Case-Based Planning for Large Virtual Agent Societies

Tomas Trescak and Anton Bogdanovych
MARCS Institute for Brain Behaviour and Development
School of Computing, Engineering and Mathematics
Western Sydney University, NSW, Australia
t.trescak@westernsydney.edu.au
a.bogdanovych@westernsydney.edu.au

Figure 1: Using Case-Based Planning for Simulating Everyday Life in Ancient Mesopotamia, 5000 B.C.

ABSTRACT

In this paper we discuss building large scale virtual reality reconstructions of historical and cultural heritage sites. One important aspect of developing such simulations is how to populate virtual reality environments with crowds of virtual agents that are capable of simulating everyday life of the reconstructed society in a complex way, while respecting the cultural and historical accuracy of agent behaviour. In many commercial video games such agents either have very limited range of actions (resulting primitive behaviour) or are manually designed (resulting high development costs). In contrast, we propose to build large virtual agent societies following the principles of automatic goal generation and automatic planning. Automatic goal generation in our approach is achieved through simulating agent needs and then producing a goal in response to those needs that require satisfaction. Automatic planning refers to techniques that are concerned with producing sequences of actions that can successfully change the state of an agent to the state where its goals are satisfied. Classical planning algorithms are computationally costly and it is difficult to achieve real-time performance for our problem domain with those. We explain how real-time performance can be achieved with Case-Based Planning, where agents build plan libraries and learn how to reuse and combine existing plans to archive their dynamically changing goals. We illustrate the novelty of our approach, its complexity and associated performance gains through a case-study focused on developing a virtual reality reconstruction of an ancient Mesopotamian settlement in 5000 B.C.

VRST'17, November 2017, Gothenburg, Sweden

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

CCS CONCEPTS

• **Software and its engineering** → **Virtual worlds software**; • **Computing methodologies** → *Artificial intelligence*; *Planning and scheduling*;

KEYWORDS

Case-Based Planning, Social Simulations, Virtual Agents

ACM Reference format:
Tomas Trescak and Anton Bogdanovych. 2017. Case-Based Planning for Large Virtual Agent Societies. In *Proceedings of 23rd ACM Symposium on*

Virtual Reality Software and Technology, Gothenburg, Sweden, November 2017 (VRST'17), 11 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

3D Visualisation and Virtual Reality simulation are popular techniques for reconstructing sites of high historical or cultural significance. They are useful for showing how these sites were enacted in the past and are particularly helpful when the reconstructed physical environment is not entirely preserved or completely destroyed at present. Populating the reconstructed 3D environments with virtual agents helps to illustrate how these buildings and objects were used by people and to highlight the key technological or cultural aspects of the simulated society. Supplying such Virtual Reality simulations with virtual agents that are capable of convincing and historically or culturally authentic behaviour beyond simple crowd simulation algorithms is difficult and costly. Virtual agents must be able to play different social roles, adhere to social norms, actively use surrounding objects, interact with other agents and even engage into interactions with humans. Modern video games are a good illustration in regards to the potential of having such simulations, but the cost of developing video games is very high. For example, the estimated cost of developing Crysis 3, one of the popular modern video games, is $66 Million [13]. Such a level of spending is not feasible when it comes to research simulations.

In the vast majority of commercial video games, similar to Crysis 3, virtual agents[1] are often manually scripted. Manual scripting, which may involve designing finite state machines, decision trees, etc., is costly due to the high time investment required from game developers. Very few commercial games, among which one of the best examples is "The Sims" are focusing on automating agent behaviour beyond simple obstacle avoidance. One of the key automation principles in The Sims [2] is to simulate agent needs (in a similar way to the Maslow's Pyramid of Needs [20]) and then make agents satisfy their needs by performing actions with the corresponding objects in the environment. Satisfying virtual agent needs by interacting with objects, however, is essentially scripted in The Sims too, as it involves executing manually designed finite state machines associated with these objects. So, while simulating agent needs allows to increase automation, manual design of the finite state machines assigned to various objects means that the development still remains costly.

A standard technique that is often used in the Artificial Intelligence (AI) community as a substitute for a set of scripted agent actions is planning [17]. AI planing involves automatically producing sequences of actions that lead to achieving a particular agent goal, which in The Sims context means making the agent state evolve to the state where the given need is satisfied. Technically speaking, planning involves conducting a search through the action space in order to find the desired sequence of actions. When the action space is large, such a search is associated with significant performance problems, which is the key reason why AI planning hasn't been widely adapted by game developers.

[1]We use this term when we refer to human-like avatars that are being controlled by the computer, not a human user/player.

In this paper we show how performance of AI planning can be significantly improved and become real-time for large scale simulations. This performance improvement can be achieved by first introducing ordering of the action space through the application of the OCMAS paradigm. Once the action space is ordered, we can further improve planning performance by utilising the so-called case-based planning, where agents memorise their past plans and try to adapt existing plans to new situations instead of generating new plans from scratch.

The remainder of the paper is structured as follows. In order to facilitate a reader's understanding of the discussed techniques we first present an example scenario related to building a large scale historical simulation in Section 2. Section 3 introduces the necessary background in relation to AI planning. Section 4 explains our approach to extending case-based planning so that it can be applied to large scale virtual agent simulations. Section 5 presents a case study that investigates the performance issues of using our approach for the simulation of Ancient Mesopotamia (Uruk) 5000 B.C. Finally, Section 6 provides some concluding remarks.

2 EXAMPLE SCENARIO: ANCIENT MESOPOTAMIA 5000 B.C.

To illustrate the issues involved in building large scale historical or cultural simulations and discuss how it is possible to automate agent behaviour in those, we suggest to consider the simulation of everyday life in ancient Mesopotamia around 5000 B.C. shown in Figure 1. For this simulation we have reconstructed 3 small ancient settlements using the settlement maps produced by the archeologists from [8].

Based on the knowledge obtained from history experts we could design the following simplified scenario portraying a day of an average citizen: an average agent in our simulation should start its day at around 6AM in the morning. Soon after waking up the agent would eat breakfast by preparing the food from storage or obtaining food through work. The agent would eat 4 times a day with intervals between meals being close to 6 hours. In-between meals the agent would work to satisfy immediate hunger or comfort. There is hardly any recreation time. If there is, the agents should explore the city or chat with each other.

The type of work an agent must perform depends on the social role this agent plays in the simulation. For simplicity, consider a scenario where each of the agents plays one of the following 4 social roles: Fisher, Baker, Potter and Shepherd. Figure 2 outlines the key tasks that these agents must perform at work: milking sheep, making pots, baking bread and fishing.

