# The Art of Balance: Problem-Solving vs. Pattern-Recognition

Martyn Lloyd-Kelly[1], Fernand Gobet[1] and Peter C.R. Lane[2]
[1]*Department of Psychological Sciences, University of Liverpool, Liverpool, L69 3BX, UK*
[2]*School of Computer Science, University of Hertfordshire, College Lane, Hatfield, AL10 9AB, UK*
{*M.Lloyd-Kelly*}, {*Fernand.Gobet*}*@liverpool.ac.uk, peter.lane@bcs.org.uk*

Keywords: Agents, Simulation, Dual-Process Theory, Reinforcement Learning, Pattern Recognition, Chunking

Abstract: The dual-process theory of human cognition proposes the existence of two systems for decision-making: a slower, deliberative, "problem-solving" system and a quicker, reactive, "pattern-recognition" system. The aim of this work is to explore the effect on agent performance of altering the balance of these systems in an environment of varying complexity. This is an important question, both in the realm of explanations of expert behaviour and to AI in general. To achieve this, we implement three distinct types of agent, embodying different balances of their problem-solving and pattern-recognition systems, using a novel, hybrid, human-like cognitive architecture. These agents are then situated in the virtual, stochastic, multi-agent "Tileworld" domain, whose intrinsic and extrinsic environmental complexity can be precisely controlled and widely varied. This domain provides an adequate test-bed to analyse the research question posed. A number of computational simulations are run. Our results indicate that there is a definite performance benefit for agents which use a mixture of problem-solving and pattern-recognition systems, especially in highly complex environments.

## 1 INTRODUCTION

The notion of a "dual-process" cognitive system proposes that human-beings are equipped with two systems capable of creating, retrieving and using "productions" (a prescribed action for a particular environment state) to achieve the agent's relevant goal(s) (Evans, 2008; Sloman, 1996). Psychological validity of this system is buttressed by human experimental evidence (de Wit and Dickinson, 2009; Gillan et al., 2011) and implementations in computational cognitive architectures designed to emulate and explain human cognition (Sun et al., 2005). Dual-process system theory suggests that a person may either use a formal logic-like "problem-solving" system that is slow and deliberative or a quicker "pattern-recognition" system that uses judgments of pattern similarity to try and solve the issue at hand (Sloman, 1996). Tension between use of problem-solving and pattern-recognition to solve problems has been identified by many (Hesketh, 1997; Zeitz, 1997), resulting in the proposal that domain experts are inflexible problem-solvers, since they are so entrenched in tried-and-tested paradigms (Sternberg, 1996; Simonton, 1999). This has been proven to be true, but only to a certain degree of expertise; once an above-average level of knowledge has been acquired about a domain, the so-called "Einstellung Effect" is removed (Bilalić et al., 2008).

An analysis of the potential effects on performance by weighting the usage of these systems in particular complexities of a stochastic environment is lacking, however. So, in this paper, we provide a quantitative investigation into what balance of problem-solving and pattern-recognition is most effective when these systems are encapsulated in a human-like computational architecture of cognition, situated in an environment whose complexity can vary considerably and where precise compile-time prescriptions of optimal action selection using techniques such as Markov Decision Processes are implausible. We will compare three ways of making decisions: pure problem solving, pure pattern-recognition and a mixture of the two methods. Our results are especially interesting for those who intend to design robust, effective systems capable of learning information autonomously and making use of this information to improve decision-making quality during run-time in reactive, sequential decision-making tasks.

Section 2 discusses the simulation environment in detail and justifies its applicability, whilst section 3 presents a relevant overview of human cognition, and discusses the computational architecture in detail.

Section 4 covers the implementation details of the agents, before section 5 outlines the simulations run to gather data to answer the research question posed. Section 6 provides details of the results obtained and a discussion of how these results may have emerged. The paper concludes with section 7, containing the salient points raised by the simulation results, their implications for the current state of the art, and some future directions of research.

## 2 SIMULATION ENVIRONMENT

Agents are situated in the Tileworld environment (Pollack and Ringuette, 1990), which typically comprises a two-dimensional grid of homogeneously-sized squares containing a number of tiles, holes and agents that exist for a finite period of time. An agent's main goal is to push tiles into holes, earning the agent a reward. The actions and goal achievement of agents are episodic and delayed since, in some cases, the agent's goal will not be achieved by performing one action; several may be required. Only one tile can be pushed at any time by an agent. For example, if an agent has two tiles to its east on consecutive squares, it is not able to push the tile closest to itself east since it is blocked by the tile two squares to the east. Explicit obstacles have not been included since a square can only be occupied by one tile, hole or agent at any time so these objects act as natural obstacles.

Tileworld's intrinsic and extrinsic environmental complexity can be precisely controlled by modifying certain parameter values. Extrinsic complexity can be altered by increasing/decreasing the number of players present in the environment whilst intrinsic complexity is controlled by parameters that define when new tiles and holes can be created, the probability of a new tile or hole being created and how long these artifacts exist for before being removed.

Depending upon the size of a Tileworld, the overall complexity of the environment can be prodigious. The total number of states possible in a simplified version of Tileworld consisting of a total of $n$ squares, one agent and only holes (no tiles or obstacles) is $n \cdot 2^n$. The base 2 in the $2^n$ expression term is derived from the fact that squares may be empty or occupied by at most, one environment object. Optimal policy calculations for a computer with reasonable resources using a Markov Decision Process (hereafter referred to as MDP) in this simplified version of Tileworld becomes intractable when $n = 16$, or $n = 25$ (Simari and Parsons, 2004). In comparison, our Tileworld is a two-dimensional grid that "wraps" (grid edges are not strict boundaries), $n = 1225$ and there are 4 classes of objects that an agent may encounter: another agent, a tile, a hole and an empty square, resulting in $1225 \cdot 4^{1225}$ possible states. We assert that this spectrum of complexity and the ability to exert fine-grained control over parameters that manage this complexity bestows a suitable environment to analyse what balance of problem-solving and pattern-recognition system use maximises agent performance given differing degrees of environmental complexity.

## 3 COGNITIVE ARCHITECTURE

In studying human cognition, much scrutiny has been focused upon explaining expert behaviour in complex domains; chess, in particular, has benefited from an investment of such effort. Research of this type has identified that the difference in performance between chess masters and amateurs does not hinge upon mental ability (search space depth differs only minimally between masters and amateurs, for example) but rather on the breadth and quality of knowledge possessed by masters (de Groot and Gobet, 1996).

