Distributed Calculus and Coordination

Emanuela Merelli

Lecture 4 - Agent & Utility function
We build agents in order to carry out tasks.
The task must be explicitly specified.
One way is to “write a program” for the agent to execute: Advantage we exactly knoe what the agent will do; the disadvantage we have exactly to think how to specify and forseen any circumstances.

Therefore, we want to tell agents what to do without telling them how to do it.

Define tasks *indirectly* via some kind of *performance measure*.
The first way to define a performance measure is to associate *utilities* with states of the environment.
Utility function

A utility is a numeric value representing how ‘good’ a state is: the higher the utility, the better.

- One possibility is to associate utilities with every environment states
- the task of the agent is then to bring about states that maximise the utility.
Utility over Environment States

Task specification

A task is a function

\[ u : E \rightarrow \mathbb{R} \]

which associated a real number with every environment state.

Given a performance measure we can define the overall utility of an agent in some particular environment in some ways:

- minimum utility of state on run - worst case
- maximum utility of state on run - best case
- sum of utilities of states on run - ...
- average of all state - medium case

The disadvantage of this approach is that it assigns utilities to local state, it is difficult to specify a long term view when assigning utilities to individual states.
Another possibility is to assign a utility not to individual states, but to runs themselves.

This approach is more appropriate for agents that operate independently over long period of time.

Task specification

A task is a function

\[ u : \mathcal{R} \rightarrow \mathbb{R} \]

which associated a real number to runs.
Tileworld example - Pollack, 1990

Tileworld

Given a two-dimensional grid environment on which there are agents, tiles, obstacles and holes. An agent can move in four directions, up, down, left, right, and if it is located next to a tile, it can push it.

- An obstacle is a group of immovable grid cells: agents are not allowed to travel freely through obstacles.
- Holes have to be filled up with tiles by agents.
- An agent scores points by filling holes with tiles, the aim being to fill as many holes as possible.
- The Tileworld is an example of dynamic environment: starting in some randomly generated world state, based on parameters set by the experimenter, it changes over time in discrete steps, with random appearance and disappearance of holes.
The performance of an agent in the Tileworld is measured by running the Tileworld testbed for a predetermined number of time steps, and measuring the number of holes that the agent succeeds in filing.

**Utility function for Tileworld**

The performance of an agent on some particular run is then defined as

\[
u(r) = \frac{\text{number of holes filled in } r}{\text{number of holes that appeared in } r}\]
Maximizing expected utility

- Write $P(r | Ag, Env)$ to denote probability that run $r$ occurs when agent $Ag$ is placed in environment $Env$.
- Note that
  \[
  \sum_{r \in \mathcal{R}(Ag, Env)} P(r | Ag, Env) = 1
  \]
- Assume that the utility $u$ has some upper bound to the utilities that is assigned. There exists a $k \in \mathbb{R}$ such that for all $r \in \mathcal{R}$ we have $u(r) \leq k$.

Expected utility

The expected utility of agent $Ag$ in environment $Env$ (given $P, u$), is then:

\[
EU(Ag, Env) = \sum_{r \in \mathcal{R}(Ag, Env)} u(r)P(r | Ag, Env)
\]
The optimal agent $A_{g_{\text{opt}}}$ in an environment $Env$ is the one that maximizes expected utility:

$$A_{g_{\text{opt}}} = \arg \max_{A_g \in AG} EU(A_g, Env)$$

The fact that an agent is optimal does not mean that it will be best; only that on average, we can expect it to do best.

The idea is similar to the approach proposed within the “decision theory” (see Russell and Norvig 1995).
Example

Consider the environment $Env_1 = \langle E, e_0, \tau \rangle$ defined as follows:

$E = (e_0, e_1, e_2, e_3, e_4, e_5)$

$\tau(e_0 \xrightarrow{\alpha_0}) = (e_1, e_2)$

$\tau(e_0 \xrightarrow{\alpha_1}) = (e_3, e_4, e_5)$

There are two agents possible with respect to this environment:

