a hospital (we assume that the ambulance always travels to the hospital of that region where the patient is currently located). (8) Hand over at a hospital: after spending some time on handing a patient over to a hospital staff, the car reports that it is ready for new assignments and starts going back to its home base (this may require crossing a border between regions, if the ambulance happens to be at a hospital of another region).

One of the most important objectives of the LAS is that requests to be served within 14 minutes from the time the call is received. More specifically, according to the government targets for response times, the activation process (i.e., call taking and mobilization decision) should be less than 3 minutes, and the travel time to the incident should be, for 95% of the time, less than 11 minutes and, for 50% of the time, less than 8 minutes [10]. Clearly, realization of this objective depends crucially on the RA's decision making strategy that can depend on the following factors: whether ambulances are allowed to cross borders between regions or not, whether the same ambulance crew can be consecutively mobilized to serve incidents without having a rest at a base, whether there is information about current locations of all available ambulances, what criteria are used to choose an ambulance, etc. We would like to show that DTGolog is a framework that is expressive enough to provide quantitative evaluation of quality of alternative designs. However, we would like to emphasize that LAS is a very complicated case study involving multiple agents. For this reason, we decide to supplement DTGolog with an elaborated (but conceptually straightforward) simulator that is responsible for generating all exogenous actions and for modeling behavior of ambulances (using assumptions formulated below). The simulator [7] also plays an important role in collecting statistics. Recall that in a single agent case, an offline DTGolog interpreter computes not only a policy (optimal in the set of policies satisfying a given Golog program) and its value. It computes also the probability that an optimal policy successfully follows constraints imposed by a Golog program: an optimal policy, its probability of success and its value are determined by the same reasoning process. However, in a multi-agent system like LAS, this cannot be done directly. For this reason, our simulator provides a probabilistic model that we use for quantitative comparisons. All alternative designs are evaluated with respect to the same probabilistic model to guarantee a fair comparison. Finally, there is another subtle difficulty that has to be mentioned. In the real system, only partial information is available to the RA due to communication failures and other factors. Because we use fully observable MDPs, we introduce additional states that represent lack of information to deal with this complication; e.g., we say, that a location of a car is "unknown", or a sensing action returned the value "unclear".

We will model the three city regions (we use constants NW, NE and S to name them), using three rectangular grid worlds 10×10. In each grid world, each cell represents a city block (an unique location), and is denoted by a term $loc(x, y)$, where x and y are the coordinates. All locations in the city will be referred to by the corresponding cells in which they reside, and the distance between any two locations, $loc(x_1, y_1)$ and $loc(x_2, y_2)$, is defined as the Manhattan distance between the two: $d = |x_2 - x_1| + |y_2 - y_1|$. We assume that each region has one ambulance station (or just base for short), one hospital, and 10 ambulance vehicles (cars for short). We skip other details of our grid-world representation, but they are intuitively clear and the interested reader can find them online at [7]. It is important to understand that the size of the state space is well beyond $30^{300} \cdot 2^{300}$ states, there are many actions in every state (which car should go to an emergency call) and, consequently, the exact solution of the problem of optimal ambulance allocation to incidents is computationally intractable. Moreover, even the task of evaluating policies on this huge state space is computationally intractable. However, we will see below that using DTGolog we can evaluate reasonably interesting domain specific programs by taking advantage of natural constraints on the decision making process in this domain.

To model, and simulate, the emergency servicing trips, we will make the following assumptions.

- The traveling speed of the ambulance in an emergency mode (i.e., when it is going to an incident, or when it is taking the patient to a hospital) is higher than that of in a normal mode (i.e., when it is going back to its home base).
- When the ambulance is outside of its home region, its speeds (both emergency and normal) are slower, due to crew members unfamiliarity with the region.
- The average amounts of time it takes to perform on-site diagnosis (*diagnosis time* from now on), and to carry the patient from an ambulance to a hospital (*unloading time* from now on) are known and the same in all regions.
- Weary crews work more slowly.
- The rate at which emergency requests are received (*request rate* from now on) is known. (We simulate them using a Poisson distribution.)
- The percentage at which patients need to be taken to hospital (*hospitalize rate* from now on) is a constant.
- The communication between the central ambulance control station (where the RA works) and ambulances is unreliable and it is modeled using an uniform distribution. However, each car can be reached reliably at the base.