The total amount of information in chess has been calculated to contain 143.09 bits of information (Jongman, 1968) or $10^{43}$ positions ($2^{143.09}$). However, some of these positions are redundant or implausible; rectified calculations give a total space of 50 bits of information or $10^{15}$ positions (de Groot and Gobet, 1996). One of the most promising theories that accounts for the ability of chess masters to learn and retain such large[1] databases of information, given known limitations on human cognition is *chunking* theory. Chunking suggests that aggregated environmental information or "chunks" (Chase and Simon, 1973) are used to store and improve new or existing information in memory. Computational cognitive models that implement chunking have closely mimicked the behaviour of humans in numerous domains (see section 3.1 for details).

With regard to decision-making, chess masters demonstrate a penchant for pattern-recognition; masters will recognise key features of certain board configurations extremely quickly (de Groot, 1978) and use typical moves despite the existence of other relatively uncommon solutions that are more optimal (Saariluoma, 1994; Bilalić et al., 2008). This would indicate that when an adequate knowledge base exists for the current domain, pattern-recognition is the preferred *modus operandi* for action selection. However, given that there may exist many possible

[1] With respect to both the number of positions and the amount of information in each position.

solutions for a particular situation, how are actions selected using pattern-recognition?

One proposal is that productions are assigned optimality ratings by a process akin to reinforcement learning (Sutton and Barto, 1998) whose presence in human cognition has been extensively validated (Erev and Roth, 1998; Holroyd and Coles, 2002). These ratings reflect an assessment of "rationality": non-optimal productions should be suppressed more than optimal ones (Miyazaki et al., 1994). When applied to domains like Tileworld, where an agent's actions and goal achievement are episodic, rating production optimality entails "discounted rewards": productions executed closer to the time a reward for goal achievement is received are assigned greater shares of the reward than actions performed further in the past according to a discount factor $\beta$, $(0 < \beta < 1)$ (Grefenstette, 1988). These reward shares are then usually translated directly into optimality ratings. Human production selection is also non-deterministic; a maximally optimal production is not guaranteed to be selected given several less optimal alternatives (Raza and Sastry, 2008; Dayan and Daw, 2008).

Consequently, we have combined the CHREST architecture (Gobet, 1993; de Groot and Gobet, 1996) with the "Profit Sharing with Discount Rate" (PSDR) reinforcement learning theory (Arai et al., 2000) and the "Roulette" selection algorithm (Baker, 1987); These elements are discussed further in sections 3.1, 3.2 and 3.3.

## 3.1 CHREST

CHREST is an example of a symbolic computational architecture (Samsonovich, 2010) and is the foundation of the cognitive architecture implemented in this paper. CHREST uses chunking to create well-organised, extensible knowledge-bases to allow human-like storage and retrieval of memory. CHREST also provides functionality to create productions by forming links between chunks and is capable of storing optimality ratings for productions by associating numeric values with these links. CHREST's validity as a theory of human-like cognition has been proven in a variety of domains, including board games (de Groot and Gobet, 1996) , implicit learning (Lane and Gobet, 2012) and natural language acquisition (Freudenthal et al., 2009) .

The version of CHREST used comprises two main components: short-term memory (STM) and long-term memory (LTM) which use "patterns" as input that can be combined to produce chunks. Patterns and chunks are created and interpreted by an agent's input/output component (discussed in section 4.1) and are stored in LTM according to their specified modality: action, visual or verbal[2].

LTM is composed of a discrimination network that acts as a retrieval device and a similarity function; its role is analogous to the hidden layers of a connectionist network, or the RETE network of Soar (Laird, 2012). Chunks are stored within *nodes* that are connected using *test links*, with a chunk input to LTM, $\theta$, used as a test. The network is traversed by sorting $\theta$ from the root node along matching test links until a leaf node is reached or no further test links match. If $\theta$ does not match the chunk retrieved from the node reached after traversal, $\vartheta$, the discrimination network is modified using one of two learning mechanisms. If $\theta$ contains patterns that are common to $\vartheta$ along with some additional patterns, then CHREST attempts to add one of these new patterns to $\vartheta$ – this operation is called *familiarisation*, and increases the size of chunks. If $\theta$ contains different information than $\vartheta$ then a new test link is added, using some of the different information in $\theta$ to form the test – this operation is called *discrimination*, and increases the number of chunks stored in LTM. More details of these learning mechanisms can be found in earlier publications (Gobet et al., 2001; Lane and Gobet, 2012).

STM consists of two fixed-length first-in-first-out lists of action and visual modality. When actions are performed in response to visual states, the relevant action and visual patterns are retrieved from STM and paired to create an "episode" (created by an agent's input/output component, see section 4.1). These episodes are stored in a fixed-length first-in-first-out "episodic memory" that enables correct operation of the PSDR reinforcement learning theory (discussed in section 3.2) and the ability to create and modify productions in LTM. Productions are implemented using hash map data structures contained in visual nodes *only*; keys contain pointers to action nodes and values denote the optimality rating of the production. Productions always begin at visual nodes and point to action nodes since actions are always produced in response to vision. Two broad types of productions can exist in LTM: productions containing a visual chunk and an explicit action chunk and productions containing a visual chunk and an action chunk that prescribes usage of the problem-solving system. Differences in how these two production types are handled creates the three different agent types (see section 4.4) mentioned.

The time taken to familiarise, discriminate and add productions can be set at run-time; note that performing one these processes blocks performance of others. This means that learning is slow at first but

[2] Chunks of verbal modality are not utilised here.

increases as the agent interacts more with the environment to create a very human-like model of cognition. Furthermore, attempts to create productions that already exist are ignored.

### 3.2 Profit Sharing with Discount Rate

PSDR was chosen as a reinforcement learning theory for three reasons: first, it can be used in domains where mathematical modeling of the domain is intractable (a property of the version of Tileworld implemented). Second, PSDR is rational in the sense defined earlier since it implements the notion of discounted rewards. Third, PSDR's effectiveness in enabling rational production optimality rating has been validated by others (Arai et al., 2000) whose aim is to grant agents with the ability to autonomously learn robust and effective productions in uncertain, dynamic, multi-agent domains, similar to the version of Tileworld used here.