The first step of building the simulation involved modelling the buildings and the settlement layout based on the results of archaeological excavations and information provided by subject matter experts. The next step was to manually design the appearance of the base population of 2 agents. Treating the appearance of the agents from the base population as genetic code allows to automatically generate a desired number n of the simulation inhabitants following the approach in [25]. Given that each of these agents must play one of the aforementioned 4 roles we can either equally allocate the generated agents into the given 4 roles ($n/4$ - shepherds, $n/4$ - potters, $n/4$ - bakers and $n/4$ - fishers) or come up with a way of

Figure 2: The Four Agent Roles: Shepherd, Potter, Baker, Fisher

specifying the role distribution in the simulated society (e.g. 20% shepherds, 20% potters, 30% bakers and 30% fishers).

The key question this paper tackles is: once an agent has been generated and assigned with a particular role how can we make this agent automatically generate goals and plans, so that it simulates believable behaviour consistent with the aforementioned scenario?

Similar to The Sims game our agents obtain their goals through continuously tracking their simulated needs. A need can be seen as a reservoir. When the reservoir is full, the need is fully satisfied. As the agent progresses through the day, its needs are depleting, and the level of the corresponding reservoirs is changing. Each agent tracks the state of each such reservoir, selects the one that requires immediate attention and generates a goal to satisfy the given need. Once the agent has the goal, it performs the planning task, which is to find the sequence of actions that lead to achieving this goal. The process of tracking needs and generating goals is outside the scope of this paper. Interested readers will find more details in [4]. The key emphasis of this paper is on agent planning, so next, we discuss the planning aspect in more details.

3 PLANNING FOR VIRTUAL SOCIETIES

In traditional AI planning an agent normally has a textual representation of its current state (e.g. "hasFire, hasRawFood") and there is a sequence of actions, that modify its state. Each action is annotated with pre-conditions (the state that the agent must have to be able to perform this action) and post-conditions (that specify how the state of the agent will change after performing this action). For example, there could be an action called "cook", and its pre-conditions could be "hasFire, hasRawFood" and the post-condition could be "hasCookedFood". The planning process then involves performing an exhaustive search that finds the sequence of actions that can successfully evolve the state of the agent to the desired state. The desired state normally has to match the post-condition of one of the actions, so the agent can perform the backwards state space search algorithm [23] by first finding the action (e.g. "cook") that has its goal ("hasCookedFood) as the action's post-condition and then look for an action that has a post-condition matching the pre-condition of this ("cook") action. The agent continues searching and building the action sequence until the last action's pre-condition matches the agent's current state. This approach works fine when there is a small set of actions. But when both the set of actions and the number of agents is large then performing a continuous search through the action space by all agents results in significant performance problems. Another weakness of classical AI planning is its inability to properly handle tasks where a high degree of interaction between different actors is necessary. Being able to associate pre-conditions and post-conditions with multiple actors that can play different roles and have a variety of possible states is a difficult task and even if it is feasible, the associated search space becomes very large.

In the specific case of simulating virtual societies, it is often required to execute and coordinate (i) agent interactions with environment (ii) agent interactions with other agents, (iii) user-agent interactions, as well as (iv) to model relations and interactions between agent groups. A popular way of modelling societies that allow for such complex forms of interaction is using Organisation Centred Multi-Agent Systems (OCMAS). The purpose of using OCMAS to define virtual societies is that we can group agents into multiple hierarchical role structures, abstract agent interactions and impose norms on agent behaviour on a group level. Thus, a simulated society can be specified through its target groups (i.e. roles) and normative structures that shape and control the interactions of actors (humans and agents) playing these roles.

There are various popular OCMAS architectures, such as MOISE [16], MOISE+ [18] or Electronic Institutions (EI) [11]. Yet, in the core of each of these OCMAS systems is a multi-agent system that uses organisational structure to define, control and coordinate individual agent behaviour. In general, OCMAS specification can be decomposed into three parts. First, a *structural specification* of OCMAS defines agent language, roles, role relations and groups that agents can belong to. Second, a *functional specification* defines structures for agent interactions. In MOISE, a Social Scheme defines a tree structure with agent goal in its root and each other node defines a sub-goal. Interactions between sub-goals are modelled using three operators, *sequence, choice and parallelism*. Electronic Institutions define Performative Structures and Scene Protocols in the form of oriented graphs. Finally, a *deontic specification* establishes a relation between a functional and a structural specification, constraining execution of structural tasks to given roles and groups. Concerning deontic specification in Electronic Institutions, each arc in a Performative Structure and a Scene Protocol represents an agent action (illocution) limited to given groups and roles, or a transition, used for synchronisation purposes. Arcs also define pre-conditions and post-conditions in form of expressions with real variables, testing the existence and modifying *agent resources* (e.g. precondition specifying that agents can make fire only if they own at least three pieces of wood). Moreover, in EI, *norms* impose further limitations on agent performance and stimulate pro-active behaviour with agent requirement to comply with these norms.

Using OCMAS to simulate virtual societies helps to obtain a granular control over the individual behaviour of each agent, yet it allows for abstraction of control using agent groups. Controlling agents on individual level involves generating their goals based

on their current state and the state of the OCMAS system. Goals represent either actions (or illocutions), represented by arcs connecting two states, or a change in agent resources (e.g. wood > 5). To execute a goal, an agent needs to find a path from its current state, to the arc that represents the desired action or modifies agent state to desired values. Such path represents agent plan, consisting of series state transformations, executing one or more actions.

The OCMAS approach is popular when it comes to simulating virtual agents societies. Bogdanovych [3] has developed a formal framework, named Virtual Institutions, which defines processes needed to model and execute societies performing in (3D) virtual environments using OCMAS approach. Virtual Institutions provide abstractions and graphical tools that allow designing societies in all their complexity, but they don't innately support AI planning.

In smaller scale simulations with a limited number of actions and action combinations, agent plans can be manually defined (static) [5]. But with large dynamic systems, where agents execute complex plans in a changing environment, static plans become increasingly demanding to maintain. For example, it is difficult to pause and resume a static plan, as an agent could have lost necessary resources to continue the plan while executing a new plan and it is safest to restart the plan. Also, modification or introduction of a new action requires re-evaluation of all affected plans or creation of new ones.

Employing AI planning, also known as *dynamic planning*, is one way of dealing with these limitations. But classical AI planning approaches are not directly applicable to OCMAS. Further in the paper, we will use the term "dynamic planning" when we refer to AI style planning with OCMAS.

