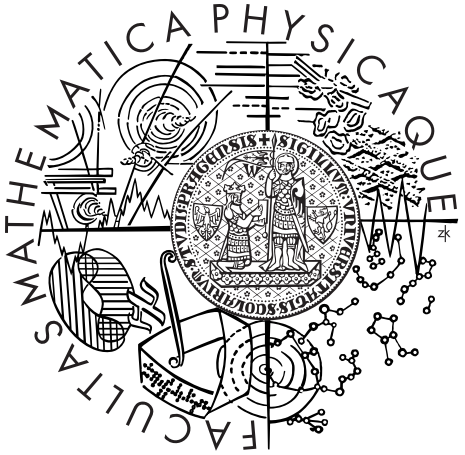
Charles University in Prague
Faculty of Mathematics and Physics

MASTER THESIS



Zdeněk Běhan

Generování životních episod za účelem testování modelů episodické paměti
Generating life episodes for the purpose of testing of episodic memory models

Department of Software and Computer Science Education

Supervisor: Mgr. Rudolf Kadlec

Study programme: Computer Science, Theoretical Computer Science

2012

I would like to thank my thesis supervisor, Mgr. Rudolf Kadlec, as well as the consultant, Mgr. Cyril Brom, PhD. for valuable support, input and direction of this work.

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

Prague, August 10, 2012 Zdeněk Běhan

Název práce: Generování životních episod za účelem testování modelů episodické paměti
Autor: Zdeněk Běhan
Katedra: Kabinet software a výuky informatiky
Vedoucí diplomové práce: Mgr. Rudolf Kadlec

Abstrakt:
Cílem práce je vytvořit generátor událostí generující korpus vstupních episod, které se dají použít jako vstupní data pro testování modelů episodické paměťi. Specielně, použité řešení by mělo zajistit, že bude výsledek bude dostatečného rozsahu pro simulaci v řádu let až celého života typického agenta. Rovněž se snaží výsledek ověřit na skutečném modelu episodické paměti, upraveném pro tento účel, a zjistit, zda generovaná data jsou dostatečně kvalitní pro testování psychologických poznatků na umělých modelech paměti.

Klíčová slova: episodická paměť, virtuální agent, korpus episod

Title: Generating life episodes for the purpose of testing of episodic memory models
Author: Zdeněk Běhan
Department: Department of Software and Computer Science Education
Supervisor: Mgr. Rudolf Kadlec

Abstract:
The goal of this work is to create a generator that provides a corpora of input episodes in the specified format, which can be used as an input to test episodic memory models. More specifically, the methods used should ensure the scope to be in years up to a possible lifetime of a typical human agent. It also attempts to verify this on an actual episodic memory model, and test if the generated data has high enough quality to be used for testing psychological paradigms on memory models.

Keywords: episodic memory, virtual agent, episode corpora

Contents

1. Introduction

Memory is a basic feature of the human brain. It retains information, but more importantly, it processes it and allows for later use. The basic distinction of the human memory is based on several criteria. Short term and long term memory, episodic, spatial and semantic memory.

Each of these is handled by a separate part of the brain, and interconnected, they help form what humans are. The actual mechanisms are being analyzed as a complex blackbox by psychologists. For that reason, what we know about these is usually experimental evidence of a paradigm in motion, but the neurosciences still only know very little about how the memory actually works internally.

Several of the neuroscience works, however, reveal important parts of the puzzle, based on which we can speculate how the memory works, and based on that, create computer algorithm representations of that. These are by no means exactly the same as the actual brain, but they attempt to mimic it to our best knowledge. This implies the need for testing such models to verify that for a particular practical purpose, the model acts the same or similar as a human agent would.

For example, scientists point out particularities of the human memory, such as *False memories* [1]. This whole field is dedicated to how people distinctly remember something they never saw before, resulting in a number of applications, such as Eyewitness Testimony debunking*Eyewitness Testimony*.

The particular topic of the capacity of the human memory has been in the focus of psychology researchers' interest for decades, and to date, an exact answer is not known, some sources claiming as much as infinite capacity, with other sources [2] being more modest with *just enough to be useful*. Some examples, more in the field of curiosities and memory defects, describe people with fascinating memory, remembering everything they ever saw and lived in vivid detail. Medical reports on memory defects after suffering a brain injury and the curious aftereffects also shed some light on the topic.

These results are frequently applied to the Computer Science topic of *Artifical Agents*. The most prominent use of virtual agents is in simulated environments and computer games, where developers strive to provide life-like experience to players, resulting in an ever increasing attempts at plausiblity of artificial agents. This includes orientation in space and *spatial memory*, planning their daily lives, achieving goals or storing life episodes into *episodic memory* and later retelling them when questioned, as well as having a general understanding of the world through *semantic memory*.

In a human, all of the systems are functioning since birth and are interconnected to form a person. In fact it is known that most of these systems are vital for proper function of all other systems. Therefore, the most significant complex-

ity in developing artificial agents in this field is in properly stubbing out some mechanisms in order to test other mechanisms, as attempting to implement the whole system would be very difficult and examining its parameters very volatile. Specifically, for testing episodic memory, you already have to have your agent be able to live their daily life, so that they can remember life's episodes, but in reality, past episodes and their recognised *schemas* are an essential part of being able to live in the first place.

The goal of this work is to provide a framework for generating large volumes of life episodes in a format that allows for testing of episodic memory, without having to do any real agent simulation. It is assumed the resulting memory model that has undergone this kind of simulation should also be able to work in scenarios that use a similar notation of life episodes, and therefore we choose to base this work on a particular another work to provide the structure of episodes.

Accompanying to this generator is a heavilly modified pre-existing memory model by Michal Cermak, used as a sink for the corpora of episodes. We also attempt to point out of some of the psychological paradigms in practice, demonstrating usability of the generator for practical purposes.

A big point to the work is ensuring the scale needed for generation of very large volumes of events. Given the computational complexity of most related algorithms, this is usually done through heuristics, rather than optimisation.

The episodic memory model is based on several psychological research articles, with the addition of ideas by *Cyril Brom*. Detailed description follows up later.

2. Related work

This work builds on a number of related research works. Most of the common terms are either based on or related to Peskova[4] and related works. The *AND-OR* trees are based on work done by Cyril Brom, specifically [12] and some preceding work.

Most work is only loosely related, using the same concepts, however, some of them served as a basis for this work. Specifically, both the memory model and the corpora generator are directly based on and extend from another credited work.

It is possible to find this work online at [15], with repository access allowing insight into which parts are original and which are derived. This information is also described by Appendix C.

2.1 Planning and Corpora generation

The episode corpora generator is inspired by and based on Monroe emergency response domain planner [13], a SHOP2-based Common Lisp planner. Contrary to the intent of SHOP2, this planner does not attempt to produce an optimum plan, rather randomizes the world state based on given rules, and finds any plan, described by a lisp tree structure. If repeated in a loop, this provides a *corpora* of plans, that are conforming to defined domain constraints.

This is further extended and built upon by the life episode generator that is part of this work. This work features a number of deep changes that significantly modify the previous behaviour. Most importantly, weight based randomization, and statefulness of multiple simulations, are described later.

Related master thesis was done by Lucie Kucerova, on generating complex scenario plans as a use for a similar purpose as this work. Due to her work being multi-agent based and the resulting combinatorial complexity, the result was mostly suggesting that this is not the way to go. It explored using HTN planning, as well as SHOP does, for the same purpose, but found no suitable planning tool that could generate outputs in a reasonable timeframe.

2.2 Episodic memory modelling

This work uses a model that is the original work by Michal Cermak, a master thesis done in parallel currently in progress. His work provided the original memory model which this work has embraced and significantly extended. Furthermore, the goal of this work is to be reasonably compliant with his event format, so that in the future, we can combine the use of episode corpora and life episodes from

the *Pogamut*[16] framework.

The model is originally designed on top of a Pogamut application, collecting events from an actual game *Agent*. This required a substantial change of the input interface. However, it also implied other, deeper conflicts with concepts of Corpora generation. For instance both the way in which *time flow* or *memory reorganisation* is handled was generally incompatible with the goals of this work.

Previous work on this topic was a master thesis by Ondrej Burkert [14], simulating agents in *Pogamut* and determining some basic time concepts from the events. However, this provided no means of storing actual memories, merely certain form of semantic memory. Some of the concepts of schematic memory used in the memory model are based on that. It also inspires the concept of *temporal schemas* used herein in the resulting memory model.

Note that while this work is based on the said model by Michal Cermak, the structure, algorithms and any aspect of the original work has significantly diverged past the fork point and any statements made about any aspects of the original model may cease to be true in the future.

3. Problem description and Basic Terms

The basic goal of this work is an interaction between a corpora generator and a memory model. Namely bulk feeding[1] all episodes into the memory in a manner similar to a pure simulation. Due to very different nature of the two processes, there is a third module that reads input lisp trees, parses them and feeds them into the memory. This replaces the interface to *Pogamut* in the original memory model.

Other parts of this work include tools to automatically analyze and inspect domains. These are written in Common Lisp and reuse some of the original SHOP code.

3.1 Memory model

The model is striving to be a plausible representation of an Agent's episodic memory. As several psychology works have shown, nothing is less plausible than remembering too much. Furthermore, storing events exactly as they come tends to be very memory inefficient, if we decide to attempt some plausibility through ondemand episode distortion. Therefore, the plausibility goal is on par with memory efficiency, ie. storing as little data as possible. This is underlined by the builtin ability of the model to dump and restore its contents to and from a file. This ability, however, is not used by this work in particular.

The memory model is consisting of several elementary structures that work together. The basic skeleton revolves around AND-OR trees, a term used in related works for Agent planning, specifically [12]. The tree describes behaviour pattern trees, Agent action planning and decomposition in order to achieve desired goals.

Affordance is as an abstraction for objects decribing their fit for a particular use. It is based on a psychological concept initially used by Gibson[7], and has been used in the related works [4] and [14] for successfully planning actions with objects. It's synonymous for the object type, providing a particular use to the Agent, without the need to understand the meaning of the object.

Affordance Slot: *Affordance Slot is a tuple (Use, Affordance), describing the requirement of a task for any object providing a particular Affordance for a particular Use. This helps distinguish when multiple objects with the same Affordance are required for two different purposes.*

[1] The term 'feeding' will further refer to the bulk import.

AND-OR Tree: *AND-OR tree is a tree with root T, referred to as Top-level Goal, consisting of alternating levels of AND and OR nodes.*
AND node, also called Intention or Goal, describes decomposition of a task into one or more OR nodes. OR node, also called Action, can either be an Atomic action or an AND node. It describes the possible ways how an action can be satisfied.
For AND nodes, all child nodes are executed or attempted in order to satisfy the Goal.
For OR nodes, exactly one child node is executed to satisfy the Action. Another child is chosen in case the first one fails.
Each node can have any number of Affordance Slots, indicating the requirements for objects.

While itself fairly elementary and obviously limited in its planning scope, [12] has shown that it is satisfactory for the most uses where we require such a task. It has been successfully used in many related works, and so it made perfect sense to simply reuse it in the episode modelling.

The memory model stores information in a number of structures that provide the means to remember, guess missing details, and time-estimate a given episode.

3.1.1 Model structures

The model itself contains its own basic set of AND-OR trees (forest) called *Decision Tree*. This is the forest that will typically be used by the parent application making use of the memory model. However, this is only used for convenience and compatibility with these apps. The memory model makes no use of the planning aspects of the tree, merely uses the restriction of number of sub-nodes imposed on the nodes.

Episode: *Episode is an ordered set of decision nodes from a given AND-OR tree, describing a particular walk through the decision tree.*
An episode is considered finished, after its Top-level goal has finished.
An episode is considered failed, after its Top-level goal has failed, otherwise it is considered succeeded. This is typically after trying all possible ways of walking through the decision tree.

Episodes come directly from the Agent planning application. Whenever an Agent makes a decision about its behaviour, it reports this into the memory.

Chronobag: *Chronobag is a set of Episodes. A chronobag has a minimum and maximum age.*

Chronobag defines times of all episodes stored within, which themselves are not bound to a particular event time.

Chronology: *Chronology is a hierarchy of Chronobags for which the following applies:*
- Each chronobag has a level and age scope.
- A Chronobag of level N, has any number of children Chronobags of level N-1, such as that their age scope is contained in parent age scope.
- A Chronobag of level N has an age scope divisible by age scope of Chronobag of level N-1.

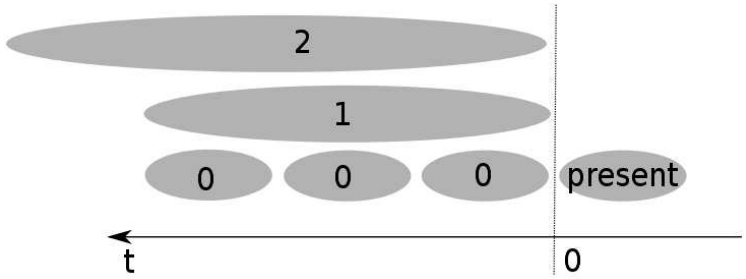


Figure 3.1: *An example of chronology. It contains the present Chronobag and a number of example levels.*

Chronology defines a partial ordering of Episodes contained within with variable Level of Detail. It forms a basic framework for managing uncertainty about an Episode's time placement. The level of detail depends on the hierarchy specification. In the memory model used in this work, each Chronobag level represents one commonly understood time schema, namely: Day, Week, Month, Year.

We can move episodes up a level in the hierarchy, removing it from one Chronobag and adding it to its parent. By this, the time information associated with the episode is permanently lost and becomes more fuzzy.

Note that the mutual congruentiality of Chronobag intervals restriction can be lifted, and so can be the non-overlapping of chronobags. However, this work wants to demonstrate, amongst others, that it can be used to represent hierarchical time concepts.