The logical statements that capture these parameters are: $avgTimePerBlockEmergHome(100)$ – and $avgTimePerBlockNorm$-

$Home(200)$ – home region, emergency (normal, respectively) speed per city block in seconds; $avgTimePerBlockEmergForeign(150)$ – and $avgTimePerBlockNormForeign(250)$ – foreign region, emergency (normal, respectively) travel time per city block in seconds; $diagTime(240)$ – 4 minutes, and $unloadTime(120)$ – 2 minutes; $tirednessLagTime(100)$ – extra time required for a fatigued crew; $requestRate(150)$ – 1 request every 150 seconds, $hospitalizeRate(0.8)$ – in the 80% of calls, a patient needs hospitalization. The rate at which communication fails, given the car is not at the base is represented by the predicate $commFailRate(0.15)$.

We provide our logical representation of the domain in the situation calculus [8] as follows: first, we describe all agents, second, we describe their actions, third, we formulate all fluents, forth, we give precondition axioms for all actions, fifth, all successor state axioms, six, theory to represent an MDP (specifically, probabilities of transitions and reward functions), and finally, we mention what logical statements are included in the initial database.

3 Domain Representation

Agents and actions

There are many roles in the real LAS system. Focusing on just the resource allocating and scheduling aspect of the system, however, only three roles are of significance: the IR, the RA, and the DSP (using the abbreviations from above). At the central position of the system is the RA. For simplicity, we assume that there is only one RA for the whole system. His job is to make resource allocation decisions in such a way that ambulances will arrive within the specified time limit (11 minutes) with a high probability. We will assume that in doing his job, the RA is continuously performing one of the following four actions.

$mobilize(c, loc, t)$ – Send the ambulance c to loc at time t.[4] This is a stochastic action axiomatized as $choice(mobilize(car, loc, t))= \{mobilizeS(car, loc, t), mobilizeF(car, loc, t)\}$. The first outcome $mobilizeS$ corresponds to successful mobilization, the second outcome $mobilizeF$ corresponds to failed mobilization (e.g., due to communication problems).

$askPosition(c, l, t)$ – A sensing agent action that, if performed at time t, will tell the RA the location l of car c. To represent that this action can fail at the current communication failure rate, the simulator can return the special location $Unclear$ (it is used later in the axioms).

$askStatus(car, status, t)$ – Another agent sensing action that determines whether car is $Busy$, $Ready$, or if the current status is $Unknown$ (if communication fails).

$wait(t)$ – A no-cost deterministic agent action that can be performed whenever the RA has nothing to do.

Exogenous actions

The front-end (i.e., call taking, etc) part of the system can be completely summarized and represented by the IR, because, from the point of view of the RA, this is where all emergency requests come from. We will assume that in doing his job, the IR will perform just one action:

$request(l, t)$ – Forward a reviewed incident request to the RA.

The back-end of the system, on the other hand, is completely summarized and represented by the DSP, since he handles all mobilization-related communications and ambulance activity reports. We will assume that, in doing his job, the DSP will perform these three actions.

$reportArrival(car, l, t)$ – Report about arrival of an ambulance car to the RA. This action will tell the RA that car has arrived at location l at time t.

$reportReady(c, l, t)$ – Report about readiness of an ambulance c to the RA. This action will tell the RA that c has become ready at location l at time t.

[4] In the real LAS, performing this action involves not only the RA who notifies the DSP, but also the DSP who informs the appropriate ambulance crew (we gloss over details to simplify our model).

$reportLocation(car, loc, t)$ – Inform about the current location loc of car at time t.

Since we consider here a DTGolog approach that accounts for a single decision maker only, we represent the behavior of the RA using a Golog program composed from his agent actions only, while treating the IR and DSP as external agents whose behaviors are simulated (using programs written in C). Note that the external agents (i.e., the IR and DSP) perform their actions in an exogenous fashion: their actions can happen any time and are outside of the direct control of our Golog program representing the RA.

To capture logical properties of the domain (state features) we introduce the following predicates.

Fluents

$ready(c, s)$ - a car c is ready in situation s

$carLocKnown(c, t, s)$ - the location of a car c is known in situation s at time t.

$carLocation(c, l, t, s)$ - c is located at l at time t in s

$commLost(c, s)$ - communication with a car c is not available in situation s

$requestPending(l, s)$ - there is an emergency request from a location l in situation s

$atBase(c, s)$ - a car c is at its home base in situation s

$consecTripCount(c, n, s)$ - c made n consecutive trips in s

Situation Independent Predicates:

$region(name, id, loc(x_1, y_1), loc(x_2, y_2))$ - the region $name$ has a bottom left corner at $loc(x_1, y_1)$ and a top right corner at $loc(x_2, y_2)$, e.g., the north-west region is described by $region(NW, 1, loc(1, 11), loc(10, 20))$.

