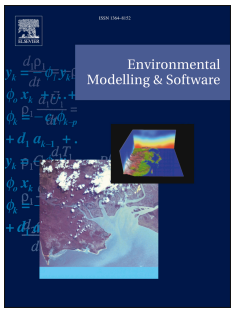
Accepted Manuscript



Repeated discrete choices in geographical agent based models with an application to fisheries

Ernesto Carrella, Richard M. Bailey, Jens Koed Madsen

PII: S1364-8152(17)31106-4

DOI: 10.1016/j.envsoft.2018.08.023

Reference: ENSO 4288

To appear in: *Environmental Modelling and Software*

Received Date: 18 October 2017

Revised Date: 9 August 2018

Accepted Date: 22 August 2018

Please cite this article as: Carrella, E., Bailey, R.M., Madsen, J.K., Repeated discrete choices in geographical agent based models with an application to fisheries, *Environmental Modelling and Software* (2018), doi: 10.1016/j.envsoft.2018.08.023.

Repeated discrete choices in geographical agent based models with an application to fisheries

*Ernesto Carrella**
Richard M. Bailey
Jens Koed Madsen

17 May 2018

Abstract

Most geographical agent-based models simulate agents through custom-made decision-making algorithms. This makes it difficult to assess which results are general and which are contingent on the algorithm's details. We present a set of general algorithms, applicable in any agent-based model for choosing repeatedly from a set of alternatives. We showcase each in the same fishery agent-based model and rank their performance under various scenarios. While complicated algorithms tend to perform better, too much sophistication lowers performance. Further, while some algorithms perform well under all scenarios, others are optimal only in specific circumstances. It is therefore impossible to produce a single, unequivocal performance ranking even for simple general algorithms. We advocate then a heuristic zoo approach where multiple algorithms are implemented in the same model; this allows us to identify its best algorithm and test sensitivity to misspecifications of the decision-making component.

Highlights

- We describe 13 model-free decision algorithms for geographical agent-based models
- Agents in geographical agent-based models often are multi-armed bandits variants
- Theoretical worst case regret is not a good indicator of agent performance
- Some algorithms perform well in general, some only in special conditions
- Imitation and information sharing may reduce overall performance

Keywords

Agent-based Models; Adaptation; Multi-armed Bandit; Fisheries; Bio-economic modelling;

Software availability section

The library containing all the algorithms, `discrete-choosers`, is open-source under MIT license and available on this repository. It is coded in pure Java.

The POSEIDON fisheries simulator, including its implementation of the algorithms, is open-source (GPL-3 license) and available on its repository. POSEIDON is implemented in Java through MASON (Luke et al. 2005).

*Corresponding author:ernesto.carrella@ouce.ox.ac.uk ; This research is funded in part by the Oxford Martin School, the David and Lucile Packard Foundation, the Gordon and Betty Moore Foundation, the Walton Family Foundation, and Ocean Conservancy.

1 Introduction

1.1 Research agenda

There are many agent-based models of fisheries, each deploying its own highly specialized decision algorithm to simulate its fishers. This creates many contingent results with no hope of general insights; O'Sullivan et al. (2016) calls this the YAAWN syndrome: "Yet Another Agent-based model, Whatever Never-mind". We want instead to find a single, simple adaptation strategy that explains the behaviour of fishers across the world. If that is not possible, we would like to isolate the smallest number of adaptation strategies, each optimal under specific environmental conditions. Eventually we would like to extend this strategy-environment mapping to all geographical exploitation problems.

This research programme involves two phases. First, we need to to list and compare candidate decision algorithms and measure their performance in theoretical scenarios. Second, we need to bring these hypotheses to the field and test their presence with real data and interviews. This paper focuses on the first phase of the programme.

If we ignore the idea of fishers as perfectly rational, perfectly informed profit maximization automata we need to grapple with the Anna Karenina principle of adaptive behaviour: every adaptive agent is adaptive in its own way. There are then many possible decision algorithms to compare. We are left then with the task of building a heuristics zoo: implement all algorithms within the same model to test their strengths and weaknesses *ceteris paribus*.

A by-product of this effort is a library of ready-made adaptive algorithms that other agent-based models can use. Our hope is that they will elucidate similar broad patterns as the ones we identify here in fisheries.

1.2 General-purpose algorithms

We present a set of decision-making algorithms usable in any geographical agent-based model. These algorithms maximize the utility of agents making repeated, discrete, geographical choices. These algorithms require little model knowledge as they proceed by trial and error.

The algorithms, listed in table 1, are particularly useful in simulating exploitation of natural resources in a coupled human-environment model. Agents, be they grazers, foragers, fishers or loggers need to continuously decide where to exploit next. They do not know which areas are more bountiful and need to explore constantly to improve their profits.

We use the agent-based fishing model POSEIDON to showcase each algorithm. Fishing is a good test bench as it involves multiple competing agents, a large space to explore, a common-pool resource and unknown biological dynamics. We tried however to present each algorithm independently of its fishery implementation to foster their reuse (Bell et al. 2015; Groeneveld et al. 2017).

We use the multi-armed bandit problem as a framing device. Most of the algorithms presented are not bandit algorithms per se but they can be understood as extensions that deal with bandit algorithms weaknesses.

Table 1: Table containing all the algorithms described in the paper, including a brief description of their memory size, how much imitation they use, whether or not they use any statistical algorithm and in which section they are introduced

Algorithm Name	Algorithm Type	Memory	Imitation	Statistical Algorithm	Paper Section
ϵ-greedy	Bandit Algorithm	Coarse Matrix	None	None	2.1.1
SOFTMAX	Bandit Algorithm	Coarse Matrix	None	None	2.1.2

Algorithm Name	Algorithm Type	Memory	Imitation	Statistical Algorithm	Paper Section
UCB 1	Bandit Algorithm	Coarse Matrix	None	None	2.1.3
Gravitational Search	Imitative Algorithm	Only current position	Extreme	None	2.2.1
Particle Swarm Optimization	Imitative Algorithm	Only current position	High	None	2.2.2
Explore-Exploit-Imitate	Imitative Algorithm	Current and previous position	Medium	None	2.2.3
Social Annealing	Imitative Algorithm	Current and previous position	Low	None	2.2.4
Nearest Neighbour	Heatmap Algorithm	All previous positions	Medium	Nearest Neighbour	2.3.1
Kernel Regression	Heatmap Algorithm	Last 100 positions	Medium	RBF or EPA regression	2.3.2
Kernel Transduction	Heatmap Algorithm	All previous positions	Medium	RBF or EPA regression	2.3.3
Kalman Filter	Heatmap Algorithm	All previous positions	Medium	Gaussian Kalman	2.3.4
GWR	Heatmap Algorithm	All previous positions	Medium	LOESS through Kalman	2.3.5
Ad-hoc	Heatmap Algorithm	All previous positions	Medium	Bayes Rule	2.3.6

1.3 The bandit problem as geographical exploitation

In the multi-armed bandit problem (Bubeck and Cesa-Bianchi 2012; Kuleshov and Precup 2014; Vermorel and Mohri 2005) a player faces K slot machines. Each slot machine dispenses a random reward drawn from a distribution unknown to the player. The objective is to make the most amount of money. The multi-armed bandit problem encapsulates the trade-off between exploration and exploitation: the player would like to play only the best slot machine but needs to find it first.

We can often model decisions in a geographical agent-based model as a multi-armed bandit problem. For example in the case of fisheries we can consider each fishing spot as a "slot machine", the fisher having to balance exploring new fishing spots versus exploiting known ones.

Bandit algorithms are heuristics designed to deal with multi-armed bandit problems. The advantage of bandit algorithms is being model-free: they make no assumption on how rewards are generated. These decision algorithms adapt to incentives, work with little knowledge of the world, are easy to code and perform well even when the reward structure changes over time. We cover them in detail in section 2.1.

Computer science examines bandit algorithms' regret: the difference between the algorithm's rewards and those achieved with perfect knowledge. Computer science often focuses on the growth of worst-case regret with respect to time. When regret grows slowly it means that the algorithm eventually plays optimally after some exploration. Regret grows logarithmically in the best algorithms. Regret grows linearly when the algorithm never settles into optimal performance.

However regret is not necessarily a good indicator of performance in an agent-based model. To myopic agents, eventually finding the best slot machine is often less important than quickly finding a "good" one (that is, we are usually more interested in satisficing performance as in Simon and Feldman 1959). As an extreme example, an agent able to isolate the top 2 slot machines out of millions is intuitively very efficient but its regret scales linearly nonetheless due to its inability to ever choose the best one(Silver 2015).

A more pertinent observation on regret is that it grows linearly with K, the number of slot machines (Auer, Cesa-bianchi, and Fischer 2002; Rusmevichientong and Tsitsiklis 2010). Kuleshov and Precup (2014) (p.9) mentions that tasks with $K > 50$ "are difficult for all algorithms". This failure to scale with the number of options justifies the other algorithms in this paper.

Bandit algorithms scale poorly because as the number of slot machine increases, exploration weakens. We minimize regret by picking the best slot machine but to find it we need to explore the entire slot machine space. Since we can only explore one slot machine at a time, knowledge about relative strengths grows slowly when K is large.

There are two ways to strengthen exploration. First, we can "parallelize" it: exploring multiple slot-machines each time step. This is possible in agent-based models where multiple players are present and may share their exploration results. We do this in section 2.2. Second, we can "add structure" by assuming some interdependency between reward distributions (Caron et al. 2012). This way playing one slot machine reveals something about other slot machines similar to it. This is a straightforward assumption to make in a geographical model. We do this in section 2.3. We provide ways to tune the parameters of each algorithm in the appendix.

1.4 Literature review

The most famous agent-based model with geographical exploitation of resources is surely Sugarscape (Epstein and Axtell 1996). In it, agents extract sugar from a grid cell to survive, reproduce and trade. It was a particularly influential example for how to simulate the exploitation of hidden geographical resources. Sugarscape agents have perfect local vision: they know everything in a radius around them but nothing else. They exploit by moving greedily towards the neighbouring cell with the most sugar.

Perfect local vision and greedy exploitation is a very common assumption in agent-based models of foragers (Pepper and Smuts 2000; Nonaka and Holme 2007) and it makes sense if we assume that agents can only exploit nearby areas and cannot afford any exploration. When simulating common-pool resources, however, it is usually the case that the entirety of the common is available for exploitation but that the precise distribution of its resources is unknown. Under these conditions purposeful exploration becomes necessary.

Bandit algorithms are often used when studying exploration in agent-based models although they are not cited as such. For example the fixed exploration-exploitation ratios in Kunz (2011) model are precisely the ϵ-greedy algorithm we describe in section 2.1.1. SOFTMAX, which we describe in section 2.1.2, powers the foragers in Roberts and Goldstone (2006), and in Satake et al. (2007) forestry managers use SOFTMAX to decide what to harvest (this paper, however, is more nuanced as it adds long-term considerations to the utility function).

Marcoul and Weninger (2008) is the first and only fisheries paper to model fishers as solving a multi-armed bandit problem. The model however is equation-based and since it focuses on optimal behaviour alone it is afflicted by the curse of dimensionality: the model has only one fisher choosing between three cells over a two periods lifetime.

Schlag (1998) developed first the idea of using imitation to improve bandit algorithms, which we explore in section 2.2. This influential paper led to empirical results like the algorithm tournament in Rendell et al. (2010). In this tournament, algorithms could use any combination of imitation and exploration to maximize unknown fitness functions and imitation-only algorithms did better overall. Boero et al. (2010) start their empirically grounded agents as ϵ-greedy bandits but then provide them "hints" about better options, and study how these hints lead to decision changes. The main issue with this literature is that it focuses only on the positive side of imitation: copying people making better choices is often faster than discovering those choices on one's own. However imitation has also a social cost when it comes to common-pool resources as it leads to congestion (many agents competing over resources at a single location) and faster depletion.

The social costs of imitation are better represented in fisheries models. Cabral et al. (2010) show how a large population of imitators fail to exploit properly a geographical resource compared to a smaller population of fishers acting randomly. Little and McDonald (2007) show how social networks and imitation can reduce

overall efficiency when there is little environmental noise. Imitation often leads to worse outcomes in our paper as well.

Little and McDonald (2007) are also a precursor (together with Dorn 2001) in the use of Kalman filters (see Faragher 2012 for an introduction) within agent-based models. We don't know of any other agent-based simulation using these methods. They are common enough in the data analysis that informs and tests models but the agents themselves are never implemented using these regressions. We believe it is simply due to the iterative versions of these estimators not being as well known as their batch counterparts (which are too unwieldy to be used in models directly). Agents using Kalman filters are so far exclusively the domain of "learning" agents in equation-based economic models (Evans and Honkapohja 2001). One notable application, albeit not agent-based model related, in the fisheries literature is Boettiger, Mangel, and Munch (2015) where a non-parametric estimator is used by a policy-maker to forecast biological dynamics. This estimator however creates dimensionality problems of its own and is unusable in an agent-based model.

Wu et al. (2017) is a recent application of statistical algorithms in human experiments of geographical exploitation. Much like this paper, the authors compare simple heuristics against more complicated statistical algorithms like the ones we present in section 2.3. We differ in practicalities: the gaussian process methods used in Wu et al. (2017) are not recursive and are too inefficient to use within agent-based models. Hutniczak and Münch (2018) is another example of non-recursive statistical methods to simulate information gathered through fishing (in this case, within a random utility model analysis).

The idea of implementing multiple decision making algorithms to test them is old. The Axelrod (1980) tournament crowned TIT-FOR-TAT as the victor amongst 14 competitors. Similarly, Fowler and Laver (2008) organized a competition for simulated political parties in which 29 algorithms participated. Sometimes comparisons are between radically different decision algorithms, such as between adaptive and non-adaptive farmers in Lansing and Kremer (1993). Other times they are between different parametrizations of the same general model, as for example between different aspiration levels in Gotts and Polhill (2009). The key of these comparisons for agent-based models is less on finding the "best" algorithm and more about the overall systemic effect that emerges from changes in individual heuristics (as in Polhill, Gotts, and Law 2001).

1.5 POSEIDON

All the algorithms we describe here have been implemented both as part of the POSEIDON agent-based model and as a standalone library `discrete-choosers`. POSEIDON simulates the fishery: its biological environment, geography, markets, boats and policies. Bailey et al. (2018) describes the model and its rules[1].

In all simulations a grid of ocean cells are available for the agents to fish in. Each trip, each agent picks one cell where to fish. This decision is made independently from all other agents (although some decision algorithms may have access to some knowledge about the competition). Each agent knows nothing of the amount of fish available or the underlying biological model.

While POSEIDON can be very complex, the simulations in this paper all occur in a very simplified setting. There is never any fishing regulation, no entry or exit of other boats and no biological stressors except for fishing mortality. Each agent, for each hour spent fishing, collects a linear percentage of available biomass:

$$\text{Hourly Catch} = q * \text{Available Biomass} \tag{1}$$

Where q is the catchability coefficient. All simulation parameters are described in the appendix.

[1] An ODD+D description of the model is available at [http://carrknight.github.io/poseidon/odd.html].

2 Algorithms

2.1 Classic bandit algorithms

In this section we introduce three classic bandit algorithms: ϵ-greedy, the upper-confidence bound (UCB1) and SOFTMAX. They each stand out for a reason: ϵ-greedy is the easiest to understand, UCB1 is proven to have optimal regret, while the SOFTMAX algorithm takes decisions as a random utility choice model.

We test these algorithms in POSEIDON where fishers face slot machines (fishing spots) distributed in a 50 by 50 grid map. We test two variants of each algorithm. In the first variant the bandit algorithm has to choose directly one of 2500 fishing spots. This variant makes no assumption on reward structure: each slot machine's payout is assumed independent of all the others. In the second variant we discretize the map coarsely into 9 parent cells; the bandit algorithm picks one parent and the actual fishing occurs at a random child fishing spot.

Discretization is a rudimentary way to "add structure" to the model. Adding structure (assuming rewards are not all independently distributed) makes sense in geographical models as we expect rewards of two close slot machines to be similar. The easiest way to define two slot machines as "close" is that they share parent cell. The two weaknesses of discretization are that it ignores the difference in rewards between slot machines belonging to the same parent and that it assumes large discontinuities between slot machines that are on the border between two parents.

Discretization is a simplification of the dependent bandits literature(Pandey, Chakrabarti, and Agarwal 2007). In it each slot machine is part of a cluster. Dependent bandits assume that slot machines all have different reward distributions whose parameters are drawn from a second distribution defined by their cluster. This means that playing one slot machine gives us some, but not all, information on the reward of other slot machines in the cluster. A simple, albeit non-optimal, strategy for these kind of problems is to use a recursive bandit algorithm: one bandit algorithm to choose the cluster, and then a second algorithm to pick among the children cells. For the POSEIDON implementation however bandit algorithms pick only the parent cell while the child is chosen at random.

Discretization is commonplace in fisheries as for example the agent-based models in Scott (2016), Soulié and Thébaud (2006) and Yu et al. (2012). While often geographical in nature, a map can be gridded according to other contours (Hynes et al. 2016 splits the map according to habitat conditions for example).

2.1.1 ϵ-greedy algorithm

The ϵ-greedy algorithm is the simplest bandit algorithm and tends to perform relatively well. Assuming K options available, algorithm 1 is a pseudocode implementation of the algorithm.

Algorithm 1 ϵ-greedy implementation

```
means[1, ..., K] ← 0 {Keep track of average reward for each arm}
while game is not over do
  random ~ U[0, 1]
  if random < ε then {with probability ε, explore}
    play arm randomly between 1 and K
  else {otherwise exploit}
    play arm with highest mean
    reward ← play(arm)
    update means[arm] with reward
  end if
end while
```

More succinctly: each step with ϵ probability try a slot machine at random (exploration) otherwise pick the

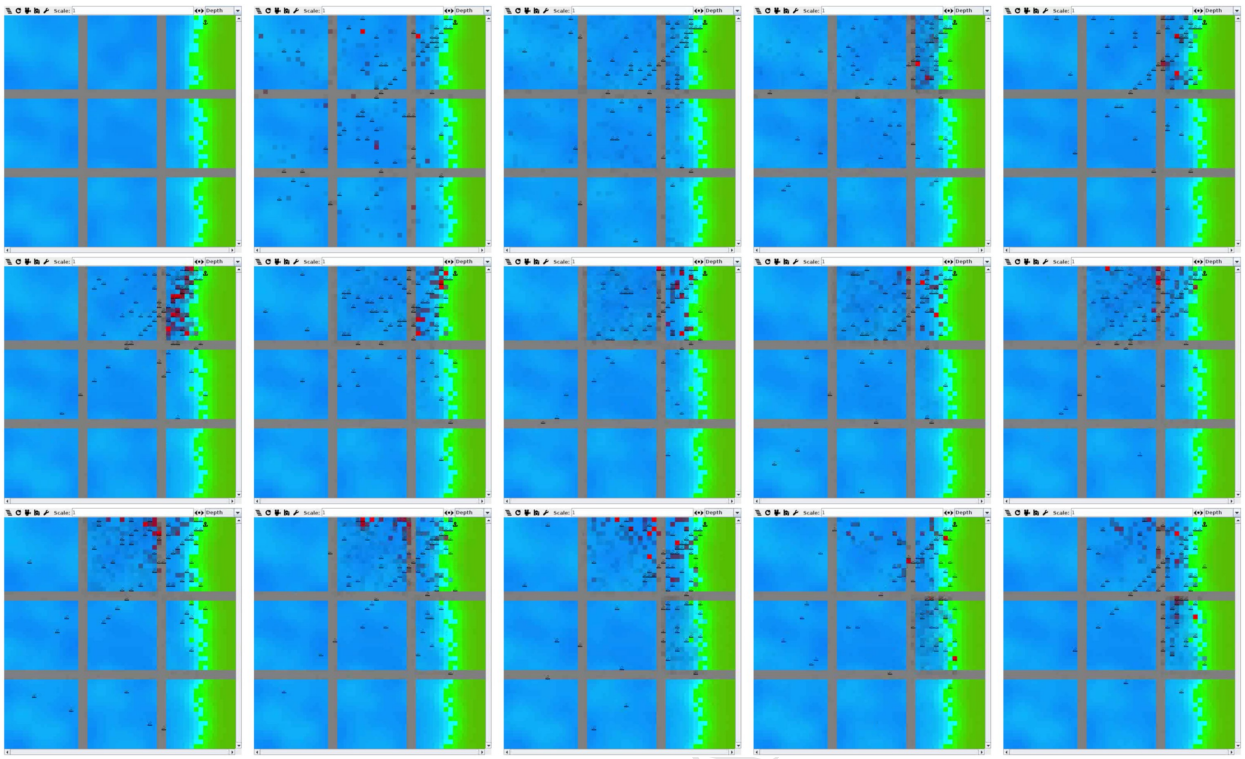


Figure 1: Animation showing a set of 100 fishers, each using a discretized epsilon-greedy decision algorithm

slot machine with the highest empirical mean (exploitation). In this paper the empirical mean is stored as an exponential moving average rather than an arithmetic one to weigh new observations more.

