5.4 Imperfect, Real-Time Decisions

- Searching through the whole (pruned) game tree is too inefficient for any realistic game
- Moves must be made in a reasonable amount of time
- One has to cut off the generation of the game tree to some depth and the absolute terminal node values are replaced by heuristic estimates
- Game positions are rated according to how good they appear to be (with respect to reaching a goal state)
- A basic requirement for a heuristic evaluation function is that it orders the terminal states in the same way as the true utility function
- Of course, evaluation of game positions may not be too inefficient and the evaluation function should be strongly correlated with the actual chances of winning

Most evaluation functions work by calculating features of the state
- E.g., in chess the number of pawns possessed by each side could be one feature
- As game positions are mapped to the values of the chosen features, different states may look equivalent, even though some of them lead to wins, some to draws, and some to losses
- For such an equivalence class of states, we can compute the expected end result
- If, e.g., 72% of the states encountered in the category lead to a win (utility +1), 20% to a loss (0) and 8% to a draw (½), then the expected value of a game continuing from this category is:
  
  \[(0.72 \times 1) + (0.20 \times 0) + (0.08 \times \frac{1}{2}) = 0.76\]
Because the number of features and their possible values is usually high, the method based on categories is only rarely usable. Instead, most evaluation functions compute separate numerical contribution for each feature $f_i$ on position $s$ and combine them by taking their weighted linear function as the evaluation function:

$$\text{eval}(s) = \sum_{i=1}^{n} w_i f_i(s)$$

For instance, in chess features $f_i$ could be the numbers of pawns, bishops, rooks, and queens. The weights $w_i$ for these features, on the other hand, would be the material values of the pieces (1, 3, 5, and 9).

Adding up the values of the features involves the strong assumption about the independence of the features. However, e.g., in chess bishops are more powerful in the endgame, when they have a lot of space to maneuver. For this reason, current programs for chess and other games also use nonlinear combinations. For example, a pair of bishops might be worth slightly more than twice the value of a single bishop, and a bishop is worth more in the endgame than in the beginning. If different features and weights do not have centuries of experience behind them like in chess, the weights of the evaluation function can be estimated by machine learning techniques.
5.5 Stochastic Games

Games can include an explicit random element, e.g., by throwing a dice.

A board game with such an element is backgammon.

Although the player knows what her own legal moves are, she does not know what the opponent is going to roll and thus does not know what the opponent’s legal moves will be.

Hence, a standard game tree cannot be constructed.

In addition to max and min nodes one must add chance nodes into the game tree.

The branches leading from each chance node denote the possible dice rolls, and each is labeled with the roll and the chance that it will occur.

In backgammon one rolls two dice, so there are 6+15 distinct pairs and their chances of coming up are 1/36 and 1/18.

Instead of definite minimax values, we can only calculate the expected value, where the expectation is taken over all the possible dice rolls that could occur.

E.g., the expected value of a max node \( n \) is now determined as

\[
\max_{s \in S(n)} \mathbb{E}[\text{MM}(s)]
\]

In a chance node \( n \) we compute the average of all successors weighted by their probability \( P(s) \) (the required dice roll occurs)

\[
\sum_{s \in S(n)} P(s) \mathbb{E}[\text{MM}(s)]
\]

Evaluating positions in a stochastic game is a more delicate matter than in a deterministic game.
Including the element of chance increases the time complexity of game tree evaluation to $O(b^{n^m})$, where $n$ is the number of distinct dice rolls.

- In backgammon $n = 21$ and $b$ is usually around 20, but in some situations can be as high as 4000 for dice rolls that are doubles.
- Even if the search depth is limited, the extra cost compared with that of minimax makes it unrealistic to consider looking ahead very far for most stochastic games.
- Alpha-beta pruning concentrates on likely occurrences.
- In a game with dice, there are no likely sequences of moves.
- However, if there is a bound on the possible values of the utility function, one can prune a game tree including chance nodes.