$base(name, id, region, l)$ - the base $name$ with an id is located in $region$ at the location l, e.g., $base(B1, 1, NW, loc(7, 14))$.

$carRange(region, rName)$ - $rName$ is the name of the set of all cars from $region$, e.g., $carRange(NW, NWcars)$.

$range(rName, f)$ - f is the finite set named $rName$, $range(NWcars, \{c1, c2, c3, c4, c5, c6, c7, c8, c9, c10\})$. There is also a finite set named All that includes all 30 cars.

There are also several other predicates with an obvious implementation and with the meaning that is easy to understand from their names: $isARegion(reg)$, $isAHospital(h)$, $isABase(b)$, $isACar(c)$, $validXCoord(region, x)$, $validYCoord(region, y)$, $hospital(name, id, region, l)$ $homeRegion(car, reg)$, $homeBase(car, base)$, $in(base, reg)$, $in(hosp, reg)$, $in(loc, reg)$, $inSameRegion(a, b)$, $locOf(base, loc)$, $locOf(hosp, loc)$; see details in [7].

Action precondition axioms

$Poss(wait(t), s)$

$Poss(mobilizeS(car, loc, t), s) \equiv$
$\quad ready(car, s) \wedge carLocKnown(car, t, s)$

$Poss(mobilizeF(car, loc, t), s) \equiv$
$\quad ready(car, s) \wedge carLocKnown(car, t, s)$

$Poss(askPosition(car, l, t), s)$

$Poss(askStatus(car, status, t), s)$

Successor state axioms

A car is ready if it is reported that it is ready, or if the RA asked about the current status of the car and the result of this sensing action was that it is $Ready$, or the car is ready in the previous situation s and the last action is not a sensing action informing the RA that the car is busy or its status is unknown or not a mobilization action.

$$\begin{aligned}
&ready(car, do(a, s)) \equiv\ (\exists l, t)a = reportReady(car, l, t) \vee \\
&\quad (\exists t)a = askStatus(car, Ready, t)\ \vee \\
&ready(car, s) \wedge \neg(\exists l, t)(\, a = mobilizeS(car, l, t) \vee \\
&\qquad a = askStatus(car, Busy, t) \vee \\
&\qquad\quad a = askStatus(car, Unknown, t)\,).
\end{aligned}$$

Communication between ambulance crews and the DSP (and hence the RA) can fail. We model this by allowing the sensing action $askPosition(car, l, t)$ to return the constant $Unclear$ instead of a genuine location. More specifically, communication with a given car is said to be lost if: the RA tried to ask for its location or for its status and the reply was unclear, and the car has not get back to the RA since then (i.e., the DSP did not execute $reportReady$ or $reportArrival$ to pass information from the ambulance crew to the RA). In addition, if the mobilization fails, this indicates that communication is lost.

$$\begin{array}{l} commLost(car, do(a,s)) \equiv \\ \quad (\exists t)\, a = askPosition(car, Unclear, t) \vee \\ \qquad (\exists t)\, a = askStatus(car, Unknown, t) \vee \\ \qquad\quad (\exists t, loc)\, a = mobilizeF(car, loc, t) \vee \\ \quad commLost(car, s) \wedge \neg(\exists l, t)\, (a = reportReady(car, l, t) \vee \\ \qquad\qquad a = reportArrival(car, l, t)). \end{array}$$

The car c is located in l at $time$ in situation $do(a,s)$, if the last executed action a is that the crew of the ambulance c reported (via DSP) about readiness at the home base, or the crew is reported that it is ready elsewhere (the simulator makes sure that the car can report its readiness only at a hospital, at a location in the city, if a patient does not need hospitalization, or at the home base) and started to move towards the base and less than p seconds passed since the moment when information about l was received. Also, the location is l if the RA asked the crew of c recently about its position and got a clear reply, or the location was l in the previous situation s and the car was not successfully mobilized more than p seconds ago (i.e., either it is not moving, or if it is moving, then it started less than p seconds ago).

