Appendix A.
Model specifications

In this appendix we specify our models in formal terms. Section 1 provides a formal description of the decision problem that an agent faces. Section 2 specifies the procedure we use to find the optimal policy. Our model is instantiated using ModelBuilder (version 1), using JAVA (version 15.0.2) and R (version 4.0.4). The ModelBuilder program has not been released yet, but in the main text we refer to the GitHub page for this program. Both models are computed using an AMD THREATRIPPER 3990WX CPU and 256 GB of 3200 MHz RAM, with operating system Windows 10 (build 19042). In section 3 we specify how we quantify delay times using these optimal policies. Finally, section 4 provides a reference table containing all important mathematical notations.

A note for preregistration. At the time of writing this preregistration, we already computed the optimal policies (section 3). However, we have not interpreted modeling results yet, nor have we made any parameter adjustment based on the results. The reason that we already compute optimal policies prior to preregistration is that ModelBuidler is currently in early alpha. As such, we wanted to ensure that all necessary computations need for our models are possible in ModelBuilder, and that the results did not show signs of error.

1. A formal description of the decision problem

Before we start formalizing the decision problem, a brief overview. Figures A1 and A2 provide schematic overviews of the decision problem, for the postponing and waiting model, respectively. An agent's state has two components: its reserves, representing the agent's condition, whether it be physical, social, or material; and its age. This state changes as an agent interacts with its environment via resource encounters and extrinsic events.

Each time step consists of two phases. In the first phase, the action phase, the agent decides which action it takes (section 1.3). If it currently is in an resource encounter, it has to decide on a delay time. If it is not in an encounter, it has to search for a resource. If it searches, it encounters a resource with certainty, and starts a new encounter.

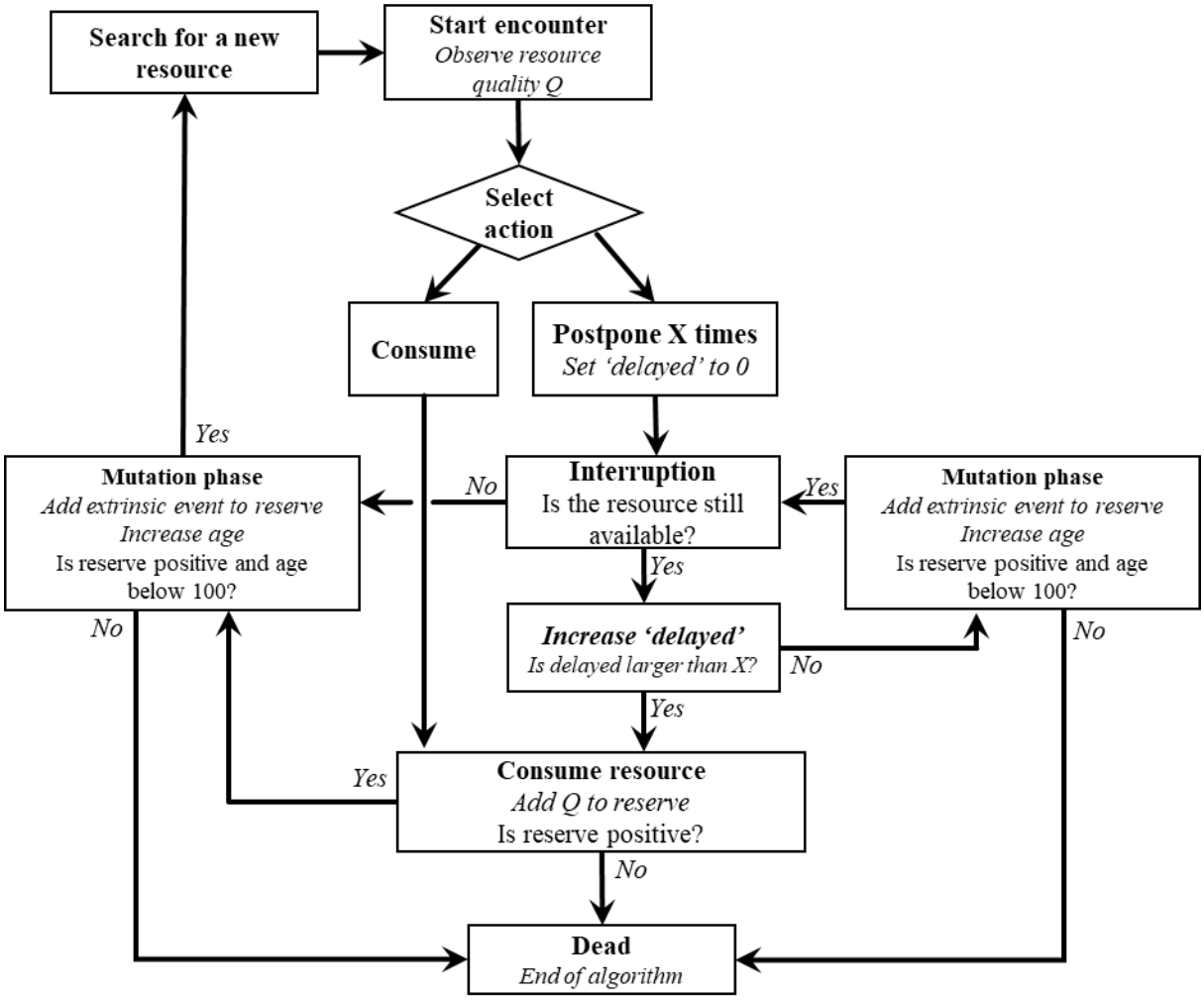

Figure A1
A schematic overview of the decision problem faced by an agent in the postponing model.

In the second phase, the mutation phase, the environment imposes changes which an agent cannot control, mitigate, or prevent. If an agent dies during the action phase, there is no mutation phase. There are three types of mutations. First, an encounter might be interrupted. Second, the agent encounters an extrinsic event, which changes its reserves. Extrinsic events happen regardless of whether an agent is searching for a resource, busy consuming one, or is waiting or postponing. The quality of an extrinsic event depends on the environment (section 1.6), and can increase or decrease reserves, or have no effect. Second, the resource quality changes each time step between encountering a resource and consuming (section 1.2.1) it. After the mutation phase the time steps ends, and the action phase of the next time step immediately starts.

1.1. Reserves and age

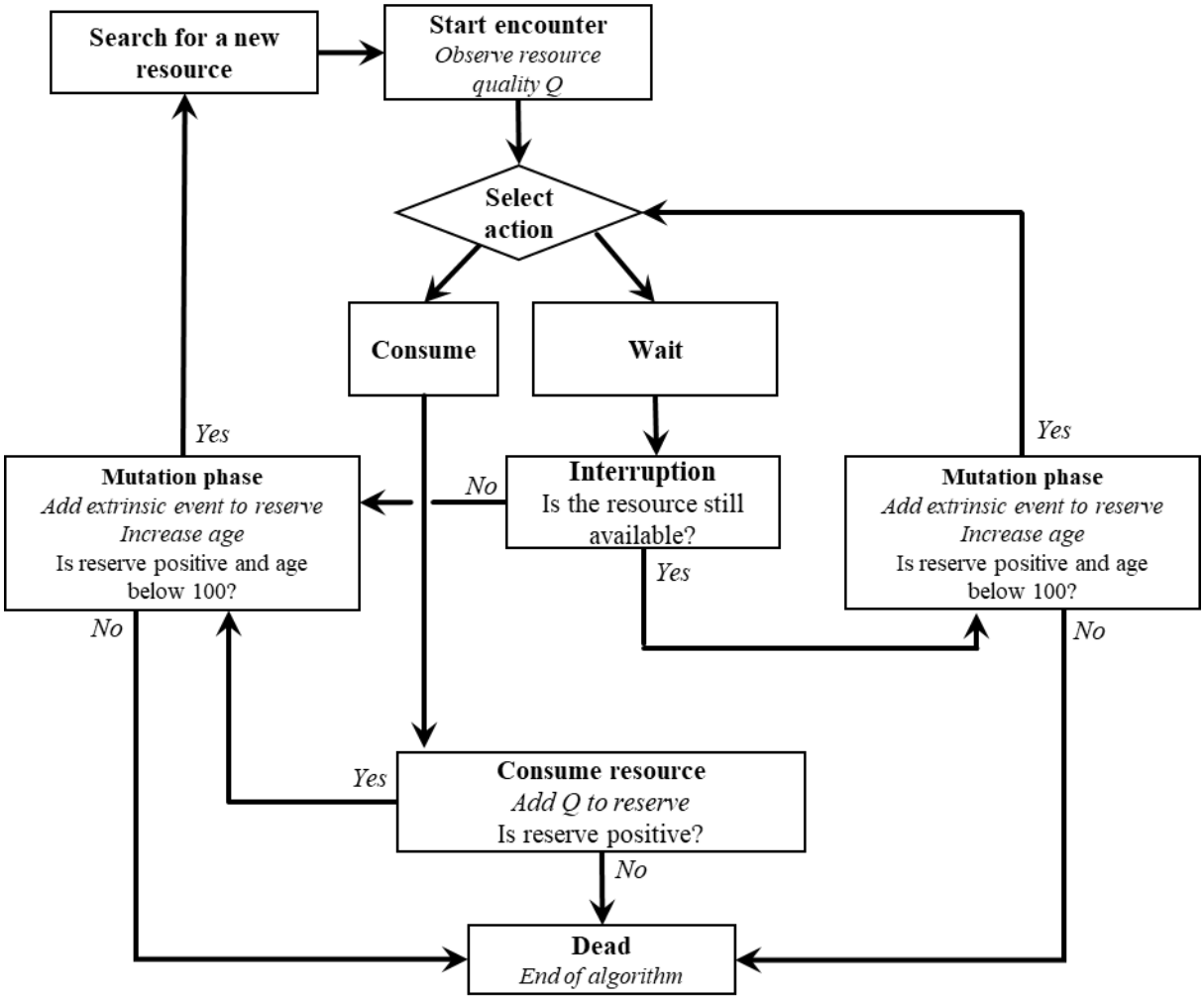

Figure A2
A schematic overview of the decision problem faced by an agent in the waiting model.