7 LOGICAL AGENTS

- We now turn to knowledge-based agents that have a knowledge base $KB$ at their disposal.
- With the help of the $KB$ the agent aims at maintaining knowledge of its partially-observable environment and make inferences of the state of the world.
- Logic is used as the knowledge representation language.
- $KB$ consists of a set of sentences.
- Initially the agent’s $KB$ contains the background knowledge given in advance.
- The agent TELLs the $KB$ all its percepts and ASKs the $KB$ for what actions to take.
- Both TELL and ASK may involve logical inference – deriving new sentences from old.
Choosing an action based on the knowledge in the KB may involve extensive reasoning.

Also the information about the executed action is stored in KB.

Using a KB makes the agent amenable to a description at the knowledge level rather than giving a direct implementation for the agent.

One can build a knowledge-based agent by simply TELLing it what it needs to know.

Operating on the knowledge level corresponds to the declarative approach.

In the procedural approach one encodes the desired behaviors directly as program code.

---

### 7.2 The Wumpus World

- An agent operates in a $4 \times 4$ grid of rooms always starting in the square $[1,1]$, facing to the right.
- In the cave of rooms there is also one static wumpus (a beast) and a heap of gold, in addition rooms can contain bottomless pits.
- The agent’s possible actions are:
  - Move forward one square at a time,
  - Turn left or right by 90°,
  - Pick up the gold by grabbing, and
  - Shooting one single arrow in the direction it is facing.
- The agent dies a miserable death if it enters a square containing a pit or a live wumpus.
- The game ends in either the agent’s death or picking up of gold.
The locations of the gold and the wumpus are chosen randomly, with a uniform distribution, from the squares other than the start square [1,1].

In addition, each square other than the start [1,1] can be a pit, with probability 0.2.

- In a square directly (not diagonally) adjacent to the wumpus (W) the agent (A) will perceive a stench (S) and in one adjacent to a pit (P) the agent will perceive a breeze (B).
- The glitter of gold is perceived in the same square,
- walking into a wall, the agent perceives a bump, and
- when the wumpus is killed, its woeful scream that can be perceived anywhere in the cave.
• From the fact that there is no stench or breeze in [1,1], the agent can infer that the neighboring squares [1,2] and [2,1] are free of dangers.
• Moving forward to [2,1] makes the agent detect a breeze, so there must be a pit in a neighboring square [2,2] or [3,1] or both.
• At this point only square [1,2] is a safe unvisited square.
• The percept in square [1,2] is stench; hence
  • The wumpus cannot – by the rules of the game – be in [1,1]. It cannot be in square [2,2] (or we would have detected a stench in [2,1]), therefore the wumpus must be in [1,3].
  • The lack of breeze in [1,2] implies that there is no pit in [2,2], so this means it must be in [3,1].

7.3 Logic

• The syntax of the sentences constituting the KB is specified by the chosen knowledge representation language.
• In logic, the semantics of the language defines the truth of each sentence with respect to each model (possible world).

  • Sentence $\beta$ follows logically from sentence $\alpha$, $\alpha \models \beta$, if and only if (iff) in every model in which $\alpha$ is true, $\beta$ is also true.
  • In other words: if $\alpha$ is true, then $\beta$ must also be true.
  • We say that the sentence $\alpha$ entails the sentence $\beta$.

  • If a sentence $\alpha$ is true in model $m$, we say that $m$ is a model of $\alpha$.
  • Let $M(\alpha)$ denote the set of all models of $\alpha$.

  • Observe that $\alpha \models \beta$ if and only if $M(\alpha) \subseteq M(\beta)$. 

Consider the squares [1,2], [2,2], and [3,1] in the wumpus-world and the question whether they contain pits. This is a binary information, so there are $2^3 = 8$ possible models for this situation.

Because there is no breeze in [1,1] and in [2,1] there is a breeze, the models in which the KB is true are those that have a pit in [2,2] or [3,1] or both.

Let $\mathcal{J}_1$ be the conclusion “There is no pit in [1,2].”

The three models of the KB together with the model that has no pit in any of the three squares are the models of the conclusion $\mathcal{a}_1$.

In every model in which KB is true, $\mathcal{a}_1$ is also true.

Hence, KB $\models \mathcal{a}_1$; there is no pit in [1,2].

Let $\mathcal{a}_2$ be the conclusion “There is no pit in [2,2].”

In some models in which the KB is true, $\mathcal{a}_2$ is false.

