SI Appendix 1

Design of the Model

This Supplemental Information is intended to describe in some detail the design of the model and the rationale for the particular computational methods that were used. The lobster model is an agent-based model. It simulates how Maine lobster fishers, the agents in the model, make their everyday decisions concerning where to put their traps and how they decide whether to cut or ignore the traps of others. The model is based on a map drawn on a 70 x 70 grid. The map describes a patchy biophysical environment, i.e., lobster distribution, depths, bottom types, and other factors important to fishers. Fishers move their traps from place to place in this environment in order to catch lobsters.

The difference between the lobster model and other agent-based models typically used in ecology (1) and in social science (2) is that the decision rules of agents in the model evolve instead of being specified by the programmer. Fishers adapt their fishing strategies through continuous interaction with one another and their surrounding biophysical environment.

In the following sections, Holland's (3) learning classifier system (LCS), the computing mechanism that allows agents/fishers to learn how to fish, is explained. Then, the implementation of the classifier system (CS) in the lobster model is shown. Last, the architecture of the model is illustrated.

1. Classifier system with genetic algorithm: the learning mechanism

Since Holland (3) introduced LCS, the idea has inspired much research into "genetics-based" machine learning (4). The CS mimics human beings' learning process. When a person sets out to accomplish a certain task in an unknown environment, his/her mental model of the new world is blank at first. Initially, in the absence of knowledge or experience in the environment the actions the person takes may be almost random. Quickly, however, he/she receives feedback from the environment and adjusts his/her mental model accordingly. The LCS mimics this process. It integrates a rule-based system with reinforcement learning and genetic algorithm-based rule discovery. In an LCS, an agent learns how to take actions by interacting with a partially known environment. For each action, the agent gets feedback in the form of numerical rewards; these are used to guide the evolution of the agent's behavior. In an LCS, the agent's behavior is represented by a set of rules, the classifiers. Because of the diversity of the classifiers, genetic algorithms are used to discover new rules by recombining better classifiers or alternating some blocks of existing classifiers.

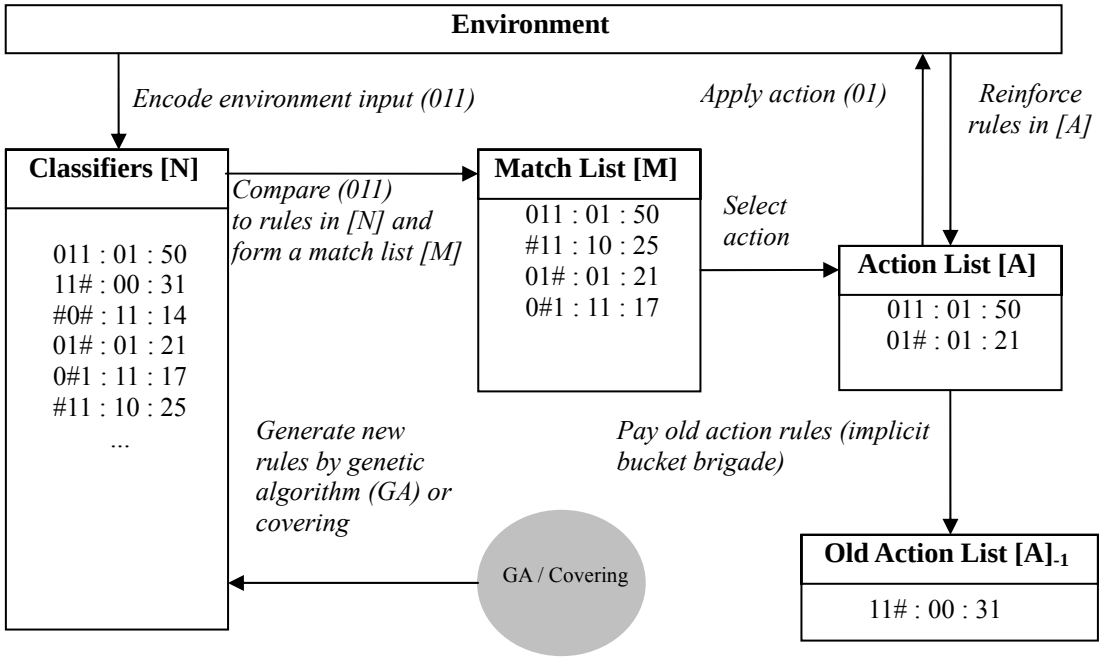
Since the original framework (5) was complex and people found it difficult to realize the envisaged behavior (6), Wilson (7) presented the "zeroth" classifier system (ZCS). The ZCS keeps much of Holland's (5) original idea but simplifies it by removing several elements: the message list, rule bid, and # sign in action part (all described below). ZCS is easier to understand and the performance is improved. Thus, in our model, the daily decision-making system of fishers is implemented using the basic framework of ZCS.