The resource quality and the value of extrinsic events, and an agents reserves have discrete values that are multiples of 0.1. As such, an agent's reserves also has a multiple of 0.1. Let B denote the set of all possible reserve states:

$$B = \{ x \mid x = 0.1n, n \in \mathbb{N}_0, 0 \leq x \leq 100\} \tag{1.1}$$

And let T be the set of all times steps (which is the same as an agent's age):

$$T = \{ x \mid x \in \mathbb{N}_0, x \leq 100\} \tag{1.2}$$

Where $\mathbb{N}_0$ is the set of all natural numbers including 0 (i.e., non-negative integers). Let b_t denote an agent's reserves at the start of time step t (we use B and b_t, short for budget, to avoid confusion; R and r_t might be misunderstood as the resource quality).

1.2. Resources

The resource quality ranges between -15 and +15 and is always a multiple of 0.1. Let Q denote the set of all possible resource qualities:

$$Q = \{ x \mid x = 0.1n, n \in \mathbb{Z}, -15 \leq x \leq 15\} \tag{1.3}$$

Where z refers to the set of all integers. Each resource encountered has to be consumed (affecting an agent's reserves) unless it is interrupted. Each time step after encounter a resource and before its consumption, the quality of a resource becomes more extreme. However, the initial quality of a resource – when it is first encountered – always is an integer value. Let $Q_{initial}$ denote the set of all possible resource qualities that can be encountered:

$$Q_{initial} = \{ x \mid x \in \mathbb{Z}, -15 \leq x \leq 15\} \tag{1.4}$$

Resources are independently and identically distributed and drawn from a truncated discrete normal distribution with mean $\mu_{resource}$ and standard deviation $\sigma_{resource}$. Let $\eta(x \mid \mu, \sigma)$ be

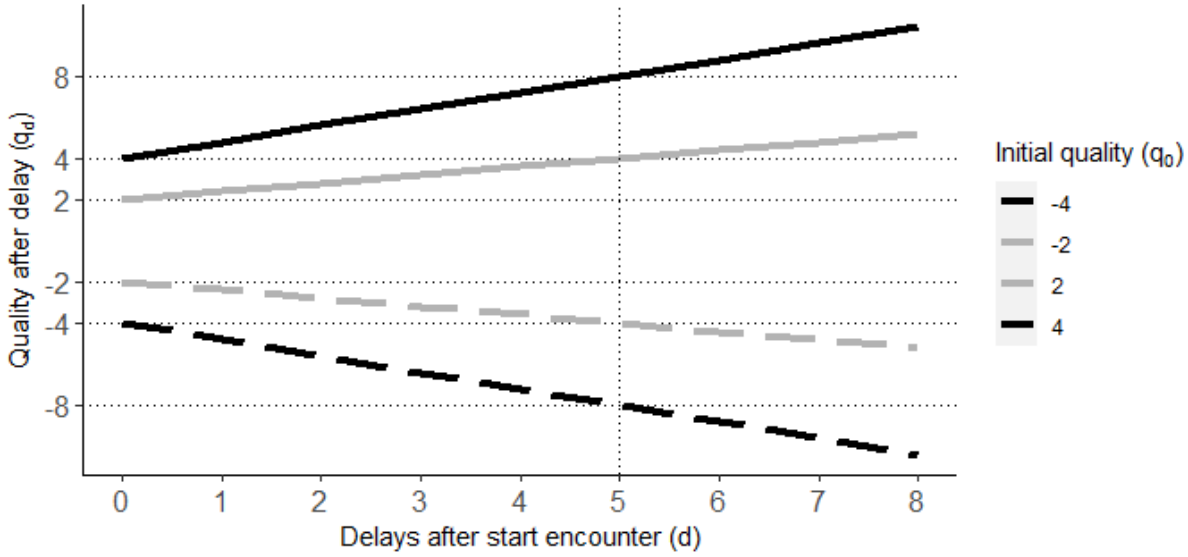

Figure A3
How the quality of a resource changes as a function of the delays during an encounter. The resource quality increases linearly so that it doubles after 5 time steps.

the probability of a value x, given that it is sampled from a normal distribution with mean μ and standard deviation σ. The probability of a specific resource quality $q_{initial} \in Q_{initial}$ is:

$$Pr(q_{initial}, \mu_{resource}, \sigma_{resource}) = \frac{\eta(q|\, \mu_{resource}, \sigma_{resource})}{\sum_{r \in Q_{initial}} \eta(r|\, \mu_{resource}, \sigma_{resource})} \quad (1.5)$$

The denominator functions as a renormalizing constant, ensuring that all probabilities sum to 1.

1.2.2. Resource qualities change over time. Each time step that a resource is not consumed, its quality becomes more extreme (Figure A3). Specifically, it's magnitude increases to that it doubles after 5 time steps. Let q_t denote the quality of a resource after it has not been consumed for d time steps:

$$q(q_0, d) = q_{inital} + \frac{q_{inital}}{5} d \quad (1.6)$$

An agent can delay (either by postponing or by waiting) a maximum of 8 times steps. After the 8th time step, the resource is automatically consumed.

1.2.3. Interruption rate. During each mutation phase there is a fixed probability that a resource encounter is interrupted. This interruption occurs with probability ρ. After an interruption the resource disappears and the encounter ends without any further changes to an agent's reserves.

1.3. Actions

Let $A(t, b_t)$ denote the set of actions that an agent can take at time t given its reserves b_t. When an agent is not in a resource encounter, it can perform only one action, to search for a new resource. Searching for a resource takes up the entire action phase. As such, an agent can encounter a maximum of 50 resources during its lifetime, should it never delay. The actions that an agent can take during an encounter differs between the waiting and postponing model.

In the postponing model, an agent must choose how many time steps p it postpones. While postponing it cannot take any other actions. After p time steps have passed the resource is consumed. An agent cannot wait more time steps than that is has alive: $p \leq 100 - t$.

In the waiting model, an agent must repeatedly choose between waiting one more time step and immediately consuming the resource. If it wait, and the resource is not interrupted, the agent faces the same decision in the next time step. An agent can wait up to 8 time steps per resource encounter, and must consume thereafter.

If an agent's reserves equal 0 after consuming the resource, it immediately dies. In this case there is no mutation phase that might increase its reserves above 0.

1.4. Extrinsic events

After the action phase, if an agent is alive, it experiences an extrinsic event during the mutation phase. Extrinsic events are outside the control of the agent. The possible value of an extrinsic event depends on the mean and range of extrinsic events in the environment. The mean of the extrinsic events is either -1, 0, or 1. The range is either 0, 2, or 4. If we let E be set of all extrinsic events that are possible, $\mu_{extrinsic}$ be the mean of extrinsic events, $\sigma_{extrinsic}$ be the range of extrinsic events, then:

$$\mathrm{E}(\mu_{extrinsic}, \sigma_{extrinsic}) = \begin{cases} \{-1\} & if\ \mu_{extrinsic} = -1 \wedge \sigma_{extrinsic} = 0 \\ \{0\} & if\ \mu_{extrinsic} = 0 \wedge \sigma_{extrinsic} = 0 \\ \{1\} & if\ \mu_{extrinsic} = 1 \wedge \sigma_{extrinsic} = 0 \\ \{-2,0\} & if\ \mu_{extrinsic} = -1 \wedge \sigma_{extrinsic} = 2 \\ \{-1,1\} & if\ \mu_{extrinsic} = 0 \wedge \sigma_{extrinsic} = 2 \\ \{0,2\} & if\ \mu_{extrinsic} = 1 \wedge \sigma_{extrinsic} = 2 \\ \{-3,1\} & if\ \mu_{extrinsic} = -1 \wedge \sigma_{extrinsic} = 4 \\ \{-2,2\} & if\ \mu_{extrinsic} = 0 \wedge \sigma_{extrinsic} = 4 \\ \{-1,3\} & if\ \mu_{extrinsic} = 1 \wedge \sigma_{extrinsic} = 4 \end{cases}$$

In each environment, the probability of specific extrinsic event value is drawn from an uniform distribution:

$$Pr(e, \mu_{extrinsic}, \sigma_{extrinsic}) = uniform(E) = \frac{1}{|E|} \tag{1.7}$$

Where $|E|$ is the number of elements in E.

1.5. Fitness

When an agent dies, it receives a fitness payoff equal to its reserves at the point of death:

$$\omega(b_t) = b_t \tag{1.8}$$

Explicitly, it receives zero fitness if it dies from starvation.

1.6. The environment: harshness and unpredictability

We vary the levels of environmental harshness and unpredictability between agents. These parameters do not change within an environment over cycles, and an agent lives its entire life in a single environment.