PSDR uses a "credit assignment function" (see equation 1) to calculate production optimality ratings, $P_\sigma$. For example: at time $t$, an agent executes an action in response to the current environment state generating a production, $P_t$. At time $t+1$, the agent executes another action in response to the current environment state, producing another production, $P_{t+1}$, and continues this cycle until it receives a reward, $R$, at time $T$. At time $T$, the agent's episodic memory will contain the following (simplified) contents: $(P_t, P_{t+1}, \ldots P_T)$. If $R = 1$ and the discount rate $\beta = 0.5$, $P_T$ receives 1 as credit, $P_{T-1}$, receives 0.5 as credit, $P_{T-2}$ receives 0.25 as credit etc. The credit generated for a production is then added to that production's current optimality rating.

$$P_\sigma = P_\sigma + (R \cdot \beta^{T-t}, (0 < \beta < 1)) \quad (1)$$

### 3.3 Roulette Algorithm

The Roulette algorithm uses production optimality ratings generated by PSDR and stored by CHREST, to select an action for execution given a state of the environment. Equation 2, generates a value, $\omega$, for a candidate production, $P$, from $P$'s optimality rating, $P_\sigma$, divided by the sum of each candidate production's optimality rating, $P_\sigma^n$ to $P_\sigma^N$. Candidate productions are then ordered according to $\omega$, from lowest to highest, and used to create a range of values where productions with greater $\omega$ values occupy greater ranges. Finally, a random number, $R$, is generated where $0 < R < 1$ and used to select a production. Therefore, productions with greater optimality ratings will have a larger $\omega$ and therefore, a greater probability of being selected. Other candidate productions still have a chance of being selected hence, the algorithm is non-deterministic.

$$\omega = P_\sigma / \sum_{n=1}^{N} P_\sigma^n \quad (2)$$

## 4 AGENT IMPLEMENTATION

The agents implemented are endowed with a combination of the cognitive architecture described in section 3 with an input/output component separate from the cognitive architecture. Agents are goal-driven, are not capable of communicating or explicitly cooperating with one another to achieve these goals and can only see a limited portion of the total environment. Agent sight is controlled by a parameter that takes a number as input to indicate how many squares north, east, south and west the agent can "see". We keep the value of this parameter constant at 2 since agent performance should not be affected by differences in "physical" capabilities and this value limits visual input. This is important since larger values may result in the agent constantly learning (this blocks other mental processes, see section 3.1). Limiting agent sight to 2 squares also allows the agent to see past a tile so that its ability to be pushed can be determined.

Given the research discussed in section 3, problem-solving and pattern-recognition systems are never active simultaneously. After observing the current state of the environment, the pattern-recognition system is consulted first; if no actions are proposed by this system, the problem-solving system is used instead. Implementing system usage in this way classifies the dual-process system implemented as "modular" (Lane and Gobet, 2012) and extends CHREST's existing functionality.

Figure 1 illustrates how the major components in the agent architecture are connected. The operation of the input/output component is outlined in section 4.1 before the problem-solving and pattern-recognition systems are discussed in sections 4.2 and 4.3, respectively. Agent types are then delineated in section 4.4 and the main execution cycle for agents is provided in section 4.5.

### 4.1 Input/Output Component

Since agents use CHREST and are situated in a particular environment, the ability to encode and interpret CHREST compatible chunks is required; such functionality is provided by an agent's input/output

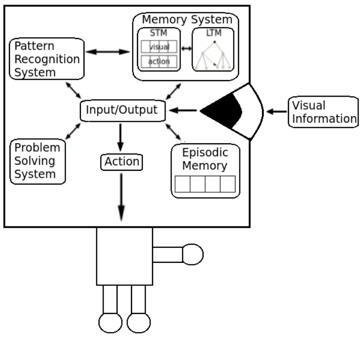



Figure 1: Agent Architecture.

component. Visual chunks are produced by the input element of the input/output component and are sent *to* LTM, action chunks are sent *from* the problem-solving or pattern-recognition system to the output element of the input/output component and are translated into operations that the agent can execute.

Visual and action chunks are composed of three-item tuples called "item-on-square" patterns that contain an identifier string and two numbers. For visual patterns the identifier string represents Tileworld objects ("T" for tile, "H" for hole and "A" for other agents), while the first and second numbers represent how many squares to the north and east of the agent the object is, respectively[3]. The visual pattern `< [T 1 2] >` states that there is a tile (T) located 1 square north (1) and 2 squares east (2) of the agent's current location. For action patterns, the identifier string represents an action ("MR" for move-randomly, "MAT" for move-around-tile, "MTT" for move-to-tile, "PT" for push-tile and "RS" for remain-stationary), while the two numbers represent the compass direction the agent should face (0: north, 90: east etc.) and the number of squares that should be moved by the agent and/or tile, respectively. For example, the action pattern `< [PT 90 1] >` states that the agent should face east (90) and push the tile there (PT) 1 square (1) in that direction.

Episodes (see section 3.1) are also encoded by an agent's input/output component and contain four pieces of information: a visual-pattern, $\upsilon$, an action-pattern, $\alpha$ (executed by the agent in response to $\upsilon$), the time $\alpha$ was executed (required by PSDR, see section 3.2) and whether $\alpha$ was produced by the problem-solving or pattern-recognition system (enables correct operation of pattern-recognition system variations, see section 4.3). If an agent saw a tile 1 square to the north (`[T 1 0]`) and used its problem-solving system (`true`) to generate a push tile north by 1 square action (`[PT 0 1]`) that was executed at time 110 (`110`), the episode created would be: `[ [T 1 0] [PT 0 1] 110 true ]`.

[3] South and west are represented by negative numbers.

## 4.2 Problem-Solving System

The problem-solving system takes visual chunks as input and generates action chunks that are used as input to the agent's LTM to initiate learning and to the agent's input/output component so it can be converted into executable instructions so the agent can act. Action chunks generated by the problem-solving system are intended to achieve the agent's currently active goal. However, an agent's goals are not explicitly represented in any data structure available to the agent. Therefore, the goal to be achieved is inferred by analysing the information contained in a visual-pattern that is passed as input. The result of this analysis is used to run one of three hand-coded procedures: "move-randomly", "secure-tile" or "push-tile-to-hole".