$$\begin{array}{l} carLocation(c, l, time, do(a,s)) \equiv \\ \quad (\exists t)(\, a = reportReady(c,l,t) \wedge t \le time \wedge \\ \qquad (\exists n, id, r) base(n, id, r, l)\,) \vee \\ \quad (\exists l, t, p)(a = reportReady(c,l,t) \wedge t \le time \le t + p \wedge \\ \qquad \neg(\exists n, id, r) base(n, id, r, l) \wedge validPeriod(p)) \vee \\ \quad (\exists l, t, p)(a = askPosition(c,l,t) \wedge t \le time \wedge \\ \qquad (\exists n, id, r) base(n, id, r, l)\,) \vee \\ \quad (\exists l, t, p)(a = askPosition(c,l,t) \wedge t \le time \le t + p \wedge \\ \quad \neg(\exists n, id, r) base(n, id, r, l) \wedge l \ne Unclear \wedge validPeriod(p)) \vee \\ \quad (\exists l', t, p)(a = mobilizeS(c, l', t) \wedge \\ \qquad validPeriod(p) \wedge time - t > p) \wedge l = Unknown \vee \\ carLocation(c, l, time, s) \wedge \neg(\exists l', t, p)(a = mobilizeS(c, l', t) \wedge \\ \qquad validPeriod(p) \wedge time - t > p)). \end{array}$$

The successor state axiom for the fluent $carLocKnown$ is very similar to the previous axiom and we omit it (see [7] for details). An emergency request from the location l for an ambulance service is pending in $do(a,s)$ if an emergency call is made from l, or if in the previous situation s, the request was pending and no ambulance is mobilized to this location l.

$$\begin{array}{l} requestPending(l, do(a,s)) \equiv (\exists t)\, a = request(l,t) \vee \\ \quad \neg(\exists c, t)\, a = mobilizeS(c,l,t) \wedge requestPending(l,s)). \end{array}$$

The ambulance car c is at its home base, if the crew of c (via DSP) reported to the RA from the location l that it is ready, and b is the home base of c and b is located at l, or the RA asked the crew of c about their location and got a reply that c is at the home base, or in the previous situation s, c is at the home base and it is not mobilized successfully to serve an incident.

$$\begin{array}{l} atBase(c, do(a,s)) \equiv (\exists l, t, b)(a = reportReady(c,l,t) \wedge \\ \qquad homeBase(c,b) \wedge locOf(b,l)) \vee \\ \quad (\exists l, t, b)(a = askPosition(c,l,t) \wedge \\ \qquad homeBase(c,b) \wedge locOf(b,l)) \vee \\ \quad atBase(c,s) \wedge \neg(\exists l, t) a = mobilizeS(c,l,t). \end{array}$$

An ambulance c made n consecutive trips in $do(a,s)$, if it made $n-1$ consecutive trips in s and is successfully mobilized again, or if it made n consecutive trips in s and it is neither mobilized, nor reported that it is ready at its home base. Once the crew of c reports that c is ready at its home base, the number of consecutive trips is 0.

$$\begin{array}{l} consecTripCount(c, n, do(a,s)) \equiv \\ \quad (\exists l, t, b)(a = reportReady(c,l,t) \wedge \\ \qquad n = 0 \wedge homeBase(c,b) \wedge locOf(b,l)\,) \vee \\ (\exists l, t) a = mobilizeS(c,l,t) \wedge consecTripCount(c, n\text{-}1, s) \vee \\ consecTripCount(c,n,s) \wedge \neg(\exists l, t, b)(a = mobilizeS(c,l,t) \vee \\ \quad a = reportReady(c,l,t) \wedge homeBase(c,b) \wedge locOf(b,l)). \end{array}$$

Initial Situation

In the initial situations, all ambulances are ready, they are at home bases and their locations are known (axioms can be found in [7]).

The basic action theory for the LAS domain includes also unique names axioms for actions: they state that all physical and sensing actions (both agent and exogenous) are pairwise unequal.

We need several additional groups of axioms to specify an MDP: probabilities of transitions in an MDP for all stochastic agent actions, reward functions and sense conditions that need to be evaluated to distinguish between different outcomes of every stochastic action.

Probabilities of outcomes are defined as follows:

$$\begin{array}{l} prob(mobilizeS(c,l,t), p, s) \overset{def}{=} carLocation(c,l,t,s) \wedge \\ \quad ((\exists n, id, r) base(n, id, r, l) \wedge p = 1 \vee \\ \quad \neg(\exists n, id, r) base(n, id, r, l) \wedge commFailRate(f) \wedge p = 1\text{-}f). \\ prob(mobilizeF(c,l,t), p, s) \overset{def}{=} carLocation(c,l,t,s) \wedge \\ \quad ((\exists n, id, r) base(n, id, r, l) \wedge p = 0 \vee \\ \quad \neg(\exists n, id, r) base(n, id, r, l) \wedge commFailRate(f) \wedge p = f). \end{array}$$