We explore two notions of harshness: as a (low or negative) mean resource quality $\mu_{resource}$, and as a (low or negative) mean extrinsic event value $\mu_{extrinsic}$. We explore three notions of unpredictability. First, we vary the resource unpredictability by varying the resource value standard deviation $\sigma_{resource}$. Second, we vary the extrinsic unpredictability by varying the

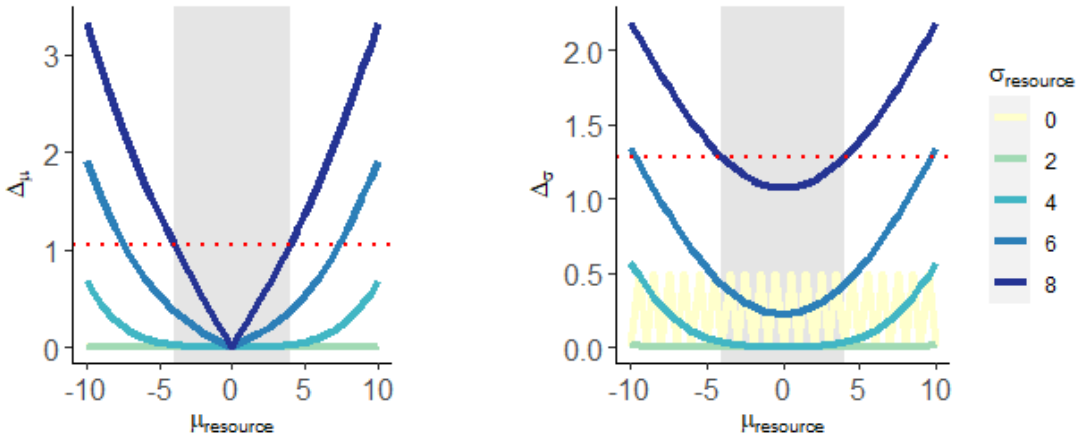

Figure A4
The absolute differences Δ_μ and Δ_σ as a function of $\mu_{resource}$ and $\sigma_{resource}$. We restrict $\mu_{resource}$ to be between -5 and 5 (grey square), and $\sigma_{resource}$ between 0 (yellow) and 8 (dark blue). Under these ranges, the maximum Δ_μ is approximately 1.06 and the maximum Δ_σ is approximately 1.28. The yellow oscillation between 0 and 0.5 occurs when there is a standard deviation of 0. In that case, a resource should only have one possible value. However, if this value is not possible because it is not an integer value (e.g., a mean of 0.5), the two nearest values are drawn proportional to the distance to the point (e.g., 0 with probability 0.5 and 1 with probability 0.5). This results in a non-zero standard deviation.

range in extrinsic event values $\sigma_{extrinsic}$. Third, we vary the interruption rate by changing the probability of an interruption during each time step of a resource encounter ρ. Let env denote a vector consisting of five state variables that together represent an environment:

$$env = \begin{bmatrix} \mu_{resource} \\ \mu_{extrinsic} \\ \sigma_{resource} \\ \sigma_{extrinsic} \\ \rho \end{bmatrix} \tag{1.9}$$

We systematically explore different combinations of these five state variables. Specifically, we explore all environments in the set ENV:

$$ENV = \left\{ \begin{matrix} \mu_{resource} \\ \mu_{extrinsic} \\ 2n * \sigma_{resource} \\ \sigma_{extrinsic} \\ \rho \end{matrix} \middle| \begin{matrix} -4 \leq \mu_{resource} \leq 4, n \in \mathbb{N}_0 \\ \mu_{extrinsic} \in \{-1,0,1\} \\ 0 \leq \sigma_{resource} \leq 8, n \in \mathbb{N}_0 \\ \sigma_{extrinsic} \in \{0,2,4\} \\ \rho \in \{0, 0.2, 0.5\} \end{matrix} \right\} \tag{1.10}$$

As we discussed above, resource qualities are always between -15 and 15. Due to the range restriction, the mean and standard deviation of the normal distribution are not independent. In the most extreme cases, if we set the mean to be 15, the maximum possible standard deviation is 0.

This restriction of range also influences the means and standard deviations in less extreme cases. That is, if we draw resources from the distribution specified in equation 1.5, the mean and standard deviation of all observed resources is not equal to $\mu_{resource}$ and $\sigma_{resource}$, respectively. Let $\mu_{observed}$ and $\sigma_{observed}$ denote mean and standard deviation of all resources in the environment:

$$\mu_{observed} = \sum_{q \in Q} q * Pr\ (q, \mu_{resource}, \sigma_{resource}) \neq \mu_{resource} \tag{1.11}$$

$$\sigma_{observed} = \sqrt{\sum_{q \in Q} ((\mu_{observed} - q)^2 * Pr\ (q, \mu_{resource}, \sigma_{resource})\ )} \neq \sigma_{resource} \tag{1.12}$$

Let Δ_μ be the absolute difference between $\mu_{resource}$ and $\mu_{observed}$, and let Δ_σ be the absolute difference between $\sigma_{resource}$ and $\sigma_{observed}$:

$$\Delta_\mu = |\mu_{observed} - \mu_{resource}| \tag{1.13}$$

$$\Delta_\sigma = |\sigma_{observed} - \sigma_{resource}| \tag{1.14}$$

Figure A4 depicts Δ_μ and Δ_σ as a function of $\mu_{resource}$ and $\sigma_{resource}$. The absolute differences increase rapidly with both extreme means and higher standard deviations. By themselves these differences are not problematic, as we are interested in qualitative patterns, not precise estimates (e.g., we ask ‘Does a higher resource standard deviation results in more delays’, rather than ‘How many more time steps should an agent delay if we increase the

resource standard deviation'). However, they can become problematic if they become so extreme that different parameter combinations result in the same outcome. For instance, settings the resource standard deviation of 100 and 1000 results in nearly similar distributions. This would falsely suggest that there is no effect of resource unpredictability. To minimize range effects, we restrict resource means to the range $[-4, 4]$ with steps of 0.5, and the standard deviations of resource quality to the range $[0, 8]$ with steps of 2. Under these restrictions the maximum Δ_μ is approximately 1.06 and the maximum Δ_σ is approximately 1.27.

2. Finding the optimal policy

A policy $\pi(s)$ specifies what an agent should do in each state. The optimal policy, denoted by $\pi^*(s)$, is the policy that maximizes the expected fitness. Here we describe how we find the optimal policy, given an agent's environment. Note that we keep the environment constant between agents. For easy of reading, we omit a specific reference to the environment env – although it should be explicitly stated in each equation. However, to be explicit, we iterate this procedure separately for each environment $env \in ENV$.

Two observations. First, we assume that an agent knows all the meta-parameters of its environment; it knows the distribution of resources, and extrinsic events, and the interruption rate. Consequentially, there is no learning; observing a resource, extrinsic event, or interruption does not change its beliefs in what will happen the next time step. Second, environmental conditions vary between agents, but are stable within an agent. Resources and extrinsic events are not depleted; observing a positive resource or extrinsic event does not reduce the expected resource quality or extrinsic event in future time steps. Rather, resources and extrinsic events are independent from one time to another.

A consequence of these observations is that the only five changes carry over from time step to time step: (i) an agent's age; (ii) agent's reserve; (iii) the quality of the resource at the start of the current encounter, if applicable; (iv) the number of time steps d is has already spend delaying during the current encounter, if applicable, and (v) the number of time steps an agent still has to postpone p, if applicable for the postponing model.

Because we include these five changes in the state of an agent (section 2.1), the optimal policy has the Markov property: it is independent of previous time step, given the current state of an agent. Or stated differently, it does not matter to the optimal policy how an agent came to be in a state; the past is irrelevant. We compute, for all possible environments, the optimal policy for all possible reserves $b_0 \in B$ that agent might have at the start of the first time step.

In section 2.1 we first describe the decision the agent faces in terms of a Markov Decision Process (MDP). Specifically, we specify all possible actions, states, state transitions, and

outcomes within a time step. In section 2.2 we describe how ModelBuilder finds the optimal policy using stochastic dynamic programming and backwards induction.

2.1. Specifying the MDP

2.1.1. Defining the state space. All possible states an agent can be in during an encounter can be described as a combination six variables: its age (always equal to t), its reserves b, the quality of a resource when it was first encountered q_0, and the time spend delaying during the current encounter d. The sixth one, that wasn't specified above, is whether an agent is in an action phase, mutation phase, or fitness phase. A fitness phase is a nice way of stating that an agent is dead and can compute its fitness (section 1.5). We represent this type of state with ph. Let S be the set of all possible states the agent can be in:

$$S = \left\{ b, q_0, d, p, t, ph \;\middle|\; \begin{matrix} b \in B \\ q \in Q_{initial} \\ d \in D \\ p \in D \\ t \in T \\ ph \in PHASE \end{matrix} \right\} \tag{2.1}$$

Where D is the set of all delay times that an agent can delay during the current encounter (usually 8, unless the agent's age approaches 100; section 1.3), and $PHASE$ is the set of all possible phases, consisting of ACT (action phase), MUT (mutation phase), and FIT (fitness phase).