Figure 1 shows its implementation in POSEIDON. We split the map into a 3 by 3 parent grid (the grey lines representing the borders). The bandit algorithm picks the parent cell and then the boat targets a random child cell. Areas turn red as they are fished, the redder the more targeted they are. Quickly the agents learn to fish first near port and then move out as that area is consumed. The discretization is visible by the way agents stick to the parent cell selection. Unrealistic but quite effective.

This algorithm depends on 2 parameters. The exploration rate ϵ and the exponential moving average weight α. When discretized, the number of parent areas is also a parameter.

2.1.2 SOFTMAX

The softmax algorithm works stochastically by choosing each slot machine i with probability

$$p_i = \frac{e^{\mu_i}}{\sum_{j=1}^{K} e^{\mu_j}} \tag{2}$$

where μ_i is the empirical mean reward obtained from previous games with slot machine i. Algorithm 2 shows a possible pseudo-code implementation.

Figure 2 shows the SOFTMAX algorithm implemented in POSEIDON. Each boat acts independently over the same 3 by 3 parent grid as in the ϵ-greedy example. The dynamic generated is similar to the other bandit algorithms: agents converge to fishing near port although the convergence is slower for the SOFTMAX algorithm.

This algorithm depends on 1 parameter: the exponential moving average weight α.

Algorithm 2 SOFTMAX implementation

means$[1, \ldots, K] \leftarrow 0$ {Keep track of average reward for each arm}
while game is not over **do**
  for i $= 0$ to K **do**
    probabilities$[i] \leftarrow e^{\text{means[i]}}$
  end for
  for i $= 0$ to K **do**
    probabilities$[i] \leftarrow \frac{\text{probabilities}[i]}{\sum \text{probabilities}}$ {normalize probabilities vector so that it sums to 1}
  end for
  pick arm to play at random with probability equal to probabilities$[i]$
  reward $\leftarrow$ play(arm)
  update means[arm] with reward
end while

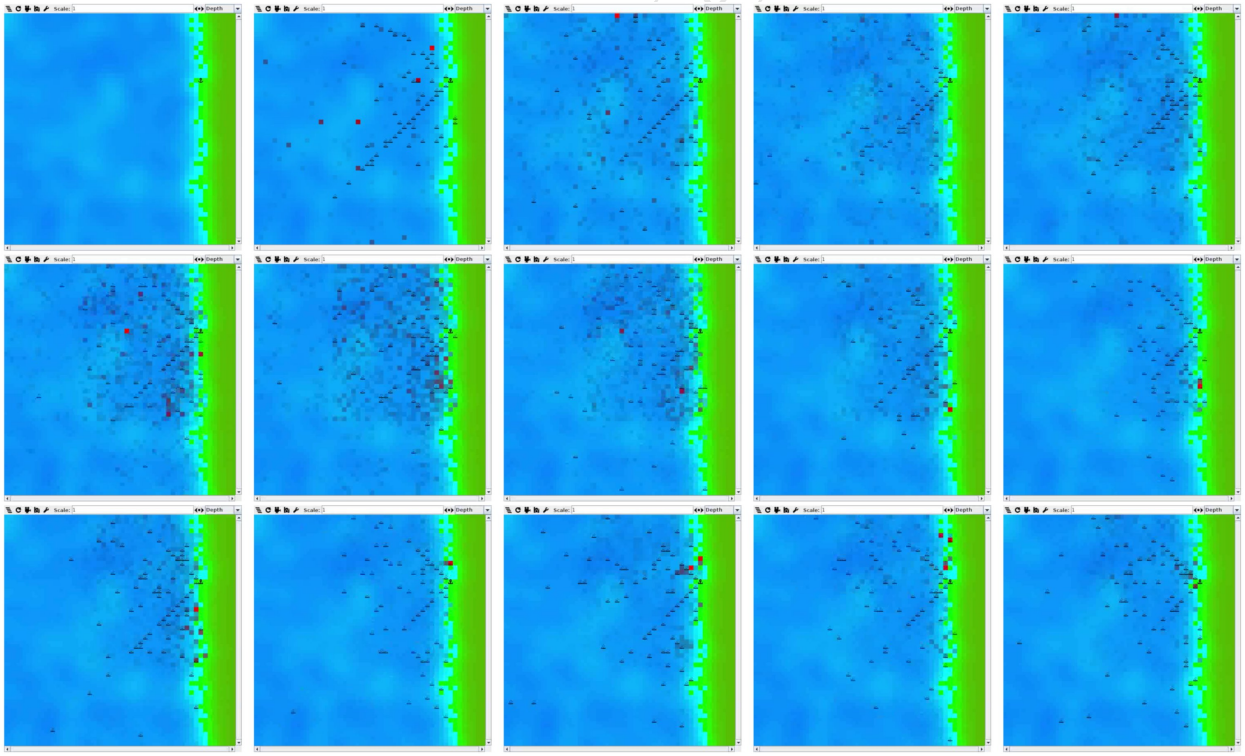


Figure 2: Animation showing a set of 100 fishers, each using a discretized SOFTMAX decision algorithm

2.1.3 Upper confidence bound

A slightly more complicated algorithm, the UCB1 keeps track not only of empirical means but also of confidence intervals. Rather than picking the option with the highest mean, it picks the one with the highest confidence bound.
This allows for the trade-off between exploration and exploitation to be decided by empirical uncertainty rather than the exogenous ϵ parameter. Because the size of the confidence interval is an indicator of uncertainty, UCB1 focuses its exploration on the least known options until sure they are no better than the current best.

To make a concrete example imagine having to choose between two slot-machines, A or B. You think slot-machine A returns a mean reward of 10 ± 1 (where the confidence interval depends on your uncertainty regarding the true mean return). You think slot-machine B returns a mean reward of 5 ± 10. Even though slot-machine A seems to have a better mean reward, UCB1 will pick slot machine B because its mean has a higher upper confidence bound (15 versus 11). Playing slot machine B will reduce its uncertainty, shrinking its confidence interval until either the mean of B is revalued higher or its upper confidence bound is below that of slot machine A.

Algorithm 3 shows a simple pseudo-code implementation of UCB1. The algorithm requires reward to be normalized from 0 to 1 unlike ϵ-greedy and SOFTMAX.

Algorithm 3 UCB1 implementation

means$[1, \ldots, K] \leftarrow 0$ {Keep track of average reward for each arm}
ns$[1, \ldots, K] \leftarrow 0$ {Keep track of how many times each arm was played}
$t \leftarrow 0$ {Keep track of how many games have been played}
for i $= 0$ to K **do** {first play one game for each possible option}
  reward $\leftarrow$ play(arm)
  update means[arm] with reward
  ns[arm] $\leftarrow$ ns[arm] $+ 1$
  $t \leftarrow t + 1$
end for
while game is not over **do**
  for i $= 0$ to K **do**
    UB$[i]$ $\leftarrow$ means$[i] + \sqrt{\frac{2 * log(t)}{\text{ns}[x]}}$ {compute upper bound for each arm}
  end for
  pick arm with highest UB
  reward $\leftarrow$ play(arm)
  update means[arm] with reward
  ns[arm] $\leftarrow$ ns[arm] $+ 1$
  $t \leftarrow t + 1$
end while

In figure 3 we show the UCB1 algorithm implemented in POSEIDON. Again boats face the same 3 by 3 parent grid. The UCB1 algorithm goes through a longer initialization phase as agents often prefer to fish far from port in order to make sure that option is not as good as fishing near port. Eventually however agents do converge to fishing very near port.

This algorithm depends on 3 parameter: the exponential moving average weight α and the maximum and minimum rewards $\bar{x}, \underline{x}$ to normalize each observation between 0 and 1.

UCB1 has optimal logarithmic regret bounds(Auer, Cesa-bianchi, and Fischer 2002) compared to ϵ-greedy's linear regret. In all POSEIDON applications however UCB1 will perform worse. This shows that theoretical regret is not a good indicator of performance in agent-based models. Kuleshov and Precup (2014) also notes that UCB1 often takes a large number of trials before finding a high-reward slot machine. This long learning period, coupled with fishing spots whose quality changes over time, ruins the performance of the theoretically superior UCB1 in POSEIDON.

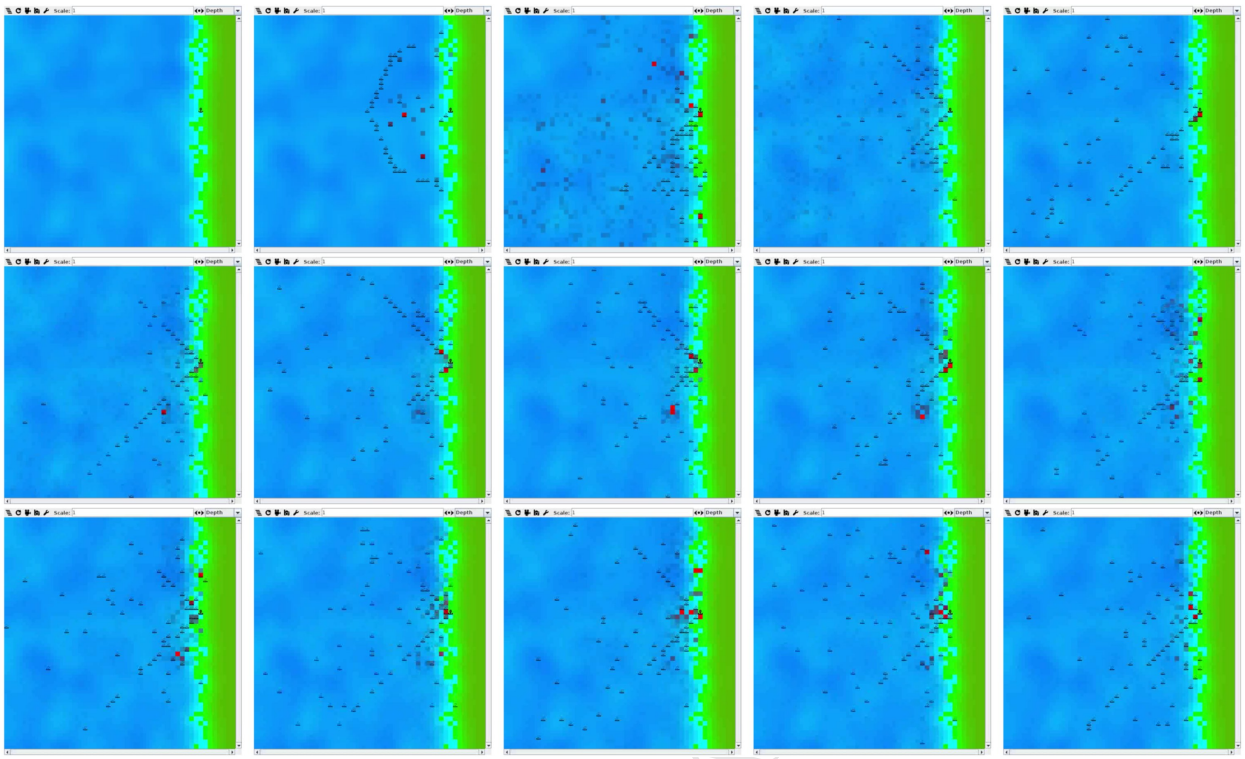


Figure 3: Animation showing a set of 100 fishers, each using a discretized UCB1 decision algorithm

2.2 Parallel exploration : imitating agents

Sampling a single slot machine achieves little when there are many to explore. We can improve by sampling many slot machines at each step. In agent-based models this is possible when agents share information about their rewards with others.

In fisheries the impossibility using experience alone to learn and the need to share exploration results with others is well known; as Morton (2005) states (as cited in Holland and Sutinen (2000)):

> The ocean is a very large, complex and rapidly changing environment. No individual fishermen acting alone could hope to acquire the experience necessary to establish the regularity or predictability required for its successful exploitation. This means that knowledge of the ocean must be acquired and exchanged by a large number of individuals.

There are however three problems with information sharing. First, while sharing knowledge might increase fishery-wide efficiency, these agents are competitors and they might be unwilling to help each other. Wilson himself mentions that "exchange [of] information creates conflicting incentives".
Second, imitation is easier when agents are homogeneous. When agents differ in skills or equipments information must be harmonized first. Third, too much imitation works poorly when the system is dynamic and punishes congestion. Too many boats exploiting the same fishing ground might deplete it.

Once we allow for information sharing it is relatively simple for agents to play large bandit problems. We present here four algorithms in decreasing order of required information. In our fishery model, algorithms that share too much perform worse.

2.2.1 Gravitational Search

The Gravitational Search Algorithm (Rashedi, Nezamabadi-pour, and Saryazdi 2009) assumes that agents move towards one another as if planets pulled by gravitational forces. The profits made by each agent represent their "planetary mass". Players earning more will pull the others towards them, while those earning less will move faster as if encumbered by less inertia. This framework assumes that each player knows the slot machine used and the profits made by everyone else at all times.

A possible implementation in pseudo-code that an agent would use follows in algorithm 4.

Algorithm 4 Gravitational search implementation

velocity $\leftarrow$ velocity$_0$
while game is not over **do**
  for player $= 0$ to N **do** {turn all other players' rewards into masses}
    mass[player] $\leftarrow \frac{\text{profits[player]}-\min(\text{profits})}{\max(\text{profits})-\min(\text{profits})}$ {peek at everyone's profits and normalize them}
  end for
  select the top Z masses
  acceleration $\leftarrow 0$
  for all i such that masses[i] is one of the top Z masses **do**
    acceleration $\leftarrow$ acceleration $+ G * \frac{\text{masses[here]}*\text{masses[i]}}{\text{distance(here,i))}}$ {selective gravitational pull}
  end for
  $\epsilon \sim U[0,1]$ {Draw random inertia}
  velocity $\leftarrow \epsilon \cdot$ velocity $+$ accelleration
  position $=$ position $+$ velocity
  play position
end while

This algorithm performs poorly in POSEIDON. Too much information is shared with too many competitors. Agents using this algorithm quickly converge to a single spot. This spot starts optimal but quickly deteriorates as too many agents exploit it. Slowly agents move towards neighbouring spots depleting each in turn. Many trips are wasted on spots that have just been depleted by competitors. A sample run is shown in figure 4.

The Gravitational Search Algorithm depends on 3 parameters. The parameter Z describing the number of most profitable competitors to imitate,the initial speed velocity$_0$ and the gravitational constant G.

2.2.2 Particle Swarm Optimisation

Particle Swarm Optimisation(Kennedy and Eberhart 2001) is a widely used "swarm intelligence" algorithm. While nowadays popular for engineering applications (due to its parallelizable nature) it was originally developed(Kennedy and Eberhart 1995) to simulate birds finding the "best" part of the cornfield to forage through imitation.

Like gravitational search, this method involves agents pulling each other. Unlike gravitational search, the amount of information available to agents is lower: everybody knows the profitability of only two other random agents. This rudimentary social network limits agents' imitation and prevents the race to a single spot the gravitational search algorithm produced. A second element of this algorithm is that each agent is pulled not just by his connections but also by the best memory it has. This means that an agent is less willing to follow others if he recently played a good slot machine. Algorithm 5 shows a pseudo-code representation.

This algorithm also performs poorly in POSEIDON. The algorithm is not designed for objective functions that change over time. Dynamic variants do exist(Blackwell and Branke 2006; Blackwell 2007) but they are hard to recast as individual decisions in an agent-based model. Following memories also lead to poor performance in a fishery scenario as it disincentivises exploration.

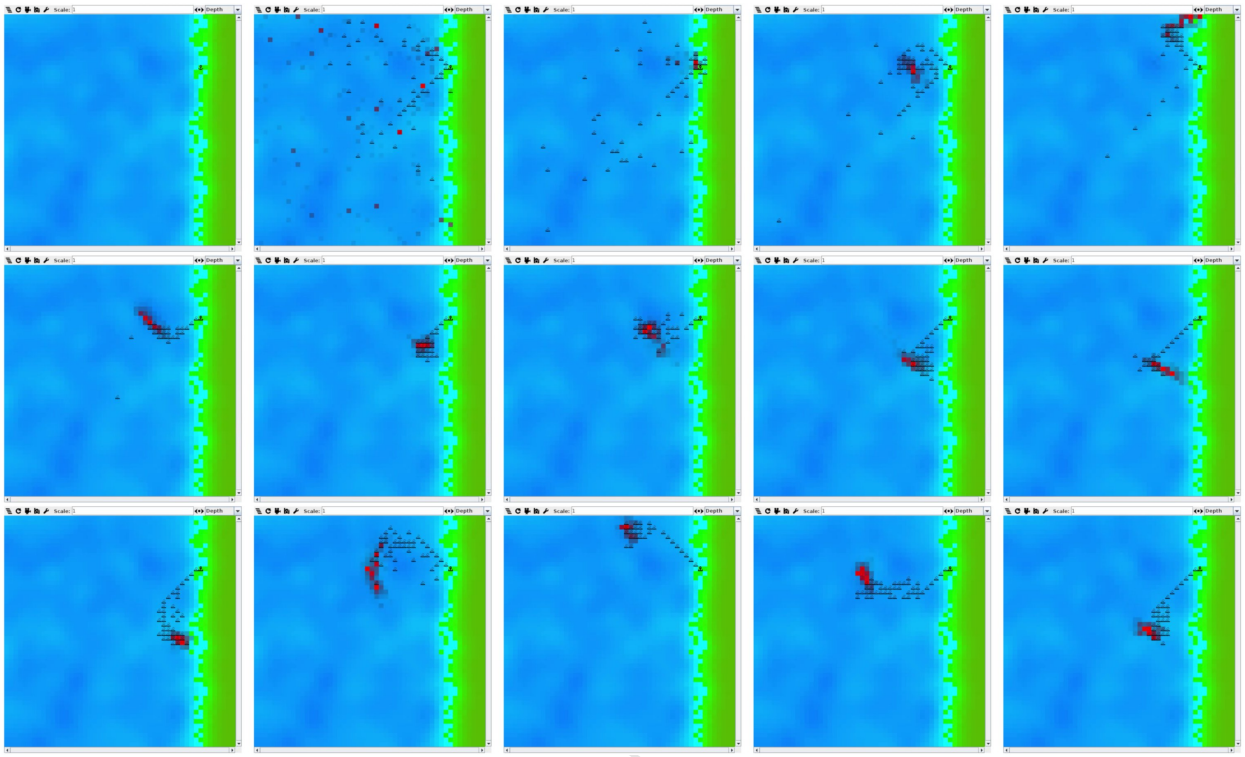


Figure 4: Animation showing a set of 100 fishers, each using gravitational search