1.1 Logic of classifier system in reference to Wilson's ZCS

In this section, the logic of the CS, as implemented in Wilson's ZCS, is introduced.

When an agent receives an input from the environment, it encodes the input and sends it to the CS, which will in turn output an action to the agent. The agent performs that action and that action usually changes the environment.

The CS has an internal working memory space ([N] in SI Fig. 10), called its rule-base, which consists of a population of *n* condition-action rules or classifiers. Each rule is a string of 0s, 1s, and #s, and consists of a triplet in the form of "condition : action : weight." The # sign acts as a wildcard, which allows generalization such that 0#1 matches both 001 and 011. A rule that uses no #s is very specific to a particular set of circumstances; a rule that uses many #s tends to be general and might apply to a broad set of circumstances. For any particular set of conditions, several rules, some general and one or two particular, might match. The action part of a rule is similar to the condition except no #s are included. Both condition and action are initialized randomly. A numerical weight that represents how fit the rule/classifier is, i.e., how well it has performed in the past, is attached to each rule/classifier. ("Rule," "decision rule," and "classifier" are used interchangeably.)



Note: Classifier format is condition : action : weight.

SI Fig. 10. Schematic of ZCS (modified from ref. 8)

SI Fig. 10 shows the work flow of the ZCS. When receiving an encoded input from the environment, ZCS scans all rules in the rule-base, [N]. Any rules in [N] that match the input from the environment form a match list, [M]. It is important to note that the use of the # sign means that multiple rules, some very specific, some more general, may match any given set of environmental circumstances. From [M], one rule with relatively high weight is picked

using a weighted random choice (roulette), and all rules in [M] that propose the same action as the one picked form the action list, [A]. Internally, a fixed fraction (β in SI Table 6) of the weight of each classifier in [A] is deducted and the total amount of the fraction is put into a "bucket." The system records all action rules in the previous time step and puts them into the old action list, $[A]_{-1}$. If $[A]_{-1}$ is not empty, all rules in $[A]_{-1}$ will share equally (are equally rewarded with) the proportion (γ in SI Table 6) of weights in the bucket. Externally, the chosen action will be applied to the environment. Later a numerical reward, positive or negative, from the environment is sent back to the CS, and this time it is evenly distributed to all rules in [A]. Then a "tax" is imposed on all rules in [M] but not in [A]; i.e., the weight of all matching rules not chosen in [A] is deducted by a fixed proportion (τ in SI Table 6). This "tax" gives the fitter classifiers a greater chance of being chosen the next time. Finally, rules in [A] are moved to $[A]_{-1}$.

In Wilson's ZCS, two mechanisms generate new rules, a covering operator and a genetic algorithm (GA) (9). On the receipt of the environmental input, if the CS finds no matching rules in the rule-base, or the matching rules are too weak, i.e., if the total weight of the matching rules is below some proportion (θ in SI Table 6) of weight of all rules in the rule-base, the covering operator generates a new rule by randomly sprinkling a certain number of #s (wildcards) into the condition string. This rule is then combined with a randomly generated action, yielding a new, general rule. The weight of the new rule is the average weight of all rules.

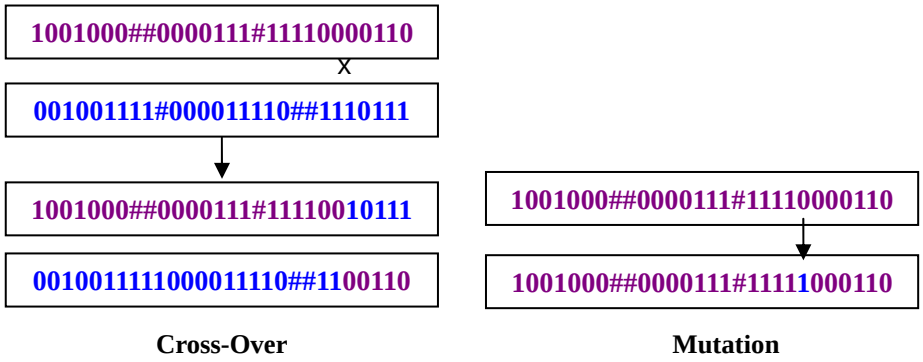
SI Table 6. Summary and explanation of parameters of ZCS