The most essential part of our theory is the set of definitions for rewards.[5] We will use two different reward functions. The first one takes into account the traveling distance only: it is intended to encourage mobilization of the car nearest to an incident without regard to all other factors. The idea is to provide the DTGolog's decision making operator $\pi(var : \tau)\delta$ (nondeterministic finite choice of action argument), where $\tau = \{c_1, \ldots, c_n\}$ is the finite set of alternative cars that can be chosen for mobilization, with the ability to pick the car with highest chance of getting to the incident on time. Essentially, given an available car c and a location l, the reward r that the program can expect to receive for mobilizing c to l is directly proportional to the probability that the travel time is no more than 11 minutes (or 660 seconds): $r = c \cdot Prob\{0 \le t \le 660\}$, where c is a constant 100 (defined by $rOntime(100)$ in the model), and t is a random variable that represents the travel time. Rewards for doing other actions (except of mobilization) are defined as 0. For simplicity, we assume that travel time t has Gaussian distribution. More specifically, $t = (d+1) \cdot (\mathcal{N}(0,1) + v)$, where $\mathcal{N}(0,1)$ is the Gaussian random variable with the mean 0 and variance 1, d is the Manhattan distance between a car and an incident (i.e., d is the number of city blocks that a car has to travel) and v is the average travel speed (in seconds per city block). (Note we assume that time is needed to travel from a city block to itself, this is why we write $d+1$ instead of d.) Consequently, $Prob\{0 \le t \le 660\}$=$Prob\{t \le 660\} - Prob\{t \le 0\}$= $Prob\{\mathcal{N}(0,1) \le \frac{660-(d+1)\cdot v}{d+1}\} - Prob\{\mathcal{N}(0,1) \le -v\}$. The reward function provided in the model, shown below, captures this equation and serves as a measure of how likely a given car, if mobilized, will make it to the incident on time.

$$\begin{array}{l} reward(r, S_0) \overset{def}{=} r = 0 \\ reward(r, do(a,s)) \overset{def}{=} r = 0 \wedge \neg(\exists c, l, t) a = mobilizeS(c,l,t) \\ reward(r, do(mobilizeS(c, loc(x,y), t), s)) \overset{def}{=} \\ \quad (\exists x_1, y_1, d, v, r_{on})\; carLocation(c, loc(x_1,y_1), t, s) \wedge \\ \qquad distance(loc(x_1,y_1), loc(x,y), d) \wedge rOntime(r_{on}) \wedge \\ \qquad\quad avgTimePerBlockEmergHome(v) \wedge \\ \quad r = (Prob\{\mathcal{N} \le \frac{660-(d+1)\cdot v}{d+1}\} - Prob\{\mathcal{N} \le -v\}) \cdot r_{on}. \end{array}$$

This first reward function reflects only the system goal that requests have to be served quickly, but neglects take into account human factors (e.g., desire of crews to have rest between assignments). Note also that if the RA uses this reward function, then he does not account for unfamiliarity with foreign regions by assuming that the travel speed is always the emergency speed in the home region (in the simulator, it varies from the home region to a "foreign" region). In other words, by using this reward function, the RA overlooks important features of the domain and assumes that ambulance crews are equally comfortable to travel in any region. We define it intention-

[5] We talk about reward functions, but we define them using predicate symbols instead of function symbols to make their implementation in Prolog straightforward.

ally in this way to demonstrate that more carefully designed reward functions are possible too.

The second reward function is more sophisticated. It takes into account not only the distance, but also crew fatigue which introduces a lag in the response time (recall the predicate $tirednessLagTime(lag)$ defined above), region familiarity (using the predicate $avgTimePerBlockEmergForeign(v)$ defined above), and the fluent $consecTripCount(c, m, s)$ that helps to determine the number m of consecutive trips done by the car c. The definition of this reward function parallels the previous definition.

$$
\begin{aligned}
& reward(r, do(mobilizeS(c, loc(x, y), t), s)) \overset{def}{=} \\
& (\exists x_1, y_1, d, m, v, lag, r_{on})\, carLocation(c, loc(x_1, y_1), t, s) \wedge \\
& \quad distance(loc(x_1, y_1), loc(x, y), d) \wedge \\
& \quad\; consecTripCount(c, m, s) \wedge \\
& \quad\;\; tirednessLagTime(lag) \wedge rOntime(r_{on}) \wedge \\
& \quad\;\;\; (\forall g)(\, homeRegion(g) \wedge in(loc(x, y), g) \supset \\
& \qquad\quad avgTimePerBlockEmergHome(v)\,) \wedge \\
& \quad\;\;\; (\forall g)(\, homeRegion(g) \wedge \neg in(loc(x, y), g) \supset \\
& \qquad\quad avgTimePerBlockEmergForeign(v)\,) \wedge \\
& r = (Prob\{\mathcal{N} \le \tfrac{660 - m \cdot lag - (d+1) \cdot v}{d+1}\} - Prob\{\mathcal{N} \le -v\}) \cdot r_{on}.
\end{aligned}
$$