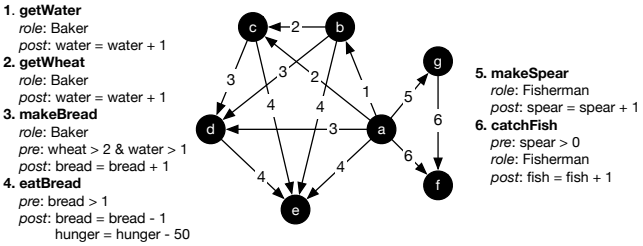

Figure 3: OCMAS specification

To illustrate how dynamic planning could be conducted consider Figure 3, which outlines a *simplified* specification of a scene protocol of a Virtual Institution with "Baker" and "Fisher" roles, where Bakers can bake and eat bread and Fishers can catch fish using required resources. For simplicity we do not depict all connections; you can assume that from any state you can go back to the initial state (a). Using this specification, the left part of the Figure 4 depicts a plan creation for the agent playing the role of Baker. The agent is currently in the state (a), it owns no resources, but it is hungry (i.e $bread = 0, wheat = 0, water = 0, hunger = 80$), and its goal is to drop hunger below threshold level, i.e. $hunger < 50$. The agent, using either forward or backward search, finds that hunger drops after performing *eatBread*. To eat bread it needs to obtain wheat and water to bake bread. This knowledge could be encoded in pre-conditions and post-conditions of an action. The right part of the Figure 4 shows a shorter plan for the same Baker agent and the same goal, yet it already owns some resources (i.e. $wheat = 2, water = 1, bread = 0, hunger = 80$).

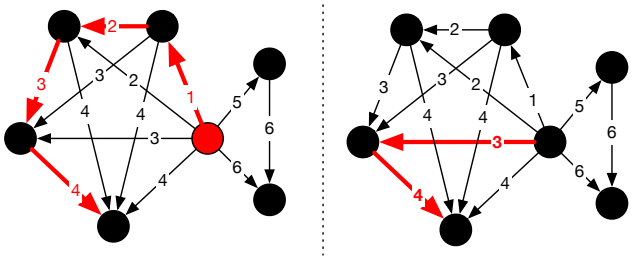

Figure 4: Dynamic specification

Such dynamic planning in OCMAS can be supported through mapping the OCMAS specifications to STRIPS [12] format. STRIPS (Stanford Research Institute Problem Solver) is an automated planner, developed in 1971, yet it also represents a formal language of inputs to dynamic planners. A STRIPS instance is composed of (i) an initial state, (ii) goal states, and a (iii) set of actions, where each action contains a list of preconditions and postconditions in form of boolean expressions. Pre-conditions contain *predicates*, while post-conditions use *operators* modifying the state of STRIPS instance. To solve a STRIPS instance means to find an ordered list of actions that leads from the start state into one of the goal states, satisfying preconditions of each action.

STRIPS is not directly suitable for dynamic planning with OCMAS and Virtual Institutions because Virtual Institutions employ *real expressions* in predicates, while classical STRIPS only works with boolean expressions. To perform a mapping from OCMAS to STRIPS, we map each arc from the structural specification of OCMAS to a new STRIPS action, where arcs in OCMAS contain pre-conditions and post-conditions. Using STRIPS definitions, we can use traditional planning methods. Below is a fragment of a STRIPS definition example from the scenario discussed in Section 2 (space limitations prevent us from publishing the complete example). The meaning of predicates, actions and operators is self-explanatory:

- **Predicates:** `Eq(var, x), Greater(var, x), Lower(var, x), HasRole(agent, y), InState(agent, z)`
- **Operators:** `ChangeValue(variableName, newValue)`
- **Actions:**
  - name: `getWater()`
    preconditions: `HasRole(agent, 'Baker')`
    postconditions: `ChangeValue(water, water + 1)`
  - name: `makeBread()`
    preconditions: `HasRole(agent, 'Baker') & Greater('wheat', 2) & Greater('water', 1)`
    postconditions: `ChangeValue(bread, bread + 1)`

At this point, we have enough mechanisms to dynamically plan single-agent goals. Yet, dynamic planning is NP complete problem and as a result, it is a very computationally expensive process.

The problem of STIRPS lies in its complexity, allowing to execute only small problems in real time. The planning complexity relates to the number of possible actions that can be considered to perform a next planning step. Thus, the complexity is exponential in relation to a number of plan items, i.e worst case is $O(n^n)$ complexity for

n plan items. A benefit of using OCMAS is that we can reduce the dynamic planning complexity, by structuring OCMAS as a Hierarchical Task Network (HTN) [9] [4]. The idea of HTN is that similar actions can be composed into compound actions (i.e. activities in Electronic Institutions, sub-goals in MOISE). Similarly, complex actions can be decomposed into several sub-actions.

Therefore, using composition to group actions into compound actions, we reduce the planning search space. Figure 5 describes the process of composition, where we have composed four new actions (A, B, C and D), which grouped actions into common functionalities. The complexity of the search space has dropped from 9^9 to $5^5 * 3^3 * 2^2 * 2^2$, that is from 387,420,489 possible plan combinations to 450,000 combinations.

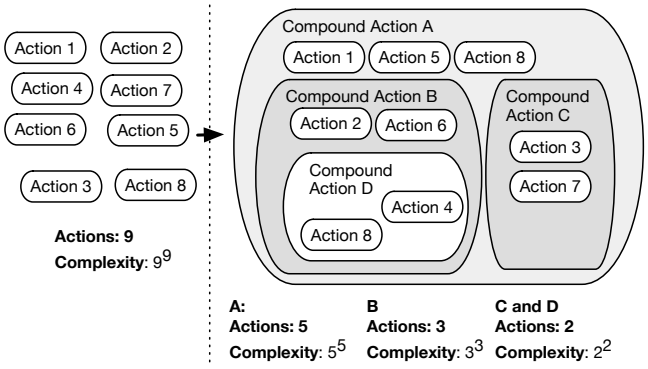

Figure 5: Composition using Hierarchical Task-Networks

While structuring OCMAS as HTN certainly allows the definition of more complex scenarios in need of dynamic planning, it does not eliminate the execution of a computationally expensive planning task whenever agent considers a new goal. This becomes increasingly difficult when simulating several individual agents, for example in VR, where smooth frame-rate is very important for the user experience. Even after ordering the action space with the help of OCMAS and HTN it is difficult to achieve real-time dynamic planning for a large number of agents. Figure 7 shows the results of our experiments that aimed to analyse the effect of dynamic planning on system performance[2]. The left axis represents *Frames-Per-Second* (FPS); the bottom axis represents the number of agents. It is evident from the figure that with the increasing number of agents, the FPS becomes unstable (spikes) and comes to halt at around 40 agents. This experiment was performed in our simulator Vittoria (see Figure 6), on Alienware 17 (2014) computer[3], running a Virtual Institution with eight actions and four different goals. While initial experiments were performed in the game development environment Unity 3D[4], soon we discovered that internal complexity of the Unity 3D engine and preference on the handling of game related elements, does not allow us to have a clear view of the algorithm performance. Therefore, we have developed Vittoria[5] simulator to evaluate OCMAS models and agent behaviour using a visual approach with high-performance 2D graphics, to collect

[2]The maximum frame-rate in the simulation environment is 60 FPS
[3]https://www.laptopmag.com/reviews/laptops/alienware-17-2014 (visited 07/2015)
[4]https://unity3d.com, visited 07/2017
[5]https://github.com/tomitrescak/EI2