Algorithm 5 Particle swarm optimization

velocity $\leftarrow 0$
while game is not over **do**
  position_f is the position of your most profitable friend
  position_m is the position of the slot machine you have observed being most profitable
  $\hat{\alpha} \sim U(0, \alpha)$
  $\hat{\beta} \sim U(0, \beta)$
  $\hat{\epsilon} \sim U(-\epsilon, \epsilon)$
  $\text{velocity} \leftarrow \text{velocity} \cdot \text{inertia} + \hat{\alpha} \cdot \text{distance}(\text{here}, \text{position}_f) + \hat{\beta} \cdot \text{distance}(\text{here}, \text{position}_m)$
  $\text{position} = \text{position} + \text{velocity}$
  play position
end while

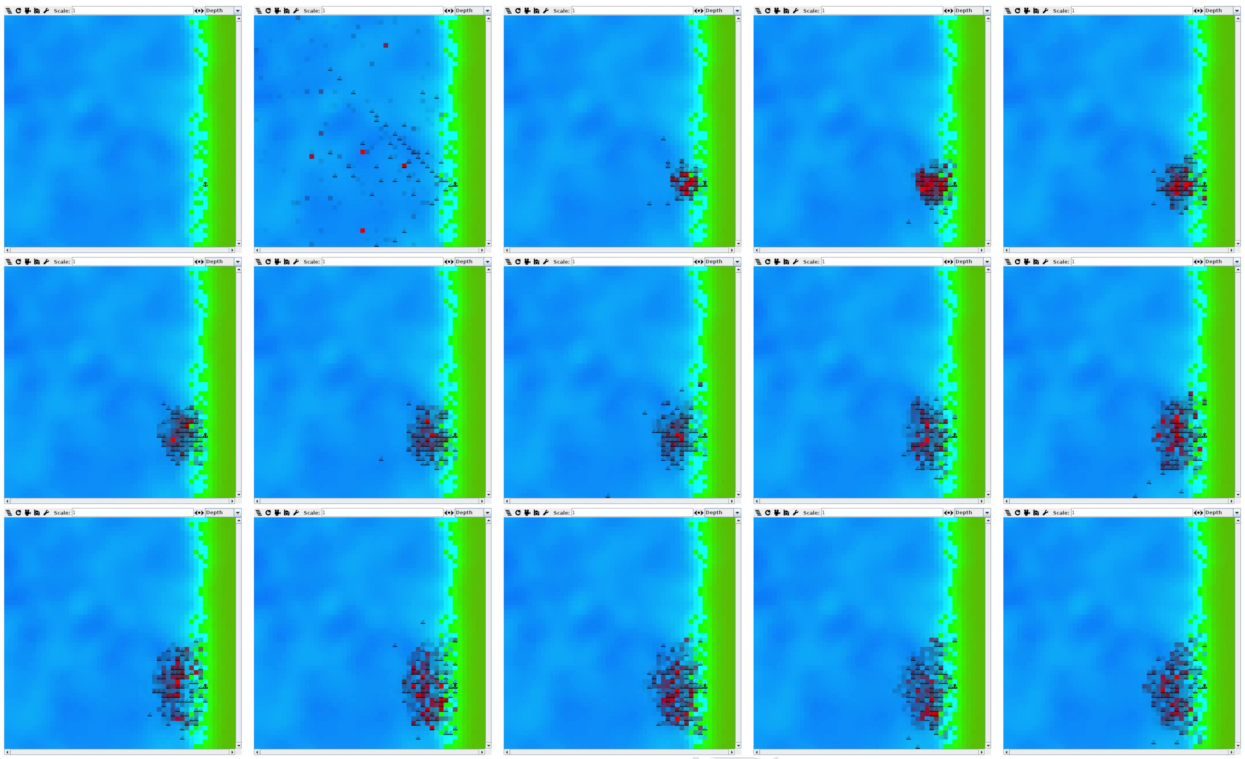


Figure 5: Animation showing a set of 100 fishers, each using a particle swarm algorithm

A sample run is shown in figure 5. Notice how the agents do not converge to the same spot as with gravitational search but rather fish near port and explore little beyond it.

This implementation of Particle Swarm Optimisation depends on 4 parameters. The inertia applied to speed `inertia`, the weights α and β applied to friend and memory observations plus the distribution of random velocity shock described in the pseudo-code as ϵ.

2.2.3 Explore-Exploit-Imitate

The explore-exploit-imitate algorithm is the default decision making in POSEIDON. It uses the same amount of information as the particle swarm optimization (a small social network) but abandons the idea of agents pulling each other. Agents directly copy their competitors' positions instead. It is effectively a beam hill-climber(Russell and Norvig 2010, vol. 140, chap. 3) with inertia. When imitating, the behaviour is similar to the "cartesian" agent in Cabral et al. (2010) and in Allen and McGlade (1986). When exploring, the algorithm acts like the ϵ-greedy algorithm described in section 2.1.1.

With probability ϵ the agent explores a new slot machine. If not exploring the agent plays the slot machine of her most profitable "friend" (imitation). If the agent is doing better than her friends, she plays the slot machine she played before (exploitation). Algorithm 6 describes explore-exploit-imitate in pseudocode.

Explore-exploit-imitate performs well in POSEIDON. While it uses information about competitors more aggressively (by copying them directly), it has a very small memory (keeping track of only the last spot visited) which is helpful in nudging the fisher towards exploring more.

A sample run of this algorithm in POSEIDON is shown in figure 6. The figure also shows the biomass left at each location to prove that the agents start by consuming fish near port and then progressively move further out in an efficient "fishing front".

Algorithm 6 Explore-Exploit-Imitate

while game is not over **do**
  if random $< \epsilon$ **then** {with probability ϵ, explore}
    $\hat{\delta} \sim U[-\delta, \delta]$
    position $\leftarrow$ position $+ \hat{\delta}$ {explore by shocking your position by δ}
  else
    profits$_f$ are the profits made by your most profitable friend
    profits$_{\text{me}}$ are the profits just made by this agent
    if profits$_f$ $>$ profits$_{\text{me}}$ **then**
      copy friend location
    else
      stay at current location
    end if
  end if
end while

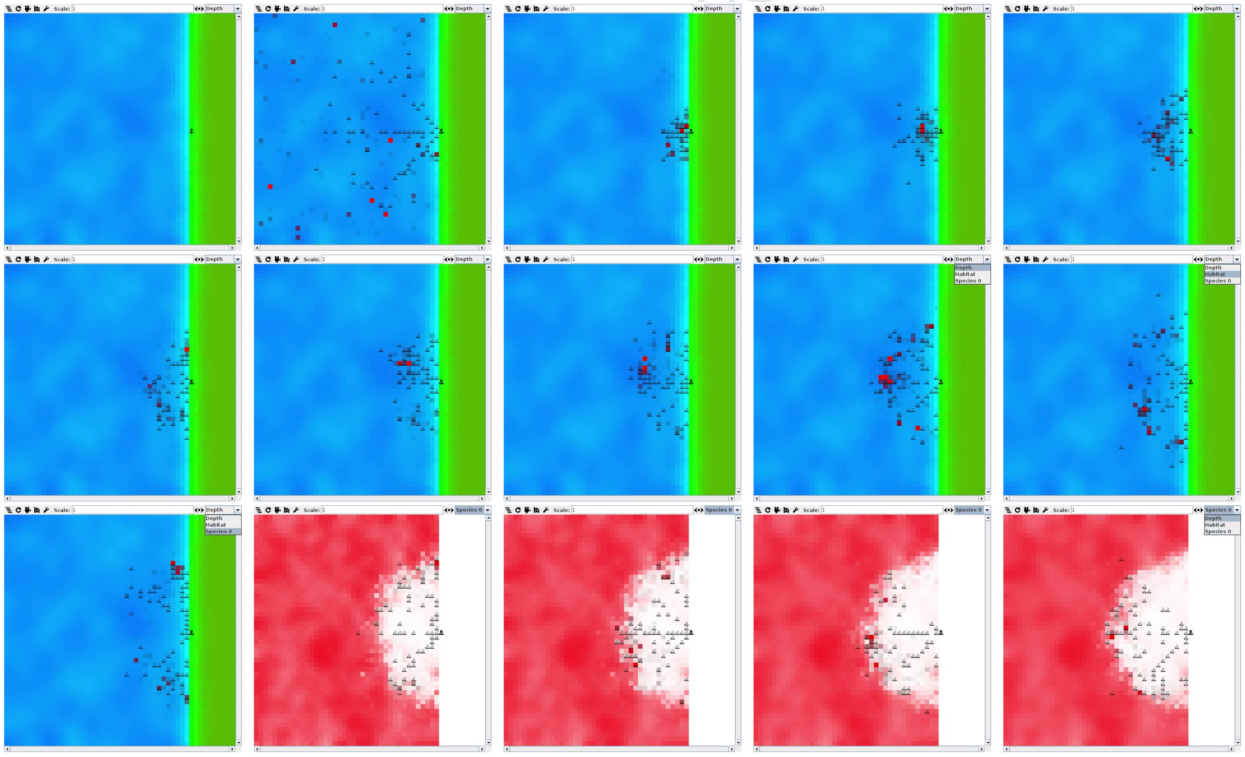


Figure 6: Animation showing a set of 100 fishers, each using explore-exploit-imitate. Latter frames show the biomass remaining in each cell (the lighter the color the more depleted the cell is)

Explore-exploit-imitate depends on 2 parameters. The exploration rate ϵ and the exploration range δ.

2.2.4 Social annealing

This algorithm uses the least amount of information. Agents know only the average fishery-wide reward earned the previous step. Whenever an agent is doing worse than average, she explores. Otherwise she exploits her current slot machine. Functionally this is an implementation of the reservation utility introduced by Caplin, Dean, and Martin (2011) to simulate satisficing agents where the threshold is the average profit level of the fishery. To the best of our knowledge this algorithm was first implemented in fisheries by Beecham and Engelhard (2007); a multiple threshold version of it appears in Cenek and Franklin (2017).
Algorithm 7 shows a pseudocode implementation of the social annealing agent.

Algorithm 7 Social Annealing

```
while game is not over do
  profits_a is the current average profits in the fishery
  if profits_a > profits_me then {explore if you are making less than average}
    δ̂ ~ U[−δ, δ]
    position ← position + δ̂ {explore by shocking your position by δ}
  else
    stay at current location
  end if
end while
```

In POSEIDON social annealing performs well for certain scenarios but is sometimes hampered by the lack of direct imitation. The reason it works well in most cases, in fact outperforming other algorithms, is that rarely too many boats fish the same spot. It performs poorly however when the high profitability areas are rare in which case imitation would have helped the agent discover the right slot machines faster than blind exploration.

Figure 7 shows a sample POSEIDON run where fishers use social annealing. The aggregate dynamic generated by the individual fishers is similar to the one generated by explore-exploit-imitate.

Social annealing depends on a single parameter: the exploration range δ.

2.3 Discovering structures: heatmap agents

We defined "structure" in a bandit problem as the interdependency between slot machines' rewards. In section 2.1 we sometimes assumed a discretized reward structure. In this section we let agents discover structure on their own. Statistically this is a regression problem and it generates a "heatmap" of expected rewards across the slot machine space.

The key implementation issue of regressions in agent-based models is that observations and predictions are interleaved. Agents play one slot machine (observation), draw a new reward heatmap (prediction) and use it to decide what to play next (observation). This behaviour constrains the kind of algorithms agents can use: batch regressions (like ordinary least squares or Kriging) are too slow.

We focus on regression methods that are iterative (sometimes called on-line or recursive regressions). We catalogue the computational cost of each algorithm by their time complexity when observing and when predicting. In general there is a trade-off between smoothness of the heatmap and the computational cost of modelling it.

Another reason to keep regression methods iterative is the need for forgetting. Newer data needs to be weighted more and iterative methods are a natural way of doing so. Forgetting is crucial in POSEIDON as fishers deplete the areas they tow and need to feature this in their prediction.

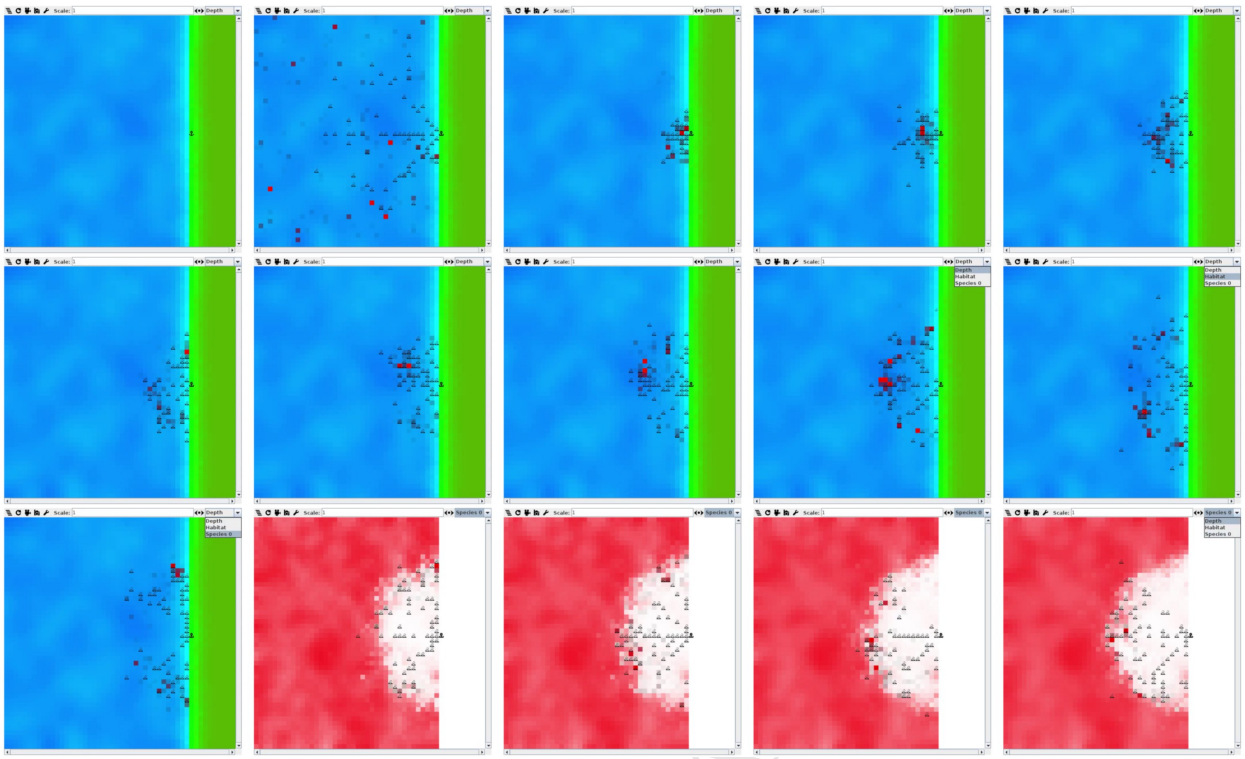


Figure 7: Animation showing a set of 100 fishers, each using social annealing.

The key assumption made by all the following algorithms is that the closer two slot machines are the closer their rewards. This way observing one slot machine will reveal the quality of all other slot machines "close" to it. This is the geographical version of the more general "contextual bandit" problem (Li et al. 2010; Agarwal et al. 2014).

In practice, geographical problems often involve close options that are related to one another. This is the "first law of geography" (Tobler 1970). Fisheries discrete choice models similarly observe a "spatial autocorrelation ripple effect" where displaced effort distributes itself along neighbouring cells (Wilen et al. 2002; Hynes et al. 2016). Intuitively this assumed autocorrelation can be misleading when fish aggregate in small, fast-moving shoals yet we will show in section 3.4 that this assumption will perform adequately even under such circumstances.

While distance in POSEIDON is geographical, we can extend its application to other domains by focusing on a factored representation of the slot machine space. For each slot machine we extract F features: numerical values associated with the slot machine. We then assume that the distance between slot machines is the sum of the distance of their features. Given slot machine a and b and their features $f_1(\cdot), f_2(\cdot), \ldots, f_F(\cdot)$ their distance is:

$$d(a,b) = \sum_{i}^{F} d(f_i(a), f_i(b)) \tag{3}$$

In POSEIDON these features are the coordinates of the fishing spot, but they could include habitat, temperature, etc.

Features are powerful but they are an inherent weakness of the approach. Unlike discretization and imitation, features are model-specific. These algorithms are only as good as the features we choose. Unlike the previous algorithms, then, heatmap agents are not model free.

Heatmaps are decision support tools, not decision algorithms per se. There needs to be a control rule that

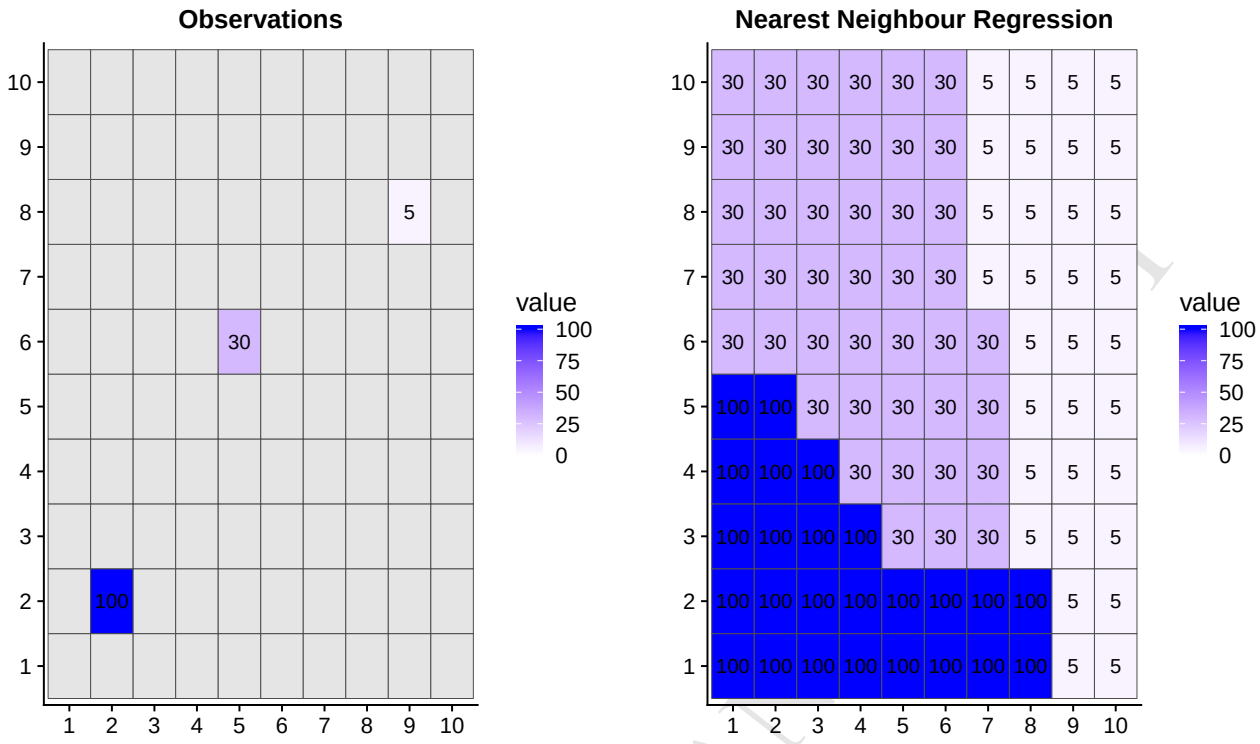


Figure 8: On the right the location and value of three observations. On the left the heatmap generated by a nearest neighbour regression using two features, the x and y of each cell, and bandwidths $h_x = h_y = 1$

given the heatmap chooses which slot machine to play next. In the Bayesian literature this is defined as "acquisition function" (Snoek, Larochelle, and Adams 2012). In POSEIDON the agents always fish at the peak of the heatmap (with an error of ± 2 cells). A better acquisition function would balance exploitation with exploration away from the peaks. We do not explore this further here although it is a key component.