Note that we have conflated the concepts of "goal" and "environment state" because actions produced by the problem-solving system are used to create productions in an agent's LTM. These productions are then used as input to the agent's pattern-recognition system (see section 4.3) whose operation is intended to be analogous to habitual behaviour in human-beings: behaviours become habitual when they are frequently selected in response to particular *goals* being activated (Aarts and Dijksterhuis, 2000). Activation of goals can be inferred to occur after analysing input from the environment as in "Belief-Desire-Intention" decision-making architectures (Bratman, 1987). Therefore, explicitly considering goals is redundant since environment states and goals have a simple one-to-one mapping.

There are three sub-goals that may need to be achieved in order to achieve the agent's main goal of "fill-hole-with-tile". These are: "find-tile", "secure-tile" and "find-hole". Each problem-solving procedure can output one of 17 action chunk classes to help achieve these goals: "move-randomly", "move-to-tile", "move-around-tile", "push-tile" (of which there are four variations each: north, south, east and west) and "remain-stationary" (of which there are no variations). The problem-solving system follows the procedure outlined below given some visual chunk, $V$ as input. Active goals are highlighted using `fixed-width` font, procedure entry points are highlighted in **bold** and actions generated are highlighted in *italics*. Note that "adjacent" is defined as an object being on a square immediately north, east, south or west of the object referred to.

1. Agent is surrounded i.e. squares immediately

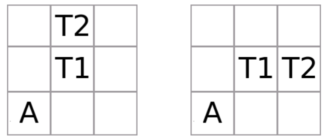
north, east, south and west of the agent are occupied by non-movable tiles, holes or other agents; *remain-stationary* generated.

2. $V$ indicates that the agent is not surrounded and tiles and holes are nearby. Determine closest hole to the agent, $H$, and tile that is closest to $H$, $T$.
   - $T$ is adjacent to agent and can be pushed closer to $H$ from agent's current position; `fill-hole-with-tile` activated, **push-tile-to-hole** run, *push-tile* generated.
   - $T$ is adjacent to agent but can't be pushed closer to $H$ from agent's current position; `secure-tile` activated, **secure-tile** run, *move-around-tile* generated.
   - $T$ is not adjacent to agent; `secure-tile` activated, **secure-tile** run, *move-to-tile* generated.
3. $V$ indicates that agent is not surrounded, tiles are nearby but holes are not. Determine distance from $T$ to agent.
   - $T$ is adjacent to agent; `find-hole` activated, **push-tile-to-hole** run, *push-tile* generated.
   - $T$ is not adjacent to agent; `secure-tile` activated, **secure-tile** run, *move-to-tile* generated.
4. $V$ indicates that the agent is not surrounded but can't see any tiles; `find-tile` activated, **move-randomly** run, *move-randomly* generated.

An important point to note is that procedures generate actions non-deterministically in some circumstances. Consider the environment states in figures 2(a) and 2(b). The goal of agent $A$ in both states is "secure-tile", specifically, tile $T1$, so it runs the "secure-tile" procedure to generate an action to try and achieve this goal. The optimal action in the case of figure 2(a) is for $A$ to move north around $T1$ so that it is able to push $T1$ to the east thus securing it. However, this action is non-optimal if the environment state in figure 2(b) is considered since $A$ can not push $T1$ east since $T2$ blocks $T1$ from this direction. Consequently, the optimal action in one state may be the non-optimal action in a similar state so in this case the secure-tile heuristic has a 0.5 probability of generating a "move-around-tile north" action and a 0.5 probability of generating a "move-around-tile-east" action if an agent's closest tile is situated to the north-east of the agent.

## 4.3 Pattern-Recognition System

The pattern-recognition system uses production optimality ratings and visual patterns as input to generate actions to perform. There are three crucial differences between this system and the problem-solving system:



(a) State 1. (b) State 2.

Figure 2: Environment state examples to justify non-determinism of actions by problem-solving procedures.

1. The pattern-recognition system is considered to be a part of CHREST since the components required to enable operation of the system are contained within CHREST.
2. The pattern-recognition system can not generate actions to create new productions, it can only select actions based upon existing productions.
3. The pattern recognition-system may have to choose an action from many potential productions depending upon how many productions exist for the current environment state i.e. visual-action mapping is not always 1:1.

The system first attempts to retrieve a visual chunk from LTM, $V'$, using the visual chunk input to the pattern-recognition system, $V$. If no visual chunks are retrieved execution of the system halts. Otherwise the optimality ratings of the productions associated with $V'$ are used as input to the Roulette selection algorithm where an action is then selected for execution. Note that, due to the intention of CHREST to simulate human cognition as closely as possible, it may be that $V \neq V'$ but $V'$ contains patterns common to itself and $V$. For example, if `< [T 1 0] [H 2 0] >` is passed to LTM as input but no LTM node contains the full chunk, `< [T 1 0] >` may be retrieved if it is stored in LTM.

## 4.4 Agent Types

In section 3.1 we mentioned that the way in which the two types of productions mentioned are handled creates three types of agents. This "handling" refers to whether a type of production can be created in LTM and whether the production can have its optimality rating calculated and stored.

- Agent Type 1 (pure problem-solver): neither type of production are created or rated in LTM.
- Agent Type 2 (pure pattern-recogniser): only productions containing visual chunks and explicit action chunks are stored and rated in LTM.
- Agent Type 3 (problem-solver and pattern-recogniser): both types of production are stored and rated in LTM. Note that, when the optimality

rating of the production represented in an episode is modified, this type of agent has a 50% chance of modifying either type of production if an episode stipulates that its action was generated by the problem-solving system (see section 4.5 for implementation details).

Agent type 1's balance between problem-solving system and pattern-recognition system use is weighted heavily in favour of the problem-solving system. Agent type 1 can be considered as a pure problem-solver since productions are never created or rated. Therefore, the pattern-recognition system can not be used since it does not have the information required to operate. Agent type 2's balance of problem-solving and pattern-recognition system use is weighted in favour of pattern-recognition since productions prescribing the use of the problem-solving system are never created or rated. Agent type 3 strikes an equal balance between problem-solving and pattern-recognition system use; the problem-solving system will be used more in initial decision-making (as it is for agent type 2) but as LTM expands, it may be that productions generated result in the problem-solving system being used more frequently.