In this work, the time schemas are slightly distorted to assert congruentiality. A week has 7 days, as expected. However, months are 4 weeks, and a year is again normally 12 months, adding up to a total of 336 days, as opposed to the more familiar 365. This restriction is intentional and artificial and will serve its purpose later. For the time being, it is plain to see that even though real-world concepts are not always congruential, they do form individual non-overlapping hierarchies *(eg. days-weeks, days-months-years-decades), ...* and implementation-wise, we could have several side-by-side Chronologies describing different hierarchies. For the purpose of this work, that was an unnecessary complexity.

Counter: *A counter is a tuple (C, N), where C is a subset of any nodes or objects appearing in a Decision Tree and N is a positive integer.*

Schemabag: *A schemabag is a tuple (D, S), where D is a decision tree, and S is a set of counters.*

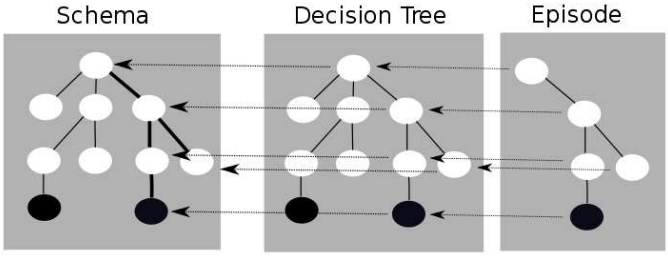


Figure 3.2: *An example structure of memories and their interconnection. Episode is a particular walk through the decision tree. Each episode node is connected to its associated node. The decision tree is connected to an associated node in the schema bag, which is roughly a usage counter of episodes that have been seen in the past.*

Schemabag is a structure used to describe collective schemas of all past episodes. It is based on the gist trace from the fuzzy-trace theory (FTT) proposed by Brainerd and Reyna[1]. After each episode happens, it is decomposed and recorded into the Schemabag. Since Schemabag contains the appearance counts of particular Node sets, the particular decomposition and recording of an Episode will have an impact on performance of the algorithm, this can be an individual parameter of each model, rather than a defined rule.

For the purpose of this model, we record each sub-combination of all Nodes appearing in the Episode as an individual count, up to a combination of 3, due to massive combinatorial explosion. This has already been done as part of the Baseline model. In addition to that, a count appears for the combination of all episode Nodes together, and all past complete tree traces that led to the creation of the episode.

Temporal Schemabag: *A temporal schema is a hierarchy of Schemabags, such as that for each Schemabag of level N, K[N] Schemabags of level N-1 exist. K is an arbitrary given distribution of repetitiveness.*

Temporal Schemabag represents a collection of Schemabags that provide schemas of episodes in a repetitive time scheme. As is obvious, the Schemabag hierarchy distribution in a Temporal Schemabag is identical to the Chronobag hierarchy distribution. It does not necessarily need to be, but it makes a lot of practical sense for later.

3.2 Corpora Generator

The episode corpora generator is a SHOP2-derived planner, based on the Monroe [13] emergency response planner. In accordance with general Planning notation, SHOP2 uses *Methods* or Goals to decompose a task at hand into smaller tasks, and *Operators* or Actions to perform an atomic action on the world state. It uses common non-procedural programming primitives such as Facts to describe knowledge.

3.2.1 Planning structures

World state: *The world state is a collection of facts.*

In planning, a state is simply an encoding of the world we're trying to plan. A plan is a Sequence of goals and actions that change state S1 into state S2. In contrast with that, Monroe planner uses the planner as a blunt tool to get his way. Actions, rather than change the state (though they still might do that), simply record the information that the particular Action has happened. This is in fact a strict requirement. As in any planning, Goals that don't decompose into any Actions are meaningless, empty and SHOP2 planner ensures they do not appear in the plan.

Method: *A Method is a tuple (Args, Conditions, SubGoals).*
Upon evaluation, Args are assigned values, and applied to any local variables that might appear in Conditions or SubGoals.
Conditions is a list of facts that need to be present in the state as a precondition for the Goal to be executed.
SubGoals is a list decomposing the Goal into individual items.
A method can have any amount of alternate definitions.

Operator: *An operator is a tuple (Args, Conditions, In, Out).*
Upon evaluation, identical to a Method, Args are evaluated, and Conditions will be checked.
In is a list of facts that will be removed from the World state.
Out is a list of facts that will be added to the World state.

Similar to Monroe, as a hack to the original purpose, this work uses the planner to generate random plans, rather than optimal plans. This is achieved by randomizing the SubGoal selection during planning, and accepting the first valid plan, rather than evaluating all plans and picking the optimum one. However, the randomization algorithm is changed.

For full Syntax and details, please refer to Appendix A.

3.2.2 Planning schemas

This work concerns itself with the time identifiability of Episodes and planning episodes in a plausible way. Therefore, just piling up a bunch of random plans as per Monroe is not likely to achieve a very good result. Furthermore, we need to have a certain sense of time flow for the episodes. Different episodes take different amounts of time, happen at different time of the day, etc. This calls for grouping Episodes into a higher-level schemas to ensure sane properties. As a perfect solution with the most flexibility, the whole corpora could be a single plan, containing all subsequent time schemas down to individual episodes. Unfortunately, as the time complexity of planning problem is exponential, this quickly becomes unmaintainable.

A solution to this is being analyzed later, restricting formal correctness but making planning viable from a practical perspective.

3.3 Planning vs. memory episodes

We've estabished that the memory model uses AND-OR trees as the basic structure for describing episodes. While planning also provides tree structures as output, there is one significant difference: In plans, every node is an AND node, decomposing the Goal to a number of SubGoals, all of which have to be satisfied. OR nodes only exist as implied, as every Method can have any number of alternate definitions. This makes the OR nodes invisible in the plans.

We perform a number of middle steps before feeding the corpora into the memory to ensure this will not be a problem.

4. Planner

The SHOP2 planner is an open source automated planner. It is an ordered task decomposition planner, that is an HTN planner that plans for tasks in the same order that they will be executed. Due to the opensource licence, it is commonly used for research projects, and has been used in hundreds of projects worldwide, according to the home page[17].

The Monroe planner, that has been used as a basis for Episode corpora planner is one of them, originally based on SHOP2. Designed by Nate Blaylock and James Allen[13], it was part of an exploratory work into corpora generation for large volume testing. In his case it was simulating events of an Emergency response domain, a world with accidents has been randomly generated, and a plan was created to clean the trouble up.

As discussed earlier, the task is very similar to the problem of generating Episode corpora, with a couple of important differences described later.

4.1 Statefulness

As discussed earlier, rather than individual episodes, we require that episodes are generated in batches that are part of a single plan, in order to fit the bigger picture. Considering the scope of an "Episode" as a particular walkthrough of an AND-OR tree, their duration being in minutes, up to hours, a single day has to consist of a a large number of episodes, and most of them are somehow tied to a particular schema of the day. For instance, it would be unusual to have Lunch at midnight (although it certainly happens in real life, our agent is too orderly for that), go to bed in the morning, or go shopping after opening hours.

The previous chapter noted, that creating a plan for a complete lifetime is not practically feasible for performance reasons. Due to the NP nature of planning, no matter how optimized a planner is, there is always a planning domain too large for it. Through series of experiments, we've estabished a single day to be generally be the highest order planning unit usable for practical purposes. This incidentally happens to be the computational unit of the episodic memory model, hence why it was solidly chosen as a planning basis. This allows for planning of some basic day-to-day activities. However, it does not allow to plan a certain day based on events of any previous day.

As a workaround for that, this work modified the Monroe planner to be able to preserve the world state in a variable, and re-initialize it on starting the next plan. This is effectively an implementation of a planner that creates plans by compositing subplans. While this is formally wrong and it's not possible to backtrack from one highest-level plan back to the previous one if something goes wrong, it matters very little for this work or the original Monroe planner. Obviously, the domains for Episode corpora are designed to offer a large variety of possible valid

subplans, as opposed to problems that are cleverly hiding an optimal solution in a large volume of invalid and sub-optimal plans.

Preserving the world state makes it possible to plan various day-to-day activities that are not strictly scheduled, rather happening on a *when it breaks, fix it* basis. For instance, nourishment happens several times a day and so does feeding. However, one does not always eat 3 times a day, sometimes the person is simply not hungry. By keeping a *(hungry)* fact in the world, removing it when fed, adding it randomly again after a certain time has passed, we can create a seemingly more complex behaviour than just three planned meals a day.

4.2 Time scheduling and complexity

Even though the day is the highest order planning unit, we sometimes desire other time concepts. Weeks, months, years, perhaps life phases (ie. acne starting in puberty). This is made possible by planning statefulness, and is demonstrated by the example domains provided by this work. We simply initialize our world with particular facts indicating a position inside a repetitive time schema, for example *(weekday monday)*. A set of methods and operators then switches these at the end of each plan, and preserves the resulting world state for the next planning. As time concepts are straightforward and all states are well defined, this poses no performance hit to the actual planning. It can only fail if the rest of the plan fails as well.

The sample domains provided by this work implement the following hierarchy of time concepts: days, weeks, months, seasons, years, life phases. Each of these can be tested for by any of the planning rules, allowing for fairly complex behavior constraints, such as: "If you're under 18, you have to walk to school because you don't own a car. After you turn 18, you can go buy one and start driving to school" or "Breastfeeding is only done by infants".

Note that in accordance with the artificial congruency restriction of the memory model, these concepts are mutually congruent in a similar manner, ie. months have 28 days, and years have 336 days. This is not a strict requirement, and the planning would handle itself exactly the same as it does with this restriction, however the time schemas of the generated corpora would be completely disruptive to any memory testing.

4.3 Randomisation and episode probability

The Monroe planner randomizes its goal selection in order to achieve a seemingly random plans. This works for that planner, but not for our Episodic planner. For example, it is possible to make an optional action by providing an empty method with no decomposition. If this method is selected, the SubGoal will be automatically satisfied with no action. However, suppose we don't want to make

the action too optional, such as eating. It's okay to skip a meal, but unhealthy to do this on a regular basis. This prompts for the implementation of weight-based rule randomisation.

This can be simulated in Monroe planner as well by placing identical methods into the domain. As the planner performs no automatic domain deduplication, it is simply going to increase the likelyhood of that variant of the method being selected. This becomes quickly unsustainable if you want to declare very rare, landmark episodes with a probability of less than 1:1000. Even if it were not terribly inconvenient to have to copy/paste the same episode 1000 times, there would be terrible performance implications past certain point.

The Episode planner implements a weighted roulette randomisation algorithm directly in the SHOP2 planner, and extends the syntax of the SHOP2 planning domains by the *:weight* modifier. This makes it feasible to declare Methods based on their weight compared to other alternatives of the same Method. This approach is both simple and semantically interchangeable for probability based deciding.

4.4 Memory feeder

A memory feeder acts as a connector of the memory model and the episode planner. It exists to resolve differences and provide data connection. The memory feeder parses lisp trees in the output format generated by Episode generator, transforms them into internal memory events, and inserts them through the standard interface.

4.4.1 Making output compatible with a memory model

We've estabished the necessity of high-level time concept planning, in particular, a day. While the underlying memory simulation also recognises days, it does not particularily care about them. The required input to the memory model is a sequence of Episodes. The time has to be advanced by external means of invoking memory reorganization. In order to facilitate for these requirements, we provide hidden Methods and Actions concept in the planner.

A hidden method or action is one whose name starts with an underscore. The point of hidden goals and actions is to encode metadata into the input data, that will not be explicitly seen by the model, but will be used by a feeding application to understand and take care of that context.

4.4.2 Scalability

Due to the requirement of processing large volumes of input data, the memory feeder cannot exercise creative freedom. Since the generation and processing of

events are two independent processes written in two different programming languages, it's essential that it attempts to put as little overhead in between them as possible. In particular, it parses the input data piece by piece rather than load it all into memory before processing it.

It also avoids having to process the domain on a multi-pass basis if a certain metadata about the domain are required in advance. In case of our model, it asks for the list of top-level goals during initialization. We work around this by defining a domain descriptor in a particular format, which is produced by a dedicated common lisp based tool operating directly with the lisp domains.

5. Memory model

The goal of this work was, amongst others, to share the structure of the memory model with the Master Thesis of Michal Cermak so that they can be experimented on and interchanged later. The major difference is that his work focuses on the shorter-term simulations and memory, while this work focuses on the long-term simulations. This implies many differences and internal changes. However, the skeleton, Java structures and inner workings are generally shared amongst these two to maintain certain degree of compatibility.

As for comparison of the code of the models, generally, this model features quite a bit of refactoring of the original model structure. The only major component of the model that has remained mostly unchanged are the decision trees, as they are not significantly relevant to this work. This model adds one level of abstraction to both Chronobags (Chronology) and Schemabags (Temporal schemas) and rewrite of some of the respective functions. These changes imply at least minor changes in most of the code as such. Also, major tweaks to the forgetting and remembering algorithms made sense in the context of hypothetical runtime of the model.

Appendix C describes the changes at code level in more detail. Also, the following text will point out and explain the major differences between the two, further denoted as *Baseline* and *Derived*.

Implementation details described are not always specific to this work and may already come from the baseline work. They are, however, a necessary part of the picture and will be described anyway.

5.1 Decision trees

Decision trees are a structure that gives form to all other basic structures, while not itself being used for the original purpose, that is deciding the agents activity. Its use is driven by the previous works on the topic of artificial agents[4]. Decision tree is a series of AND-OR trees each describing one possible top-level goal of the agent.

5.2 Episodes and Chronobags

As described earlier, the core structure of an episode is a trace of a decision tree. It contains the so called Episode nodes, which can be either goals, actions or atomic actions, associated with particular affordances.