Symbol	Description
β	(β * (the weight of classifiers in [A])) is deducted from classifiers in [A] and goes to the bucket
γ	(γ * bucket) is shared by classifiers in $[A]_{-1}$
τ	Tax rate that is imposed on classifiers in [M] but not in [A]
ρ	The probability of invoking cross-over
θ	Covering is triggered if the total weight of [M] is less than (θ * total weight of [N])
wildcard	The percentage of the bit string that is changed to #s when covering
mRate	The probability that mutation is triggered
mPercent	The percentage of the bit string that is flipped when mutation is triggered
n	The total number of classifiers in [N]

1.2 Genetic algorithm

Holland created his famous GA in 1975. The GA is an evolutionary computing technique based on the mechanisms of natural selection and genetics; i.e., the GA applies Darwin's principle of the survival of the fittest among computational structures with the stochastic processes of gene mutation, recombination, etc. The central idea of GA is to search a problem space by evolving a population of solutions such that better, or fitter, solutions are generated over time. Eventually the population of candidate solutions adapts to the problem.

In GA, two major genetic operators are applied, cross-over and mutation. SI Fig. 11 shows how they work. The first is cross-over, which is also called recombination. The genetic materials (bits) are directly exchanged between two parents to produce two new individuals. The chromosome (bit string) is transposed between the two parents. The genetic information of children has some of the characteristics of each parent. Cross-over can happen at any random site along the bit string. The second major operator is mutation, which flips bits at random points along the bit string. The number of bits that are flipped is predetermined by a parameter (mPercent in SI Table 6).



SI Fig. 11. Genetic algorithm's operators: cross-over and mutation

In a CS, when the GA is triggered, two classifiers are used as parents to do cross-over and one classifier as single parent to do mutation. The parents are identified based on a roulette selection based on their weights from [N] (SI Fig. 10). The weight of a child generated by cross-over is the average weight of its parents. An offspring from mutation inherits the weight of its parent. The offspring replace the weak classifiers in the population that are chosen via roulette selection on the inverse of their weights. In this way, the population size remains constant (n in SI Table 6), and favorable classifiers have a greater chance to reproduce.

The GA plays a very important role in the CS. After each input-output cycle, the CS decides whether to apply a GA to [N]. The frequency of use of the GA is determined by two parameters, ρ and mRate (SI Table 6). Once a GA is invoked, it generates new rules, and weeds out weak rules. Thus the system gets a chance to try new rules, and evolves through the trial-and-error processes.

2.The design of the trap placement classifier system of the lobster model

Section 2.1 describes the environmental information that agents need to make decisions, and section 2.2 describes all possible actions they might take. The architecture of the CS is discussed in section 2.3. The calculation of feedbacks that enforce rules in CSs is discussed in section 2.4.

2.1 The environment of the model

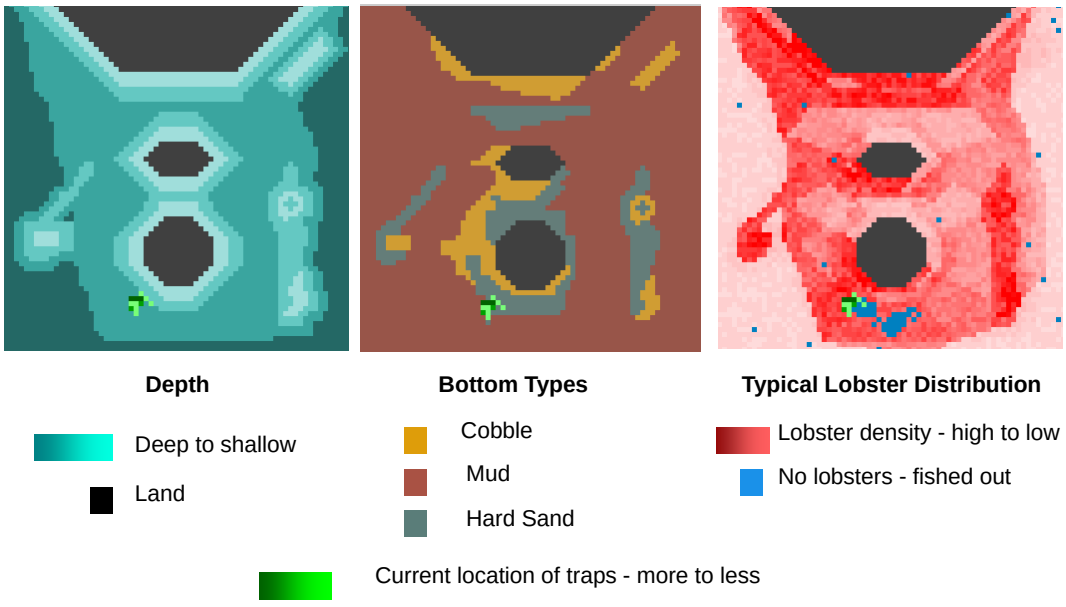
Fishers interact within a complex social-ecological system. Seasons, lobster habitats, and the spatial heterogeneity in the distribution of fishing efforts affect the distribution of lobster, which in turn affects fishers' behaviors. On the basis of extensive experience on the part of