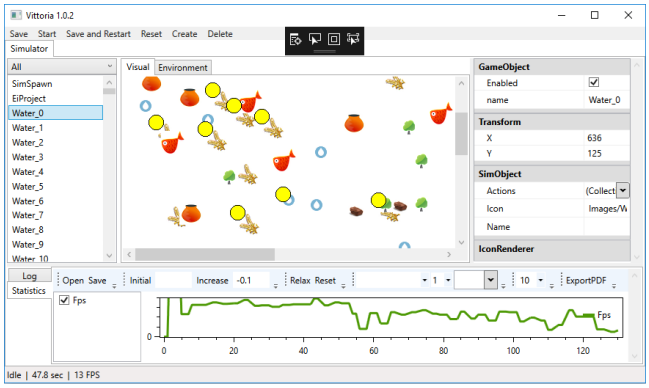

Figure 6: Vittoria simulator

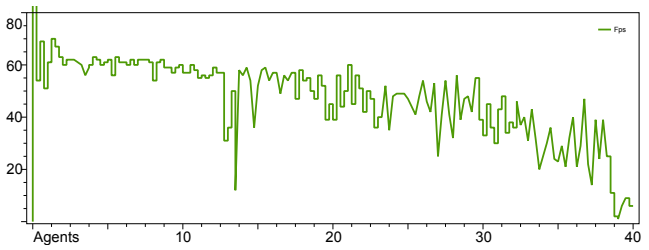

Figure 7: Unity3D performance with dynamic planning

various statistics and measure the performance of our algorithms. The positive aspect of this simulator is that agent behaviour developed for Vittoria can be used directly in a game development environment Unity 3D. This allows us to simulate a period of time in Vittoria and then visualise it in Unity 3D.

While we were able to execute 40 agents in the Vittoria simulator, the unstable FPS performance and sudden FPS spikes make it impossible to deploy this solution in VR or gaming. Therefore, we have analysed several institutions of various complexity, the content and variety of generated plans and realised that generated plans are largely similar and differ only in details, such as agent resource quantities, state of the environment and agent state.

As a result, we have stored the generated plans and if (i) agents had matching goals, (ii) they were in the same state and (iii) their resources and environment resources were equal, we have reused the plan. While this solution seemed feasible, we have quickly discovered that these conditions were too restrictive and agents were still generating large amounts of new plans. For example, when a stored plan would satisfy a goal of $wheat = 1$ for an agent having no resources (i.e. $wheat : 0$), we still had to generate a new plan for $wheat = 2$ for agent owning $wheat : 1$, although the generated plan can be the same. This approach was significantly inflating a case database with identical plans.

While we could tune the restrictions of storing plans, we have instead decided to perform a study to find a possible technology that can help us solve this problem systematically. Looking at the planning domain, we have found a Case-Based Planning methodology, which retrieves previously generated plans and adapts them to a current scenario. This seemed like a good fit for our purposes.

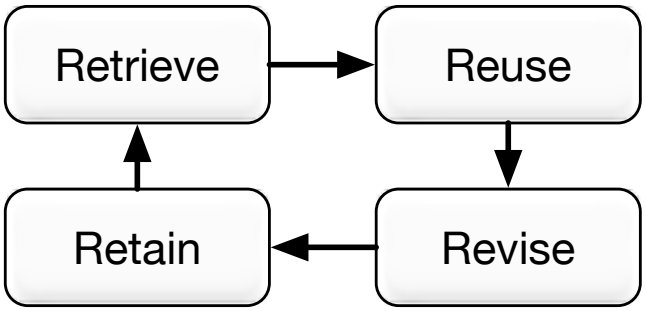

Figure 8: Case-Based Reasoning cycle

4 CASE-BASED PLANNING

Case-Based Planning [14] involves reuse of previous plans and adapts them to new situations. In the artificial intelligence domain, Aamodt et al. [1] developed Case-based reasoning (CBR) a problem-solving methodology for reusing, combining and adapting previous experiences to new problem situations (cases). CBR executes this process in four steps (see Figure 8):

(1) **Retrieve** step selects similar cases from the case database relevant to the current problem
(2) **Reuse** step adapts selected case to the new situation and create a new case
(3) **Revise** step verifies the obtained solution, usually with the help of an expert
(4) **Retain** step decides whether a case will we stored in the database or not

Authors have used and adapted Case-Based Planning in a multitude of different scenarios (see [24] for a comprehensive list of applications). One of the first case-based planning systems was CHEF, creating recipes in Szechuan kitchen. The main difference to classic planning systems was, that it was capable of learning from previous failures, adjusting generated recipes. While CHEF [14] represents a strict domain specific planner, there were attempts to create domain independent solutions [15]. For example, PRIAR [19] system focuses purely on reuse of existing plans (no retrieve phase). The problem of domain-independent planning is that was proven to intractable [10], thus approaches for improving planning cost are currently explored.

Similar to our approach, Prodigy/Analogy [7] system does not reuse previous plans but it reuses planning decisions stored in planning traces (in our case we reuse both plans and planning decisions). Such traces include information on which decisions were taken while planning, why other choices for the decision were not considered, etc. When a new plan is constructed Prodigy/Analogy replays stored traces, learning from previous experiences to make valid choices.

While most authors deal with systems where response time is not critical, some focused on real-time response capabilities of CBP, similar to our case. Otanon et al. [22] defined Online Case-Based Planning named "Darmok", able to play real-time strategy game WARGUS, competing with human players. Their system can retrieve and adjust cases in real time, yet requiring initial database of cases, created from annotated traces of human expert players.

In most of these systems (apart from PRIAR), with a robust database of cases, CBP using the CBR approach can explore similarities in existing cases, devise new solutions and eliminate the necessity of performing dynamic planning. This can greatly increase the efficiency of plan creation. Yet, under specific conditions CBP can become even more complex then dynamic planning itself and the plan quality can deteriorate [21]. As a result, we add specific constraints to the Reuse step. Next, we introduce a structure of stored cases and processes involving all four stages of a CBR cycle.

4.1 CBP Case

Before we proceed with the description of a CBR cycle, we need to define the structure of a stored (CBP) case. The strength of a CBR and CBP approach (further we only assume CBP) is its efficiency in a specific domain, where CBP cycle is tightly bound to the domain. Yet, it is also its weakness as CBP systems cannot be reused or shared between domains. In scenarios of our interest, we partially overcome this deficit by defining a CBP system that works with OCMAS, allowing us to define various agent and societal models.

The structure of a CBP case defines the situation at which agent encounters when the plan is executed, and the effect of plan steps (i.e. actions stored in a plan step) on its resources. We assume an *agent resource* to be a property of agent with a real value. In Electronic Institutions agent properties are defined for each agent role (e.g. Fisher role defines a property *fishCount*, *spearCount*, Baker role defines a property *breadCount*). Before the main definition, let us define the individual *plan item* of a plan stored in a *plan case*.