For immediate storage, the model maintains a *Present* chronobag. This is the collection of current episodes that happened today. The present chronobag always exists. During the memory reorganization, the Present chronobag is promoted to the most recent level 0 past chronobag, and the age of the rest of the

hierarchy is increased by one day.

The process of memorizing is simply storing episodes into the present chronobag as they come. No modifications whatsoever are applied to these episodes. They become memories when the present chronobag is moved into the past.

An episode is internally time ordered. While nodes are not stored in a sequence anywhere, each node has either zero or one child node, or the child nodes itself are ordered via pointers.

While each episode is a standalone structure, all of its nodes map to exactly one node of the decision tree. For convenience, they also contain an explicit link to them.

Chronobags each contain a list of episodes that happened in the given time frame. They have a starting and ending age. The main ability of a chronobag is to forget an episode it is holding, and transferring it to its immediate parent. They are connected together in a tree-like structure so that standard graph algorithms can be used to traverse them and search for contained episodes.

5.3 Schemas

The memory stores statistical information about the incoming episodes using Schemabag counters, based on FTT[1], as described earlier. Whenever a new episode happens, it is deconstructed into smaller subsets and inserted into Schemabags in the appropriate Temporal schemas. For each episode, the following Counters are incremented:

1. All individual subtraces starting from the root.
2. A complete episode.
3. All subsets of all of the above.

All this is done for exactly one schemabag at each temporal schema level depending on the current day, including the level 0 (global) one.

Ideally, the combinatorial approach is the optimal one. For each set of nodes, all possible subsets are incremented. This, however, runs into practical limitations. The highest value that is feasible are all subtriplets, any higher value starts a combinatorial explosion both for memory and time complexity of the learning process.

Assuming that for all past episodes, all node subsets of size 2 are added each time. Then, given a single node from an episode, schemas form a *Markov chain* of the most common paths. For all triplets, we get second order *Markov chains*.

The baseline work only includes the combinatorial approach, and even in its limited scope seems to have had success with that demonstrating False memories in action. This work implemented the addition of the more static approach, and partially limited the use of the combinatoric algorithms to increase performance for long-term usage.

Internally, schemabags have the same structure and nodes as a decision tree. The only reason there are separate structures called schema nodes is because for schemas, we retain different meta information than for the decision tree. Each node in the decision tree has a link to its schema node counterpart.

In addition to schema nodes, schema bags also contain individual object nodes. While the decision tree contains Affordance slots, in schemas, these slots are filled by a particular object providing that affordance, similar to the Episodes.

5.3.1 Episode reconstruction

Using the information stored in schema bags, the model can make certain assumptions about incomplete episodes. Shemas represent what the agent has seen in the past, and can make suggestion on what the next step would be. Given the nodes in an existing episode, it's easy to determine the link to an associated schema node, revealing all the important information about the node in question.

Each episode has a fairly limited number of variations, given a static Decision Tree. Assuming that we have an episode that is partially forgotten, it is possible to determine the most likely node K to be added into the episode, given current schemas.
For set of nodes K, that are not already part of the episode nodes E, and a node L exists such as that L is a parent of K and $L \in E$, For all schema counters (S_m, N_m) such as that S_m is a subset of $E \cup K$, pick i, such as that $N_i = max\{N_m\}$.
Following this, K_i is the most likely node to be added. By iterating this algorithm and adding nodes into the episode until no more nodes can be added (K is an empty set), we can reconstruct the whole episode according to what the schema suggests the episode originally was.

This algorithm can also take a number of episode *cues* and attempt to guess an episode from them, regardless of whether the episode has happened or is remembered. This would be equivalent to asking the model a question of the form: *What were you most likely doing when you used these nodes together?* The answer to that may be extremely ambiguous or very specific, depending on the number of cues given.

The idea of reconstructing episodes comes from work concerning Fuzzy-trace theory[1]. The research examines how the human memory can very vividly remember memories that never happened. Used particulary in the analysis of Eyewitness testimony, injecting *False memories* that did not ever happen is the

primary focus of baseline work of Michal Cermak, and therefore this thesis does not examine this aspect in detail.

5.3.2 Temporal schemas

While a single schemabag can be used to find out what combinations of nodes are most often used together, a Temporal Schemabag can provide temporal context to these searches. Suppose we have an episode in a chronobag of level N other than 0. For each level between 1 and N, Temporal schemabag provides an estimate in which of the children of the given chronobag is the episode most likely to reside in.

Conversely, if the complete temporal information of the episode is known, the particular Temporal Schemabag being pointed to can provide a more accurate estimate of the episode contents based on known time.

While theoretically, temporal schemas are a separate schemabag, practically, they are implemented by extending the schema counters structure to be a hierarchy instead. This makes it simpler for the rest of the model, as it is still possible to reach all temporal schemas using the same schema node and its associated decision tree node.

The schemas feature an internal day counter. The counter is incremented whenever memory reorganisation occurs, and when episodes are recorded into the schema, the particular temporal schemas where the episode should be recorded are determined using a modulo function given the appropriate temporal schema periodicity.

Temporal schemas are only implemented and used in this model, not the baseline.

5.4 Retention and forgetting

Plausible episode retention is an essential task peformed by the Chronology hierarchy. The findings that this is based on are many, notably in this work Autobiographical memories[3] and Wagenaar's self-study[5], as retention quantification has always been an important topic in psychology research. The most important takeaway to retention and forgetting is that there is likely no single mechanism performing this task[1]. We remember and forget in many different ways which may be subconsciously confused for one another, and are only uncovered using experimental research.

The chronology hierarchy provides a *Level of Detail* index of all the episodes stored within, and can be searched using standard tree algorithms for episodes of matching criteria. It allows searching for episodes within a certain timeframe,

equal to the size of a chronobag of a given level.

The model does not implement any Query interface that would provide answers to questions. Testing and verification of this can be done by simply reaching directly into the chronobag and retreiving the necessary structures.

5.4.1 Score based forgetting

One desired effect of a memory model is emphasizing important memories over unimportant memories, as we do not want to simply forget everything based on a curve. Research into memory importance shows that people are doing a very good job of distinguishing precisely the important kind of memories to remember and forgetting the ones that are useless.

When forgetting, each node of an episode is evaluated individually. This allows for continual and partial forgetting of details of an episode, until the whole episode is lost completely. Each node is assigned a score that reflects its percieved importance. Several attributes are added together to form the final score:

1. Node reconstructability for nodes in the present chronobag.
2. Number of times the node appears in a schema.
3. Number of times the node used to appear in a schema at the time of creation.
4. Score of the object nodes associated with the given node, determined in a similar way.

The formula also contains attractivity of a given node, which is generally a hardcoded value when initializing the model, and is only used by the baseline.

The score generally reflects how important an episode node is. When it comes to forgetting, we take all nodes and the ones that do not surpass a maximum score of the given chronobag are forgotten.

We implement two different methods of forgetting memories, which contribute together to the result:

5.4.2 Time based forgetting

The most obvious and elementary way to forget memories is fading away with time. This concept has been explored by many studies. In the memory model, this is performed by a progression that sets the ceiling for number of nodes/episodes that are contained in a chronobag of a particular age. The general formula for the volume of past memories given by Wagenaar[5] or Rubin[6] is a power function, decreasing with time:

$R = a * t^{-b}$ The constants cited in Wagenaar[5] are $a = 0.54$ and $b = 0.36$, however, different sources found different constants using other memorizing techniques and even these are irrelevant to this model, used only as a reference we can compare to. However, the power function itself is of vital importance and drives this model forgetting. Chronobags of each level have parameters in place of constants a and b, allowing for experiments with the retention. This is an important difference from the baseline model that used a linear forgetting function.

The power function defines the K quotient that is used to determine the maximum chronobag score. As noted above, all nodes that do not surpass this value are forgotten, generally trimming down the contents of the chronobag until it's empty. This directly influences how many memories stay, and how many go.

5.4.3 Schema based forgetting

The schema counters are part of the node scoring function, both the count at the time of the node creation and the current count of the same node. For this purpose, single node counters are used. This directly influences in how likey a node is to be forgotten.

Furthermore, some of the memory research, specifically False memories[1] explores a secondary forgetting mechanism. Instead of forgetting memories continually, we simply do not remember them at all as they can be easilly reconstructed from the current world schema. Concerning the memory model, this is implemented using Schemabags.

For each episode exiting the present chronobag, we attempt to remove episode nodes, ensuring that using the current schema, the episode can be correctly reconstructed. This is called derivability by the baseline model and described above, and contributes to the score of a node.

The related research seems to indicate that memories that we hide in this particular way are in reality never even remembered. This is typically some mundane details in a very common scenario, for example a color of a lamp on someone's desk that will automatically be assumed to be the same color as the lamp on some other desk that the viewer has been seeing repeatedly. For that reason, it seems to only make sense to perform this on present memories, before they enter the past chronobags.

As a significant difference, the baseline memory model did perform this as part of a common forgetting function. As memory reconstruction is a fairly computationally restrictive process, this limited performance of the model severely, and that is also one of the reasons it was changed in this work. As described earlier, it also made sense to perform this on present chronobags only.

5.4.4 Landmarks

Some episodes of particular importance are shielded from forgetting, these are called landmark episodes[8]. Typically, these will be episodes that happen once or twice in a lifetime. These memories are typically not just unique, but sometimes even emotionally charged. The mechanism through which these are remembered seemed to be slightly different, as even after a very long time, it is possible to recall the full detail of them in vivid detail.

This memory model does not use the way to indicate emotional importance of an episode (attractiveness), therefore it has little chance of reconstructing the phenomenon. An attempt was made in the baseline model. In particular, it proposed that whole chronobags be turned landmark, after they have passed a certain age threshold. This means they had large enough score to survive the retention function for long enough. As a reward, nothing is ever forgotten from these again. One downside of this approach is that obviously, this may also include some memories that are simply not landmarks. This mechanism also tends to distort other tests related to retention, so it has been sharply limited in this memory model.

5.4.5 Forgetting and chronobag levels

It is apparent, that the mechanics applying to a chronobag at one level should not apply to one at a different level. They technically serve the same purpose, but the volume of information stored therein is vastly different. For instance, we expect that a chronobag at level 0 will inherently disappear after a fairly short time, regardless of what it ever contained. Chronobags of higher levels should have higher ability to retain the information for a longer time period. This can be done in many ways.

The baseline model implements a score computation of a chronobag depending on the score of children chronobags. This theoretically ensures a to-scale inflating of higher level bags.

This work defines time as a function of chronobag level, slowing down timeflow for higher-order chronobags. That has the potential to retain the same properties of a lower-level chronobag but scale them to a longer time periods. For corrections, the power function is parametrised based on chronobag level.

5.5 Memory parameters

As the model is an experiment whose properties we explore, it depends on a fairly large number of tweakable parameters. These can influence all aspects of the memory and their particular values will be discussed later.

5.5.1 Chronobag and schema sizes

The hierarchy of chronobags is defined by their respective interval lengths. These were already present in the baseline model, but have been changed here. In particular, the original model was constructing the hierarchy in a very different way with non-congruential levels and overlapping chronobags on all levels but 0.

To simplify the problem, these have been changed and fixated in this work, as the same time schemas also need to be solidly defined in the episode generator.

5.5.2 Retention function

The retention power function discussed earlier has the most significant impact on the amount of retained memories. Choosing the wrong quotient can lead to garbage results, no forgetting and other negative effects. In order to evaluate it, we distinguish the power quotient for different chronobag levels.

5.5.3 Chronobag capacity

The size of a chronobag in this model is measured in episode nodes. The reasoning behind that is in psychological research that suggests that weaker memories in the same context can sometimes go and make space for stronger memories. Being able to slowly fade episodes to preserve higher-scored ones is one of the desired abilities of the model. This parameter governs the basic property of being able to store certain amount of nodes into the chronobag.

5.5.4 Debug parameters

A number of debugging parameters are inherited from the baseline model and can be used to tweak features that should not be part of actual experiments. Examples include turning off memories forgetting entirely, no remembering of objects or disabling higher level chronobags altogether. These parameters have mostly been ignored by this work.

6. Validation and tests

Evaluating the results of the experimentation with the model are the most important part of this work. It provides both an episode generator and a memory model adapted to work with it. A system of scripts and Makefiles has been devised to easilly script creation of episode corpora and feeding them into the memory of particular parameters. The following tests attempt to shed some light on the properties of the combination.

It should be noted, that as part of the validation of the memory model, in addition to own tests, the original tests that were already present in the Baseline model, specifically the tests for demonstrating False Memories capability were rerun with similar input conditions to make sure the original model properties still hold. The only modifications to that were to accomodate for the codebase structural changes, and as such should not be considered part of this work or the results of that reported in detail. This mention is only included for completeness sake.

6.1 Memory model

6.1.1 Temporal estimation of episodes

Description

Temporal schemas is a concept introduced by Larsen[9] for remembering additional information about repetitive time schemas like weeks, months, years. He peformed a very large experiment in which large groups of people attempt to remember events that were recorded along with actual time when they happened using the diary method.

It determines precision of the measurements by comparing remembered and actual time of an episode. It measures a variable called *dating error*, a number of recalled episodes as a function of time offset of the recalled and actual time. Larsen notes that for any results to be sensible, the episodes have to be *highly predictable and personal*, otherwise the error distribution is normal.

The result of the experiment is a graph describing peaks in dating errors that are a multiples of particular time concept. Both weeks and months are demonstrated. This implies that there is a mechanism to schematize time concepts and use them as additional subconscious information when recalling episodes, and is one example use of the semantic memory to compress episodic memory.

In a practical setting for our purpose, it is obvious we cannot be able to reproduce the experiment as such. First and foremost, we don't really have a definition of a *highly personal* episode, as opposed to mundane *impersonal* episode. Everything the agent lives is equally thrilling and important. However, the episode

planner allows to define a form of time predictability.