To increase readability, we group these variable on three lines. The first line provides information about the reserves and currently encountered resource, the second line contains all 'time' related information. The third line indicates in which phase the state is:

$$S = \left\{ \begin{matrix} b, q_0, \\ d, p, t \\ ph \end{matrix} \;\middle|\; \begin{matrix} b \in B \\ q \in Q_{initial} \\ t \in T \\ d \in D \\ p \in D \\ ph \in PHASE \end{matrix} \right\} \tag{2.2}$$

For convenience, we also define three subsets of S:

$$S^{action} = \left\{ \begin{matrix} b, q_0, \\ d, p, t \\ ph \end{matrix} \middle| \begin{matrix} b \in B \\ q \in Q_{initial} \\ t \in T \\ d \in D \\ p \in D \\ ACT \end{matrix} \right\} \tag{2.3}$$

$$S^{mutation} = \left\{ \begin{matrix} b, q_0, \\ d, p, t \\ ph \end{matrix} \middle| \begin{matrix} b \in B \\ q \in Q_{initial} \\ t \in T \\ d \in D \\ p \in D \\ MUT \end{matrix} \right\} \tag{2.4}$$

$$S^{fitness} = \left\{ \begin{matrix} b, q_0, \\ d, p, t \\ ph \end{matrix} \middle| \begin{matrix} b \in B \\ q \in Q_{initial} \\ t \in T \\ d \in D \\ p \in D \\ FIT \end{matrix} \right\} \tag{2.5}$$

Next, we define seven functions to retrieve properties of a state. First, let $b(s)$ be a function which results in an agent's reserves in state s. Second, let $d(s)$ result in the number of time steps already delayed in state s. Third, let $q_0(s)$ result in the quality of the resource at the start of the encounter. Fourth, let $t(s)$ result in the agent's age in state s. Fifth, let $post(s)$ denote the number of time step an agent still has to postpone. Sixth, let $phase(s)$ result whether an agent is in the action or mutation phase of a time step. And seventh, let $s(b, q_0, d, t, p, ph)$ refer to the state where an agent's reserves is b, it has encountered a resource with initial quality q_0, has delayed d time steps, still has to postpone p time steps, has age *t*, and is currently in phase p. If an agent is not currently in a resource encounter, $q_0(s)$, $d(s)$, and $post(s)$ are result in $\emptyset$. In the waiting model there is no postponing, and p is always $\emptyset$.

At the start of its life, an agent has reserves b_0 and has not yet encountered a resource. Let S^{start} be the set of all possible starting states:

$$S^{start} = \left\{ s\begin{pmatrix} b_0, \emptyset, \\ \emptyset, \emptyset, 0 \\ ACT \end{pmatrix} \middle| b_0 \in B \right\} \tag{2.6}$$

When an agent's reserves is 0 or its age t is 100, it dies – i.e., goes to a fitness state. Note that states in S^{fitnes} states are absorbing; once entered they cannot be left. Let $P(s'|s)$ be the transition probability function that specifies the probability of going from state s to state s':

$$P(s'|s = s_{fitness},*) = \begin{cases} 1 & s' \in S^{fitness} \\ 0 & otherwise \end{cases} \tag{2.7}$$

As such, once an agent reaches a fitness state, the game is over. For ease of reading we use $s_{fitness}$ as a wildcard token that can refer to any state in $S^{fitness}$:

$$s_{fitness} = s\begin{pmatrix} *,*, \\ *,*,*, \\ FIT \end{pmatrix} \tag{2.8}$$

Where $*$ can take any possible value.

To check if an agent is alive or dead, let $alive$ be a function that takes as input a current state s and a possible successor state s'. This function returns s' if an agent is alive, and $s_{fitness}$ otherwise:

$$alive(s, s') = \begin{cases} s_{fitness} & t(s') = 0 \\ s_{fitness} & b(s') = 0 \\ s' & otherwise \end{cases} \tag{2.9}$$

2.1.2. Defining the action state transitions. In section 1.3 we describe which actions an agent can take, given its age and current reserves. Let $A(s)$ be the set of all actions possible in state s. Specifically, the set of actions an agent can consist of the following actions:

- 'search': the agent searches for a new resource
- 'wait' (waiting model only): an agent wait one more time step;

- ‘postpone(p)’ (postponing model only): an agent will postpone p times;
- ‘none’ (postponing model only): no choice, an agent is still busy with postponing;
- ‘consume’: it consumes the resource, terminating the encounter.

2.1.3. State transitions in the action phase. A transition function $P_{action}(s'|s,a)$ returns the probability that an agent moves from state $s \in S^{action}$ to $s' \in S^{mutation}$ after it takes action a in state s. Every subsequent state s' that has a non-zero transition probability is called an immediate successor state of s.

If an agent searches, it starts a new encounter with certainty. The quality of the resource is sampled from the set of all initial resource qualities:

$$P_{action}(s'|s, a = search) = \begin{cases} \Pr(q, \mu_{resource}, \sigma_{resource}) & if\ s' = s\begin{pmatrix} b(s), \mathrm{q}_{initial}, \\ 0,0, t(s), \\ MUT \end{pmatrix} \\ 0 & otherwise \end{cases} \quad (2.10)$$

If an agent waits, it transitions from the action phase of the time step to the mutation phase without any other changes:

$$P_{action}(s'|s, a = wait) = \begin{cases} 1 & if\ s' = s\begin{pmatrix} b(s), q_0(s), \\ d(s), \emptyset, t(s), \\ MUT \end{pmatrix} \\ 0 & otherwise \end{cases} \quad (2.11)$$

Similarly, if it is busy postponing, it decrements p, and transitions to the mutation phase.

$$P_{action}(s'|s, a = none) = \begin{cases} 1 & if\ s' = s\begin{pmatrix} b(s), q_0(s), \\ d(s), post(s) - 1, \mathrm{t(s)}, \\ MUT \end{pmatrix} \\ 0 & otherwise \end{cases} \quad (2.12)$$

If it postpones (i.e., at the start of the encounter), an agent will postpone for x time steps:

$$P_{action}(s'|s, a = postpone(p)) = \begin{cases} 1 & if\ s' = s\begin{pmatrix} b(s), q_0(s), \\ d(s), p, \mathrm{t(s)}, \\ MUT \end{pmatrix} \\ 0 & otherwise \end{cases} \quad (2.13)$$

If an agent consumes a resource, the quality of the resource is added to the reserves. Note that the quality has changed during the resource encounter (section 1.2.1). If an agent has not died after consuming the resource (i.e., its reserves are above 0), it transitions to a mutation state where it has not yet encountered a new resource:

$$P_{action}(s'|s, a = consume) = \begin{cases} 1 & if\ s' = alive\left(s\begin{pmatrix} b(s) + q(q_0, s(d)), \emptyset, \\ \emptyset, \emptyset, t(s), \\ MUT \end{pmatrix}, s\right) \\ 0 & otherwise \end{cases} \quad (2.14)$$

2.1.4. Transitions in the extrinsic phase. The mutation transition function $P_{mutation}(s'|s)$ takes at input a state $s \in S^{mutation}$, and returns a state $s' \in S^{action}$ (note that s' is in the next time step). During the mutation phase an agent transitions three times in sequential order. First, if the agent is in a resource encounter, there is a fixed probability that the encounter is interrupted:

$$P_{interruption}(s'|s) = \begin{cases} \rho & if\ s' = s\begin{pmatrix} b(s), \emptyset, \\ \emptyset, \emptyset, t(s), \\ MUT \end{pmatrix} \\ 1-\rho & if\ s' = s\begin{pmatrix} b(s), q_0(s), \\ d(s), post(\mathrm{s}), t(s), \\ MUT \end{pmatrix} \\ 0 & otherwise \end{cases} \quad (2.15)$$

Second, an agent experiences an extrinsic event, which affects its reserves. Afterwards, if its reserves are 0, it dies. The value of an extrinsic event depends on the environment:

$$P_{extrinsic}(s'|s) = \begin{cases} Pr\ (e, \mu_{resource}, \sigma_{resource}) & if\ s' = alive\left(s\begin{pmatrix} b(s) + e, \mathrm{q_0(s)} \\ d(s), post(s), t(s), \\ MUT \end{pmatrix}, s\right) \\ 0 & otherwise \end{cases} \quad (2.16)$$

Third and finally, at the end of the mutation phase, an agent's age and delay increases with 1:

$$P_{increment\ time}(s'|s) = \begin{cases} 1 & if\ s' = alive\left(s\left(\begin{matrix} b(s), q_0(s), \\ d(s)+1, post(s), t(s)+1, \\ ACT \end{matrix}\right), s\right) \\ 0 & otherwise \end{cases} \quad (2.17)$$

By chaining these three transition functions together, we get the mutation transitions function:

$$P_{extrinsic}(s'|s) = P_{interruption}(s'|s) * P_{extrinsic}(s'|s) * P_{increment\ time}(s'|s) \quad (2.18)$$

2.1.5. Successor states. Above we noted that a state s' is an immediate successor of state s if there is a non-zero transitions probability of going from s to s'. Using the functions defined in 2.1.3 and 2.1.4, we can formalize this as:

$$successors(s) = \begin{cases} \forall_{a \in A, s \in S}: \{\mathrm{P_{action}}(\mathrm{s'|s, a}) > 0\} & if\ s \in \mathrm{S^{action}} \\ \forall_{s' \in S}: \{\mathrm{P_{mutation}}(\mathrm{s'|s}) > 0\} & if\ s \in \mathrm{S^{mutation}} \end{cases} \quad (2.19)$$

This function returns the set of all states that are successors of s. Moreover, let $isSuccessor(s, s')$ be a function that returns a Boolean value indicating whether s' is a successor of s':

$$isSuccessors(s, s') = \begin{cases} \mathrm{TRUE} & if\ s' \in \mathrm{successors(s)} \\ \mathrm{FALSE} & otherwise \end{cases} \quad (2.20)$$

2.2. Finding the optimal policy

To find the optimal policy $\pi^*(s)$, ModelBuilder proceeds in two steps. During the forwards pass it constructs a decision tree (section 2.2.1). During the backwards pass, it uses backwards inductions to compute the optimal policy in that tree (section 2.2.2).

