CS 5260: Intro to Artificial Intelligence

Week 4

Will Hedgecock, Ph.D.

Programming Project, Part I

- Purpose
  - Allow you to explore various AI search strategies covered in the class
  - Give you hands-on experience in developing heuristics and utility functions
  - Show how the theoretical concepts of this course can be implemented programmatically
  - Get you to think critically about how human knowledge and intuition can be used to implement AI agents that behave rationally

- Antigoals
  - Push the limits of your coding and programming language knowledge
  - Have you create a “correct” solution with a predefined/expected outcome
  - Make you learn in-the-weeds details about world health measurements and models
  - Pit you against your classmates to come up with the “best” solution

- World Trade Game

Programming Project, Part I

- Mapping of project terminology to already-covered concepts

  - **State Space:** Set of all possible resource levels for all countries
  - **State:** Set of *current* resource levels for all countries
  - **Actions:** Concrete instantiations of the TRANSFORM and TRANSFER operations
  - **Heuristic:** The “State Quality” of a country
    - How you choose to weight a state in terms of its utility/goodness
  - **Reward Value:** How much the state of your country has improved over time
    - **Raw:** State Quality of a state at some time, *t*, minus its value at *t=0*
    - **Discounted:** Raw Reward Value of a state penalized for the amount of time required to get into that state
  - **Evaluation Value:** The “Expected Utility” of a future/unreached state

Programming Project, Part I

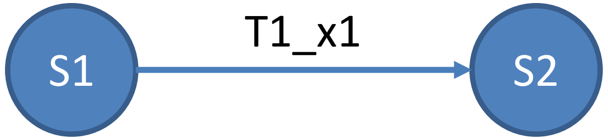
S1 = Current State:
- 5 Water
- 5 Lumber
- 2 Population

T1 = Transform Operator:
- Input:
  - 2 Water
  - 2 Lumber
  - 2 Population
- Output:
  - 1 House
  - 2 Population

Programming Project, Part I

S1 = Current State:
5 Water
5 Lumber
2 Population

T1 = Transform Operator:
Input:
2 Water
2 Lumber
2 Population
Output:
1 House
2 Population



S2 = Child State:
3 Water
3 Lumber
2 Population
1 House

Programming Project, Part I

S1 = Current State:

5 Water
5 Lumber
2 Population

T1 = Transform Operator:

Input:

2 Water
2 Lumber
1 Population

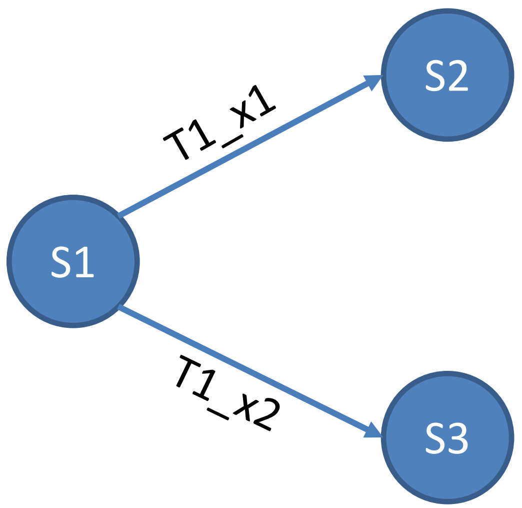
Output:

1 House
1 Population

Programming Project, Part I

S1 = Current State:
- 5 Water
- 5 Lumber
- 2 Population

T1 = Transform Operator:
- Input:
  - 2 Water
  - 2 Lumber
  - 1 Population
- Output:
  - 1 House
  - 1 Population



S2 = Child State:
- 3 Water
- 3 Lumber
- 2 Population
- 1 House

S3 = Child State:
- 1 Water
- 1 Lumber
- 2 Population
- 2 Houses

Programming Project, Part I

- Mapping of project terminology to already-covered concepts

  - **State Space:** Set of all possible resource levels for all countries
  - **State:** Set of *current* resource levels for all countries
  - **Actions:** Concrete instantiations of the TRANSFORM and TRANSFER operations
  - **Heuristic:** The “State Quality” of a country
    - How you choose to weight a state in terms of its utility/goodness
  - **Reward Value:** How much the state of your country has improved over time
    - **Raw:** State Quality of a state at some time, t, minus its value at $t=0$
    - **Discounted:** Raw Reward Value of a state penalized for the amount of time required to get into that state
  - **Evaluation Value:** The “Expected Utility” of a future/unreached state