Imitation is also straightforward with these algorithms. An agent can input the competitors' trips as an additional observation for their own geographical regression. A more nuanced approach to this subject is possible: agents could discount competitors' observations depending on its quality (the competitor might be lying, or one might be unsure about where their competitor went fishing). Here however we ignore this complication and assume each agent keeps track of two of its competitors and monitor their trips perfectly.

2.3.1 Nearest Neighbour

Nearest neighbour regressions build the simplest heatmaps. The regression predicts at any cell the closest past observation. It is easy to implement and fast at prediction but it generates discontinuous maps.

After choosing which feature to use for the distance metrics we must also choose their relative importance. This is done by choosing a bandwidth h_i for each feature f_i such that the final distance function between slot machines a and b is:

$$d(a,b) = \sum_i \frac{|f_i(a) - f_i(b)|}{h_i} \tag{4}$$

Figure 8 shows an example nearest neighbour regression.

Nearest neighbour regressions can be implemented quickly by using k-d trees (this implementation is quite common as for example in Pedregosa et al. 2011). The average time complexity for both search and insertion

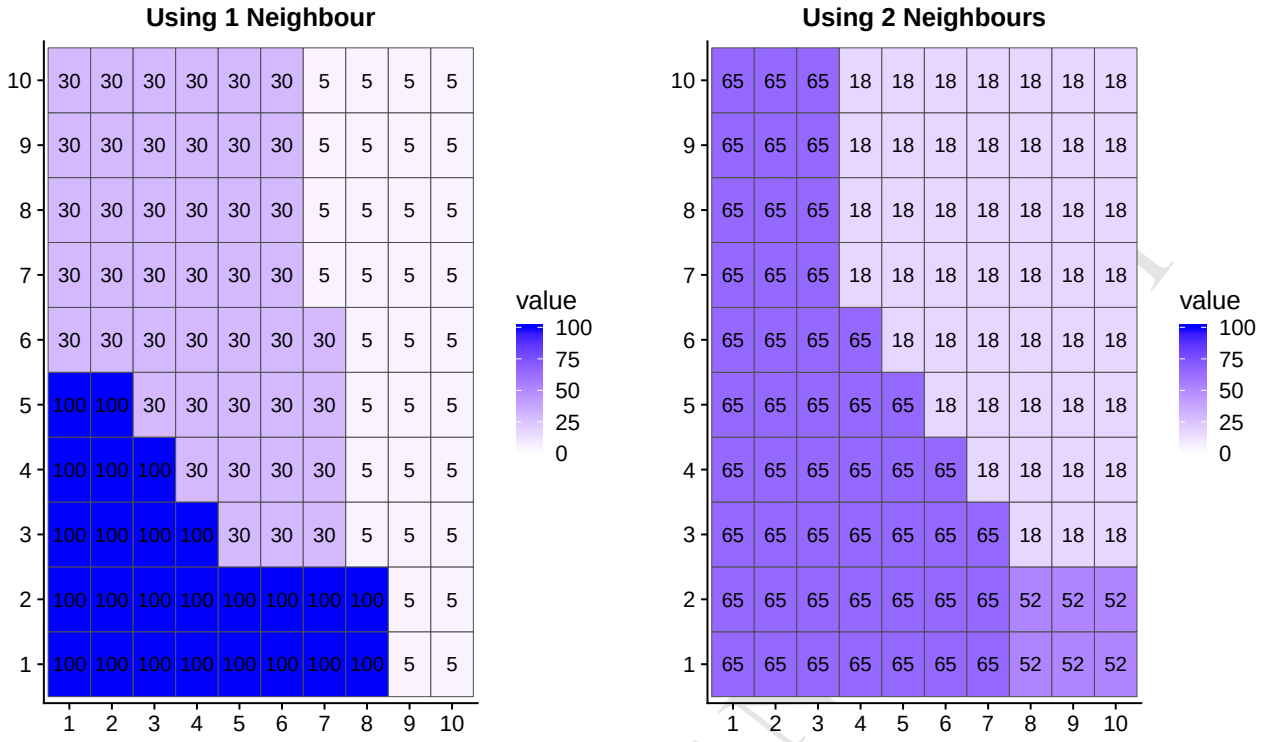


Figure 9: The difference in heatmaps generated depending on whether we use the closest neighbour the average of the closest two. The features used are two, the x and y of each cell, and their bandwidths are $h_x = h_y = 1$

is $O(\log N)$ where N is the number of observations. So implemented the nearest-neighbour's performance degrades as memory grows much less than other non-parametric methods.

We can smooth heatmaps by using multiple neighbours for each prediction and taking their average as shown in figure 9. There are two drawbacks to this. First, there is a loss in performance as tree searches become more expensive. Second, if data is sparse the neighbours will be far apart but weighed equally.

Figure 10 provides an example of how the nearest neighbour regression both drives the agent fishing decisions and updates itself in POSEIDON.
On the left a POSEIDON simulation with 100 fishers is running. The behaviour is the classic fishing front where agents first fish near the port and then progressively move further away as biomass is depleted. One of the agent is highlighted in blue, her heatmap is shown on the right. This heatmap is generated by a nearest neighbour regression trying to predict the profitability of each cell. Blue means high profits, red means negative profits and white means little or no profits.
The heatmap starts at a default negative value (hence the red initial state) forcing the agent to act at random. As the fisher completes trips the heatmap quickly turns blue in the area near port as fishing there is more profitable. The agent responds to this information by fishing around the blue area. The area close to port turns whiter as fish there is consumed eventually turning red as it depletes entirely.

The nearest neighbour regression used here has 4 features: x, y, time of observation (in hours) and the Manhattan distance from port. The "angular" contours in figure 10 are caused by using Manhattan distance rather than Euclidean.

This algorithm has a vector of bandwidth parameters $h = \begin{bmatrix} h_1 & h_2 & \dots & h_F \end{bmatrix}$ where F is the number of features (4, in this application). Also the algorithm depends on the `neighbours` parameter describing how many observations to weigh together when making a prediction.

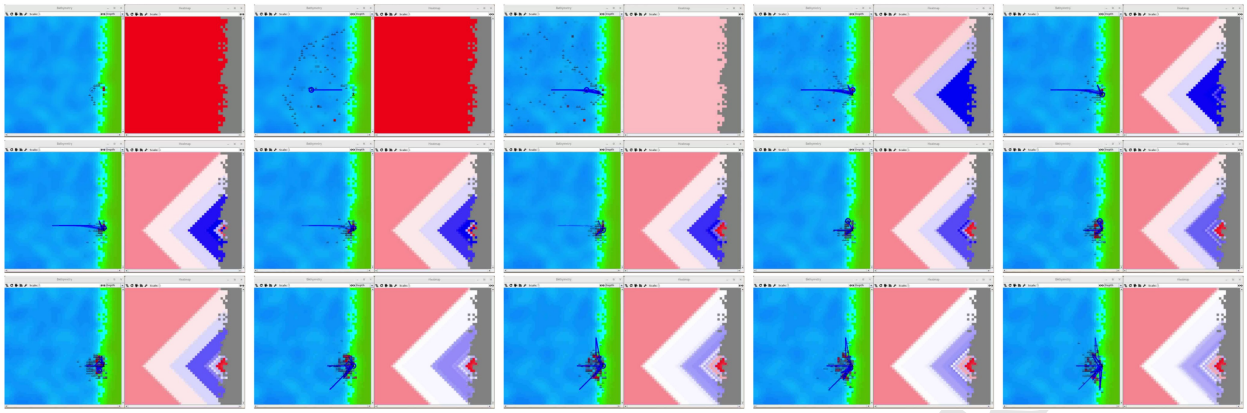


Figure 10: Animation tracking one fisher using nearest neighbor regression; in each frame on the left we show the movement of the fisher and on the right we show its predicted profitability for each cell of the map

2.3.2 Kernel Regression

Kernel regression (Nadaraya 1964; Watson 1964) draws heatmaps by averaging all observations in memory, weighing closest memories more.

Given a series of past slot machines played $x_1, x_2, \ldots, x_n$ and the rewards observed $y_1, y_2, \ldots, y_n$ we predict the rewards at slot machine x^* as:

$$y^* = \frac{\sum_i K(x_i, x^*) y_i}{\sum_i K(x_i, x^*)} \tag{5}$$

The kernel function $K(\cdot, \cdot)$ is a metric of similarity: the higher the kernel the higher the weight.

Here too we find useful to first split each observation x into a series of features: $f_1, f_2, \ldots, f_z$, compute the kernel for each feature separately and then multiply them together, so that the kernel between slot machines a and b is:

$$K(a, b) = \prod_{k=1}^{F} K(f_k(a), f_k(b)) \tag{6}$$

While there are many kernel functions, the choice of functional form is usually not important (Shalizi 2013, 44). Here, we implemented both the Gaussian(RBF) kernel:

$$K(a, b) = e^{-\frac{|a-b|^2}{2\sigma^2}} \tag{7}$$

and the Epanechnikov(EPA) kernel:

$$K(a, b) = \frac{3}{4}\left(1 - \frac{|a-b|^2}{h}\right) \tag{8}$$

The Gaussian kernel is more popular but the Epanechnikov kernel computes faster. While the functional form may not matter, the bandwidths (σ^2 or h) do as they encode a feature's relative importance. Kernel regressions produce smoother heatmaps than nearest neighbour, as figure 11 shows.

Unlike the nearest-neighbour regression, kernel regression scales poorly. While inserting new data is instantaneous, $O(1)$, prediction requires us to loop through all known observations to compute the weighted average. Prediction complexity is approximately $O(NF)$ where F is the number of features and N is the number of observations. This means that kernel regression performance degrades rapidly and we can only use it in agent-based models by restricting memories. Agents in our model keep track of only their latest $N = 100$ observations (rolling window forgetting).

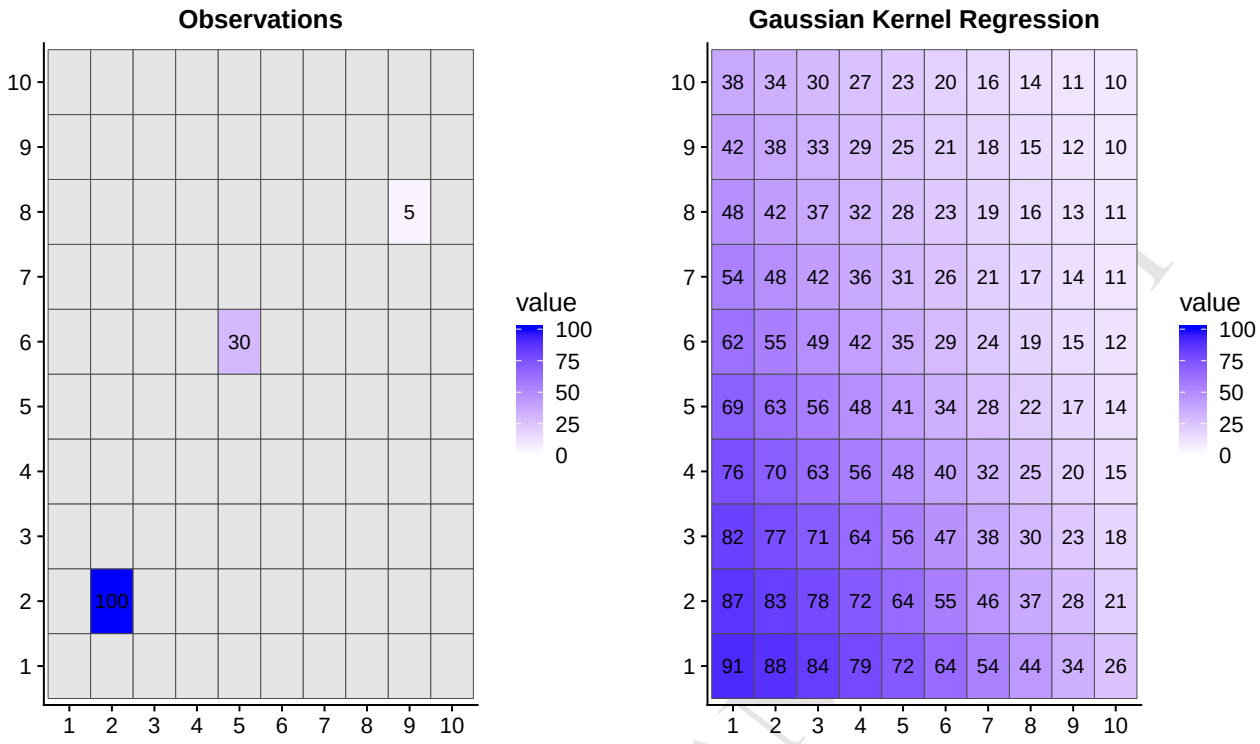


Figure 11: On the right the location and value of three observations. On the left the heatmap generated by a RBF kernel regression using two features, the x and y of each cell, and using bandwidths $\sigma_x^2 = \sigma_y^2 = 20$

In figure 12 we show the kernel regression in action using an Epanechnikov kernel. On the left a POSEIDON simulation with 100 fishers is running. One of the agent is highlighted in blue, her heatmap is shown on the right. Blue means high profits, red means negative profits and white means little or no profits. Grey means no prediction. The method refuses to predict for areas far away from all observations (where the kernel is 0) and as such some of the heatmap cells remain grey. Again as high profit areas near port are discovered and exploited they turn paler until the agents move to other locations.

This kernel regression uses 4 features: the cell's x and y grid coordinates, the distance from port and the square root of time. It uses the Epanechnikov kernel.

This algorithm depends, like the nearest neighbour, on a vector of bandwidths $h = \begin{bmatrix} h_1 & h_2 & \dots & h_F \end{bmatrix}$ where F is the number of features but also a parameter N describing the size of the memory.

2.3.3 Kernel transduction

In POSEIDON, agents predict more often then they observe. They make one observation per trip but they need to predict the profits of many candidate coordinates to choose where to go next. If the cells' features do not change over time, we can implement a recursive version of the kernel regression (Krzyzak 1992).

For every slot machine x we want to keep track of, we initialize two variables $g_0 = m_0 = 0$. Then at each new observed reward y_n at slot machine x_n we update the two variables as follows:

$$\begin{aligned} g_n &= g_{n-1} + K(x, x_n) \\ m_n &= m_{n-1} + \frac{(y_n - m_{n-1})\, K(x, x_n)}{g_n} \end{aligned} \tag{9}$$

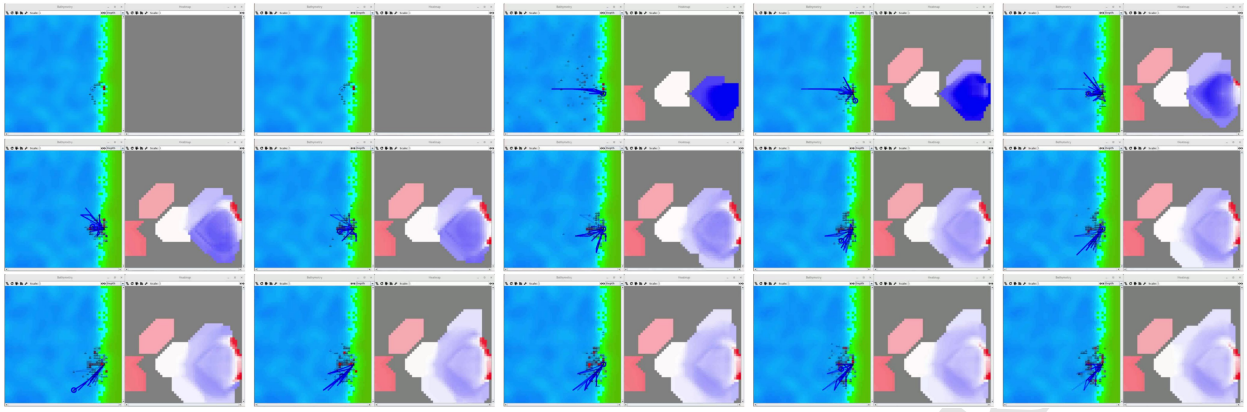


Figure 12: Animation tracking one fisher using kernel regression; in each frame on the left we show the movement of the fisher and on the right we show its predicted profitability for each cell of the map

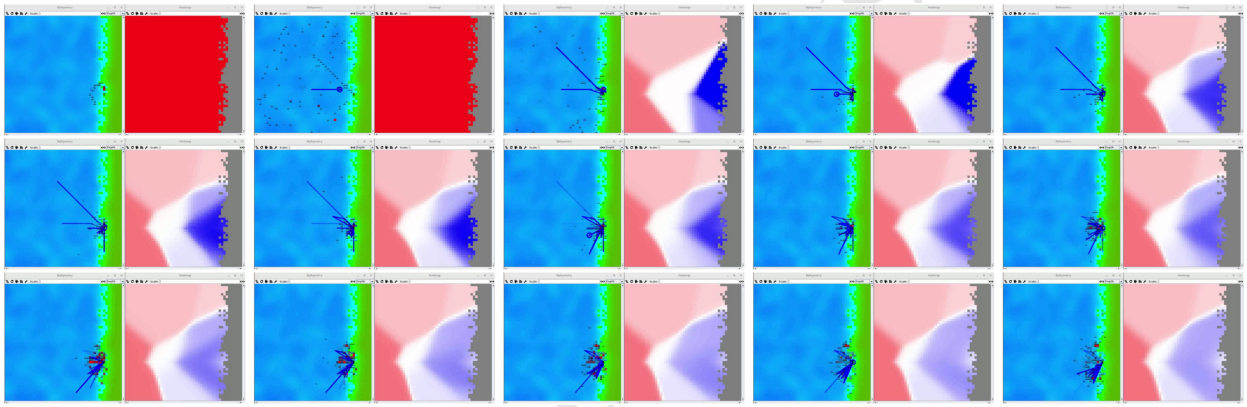


Figure 13: Animation tracking one fisher using kernel transduction; in each frame on the left we show the movement of the fisher and on the right we show its predicted profitability for each cell of the map

Our prediction for slot machine x is the value m_n. To give more weight to more recent observations we add a forgetting factor $0 < \lambda < 1$ as follows:

$$g_n = \lambda g_{n-1} + K(x, x_n) \tag{10}$$

This formulation is faster at predicting but slower at observing than the standard kernel regression. Recursive kernel's prediction is $O(1)$, but observations take $O(KF)$ operations where K is the number of cells we need to keep track of and F is the number of features. Depending on the use case this formulation could make models faster or slower.

Not having to keep track of observations may make this predictor run faster but it makes tuning (see appendix) harder. It has however the advantage of forgetting smoothly compared to the standard kernel regression.

In figure 13 we show the kernel transduction in action using the Gaussian kernel. On the left a POSEIDON simulation with 100 fishers is running. One of the agent is highlighted in blue, her heatmap is shown on the right. Blue means high profits, red means negative profits and white means little or no profits. Unlike the kernel regression here by construction we make a prediction for each cell even if no close observation is present. Again as high profit areas near port are discovered and exploited they turn paler until the agents move to other locations.

This algorithm uses 3 features: the cell's x and y grid coordinates, the distance from port. Kernel transduction depends on a bandwidth vector h of size F (3 in this application) as well as the forgetting factor λ.

2.3.4 Kalman filter

The Kalman filter infers and tracks the state of a hidden Markov chain model given a series of observations (Minka 1999). The strength of Kalman filters is their flexibility as model knowledge can be included in the transition and emission matrices of the Markov chain.

However a more intuitive explanation of one-dimensional Kalman filters is to cast them as Gaussian Bayesian estimators (Faragher 2012). Assume that for each slot machine we have a Gaussian prior about its reward x, that is $x \sim N(\hat{x}, \sigma_x^2)$; further assume that each observed reward z is also Gaussian distributed $\sim N(z, \sigma_z^2)$. Bayes rule updates our belief every time a new observation z comes in. Because the posterior is just the product of two Gaussians (and a normalization), the posterior is itself Gaussian. The Kalman filter is the mathematical tool that performs this belief updating over time.