2.2.1. Forward pass: computing all possibilities

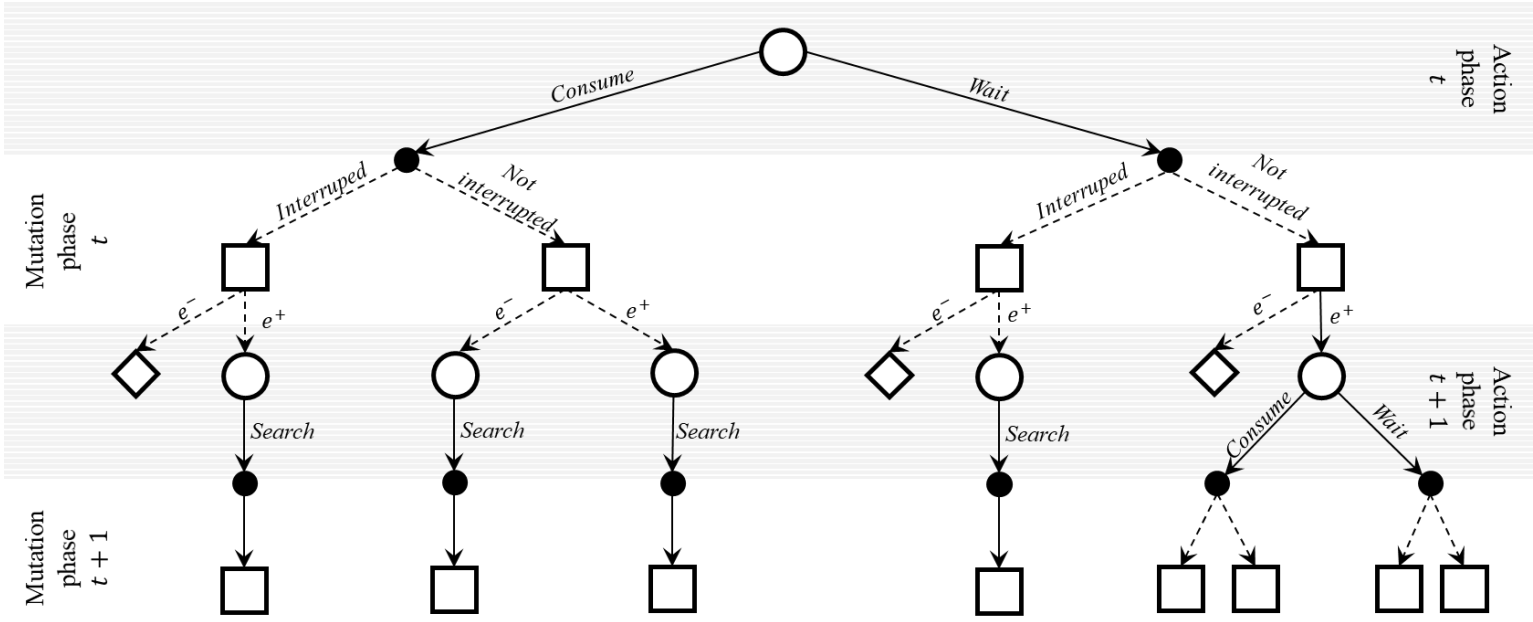

Figure A5.
An example decision tree after the forward pass in the waiting model. This tree is truncated after the second mutation phase – the tree should continue, but it not shown here. The large round circles represent action states, where an agent decides which action to take. Here it can consume or wait. After taking an action, it proceeds to an transitionary state, represented here with a black dot. After this transitionary state, representing whether a resource is interrupted, it proceeds to a mutation state. During the mutation state, represented with a square, it increases in age, and receives an extrinsic event. Should it die during any moment, it proceeds to a fitness state, represented by a large diamond. Finally, edges between nodes can be deterministic (solid) or probabilistic. For completeness, this example shows a hypothetical situation where $b_t = 0$, $e = uniform(-1,1)$, $q_0 = 4, q_1 = 4.8,\ q_2 = 5.6$, and $\rho = 0.5$.

First is the forward pass, where the program builds a decision tree for all possible starting states. A decision tree is a directed a-cyclical graph (a DAG; Figure A5). In this graph there are nodes and edges. Nodes represent a state; there is one node per state, and one state per node. For convenience, we treat states and nodes to be the same thing.

Nodes come in four flavors. First, there are action nodes where an agent has to make a decision (depicted as large circles). Second, there are transition nodes where an agent has made a decision, but does not know the outcome yet (small circles). Action states do not really exist; rather, we use them as a symbol to show which action follow an action state after taking a particular action. Third are the mutation nodes that represent mutation states (depicted as large rectangles). And fourth, there are fitness nodes that represent fitness states (large diamonds). With the exception of the action transition nodes, all nodes refer to exactly one state $s \in S$.

Nodes are connected via edges. There are two types of edges. The first type is deterministic (solid lines), where an agent knows what will happen next (e.g., after taking an action, it

knows to which action state it will transition). The second is weighted. Here the agent does not know which edge it will follow. However, it does know the probability of each edge, represented by its weight. Let $edge(s, s', p)$ be an edge that connects a state s with s', with a weight of p. Note that $p = 0$ if $s' \notin successor(s)$.

2.2.1.1. Step 1: initialization

This decision tree is constructed in an iterative fashion. The algorithm starts by creating a $Tree(S, EDGE)$, consisting of states S^{tree} and edges $EDGE^{tree}$. While constructing a tree, each state can be either explored or unexplored. A state is explored if the algorithm already computed all possible successor states. It is unexplored otherwise. By definition, a fitness state is always explored as it has no successor states. Let EXP be the set of all explored states.

During the initialization step, the algorithm creates a set of non-explored states called the 'frontier', F. Let $F_{action}(t)$ return all action states $s \in S^{action}$ in the frontier where the agent's age is t, let $F_{mutation}(t)$ return all mutation states $s \in S^{mutation}$ in the frontier where the agent's age is t, and let $F_{fitness}$ be the set of all fitness sets in F.

At the start of the algorithm, all states $s_{start} \in S^{start}$ are placed in the frontier. EXP contains only $s \in S^{fitness}$:

$$F = S^{start} \bigcup S^{fitness} \tag{2.21}$$

$$EXP = S^{fitness} \tag{2.22}$$

Moreover, let $EDGE_{tree}$ be the set of all edges included in the tree, which is initially empty:

$$EDGE^{Tree} = \emptyset \tag{2.23}$$

2.2.1.2. Step 2: selecting a state in the frontier

Next, the algorithm selects a focal state, s_{focal} from F. Let i be an index value, starting at 0. It first selects all states in $F_{action}(i)$. If this selection is empty, it then selects $F_{mutation}(i)$. If this selection is also empty, it increments i with 1 and repeats the last two steps.

2.2.1.3. Step 3: exploring a state

After selecting a new focal state, the state is explored. Specifically, it first moves s_{focal} from F to EXP:

$$F = F \setminus s_{focal} \tag{2.24}$$

$$EXP = EXP \bigcup s_{focal} \tag{2.25}$$

Where '$x \backslash y$' means 'remove y from the set x'. It then finds $newStates$, the set of all immediate successor states not already explored:

$$newStates(s) = \forall s' \in S: \{isSuccessor(s, s') \wedge s' \notin EXP\} \tag{2.26}$$

It adds all unexplored successor states to the frontier F:

$$F = F \bigcup newStates(s_{focal}) \tag{2.27}$$

Moreover, it adds an edges going from s_{focal} to a successor states of s', for all successor states. Specifically, if s_{focal} is an action state, it adds edges for all possible consequences of all possible actions (Figures A5, A6, and A7 use the action transition nodes for clarity). If s_{focal} is a mutation state, we add an edge for all immediate successor states.

$$newEdges(s) = \begin{cases} \{ edge(s, s', P_{action}(s|s', a)) \mid \forall a \in A(s), s' \in successors(s) \} & if\ s \in S^{action} \\ \{edge(s, s', P_{mutation}(s, s') \mid s' \in successors(s)\} & if\ s \in S^{mutation} \end{cases} \tag{2.28}$$

$$EDGE^{tree} = EDGE^{tree} \bigcup newEdges(s_{focal}) \quad (2.29)$$

If there are still actions or mutations states in the frontier, the algorithm proceeds to step 2. Otherwise, the algorithm terminates; the tree is finished.

2.2.2. Finding the optimal policy: the backward pass. After the forward pass, we can compute the expected fitness of each state using stochastic dynamic programming and backwards induction. This is called the *backwards pass* (Figure A6)

Let $\omega(s)$ be the expected fitness when an agent is in a state s. We gradually build up this function. First, it is trivial to compute the expected fitness for a fitness state. In section 1.5. we defined the fitness to be the same as the agent's reserves:

$$\omega(s) = b(s) \;\; if \;\; s \in S^{fitness} \quad (2.30)$$

Using backwards induction, we can compute the expected fitness $\omega(s)$ of being in a non-final state by computing the expected fitness of the states with the highest age, and working backwards (i.e., towards the node s_{start}). Conceptually, if we know the expected fitness of all successor states, we can compute the expected fitness of being in that state, as it is the weighted average of all successor states, weighted for the probability that an agent will go to that successor state.