### 4.5 Execution Cycle

The agent execution cycle runs for each agent after every time increment in the Tileworld environment. The agent begins by checking to see if there is an action, $\alpha$, currently loaded for execution. Note that agents have a specific *intention reconsideration* strategy implemented: when the current time, $T$, equals the time that the loaded action, $\alpha$, is to be performed, $t$, the agent generates a new visual-pattern, $V'$, and compares this to the visual-pattern, $V$, used to generate $\alpha$. If $V \neq V'$, the agent does not perform $\alpha$ and instead generates and loads a new action for execution based upon the information in $V'$. This, the execution cycle is:

1. $\alpha$ loaded for execution, check to see if $T = t$.
   (a) $T = t$: generate new visual chunk, $V'$, and compare this to the visual chunk used to generate $\alpha$, $V$.
      i. $V = V'$: perform $\alpha$.
         A. If $\alpha$ is not a "move-randomly" action chunk, create new episode and attempt to create a production in LTM between $V$ and $\alpha$ (biasing random movement is not beneficial given Tileworld stochasticity).
         B. If a point is scored, apply PSDR to each episode, modify production optimality ratings and clear episodic-memory. If episode indicates that action was generated using problem-solving, generate a random float $R$, $(0 <= R < 1)$. If $R <= 0.5$, modify optimality rating of the production containing the action chunk that prescribes use of problem-solving otherwise, modify optimality rating of the production containing the explicit action chunk[4].
      ii. $V \neq V'$: go to step 2.
   (b) $T \neq t$: stop current execution cycle.
2. No action loaded for execution:
   (a) Generate visual-pattern, $V$.
   (b) Pass $V$ as input to LTM and attempt to learn.
   (c) Use $V$ to generate a new action, $\alpha$, using problem-solving or pattern-recognition system depending upon agent type.
   (d) Load $\alpha$ for execution and attempt to learn $\alpha$ by passing it as input to LTM.

## 5 SIMULATION DETAILS

To investigate what balance of problem-solving and pattern-recognition system use maximises agent performance given differing environmental complexities, 27 conditions were simulated and run. Conditions are representative of various degrees of instrinsic/extrinsic environmental complexity and agent types. For each condition, the average score of all agents were recorded together with average frequencies of problem-solving/pattern-recognition system use. Each condition was repeated 10 times to harvest a data set large enough to provide a robust analysis. Our null hypothesis states that different balances of problem-solving and pattern-recognition do not have any effect on the performance of agents and altering extrinsic and intrinsic environment complexity does not have any effect upon problem-solving or pattern-recognition use.

Intrinsic environment complexity is controlled by the values of the "hole-birth-probability", "tile-birth-probability", "hole-lifespan" and "tile-lifespan" parameters (see section 2); higher tile/hole birth prob. values and lower tile/hole lifespan values equate to greater complexity since more tiles/holes will appear but for shorter periods of time. One may expect the value for the "tile/hole-born-every" parameter to also be varied. However, the instrinsic complexity of the environment can be significantly modified by varying

[4] Since CHREST's actions are time-limited, both productions can not be reinforced simultaneously.

the values of the parameters mentioned. Values for the "tile/hole-birth-probability" parameters were derived by simply taking the median probability, 0.5, as the moderate complexity value and then taking the lowest/highest values possible without guaranteeing tile/hole birth since this would significantly skew the results. Mappings for the values that the "tile/hole-birth-probability" and "tile/hole-lifespan" parameters are set to for each level of environment complexity are provided below:

- Environment complexity: low
  - tile/hole-birth-probability: 0.1
  - tile/hole-lifespan: 80 seconds
- Environment complexity: moderate
  - tile/hole-birth-probability: 0.5
  - tile/hole-lifespan: 40 seconds
- Environment complexity: high
  - tile/hole-birth-probability: 0.9
  - tile/hole-lifespan: 20 seconds

Extrinsic environmental complexity is controlled by the number of players and can be set to either 2, 4 or 8. Increasing the number of players can be interpreted as increasing the environment complexity since environment resources are shared in Tileworld and a greater number of environment states may be encountered by an agent due to an increased number of interactions between other agents and resources.

All other variable values are kept constant; Table 1 provides the mappings of each independent variable to its type (CHREST/agent/environment), its value and a justification for why this value was selected (if applicable).

There are three major groups of conditions differentiated by the degree of intrinsic environment complexity used. These major groups then consist of a further three sub-groups of conditions differentiated by the degree of extrinsic environmental complexity (number of players) present. Finally, each sub-group consists of three sub-sub-groups differentiated by agent type (see section 4.4). For example, extrinsic environment complexity is set to 1 (low) in conditions 1-9; number of players is set to 2, 4 and 8 for conditions 1-3, 4-6 and 7-9, respectively; the type of agents in each condition is set to **1** for conditions 1, 4 and 7, **2** for conditions 2, 5 and 8 and **3** for conditions 3, 6 and 9.

# 6 RESULTS AND DISCUSSION

All results are analysed using a $3 \times 3 \times 3$ analysis of variance (ANOVA), with environment complexity, number of agents and agent type as between-subject variables. As mentioned in section 5 we have collected data for three dependent variables: average score, average frequency of problem-solving system use and average frequency of pattern-recognition system use. The section is split into two subsections: section 6.1 discusses results concerning average scores and section 6.2 discusses results concerning average frequency of problem-solving and pattern-recognition system use. Both null hypotheses stated in section 5 are disproved.

## 6.1 Average Scores

Figures 3, 4 and 5 show average scores achieved by each agent type for each degree of environment complexity organised by number of players. The three main effects were statistically significant: environment complexity, $F(2,243) = 2,437.7$, number of agents, $F(2,243) = 16.6$, and agent type, $F(2,243) = 70.8$ and all $p < 0.001$.

Irrespective of the number of players and complexity used, the average score achieved by agent type 2 is either approximately equal or greater than the average score of agent type 3. Agent type 1 consistently achieves the lowest average scores. By increasing the number of players, average scores are decreased for each agent type whilst increasing environment complexity causes average scores to increase for each agent type. The only exception to this trend are the average scores obtained by agent type 1 when there are 4/8 players in the environment and environment complexity is increased from 2 to 3. In these conditions, agent type 1's average score either remains equal or decreases.