Resources and Resource Types

- Basic/raw vs. manufactured/created
  - Start with **only** raw resources
- Explicit vs. implicit worth
- Resource waste
  - Balances world state and helps remove potential for search loops
- Cannot transfer land, people, potential fossil energy sources (in-ground oil), or potential or usable renewable energy sources (solar, wind, hydroelectric)
  - Anything else can be transferred
  - *You can play around with transferring people if you wish, but not required*
- Exhaustive list of potential resource types for Part 1 in instructions

IGNORE ALL INSTRUCTIONS ABOUT "RENEWABLE" OR "RECYCLABLE" RESOURCES FOR PART 1

Programming Templates, Transformations

- Templates are used to define available **ACTIONS**
- Each **ACTION** will have a corresponding set of **PREREQUISITES**
- Sample **TRANSFORMATION** Template:

Alloy Creation Template

```
(TRANSFORM C
          (INPUTS (Population 1
                  (MetallicElements 2
                  (PotentialEnergyUsable 3)
                  (Water 3))
          (OUTPUTS (Population 1)
                   (MetallicAlloys 1)
                   (MetallicAlloysWaste 1)
                   (Water 2)))
```

Programming Templates, Transformations

•

Housing Creation Template

```
(TRANSFORM C
          (INPUTS (AvailableLand 1)
                  (Population 5)
                  (Water 5)
                  (MetallicElements 1)
                  (Timber 5)
                  (MetallicAlloys 3)
                  (PotentialEnergyUsable 5))
          (OUTPUTS (Housing 1)
                   (HousingWaste 1)
                   (Population 5)
                   (Water 4)))
```

Programming Templates, Transfers

- Single TRANSFER template:

(TRANSFER C_i C_k ((R_{j1} X_{j1}) ... (R_{jm} X_{jm})))

- Example Usage:

(TRANSFER C_1 C_2 ((Housing 3) (Water 2)))

- List of TRANSFERS should be treated as ordered singleton transfers:

(TRANSFER C_i C_k ((R_{j1} X_{j1})))
...
(TRANSFER C_i C_k ((R_{jm} X_{jm})))

- **Should primarily use singleton transfers in Part 1, but you are allowed to use non-singletons specifically to trade *waste* along with another resource if so desired**

Programming Templates, Transfers

- Except for waste, concurrent transfers should be modeled as multiple separate transfer steps for Part 1

- You **must** include your “self” country in every transfer operation
  - You can’t force other countries to transfer their goods to each other against their will
  - In Part 2, if you choose to investigate multi-agent game play, each agent will represent its own “self” country
  - In Part 2, macros can be created to investigate complicated transfers or to bridge the gap between “bad states”

- If you feel like you want an extra challenge and have already completed most of your Part 1 project, you *might* be allowed to investigate using macros in this part, but please first send me an email

State Quality Function

- Is a function
  - Example: $Q(s) = \sum_{r \in R} \frac{w_r * c_r * A_r}{A_{pop}}$
    - where $R =$ set of available resources
      $A_r =$ amount of resource *r*
      $c_r =$ proportionality constant for *r* (e.g., 2 units food per person)
      $w_r =$ weight/importance of resource *r*

- Each student develops their own
- Must be substantially dependent on a country's resources
- Does not have to be solely about the "happiness" of a population, could be about carbon footprint
  - Refer to Project Instructions for a couple of links with more ideas
  - Don't spend too much time on this, important thing is to think about how to quantify a desired outcome and how that choice of outcome will affect your search strategy

Resource Utility Weighting

- Resource weighting factor in previous slide affects determination of overall state quality
  - All resources are not created equal
  - You can choose your own resource weighting scheme, but it must make sense and be dictated by your utility function
  - All manufactured resources **must** create a corresponding negatively valued waste resource
    - But the value of the waste does **not** have to have the same magnitude as the value of the manufactured resource

- For Part 1, all resource weights are static and shared by all countries
  - For Part 2, different countries can have different weights or even time-varying weights