According to this reward function, if lag is sufficiently large number (say, lag=100) and an ambulance c did already several trips (say, $m = 6$ trips), then the reward for choosing this ambulance is low, because chances for this ambulance to arrive in time are less than $Prob\{\mathcal{N}(0, 1) \le \frac{60-(d+1) \cdot v}{d+1}\}$, which is a very small number even if an incident happened in the same city block, i.e., d=0 (recall that v=100). Thus, this reward function takes into account not only the system goals, but also interests of crews.

This completes our logical representation of the domain and the MDP associated with the problem of allocating ambulances.

4 Golog Procedures and Simulation Results

We model three allocation strategies using two Golog procedures and two reward functions. The first procedure resembles the strategy used by the human RA of the manual system as described in [10]. For this reason, we call it a *manual* system, and implement it by a deterministic Golog program that does not do any decision theory. The main idea of this procedure is that the RA picks the ambulance car nearest to the incident by comparing distances of all cars from the incident. However, this choice is subject to important restriction that ambulances cannot cross borders between regions, and they serve incidents (and hospitals) from their home regions only. The second Golog program resembles the strategy used by an *automated* system as described in the LAS report, and this program is coupled with the first reward function. This second Golog program makes nondeterministic choice from a finite set of cars and, consequently, allows non-trivial decision making: the car nearest to an incident is picked using the first reward function. Any ambulance can be send to any region, as long as it is deemed as the ambulance that will reach the incident in time with the highest probability. The third strategy (a combination of the same second Golog program with the second reward function) is a hypothetical *optimized* system, in which the ambulance allocation task is also casted as a decision theoretic problem, but interests of ambulance crews are also reflected in the reward function. We could also implement in DTGolog more sophisticated strategies (e.g., those which require planning for several steps ahead), but for simplicity of presentation we limit ourself with strategies mentioned above. We do quantitative comparison of these 3 strategies using our simulator. To present results, we explain the structure of both Golog programs and then we provide tables with numerical data.

The first Golog program does the following. For a duration of d seconds, it checks continuously if there is a region with at least one request. If yes, and if there is at least one car with the home base in this region that is currently ready and whose location is known, then procedure chooses a region and an incident in this region, finds a car that is the nearest to the incident, the distance between this car and the incident, and finds also the distance between the incident and the base. If there is a car at the base and the distance from the base to the incident is no more than 2 city blocks greater than the distance from the nearest car to the incident, the procedure mobilizes a car from the base; otherwise, it mobilizes the nearest car. In the case when there is no pending request, or there are pending requests, but all cars are busy, the procedure just performs the no-cost action wait() and then calls itself recursively. If more than d seconds passed (the end of working shift), then the procedure executes no cost $noOp()$ action that does nothing and quits. (Due to space restrictions, this Golog program is not included here, but it is included in the extended version of the paper in [7].)

The automated system does not take into account human factors (such as crew fatigue and unfamiliarity with "foreign" regions) and uses the first reward function. (Note: The simulator always does calculations that take all these factors into account, regardless of what allocation strategy is used.) The second Golog program that implements the automated system works in much the same way as the Golog program for the manual system, except a few important details. The main difference is that it picks (via DTGolog's nondeterministic finite choice $\pi(car, rangeName)$) the car that it believes to have the highest chance of getting to the incident on time and mobilizes it. Because this non-deterministic operator occurs inside the scope of the $limit()$ operator, the choice of the car to mobilize is made by solving a simple decision task with the horizon 1. Note that DTGolog can also accommodate more far-sighted decision making (DTGolog was designed for finite horizon decision-theoretic planning with constraints), and consequently, more sophisticated decision making agents can be considered as well. The second Golog program gives no preference to cars at the base, and to serve a request, it considers all cars from all regions. Thus, this second program provides a good illustration how DTGolog can take advantage of the structure of this domain by providing natural constraints on the set of policies that need to be considered.

```
proc $allocResAuto(d)$
 $\pi(t)$ [ $(now(t))?;$
  if $t < d$ then $limit($ $queryAllCars(t)$ ;
   if % At least one request is pending
      % and at least one car is mobilizable
     $(\exists r, x, y, c_1)(\, isARegion(r) \wedge validXCoord(r, x) \wedge$
      $validYCoord(r, y) \wedge requestPending(loc(x, y)) \wedge$
      $isACar(c_1) \wedge ready(c_1) \wedge carLocKnown(c_1, t)\,)$
   then $\pi(r, x, y, rName)$ [ $incidentLoc(r, x, y)$ ;
          % Pick the best car and mobilize it
      $(range(All, rName))?;$
     $\pi(car, rName)\ mobilize(car, loc(x, y), t)$ ]
   else $wait(t)$ % end of "limit"
  $)$ ; $allocResAuto(d)$ % Recursive call
  else $noOp(t)$ ]
endProc
```