In POSEIDON we instantiate a 1D Kalman filter for each cell of the map. This requires us to specify an initial uncertainty σ_x^2. When updating the belief at slot machine a given an observation at slot machine b we define the measurement noise as

$$\sigma_z^2 = \alpha + e^{\frac{d(a,b)^2}{2\beta}} \tag{11}$$

where the distance function $d(a, b)$ is the Manhattan distance. The larger σ_z^2 the less the observation affects the prior. We also add a daily drift ϵ to the prior such that a lack of observations increases our uncertainty over time, simulating forgetting:

$$\sigma_x^2 = \sigma_x^2 + \epsilon \tag{12}$$

Figure 14 simulates a Kalman generated heatmap.

Figure 15 shows the 1D Kalman filter in action. On the left a POSEIDON simulation with 100 fishers is running. One of the agent is highlighted in blue, her heatmap is shown on the right. Blue means high profits, red means negative profits and white means little or no profits. Initially all cells are initialized with a random prediction between -100 and 100 and an initial uncertainty $\sigma_x^2 = 100^2$. As observations come in the posteriors start to correlate geographically. The "diamond" shaped contours are due to our choice of using Manhattan distance.

Unlike the kernel regression here by construction we make a prediction for each cell even if no close observation is present. Again as high profit areas near port are discovered and exploited they turn paler until the agents move to other locations.

The major advantage of Kalman filters is their flexibility to add model knowledge and dimensions. This flexibility however comes with a large number of parameters to tune. The computational efficiency is similar to the kernel transduction as you need to update all filters at each new observation but prediction is immediate. Adding dimensions is expensive however as Kalman filters require a matrix inversion so that their time complexity at observation is approximately $O(KF^3)$ where F is th number of features.

Another advantage of Kalman filters is that they generate not just point predictions but entire confidence intervals (as long as our Gaussian assumptions hold) which allows for risk seeking and avoidance behaviour to be implemented easily. We can do this by having agents targeting not the highest predicted reward x but the highest confidence bound $x + c\sigma$ where c is positive for risk seeking and negative for risk avoidance. This way we use the optimism principle behind the UCB1 algorithm, described in section 2.1.3, but informed by Kalman posteriors.

Kalman filters, unlike kernel and nearest-neighbour regressions, do not depend on a vector of bandwidths. As implemented here each Kalman filter depends on 6 parameters. The drift variable ϵ, the initial uncertainty σ_x^2, the evidence uncertainty σ_z^2, the evidence distance penalty β and the distribution needed to generate the initial guessed states x before any evidence is produced. We also need to define the risk parameter c.

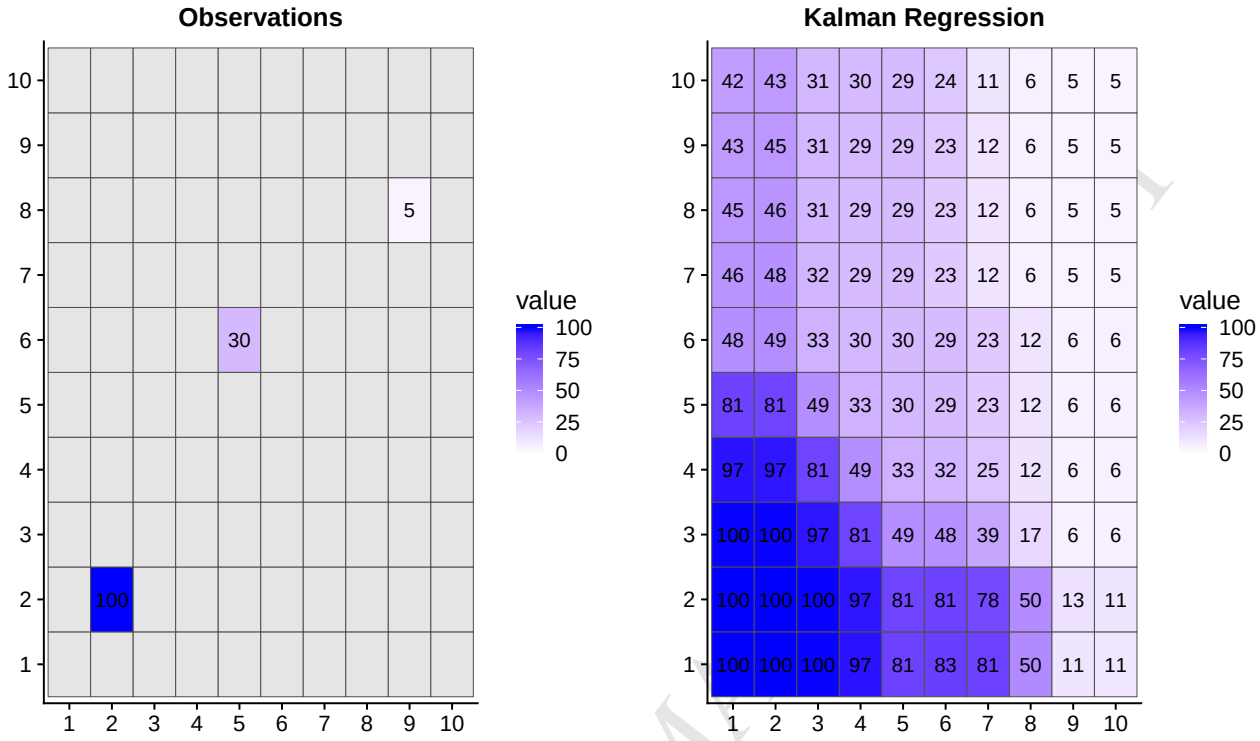


Figure 14: On the left the 3 observations that the Kalman filter transforms into the heatmap on the right. The parameters are $\sigma_x = 50$, $\alpha = .1$ and $\beta = 10$

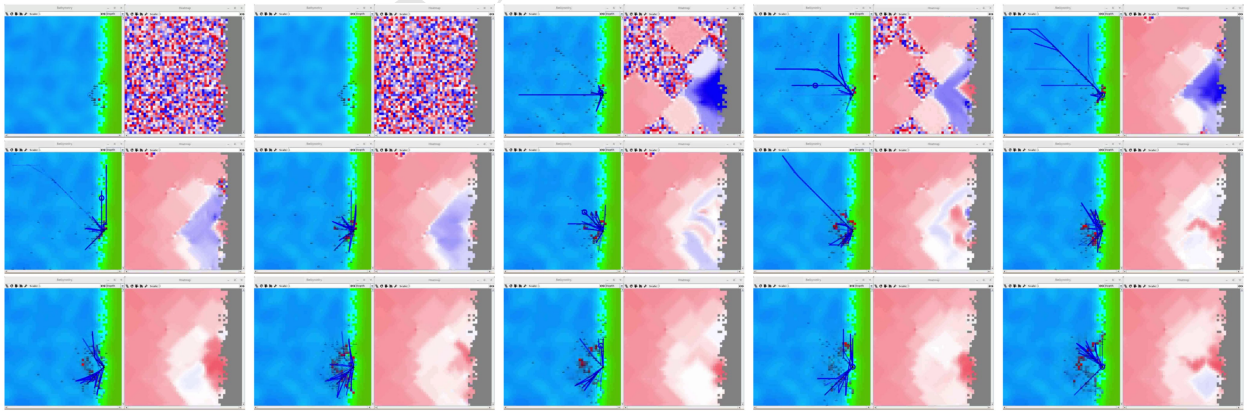


Figure 15: Animation tracking one fisher using kalman regression; in each frame on the left we show the movement of the fisher and on the right we show its predicted profitability for each cell of the map

2.3.5 Geographically weighted regression

Geographically weighted regression(Brunsdon, Fotheringham, and Charlton 1998) is a common method to draw inference over non-stationary spatial data. Its main advantage for our purposes is that it can be easily turned into a recursive estimator whose performance is compatible with the needs of agent-based models.

Geographically weighted regressions are a spatial application of the locally weighted linear regression estimator (Hastie, Tibshirani, and Friedman 2009; Cleveland 1979). We fit a weighted linear regression in each cell where the weight of each observation is the kernel distance between the cell we are predicting in and the where the observation occurred.

More precisely, describing each observation and each cell by a series of features $f_1, f_2, \ldots, f_F$, we predict the value of cell a as:

$$y_a = \beta_{0,a} + \sum_{i=1}^{F} \beta_{a,i} f_i(a) \tag{13}$$

where the vector of coefficients for this cell is obtained by weighted least squares

$$\beta_a = (X^T W_a X)^{-1} X^T W_a y \tag{14}$$

where the weight vector is the vector of Gaussian kernel distances between cell a and the location x_i where observation i took place:

$$W_a = \begin{bmatrix} e^{\frac{d(a,x_1)^2}{2h}} \\ e^{\frac{d(a,x_2)^2}{2h}} \\ \vdots \\ e^{\frac{d(a,x_N)^2}{2h}} \end{bmatrix} \tag{15}$$

and the distance function $d(\cdot,\cdot)$ is the Manhattan distance between cells.

Linear regressions are commonly computed in batch (that is, having all the observations in the design matrix X we compute the final β). The recursive least squares filter (in essence a simplified Kalman filter) is a way to compute regressions one observation at the time (Rogers 1996). We implement one filter for each cell of the map to compute independently its β vector. Our implementation includes a forgetting factor $0 < \lambda <= 1$ to weigh new observations more. Figure 16 shows how the geographical weighted regression generates heatmaps.

Figure 17 shows the geographical weighted regression in action. On the left a POSEIDON simulation with 100 fishers is running. One of the agent is highlighted in blue, her heatmap is shown on the right. Blue means high profits, red means negative profits and white means little or no profits. Initially all cells are initialized with a random prediction between -100 and 100 and an initial uncertainty $\sigma_x^2 = 100^2$. This regression is intercept only and has forgetting factor $\lambda = .96$ and a distance bandwidth $h = 35$.

The main advantage of geographically weighted regressions is that we do not need to define the importance of each feature a priori as in the nearest neighbour or kernel regression, since the β vector is discovered automatically. Moreover this method also produces a full confidence interval for each point prediction. Like with the Kalman filter, prediction is $O(1)$ but observing new data is expensive $O(KF^3)$.

This implementation of geographically weighted regression depends on 4 parameters. The initial uncertainty σ_x^2, the distance bandwidth h, the forgetting factor λ and the random distribution that initializes the predictions at each cell.

2.3.6 Ad hoc methods

It is often the case that social simulations incorporate interviews and ideas from the people we are trying to simulate. Often from these interviews we can infer how experts predict and extrapolate. Keeping in mind the risk of over-fitting, these methods tend to be domain specific but computationally efficient. They could be used in place of the more open-ended statistical methods suggested above. These methods are ad-hoc and do

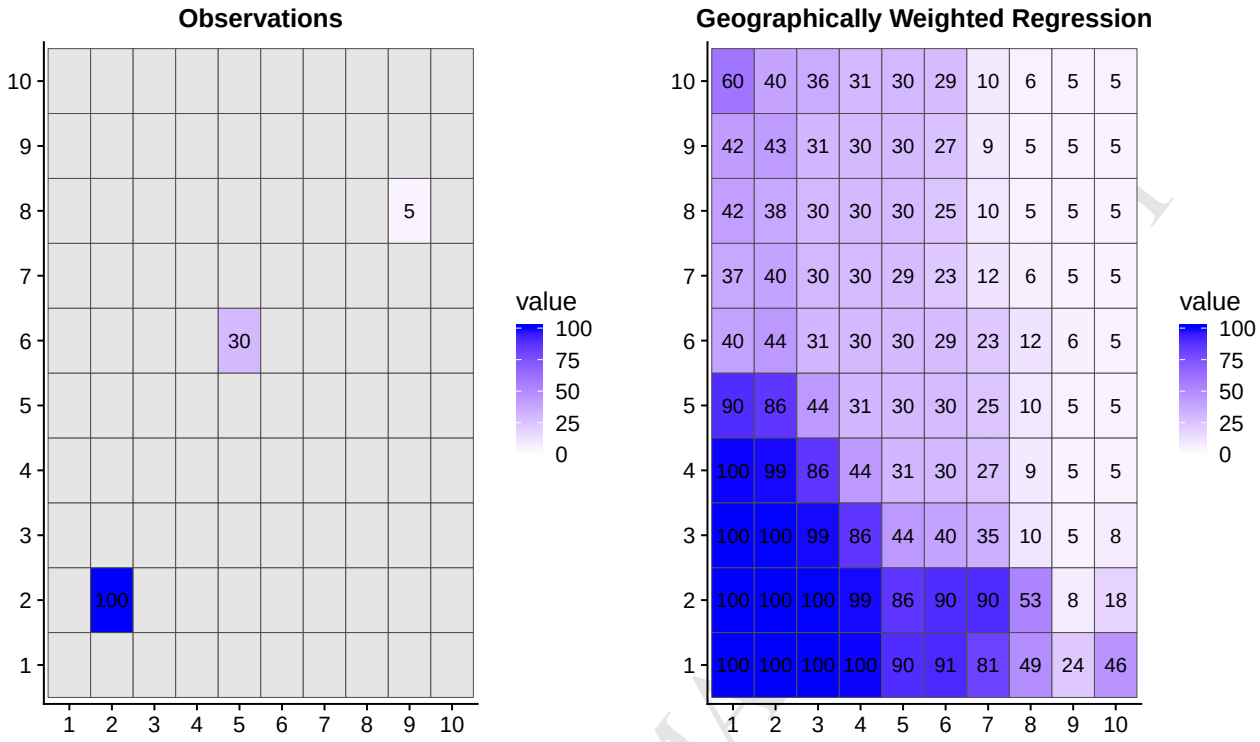


Figure 16: On the left the 3 observations we input in the geographical weighted regression. On the right the heatmap generated by an intercept-only geographical weighted regression, distance bandwidth $h = 5$ and no forgetting ($\lambda = 1$)

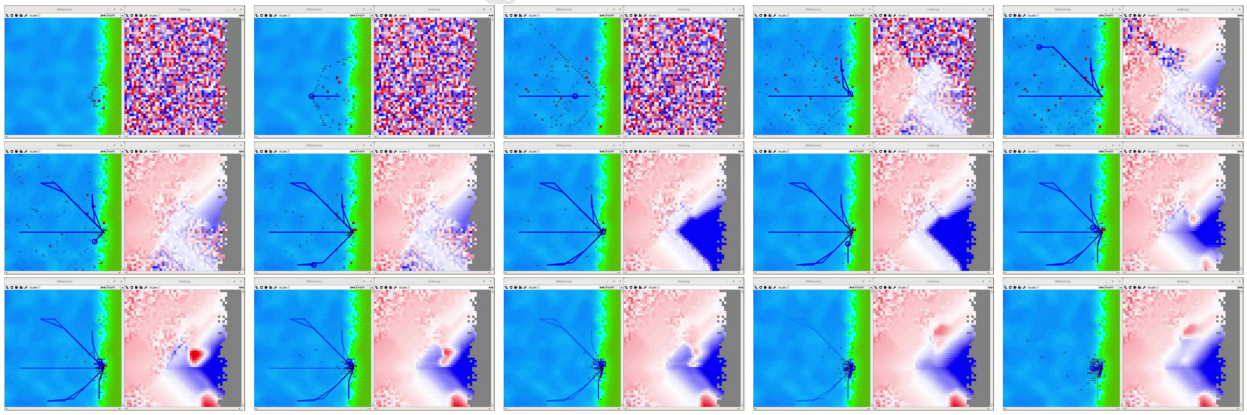


Figure 17: Animation tracking one fisher using geographically weighted regression; in each frame on the left we show the movement of the fisher and on the right we show its predicted profitability for each cell of the map

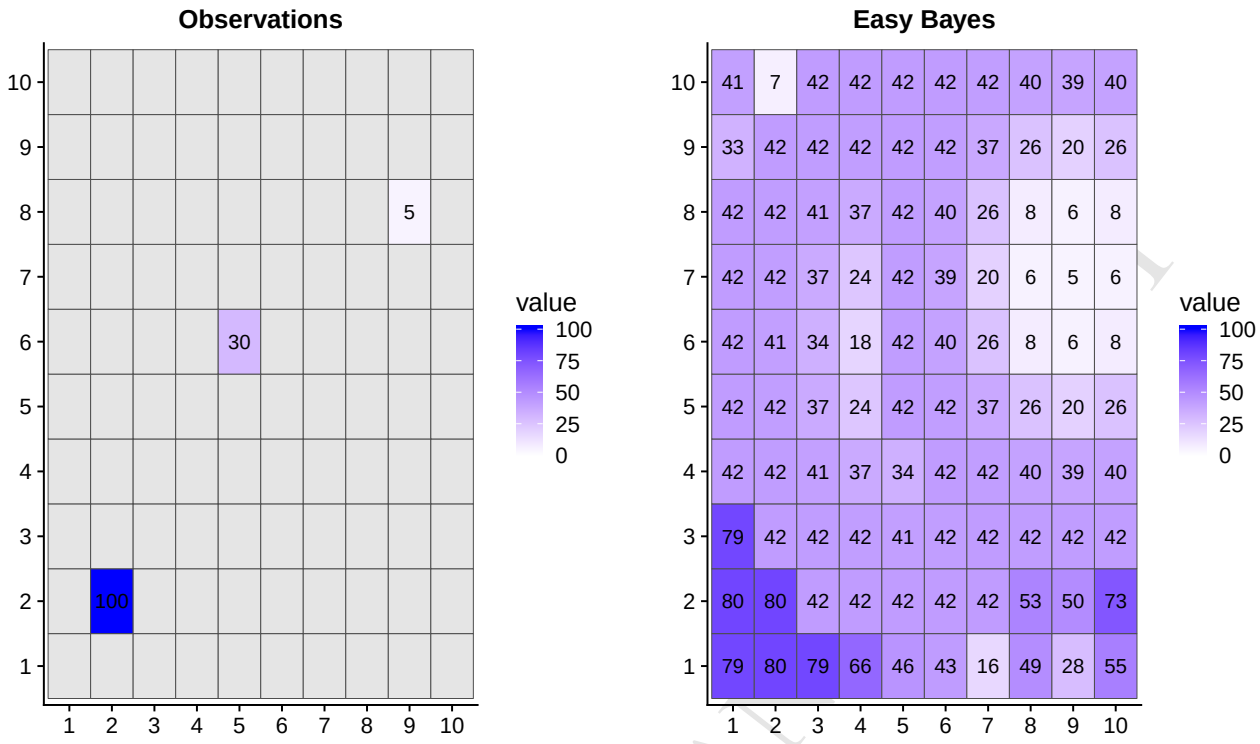


Figure 18: A predicted heatmap generated by the Ad-Hoc classifier

not belong to this paper but in some cases they can be generalized and used broadly. Here we present one example.

Simplify the problem from trying to predict the precise reward of each slot machine to guessing whether a slot machine is "good" or "bad" according to some pre-defined threshold. Statistically we are moving from regression to classification which opens entire new vistas of available algorithms. Here however we will use a simpler ad-hoc classifier.

Assume that "good" slot machine returns a random reward $\sim N(\mu_g, \sigma^2)$ while a "bad" slot machine returns $\sim N(\mu_b, \sigma^2)$ where $\mu_g > \mu_b$. For a slot machine a the agent maintains a subjective probability p_a that a is a good. Whenever the agent plays a again and receives the reward r and updates the probability through Bayes:

$$\Pr(\text{good}|r) = \frac{\Pr(\text{r}|\text{good})\Pr(\text{good})}{\Pr(\text{r}|\text{good})\Pr(\text{good}) + \Pr(\text{r}|\text{bad})\Pr(\text{bad})} \tag{16}$$

Where Pr(good) is p and Pr(r|good) is the normal CDF evaluated at r.