- Must be able to parse from CSV file

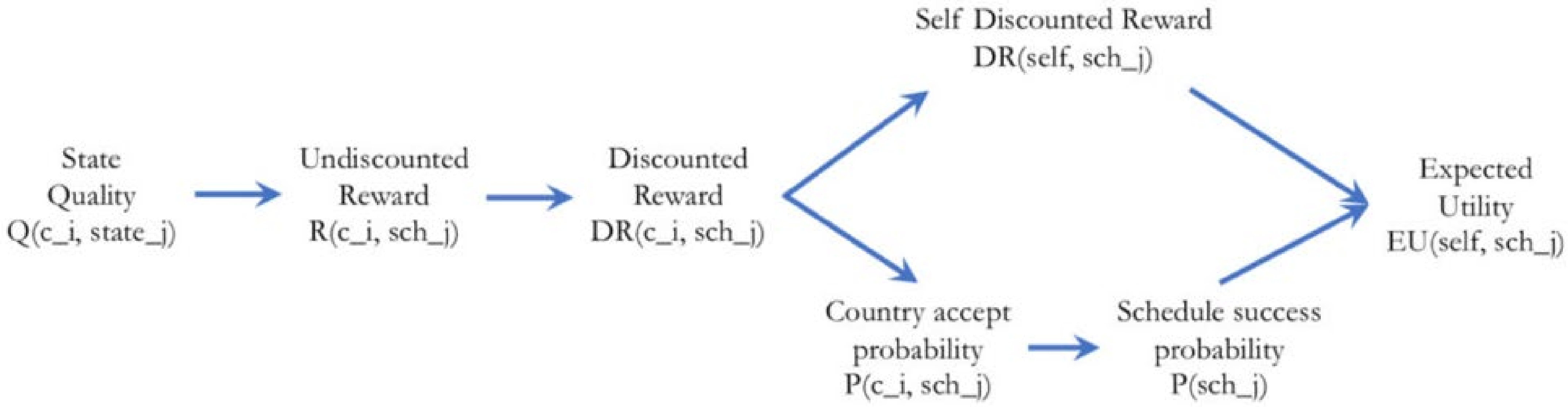
Representing Resource Utility

	A	B
1	**Resource**	**Weight**
2	Population	0
3	MetallicElements	0
4	Timber	0
5	MetallicAlloys	0.2
6	Electronics	0.5
7	Housing	0.8
8	MetallicAlloyWaste	-0.5
9	ElectronicWaste	-0.8
10	HousingWaste	-0.4

- Represented in a CSV file
- Must be able to be parsed dynamically by your program and changed on-the-fly
- Waste must be included for every manufactured resource
- Waste weight does **not** have to be the negative of the associated manufactured resource
- Weights must make sense given your chosen State Quality function

Expected Utility

- Follows directly from your State Quality function and set of resource weights
- Used to determine which actions lead to higher quality states
  - I.e., used to order the placement of items onto the frontier
- Includes probability that a given schedule *would actually happen in the real world*