Definition 4.1. We define n-th *plan item* p_n as tuple $\langle a, c, \Delta^n(r)\rangle$, where:

- a is a name of an action performed,
- c is an arc that executes action a in OCMAS system,
- r is agent resource,
- $c_{post}(r)$ is a value of a resource r after performing action in arc c, and
- $\Delta^n(r)$ is a function, where $\Delta^0(r) = 0$ and $\Delta^n(r) = \Delta^{n-1}(r) + c_{post}(r)$, describing the change in value of agent resource r_i when performing the plan item p_n

Definition 4.2. We define a *plan case* C as tuple $\langle s, P, \Delta^{max}, \Delta^{min}\rangle$, where:

- s is the agent's current position in OCMAS system (i.e. state)
- P is an ordered list of *plan items*, $p \in P$
- $\Delta^{max}(r)$ is a function, that for resource r returns its maximum accumulated value *increase* during the execution of a plan of length n.
- $\Delta^{min}(r)$ is a function, that for resource r returns its maximum accumulated value *decrease* during the execution of a plan of length n.

For example, Table 1 depicts a plan case from the Uruk 5000 B.C. Please note, that each arc is specified as $s_x \rightarrow c_y \rightarrow s_z$, where s_x represents a "from" state of the connection c_y and s_z represents a "to" state of c_y:

Note that Δ^{min} contains only one item, as during plan execution only *hunger* resource's value has decreased, resulting in the agent having less hunger than the initial value. While other resource values dropped as well (e.g. *water, wheat*), overall their value did

not drop below zero in relation to the start of the plan. In other words, if an agent would stop the plan at any given step, it would not end up with less *water* and *wheat* than when it started. On contrary, Δ^{max} contains more items. For example, if the agent would stop the plan execution after step three, it would end up with two units of water and one unit of wheat (hence the increase).

State	Main/Idle (main activity, state idle)
Plan	
(1)	`getWater`, $s_0 \rightarrow c_1 \rightarrow s_0$, water: +1
(2)	`getWater`, $s_0 \rightarrow c_1 \rightarrow s_0$, water: +1
(3)	`getWheat`, $s_0 \rightarrow c_2 \rightarrow s_0$, wheat: +1
(4)	`makeBread`, $s_0 \rightarrow c_3 \rightarrow s_0$, bread: +1, water: -2, wheat: -1
(5)	`eatBread`, $s_0 \rightarrow c_4 \rightarrow s_0$, bread: -1, hunger - 50
Δ^{max}	$water \rightarrow +2$, $wheat \rightarrow +1$, $bread \rightarrow +1$
Δ^{min}	$hunger \rightarrow -50$

Table 1: Plan case from Mesopotamia 5000 B.C.

4.2 Retrieve

With the definition of the stored case, we are ready to specify the Case-Based Reasoning cycle. In this section, we define all processes for case retrieval. This process is depicted in Figure 9. First, let us describe the structure of agent goals.

4.2.1 Resource-Based Goals and Case Satisfaction. The Retrieve stage of the CBR cycle is responsible for selecting one or more cases that are similar to the current problem. In our case, the similarity of cases is given by their potential to *satisfy* agent goals. Goals in OCMAS systems are often represented as actions or illocutions (e.g. eatBread). For simulation of large societies of various cultures, we find this too restrictive and prefer goals modelled by the change in agent resources. Using such goals, we can create goals and generate plans more loosely. For example, goal `religiousSatisfaction > 0.5` requires to maintain religious satisfaction above threshold level for all agents belonging to any religious group, which can be performed by several actions, e.g `prayInChurch` (Catholics), `prayInMosque` (Muslim), or `goToBeach` (atheists). Moreover, with resource-based goals, we can either try to increase a resource value (e.g. wheat), or lower it (e.g. hunger).

If we specify agent goals as either positive or negative change in agent resources, we define:

Definition 4.3. Case $C = \langle s, P, \Delta^{max}, \Delta^{min} \rangle$ *weakly satisfies* goal g with n plan items $p_{i \leq n} \in P$ of agent with s resources $r_{i \leq s}$, if:

(1) Agent state is the same as start state of arc in p_1
(2) g is increasing value of resource r_j by k, and $k < \Delta^{max}(r_j)$, or
(3) g is decreasing value of resource r_j by k, and $k > \Delta^{min}(r_j)$

In other words, case C *weakly satisfies* goal g of agent a, if (i) agent is in the same position in OCMAS as agent from the stored case before the plan execution, (ii) agent is trying to increase a resource value by k and case has increased value of this resource by more than k, or (iii) agent is trying to decrease a resource value by k and case has decreased the value by more than k. Partial satisfiability does not ensure that found plan can be executed by an agent since the agent does not necessarily have all the required resources to perform the plan. Adding following restriction to case satisfiability, we define:

Definition 4.4. Case $C = \langle s, P, \Delta^{max}, \Delta^{min} \rangle$ *strictly satisfies* goal g with n plan items $p_{i \leq q}$ of agent a with t resources $r_{i \leq s}$, if

(1) it weakly satisfies goal g and
(2) every agent resource but the resource contained in goal never passes the minimum value r^{min}, $\forall m \leq t, \forall i \leq n, r_m + \sum_{j=0}^{i} \Delta^j(r_m) \geq r_m^{min}$.

This represents that a case C *strictly satisfies* goal g if (i) it weakly satisfies goal g and (ii) agents resource values never reaches less than allowed minimum value r^{min} when executing a plan from the stored case. Passing minimum resource values represents that agent has insufficient resources to perform this plan. Minimum resource values are often set to zero (e.g. $wheat^{min} = 0$ specifies that agent can never use more *wheat* than it currently owns). We name such case a *strict case*.

The retrieval phase checks whether a case strictly satisfies goal g by reconstructing the stored plan using agents current resources, described in the next section.

4.2.2 Plan Reconstruction and Plan Pruning. Assume that the retrieved case C contains a plan P with plan items $p_1, p_2 \ldots p_n$. Applying the $\Delta^{0<i<n}$ functions to agents current resources, our system can *reconstruct* how resources change during the plan execution. It also detects, if agent goal is fulfilled after $p_{i<n}$. If such p_i is found, the system creates a new case C^* and removes all items $p_{i+k}, n - i < k < n$ from the original plan. That is, from the definition of a stored case it is trivial to detect p_i that satisfies agents goal by taking agents current resources and increasingly apply resource deltas $\Delta^{0<i<n}$ until agents resources reach a goal value.

Step	Delta	Water	Wheat	Goal?
Start	–	0	0	No
p_1	water + 1	1	0	No
p_2	water + 1	2	0	No
p_3	wheat + 1	2	1	Yes

Table 2: Plan Reconstruction and Pruning

For example, consider a case from Table 1 and a goal of *wheat >= 1* for agent owning no resources. The plan reconstruction process is depicted in Table 2, where the system detected that goal is fulfilled after p_3. Therefore, the system can prune p_4 and p_5.