These results are explained by considering the ef-

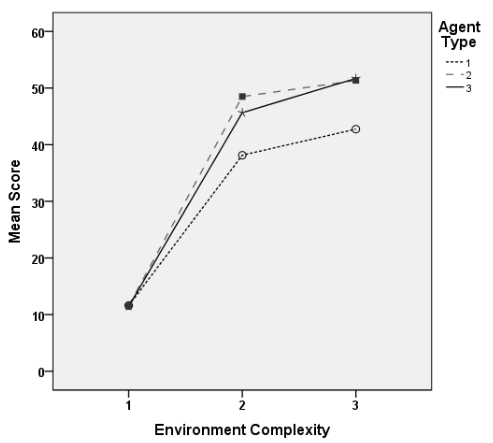



Figure 3: Average agent type scores for each level of environment complexity when 2 players are present in Tileworld.

Table 1: Independent variable names, type (agent, CHREST, environment), value used and justification of value mappings.

| Independent Variable | Type | Value | Justification |
|---|---|---|---|
| problem-solving-time | Agent | 1 sec | Equals value of the “tile/hole-born-every” parameters so planned actions may be reconsidered due to the appearance of a new tile or hole. |
| sight-radius | Agent | 2 | See section 4. |
| add-link-time | CHREST | 10 sec | Taken from (Simon, 1969). |
| discount-rate | CHREST | 0.5 | Median value selected. |
| discrimination-time | CHREST | 10 sec | Taken from (Simon, 1969). |
| episodic-memory-size | CHREST | 10 | Irrelevant productions less likely to have their optimality increased when agent achieves its main goal. |
| familiarisation-time | CHREST | 2 sec | Taken from (Simon, 1969). |
| pattern-recognition-time | CHREST | 0.2 sec | Taken from (Gobet, 1997). |
| hole-born-every | Env. | 1 sec | N/A. |
| play-time | Env. | 28800 sec | Allows pattern-recognition systems to learn enough information to be useful. |
| reward-value | Env. | 1 | Equals value of the “problem-solving-time” parameter. |
| tile-born-every | Env. | 1 sec | Equals value of the “problem-solving-time” parameter. |

fects of increasing the number of agents and environmental complexity upon intention reconsideration and resource availability. Increasing environmental complexity increases the amount of resources required to score, so average scores are likely to increase. However, increasing the number of agents elevates competition for resources and decreases their availability which can result in a general decline of scores. Furthermore, increasing both the number of players present in the environment and environmental complexity can create an increase in the total number of environment states encountered by agents since more opponents may be encountered and a greater number of interactions between agents and resources can occur. Ultimately, this increases the likelihood that intention reconsideration will be employed since it is more likely than an agent’s observable environment will change between action deliberation and performance.

Consequently, being able to perform more actions before the observable environment changes is the key determinant of performance. Since the interval of time for an agent generating an action using problem-solving and the environment potentially creating new tiles and holes is 1 second, intention reconsideration is more likely to be triggered when the problem-solving system is used and when extrinsic and intrinsic environment complexity is increased. Therefore, in the space of time where the environment remains static, agents that use pattern-recognition can perform up to

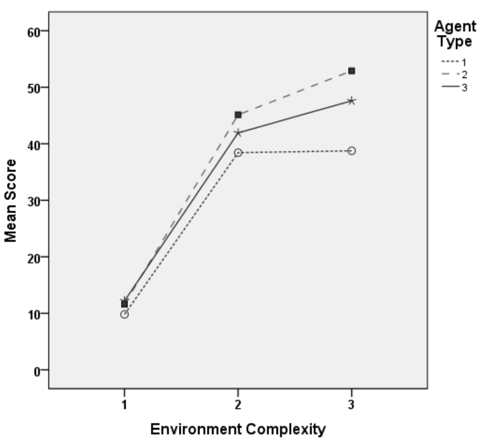



Figure 4: Average agent type scores for each level of environment complexity when 4 players are present in Tileworld.

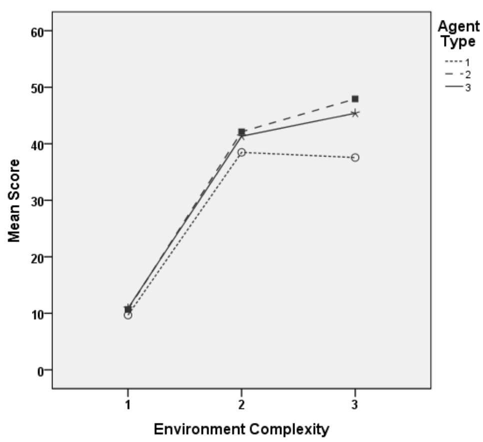



Figure 5: Average agent type scores for each level of environment complexity when 8 players are present in Tileworld.

5 actions whereas only 1 action can be performed using problem-solving. This increased volume of action enables agents to score more frequently when using pattern-recognition compared to problem-solving; demonstrated by comparing the performance of type 1 agents against types 2 and 3 generally and especially when agent type 1's performance is analysed in context of increasing environmental complexity.

## 6.2 Problem-Solving and Pattern-Recognition Use

Average frequencies of problem-solving and pattern-recognition system use are shown in figure 6. Results indicate a main effect of environment complexity, $F(2,243) = 295.2$, number of agents, $F(2,243) = 24.0$, and agent type, $F(2,243) = 1,511.7$ with all $p < 0.001$. As expected, agent type 1 never uses pattern-recognition whilst agent type 2 uses pattern recognition more frequently than agent type 3. By increasing the number of agents, pattern-recognition system use by agent types 2 and 3 decreases whilst increasing environment complexity increases the average frequency of pattern-recognition use by these agent types. These main effects are further qualified by the fact that environment complexity $\times$ agent type and number of agents $\times$ agent type are both statistically significant for agent types 2 and 3: $F(4,243) = 113.6$, $p < 0.001$ and $F(4,243) = 15.0$, $p < 0.001$, respectively.