Conduct

We have taken a fairly large and repetitive domain that has been used as a background noise for most experiments. In addition to it, we've added rules for episodes that only happen on particular days of the week. In particular:

```
(:method :weight 60 (_ExtensionGoal)
    empty
    ((day sun))
    ((SunGoal))
)
(:method :weight 1 (_ExtensionGoal)
    () ; Occasionally do it even on not-sunday
    ((SunGoal))
)
(:method (SunGoal)
    ()
    ((SunFill))
)
(:method (SunFill)
    ()
    ((!sunaction))
)
(:operator (!sunaction)
    () () ()
)
```

This goal was also repeated for a randomized wednesday and tuesday actions, each with its own specific probability of appearance. The *ExtensionGoal* is an optional goal that is part of the basic domain, that is repeated multiple times a day to allow for *hooks* into the dependency tree. Note that according to the above code, the schema specific goal does not always happen on a specific day, but with a much lower chance it can happen on any other day.

We cannot ever reach the volume of data of the original experiment as it was conducted using a large group of select people representing the population with significant variations between individuals. While we can repeat this on a number of independent simulations, this is unlikely to improve the data quality, rather just increase the scale and amount of noise. Furthermore, regardless of the volume of input data, it is quite sensible that a large part of the time specific Episodes are going to be forgotten, so our testing has to account for that.

As a solution, instead of measuring the amount of hits for each date offset, we simply determine the exact probability distribution of an episode appearing in a given day slot for the given chronobag in which the episode is currently stored. It

is arguably identical to the approach of attempting to find an episode and dating it. Given an episode and performing its recall, we'd have to answer the question *when* by selecting a date inside the specified chronobag more or less randomly based on the probability. Iterated over a large volume of episodes (which we don't have), this would outline the resulting distribution exactly anyway. The probabilistic approach accounts for an answer to the question that is fuzzy at the specified chronobag level, ie. *sometime this year* becomes an answer.

We enumerate the probability distribution for each temporal schema, split the episode into smaller pieces according to the distribution and add these together to form a function.

Furthermore, we require to measure the exact age of the episode, which is an information the chronology hierarchy is trying to hide. For that purpose, we added a debug value to all Episodes that stores their creation age exactly. This value, however, is not used for anything but the final comparison.

Results

The test was conducted on a two sets of episodes, our select episodes that follow a time schema and the rest, both using temporal schema recognition and plain recognition, which distributes the parts of an episode across a chronobag interval equally distributed.

For the results to make any sense at all, we have to ensure that there's a number of episodes at all chronobag levels. This done through a specific printout of all the episodes, levels and their real ages and verified manually after the experiment.

On figure6.1 we can observe that determining episode age simply by belonging to a particular Chronobag gives us a fairly regular distribution, where the items belonging to a daily chronobag are dated exactly and giving a large peak at the zero point, while items belonging to higher level chronobags are divided across the respective interval. The slight centering defect is probably caused by uneven distribution of episodes inside the chronobag itself, specifically more towards the edges, and therefore contributing more to a particular side of the curve.

When using temporal schemas, 6.2 shows that suddenly, a lot of the recognition for episodes that were previously evened out are now peaking repetively with an offset of exactly 1 week, but the distribution that was previously spread to the whole interval now clearly points at a particular day of the week. While the noise in the month schema is mostly random as the measured episodes did not belong to a month schema, we can see some small non-uniform distribution in the data as well. The main peak remains at 0, as a number of episodes are simply known exactly. This generally mimics a figure given by Larsen's[9] figure 5.3, showing the peaks of dating errors at the edge of weeks.

For comparison, the same test has been conducted using the episodes that were not scheduled in any time schemas and appear distributed evenly. Figures 6.3 and 6.4 show that there is no major difference in recognition of these, whether temporal schemas are used or not. This is not only due to the increased scale and volume of the schema-unspecific episodes. Even though we see some very vaguely uneven distribution of individual episodes in 6.4, it does not seem to follow a particular time schema, and can be considered to be just noise.

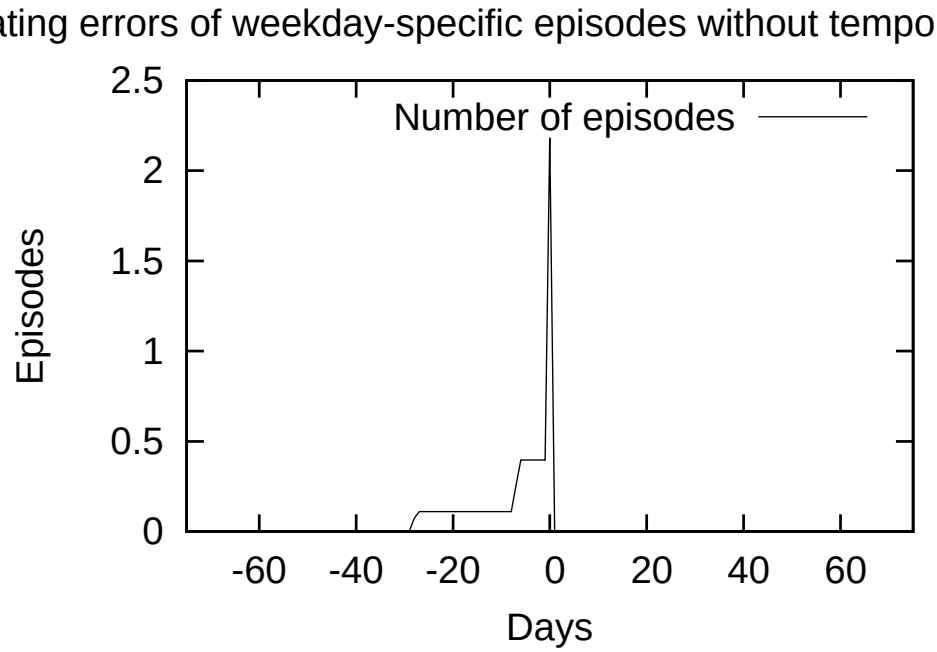


Figure 6.1:

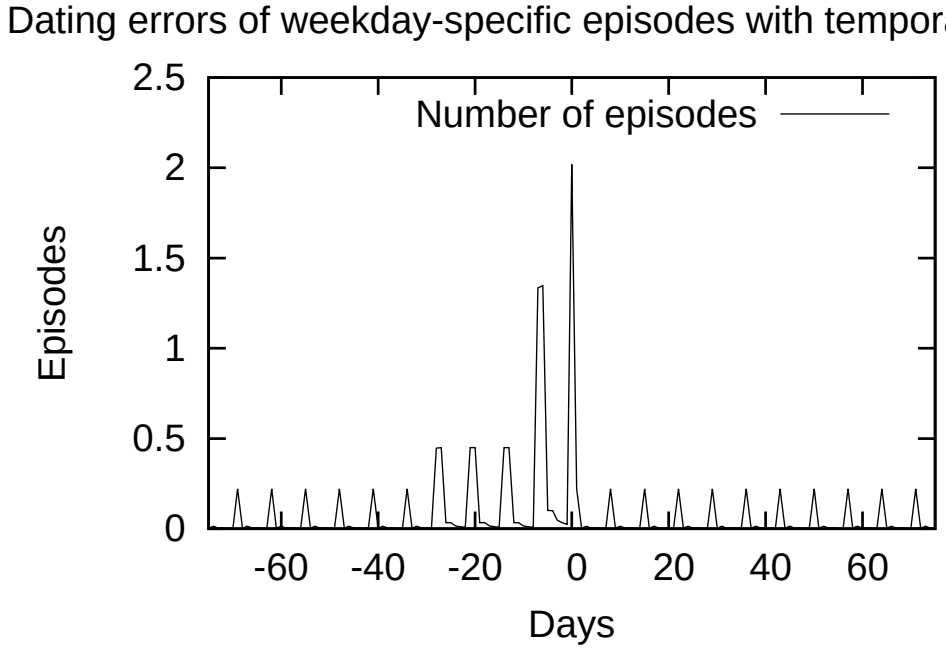


Figure 6.2:

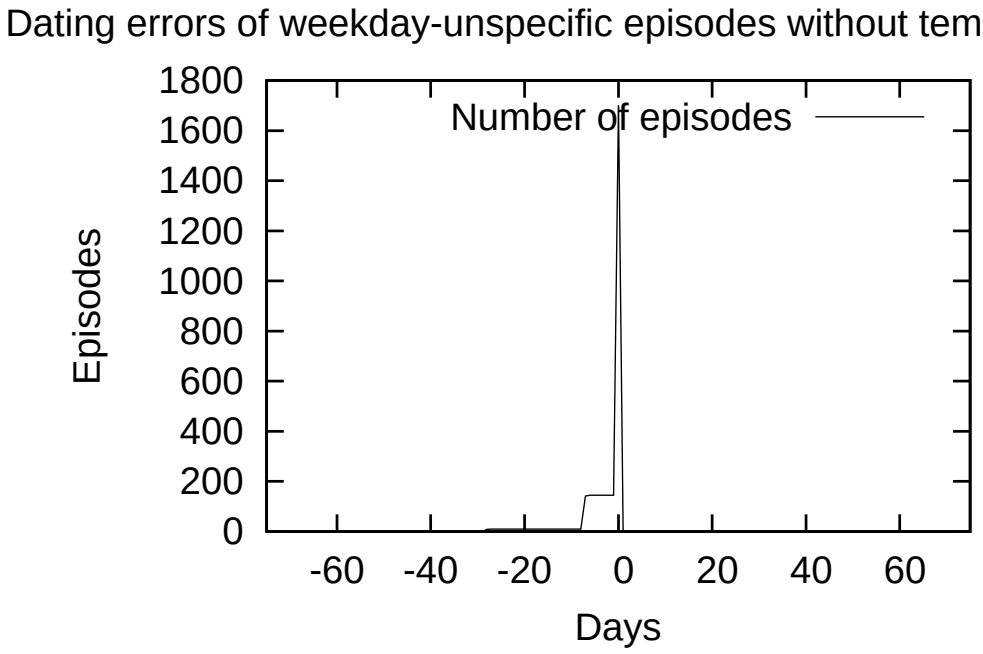


Figure 6.3:

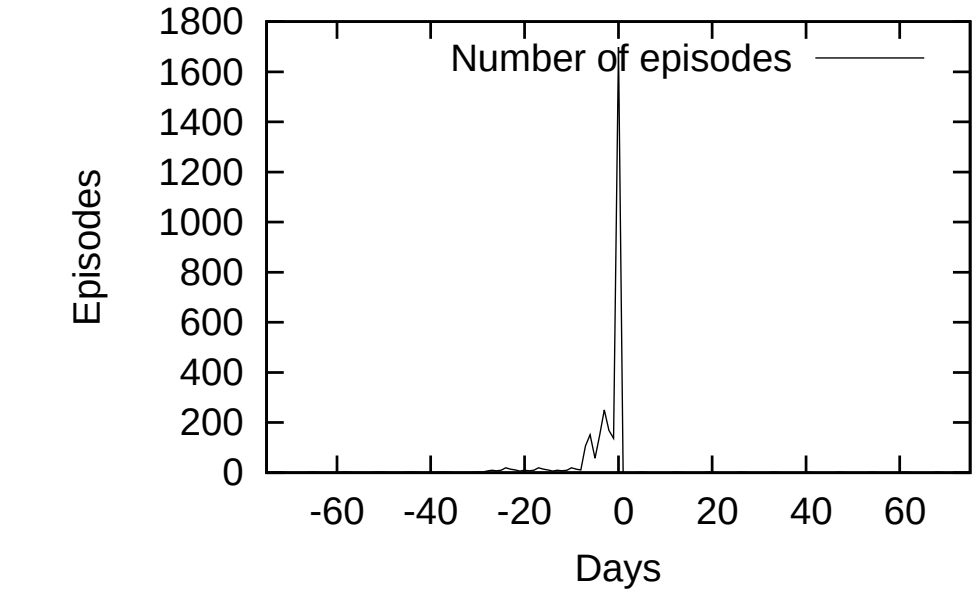


Figure 6.4:

6.1.2 Total memory volume after simulation

Description

Retention curve is the volume of episodes or information retained at specific age of memories, measured at present time. It's one of the more important aspects of the memory model, providing direct information about how much information is really retained.

Much research has been done in that topic, notably *Autobiographical memories*[3], which give closer insight not only into how information is structured but also how much of it is retained. The retention curve in practical measured cases is a complex curve formed by application of several psychological phenomena and is described in *Autobiographical memories*[3] in detail.

The start of the curve is *Childhood amnesia* where most people don't actually remember anything at all. This phenomenon has been originally described by Freud. Just past that, childhood tends to be heavilly populated by creation of new semantic concepts, or in our memory model terms, schemas, seeing a fairly large increase of remembered memories. The suggested interpretation is that we simply remember more information about episodes we're seeing for the first time and only just learning. That is in accordance with Pillemer's *Memories of College*[10], which finds a surprising increase of retained education-related memories at the beginning and end of each college year.

Further described by [3] is the so called *Reminiscence bump*. That is a hump in the curve around 40 years of age. The explanation is not known, but alternate theories suggest midlife *consolidation of the self* related to psychosocial development, reaching much further than this work intends to.

A retention curve of a single episode is a curve describing the decline of rememberability of a particular episode as a functon of time. This is widely believed to be a power function that was already described earlier, $y = a * x^{-b}$. The exact function is taken from Wagenaar[5], but has been independently shown by other research works. An example that we're particularily looking at is given by Larsen[3].