While pruning of case plans is often performed in the *Reuse* phase, in specific cases agent needs to perform it also during the case retrieval. These cases deal with the situations when there is no strict case to satisfy agent goal g, but we can combine existing cases to create a plan to reach the desired goal.

4.2.3 Plan Extension. When agent a plans for its goal g it will first try to retrieve cases that *strictly* satisfy goal g. When no such cases exist, it will retrieve those that *weakly* satisfy goal g and order them by the amount of missing resources, detected by reconstructing the plan from the current state of the agent. For example, consider a following plan from a weak case for agent owning no resources, having *hunger = 80* and the goal *hunger < 50*:

(1) `makeBread`, $s_0 \rightarrow c_3 \rightarrow s_0$, bread: +1, water: -2, wheat: -1
(2) `eatBread`, $s_0 \rightarrow c_4 \rightarrow s_0$, bread: -1, hunger: -50

The agent cannot perform this plan as both water (-2) and wheat (-1) would reach negative values during the plan execution, totalling

to three missing resources. The system can extend this plan and search for other cases that would satisfy all missing resources (i.e. goals *water* >= 2 and *wheat* >= 1). This means that before the execution of a plan item (1), the system will try to retrieve a case with a plan that starts in s_0 (current agent state), satisfies *water: +2*, and ends in s_0 (initial state of step (1) in the original plan. Similarly, we try to find a case that satisfies *wheat: +1*.

Definition 4.5. In order to satisfy a missing resource (*partial goal* g^*) from an *original plan* P, we define an *extension* of plan $P \rightarrow P'$ as insertion of a *pruned* plan P_n^*, before the n-th plan item P. Plan P_n^* can extend plan P only if the output state of the arc c of the plan item $p_{n-1} \in P$ is the same as the input state of the arc in the first item of $p_1 \in P_n^*$ and that the input state of the arc c in the plan item $p_{n+1} \in P$ is the same as the output state of the arc in the last item of $p_{last} \in P_n^*$. We name plan P_n^* an *extended plan.*

In the definition above, we require that plan P_n^* is *pruned* to minimise the side-effects of execution of the inserted plan. By side-effects, we mean the modification and utilisation of resources that are needed to execute the original plan. Therefore, we assume the plan extension to be:

(1) *safe* if it does not modify any resources consumed in the original plan
(2) *unsafe* if it does modify resources consumed in the original plan, but still allows execution of the extended plan
(3) *invalid* if it consumes enough resources to prohibit execution of the extended plan

The possible drawbacks of this approach are that in specific cases, the complexity of trying to reconstruct the plan from partial plans is very high and the plan quality is low (agents perform many unnecessary steps, e.g. collect many unnecessary resources). Therefore, we propose several heuristics that limit these drawbacks:

- $Width(m) \rightarrow n$ specifies how many new items can system insert into original plan of length m to strictly satisfy a goal.
- $Depth(m) \rightarrow n$ specifies how many times the plan of length m can be extended to strictly satisfy a goal.
- $Value(r) \rightarrow n$ specifies how many plan items can be inserted in order to retrieve one unit of resource r.

Unfortunately, there are no constant values for these heuristics as they are domain dependent, needing to be adjusted per each system. In our case of Uruk 5000 B.C., we have set following values `Width(m) = 10/m + 1` since no plan should contain more than ten items. We have also specified several Value heuristics, e.g. `Value(wheat) = 2, Value(water) = 1`. This tells the CBP system that to recuperate one unit of wheat a maximum length of the extension is two plan items. In case, when no stored plan exists to satisfy the partial goal g^*, CBP system performs a dynamic search to find the plan from g^*. If the search is successful, (i) it is pruned and stored in the case base and (ii) inserted in the original plan.

Dynamic search is also performed when no similar case for a current goal exists (e.g. empty case base, the goal requires new resources, the goal starts in a new state or heuristic failure). In this case, we dynamically search for the current goal and set up a new case. Figure 9 shows the retrieval process as an activity diagram.

4.2.4 Plan ordering. In case, when more then one plan is available to fulfil an agent's goal, it comes naturally to order plans by

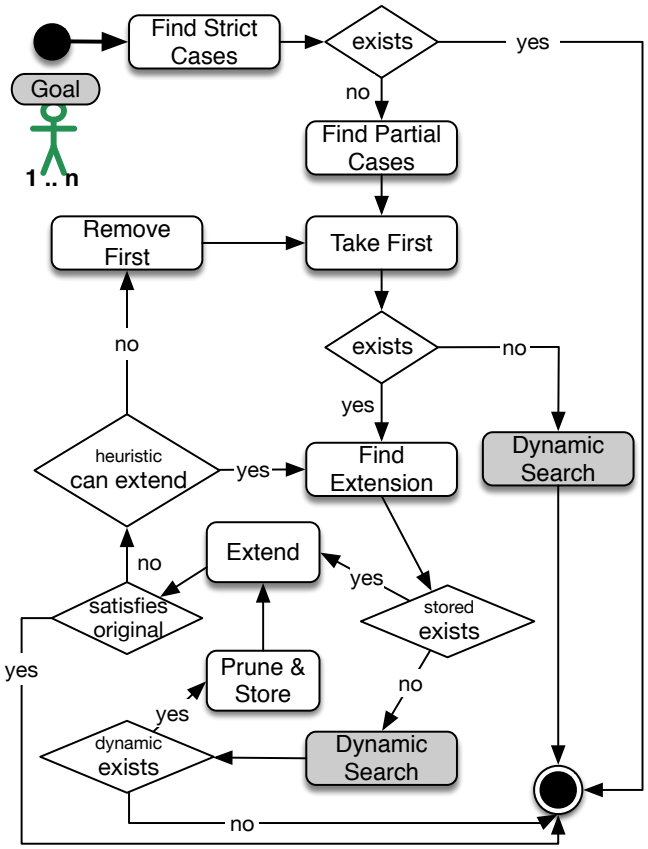

Figure 9: Retrieve cycle

a number of plan items. This would allow agents to finish their plans in the fastest possible way. Dealing with resources this is often not desirable. During plan execution for goal g, as a matter of serendipity, agents can obtain valuable resources with low effort, or satisfy the next planned goal. Therefore, we propose to order plans by the total *value* they provide for current needs depending on number of performed plan steps, $\frac{\sum_{i=0}^{k} value(r_i)}{steps}$. *Value* is often modelled as a linear or quadratic function, for example, if an agent owns no wheat, it may provide a higher value in contrast to owning a lot of wheat, where additional wheat provides less value.