Average frequency of problem-solving use yields results that tend to be the mirror-image of those obtained for average frequency of pattern-recognition use. This is expected since, if an agent does not use problem-solving it will use pattern-recognition

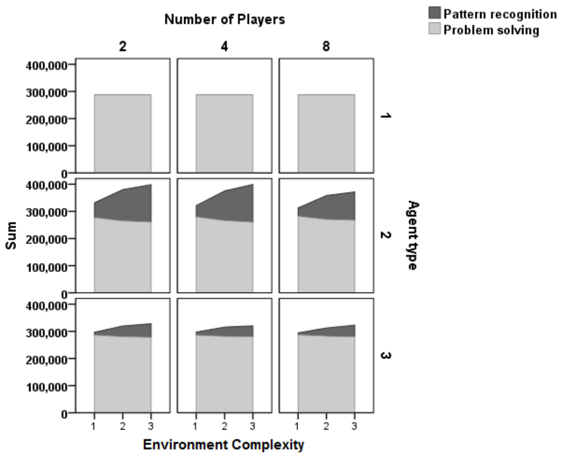



Figure 6: Average frequency of problem-solving/pattern-recognition system use for each agent type for all variations of environment complexity and number of players.

and vice-versa (if it is able to). There is a main effect of agent type, $F(2,243) = 1,512.1$, $p < 0.001$ with agent type 1 using problem-solving most frequently on average (as expected) and agent type 2 least frequently on average. The main effect of environment complexity, $F(2,243) = 295.2$, $p < 0.001$, reflects the fact that average frequency of problem-solving use tended to decrease with increasing complexity. Finally, the main effect of number of agents was significant, $F(2,243) = 24.0$, reflecting a small increase in problem-solving use as the number of agents increases. Again, these main effects are further qualified by the fact that environment complexity $\times$ agent type and number of agents $\times$ agent type are both statistically significant for agent types 2 and 3 ($F(4,243) = 113.6$, $p < 0.001$, and , $F(4,243) = 15.0$, $p < 0.001$).

When compared with results for average scores, problem-solving appears to have a negative effect upon performance when the number of players and environmental complexity is increased (see results for agent type 1 in figures 3-5 and 6), whilst pattern-recognition appears to positively affect performance (see results for agent types 2 and 3 in the aforementioned figures). Bias on pattern-recognition appears to produce better performance than balanced problem-solving and pattern-recognition, since agent type 2 has a higher average frequency of pattern-recognition use and achieves higher average scores than agent type 3 (as noted in section 6.1). Therefore, acting quickly and, potentially, sub-optimally in complex environments appears to be beneficial with regards to performance.

More frequent use of the problem-solving system on average as the number of players is increased can be explained by considering the effect that increasing the number of players has on environment dynamism (see section 6.1 for a discussion of this). The increase in interactions between agents and environment resources could decrease the potential space of environment states seen by an agent. This would be caused by other agents moving resources away from a particular agent, frequently resulting in a "blank" observable environment. This results in agent types 2 and 3 having less completely familiarised and associated chunks in their CHREST architectures, and hence a reduced use of their pattern-recognition systems.

The observed increase in pattern-recognition use as environment complexity is elevated is somewhat paradoxical to the above argument since this increase can also be explained by the consequences of heightened environmental dynamism. However, the increase in dynamism in this case stems from a consideration of intrinsic environment complexity rather than the number of agents present in the environment. Increasing intrinsic environment complexity increases the potential state space of the environment, resulting in increased discrimination and familiarisation for the CHREST architectures of agent types 2 and 3. Consequently, visual-action chunk association may be "crowded-out", intuitively implying that less pattern-recognition should occur. However, many resources are likely to exist at any given time in a highly complex environment so it is likely that an agent will encounter the same environment state frequently, increasing the number of completely familiarised chunks and associations in its CHREST architecture. Thus, it becomes more likely that an agent that can use pattern-recognition will do so.

## 7 CONCLUSIONS AND FUTURE WORK

In this paper we have described and implemented a novel computational architecture for agents using a modular dual-process architecture (Lane and Gobet, 2012). This architecture consists of a problem-solving and pattern-recognition system, created using a combination of environment-specific procedures for generating actions: the CHREST architecture, the PSDR algorithm and the Roulette selection mechanism. The system is capable of representing different balances of problem-solving and pattern-recognition use: pure problem-solving, pure pattern-recognition and a mixture of both. These balances of the two systems were embodied as three types of agent that were situated in the Tileworld environment. We used these agents to ascertain the effects of problem-solving and pattern-recognition system use on agent performance in this environment given differing degrees of intrinsic and extrinsic environmental complexity.

We discovered that use of pattern-recognition is beneficial to agent performance especially when intrinsic and extrinsic environment complexity is increased, whereas use of problem-solving is detrimental, due to the required time to solve problems. As overall environmental complexity increases, we further find that agents using pure problem-solving (that is, the complete absence of pattern-recognition) are further disadvantaged whereas agents that were more likely to use pattern-recognition performed best. Our results therefore demonstrate that an agent which can use both problem-solving and pattern-recognition is at an advantage in the complex, dynamic environment modeled and even more so when pattern-recognition is favoured. Essentially, these results would indicate that it is more important for an agent to deliberate upon actions quickly (resulting in potentially sub-optimal actions) rather than slowly (producing more optimal actions), in the Tileworld environment modeled at least. This is an interesting finding given that the Einstellung effect (Luchins, 1942) may be manifest in the agents that perform best; further simulations will be run to ascertain if this result still holds when the same agents used in this paper are allowed to exist in the same Tileworld environment for longer periods of time and when heterogeneous agent types can co-exist.

In future work, we also intend to consider different reinforcement-learning theories and action-selection algorithms, and compare these results to those delineated in this paper. In addition, we will consider alternative discount rates, and how altering the observable environment range affects performance. Furthermore, we aim to establish whether a more complex environment causes balanced problem-solvers and pattern-recognisers to gain an advantage over problem-solvers and pattern-recognisers whose pattern-recognition is favoured given that a greater range of possible solutions to problems may exist.

## REFERENCES


Aarts, H. and Dijksterhuis, A. (2000). Habit as knowledge structures: Automaticity in goal-directed behavior. *Journal of Personality and Social Psychology*, 78(1):53–63.

Arai, S., Sycara, K. P., and Payne, T. R. (2000). Experience-based reinforcement learning to acquire effective behavior in a multi-agent domain. In *Proceedings of the 6th Pacific Rim International Conference on Artificial Intelligence*, pages 125–135.