Hence, KB $\not\models \mathcal{a}_2$.

The agent cannot conclude that there is no pit in [2,2] (nor that there is a pit in [2,2]).
Logic /3

• The logical inference algorithm working as described above is called model checking
• It enumerates all possible models to check that \( \alpha \) is true in all models in which KB is true

• If an inference algorithm \( i \) can derive \( \alpha \) from KB, we write \( \text{KB} \vdash i \alpha \)

• An inference algorithm that derives only entailed sentences is called sound (or truth-preserving)
• Another desired property of an inference algorithm is completeness: it can derive any sentence that is entailed

7.4 Propositional Logic

• Atomic sentences consist of a single proposition symbol \( P, Q, R, \ldots \)
• Each such symbol stands for a proposition that can be true or false
• Proposition symbols with fixed meanings: \( T \) is always true and \( F \) is always false
• Complex sentences are constructed from simpler ones using logical connectives
  • \( \lnot \) Negation. A literal is either an atomic sentence (a positive literal) or a negated atomic sentence (a negative literal)
  • \( \wedge \) (and) Conjunction the parts of which are conjuncts
  • \( \vee \) (or) Disjunction the parts of which are disjuncts
  • \( \Rightarrow \) Implication has a premise (or antecedent) and conclusion (or consequent)
  • \( \Leftrightarrow \) (iff) Equivalence (or biconditional)
• A BNF grammar of sentences in propositional logic:

Sentence → AtomicSentence | ComplexSentence
AtomicSentence → T | F | Symbol
Symbol → P | Q | R | ...
ComplexSentence → ( Sentence )
  | ¬ Sentence
  | (Sentence ∧ Sentence)
  | (Sentence ∨ Sentence)
  | (Sentence ⇒ Sentence)
  | (Sentence ⇔ Sentence)

To avoid using an excessive amount of parentheses, we agree the order of precedence for the connectives: ¬, ∧, ∨, ⇒, ⇔
• Hence, the sentence ¬P ∨ Q ∧ R ⇒ S is equivalent to the sentence
  ((¬ P) ∨ (Q ∧ R)) ⇒ S

• The semantics of propositional logic defines the rules for determining the truth value of a sentence with respect to a particular model
• In propositional logic, a model simply fixes the truth value for every proposition symbol
  • $M_1 = \{ P_{1,2} = F, P_{2,2} = F, P_{3,1} = T \}$
Truth Table

- The truth value of an arbitrary sentence can be computed recursively
  - \( F \) is false and \( T \) true in every model
  - The model assigns a truth value to every proposition symbol
  - The value of a complex sentence is determined by truth table

<table>
<thead>
<tr>
<th>( P )</th>
<th>( Q )</th>
<th>( \neg P )</th>
<th>( P \land Q )</th>
<th>( P \lor Q )</th>
<th>( P \Rightarrow Q )</th>
<th>( P \Leftrightarrow Q )</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td></td>
</tr>
</tbody>
</table>

For example, the sentence \( \neg P_{1,2} \land (P_{2,2} \lor P_{3,1}) \) evaluated in \( M_1 \) gives \( T \land (F \lor T) = T \land T = T \).

- A logical knowledge base \( KB \) which started as empty and has been constructed by operations \( Tell(KB, S_1), ..., Tell(KB, S_n) \) is a conjunction of sentences \( KB = S_1 \land ... \land S_n \).

- We can, thus, treat knowledge bases and sentences interchangeably.

- In the following the interpretation of proposition symbols is:
  - \( P_{i,j} \) is true if there is a pit in \([i, j]\)
  - \( B_{i,j} \) is true if there is a breeze in \([i, j]\)
Knowledge Base (1)

- Part of the background knowledge – i.e., the rules of the game – and the first percepts

\begin{align*}
R_1 & : \neg P_{1,1} \\
R_2 & : B_{1,1} \Leftrightarrow (P_{1,2} \lor P_{2,1}) \\
R_3 & : B_{2,1} \Leftrightarrow (P_{1,1} \lor P_{2,2} \lor P_{3,1}) \\
R_4 & : \neg B_{1,1} \\
R_5 & : B_{2,1}
\end{align*}