As we mentioned above, an "optimized" system uses also the second Golog program, but it makes better decisions than the automated system by relying on the second reward function. The interested reader can find all details about these Golog programs, their implementation in Prolog (as well as numerical data we collected) in [7].

The simulator calculates travel times for ambulances using a simple assumption that travel time of any ambulance along a city block has the Gaussian distribution (because travel time cannot be negative the simulator samples until it gets a positive value). To simulate the travel time between $loc(x_1, y_1)$ and $loc(x_2, y_2)$, let d be the Manhattan distance between them, v be the average number of seconds it takes for the car to travel one city block, and apply this formula:

$t(loc(x_1, y_1), loc(x_2, y_2)) = \sum_{i=1}^{d} \mathcal{N}(v, 1)$, where $t(loc_1, loc_2)$ is the time we want to calculate, and $\mathcal{N}(v, 1)$ is a positive random number drawn from the Gaussian distribution with mean v and variance 1. The simulator takes a short trajectory between two locations and if it intersects a border between the home region and another region, the simulator uses the appropriate value of speed v for each city block (as defined above). In addition, the simulator uses a data structure that allows appropriate exogenous actions to be generated at the right time and be inserted into the situation term.

To collect the statistics, we performed simulation of the three allocation strategies using several request rates. For each request rate, each strategy was called 10 times, each time for approximately 300 requests. We run simulation on an AMD 1800 Mhz machine with 1GB of memory running Linux kernel 2.6.8. For each request rate, the process of collecting data for 300 requests takes approximately 5 hours. This indicates that decision making for one request takes less than 1 minute on average (because the simulator takes some time to do required computations). This time is mostly due to unoptimized implementation of the DTGolog interpreter in Prolog. The interpreter calls external functions and gets results from our simulator using C–Prolog interface. The resulting values (presented in the tables) are averages over 10 runs. The entries in the table, $A(B + C + D)$, mean that in the given design at the given request rate, A percents of the time, it took more than 8 or 11 minutes for the ambulance to reach its incident's location. Out of this A percents, B percents was caused by long travel time (i.e., the car simply spent more than 11 minutes in traffic), C percents was caused by mobilization delay (i.e., all cars were busy at the time the incident occurred), and D percents was the result of both mobilization delay and long travel time. The first table represents percentage (rounded to an integer) of those trips that take more than 8 minutes, the second table - percentage of those trips that take more than 11 minutes. As one might expected, the performance of different strategies are in the right order: the "optimized" allocation strategy is better than "automated", and "manual" allocation strategy is somewhat better than "automated". Additional experimentation (with different sets of parameters) might be necessary to provide statistically significant comparison between "optimized" and "manual" strategies. We did not do this in our paper, because our intention was not to advocate the superiority of one particular strategy, but that each design of a decision making agent can be expressed and compared with other alternatives using DTGolog.

Rates	Manual	Automated	Optimized
60	70(14+35+20)	68(16+11+42)	62(16+9+36)
70	55(21+21+13)	66(23+9+35)	57(26+5+26)
80	45(25+13+7)	56(34+4+18)	32(32+0+0)
90	39(29+6+4)	37(37+0+0)	31(31+0+0)
120	30(30+0+0)	35(35+0+0)	32(32+0+0)
150	31(31+0+0)	33(33+0+0	28(28+0+0)

Rates	Manual	Automated	Optimized
60	56(4+45+7)	58(7+15+36)	53(9+13+31)
70	36(5+27+4)	53(10+12+31)	43(12+7+24)
80	24(6+17+2)	35(14+6+16)	11(11+0+0)
90	16(7+8+1)	11(11+0+0)	9(9+0+0)
120	8(8+0+0)	8(8+0+0)	8(8+0+0)
150	8(8+0+0)	8(8+0+0)	7(7+0+0)