Conduct

As is clearly apparent, many of the concepts are hard to achieve without an actual personality. While the model does indeed forget more memories that are described by an existing schema, the effect of this should not be overestimated. The schemas are created very swiftly and the positive reinforcement of these has merely preferential effect on these in case conflicting schemas exist. Some degree of consolation to this problem could be given by using a different reinforcement function than linear, as discussed in the Future Work section, however, this is not addressed in this model.

For that reason, the result of measurements is expected to be rather flat with a rise in the most recent history. This is because in the long term, chronobags at each hierarchy level are expected to get saturated and lose old information as fast as they gain new information, forming a complex queue of variable length (depending on the chronobag level). Of course, the expected queue length for the highest level chronobags spans the whole lifetime of an agent, so certain amount of information should be retained forever. There's also the landmark threshold parameter that allows memories to be *fossilised* and stored forever.

As a shady workaround for the curve being too flat, we could use a retention modifier function that would change in time and cause the model to copy the retention curve given by cited papers. However, the plausibility of that would be highly debatable. We'd expect that the memory is going to remember more of new episodes and episodes seen for the first time ever, while with this correction, it would simply remember larger volume of the same structure of memories, giving no significant result.

This expriment is thus more evaluative than demonstrative. We conduct it on various time scales to see how the curve really progresses over time. However, at the very least we expect some basic properties:

1. There should be a significant peak of generally constant number of last days, demonstrating larger volume of fresh memories that will fade away with time.
2. The amount of memories retained should loosely follow the power curve, or to be exact, a composite of all the power curves, given different parameters for each chronobag level.
3. During the whole lifetime, there should be memories, ie. the model itself should not act as a queue buffer for last N episodes, even though the fundamental workings on individual chronobags are expected to be that.

In order to evaluate the curve, we plot a function of amount of episodes for the last N days in each chronobag equally distributed across the whole chronobag interval, all added together in a function of time. Note that the data is taken as a snapshot of the last N days at various time points, so the overlapping part of the data should share similar properties but not be completely identical. If a snapshot is taken at for 10 days and then 100 days, the original 10 days will still be present in the 100 days dataset, but as the first 10 days, significantly faded at the end of the dataset. Because of the nature of the data, it makes sense to plot on a log scale for age axis.

This approach again differs from what a precise recall mechanism might do, but should be representative enough.

Results

The experiment has been conducted repeatedly a large number of times, with different parameters as part of evaluating the model. One thing it did demonstrate is that the memory model is very volatile with respect to parameters. It is very easy to tip the balance and get completely garbage results.

Achieving some meaningful data has been difficult, but fairly sensible (attached) data were as a result of these parameters:
$A = 0.5$
$B[0] = 1.2$
$B[1] = 0.7$
$B[2] = 0.5$
$B[3] = 0.4$

Figures 6.5 through 6.8 show the retention curves of the last N days for a varying N. We can note that the memory model expectedly remembers a larger amount of information in the near past, and after certain point, regresses to a relatively stable amount of memories on a time unit. The falloff is fairly slow and generally, the memory retains episodes scattered throughout the whole time period, no matter how long.

Specifically, for the short term figures, 6.5 and 6.6, we can see the period when daily chronobags retain most information. Past that, higher level chronobags take over, but past age around 50, the general volume of retained episodes begins dropping. 6.8 shows that for a very distant past, the agent remembers about 1/5th of the fresh episodes, furthemore, in a distorted way, as anything stored at that age may only reside in a yearly chronobag.

This generally plays well with the expectation that the resulting curve is a composite of four power functions of the given parameters. Naturally, the exact volume of episodes that should be retained by a human being are not known, and even if they were, they would depend on both the nature of the memories and certain physiological factors.

The data provided by the planner are still uniform to a certain degree, and it is generally both sensible and supported by research like *Memories of College*[10] that the human brain will phase out episodes that are becoming too repetitive, and the actual memories are made more of events that are semantically novel. In that sense, the resulting falloff in the retention curves could be even higher, but this depends on what the desired behaviour and application for the model would be. This test merely demonstrates that sensible values are achievable. By tweaking the parameter A, it is possible to achieve overall reduction of memories and scale the curves down without changing its nature dramatically.

The distribution of the episodes across various chronobag levels is examined by the next test.

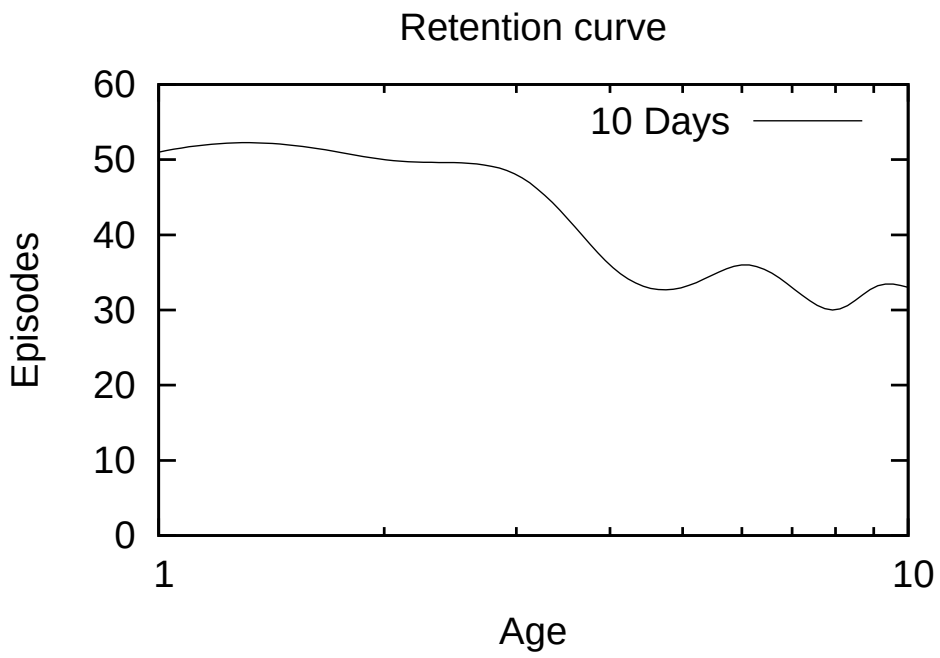


Figure 6.5:

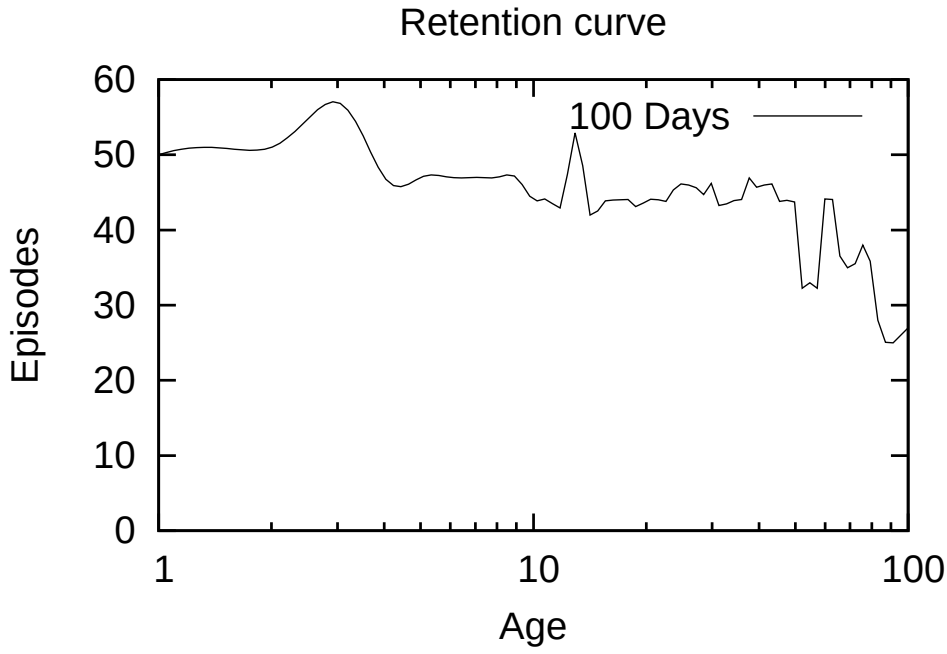


Figure 6.6:

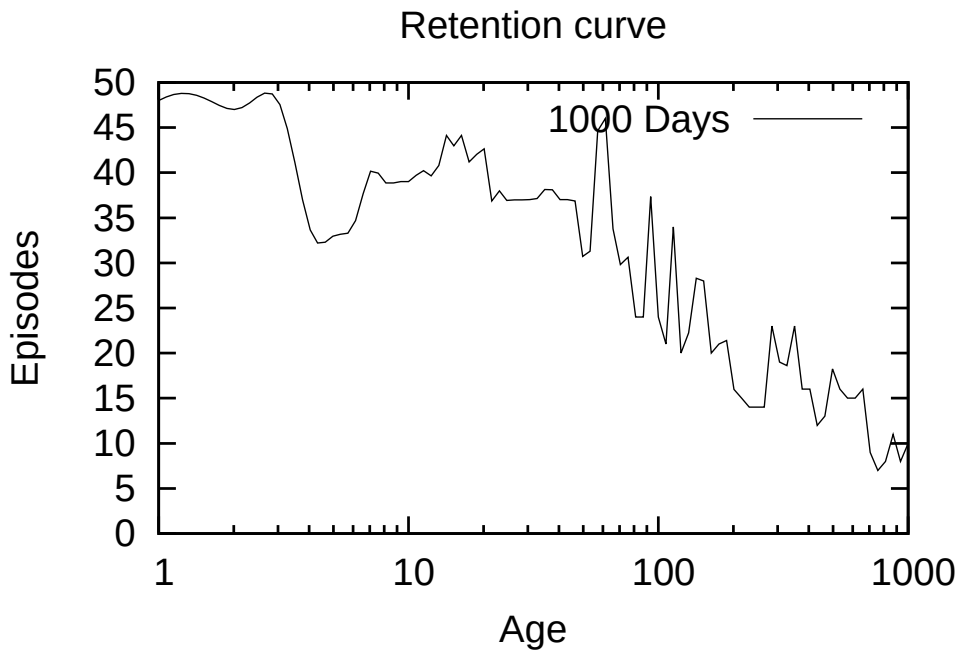


Figure 6.7:

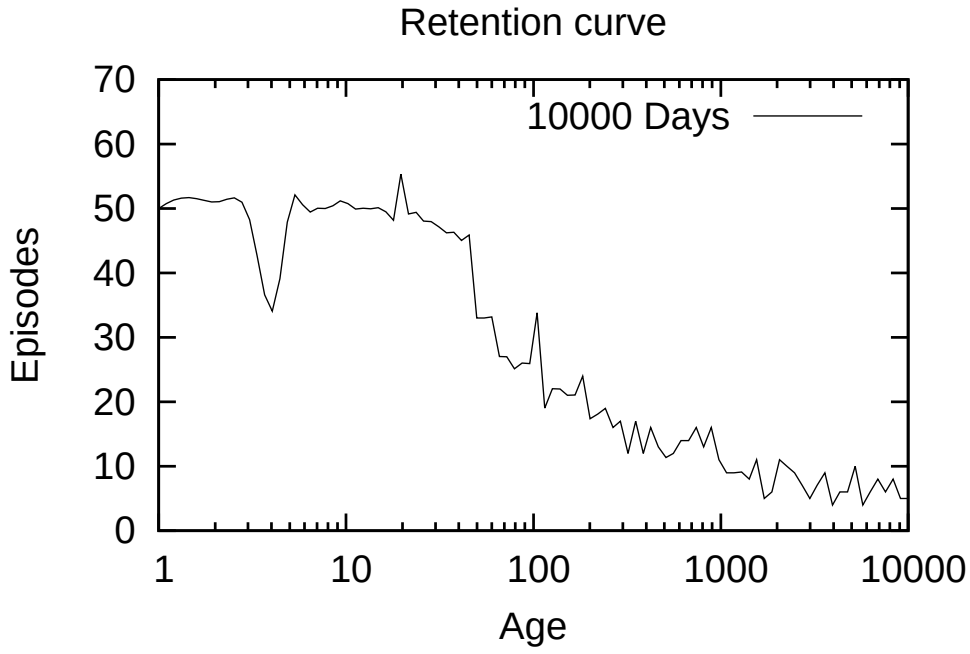


Figure 6.8:

6.1.3 Total Volume progression

Description

As important as how much information is stored is the level of detail of the retained memories, represented by membership in a chronobag. Clearly, a model that would remember episodes only at a particular level would not be much use, as we expect to have a level-of-detail function for the retained episodes.

An analogy to reality can be assumed to be same level of recognition precision on a log time scale, ie. time information compressing exponentially into the past. This is represented by the model's chronobag hierarchy, where transcending an episode into a higher-level chronobag increases the fuzziness of time information by an order of magnitude. This is, of course, just a speculation, but one we have made while proposing the model.

Contrary to the retention curve, we generally examine more present volume information as a function of time, as opposed to the age distribution of this information. There are several interesting results, namely:

1. Volume of information retained in chronobags of a particular level.
2. Ratio of episodes that happened in the input corpora vs. the information remembered.
3. Volume progression retained at age N of each chronobag level.

The first result describes how the chronobags of particular level cope with the inflow of information. Note that the number of chronobags is increased every day, and quite sensibly, so may the overall amount of stored information.

Second result gives us a rough idea of how much does the agent remember of what it saw. This can be thought of as the compression ratio. This is generally a composite of all values from first result across all chronobag levels, compared to the total number of episodes that happened in the corpora.

We expect the total number of episodes to be growing at a constant pace, while the growth of information retained should be very slightly slowing down, slowly improving the compression ratio over time.

Last result gives an insight to how the power function arguments are working for each individual chronobags. In a simple terms, we expect the chronobags to contain everything that happened at a present time, slowly losing this information according to the power function. After certain point, they should become empty, with the exception of Landmark memories. Each chronobag level should have very similar properties, scaled up by an order of magnitude from the previous chronobag.