4.3 Reuse, Revise and Retain

The Reuse stage is responsible for adapting the retrieved case to the current problem. In our case, we only prune the plan to avoid unnecessary actions when achieving a goal. These actions relate to the ones that happen after we have achieved our goal (see Section 4.2.2).

The Revise stage is responsible for verification of the adapted case, usually by an expert. In our case, the adapted cases can be verified automatically, by performing *plan reconstruction* described in Section 4.2.2. In Section 4.4 we will introduce more verifications that relate to use of CBP planning for virtual environments.

The Retain stage decides whether or not to store the new case into the case base. In our case, we store only new, unique solutions that strictly satisfy the current goal. The retain stage in our case

also has an *exploratory role*, which generates as many new cases as there exist *feasible paths* from any state of OCMAS system to the start state of plan P. By *feasible path* we mean a sequence of actions that can be performed from state s_n to state s_{n+m} considering the agent has no resources. Using feasible paths, we assure that the agent has enough resources to reach the start state of plan P and that it does not lose vital resources to execute plan P along the way. This further reduces the necessity for dynamic search in similar cases. With the current definition of CBP cycle we account for the creation of plans based on agents knowledge encoded in OCMAS system. In 3D virtual environments, we need to account for the possibilities given by the environment in which agent performs, often dynamically changing with limited resources.

4.4 CBP cycle for virtual environments

In 3D virtual environments, agents perform their actions by interacting with some objects or other agents. Therefore, each interactive object in the 3D environment needs to be annotated with names of actions it provides, along with other meta-data (e.g. animation that executes during interaction). When such annotation exists, we propose further restrictions and orderings on case retrieval:

(1) **Restriction:** Only retrieve and extend cases containing actions that can be executed in the current state of the environment. That is, do not allow the system to select cases that cannot be performed in the environment. For example, if all wheat has been harvested, bakers can no longer bake bread and have to find other means of obtaining food.
(2) **Ordering:** Order cases by the distance that needs to be travelled to complete the plan, assuming that agents prefer faster plan execution. This can be further optimised for the *delay discounting* (a behavioural measure of impulsivity, often used to quantify human tendency to choose a smaller, faster reward) according to the value that a given plan provides.

5 CASE STUDY AND EXPERIMENTS

To show the benefits of our approach, we apply it to the simulation of Ancient Mesopotamia (Uruk) that was introduced in Section 2.

For purposes of our evaluation, we assumed our existing simulation with roles: Baker, Fisher, Shepherd and Potter. Bakers bake bread made from wheat (processed to flour) and water in clay ovens. Fishers catches fish with a partner, where one is rowing a boat and another fishes using a spear made of reed. Fish is stored in pots. Shepherds walk sheep and milk them. Potters make pots from clay mixed with water, which they exchange for food. All Uruk citizens need to eat (bread, milk or fish), drink (water or milk from a pot), rest and sleep in their beds. They can socialise by talking to each other, they take baths in the river and clean their houses and work environments. The necessary resources to perform these actions are limited and scattered in the environment.

To generate agent goals, we have used a Sim-based approach [5], which allows automatic generation of goals according to various needs, i.e. physiological, social, fun, hygiene, tidiness. To satisfy these goals agents generate plans which are executed in the virtual environment by interaction with virtual objects and other agents. Such objects are annotated with actions they provide, the location at which the interaction starts and animations that are played by an agent during the interaction with the object.

Organisational structure of agent knowledge is encoded in a Virtual Institution [5], structured as hierarchical task network with compound activities. We have isolated 26 individual actions, organised into 5 compound actions on two levels. The top level contains four individual actions and five compound actions. The longest sequence of actions to satisfy a goal (hunger) is 7, involving the making of a spear from reed, fishing, cooking fish and eating it. The maximum number of considered actions after HTN optimisation on the top level is 9, giving worst case scenario of approx. 400.000.000 possibilities for plan construction.

In the second step of our evaluation, we ran the simulation using dynamic planning of actions, recorded the generated tests and measured performance. As expected, dynamic planning caused many FPS spikes and allowed us to execute only around 25 agents at a satisfying frame rate. Increased number of agents decreased system performance due to several concurrent planning processes. The generated plans formed our "ground truth" of believable plans for given contexts since they were evaluated in [4] and [6].

In the third step of our evaluation, we have employed CBP algorithm for plan generation, measured system performance and compared the generated plans to "ground truth", assuring the believability of agent performance.

Using Case-Based Planning, the frame rate has stabilised, allowing smoother experience in VR, using HTC Vive[6]. We were also able to increase the number of running agents, but due to graphics processing constraints of Alienware 17 PC, we were only able to execute around 50 agents at an acceptable frame-rate for VR. This limitation is due to the use of high-poly avatars and is currently in the process of optimisation.

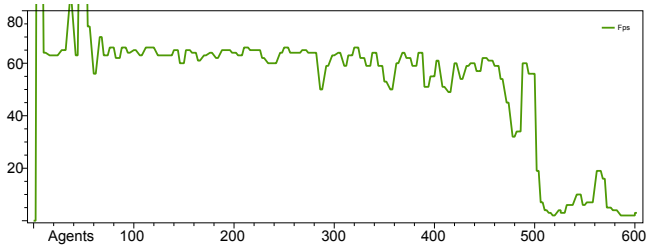

Figure 10: System frame-rate with CBP

Concerned by the graphical constraints of the VR system, we have decided to conduct further experiments in our high-performance simulation environment Vittoria. Figure 11 depicts the frame-rate of the simulation depending on the number of executed agents. Visibly, the frame-rate is more stable than in the dynamic-planning case in Figure 7 and starts to spike at around 300 agents, coming to a halt with 400 executed agents.

Upon investigation of this issue, we have discovered, that the performance drop was accounted to the current implementation of Virtual Institution execution environment, which is not able to execute a large number of agents. The reason was the garbage collection process performed by the .NET framework, removing resources used during agent communication and evaluation of

[6] https://www.vive.com/

agent actions. We are currently working on a high-performance implementation of Virtual Institutions execution environment, and it is part of our future works. Another factor that affected the "lower" number of executed agents is that we simulated a fast-forward mode, running one day in 60 seconds. As a result, agents were planning more frequently than in the "real" execution, not giving the .NET framework enough time to clean up resources.

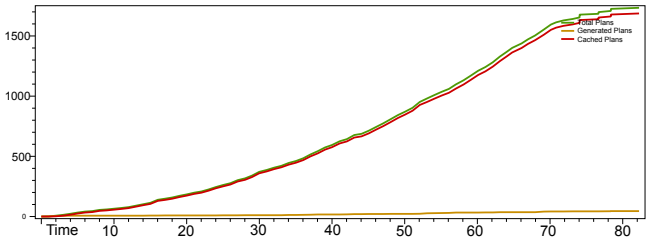

Figure 11: Plan reuse with CBP

We have also tested the capabilities of our CBP system in reusing and adapting plans. We have started the simulation with no stored plans in the case base and observed the result. Figure 11 depicts the CBP performance, where with time, most of the plans are reused by the CBP process. Left axis represents the number of generated plans; the bottom axis represents execution time. The slight increase in dynamic plans over time is accounted to the dynamic nature of our virtual environment where agents often need to come up with new ways of satisfying their goals, due to missing resources.