If we assume that good slot machines neighbour other good machines (and vice-versa) we can update our belief about a cell b given reward r at machine a by using the same formula but increasing the σ^2 of Pr(r|good) by multiplying it by $e^{\frac{d(a,b)^2}{2h}}$ where h is a given parameter. The farther b is from a the larger the σ^2 and the less an observation changes our belief of b. We add forgetting by letting p drift by δ towards 0.5 each passing day.

Compared to other algorithms, the ad-hoc classifier produces quite poor heatmaps as shown in figure 18. However the objective of these algorithms is not to predict well everywhere but to guide where to exploit next. The overall predictive quality of the heatmap is not as important as the ability to identify peaks which this classifier does well.

This simple Bayes updating rule is computationally efficient: observation is $O(1)$ and prediction is $O(K)$. This

implementation of Bayes updating needs 5 parameters. It needs the average rewards expected at good and bad locations μ_g,μ_b and their standard deviation σ. It also needs a drift factor δ and a distance bandwidth h.

3 Simulations

We present here 4 biological scenarios and rank decision algorithms by the profits they make when facing them. For each scenario and decision algorithm pair, we run 100 simulations, each 5 year long and record the average fishery profits. We added two control strategies to the evaluations: "random" agents that pick a cell at random each trip and "fixed" agents which pick a cell at the beginning of the simulation and never switch away from it.

3.1 Baseline scenario

3.1.1 Description

The first scenario we run is the standard POSEIDON baseline scenario. We list this scenario's parameters in the appendix. There is one species of fish, it grows logistically and spreads slowly from cell to cell. Biomass is initially distributed at random. It is a plentiful and easy scenario: almost any fishing spot is profitable. As such even exploiting entirely at random is a good strategy. You can improve slightly by focusing on fishing spots nearer to port first and move out as their biomass is depleted.

3.1.2 Results

Figure 19 ranks algorithms by profits made.

Except for the non-discretized version of UCB1 all bandit algorithms outperform the fixed strategy and most algorithms outperform the random strategy. UCB1 performs poorly because it spends too much time exploring every possible cell before starting to exploit.

Imitation based algorithms perform worse than bandit algorithms and in fact worse than random choice. Imitation drives agents toward better areas but congestion accelerates local depletion and lowers profits. The baseline scenario is plentiful enough that the benefits from imitation are lower than the congestion costs it generates. Social annealing and explore-exploit-imitate outperform gravitational search and particle swarm.

Heatmap agents perform as well as bandits. All except geographical weighted regressions perform better than random.

3.2 Fine scenario

3.2.1 Description

In this scenario there are two species of fish. Red fish live on the north side of the map. Blue fish on the south side. There is a fine for landing blue fish making it unprofitable. Agents need to learn to fish on the north side of the map only as they do not know the distribution of fish.

3.2.2 Results

The results are summarised in figure 20. In this scenario random and fixed strategies fail.

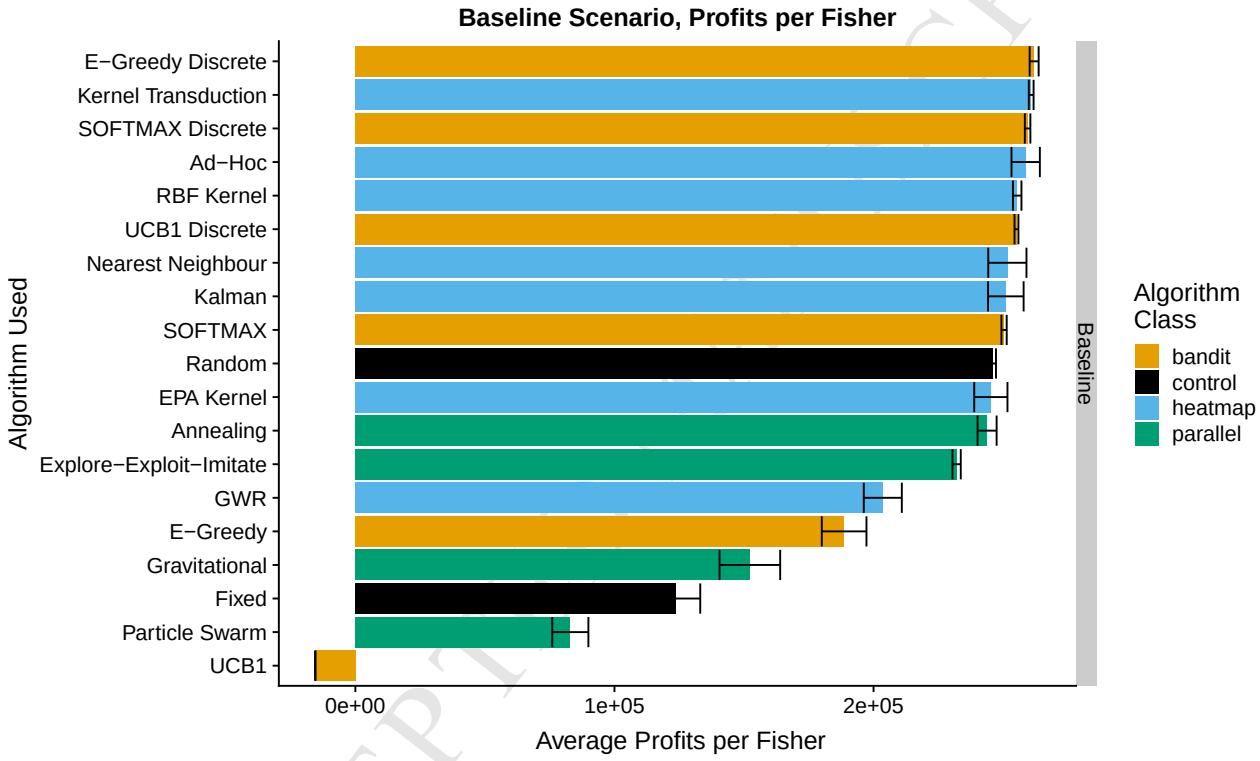


Figure 19: Average profits made in the baseline scenario by each algorithm. The black interval line represents the 2 standard deviations interval of per-scenario performance

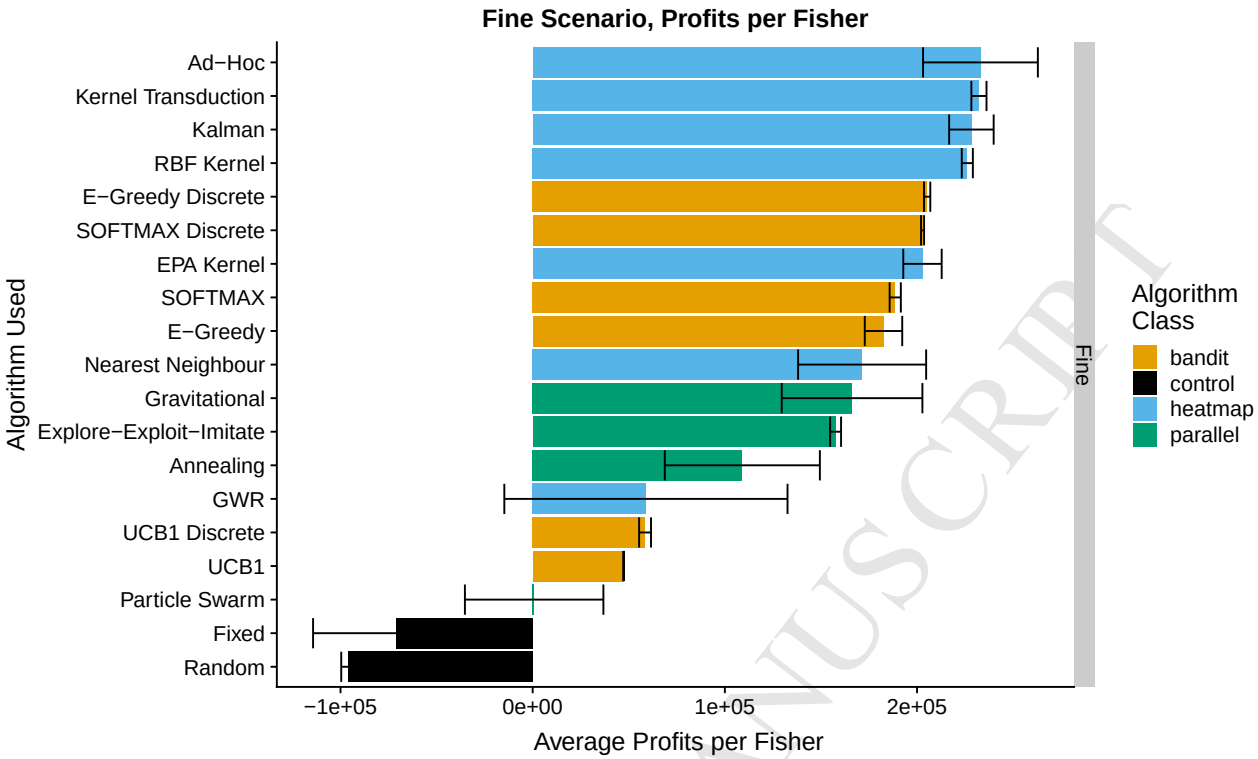


Figure 20: Average profits made in the fine scenario by each algorithm. The black interval line represents the 2 standard deviations interval of per-scenario performance

Heatmap agents outperform bandit and imitation algorithms in this scenario. SOFTMAX and ϵ-greedy perform better than UCB1. The UCB1 algorithm spends too much time exploring the south part of the sea to make sure the profits are negative there.

Imitation algorithms still perform poorly compared to bandits and heatmap agents. Imitation is useful to quickly avoid blue fish areas but the congestion costs from the baseline scenario are also present here. Imitation again leads to mediocre performance.

3.3 Gear change scenario

3.3.1 Description

This scenario tests the ability of fishers to re-adjust to policy shocks. Again the fish is either red or blue, with red fish in the north side of the map and blue fish in the south. Initially we give agents gear that catches only red fish. They need to learn to tow on the north side of the map. However at the end of the second year of simulation we change the gear of the fishers so that it only catches blue fish. Agents need to forget past experience and switch to fish in the south. They are not notified of the change and have to learn by trial and error.

We run the simulation for 3 years and rank the decision algorithms by the profits made on the last year.

3.3.2 Results

The results are summarised in figure 21.

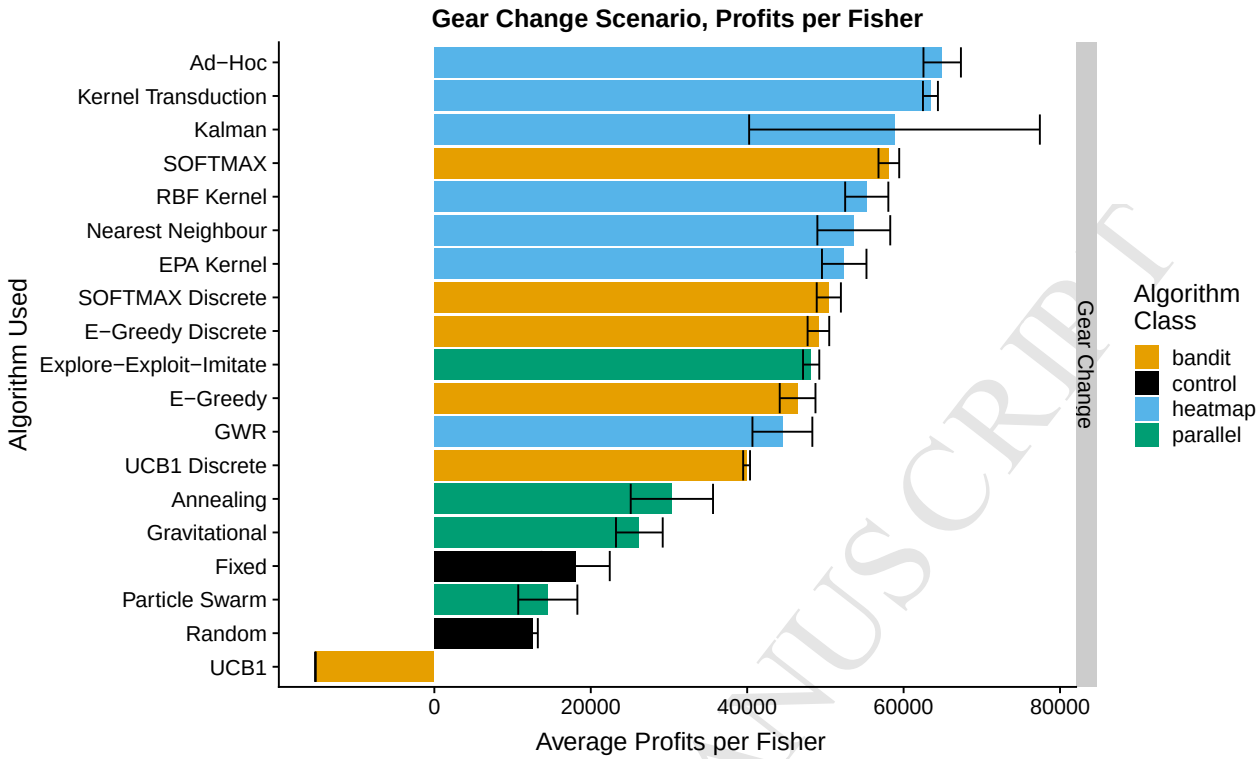


Figure 21: Average profits made in the gear change scenario by each algorithm. The black interval line represents the 2 standard deviations interval of per-scenario performance

Again, heatmap algorithms outperform bandit and imitation algorithms. This scenario tests statistical algorithms because previously discovered heatmaps become actively misleading after the gear shock (two years into the simulation). Heatmaps algorithms forget fast enough to outperform the alternatives. Unsurprisingly the ad hoc method described in section 2.3.6 performs best since it has the least to forget.

By comparison the UCB1 bandit algorithm is the worst performer (even worse than random and fixed). This is because the UCB1 algorithm needs to forget about every cell independently unless discretized. SOFTMAX and ϵ-greedy perform better because by the end of year 2 they have focused on only a few fishing spots and have only those to forget.

When agents need to switch fishing location due to gear shock, only explore-exploit-imitate does well among the imitative algorithms. This is because explore-exploit-imitate forces fishers to explore and when someone randomly falls into the right part of the sea she is quickly imitated by the rest. Gravitational search and particle swarm perform poorly because the agents are too close together and it takes time for anyone to land on the correct part of the sea. Social annealing performs better because it also forces agents to explore (since there is always plenty of agents doing slightly worse than average) but not as good as explore-exploit-imitate because those that do land on the profitable areas cannot be copied directly.

3.4 Chasing school scenario

3.4.1 Description

In this scenario all of the sea is empty except for a few cells where a fish school of infinite biomass lives. Fishers need to find this school and tow there. The fish school moves over time however so that fishers need

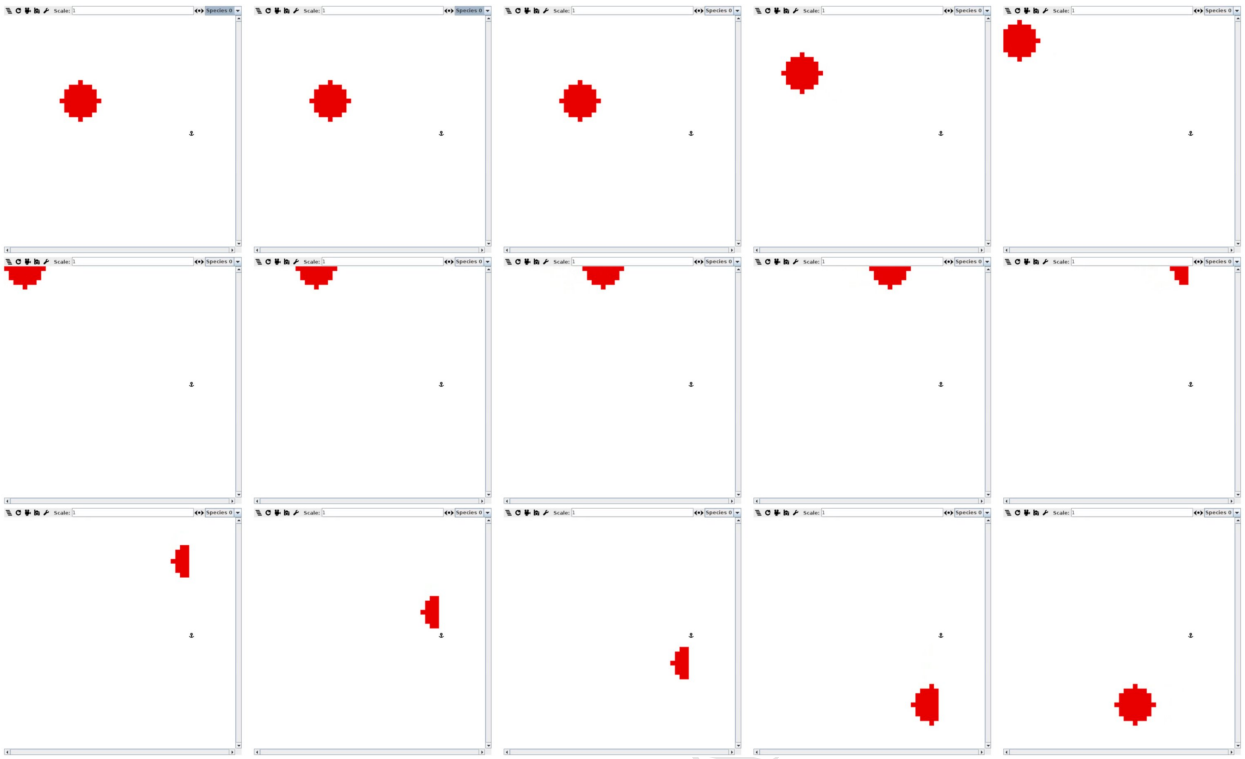


Figure 22: Animation showing one school roaming the map in a predictable fashion. The darker the color of the cell the more biomass will be caught in that area.

to chase it. In this scenario the motion is very simple as the fish school traces a square around the map as shown in figure 22.

3.4.2 Results

The results are summarised in figure 23.

This is the only scenario where gravitational search performs well and in fact is the best among all the algorithms. This is because this scenario has no penalty for overcrowding and presents a moving target that requires information sharing. Explore-exploit-imitate performs less well because imitation happens slowly and often fishers lose track of the school before they managed to attract the rest of the fleet. Particle swarm performs poorly because memory weighs people away from chasing. Social annealing also does badly because it does not allow direct imitation.

With the exception of gravitational search,the heatmap based algorithms again outperform all others. This performance however suffers from a large standard deviation primarily due to scenario runs where fishers fail to ever discover where the fish school is.

Bandit algorithms fail this scenario. Chasing fish require either imitation or better geographical inference that bandit algorithms do not possess. This is a clear scenario where thinking geographically helps even if agents do not have a full model representation of fish schools and their movement (which could be inputted in the Kalman filter to improve it further).