Conduct

The goal of this is again to experiment with various memory model parameters and determine some sensible values. We collect the simple information in a large array during simulation, and plot it afterwards to provide the details. We also attempt to fit the data to a particular form.

We expect the memory volume progression of a particular chronobag level to be fairly steep at first, but then slow down to either a very slow pace or a near halt. Figure 6.9 attempts to describe the theoretical volume progression for an individual level, based on parameters.

The phase denoted as "init" is most influenced by parameter A, which is the quotient from the power function. By multiplying the power function, we can generally influence the size of the initial peak and therefore the amount of information ultimately stored within.

The "saturation" phase is expected to be mostly flat, but can have a very little hump on top. It is the phase when all chronobags of the given level already contain as much information as they can, and when new information is added, an equal amount of information is forgotten at the opposite end. The exponent B of the power function has the primary role in this case. It not only affects how long will the saturation take, but also, indirectly, affects the height of the initial peak, reinforcing the function of the parameter A.

The "sustain" phase is usually expected to be flat. However, the model contains a parameter C, called Landmark threshold. After an episode passes this threshold, it can no longer be forgotten. That means, given a proper value of this parameter, we can take the few remaining episodes at the end of the history of the given level and preserve them, causing a permanent long term rise in the described function. Obviously, the required value of the parameter C greatly depends on the volume of information stored in the first place, and this parameter should only be changed once both A and B have been fixed to a static value, as a too low value of C can cause the forgetting process to simply not work.

Results

Again, testing has shown how unstable the model can be with its parameters. Specifically, changing the power function of individual levels, one of the two things happens:

1. Higher level chronobag starvation happens if the power constant is too low. The given level chronobag tends to not forget enough episodes, which are then not evicted into a higher level chronobag.

2. Lower level chronobag flushing if the power constant is too high, and most episodes are immediately forgotten from level 0 chronobags.

Achieving and maintaining a balance tends to be rather difficult, one has to not only find the proper parameters, but also change them in tandem. Specifically for the power curve constants, they have the ability to scale the volume of memories as a whole, but they all have to be changed by the same quotient.

The parameters in this practical result have been iteratively fitted to provide the resulting curve. Starting from the lowest level chronobags, the parameters were fixed before moving on to experimenting with higher level chronobags. This is possible, as changing higher level chronobag parameters has no effect on lower level chronobags (but not vice versa).

The following parameters were determined to be usable, and used for the resulting data:
$A = 1.1$
$B[0] = 1.7$
$B[1] = 1.0$
$B[2] = 0.5$
$B[3] = 0.2$

Figure 6.10 shows the real volume information of particular chronobags dated by day of simulation. This indicates that the daily chronobags eventually settle down on a certain volume of information as expected. Higher level chronobags seem to retain slightly too much information, rising permanently after a certain point, though it should be noted the rise is optically high due to the log scale. This still seems somewhat sensible, as we do expect to retain certain amount of information permanently, with a diminishing amount of detail and time scale, which is precisely what happens.

For the second result, figure 6.11 indicates an ever improving compression ratio of input episodes, as expected. Around *day 100*, the compression ratio is *1:5*, while at *day 10000*, the compression rises up to *1:50*. This outlines the effectiveness of the model as a Level of Detail storage for data in the input episode format.

Finally, figure 6.12 shows distribution of the information the past 10000 days. Clearly, we can see that information stored in daily chronobags only reach as far as 70 days of age before they are either forgotten or moved to a higher chronobag level. Monthly and weekly chronobags follow a similar distribution except significantly lagging behind daily. This is expected, as they should reach further into the history by an order of magnitude each. This also shows that there is a small amount of residual information stored in each of those two even after the whole simulation. This can be due to the landmark threshold being too low, but can also be simply a consequence of the power function not falling off nearly so quickly at that level. We can spot the characteristic *discretisation* of the higher-level chronobags, as they represent a large age interval with a single value. It should be noted that for yearly chronobags, even the longest simulation only contains a handful of them, resulting in graphical display of the data being skewed and less descriptive.

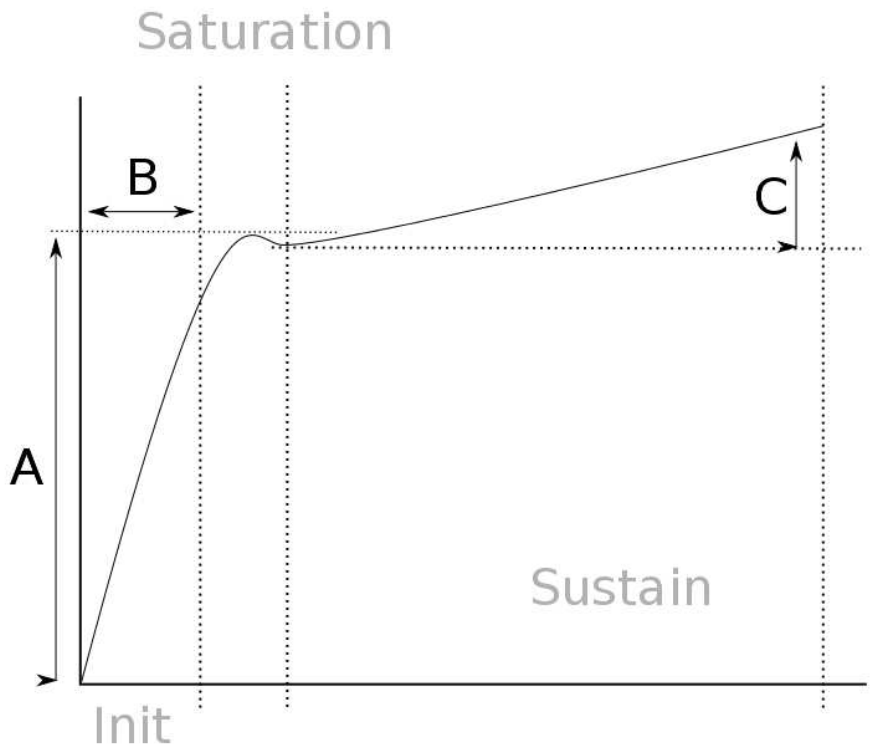


Figure 6.9:

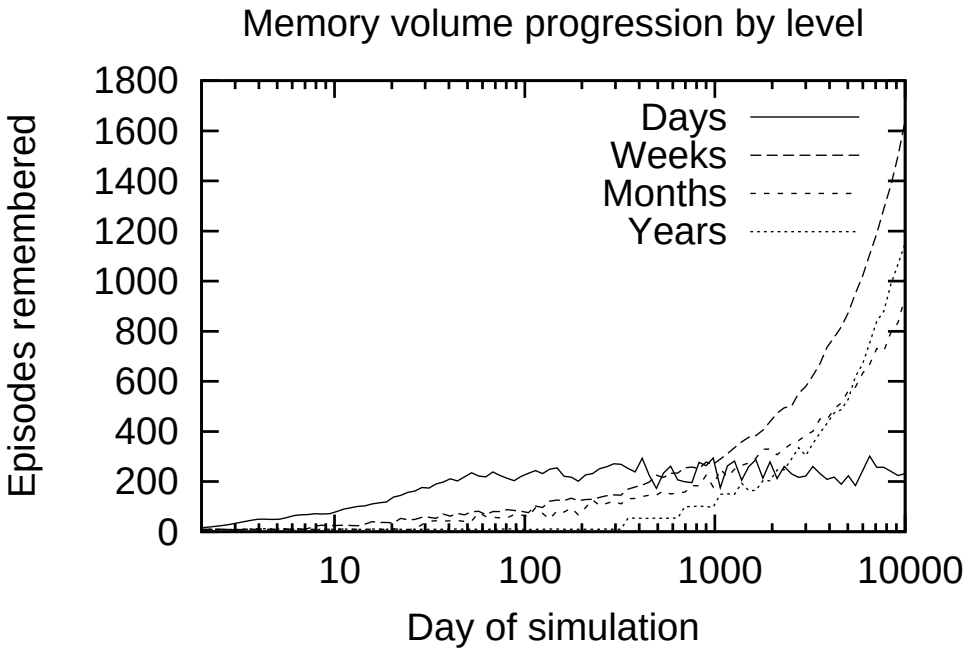


Figure 6.10:

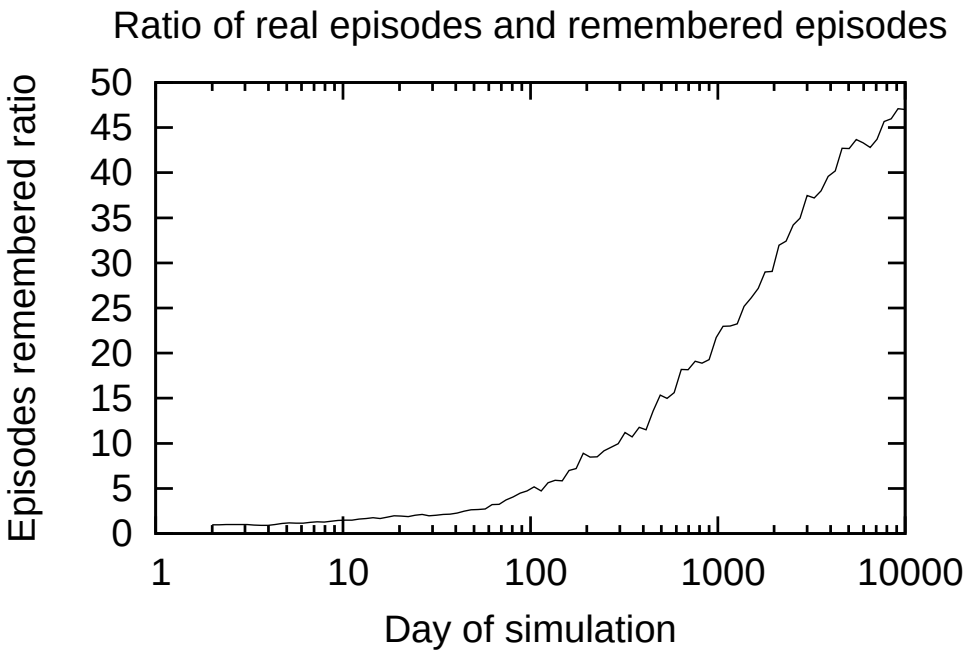


Figure 6.11:

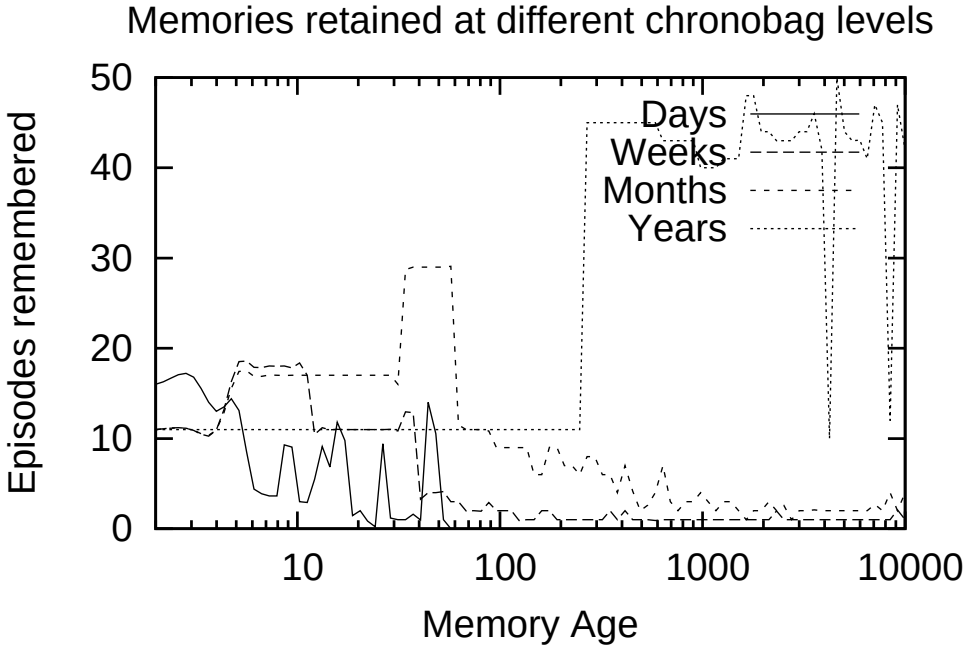


Figure 6.12:

6.2 Episode generator

6.2.1 Domain Analysis

The imporant question raised when looking at generated corpora data is: "How complex are they really?" It is plain to see that the generator will create a fairly wide variety of episodes, but isn't that going to boil down to just a couple of schemas? How does it compare to actual human interaction log?

This question was asked and partially answered by Rudolf Kadlec in a paper[11]. For the purpose of that article, a domain analyzer was created as part of this work that provides some insight into the inner domain combinatorics.

The analyzer calculates a value C for each node N. For each alternative definition of a method, its C is a sum of C of all subgoals. For each goal, C is a product of C of all alternative definitions.

For each goal, the value C describes an uppper bound of the number of possible decompositions of the goal. In the topmost goal, this leads to a rough estimate of in how many different types of the associated episode are there. See 6.13 for a sample product of the domain analyzer and the associated C values. Please note that domain graphs tend to be fairly large and difficult for imaging so readability may be impacted. Refer to the code for the domain analyzer to generate your own, full size graph, described in Appendix B.

Due to combinatorial explosion, it is apparent the top-level method will have a very high C. However, it should be noted that as far as actual planning episodes are concerned, the memory-visible nodes of the episodes start with the Methods that are not prefixed with an underscore, which have much more sane values between 10 and 100. All other nodes are internal to the planning algorithm, but not exposed to the memory model.