two of the authors and informal conversations with fishers in the Gulf of Maine (GOM), we listed all environmental conditions that are important to fishers' decisions.

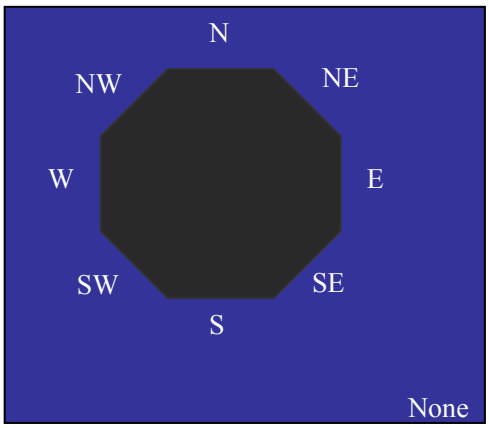
Ecological environment

In GOM, most lobster fishers actively fish in summer, fall, and early winter and take a break in late winter and spring. Water temperature changes with seasons and affects the catchability of lobsters. When the surface water is warm (late summer and early fall), lobsters are more likely to be caught near shore, because they are more active and feed more. In the model, there are eight months for each year, from July to the next February, and each month has 30 days, so one year has 240 days. Lobster catchability is a function of water temperature, which is a function of month.

The spatial-ecological information, such as water depth, bottom types, and orientation, determine lobster distribution and catchability as well. Water temperature is also a function of depth. In summer, the shallow water warms up more quickly than deep water, and in winter, it cools down more quickly. So the catchability is relatively high in shallow water in warm weather and in deep water in cold weather. Depth, bottom type (SI Fig. 12), and exposure to prevailing winds and seas (SI Fig. 13) play important roles in lobster distribution. Lobsters prefer a cobble to sandy to muddy bottom. The larvae of lobsters tend to settle in areas exposed to the south and southwest because of the currents and winds in GOM. Once they settle, they tend to be very sedentary until nearly mature and catchable. In the model, lobsters are assumed sedentary throughout their lives although it is known to be false, especially in larger, older lobsters. To the extent that migration patterns are known, catchable lobsters tend to move into deeper water in the winter. The model captures the resulting increase in catchability at that time by a different mechanism: the temperature-driven changes in metabolism described above. So in the model, we have four depths, three bottom types, and nine orientations (including None). These attributes are recorded for each cell on the grid. All of them contribute to the patchiness of the distribution of lobster.



SI Fig. 12. Maps of the biophysical environment (Screen shots are in year 10, with 10 fishers and 20 traps.)



SI Fig. 13. Orientation (Water that is far away from land has no orientation.)

SI Table 7 lists the weight of lobster distribution for each combination of the environmental information. The number of lobsters that will be put in one cell is calculated as follows. First, calculate the weight of the cell according to SI Table 7. For example, if the depth of the cell is 1, the bottom type is sandy, and it faces southwest, then the weight of the cell is 2 (part *a* of SI Table 7) * 6 (part *b* of SI Table 7) = 12. After calculating all cells' weights, we normalize them. The number of lobsters that will go to each cell is the normalized weight times the total number of lobsters on the map. The typical lobster distribution in SI Fig. 12 shows the result of the distribution calculation.