Baker, J. E. (1987). Reducing bias and inefficiency in the selection algorithm. In Grefenstette, J. J., editor, *Proceedings of the Second International Conference on Genetic Algorithms on Genetic Algorithms and Their Application*. L. Erlbaum Associates Inc.

Bilalić, M., McLeod, P., and Gobet, F. (2008). Inflexibility of experts - reality or myth? Quantifying the Einstellung effect in chess masters. *Cognitive Psychology*, 56(2):73–102.

Bratman, M. E. (1987). *Intentions, Plans, and Practical Reason*. Harvard University Press.

Chase, W. G. and Simon, H. A. (1973). Perception in chess. *Cognitive Psychology*, 4:55–81.

Dayan, P. and Daw, N. D. (2008). Decision theory, reinforcement learning, and the brain. *Cognitive, Affective and Behavioral Neuroscience*, 8(4):429–453.

de Groot, A. D. (1978). *Thought and Choice in Chess (First edition in 1946)*. Mouton, The Hague.

de Groot, A. D. and Gobet, F. (1996). *Perception and Memory in Chess: Heuristics of the Professional Eye*. Van Gorcum, Assen.

de Wit, S. and Dickinson, A. (2009). Associative theories of goal-directed behaviour: a case for animal-human translational models. *Psychological Research*, 73(4):463–476.

Erev, I. and Roth, A. E. (1998). Predicting how people play games: Reinforcement learning in experimental games with unique, mixed strategy equilibria. *The American Economic Review*, 88(4):pp. 848–881.

Evans, J. S. B. T. (2008). Dual-processing accounts of reasoning, judgment and social cognition. *Annual Review of Psychology*, 59:255–278.

Freudenthal, D., Pine, J. M., and Gobet, F. (2009). Simulating the referential properties of Dutch, German and English root infinitives in MOSAIC. *Language Learning and Development*, 15:1–29.

Gillan, C. M., Papmeyer, M., Morein-Zamir, S., Sahakian, B. J., Fineberg, N. A., Robbins, T. W., and de Wit, S. (2011). Disruption in the balance between goal-directed behavior and habit learning in obsessive-compulsive disorder. *American Journal of Psychiatry*.

Gobet, F. (1993). *Les mémoires d'un joueur d'échecs*. Editions Universitaires, Fribourg, Switzerland.

Gobet, F. (1997). A pattern-recognition theory of search in expert problem solving. *Thinking and Reasoning*, 3:291–313.

Gobet, F., Lane, P. C. R., Croker, S. J., Cheng, P. C.-H., Jones, G., Oliver, I., and Pine, J. M. (2001). Chunking mechanisms in human learning. *Trends in Cognitive Sciences*, 5:236–243.

Gobet, F. and Simon, H. A. (2000). Five seconds or sixty? Presentation time in expert memory. *Cognitive Science*, 24:651–82.

Grefenstette, J. J. (1988). Credit assignment in rule discovery systems based on genetic algorithms. *Machine Learning*, 3:225–245.

Hesketh, B. (1997). Dilemmas in training for transfer and retention. *Applied Psychology*, 46(4):317–339.

Holroyd, C. B. and Coles, M. G. (2002). The neural basis of human error processing: Reinforcement learning, dopamine, and the error-related negativity. *Psychological Review*, 109(4):679–709.

Jongman, R. W. (1968). *Het Oog Van De Meester*. Assen: Van Gorcum.

Laird, J. E. (2012). *The Soar Cognitive Architecture*. MIT Press.

Lane, P. C. R. and Gobet, F. (2012). CHREST models of implicit learning and board game interpetation. In Bach, J., Goertzel, B., and Ikle, M., editors, *Proceedings of the Fifth Conference on Artificial General Intelligence*, volume LNAI 7716, pages 148–157, Berlin, Heidelberg. Springer-Verlag.

Luchins, A. S. (1942). Mechanization in problem solving: The effect of einstellung. *Psychological Monographs*, 54(6):i–95.

Miyazaki, K., Yamamura, M., and Kobayashi, S. (1994). On the rationality of profit sharing in reinforcement learning. In *3rd International Conference on Fuzzy Logic, Neural Nets and Soft Computing*, pages 285–288. Korean Institute of Intelligent Systems.

Pollack, M. and Ringuette, M. (1990). Introducing the Tileworld: Experimentally evaluating agent architectures. In *Eighth National Conference on Artificial Intelligence*, pages 183–189. AAAI Press.

Raza, M. and Sastry, V. (2008). Variability in behavior of command agents with human-like decision making strategies. In *Tenth International Conference on Computer Modelling and Simulation*, pages 562–567.

Saariluoma, P. (1994). Location coding in chess. *The Quarterly Journal of Experimental Psychology*, 47A:607–630.

Samsonovich, A. (2010). Toward a unified catalog of implemented cognitive architectures. In *Proceedings of the 2010 Conference on Biologically Inspired Cognitive Architectures*, pages 195–244, Amsterdam, The Netherlands. IOS Press.

Simari, G. I. and Parsons, S. D. (2004). On approximating the best decision for an autonomous agent. In *Sixth Workshop on Game Theoretic and Decision Theoretic Agents*, Third Conference on Autonomous Agents and Multi-agent Systems, pages 91–100.

Simon, H. A. (1969). *The sciences of the artificial*. MIT Press, Cambridge, MA.

Simonton, D. K. (1999). *Origins of genius: Darwinian perspectives on creativity*. Oxford University Press, New York.

Sloman, S. (1996). The empirical case for two systems of reasoning. *Psychological Bulletin*, 119:3–22.

Sternberg, R. J. (1996). *The road to excellence: The acquisition of expert performance in the arts and sciences, sports, and games*, chapter Costs of expertise, pages 347–354. Hillsdale, NJ: Lawrence Erlbaum Associates.

Sun, R., Slusarz, P., and Terry, C. (2005). The interaction of the explicit and the implicit in skill learning: A dual-process approach. *Psychological Review*, 112(1):159–192.

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

Tversky, A. and Kahneman, D. (1983). Extensional versus intuitive reasoning: The conjunction fallacy in probability judgment. *Psychological Review*, 90(4):293–315.

Zeitz, C. M. (1997). *Expertise in context: Human and machine*, chapter Some concrete advantages of abstraction: How experts' representations facilitate reasoning, pages 43–65. Cambridge, MA: The MIT Press.