This approach to measuring combinations is not precise. It is itself not a planner and performs no constraint checking, so it can only give us a theoretical upper-bound. That said, the domain is designed to put as few obstacles in the way as possible, and make the plans generally viable to be reached easilly.

6.2.2 Duration testing

An important point of both the generator and the memory model is the time longevity of the simulation. The baseline memory model on which this model is based works with much shorter durations, while we expect to generate and process memory data of very large volumes, consisting of an equivalent of up to a lifetime of an agent's life.

In a sample run determining just that, a large planning domain with time-schema specific events has been used to generate a corpora of *30000* days,

equalling to approximately 90 years of Agent's life. The average speed of planning is not time dependent and has not gone over *0.5 seconds* per simulated day totalling about 4 hours of corpora generation, resulting in a file of size *213998285 bytes*.

As for processing the input and feeding into memory, several memory-level optimisations have been made to reduce the complexity of forgetting. In particular, the baseline model has an $O(t^2)$ time complexity for memory reorganization, making it very difficult to simulate longer time frames.

In a sample run of this model, the aforementioned file has been processed into the memory, averaging roughly *1 second* per simulated week, making this even faster than the corpora planning.

These measurements naturally depend on the computer used, but should be fairly similar on most modern machines. Further optimisations may be possible, but seemed unnecessary at this moment. Specifically, the algorithms are entirely independent and may run in parallel or independently to save time of experimentation. See section Future work for discussion of this.

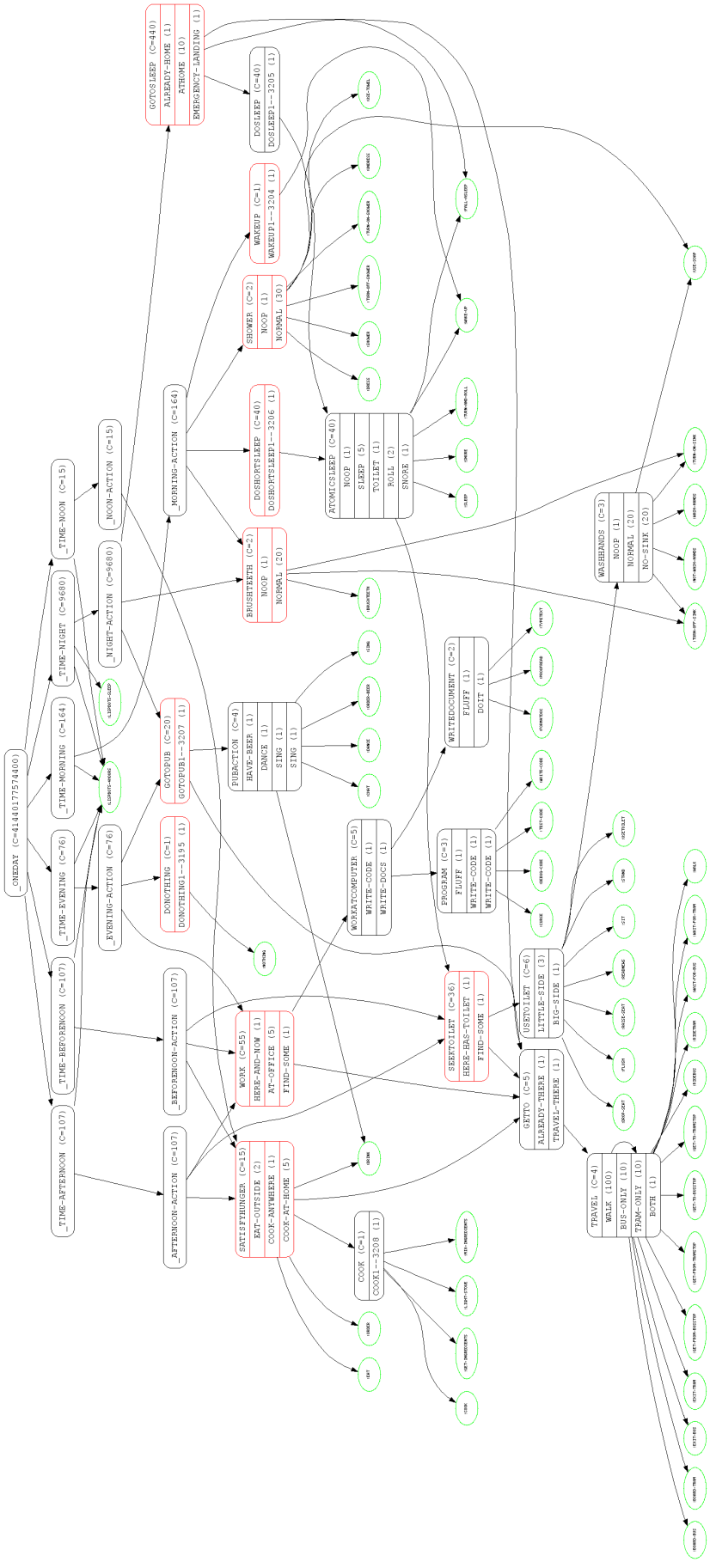


Figure 6.13: A dependency graph of a select domain.

7. Future Work

Due to running out of time for conducting experiments on this project, some topics remained unexplored. However, some were not planned to be a part of it, but may deserve some exploration anyway.

7.1 Memory model

The memory model has proven to be quite unstable with respect to parameters. Given the exponential nature of the retention function, a simple change of parameters has the potential to inflict a significant change into the model resulting in garbage data. Some experiments were made at determining sensible boundaries of the variables, but mostly rendered no publishable results.

While working on it, some topics came up that I believe to be an interesting future direction.

7.1.1 Temporal and chronobag hierarchies

The hierarchy for both chronobags and temporal schemas in this model is hard-coded as a parameter giving individual levels explicitly, at the beginning of the simulation. Obviously, it would be much better, if the model could determine these for itself, given the input corpora of episodes.

Related work by Ondrej Burkert[14] has been exploring automated creation of time concepts using neural networks on top of a Pogamut simulation. Taking a similar approach, a hierarchy could be determined and created live throughout the lifetime of a simulated Agent.

Certain psychological topics, for instance the psychosocial model of the retention curve[3] including the Reminiscence bump and childhood amnesia, are well explained through learning of schemas.

Given that the total volume of memories are very dependent on the size of the Chronobag hierarchy, exploring automatic schematisation seems very prudent and this work could serve as a basis for that.

7.1.2 Degressive schematisation

Both in this and the baseline model, the rise of schemas is generally linear. This means that the memory can tell exactly how many times did a certain schema happen more than another schema. This is contrary to the vaguely related memory learning research showing the usual reinforcment function following a logarithmic curve.

Several concepts that the memory model would strive to demonstrate make use of slower formation of schemas. Specifically, newly introduced schemas are believed to produce more episodic memories until the person is more familiar with them[3]. This would produce a "hump" in the memory retention in life phases which are rich in changes.

This model tends to recognise and remember schemas very fast, with very low uncertainty. Since most schemas are going to be fairly mundane, a new schema can be used as soon as in the next day to relieve Chronobags of some episodes or their details that can be reconstructed. Further reinforcements to these schemas only make them preferable over other schemas that might match in the future. This is the method how *False memories* are created by baseline memory model: A schema is devised in the present that will apply to past memories which have been reduced based on a different schema, making recall of them turn them into something else.

7.1.3 Internally threaded memory

Some of the algorithms used for the Memory model have a fairly terrible time complexity. However, most of them are also parallelizable, as much of it is processing a large chunks of individual structures, conveniently structured into trees.

The model would greatly benefit from a threaded approach, making more complex memory operations and experiments viable. The reason this project didn't do it was that it would generally imply a rewrite of most of the code structures, a fairly implementation-complex task.

7.2 Episode generator

7.2.1 Parallel processing of experiments

Experiments are executed on a case-by-case basis, with a corpus being created and then passed as an input to a java application. As both processes can take up to hours, it would be effective to streamline this process, and run them in parallel.

Similarily, an implementation of SHOP2 exists in java (JSHOP2), that could be used to directly produce corpora as Java structures. This, however, does not include Monroe modifications which would have to be ported, and that would be a fairly expensive task given the way the code bases have diverged by now.

This project got around this restriction by generating the corpus and passing it to the memory application using a commandline option handled by Makefile, which will automatically skip the first step if the input domain remains unchanged. However, if doing the processes in parallel would pose no performance impact as most current computers have multiple CPU cores, testing would benefit

from automating this process.

7.2.2 Massive domain simulation

While the domains provided by this project are large, they mostly demonstrate concepts that are possible to create with the episode generator. For more life-like testing, a much larger domain could be created to facilitate not just for repetitive actions but also personal development, life phases, etc.

Creating such domain would be best done based on data collected from real people, if they are willing to share them. Several related works already contain voluntary recordings of people's lives, which could be used to more carefully craft goals and actions to represent a more plausible corpora.

A lifetime scope would probably make this a project of its own, given the amount of data that has to be collected or data mined from other sources.

8. Conclusion

This work has presented an interconnected system of a memory episode generator and a memory model that allows pipelining of experiments related to episodic memory.

The episodic memory interface is a modified interface of a related memory model by Michal Cermak, and should be easy to adapt for various other purposes. Conversely, projects based on this baseline model should be adaptible to accept events from the episode generator.

The HTN planner based on Monroe emergency response domain generator has provided means to simulate large volumes of complex and structured data that meets constraints imposed on it. When employed to the extreme, it could provide a massive corpus used as a baseline for evaluating memory models of various kinds. Due to the nature of it, it should be mostly model-agnostic as it describes episodes that happen in their raw form, provided a parser is created to input the generated data into the memory model. One such parser is provided as an example for the episodic memory model.

A practical use of this project may be for Game designers, who would like to generate repetitive but more complex corpora of agent's life episodes, both for the purpose of performance testing their own memory model, and to create initialisation vectors for real in-game agents.

The presented memory model has been demonstrated to meet some novel non-trivial expectations placed on it that the original model did not meet. However, working with it was rather difficult and time consuming, both because of complexity of the tasks at hand, and because of some pre-existing bugs in the code, that were hopefully fixed as part of this work. Some of the experiments that were intended to be conducted did not end up with sensible results and remained unfinished at the time of publishing. In retrospect, It would probably have been easier and more useful for future to develop an entirely new model from scratch with some of the traits discussed in Future Work.

Bibliography

[1] Brainerd, C.J, Reyna, V.F.: *The Science of False memory*, 2nd Edition. Oxford University Press, 2005. ISBN 0-201-52983-1.

[2] Dudai, Y.: *(1997): How big is human memory, or on being just useful enough*, Journal of Learning and Memory, 3: 341-365

[3] Williams, H.L., Conway, M.A., Cohen, G. *(2008): Autobiographical memory*, In Memory in the real world 21-90. Psychology Press.

[4] Brom, C., Peskova, K., Lukavsky, J. *(2007): Modelling human-like RPG agents with a full episodic memory*, Technical Report No. 2007/4 of the Department of Software and Computer Science Education. Charles University in Prague.

[5] Wagenaar, W.A. *(1985): My memory: A Study of Autobiographical Memory over Six Years*. Journal of Cognitive Psychology, Volume 18, Issue 2, April 1986, pages 225-252.

[6] Rubin, D.C. *(1982): On the retention function for autobiographical memory.*, Journal of Verbal Learning and Verbal Behaviour, 21, pages 21-38.

[7] Gibson, J.J. *(1986): The ecological approach to visual perception.*, Lawrence Erlbaum

[8] Brown, R., Kulik, J. *Flashbulb memories*, Journal of Cognition, Volume 5, Issue 1, 1977, Pages 73-99.

[9] Larsen, S.F., Thompson, Ch.P., Hansen, T. *Time in autobiographical memory*, Remembering our Past: Studies in Autobiographical Memory, Cambridge University Press, 1995. ISBN 0-521-46145-6 hardback.

[10] Pillemer, D.B., Picariello, M.L., Law, A.B., Reichman, J.S. *Memories of College: The importance of specific educational episodes*, Remembering our Past: Studies in Autobiographical Memory, Cambridge University Press, 1995. ISBN 0-521-46145-6 hardback.

[11] Kadlec, R., Cermak, M., Behan, Z., Brom, C. *Generating Corpora of Activities of Daily Living and Towards Measuring the Corpora's Complexity*, Unpublished article.

[12] Brom, C. *Rizeni virtualnich lidi ve velkych virtualnich svetech*, Disertation thesis. Charles University in Prague.

[13] Blaylock, N., Allen, J. *(2005): Generating Artificial Corpora for Plan Recognition*, Lecture Notes in Computer Science 2005. Volume 3538/2005.

[14] Burkert, O. *(2009): Connectionist Model of Episodic Memory for Virtual Humans*, Master thesis. Charles University in Prague.

[15] Behan, Z. *A plan generator for episodic memory simulation and modelling*, Google Code project. *http://code.google.com/p/epis-planner/*

[16] *Pogamut*, Online webpage, *http://pogamut.cuni.cz/*

[17] *SHOP and SHOP2 planner*, Home page of the project. *http://www.cs.umd.edu/projects/shop/*

9. Appendix A - Domain syntax

9.1 Structure

The planner is a modified Monroe Corpus Generator planner, which is a modified SHOP2 planner. Most syntax traits are inherited from SHOP2, and changes are either semantic, or in the planning algorithm. That said, some small modifications to the syntax should be noted.
Each plan consists of **state**, located in state.lisp, and **domain**, located in plib.lisp, both located in the respective problem directory. (eg. longlife/)

9.1.1 State

State is most basically a listing of facts, in the form of lisp lists. Facts can mean anything, and describe the invariable parts of the world. Monroe planner introduces slight variability to the state, by allowing random generation of some parts of the state. In case of general planners, state can be anything, but for world simulations, state usually contains objects/locations, and facts about these (e.g. affordances, routes, ..).