SI Table 7. Weights of lobster distribution. (a) The lobster distribution weight for each combination of depth and bottom types: 1 is the shallowest and 4 is the deepest. (b) The lobster distribution weight for each orientation.

a

Depth	Cobble	Sandy	Muddy
1	4	2	2
2	3	4	3
3	2	3	4
4	1	2	4

b

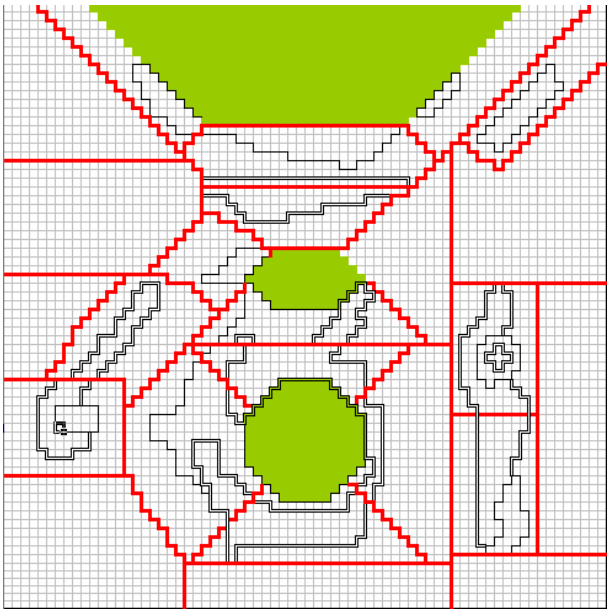
N	NW	W	SW	S	SE	E	NE	None
2	3	5	6	5	4	3	2	1

At the end of each year, lobsters are replenished on the map mainly by two methods, which can be switched by a parameter. One uses a logistic model to calculate the number of lobsters recruited every year. In this circumstance, lobsters could be overfished if fishing efforts go beyond the limit that the system can sustain. Except in the case of the Gordon-Schaefer model (11), the logistic recruitment function is not used in this article because our focus is on the fine-scale, short-run aspects of competition. In the other scenario, the total number of lobsters at the beginning of every year is kept constant in order to allow fishers to have stable mental models of the fishing environment. The other reason to keep the lobster recruitment

constant is that there is no obvious evidence in GOM that the distribution of lobsters or size of the population changes from year to year in response to its own population size.

Location is another piece of important information for fishers. In the real world, fishers have global positioning systems (GPSs) and know exactly where they are fishing. However, their memory of past fishing is not nearly as specific as a GPS; nor would memory that specific be useful, given variations in very local lobster abundance due to both recruitment and fishing. Additionally, but not as important, if the 4900 cells were built into the CS, the system would need much more memory for calculation, which would slow down the performance of the simulation and may make the model infeasible for a typical desktop or portable computer. So, in the model, the map is divided into 24 ecological zones (SI Fig. 14), in which most cells share the same ecological characteristics. Fishers in the model know in which ecological zones they are fishing and are able to decide in which zone to put their traps next. The total number of lobsters that are put in the whole map is constant every year, and the lobster distribution is relatively stable. In order to consider the movement of lobsters, the lobster distribution weight of each ecological zone is changed slightly every year; i.e., the weight shifts around the value calculated as above by a certain percentage, e.g., ±20%, which is a parameter of the model. Then we normalize their values and distribute the lobsters.

In summary, the explicit ecological information in the model consists of month, depth, bottom type, and ecological zone. Additionally, the orientation of the area to prevailing winds and water temperature are implicit in this information. For example, time of year includes information about water temperature, and ecological zones include information about orientation. But fishers do not know the relationship between lobster distribution and the ecological information. They learn it by fishing in different situations. In other words, their mental models of the distribution of lobster are adjusted gradually by feedback from the environment. They do not keep detailed information about the historical abundance in each cell because it is burdensome and because those data age rapidly.



SI Fig. 14. Twenty-four ecological zones of the model. Ecological zones are divided by red lines. Single and double gray lines show the contours of different bottom types. Green areas are land.