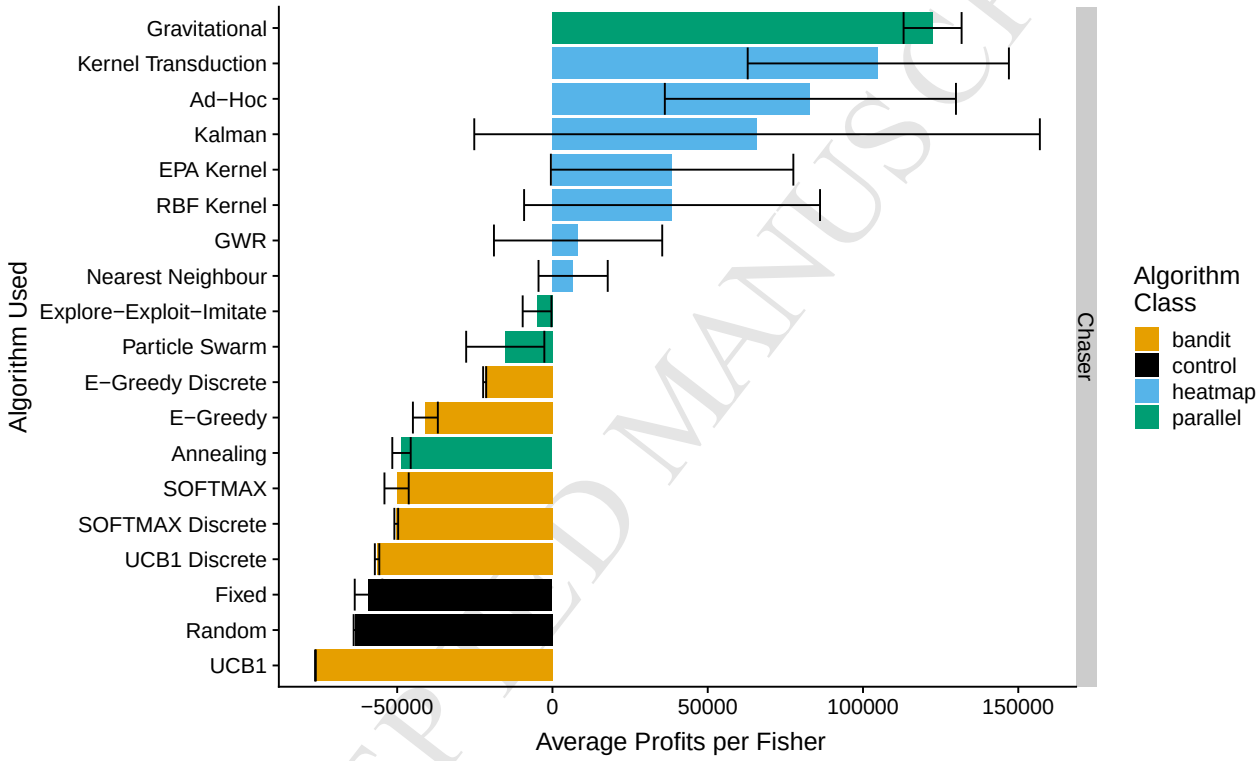


Figure 23: Average profits made in the chaser scenario by each algorithm. Bandit and imitation algorithms are greyed out. The black interval line represents the 2 standard deviations interval of per-scenario performance

4 Discussion

4.1 General results

In all scenarios, the most complicated class of agents (heatmap) performed better. However within each class, simpler agents outperformed complicated ones. Among heatmap agents, ad-hoc classification outperformed geographical weighted regression; among imitative algorithms, explore-exploit-imitate outperformed particle swarms; among bandit algorithms, ϵ-greedy outperformed UCB1.

There is therefore a balance to strike between sophistication and flexibility. Adding imitation and spatial awareness may improve overall profits (agent efectiveness) but too much sophistication may make agents unable to adapt to a changing environment (as greater accumulation of data and finer tuned parameters are needed to get the best from such algorithms). This matches the split in fishery science(Branch et al. 2006) between "generalist fishers" (more flexible) and "specialist" (more sophisticated). Because all the scenarios studied here had fast-changing biologies, generalist agents had an edge which matches empirical observations.

Some strategies (especially heatmap ones) performed well in all scenarios, including when a policy shock made all their memories obsolete and misleading (section 3.3). Bandit strategies performed well as long as they faced simple scenarios. When fish aggregated into a hard to track school (section 3.4) all bandit algorithms performed no better than random. Vice-versa, imitative algorithms performed poorly except when facing a moving school. This is because for most examples the congestion costs (multiple boats fishing the same spot) outweighed the benefits of learning from others.

We expect therefore bandit strategies to explain well the behaviour in fisheries with a limited number of fishing spots where information sharing is not necessary (such as clam fisheries) or where few boats can quickly consume local aggregations of fish (such as herring). We expect imitative algorithms to better explain either fleets chasing fast moving schools of fish (such as dolphin-set tuna) or small fisheries where the effort of many boats produces little congestion.

Finally, a comment on the importance of applying well-known and well-understood algorithms to specific problems through agent-based models. Most of these algorithms are decades-old and within the field of computer science, their complexity and regret bounds have been studied extensively. Solutions are typically framed in terms of worst case performance, but our findings suggest these are not necessarily relevant for judging algorithm performance for other less extreme cases and for the very specific problem of fishing behaviour. Hence we show that the theoretically inferior ϵ-greedy algorithm outperforms UCB1. Other applied work will without doubt uncover other such reversals, as in fact the "no free lunch" theorems predicts (Wolpert and Macready 1995).

4.2 Relevance

O'Sullivan et al. (2016) warns of the YAAWN (yet another agent-based model) syndrome: fitting tightly to a specific scenario with no hope to generate general insights. Many agent-based models of fisheries exist, for example Gao and Hailu (2011) models recreational fishers in Western Australia through random utility models, Dorn (2001) models commercial hake fishers in North America through fixed thresholds and Kalman filters, Elliston and Cao (2006) models conceptual fishers by Bellman equation programming. Each decision algorithm is incompatible with the other models. It may be that the fisheries' economic and biological features are all so unique to require highly specialized algorithms. We showed however that some algorithms are general enough to be useful in all scenarios, challenging the assumed need for specificity.

On the other hand we did show that some algorithms perform well in one scenario and poorly in another. It is possible then, perhaps due to a policy shock or climate change, for system conditions to change to the point where a previously optimal heuristic becomes inefficient. As such, the endogeneity of the heuristics is a serious risk for policy analysis: even a perfectly identified heuristic (due to surveys, interviews or data-analysis) may be too contingent on current conditions and therefore of limited use for prediction. Our heuristic zoo

approach however can help through sensitivity analysis. We can ask ourselves what would be the worst possible heuristic to emerge for a given policy shock and plan around it.

4.3 Future work

Through POSEIDON we built a platform to study the patterns generated by different decision algorithms in the same environment. The next step of our work is to apply this to real fisheries and compare them to simulated behavioural patterns. It may be that a single statistical agent will be able to describe fisher behaviour in a wide range of fisheries, but it may be that each single area will exhibit its own highly specialized algorithm.

We extracted all the decision algorithms from POSEIDON in pseudo-code in this paper, and in the separate Java library `discrete-choosers`. We hope other agent-based modellers will find this library useful and test its efficiency in other geographical agent-based models. We agree with Groeneveld et al. (2017) that re-use is fundamental for the progress of agent-based modelling and we hope a ready-made library will contribute to the task.

4.4 Weaknesses

These simulations contained no heterogeneity: all agents within the same run used the same decision algorithm. This was done to showcase each algorithm in isolation but as Klein, Barbier, and Watson (2017) interviews show: some fishers always deviate from the majority. Not pursued here is also any attempt to model an ecology of algorithms to see which would survive. For example we show that a population using kernel regressions is more efficient in general than one using imitation techniques. However it may be the case that when the population is split in half where some use regressions and some imitate, the imitative sub-population would do better. A replicator dynamic (see Sandholm 2009) could be implemented to see which algorithms are imitated and which discarded when personal rather than global profitability is at stake.

Missing from this paper is any algorithm that performs long-term planning. Bandit algorithms manage the longer term benefits of exploration in an implicit and hard to control way. If we had data on the discount rate of fishers however we might use it to solve the bandit problem through explicit dynamic programming. This is the Gittins index strategy(Gittins 1979) and requires model knowledge. Agent based models are hard to turn into well-behaved transition functions so that perhaps only reinforcement learning could help but this avenue is not pursued here.

4.5 Conclusion

We presented 13 decision-making algorithms for exploiters in agent-based models. We used the fishery model POSEIDON to isolate their strengths and weaknesses. We presented 3 general tuning methods to discover the best parameters for these algorithms both exogenously and within the model. We strived to keep this presentation as general as possible to show how these algorithms can be used in other models.

On the one hand we demonstrated it is easy to adapt decision making algorithms from computer science to agents within agent-based models. On the other hand we showed that theoretical bounds judging these algorithms will predict poorly their performance for the very specific tasks an agent-based model require. It will pay then to test most available algorithms for each new application and we hope our library will make the task easier for other modellers.

Appendix

Parameters

Baseline scenario

Table 2: The parameters of the baseline scenario; these are actually just the parameters of the default POSEIDON run except for $r = 0$ (no regrowth).

Parameter	Value	Meaning
Biology	One Species	
K	5000	max units of fish per cell
m	0.001	fish speed
r	0	Malthusian growth parameter
Fisher	Explore-Exploit-Imitate	
rest hours	12	rest at ports in hours
fishers	100	number of fishers
max days at sea	5	time after which boats must come home
Map		
width	50	map size horizontally
height	50	map size vertically
port position	40,25	location of port
cell width	10	width (and height) of each cell map
Market		
market price	10	$ per unit of fish sold
gas price	0.01	$ per litre of gas
Gear		
catchability	0.01	% biomass caught per tow hour
speed	5.0	distance/h of boat
hold size	100	max units of fish storable in boat
litres per unit of distance	10	litres consumed per distance travelled
litres per trawling hour	5	litres consumed per hour trawled

Fine scenario

Table 3: The parameters changed between the baseline and fine scenario

Parameter	Old Value	New Value
Biology	1 Species	2 Species
K	5000	10000 red; 10000 blue;
market price	10$	10$ red; -10$ blue
Gear		
catchability	0.01	0.01 red; 0.01 blue

Gear scenario

Table 4: The parameters changed between the baseline and gear scenario

Parameter	Old Value	New Value
Biology	1 Species	2 Species
K	5000	10000 red; 10000 blue;
market price	10$	10$ red; 10$ blue
Gear		
catchability	0.01	0.01 red; 0 blue
catchability at year 2	-	0 red; 0.01 blue

Chasing School scenario

Table 5: The parameters changed between the baseline and chasing school scenario

Parameter	Old Value	New Value
Biology	One Species	One School
K	5000	-
m	0.001	-
r	0	-
School diameter (cells)	-	4
Speed in days	-	10
Waypoints	-	$(0,0);(40,0);(40,40);(0,40)$

Tuning

The algorithms presented are general purpose enough that behaviour can change radically with the parameters

When algorithms are as general as the ones listed above, parameters have a large effect on performance and behaviour. Figure 24 shows the performance of the discretized ϵ-greedy algorithm in the "fine" scenario. The same algorithm can generate average fisher profits of 200,000 when properly tuned or lose money when poorly tuned.

This section contains three separate ways to tune parameters. In POSEIDON we used exogenous tuning for bandit and imitation algorithms, social tuning for heatmap algorithms. We tuned the bandit and imitation algorithms to optimize their performance in the baseline scenario. Those parameters were then used without being re-optimized in all the other tests. We did so in order not to exaggerate their performance. The list of all parameters used is in table 6.

Table 6: List of parameters used and how they were tuned. For algorithms tuned socially we list the initial range at which the parameters are initialized.

parameter	algorithm	tuning	value
ϵ	E-Greedy	exogenous	0.009583
α	E-Greedy	exogenous	0.97
$\underline{x}$	UCB1	exogenous	0
$\bar{x}$	UCB1	exogenous	12
α	UCB1	exogenous	0.274

parameter	algorithm	tuning	value
Z	Gravitational	exogenous	1
velocity_0	Gravitational	exogenous	-10
G	Gravitational	exogenous	10
`inertia`	Particle Swarm	exogenous	0.733
α	Particle Swarm	exogenous	0.1
β	Particle Swarm	exogenous	0.37
ϵ	Particle Swarm	exogenous	0.001661
ϵ	Explore-Exploit-Imitate	exogenous	0.48
δ	Explore-Exploit-Imitate	exogenous	20
δ	Social Annealing	exogenous	10
h_x	Nearest Neighbour	social	1-1000
h_y	Nearest Neighbour	social	1-1000
h_{time}	Nearest Neighbour	social	1-1000
$h_{\text{port distance}}$	Nearest Neighbour	social	1-1000
`neighbors`	Nearest Neighbour	social	1-10
h_x	RBF Kernel	social	50-500
h_y	RBF Kernel	social	50-500
h_{time}	RBF Kernel	social	100-100000
$h_{\text{port distance}}$	RBF Kernel	social	1-1000
h_x	EPA Kernel	social	50-500
h_y	EPA Kernel	social	50-500
h_{time}	EPA Kernel	social	100-100000
$h_{\text{port distance}}$	EPA Kernel	social	50-500
h_x	Kernel Transduction	social	1-200
h_y	Kernel Transduction	social	1-200
$h_{\text{port distance}}$	Kernel Transduction	social	1-200
λ	Kernel Transduction	fixed	.95
ϵ	Kalman	social	1-100
σ_x^2	Kalman	fixed	10000
σ_z^2	Kalman	social	1-100
β	Kalman	social	1-100
c	Kalman	social	-2-2
σ_x^2	GWR	fixed	10000
h	GWR	social	.1-50
λ	GWR	social	.8-1
μ_g	Ad-Hoc	social	10-30
μ_b	Ad-Hoc	social	-20-0
σ^2	Ad-Hoc	social	10-30
h	Ad-Hoc	social	0.1-50
δ	Ad-Hoc	fixed	.005

Exogenous tuning

Assume that the people being simulated are experts and would know the best way to aggregate information. It would be fair then to simulate these agents as if using the best parameters. Finding these parameters is an optimisation problem where the model is a black box and we are looking for the inputs that will generate the maximum performance. Once the optimal parameters are found we assume that agents use them.

To optimize parameters we need to score outputs. The choice of score function however carries a series of unintended consequences in the POSEIDON model. If we try to maximize average profits, the optimizer discovers parameters, especially for complicated tools like Kalman, that conserve biomass rather than greedily exploit everything at once. Conservationist agents generate more profits in the long run but we are looking

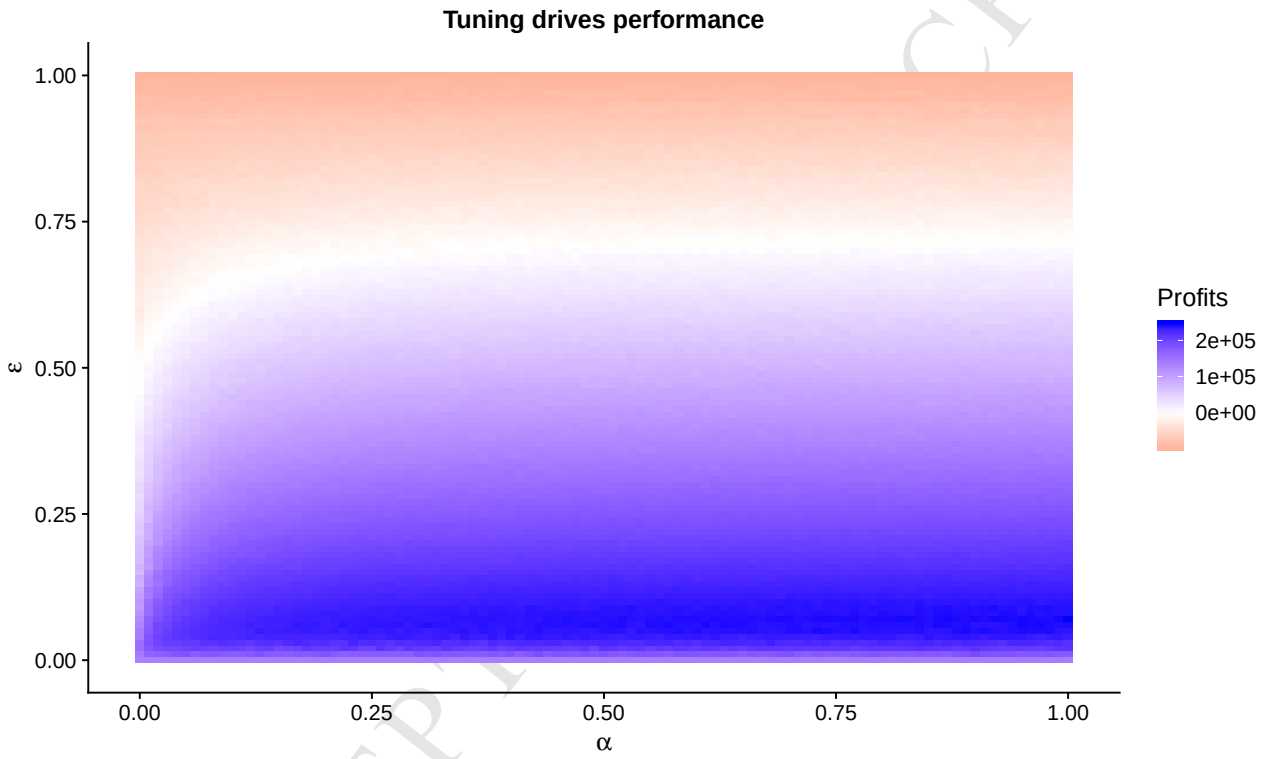


Figure 24: The performance of the ϵ greedy bandit algorithm in the fines scenario as its two parameters are changed

for realistic exploiting behaviour. We avoided this problem in this paper by assuming biomass never regrows in tested scenarios.

An alternative score for heatmap algorithms would be to judge them by the prediction error made. Minimizing prediction errors made might however lead to agents that are very conservative and fish always at the same location which tend to produce small prediction errors but also poor profits. The best option is always to minimize the statistical distance to behavioural data(Filatova et al. 2013; Smajgl et al. 2011).

This tuning strategy is completely exogenous as the exploration is done by the optimizer before the simulation starts. This tuning is non-adaptive since the parameters are fixed for the duration of the simulation. It is best used to simulate long-term expertise or human action over a steady geographical system.

Social tuning

Here we let the agents adapt their parameters over time within the simulation. The idea is to have a low frequency exploration of the parameter space while the agents are performing a high-frequency exploration of the geographical space.

We do this by running the explore-exploit-imitate algorithm over the parameter space of the decision algorithm. Every period, which for the POSEIDON example is two simulated months, each agent has a probability ϵ of shocking its current decision parameters (exploration), otherwise copying the parameters of a friend who is doing better (imitation) and maintaining its current parameters if friends are doing worse (exploitation).

Because this tuning progresses with the simulation when conditions change agents can adapt their parameters in accordance. However for this method to work imitation must make sense. This means that agents have to be similar enough so that copying other people's parameters is a reasonable strategy. We must also assume agents can tell their competitors' parameters. Finally, agents' adaptability weakens over time as better parameters are imitated and overall diversity decreases unless exploration is frequent. Algorithm 8 is a pseudo-code representation of social tuning.

Algorithm 8 Social tuning

```
while game is not over do
  if random < ε then {with probability ε, explore}
    δ̂ ~ U[−δ, δ]
    parameters ← parameters + δ̂
  else
    profits_f are the profits made by your most profitable friend
    profits_me are the profits just made by this agent
    if profits_f > profits_me then
      copy friend parameters
    else
      do not change parameters
    end if
  end if
end while
```

Personal tuning

Alternatively for some algorithms we can have each agent tune parameters on their own. Since heatmaps are used for prediction the agent can adjust her parameters over time toward values that minimize prediction errors. We simulate this through gradient descent (Jones 2008, chap. 4 and 18) applying the algorithm at each observation as shown in algorithm 9.

where η and ϵ are hyper-parameters.