$Ag_1(e_0) = \alpha_0$

$Ag_2(e_0) = \alpha_1$
The probabilities of the various runs are as follows:

\[ P(e_0 \xrightarrow{\alpha_0} e_1 \mid Ag_1, Env_1) = 0.4 \]
\[ P(e_0 \xrightarrow{\alpha_0} e_2 \mid Ag_1, Env_1) = 0.6 \]
\[ P(e_0 \xrightarrow{\alpha_1} e_3 \mid Ag_2, Env_1) = 0.1 \]
\[ P(e_0 \xrightarrow{\alpha_1} e_4 \mid Ag_2, Env_1) = 0.2 \]
\[ P(e_0 \xrightarrow{\alpha_1} e_5 \mid Ag_2, Env_1) = 0.7 \]

Assume the utility function \( u_1 \) is defined as follows:

\[ u_1(e_0 \xrightarrow{\alpha_0} e_1) = 8 \]
\[ u_1(e_0 \xrightarrow{\alpha_0} e_2) = 11 \]
\[ u_1(e_0 \xrightarrow{\alpha_1} e_3) = 70 \]
\[ u_1(e_0 \xrightarrow{\alpha_1} e_4) = 9 \]
\[ u_1(e_0 \xrightarrow{\alpha_1} e_5) = 10 \]

What are the expected utilities of the agents for this utility function?
A special case of assigning utilities to histories is to assign 0 (false) or 1 (true) to a run.
If a run is assigned 1, then the agent succeeds on that run, otherwise it fails.
Call these predicate task specifications.
Denote predicate task specification by $\Psi$

$$\Psi : \mathcal{R} \rightarrow \{0, 1\}$$
A task environment is a pair $\langle Env, \Psi \rangle$ where $Env$ is an environment, and $\Psi : \mathcal{R} \to \{0, 1\}$ is a predicate over runs. Let $\mathcal{TE}$ be the set of all task environments.

Write $\mathcal{R}_\Psi(Ag, Env)$ to denote the set of all runs of the agent $Ag$ in the environment $Env$ that satisfy $\mathcal{R}_\Psi(Ag, Env) = \{ r | r \in \mathcal{R}(Ag, Env) \text{ and } \Psi(r) = 1 \}$

We then say that an agent $Ag$ succeeds in task environment $\langle Env, Ag \rangle$ if $\mathcal{R}_\Psi(Ag, Env) = \mathcal{R}(Ag, Env)$.
Let $P(r \mid Ag, Env)$ denote probability that run $r$ occurs if agent $Ag$ is placed in environment $Env$.

Then the probability $P(\Psi \mid Ag, Env)$ that $\Psi$ is satisfied by $Ag$ in $Env$ would then simply be:

$$P(\Psi \mid Ag, Env) = \sum_{r \in \mathcal{R}_\Psi(Ag, Env)} P(r \mid Ag, Env)$$
Agent Synthesis

Agent synthesis is automatic programming: goal is to have a program that will take a task environment, and from this task environment automatically generate an agent that succeeds in this environment:

\[ \text{syn} : \mathcal{T} \mathcal{E} \rightarrow (\mathcal{A} \mathcal{G} \cup \{\bot\}) \]

(Think of \( \bot \) as being like null in JAVA).

Synthesis algorithm:

- is **sound** if, whenever it returns an agent, then this agent succeeds in the task environment that is passed as input; and
- is **complete** if it is guaranteed to return an agent whenever there exists an agent that will succeed in the task environment given as input.
Synthesis algorithm syn is sound if it satisfies the following condition:

\[ \text{syn}(\langle Env, \Psi \rangle) = Ag \implies R(Ag, Env) = R_\Psi(Ag, Env) \]

and complete if:

\[ \exists Ag \in AG \text{s.t.} R(Ag, Env) = R_\Psi(Ag, Env) \implies \text{syn}(\langle Env, \Psi \rangle) \neq \bot \]
This view of agents leads to a kind of post-declarative programming:

In **procedural programming**, we say exactly what a system should do

In **declarative programming**, we state something that we want to achieve, give the system general info about the relationships between objects, and let a built-in control mechanism (e.g., goal-directed theorem proving) figure out what to do

With agents, we give a very abstract specification of the system, and let the **control mechanism** figure out what to do, knowing that it will act in accordance with some built-in theory of agency.