Concerning the quality of plans, we have performed experiments measuring their average length and the average number of obtained resources. Using plans from strict cases, we were getting almost the same results as when using dynamic planning. Allowing plan merging we were initially obtaining some large results (20 items+), yet, most of the plans were acceptable, i.e. no significant number of unnecessary actions was performed. Using plan extension heuristics, we were able to cut down the plan size of all plans to acceptable size, with agents rarely performing some additional actions.

As a result, our CBP approach allows agents to dynamically create believable plans from the pre-generated database of plans with limited impact on system performance. Using Virtual Institutions, it reduces the necessity to program individual agents and allows to define behaviour on group (society) level using declarative approach. Thus we can simulate agent performance in Vittoria before deploying it to Unity 3D and simulation designers can focus on delivering believable art, automating most of the technical issues.

Planning	Plan Length	Resources
Dynamic planning	6.45	4.3
Strict cases	6.48	4.3
Plan merging (no heuristics)	6.92	4.9
Plan merging (heuristics)	6.61	4.5

Table 3: Average plan length and obtained resources

6 CONCLUSION

We have showed how AI planning can be successfully applied for automating the process of populating historical and cultural Virtual Reality simulations with large numbers of virtual agents capable of complex action. One of the key techniques that helped us to achieve real-time performance was the use of case-based planning. Instead of generating a new plan every time an agent must achieve a particular goal, our approach is to first try to modify and combine existing (previously generated) plans. Only if this fails, meaning that no matching plan can be found, then agents try to achieve their goals by searching for a new plan. Introducing case-base planning allowed us to comfortably simulate over 300 agents and maintain a stable and reasonable frame-rate, while without case-based planning we were only able to simulate close to 40 agents.

REFERENCES

[1] Agnar Aamodt and Enric Plaza. 1994. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI communications* 7, 1 (1994), 39–59.
[2] AI Game Programmers Guild, http://gameai.com/wiki/index.php?title=The_Sims. 2011. The Sims. (2011).
[3] Anton Bogdanovych. 2007. *Virtual Institutions.* Ph.D. Dissertation. University of Technology, Sydney, Australia.
[4] Anton Bogdanovych and Tomas Trescak. 2016. Generating Needs, Goals and Plans for Virtual Agents in Social Simulations. In *Intelligent Virtual Agents IVA 2016 Proceedings, Los Angeles, CA, USA.* 397–401.
[5] Anton Bogdanovych and Tomas Trescak. 2017. *To Plan or Not to Plan: Lessons Learned from Building Large Scale Social Simulations.* Springer, 53–62.
[6] Anton Bogdanovych, Tomas Trescak, and Simeon Simoff. 2016. What makes virtual agents believable? *Connection Science* 28, 1 (2016), 83–108.
[7] Jaime Carbonell, Oren Etzioni, Yolanda Gil, Robert Joseph, Craig Knoblock, Steve Minton, and Manuela Veloso. 1991. PRODIGY: An Integrated Architecture for Planning and Learning. *SIGART Bull.* 2, 4 (July 1991), 51–55.
[8] Nicola Crüsemann, Margarete van Ess, Markus Hilgert, and Beate Salje. 2013. *Uruk. 5000 Jahre Megacity.*
[9] Kutluhan Erol, James Hendler, and Dana S Nau. 1994. HTN planning: Complexity and expressivity. In *AAAI*, Vol. 94. 1123–1128.
[10] Kutluhan Erol, Dana S. Nau, and V. S. Subrahmanian. 1995. Complexity, Decidability and Undecidability Results for Domain-independent Planning. *Artificial Intelligence* 76, 1-2 (July 1995), 75–88.
[11] Marc Esteva. 2003. *Electronic Institutions: From Specification to Development.* Ph.D. Dissertation. Institut d'Investigació en Intelligència Artificial (IIIA), Spain.
[12] Richard E Fikes and Nils J Nilsson. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence* 2, 3-4 (1971), 189–208.
[13] John Gauder. 2013. Crysis 3 cost $66 million to make, can next gen sustain such budgets? GameChup Video Games News at http://www.gamechup.com/crysis-3-cost-66-million-to-make-can-next-gen-sustain-such-budgets/. (2013).
[14] Kristian J Hammond. 1990. Case-Based Planning: A Framework for Planning from Experience. *Cognitive science* 14, 3 (1990), 385–443.
[15] Steve Hanks and Daniel S. Weld. 1995. A domain-independent algorithm for plan adaptation. *Journal of Artificial Intelligence Research* 2 (1995), 319–360.
[16] Mahdi Hannoun, Olivier Boissier, Jaime S. Sichman, and Claudette Sayettat. 2000. *MOISE: An Organizational Model for Multi-agent Systems.* Springer Berlin Heidelberg, Berlin, Heidelberg, 156–165.
[17] James A Hendler, Austin Tate, and Mark Drummond. 1990. AI planning: Systems and techniques. *AI magazine* 11, 2 (1990), 61.
[18] Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. 2002. *A Model for the Structural, Functional, and Deontic Specification of Organizations in Multiagent Systems.* Springer Berlin Heidelberg, Berlin, Heidelberg, 118–128.
[19] Subbarao Kambhampati and James A. Hendler. 1990. *A Validation Structure Based Theory of Plan Modification and Reuse.* Technical Report. Stanford, CA, USA.
[20] Abraham Harold Maslow. 1943. A theory of human motivation. *Psychological review* 50, 4 (1943), 370–396.
[21] Bernhard Nebel and Jana Koehler. 1992. *Plan modifications versus plan generation: A complexity-theoretic perspective.* Technical Report.
[22] Santi Ontanón, Kinshuk Mishra, Neha Sugandh, and Ashwin Ram. 2010. ON-LINE CASE-BASED PLANNING. *Computational Intelligence* 26, 1 (2010), 84–119.
[23] Stuart Russel and Peter Norvig. 2003. *Artificial Intelligence a Modern Approach.* Pearson Education International, Upper Saddle River, New Jersey, USA.
[24] Luca Spalzzi. 2001. A Survey on Case-Based Planning. *Artificial Intelligence Review* 16, 1 (Sept. 2001), 3–36.
[25] Tomas Trescak, Anton Bogdanovych, Simeon Simoff, and Inmaculada Rodriguez. 2012. Generating Diverse Ethnic Groups with Genetic Algorithms. In *Proceedings of the 18th ACM Symposium on Virtual Reality Software and Technology (VRST '12).* ACM, New York, NY, USA, 1–8.