Social Environment

In the real world, experienced fishers have very good knowledge about lobster habitats and the way season affects lobster behavior; so in the model, fishers are assumed to have perfect information about these longer-term "constants" in the ecological environment. They are not given perfect knowledge about the current location of lobsters because the patchy distribution of lobsters changes from year to year due to recruitment variability and, especially in the short term, in response to fishing activity. Fishers' knowledge of other fishers is especially incomplete. Fishers are able to guess other fishers' performance by monitoring their neighbor fishers, chatting with fishers in coffee shops, etc. A global message board does not exist to broadcast all fishers' catch rates and the locations where they put their traps. In the model, this information flow among fishers is controlled by a parameter. In order to compare different situations, one option gives fishers perfect information of others' catch rates. The other option is closer to the real world, namely fishers know their neighbors or friends better than others who are far away from them, either in social or geographical distance. In the model, fishers broadcast their catch rates every day to everybody else. When fishers receive the broadcast, they discount the catch rate by a percentage, which indicates how much they trust the fisher who sends it. In the real world, fish and deceive is often the rule. The percentage is a function of the number of times two fishers encounter when they fish. Thus social information is not symmetric and far from perfect; i.e., individuals have access to different information about one another and they interpret the information they do share in different ways.

When fishers in the model make decisions, the social/competitive conditions they need to consider are as follows:

"Is the catch of the current trap the best among all his or her traps?"
"Is the catch of his or her own best trap better than other fishers' best?"
"Is the catch of his or her current trap better than other fishers' best?"
"Has his or her own average catch rate changed?"
"Has the average catch rate of all fishers changed?"

2.2 Possible actions of fishers

SI Table 8 lists all possible actions after a fisher hauls a trap.

SI Table 8. Possible actions for trap placement

Actions	Meaning
Stay	Put current trap near where it was, i.e., within a 3 x 3 neighborhood around its current location
Own best	Put current trap near own best trap, i.e., within a 3 x 3 neighborhood
Imitate	Put current trap near a trap of the fisher thought to be the best, also within a 3 x 3 neighborhood
Explore	Put current trap in another area that was historically productive at this time of the year

When trap A is put near trap B, the nearest vacant cell around B will be searched, usually within a 3 x 3 range. If there is no vacancy found in the small range, a larger range of area will be searched. When a fisher decides to imitate another's best, he/she has no idea of the location of the best trap and puts his/her current trap next to the nearest trap of the best fisher.

Imitation makes the learning mechanism of the model different from other LCS models in which agents develop new rules (learn) only through a reinforcement mechanism and or through cross-over and mutation carried out in the GA. By imitating, fishers are able to improve their mental models and their ability to adapt to a complex environment. Imitation is a critical modification of the usual mechanisms of an LCS and is the principal source of the social arrangements that emerge in the fishery.

Fishers acquire experience by imitating. That experience is translated into a new rule with the current conditions of the trap he/she hauls as the condition part, and the current action, i.e., where he/she put it, as the action part. The weight of the new rule is assigned later as the number of lobsters caught using this rule. Next time, when the fisher meets the same condition, the new classifier will show up on the match list and compete with other matching rules. In other words, the fisher could imitate again or explore using the new rule. (See SI Fig. 15 for more information.)

2.3 The architecture of classifier systems

Lessons we learned from the design

Once we listed all the conditions and actions that were relevant to the conduct of the real fishery, we expected the design of the LCS to be straightforward. We created a large classifier that was able to contain all possible combinations of conditions and actions. SI Table 9 lists the first design.

SI Table 9. The original format of the classifiers

Condition	Action
(1) Time	(1) Stay
(2) Bottom	(2) Imitate
(3) Depth	(3) Own best
(4) Orientation	(4) Go to a specific bottom type and depth in a specific ecological zone
(5) Performance Comparisons, i.e., the competitive comparisons listed above	
(6) Ecological zone	
(7) Steaming[a]	
(8) Spread of own traps[a]	

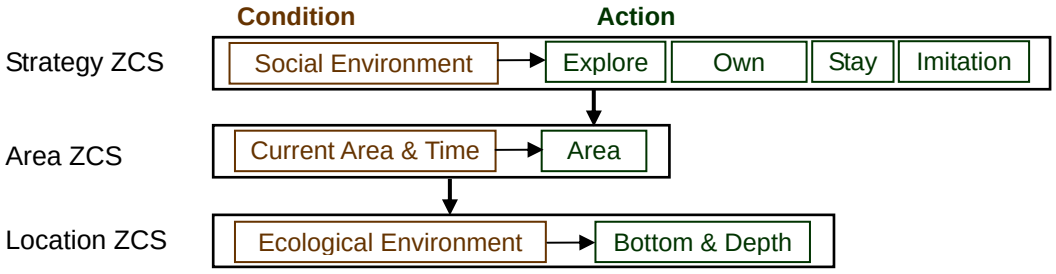
[a] These two conditions were added in the first design to consider the steaming (travel) costs of each move, but they are not used in later versions of the model.