The fitness of being in an action state s is the expected fitness of the fitness-maximizing action in that state. Note that multiple actions might be tied as the best action, in which case we assume that an agent randomly selects an action.

Formally, let $\omega(s, a)$ be the expected fitness of performing an action a in action state s, and let $A^*(s)$ be the set of all actions that maximize fitness. The expected fitness of being in a state s is then:

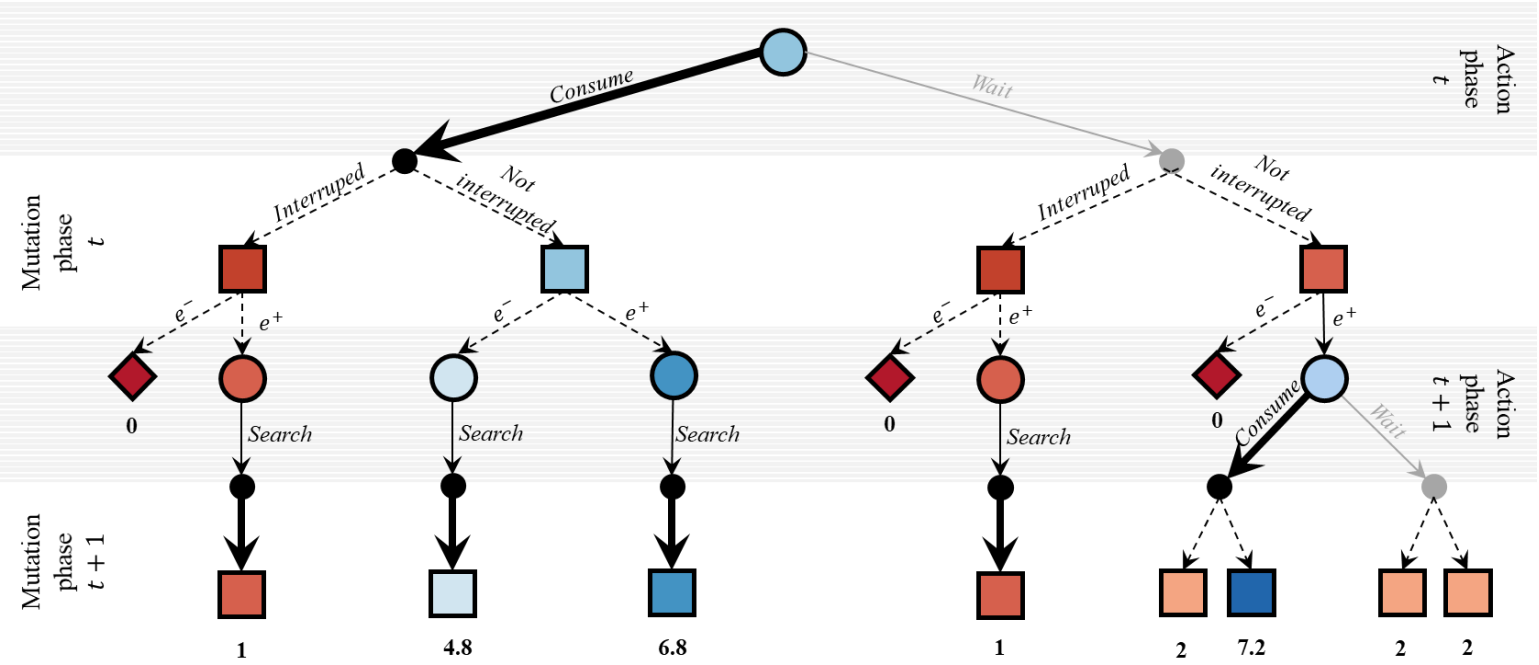

Figure A6.
An example decision tree after the backwards pass in the waiting model. This tree is truncated after the second mutation phase – the tree should continue, but it not shown here. The numbers underneath represent the current reserves of the agent at that point. The colors represent the expected reserves (a proxy for fitness) in each state; redder colors representing lower values, and bluer colors representing higher values. Note that this graph is not precise, but rather, provides a broad impression. Finally, thick lines represent which action an agent takes in each action phase, assuming that it maximizes long-term fitness. For completeness, this example shows a hypothetical situation where $b_t = 0$, $e = uniform(-1,1)$, $q_0 = 4, q_1 = 4.8,\ q_2 = 5.6$, and $\rho = 0.5$.

$$\omega(s) = \begin{cases} b(s) & if\ s \in S^{fitness} \\ argmax_{a \in A(s)}\ \omega(s,a) & if\ s \in S^{action} \end{cases} \tag{2.31}$$

The expected fitness of an action a in state s is the sum of the fitness of each of successor states s', weighted for the transition probability to that successor state:

$$\omega(s,a) = \sum_{s' \in S} P_{action}(s'|s,a) * \omega(s') \tag{2.32}$$

The set of best actions for that state is then the set of all actions that result in the maximum fitness:

$$A^*(s) = \{a|\ \omega(s,a) = \omega(s)\ \} \tag{2.33}$$

The expected fitness of a mutation state is the expected fitness of all success states of that mutation state, weighted by the probability of ending up in that successor state:

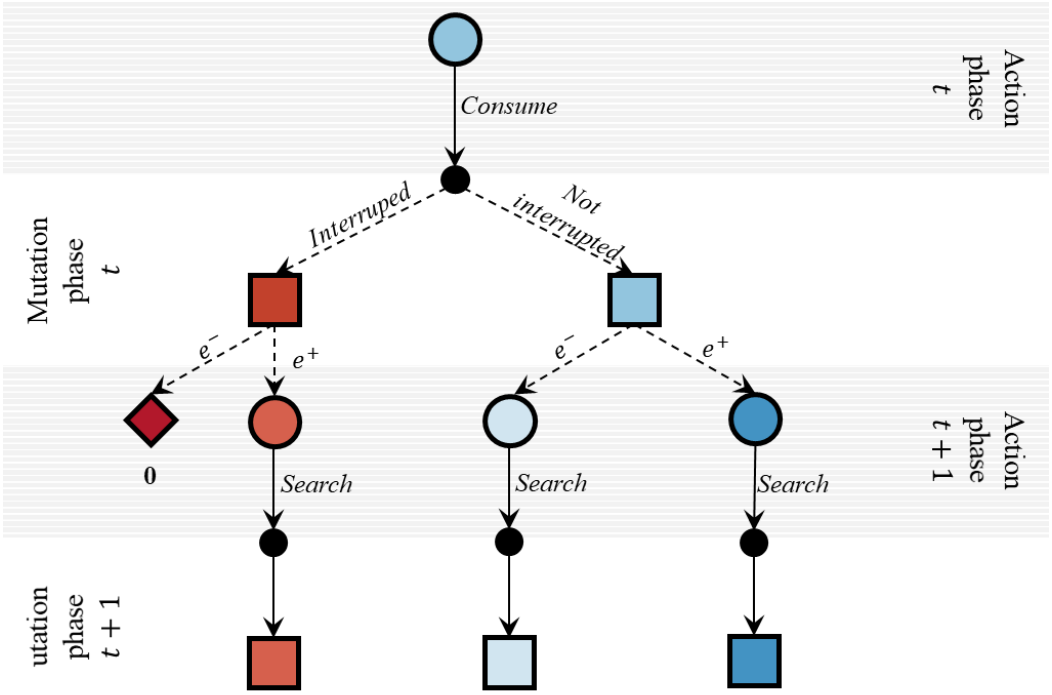

Figure A7.
An example decision tree after all non-optimal actions are removed. This tree is truncated after the second mutation phase – the tree should continue, but it not shown here. The numbers underneath represent the current reserves of the agent at that point. The colors represent the expected reserves (a proxy for fitness) in each state; redder colors representing lower values, and bluer colors representing higher values. Note that this graph is not precise, but rather, provides a broad impression. Finally, thick lines represent which action an agent takes in each action phase, assuming that it maximizes long-term fitness. For completeness, this example shows a hypothetical situation where $b_t = 0$, $e = uniform(-1,1)$, $q_0 = 4, q_1 = 4.8,\ q_2 = 5.6$, and $\rho = 0.5$.

$$\omega(s) = \begin{cases} b(s) & if\ s \in S^{fitness} \\ argmax_{a \in A(s)}\ \omega(s,a) & if\ s \in S^{action} \\ \sum_{s' \in S} P_{mutation}(s'|s) * \omega(s') & if\ s \in S^{mutation} \end{cases} \quad (2.34)$$

2.2.3. Turning the tree into a policy. Finally, during the backwards pass, the algorithm remove all action nodes and connecting edges from actions that are not in $A^*(s)$ (Figure A7). Likewise, all states that are no longer reachable from a root node are removed as well. A state s' is reachable from s if, following the tree from s, there is at least one path that results in s'. This results in the recursive function:

$$reachable(s,s') = \begin{cases} true & s = s' \\ false & s \in S^{fitness} \\ \exists_{s'' \in successors(s')}: reachable(s'',s') = \text{true} & otherwise \end{cases} \quad (2.35)$$

Let $Tree(s)$ refer to the reduced decision tree that contains only nodes reachable from $s \in S$:

$$TREE(s) = \forall_{s' \in S}\{reachable(s, s') = true\} \quad (2.36)$$

(Implicitly, this tree also contains all edges between the states in the tree). If an agent takes a random action from $A^*(s)$, and starting from root node s, $TREE(s)$ describes the optimal policy $\pi^*(s)$. We'll call this tree $TREE_{\pi^*}(s)$.