5 Discussion and Conclusion

We consider applicability of DTGolog to the task of evaluation of alternative designs of a decision making agent. We show that domain dependent constraints on the set of policies can range from a purely deterministic program (no decision making at all) to a Golog program with a limited number of decision points. In the latter case, a finite horizon planning can be accomplished by the off-line DTGolog interpreter that resolves nondeterministic choices in an optimal way using one of the given reward functions. In this paper, we consider only very simple Golog programs, but DTGolog can handle also more sophisticated strategies and longer horizons. We demonstrate that DTGolog is a very expressive framework that seamlessly combines programming with decision-theoretic planning in the fully observable MDPs. To represent a process-oriented problem that continues indefinitely (as long as new exogenous requests arrive), we supplement DTGolog with a simulator that generates randomly requests and several other exogenous actions (e.g., arrivals of ambulances) and also computes traveling time. Because we choose a domain where the state space has well beyond $30^{300} \cdot 2^{300}$ states (3 grid-worlds 10×10 with at least 1 request anywhere and 30 cars located anywhere) and there are many actions available in every state, it is doubtful that even state-of-the art MDP solvers (such as SPUDD) can solve the decision-theoretic planning problem in this domain. As a consequence, quality of alternative designs cannot be evaluated by comparison with policies computed by the decision-theoretic planners. However, our simulator provides statistical data about traveling time, and using these data, we can determine the number of cases when ambulances arrive in time or too late, and, as a consequence, it is possible to evaluate quantitatively each design.

The large number of choices and the large number of actions with uncertain outcomes present computational challenges that have to be addressed in future work. The most important direction for future research is overcoming computational challenges of the DTGolog framework: using sampling to deal with large branching factor (in the version of directed value iteration that provides semantics for a DTGolog interpreter [2]) and using progression to deal with long situations [8]. Our research goal is a more advanced framework to handle models that are large enough to be of use in software design applications such as this one. In 2004, the real LAS-CAD system included about 30 regions, about 400 vehicles and was the largest public ambulance system in the world.[6]

REFERENCES

[1] Craig Boutilier, Thomas Dean, and Steve Hanks, 'Decision theoretic planning: Structural assumptions and computational leverage', *J. of Artificial Intelligence Research*, **11**, 1–94, (1999).

[2] Craig Boutilier, Ray Reiter, Mikhail Soutchanski, and Sebastian Thrun, 'Decision-theoretic, high-level agent programming in the situation calculus', in *Proceedings of the 17th Conference on Artificial Intelligence (AAAI-00)*, pp. 355–362, Austin, TX, (July 30–3 2000). AAAI Press.

[3] A. Ferrein, Ch. Fritz, and G. Lakemeyer, 'Using Golog for deliberation and team coordination in robotic soccer', *KI Künstliche Intelligenz*, **05**(1), 24–43, (2005).

[4] Alberto Finzi and Thomas Lukasiewicz, 'Game-theoretic agent programming in Golog', in *Proceedings of the 16th biennial European Conference on Artificial Intelligence (ECAI 2004)*, Valencia, Spain, (August 2004). IOS.

[5] Christian Fritz and Sheila McIlraith, 'Compiling qualitative preferences into decision-theoretic Golog programs', in *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR-06)*, ed., John Mylopoulos, Lake District, UK, (June 2006).

[6] Emmanuel Letier and Axel van Lamsweerde, 'Reasoning about partial goal satisfaction for requirements and design engineering', in *Proceedings of the 12th International Symposium on the Foundation of Software Engineering FSE-04*, pp. 53–62, Newport Beach, CA, (November 2004). ACM Press.

[7] Huy Pham, Mikhail Soutchanski, and John Mylopoulos, 'A simulator and a Golog implementation of the London Ambulance Service (LAS) Computer-Aided Dispatch (CAD) system', Technical report, Department of Computer Science, Ryerson University, http://www.cs.toronto.edu/˜mes/papers/LAS/index.html, (2006).

[8] Raymond Reiter, *Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems*, MIT Press, 2001.

[9] Mikhail Soutchanski, *High-Level Robot Programming in Dynamic and Incompletely Known Environments*, Ph.D. Thesis, Computer Science Dep., University of Toronto, http://www.cs.toronto.edu/˜mes/papers/index.html, 2003.

[10] The Communications Directorate, *Report of the Inquiry Into The London Ambulance Service (South West Thames Regional Health Authority)*, The 8th International Workshop on Software Specification and Design Case Study. Electronic Version prepared by Anthony Finkelstein. Available at http://www.cs.ucl.ac.uk/staff/A.Finkelstein/las.html, February 1993.

[11] Richard Waldinger, 'Achieving several goals simultaneously', in *Machine Intelligence*, eds., E. Elcock and D. Michie, volume 8, (1977).

[6] http://www.lond-amb.sthames.nhs.uk/news/performance/performance.html