The number of combinations of all conditions and all actions is a huge number, 14 trillion; the search space defined this way is too large to make an effective search even given the frequent number of tests (trap hauls) each fisher conducts. As a result, we did not observe learning on the part of agents. Consequently, we modified the model in order to reduce the search space.

A hierarchical approach to the trap-placement classifier systems

When we, human beings, make decisions in a complicated situation, we normally consider first the range of relevant conditions and give higher priority to major decisions. After that, other conditions are considered. The new design was inspired by this idea (10). When a fisher hauls a trap, he/she knows how many lobsters were caught, and how that number compares to (at least some) other fishers. If satisfied with the catch, the fisher might stay there for several days or put the trap near his/her own best trap; otherwise, he/she can imitate another fisher's trap or go to another location that was historically productive in similar circumstances.

SI Fig. 15 shows the different levels of decision making. Fishers use the first-level CS, the Strategy CS, to consider social conditions to decide what fishing strategy to use, imitation, exploration, stay, or move to own best trap. If imitation, stay, or move to own best trap is decided, fishers search locally around the other trap looking for a vacant spot to put their traps. If exploration is chosen, fishers use the second-level Area CS, in which they use time and current ecological zone as inputs to decide what ecological zone to move their traps to next. This CS connects the season with different ecological zones and accumulates experience of what zone is more productive in each month. Then the third-level Location CS is triggered. At this point, the fisher considers the bottom type and depth to go to within the chosen ecological zone. The Location CS connects the season with different bottom types and depths.



SI Fig. 15. Hierarchical decision making

SI Table 10 gives a detailed design of each CS and number of bits of the classifiers.

SI Table 10. Detailed design of each classifier system in the hierarchical approach

CS#1 – Strategy CS:

a1

Conditions	Results	# of bits
(1) Current catch vs. own best	0~1	1
(2) Current catch vs. other's best	0~1	1
(3) Own best vs. other's best	0~1	1
(4) Own current avg. CR – own previous avg. CR	0~1	1
(5) Avg. coffee shop CR vs. own avg. CR	0~1	1
(6) Current avg. coffee shop CR vs. previous coffee shop CR	0~1	1
Total # of bits		6

a2

Actions	
(1) Stay	
(2) Own best	
(3) Imitate	
(4) Explore	
Total # of bits	2

CS#2 – Area CS:

b1

Conditions	Results	# of bits
(1) Current location	0~23 (totally 24 ecological zones)	5
(2) Month	0~7 (totally 8 months)	3
	Total # of bits	8

b2

Actions	Results	# of bits
Ecological zone	0~23 (totally 24 ecological zones)	5

CS#3 – Location CS:

c1

Conditions	Results	# of bits
(1) Time	0~7	3
(2) Depth	0~3	2
(3) Bottom type	0~2	2
	Total # of bits	7