3. Quantifying delays and other interesting things

There are (at least) six ways to quantify the expected delay given the optimal policy $\pi^*(s)$, each providing different insights (Table A1). The first four measures differ in their time frame; we can either study the expected number of delays during an agent's entire life, or only during the first encounter. Moreover, these four differ in whether we measure actual outcomes or intentions. Finally, if we look at the entire lifespan, we can compute the expected proportion of time that an agent spends delaying by dividing the expected number of delays by the expected age. We first formalize the two dimensions.

3.1. Time frame.

The first dimension is how far we look into the future. Or specifically, which states we include when computing the expected number of delays. First, we can compute the expected delay times in the first resource encounter as the number of times an agent delays (waiting or postponing), during the first resource it encounters. We can find all states within the same encounter as state s using the recursive function $SE(s)$ (a short-hand for "Same Encounter'):

$$SE(s) = \begin{cases} \emptyset & q_0(s) = \emptyset \\ \forall_{s\prime \in successors(s)}: \{q_0(s) \neq \emptyset\} & if\ s \in \mathrm{S}^{\mathrm{action}} \\ \forall_{s\prime \in successors(s)}: \{q_0(s) \neq \emptyset\} \bigcup SE(s') & if\ s \in \mathrm{S}^{\mathrm{mutation}} \end{cases} \quad (3.1)$$

To compute the expected number of delays, we use all states in the same encounter as the root node:

$$includedStates_{first}(s^{start}) = SE(s^{start}) \quad (3.2)$$

Second, we can study the entire lifespan by including all states $Tree_{\pi^*}(s)$:

$$includedStates_{life}(s^{start}) = Tree_{\pi^*}(s^{start}) \quad (3.3)$$

And, finally, let $includedStates_*$ refer to either of the two.

3.2. Actual behavior or intent

Table A1

Six ways to measure delay times

	Time span		
	First	Lifetime	Proportion
Outcome	The expected time steps an agent will delay during the first encounter. + Sensitive to intial conditions + Easy to interpret - Confounded with interruption rate *Appendix B*	The expected number of time steps an agents spends delaying across the lifespan - Confounded with interruption rate - Confounded with life expectancy *Appendix B*	The proportion of its life an agent expects to delay. + Independent of life expectancy - Confounded with interruption rate *Appendix B*
Intention	The expected number of times steps an agent would delay during the first encounter, if there would be no interruptions + Independent of interruption rate + Mostly independent of life expectancy - Cannot compute when waiting ***Main text;*** *Appendix B*	The expected number of times steps an agent would delay over its lifespan, if there would be no interruptions - If there are interruptions, this measure can be higher than an agent's expected age - Cannot compute when waiting *Appendix B*	The proportion of its lifespan an agent would wait, if there would be no interruptions - Values can be higher than 1 if there are interruptions *Not computed*

Note. Different measures are used for the waiting and postponing model. However, in absence of interruptions, these measures are comparable, allowing a direct comparison between the qualitative predictions of both models.

The second dimension is whether we consider expected actual outcomes, or expected intended behavior (i.e., what an agent 'plans' to do). The expected actual outcome quantifies how often an agent will delay in the relevant set of states. The expected intention is how often an agent selects a delaying action. Consider an agent that 'wants' to delay the maximum number of eight time steps. If there is an interruption, the decision might be over before this maximum is reached. For instance, an agent 'wanting' to delay might end up delaying equally long as an agent that 'wants' to delay less.

3.2.1. Expected actual delays. The expected number of actual delays in state s consists of two parts: how many times an agent delays in s, plus the expected number of delays there will be in all successor states s'. We can formalize this if we let A^{actual} be the set of actions that will result in one more time step spend delaying: 'wait' and 'none'. If we use $ImAct$ as a

short-hand notation for "immediate actual", and IS_* as a short-hand for $includedStates_*$, then:

$$imAct(s) = \begin{cases} \frac{|A^* \in A^{actual}|}{|A^*|} & if\ s \in S^{action} \\ 0 & if\ s\ \in S^{mutations} \end{cases} \quad (3.4)$$

And $futAct$ (short for 'future actual') is:

$$futAct(s, IS_*) = \sum_{s' \in succesor(s)} \begin{cases} 0 & if\ s' \notin IS_* \\ \sum_{a \in A^*} P_{action}(s'|s,a) * actual(s', IS_*) * \frac{1}{|A^*|} & if\ s \in S^{action} \\ P_{mutation}(s'|s) * actual(s', IS_*) & if\ s \in S^{mutation} \end{cases} \quad (3.5)$$

Combining the two gives:

$$actual(s, includedStates) = imAct(s) + futAct(s, includedStates) \quad (3.6)$$

3.2.2. Expected intended delays. Instead of the actual delays, we can compute the expected intended delays. Specifically, the intended delay of the set of optimal actions is:

$$intended(A^*(s)) = \frac{\sum_{a \in A^*(s)} \begin{cases} 0 & if\ a = postpone(0) \\ 1 & if\ a = postpone(1) \\ 2 & if\ a = postpone(2) \\ 3 & if\ a = postpone(3) \\ 4 & if\ a = postpone(4) \\ 5 & if\ a = postpone(5) \\ 6 & if\ a = postpone(6) \\ 7 & if\ a = postpone(7) \\ 8 & if\ a = postpone(8) \\ 1 & if\ a = wait \\ 0 & otherwise \end{cases}}{|A^*(s)|} \quad (3.7)$$

Following the same idea, we can decompose a state's expected intended delays as the sum of the expected immediate intended delay and the expected future intended delay. The immediate expected intended delay ('$imInt$') then is:

$$imInt(s) = \begin{cases} intended(A^*(s)) & if\ s \in S^{action} \\ 0 & if\ s\ \in S^{mutations} \end{cases} \quad (3.8)$$

And the future intended expected delay ('$futInt$') is:

$$futInt(s, IS_*) = \sum_{s\prime \in succesor(s)} \begin{cases} 0 & if\ s' \notin IS_* \\ \sum_{a \in A^*(s)} P_{action}(s'|s,a) * intent(s', IS_*) * \frac{1}{|A^*(s)|} & if\ s \in S^{action} \\ P_{mutation}(s'|s) * intent(s', IS_*) & if\ s \in S^{mutation} \end{cases} \quad (3.9)$$

Combining the two gives:

$$intent(s, includedStates) = imInt(s) + futInt(s, includedStates) \quad (3.10)$$

3.3. Computing the expected delay

ModelBuilder uses a similar approach as with the backwards pass for fitness (section 2.2.2) to compute the life expectancy. Specifically, let $expectedAge(s)$ be the age the agent expects to die on when it is currently in state s. Computing the expected age is trivial if the agent is already dead:

$$expectedAge(s) = \{t(s) \quad if\ s \in S^{fitness} \quad (3.11)$$

The expected age of an action state is the weighted sum of the expected age of each successor state of taking a random action from A^*. Let $sum_{age}(s)$ be a shorthand notation this weighted sum:

$$sum_{age}(s) = \sum_{a \in A^*(s)} \sum_{s\prime \in successors(s)} \frac{expectedAge(s') * P_{action}(s'|s,a)}{|A^*(s)|} \quad (3.12)$$

$$expectedAge(s) = \begin{cases} t(s) & if\ s \in S^{fitness} \\ sum_{age}(s) & f\ s \in S^{action} \\ \sum_{s' \in S} P_{mutation}(s'|s) * \omega(s') & f\ s \in S^{mutation} \end{cases} \quad (3.13)$$

Finally, the expected age of a mutation state is weighted sum of the expected age of all successor states:

$$expectedAge(s) \quad (3.14)$$
$$= \begin{cases} t(s) & if\ s \in S^{fitness} \\ sum_{age}(s) & f\ s \in S^{action} \\ \sum_{s' \in S} P_{mutation}(s'|s) * expectedAge(s') & f\ s \in S^{mutation} \end{cases}$$

3.4. Computing the expected delay

Using the definitions from 3.1 and 3.2, we can define the expected intended delay and the expected actual delay for both the first encounter and the lifespan.

$$actual_{first\ encounter}\ (s^{start}) = actual\big(s, includedStates_{first}\big) \quad (3.15)$$

$$actual_{lifetime}\ (s^{start}) = actual\big(s, includedStates_{life}\big) \quad (3.16)$$

$$intend_{first\ encounter}\ (s^{start}) = intent\big(s, includedStates_{first}\big) \quad (3.17)$$

$$intend_{liftime}\ (s^{start}) = intent\big(s, includedStates_{life}\big) \quad (3.18)$$

Moreover, using the expected age described in section 3.3, we can compute the proportion of the lifespan that an agent expects to delay to be:

$$actual_{proportion}\ (s^{start}) = \frac{actual\big(s, includedStates_{life}\big)}{expectedAge(s)} \quad (3.19)$$

$$indent_{proportion}\ (s^{start}) = \frac{intent\left(s, includedStates_{life}\right)}{expectedAge(s)} \tag{3.20}$$

3.6. Measuring sensitivity

Finally, ModelBuilder computes the sensitivity of the policy during the first time step. The sensitivity is a measure of how much the optimal action differs from all other possible actions in terms of its expected fitness. Specifically, it is the standard deviation of the expected fitness of all actions in the first time step:

$$sensitivity(s^{start}) = \frac{\sum_{s\prime \in IS_{first}} \sum_{a \in A} \left(\omega(s) - \omega(s'|a)\right)^2}{n} \tag{3.21}$$

Where n is the total number of actions in all states in SI_{first}.