Algorithm 9 Personal tuning

while game is not over **do**
  error $\leftarrow$ (prediction $-$ reward)^2
  for i $= 0$ to number of parameters **do** {shock the parameters by a small value and see if that would have predicted better}
    $h[i] \sim U[-\epsilon, \epsilon]$ {Where ϵ is a small number}
    alternative$[i] \leftarrow$ parameter$[i] + h[i]$
  end for
  error$_a$ is the prediction error using the alternative parameters
  for i $= 0$ to number of parameters **do** {modify all parameters depending by moving along the gradient}
    gradient$[i] = \frac{\text{error}_a - \text{error}}{h[i]}$ {compute change in prediction error}
    parameter$[i] \leftarrow$ parameters$[i] + \eta \cdot$ gradient$[i]$
  end for
end while

This tuning algorithm is adaptive like social tuning but assumes no information sharing. It makes no assumption on the fisher except a willingness to revise parameters as new observations come in. Its main weakness is the need to compute numerical derivatives of parameters on error. For the methods listed in section 2.3 only the kernel regression (and Nearest Neighbour regression with enough neighbours) can be used. For the other methods a change in parameters has no instantaneous effect until more observations come in and are therefore unsuitable for this tuning.

We did not use this method in the POSEIDON runs because it progresses too slowly to make much difference.

References

Agarwal, Alekh, Daniel Hsu, Satyen Kale, John Langford, Lihong Li, and Robert E. Schapire. 2014. "Taming the Monster: A Fast and Simple Algorithm for Contextual Bandits." *Journal of Machine Learning Research* 32 (i):1–28. https://doi.org/10.1613/jair.301.

Allen, P M, and J M McGlade. 1986. "Dynamics of discovery and exploitation: The case of the Scotian Shelf groundfish fisheries." *Canadian Journal of Fisheries and Aquatic Sciences* 43 (6). NRC Research Press Ottawa, Canada:1187–1200. https://doi.org/10.1139/f86-148.

Auer, Peter, N Cesa-bianchi, and Paul Fischer. 2002. "Finite time analysis of the multiarmed bandit problem." *Machine Learning* 47 (2-3). Kluwer Academic Publishers:235–56. https://doi.org/10.1023/A:1013689704352.

Axelrod, Robert. 1980. "Effective Choice in the Prisoner' s Dilemma." *The Journal of Conflict Resolution* 24 (1):3–25. https://doi.org/10.1177/002200278002400101.

Bailey, Richard, Ernesto Carrella, Robert L. Axtell, Matthew G. Burgess, Reniel B. Cabral, Michael Drexler, Christopher Dorsett, J.K. Madsen, Andreas Merkl, and Steven Eugene Saul. 2018. "A computational approach to managing coupled human-environmental systems: the POSEIDON model of ocean fisheries." *Sustainability Science* in press.

Beecham, J. A., and G. H. Engelhard. 2007. "Ideal free distribution or dynamic game? An agent-based simulation study of trawling strategies with varying information." *Physica A: Statistical Mechanics and Its Applications* 384 (2). North-Holland:628–46. https://doi.org/10.1016/j.physa.2007.05.053.

Bell, Andrew R., Derek T. Robinson, Ammar Malik, and Snigdha Dewal. 2015. "Modular ABM development for improved dissemination and training." *Environmental Modelling and Software* 73:189–200. https://doi.org/10.1016/j.envsoft.2015.07.016.

Blackwell, Tim. 2007. "Particle Swarm Optimization in Dynamic Environments." In *Evolutionary Computation in Dynamic and Uncertain Environments*, edited by Shengxiang Yang, Yew-Soon Ong, and Yaochu Jin,

49:29–49. Studies in Computational Intelligence. Springer Berlin / Heidelberg. http://www.springerlink.com/content/u36w892m26240g7k/abstract/.

Blackwell, Tim, and Jürgen Branke. 2006. "Multiswarms, exclusion, and anti-convergence in dynamic environments." *IEEE Transactions on Evolutionary Computation* 10 (4):459–72. https://doi.org/10.1109/TEVC.2005.857074.

Boero, R, G Bravo, M Castellani, and F Squazzoni. 2010. "Why Bother with What Others Tell You?" *An Experimental Data-Driven Agent-Based Model* 13 (3):3.

Boettiger, C., M. Mangel, and S. Munch. 2015. "Avoiding tipping points in fisheries management through Gaussian process dynamic programming." *Proceedings of the Royal Society B: Biological Sciences* 282 (1801):20141631–1. https://doi.org/10.1098/rspb.2014.1631.

Branch, Trevor A, Ray Hilborn, Alan C Haynie, Gavin Fay, Lucy Flynn, Jennifer Griffiths, Kristin N Marshall, et al. 2006. "Fleet dynamics and fishermen behavior: lessons for fisheries managers." *Canadian Journal of Fisheries and Aquatic Sciences* 63 (7):1647–68. https://doi.org/10.1139/f06-072.

Brunsdon, C., S. Fotheringham, and M. Charlton. 1998. "Geographically Weighted Regression." *Journal of the Royal Statistical Society: Series D (the Statistician)* 47 (3):431–43. https://doi.org/10.1111/1467-9884.00145.

Bubeck, Sébastien, and Nicolò Cesa-Bianchi. 2012. "Regret Analysis of Stochastic and Nonstochastic Multi-armed Bandit Problems." https://doi.org/10.1561/2200000024.

Cabral, Reniel B., Rollan C. Geronimo, May T. Lim, and Porfirio M. Aliño. 2010. "Effect of variable fishing strategy on fisheries under changing effort and pressure: An agent-based model application." *Ecological Modelling* 221 (2):362–69. https://doi.org/10.1016/j.ecolmodel.2009.09.019.

Caplin, Andrew, Mark Dean, and Daniel Martin. 2011. "Search and satisficing." *American Economic Review* 101 (7):2899–2922. https://doi.org/10.1257/aer.101.7.2899.

Caron, Stéphane, Branislav Kveton, Marc Lelarge, and Smriti Bhagat. 2012. "Leveraging Side Observations in Stochastic Bandits." *Uncertainty in Artificial Intelligence*. http://arxiv.org/abs/1210.4839.

Cenek, Martin, and Maxwell Franklin. 2017. "An adaptable agent-based model for guiding multi-species Pacific salmon fisheries management within a SES framework." *Ecological Modelling* 360 (September):132–49. https://doi.org/10.1016/j.ecolmodel.2017.06.024.

Cleveland, William S. 1979. "Robust Locally Weighted Regression and Smoothing Scatterplots." *Journal of the American Statistical Association* 74 (368):829. https://doi.org/10.2307/2286407.

Dorn, Martin W. 2001. "Fishing behavior of factory trawlers: A hierarchical model of information processing and decision-making." *ICES Journal of Marine Science* 58 (1). Oxford University Press:238–52. https://doi.org/10.1006/jmsc.2000.1006.

Elliston, Lisa, and Liangyue Cao. 2006. "An agent-based bioeconomic model of a fishery with input controls." *Mathematical and Computer Modelling* 44 (5-6). Pergamon:565–75. https://doi.org/10.1016/j.mcm.2006.01.010.

Epstein, Joshua M., and Robert. Axtell. 1996. *Growing artificial societies : social science from the bottom up*. Brookings Institution Press.

Evans, George W, and Seppo Honkapohja. 2001. *Learning and Expectations in Macroeconomics*. Vol. 24. 2008. Princeton University Press. https://doi.org/10.1016/j.jedc.2005.06.009.

Faragher, Ramsey. 2012. "Understanding the basis of the kalman filter via a simple and intuitive derivation." *IEEE Signal Processing Magazine* 29 (5):128–32. https://doi.org/10.1109/MSP.2012.2203621.

Filatova, Tatiana, Peter H. Verburg, Dawn Cassandra Parker, and Carol Ann Stannard. 2013. "Spatial agent-based models for socio-ecological systems: Challenges and prospects." *Environmental Modelling and Software* 45:1–7. https://doi.org/10.1016/j.envsoft.2013.03.017.

Fowler, James H, and Michael Laver. 2008. "A tournament of party decision rules." *Journal of Conflict Resolution* 52 (1):68–92. https://doi.org/10.1177/0022002707308598.

Gao, Lei, and Atakelty Hailu. 2011. "Evaluating the effects of area closure for recreational fishing in a coral reef ecosystem: The benefits of an integrated economic and biophysical modeling." *Ecological Economics* 70 (10). Elsevier:1735–45. https://doi.org/10.1016/j.ecolecon.2011.04.014.

Gittins, John C. 1979. "Bandit processes and dynamic allocation indices." *Statistics* 41 (2):148–77. https://doi.org/10.2307/2985029.

Gotts, Nicholas M., and J. Gary Polhill. 2009. "When and how to imitate your neighbours: Lessons from and for FEARLUS." *Journal of Artificial Societies and Social Simulation* 12 (3). JASSS. http://jasss.soc.surrey.ac.uk/12/3/2.html.

Groeneveld, J., B. Müller, C.M. Buchmann, G. Dressler, C. Guo, N. Hase, F. Hoffmann, et al. 2017. "Theoretical foundations of human decision-making in agent-based land use models – A review." *Environmental Modelling & Software* 87:39–48. https://doi.org/10.1016/j.envsoft.2016.10.008.

Hastie, Trevor J, Robert John Tibshirani, and Jerome H Friedman. 2009. *The elements of statistical learning*. 1st ed. Vol. 1. Berlin: Springer series in statistics Springer, Berlin. http://opac.inria.fr/record=b1127878.

Holland, Daniel S., and Jon G. Sutinen. 2000. "Location Choice in New England Trawl Fisheries: Old Habits Die Hard." *Land Economics* 76 (1):133. https://doi.org/10.2307/3147262.

Hutniczak, Barbara, and Angela Münch. 2018. "Fishermen's location choice under spatio-temporal update of expectations." *Journal of Choice Modelling* 28 (September):124–36. https://doi.org/10.1016/j.jocm.2018.05.002.

Hynes, Stephen, Hans Gerritsen, Benjamin Breen, and Mark Johnson. 2016. "Discrete choice modelling of fisheries with nuanced spatial information." *Marine Policy* 72:156–65. https://doi.org/10.1016/j.marpol.2016.07.004.

Jones, Tim. 2008. *Artificial Intelligence: A Systems Approach*. Prentice Hall. https://doi.org/10.1017/S0269888900007724.

Kennedy, James, and Russel C Eberhart. 2001. *Swarm Intelligence*. Morgan Kaufmann. https://doi.org/10.1007/978-3-319-09952-1.

Kennedy, J, and R Eberhart. 1995. "Particle swarm optimization." *Neural Networks, 1995. Proceedings., IEEE International Conference on* 4:1942–8 vol.4. https://doi.org/10.1109/ICNN.1995.488968.

Klein, E. S., M. R. Barbier, and J. R. Watson. 2017. "The dual impact of ecology and management on social incentives in marine common-pool resource systems." *Royal Society Open Science* 4 (8):170740. https://doi.org/10.1098/rsos.170740.

Krzyzak, Adam. 1992. "Global Convergence of the Recursive Kernel Regression Estimates with Applications in Classification and Nonlinear System Estimation." *IEEE Transactions on Information Theory* 38 (4):1323–38. https://doi.org/10.1109/18.144711.

Kuleshov, Volodymyr, and Doina Precup. 2014. "Algorithms for multi-armed bandit problems." https://doi.org/10.1145/1109557.1109659.

Kunz, Jennifer. 2011. "Group-level exploration and exploitation: A computer simulation-based analysis." *Jasss* 14 (4):18. https://doi.org/10.18564/jasss.1798.

Lansing, J Stephen, and James N Kremer. 1993. "Emergent Properties of Balinese Water Temple Networks: Coadaptation on a Rugged Fitness Landscape." *American Anthropologist* 95 (1):97–114. https://doi.org/10.1525/aa.1993.95.1.02a00050.

Li, Lihong, Wei Chu, John Langford, and Robert E. Schapire. 2010. "A Contextual-Bandit Approach to Personalized News Article Recommendation." In *Proceedings of the 19th International Conference on World Wide Web*, 661–70. WWW '10. ACM. https://doi.org/10.1145/1772690.1772758.

Little, L. R., and A. D. McDonald. 2007. "Simulations of agents in social networks harvesting a resource." *Ecological Modelling* 204 (3-4):379–86. https://doi.org/10.1016/j.ecolmodel.2007.01.013.

Luke, Sean, Claudio Cioffi-Revilla, Liviu Panait, Keith Sullivan, and Gabriel Balan. 2005. "MASON: A Multiagent Simulation Environment." *Simulation* 81 (7). Sage PublicationsSage CA: Thousand Oaks, CA:517–27. https://doi.org/10.1177/0037549705058073.

Marcoul, Philippe, and Quinn Weninger. 2008. "Search and active learning with correlated information: Empirical evidence from mid-Atlantic clam fishermen." *Journal of Economic Dynamics and Control* 32 (6). North-Holland:1921–48. https://doi.org/10.1016/j.jedc.2007.07.003.

Minka, Tom. 1999. "From hidden markov models to linear dynamical systems." Tech. Rep. 531, Vision; Modeling Group of Media Lab, MIT.

Morton, Margaret. 2005. "Fishing for knowledge." *Australian Journal of Pharmacy* 86 (1020):202. https://doi.org/10.2307/3146679.

Nadaraya, E. A. 1964. "On Estimating Regression." *Theory of Probability & Its Applications* 9 (1):141–42. https://doi.org/10.1137/1109020.

Nonaka, Etsuko, and Petter Holme. 2007. "Nordic Society Oikos Agent-Based Model Approach to Optimal Foraging in Heterogeneous Landscapes: Effects of Patch." *Source: Ecography* 30 (6):777–88.

O'Sullivan, David, Tom Evans, Steven Manson, Sara Metcalf, Arika Ligmann-Zielinska, and Chris Bone. 2016. "Strategic directions for agent-based modeling: avoiding the YAAWN syndrome." *Journal of Land Use Science* 11 (2). NIH Public Access:177–87. https://doi.org/10.1080/1747423X.2015.1030463.

Pandey, Sandeep, Deepayan Chakrabarti, and Deepak Agarwal. 2007. "Multi-armed bandit problems with dependent arms." In *Proceedings of the 24th International Conference on Machine Learning - Icml '07*, 721–28. ACM. https://doi.org/10.1145/1273496.1273587.

Pedregosa, Fabian, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, et al. 2011. "Scikit-learn: Machine Learning in Python." *Journal of Machine Learning Research* 12 (Oct):2825–30. https://doi.org/10.1007/s13398-014-0173-7.2.

Pepper, John W, and Barbara B. Smuts. 2000. "The evolution of cooperation in an ecological context." *Dynamics in Human and Primate Societies: Agent-Based Modeling of Social and Spatial Processes*, 45–76.

Polhill, J. G., N. M. Gotts, and A. N.R. Law. 2001. "Imitative versus nonimitative strategies in a land-use simulation." *Cybernetics and Systems* 32 (1-2). Informa UK Ltd:285–307. https://doi.org/10.1080/01969720130000188 5.

Rashedi, Esmat, Hossein Nezamabadi-pour, and Saeid Saryazdi. 2009. "GSA: A Gravitational Search Algorithm." *Information Sciences*, Special section on high order fuzzy sets, 179 (13):2232–48. https://doi.org/10.1016/j.ins.2009.03.004.

Rendell, L., R. Boyd, D. Cownden, M. Enquist, K. Eriksson, M. W. Feldman, L. Fogarty, S. Ghirlanda, T. Lillicrap, and K. N. Laland. 2010. "Why Copy Others? Insights from the Social Learning Strategies Tournament." *Science* 328 (5975):208–13. https://doi.org/10.1126/science.1184719.

Roberts, M. E., and R. L. Goldstone. 2006. "EPICURE: Spatial and Knowledge Limitations in Group Foraging." *Adaptive Behavior* 14 (4):291–313. https://doi.org/10.1177/1059712306072336.

Rogers, Steve. 1996. *Adaptive filter theory*. Vol. 4. Pearson Education. https://doi.org/10.1016/0967-0661(96)82838-3.

Rusmevichientong, Paat, and John N. Tsitsiklis. 2010. "Linearly Parameterized Bandits." *Mathematics of Operations Research* 35 (2):395–411. https://doi.org/10.1287/moor.1100.0446.

Russell, Stuart, and Peter Norvig. 2010. *Artificial Intelligence: a modern approach*. 3rd ed. Vol. 140. 6. Upper Saddle River, NJ, USA: Prentice Hall Press. https://doi.org/10.1049/me:19950308.

Sandholm, William H. 2009. "Evolutionary Game Theory *." Springer.

Satake, Akiko, Heather M. Leslie, Yoh Iwasa, and Simon A. Levin. 2007. "Coupled ecological-social dynamics in a forested landscape: Spatial interactions and information flow." *Journal of Theoretical Biology* 246 (4):695–707. https://doi.org/10.1016/j.jtbi.2007.01.014.

Schlag, Karl H. 1998. "Why Imitate, and If So, How?" *Journal of Economic Theory* 78 (1):130–56. https://doi.org/10.1006/jeth.1997.2347.

Scott, Stephen Lewis. 2016. "Computational modelling for marine resource management." PhD thesis. George Mason University.

Shalizi, Cosma Rohilla. 2013. *Advanced data analysis from an elementary point of view*. Cambridge University Press. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.371.4613&rep=rep1&type=pdf.

Silver, David. 2015. "Reinforcement Learning: Lecture 9: Exploration and Exploitation." http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/XX.pdf.

Simon, Herbert A, and Julian Feldman. 1959. "Theories of Decision-Making in Economics and Behavioral Science." *Source: The American Economic Review* 49 (3):253–83. https://doi.org/10.1257/aer.99.1.i.

Smajgl, Alex, Daniel G. Brown, Diego Valbuena, and Marco G.A. Huigen. 2011. "Empirical characterisation of agent behaviours in socio-ecological systems." *Environmental Modelling and Software* 26 (7). Elsevier:837–44. https://doi.org/10.1016/j.envsoft.2011.02.011.

Snoek, Jasper, Hugo Larochelle, and Rp Adams. 2012. "Practical Bayesian Optimization of Machine Learning Algorithms." In *Nips*, edited by F Pereira, C J C Burges, L Bottou, and K Q Weinberger, 1–9. Curran Associates, Inc. https://doi.org/2012arXiv1206.2944S.

Soulié, Jean Christophe, and Olivier Thébaud. 2006. "Modeling fleet response in regulated fisheries: An agent-based approach." *Mathematical and Computer Modelling* 44 (5-6):553–64. https://doi.org/10.1016/j.mcm.2005.02.011.

Tobler, W. R. 1970. "A Computer Movie Simulating Urban Growth in the Detroit Region." *Economic Geography* 46 (June). Taylor & Francis, Ltd.:234. https://doi.org/10.2307/143141.

Vermorel, Joannes, and Mehryar Mohri. 2005. "Multi-Armed Bandit Algorithms and Empirical Evaluation BT - Machine Learning: ECML 2005." In *Machine Learning: ECML 2005*, 437–48. Springer. https://doi.org/10.1007/11564096_42.

Watson, Geoffrey S. 1964. "Smooth regression analysis." *The Indian Journal of Statistics* 26 (4):359–72. https://doi.org/10.2307/25049340.

Wilen, James E., Martin D. Smith, Dale Lockwood, and Louis W. Botsford. 2002. "Avoiding surprises: Incorporating fisherman behavior into management models." *Bulletin of Marine Science* 70 (2):553–75.

Wolpert, David H., and William G. Macready. 1995. "No free lunch theorems for search." Santa Fe Institute. https://doi.org/10.1145/1389095.1389254.

Wu, Charley M., Eric Schulz, Maarten Speekenbrink, Jonathan D. Nelson, and Björn Meder. 2017. "Mapping the unknown : The spatially correlated multi-armed bandit." *bioXriv*, April. Cold Spring Harbor Laboratory, 2–7. https://doi.org/10.1101/106286.

Yu, Run, PingSun Leung, Minling Pan, and Steven F Railsback. 2012. "Introduction of the Agent Based Fishery Management Model of Hawaii's Longline Fisheries." In *Proceedings of the Winter Simulation Conference*, 369:1——369:2. WSC '12. Winter Simulation Conference. http://dl.acm.org/citation.cfm?id=2429759.2430243.