9.1.2 Domain

Domain defines the world mechanics, or what are all the things that can happen. For planning in general this means rules that transition from one state to another state. For world planning specifically, this will usually mean goals and actions.

9.2 Syntax

Syntactically, both file specifications very closely follow the SHOP2 syntax with one notable exception of the :weight modifier. Both files are technically data files, but are written in LISP syntax. Therefore it is possible to comment (;), and also insert as much whitespace anywhere, as long as the parentheses are paired. They also allow function execution, which is why all lists have to be quoted (') at the top level. Also, all definitions, function names and atom names are case-insensitive.

9.2.1 State

Monroe planner recognises many parameters inside the state file, most notably ***fixed-state***, and ***fixed-state-need-loc***. These two generate a list of facts in the following format:

```
(defparameter *fixed-state*
'(
(fact1 param1 param2 param3 ... paramN)
(fact2 param1 param2 ... paramM)
...
(factK ...)
```

)
)

Facts are pieces of the world that are acted upon later. An example of a fact could be **(affordance home to-eat apple1)**. That describes an apple at home, that can be eaten (has the to-eat affordance). The ordering of fact arguments is very loose, but exactly the same syntax is then invoked in the domain rules. It is therefore important to fixate the same facts to always use the same argument meanings, to not accidentally switch f.e. an actor with an actee.
Another important use of a state file is describing fixed starting conditions. If the plan tracks the location of something by means of moving an *(atloc ?)* fact, it is important to initialize its position before the plannig starts.

9.2.2 Domain

Domain definition is a list of statements describing various actors on the world state. The syntax of the file is most basically the following:

(make-domain 'domainname '(
statement1
statement2
...
statementN
)
)

Statements can be either **methods**, **operators** or **axioms**.

9.2.3 Methods

Methods are high level goals which break down to smaller sub-goals or atomic actions (operators). They have the following syntax

(:method [:weight N] (Name arg1 arg2 ... argN)

variant

(preconditions)
(subgoals)
)

Each method has a name. It is going to be invoked by that exact name during planning. Methods are by definition overloaded. Each new definition of the same method is another variant how that goal can be reached. It is possible to have as many variants as the memory can take. Variant is a string that doesn't have to be present and is only useful for readability, to describe what does the variant do. Preconditions is a list of facts or axioms that have to hold true for the given variant to be planned, or (not (fact—axiom)) for negation. Subgoals is a list of either methods to be achieved or operators to be executed for this method to succeed. A variable modifier :weight states how likely is this given variant of

all methods of the given Name to be picked for planning. This can be used to describe more likely scenarios.
An example method would be:

(:method :weight 20 (DoSomething ?where ?when)
normal-variant
((time ?when) (i-am-at ?where) (mood ?mood))
((DoALittleBitOfIt ?where) (Complain ?mood) (!climb-down-stairs ?where))
)

9.2.4 Operators

Operators are actors in the world. They have three arguments: 1) a list of facts to hold true, 2) a list of facts pre-existing in the world, 3) a list of facts post-existing in the world. For an operator to work, both facts listed in 1 and 2 have to exist in the current state. After the operator has successfully acted, the list 2 is removed from the world, and the list 3 is added to the world.

(:operator (!wake-up) ((time morning)) ((position sleeping)) ((position sitting)))

9.2.5 Axioms

Axioms are most simply complex facts, consisting of a combination of multiple facts. They can be used in place of facts both in operators (list 1 only) and methods for checking pre-existing conditions. They are a list of sub-facts/sub-axioms to be checked, in order for the axiom to hold true.

(:- (CanIWalk ?here) ((whoami ?me) (DoIHaveLegs ?me) (IsThere-AFloor ?here) (not (Lazy ?me))))

10. Appendix B - Runtime Requirements and Instructions

10.1 Project requirements

The project consists of two parts, Memory model and associated Lisp tree parser written in Java, and episode corpus generator written in Common Lisp. In order to run experiments, all of these are needed.
The project has been tested on:

* Java runtime environment 1.6
Apache Maven 3.0.4
Netbeans 7.1 (not needed for running experiments)
GNU Clisp 2.48
GNU Make 3.82
Graphviz 2.28.0
POSIX-compliant unix system
Gnuplot 4.6 for plotting results

In addition to these, it is quite likely that the project will run on Win32-based *Cygwin* or *Mingw* systems, but this has not been practically verified.

10.2 Running experiments

For convenience, a toplevel Makefile is provided to automate the whole process. Its main goals are called *experiment1..exprimentN*. To launch the first experiment, simply run:

```
$ make experiment1
```

10.3 Running the Generator

The generator project is located in *episode-planner* subdirectory of the source repository root. In order to manually launch the corpus generator (*create-corpus.lisp*), you need to invoke Common Lisp explicitly, and give appropriate arguments to the generator. As most tools, the generator will complain if given the wrong syntax.

```
$ cd episode-planner
$ clisp create-corpus.lisp -h

usage: create-corpus.lisp corpus-name corpus-size
```

Note that running *clisp -C* will significantly speed up execution by precompiling the project into bytecode first. This is very useful for long-running experiments.

10.4 Running Memory model

Both the memory model and lisp parser are Maven projects. The easiest way to run them is to open them in Netbeans. There are more or less complicated ways to do the same from CLI. Please refer to the Makefile for examples on how to do that.

10.5 Helper tools

There are two additional tools provided for the corpus generator. *domainsummary.sh* is a tool used to generate domain descriptions and is used during all experiments. *depgraph.sh* is a script to generate a dependency graph for the given domain using graphviz. This is the tool used during experiment 5.

11. Appendix C - Source credits disclaimer and online location

The project is based on various other projects cited throughout the text. This attempts to outline the amount of work credited to other people contributing to this thesis.

11.1 Complete source code location

The sources to this thesis are available online at http://code.google.com/p/episplanner/, as well as on the supplied CD. The CD has the source code in a git repository for easier browsing of the commit history, along with this text and a simple Readme.txt.

11.2 Memory model

The memory model was originally written by Michal Cermak and heavilly refactored and bent for this purpose by me. Following is a diffstat of the code changes done solely by me.

```
EpisBotMemory/pom.xml                                |    8 +-
 .../pogamut/episodic/decisions/AffordanceSlot.java |    4 +-
 .../pogamut/episodic/decisions/DecisionTree.java   |  108 ++--
 .../amis/pogamut/episodic/decisions/Intention.java |    2 +-
 .../cuni/amis/pogamut/episodic/decisions/Node.java |   10 +-
 .../pogamut/episodic/episodes/AgeInterval.java     |  105 ----
 .../amis/pogamut/episodic/episodes/ChronoLogy.java |  324 ++++++++++
 .../amis/pogamut/episodic/episodes/Chronobag.java  |  505 ++++++----------
 .../amis/pogamut/episodic/episodes/Episode.java    |  176 ++++--
 .../pogamut/episodic/episodes/EpisodeNode.java     |  536 ++++++++++++-----
 .../amis/pogamut/episodic/episodes/ObjectNode.java |   13 +-
 .../amis/pogamut/episodic/episodes/ObjectSlot.java |   39 +-
 .../amis/pogamut/episodic/memory/Affordance.java   |   23 +
 .../pogamut/episodic/memory/AffordanceUsed.java    |   51 --
 .../amis/pogamut/episodic/memory/AgentMemory.java  |  504 ++++++++++++++--
 .../amis/pogamut/episodic/memory/IAgentMemory.java |   76 +++-
 .../amis/pogamut/episodic/memory/Parameters.java   |   79 ++-
 .../amis/pogamut/episodic/query/ComboTexts.java    |   12 +-
 .../amis/pogamut/episodic/query/QueryExecutor.java |   64 +-
 .../amis/pogamut/episodic/query/QueryModule.java   |    4 +-
 .../amis/pogamut/episodic/schemas/SchemaBag.java   |  411 ++++++++++++++
 .../pogamut/episodic/schemas/SchemaCounter.java    |   55 +-
 .../episodic/schemas/SchemaEpisodeNode.java        |   35 +-
 .../amis/pogamut/episodic/schemas/SlotContent.java |   28 +-
```

```
.../pogamut/episodic/schemas/TemporalSchema.java   |   81 +++
.../amis/pogamut/episodic/visualizer/EdgeType.java |    2 +-
.../pogamut/episodic/visualizer/VertexType.java    |    2 +-
.../episodic/visualizer/VisualizationCreator.java  |  100 ++--
.../episodic/visualizer/VisualizationRenderer.java |   24 +-
.../pogamut/episodic/decisions/ActionTest.java     |    8 +-
.../episodic/decisions/AffordanceSlotTest.java     |    4 +-
.../episodic/decisions/AtomicActionTest.java       |    4 +-
.../episodic/decisions/DecisionTreeTest.java       |    4 +-
.../pogamut/episodic/decisions/IntentionTest.java  |    2 +-
.../pogamut/episodic/episodes/AgeIntervalTest.java |   62 --
.../pogamut/episodic/episodes/ChronobagTest.java   |  265 +++++----
.../pogamut/episodic/episodes/EpisodeNodeTest.java |  187 ++++---
.../pogamut/episodic/episodes/EpisodeTest.java     |   66 ++-
.../pogamut/episodic/episodes/ObjectNodeTest.java  |    2 +-
.../pogamut/episodic/episodes/ObjectSlotTest.java  |    6 +-
.../pogamut/episodic/memory/AgentMemoryTest.java   |  632 ++++++++++++++++---------
.../pogamut/episodic/schemas/SchemaBagTest.java    |  109 ++--
.../episodic/schemas/SchemaCounterTest.java        |   36 +-
.../episodic/schemas/SchemaEpisodeNodeTest.java    |   30 +-
.../episodic/schemas/SchemaObjectNodeTest.java     |    4 +-
.../pogamut/episodic/schemas/SchemaSlotTest.java   |    4 +-
.../pogamut/episodic/schemas/SlotContentTest.java  |   32 +-
47 files changed, 2688 insertions(+), 2150 deletions(-)
LispBots/.gitignore                                |    1 +
LispBots/default.desc                              |   84 +
LispBots/default.lisp                              |  600 +
LispBots/nbactions.xml                             |   40 +
LispBots/pom.xml                                   |   34 +
.../amis/pogamut/episodic/lispbots/CliArgs.java    |   68 +
.../episodic/lispbots/ExperimentRunner.java        |  733 +
.../pogamut/episodic/lispbots/LispBotMain.java     |   25 +
.../amis/pogamut/episodic/lispbots/LispTree.java   |  463 +
.../pogamut/episodic/lispbots/MemoryFeeder.java    |  197 +
.../pogamut/episodic/lispbots/VisitableTree.java   |   73 +
.../pogamut/episodic/lispbots/LispBotTest.java     |  116 +
.../amis/pogamut/episodic/lispbots/MichalTest.java |  413 +
LispBots/testdata.desc                             |   84 +
LispBots/testdata1.lisp                            |  220 +
LispBots/testdata30.lisp                           | 5586 ++
LispBots/testdata7.lisp                            | 1448 +
17 files changed, 10185 insertions(+), 0 deletions(-)
```

11.3 Episode planner

The episode planner is based on an original Monroe[13] emergency response domain planner written by Nate Blaylock. That is based on SHOP2 project licenced under a GPL/LGPL/MPL opensource licence. In order to properly distinguish

local changes and original code, the first version is commited into the repository as-is.

```
 episode-planner/.gitignore                          |    7 +
 episode-planner/README.txt                          |    6 -
 episode-planner/analyze.lisp                        |  479 ++++
 episode-planner/create-corpus-code.lisp             |  299 --
 .../create-corpus-code.sliceoutput.lisp             |  272 --
 episode-planner/create-corpus.lisp                  |   19 +-
 episode-planner/depgraph.sh                         |   17 +
 episode-planner/domainsummary.sh                    |    8 +
 episode-planner/examples/phases/README              |    1 +
 episode-planner/examples/phases/plib.lisp           |  100 +
 episode-planner/examples/phases/state.lisp          |   27 +
 episode-planner/examples/prob-test/README           |    1 +
 episode-planner/examples/prob-test/plib.lisp        |   80 +
 episode-planner/examples/prob-test/state.lisp       |   26 +
 episode-planner/examples/simple-counter/README      |    1 +
 episode-planner/examples/simple-counter/plib.lisp   |   82 +
 episode-planner/examples/simple-counter/state.lisp  |   28 +
 episode-planner/longlife/plib.lisp                  |  932 +++++++
 episode-planner/longlife/retention/plib.lisp        |  932 +++++++
 episode-planner/longlife/retention/state.lisp       |  153 ++
 episode-planner/longlife/state.lisp                 |  153 ++
 episode-planner/longlife/temporal/plib.lisp         |  932 +++++++
 episode-planner/longlife/temporal/state.lisp        |  153 ++
 episode-planner/monroe/plib.lisp                    |  684 +++++
 episode-planner/monroe/state.lisp                   |  364 +++
 episode-planner/monroe_plib.lisp                    |  684 -----
 episode-planner/monroe_state.lisp                   |  364 ---
 episode-planner/nlib.lisp                           |  131 -
 episode-planner/planlib.lisp                        |  103 -
 episode-planner/shop2random.lisp                    | 2833 ----------
 episode-planner/src/README.txt                      |    6 +
 episode-planner/src/create-corpus-code.lisp         |  302 ++
 .../src/create-corpus-code.sliceoutput.lisp         |  273 ++
 episode-planner/src/nlib.lisp                       |  167 ++
 episode-planner/src/planlib.lisp                    |  105 +
 episode-planner/src/shop2random.lisp                | 2872 ++++++++++++++++++++
 36 files changed, 8896 insertions(+), 4700 deletions(-)
```