4. Table containing all symbols and functions used

Variable	Description	Domain/range
Description of decision problem		
T	The set of all time steps. This time step is the same as an agent's age.	$\mathbb{Z} \in [0,100]$
t	One specific time step	T
D	The set of all possible delay times. In the postponing model, the number of time steps an agent can postpone. In the waiting model, the maximum number of times an agent can wait.	$\mathbb{Z} \in [0,8]$
d	The number of time steps that have passed since the start of the current resource encounter.	D
B	The set of all possible reserves that an agent can have.	$\mathbb{R} \in [0,100]$ Step size: 0.1
b_t	An agent's reserve at time t.	
Q	The set of all possible resource qualities.	$\mathbb{R} \in$ [-15,15] Step size: 0.1
$Q_{initial}$	The set of all possible qualities a resource can have at the start of an encounter.	$\mathbb{Z} \in$ [-15,15]
q	The quality of one specific resource.	
$Pr\ (q, \mu_{resource}, \sigma_{resource})$	The probability that a resource has quality q at the start of the encounter, given the mean and standard deviation of resources in the environment.	$\mathbb{R} \in [0,1]$
q_x	The quality of a resource after x time steps in the encounter.	Q
$q(q_0, d)$	The quality of a resource, given that it had value q_0 at the start of the encounter, and d time steps have passed since.	Q
A	The set of all actions.	
$A(t, b_t)$	The set of all actions possible at time t, given that an agent's current reserves is b_t.	
a	A particular action.	A
p	The number of time steps an agent chooses to postpone. Postpone model only.	$\mathbb{Z} \in [0,8]$
E	The set of all possible extrinsic event values.	Depends on environment
e	One specific extrinsic event value.	
$Pr\ (e, \mu_{extrinsic}, \sigma_{extrinsic})$	The probability that an extrinsic has value e at the start of the encounter, given the mean and range of extrinsic events in the environment.	$\mathbb{R} \in [0,1]$
$\omega(b_t)$	The fitness an agent has at the point of death, given its reserve is b_t. We use a linear mapping; the fitness of b_t is b .	$\mathbb{R} \in [0,100]$
Environmental variables		
$\mu_{resource}$	Mean resource quality, a type of harshness.	$\mathbb{R} \in$ [-4,4] Step size: 1
$\sigma_{resource}$	Standard deviation of resource quality, a type of unpredictability.	$\mathbb{R} \in [0,8]$ Step size: 2
$\mu_{extrinsic}$	Mean extrinsic event quality, a type of harshness.	$\mathbb{R} \in$ [-1,1] Step size: 1

$\sigma_{extrinsic}$	Range of extrinsic event quality, a type of unpredictability.	Can be 0, 2, or 4
ρ	Interruption rate – the rate at which resources become unavailable after each time step. A type of unpredictability.	Can be 0, 0.2, or 0.5
env	An environment. Environments are a 5-tuple containing a $\mu_{resource}$, $\mu_{extrinsic}$, $\sigma_{resource}$, $\sigma_{extrinsic}$, and ρ.	
ENV	The set of all environments.	

Finding the optimal policy

S	The set of all possible states.	
S^{start}	The set of all possible starting states.	
S^{action}	The set of all possible states where an agent is currently in an action phase (i.e., it makes a decision).	
$S^{extrinsic}$	The set of all possible states where an agent is currently in a mutation phase.	
$S^{fitness}$	The set of all possible states where an agent is dead – i.e., has 0 reserves or has reached the maximum age	
s	The current state of an agent, consisting of a reserve b, an initial resource quality q_0, the number of time steps that has passed since the start of an encounter d, the number of times an agent wants to postpone p, an age t, and a phase ph.	S
$s_{fitness}$	A wildcard state referring to all states in $S^{fitness}$	
$b(s)$	Returns the reserves of an agent in state s.	B
$d(s)$	Returns the number of time steps since the start of the resource encounter in state s. Returns $\emptyset$ when not in an encounter.	d
$q_0(s)$	Returns he resource quality at the start of the current encounter when in state s. Returns $\emptyset$ when not in an encounter.	Q_0
$t(s)$	An agent's age in state s.	T
$post(s)$	How long an agent still has to postponing before it consumes the resource in state s. Returns $\emptyset$ when not in an encounter or in the waiting model.	T
$PHASE$	The set of all possible phases of a time step an agent can be in.	$\{ACT, MUT, FIT\}$
$phase(s)$	Returns the phase of the time step the agent is in in state s.	$PHASE$
$s(b, q_0, d, t, p, ph)$	Returns the unique state where an agent has reserves b, encountered a resource with quality q_0, has delayed d times in the current encounter, has age t, will postpone a total of p times, and is currently in the ph phase of the tth time step.	S
$alive(s, s')$	If the agent is alive in state s', this function returns s'. It returns $s_{fitness}$ otherwise.	S
$P_{action}(s' \mid s, a)$	The action transition probability function; takes as input a state $s \in S^{action}$ and $s' \in S^{mutation}$, and returns the probability that an agent transitions from s to s' if it takes action a in state s.	$\mathbb{R} \in [0,1]$
$P_{mutation}(s' \mid s)$	The mutation transition probability function; takes as input a state $s \in S^{mutation}$, and $s' \in S^{action}$. Returns the probability that an agent transitions from s to s'.	$\mathbb{R} \in [0,1]$
$successors(s)$	Returns the set of all states that have a non-zero transition probability from s.	S
$isSuccessor(s, s')$	Returns true if $s' \in successors(s)$, and false otherwise	Boolean

$TREE(S^{tree}, EDGE^{tree})$	A tree, consisting of a set of states (or 'nodes) S^{tree} and a set of edge $EDGE^{tree}$.	
S^{tree}	The set of all states in tree $tree$	S
$EDGE^{tree}$	The set of all edges in the tree $tree$.	
$edge(s, s', p)$	An edge connecting s to s' with weight p. An edge represents a transition probability.	
F	The frontier; a set of states not yet explored by the algorithm. Can contain either action states, mutation states, or fitness states.	
EXP	The explored set; a set of states already explored by the algorithm.	
$A(s)$	The set of all action possible when in state s.	A
$A^*(s)$	The set of all action in state s that result in the highest expected fitness.	A
$\omega(s)$	The expected fitness when in state s.	$\mathbb{R} \in [0,100]$
$\omega(s\|a)$	The expected fitness when in state $s \in S^{action}$ when taking action a.	$\mathbb{R} \in [0,100]$
$reachable(s, s')$	Returns true is there is a path from s to s'	Boolean
$TREE(s)$	Returns a tree containing all (and only) states reachable from s, and the edges between those states.	$TREE$
$TREE_{\pi^*}(s)$	Returns a tree containing all (and only) states reachable from s if the agent follows the optimal policy, and the edges between those states.	$TREE$

Quantifying delays and other interesting things

$SE(s)$	Short for 'Same Encounter'. Returns the set of all states that are in the same resource encounter as s	S
$imAct(s)$	Short for 'immediate actual'; the expected number of times an agent expect to delay during the current time step.	$\mathbb{R} \in [0,1]$
$futAct(s, S)$	Short for 'future actual'; the expected number of times an agent expects to delay after this moment, taking into account all states that are in s.	$\mathbb{R} \in [0, T-1]$
$actual(s, S)$	The expected number of times an agent expects to delay, including in the current state, considering all states in S	$\mathbb{R} \in [0, T]$
$imInt(s)$	Short for 'immediate intended; the expected number of times an agent expect to intent to delay during the current time step. Postponing model only.	$\mathbb{R} \in [0,1]$
$futInt(s, S)$	Short for 'future actual'; the expected number of times an agent expects to intent to delay after this moment, taking into account all states that are in s. Postponing model only.	$\mathbb{R} \in [0, \infty]$
$actual(s, S)$	The expected number of times an agent expects to delay, including in the current state, considering all states in S	$\mathbb{R} \in [0, \infty]$
$expectedAge(s)$	The number of time steps (including the current one) that an agent expects to live, given that it follows the optimal policy.	$\mathbb{R} \in [0,100]$
$actual_{proportion}(s)$	The proportion of its total lifetime an agent expects to delay, assuming it follows the optimal policy.	$\mathbb{R} \in [0,1]$
$intent_{proportion}(s)$	The proportion of its total lifetime an agent expects to intent to delay, assuming it follows the optimal policy. Postponing model only.	$\mathbb{R} \in [0, \infty]$
$sensitivity(s)$	The standard deviation of the expected fitness of all actions possible in s. Higher numbers indicate that following the optimal policy results in better outcomes than a random policy.	$\mathbb{R} \in [0, \infty]$

Note: $\mathbb{R}$ refers to the set of all real numbers; $\mathbb{Z}$ refers to the set of all integers.