Expected Utility

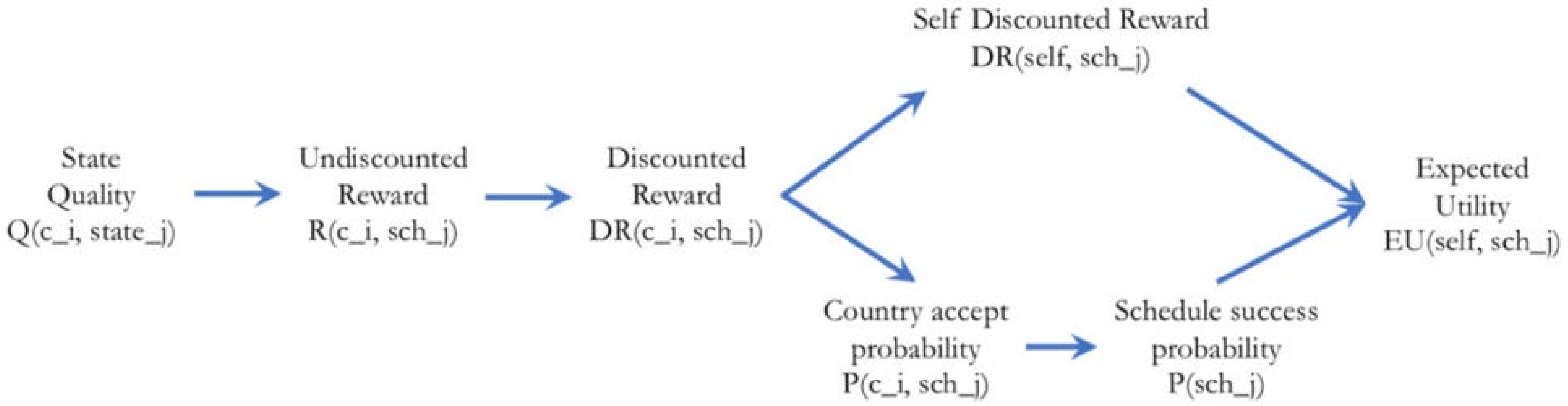
- Undiscounted Reward: $R(c_i, s_j) = Q_{end}(c_i, s_j) - Q_{start}(c_i, s_j)$
- Discounted Reward: $\mathrm{DR}(c_i, s_j) = \gamma^N * R(c_i, s_j)$ where: N = num time steps
- Country Accept Probability: $P(c_i, s_j) = {}^{1}\!/_{1+e^{-k(DR(c_i, s_j) - x_0)}}$ where: x_0 is the sigmoid midpoint

  k affects curve steepness
- Schedule Success Probability: $P(s_j) = \prod_{c \in C} P(c, s_j)$

Expected Utility

- Expected Utility: $EU(self, s_j) = P(s_j) * DR(self, s_j) + \left(\left(1 - P(s_j)\right) * \mathrm{C}\right)$

  where *C* is negative constant representing the cost of creating a failed plan
- Your choice of *C* should be justified



Representing World State (Input)

- Represented in a CSV file
- Must be able to be parsed dynamically by your program and changed on-the-fly
- Must include a column for **all** resources being investigated by your agent
- Must **only** include resource values for **raw** resources
  - All manufactured resources must have an initial value of 0

	A	B	C	D	E	F	G
1	**Country**	**Population**	**MetallicElements**	**Timber**	**MetallicAlloys**	**Electronics**	**Housing**
2	Atlantis	100	700	2000	0	0	0
3	Brobdingnag	50	300	1200	0	0	0
4	Carpania	25	100	300	0	0	0
5	Dinotopia	30	200	200	0	0	0
6	Erewhon	70	500	1700	0	0	0

Representing Schedules (Output)

- Required format of output text file:

```
[
   (TRANSFORM self …) EU: val_for_first_state
   (TRANSFER self C2) ((Housing 3))) EU: val_for_next_state
   …
   (TRANSFORM self …) EU: val_for_final_state
]
```

- Number of operations in each schedule is dictated by a *depth-bound* parameter that will be passed to the top-level function of your agent:

```
def country_scheduler(your_country_name, resources_filename,
                      initial_state_filename, output_filename,
                      num_output_schedules, depth_bound, max_frontier_size)
```

Suggestions for Getting Started

- Decide on a State Quality function for your agent
- Create your State representation in code
- Come up with your initial listing of all available resources, weights, and the initial world state
- Write code to calculate the State Quality, rewards, and success probabilities for a given country and schedule
- Write code to calculate the Expected Utility of a given schedule
- Write code to parse TRANSFORM operators into Actions in your code
- Write code to expand a current State into a set of possible next States given a set of possible Actions (along with their prerequisites)
- Fine-tune your code
  - Pre-shrink any variable domains using the GAC algorithm
  - Implement different search strategies
  - Add more resources and/or transformation operators
  - Play with the Expected Utility gamma, k, and x0 values
  - Try sorting any priority queues on different values

Suggestions for Getting Started

- Don’t create too many countries or resources initially
    - Increases branching factor
- Carefully read through the Programming Project section entitled “Search Strategy” for more explicit suggestions for getting started on Part 1
- Ignore the “Deliverables” section for now
- Take a look at my sample Python code base on GitHub for ideas of how to get started implementing different search strategies in a generalized fashion