c2

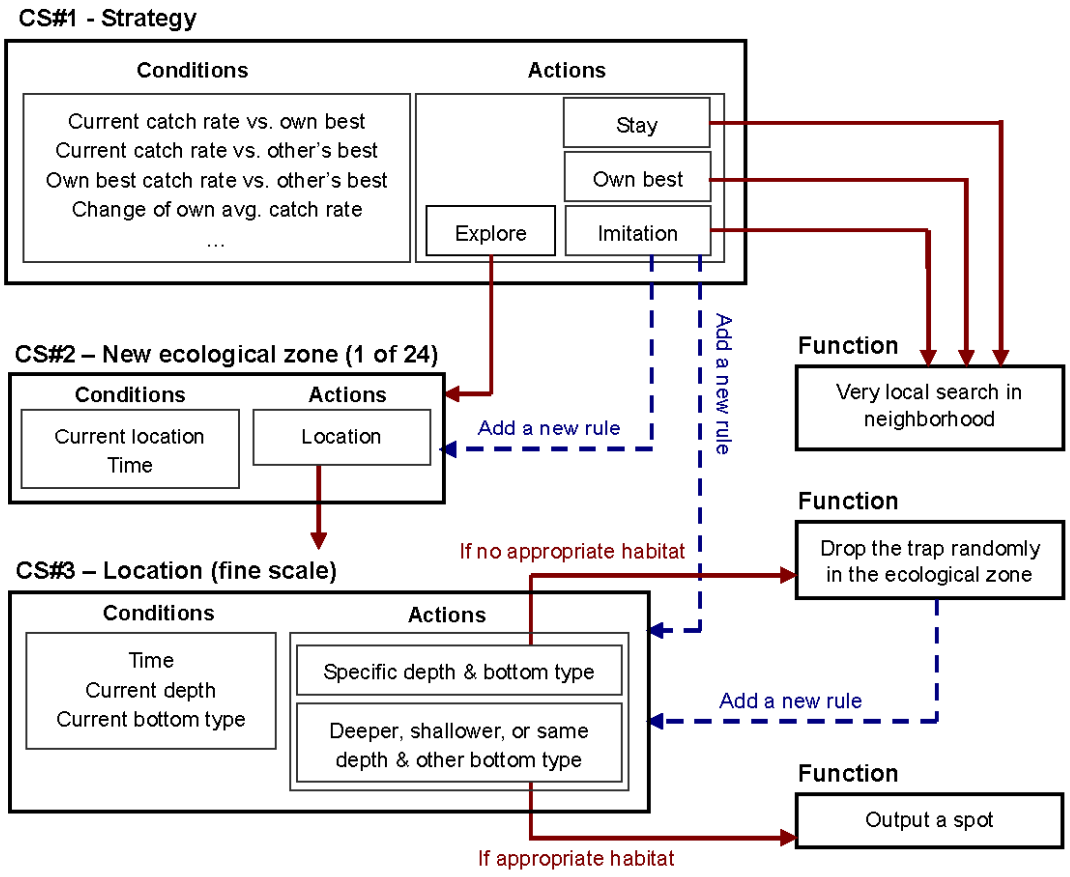
Actions		
(1~12) Combinations of different depths and bottom types		
(13) Go deeper & other bottom type		
(14) Go shallower & other bottom type		
(15) Same depth & other bottom type		
	Total # of bits	4

SI Table 11 lists the number of possible rules of each CS. Most trap-placement decisions only search the first-level Strategy CS, because once the fisher has decided to move to his/her own best trap or to imitate or to simply keep the trap where it is, any further decisions about area, depth, and bottom type are irrelevant. As a result, the search space for all these strategies has only 2916 possible rules. Generally, far less than 25% of decision making needs to search all three levels of the hierarchy. Thus, the hierarchical approach reduces the search space dramatically (compared with 14 trillion, for example) and allows the rapid evolution of effective rules.

SI Table 11. Number of possible rules in each classifier system

	# of possible conditions	# of actions	# of possible rules
Strategy CS	3^6 = 729, 3 ~ (0,1,#)	4	2,916
Area CS	25*9 = 225, 25 ~ (24 ecological zones, #), 9 ~ (8 months, #)	24	5,400
Location CS	9*4*5 = 180, 9 ~ (8 months, #), 4 ~ (3 bottom types, #), 5~(4 depths, #)	15	2,700

SI Fig. 16 shows how learning through imitation, i.e., outside the GA learning mechanism, fits in the model. When imitation happens, fishers put their current traps near others' best traps. Once they find a vacancy, they know the characteristics of the location where they put the traps, i.e., the ecological zone, bottom type, and depth. They use that information to create two new classifiers, which are added into Area CS and Location CS, respectively. When exploration takes place, fishers consult the second and third levels of CS and get the location, the ecological zone, the bottom type, and the depth. Then they bring their traps, go to the specified ecological zone, and try to locate the proper bottom type and depth. Sometimes they cannot find a certain bottom type and depth in that ecological zone. In this case, they drop their traps randomly in that zone and test if they can catch anything. At the same time, they create a new rule with the location information in the third-level CS.



SI Fig. 16. The architecture of the classifier systems of the model. Red lines indicate the flow of decision making, and blue lines indicate how learning, except genetic algorithm, happens in the model.

2.4 Feedbacks

The calculation of feedback is important for the CS. Feedback determines whether the system can evolve or not and should be a function of all elements in the condition part. Otherwise, the effectiveness of classifiers cannot be differentiated through feedback. For example, we have two classifiers, $c_1c_2c_3$:a and $c_1c_2c_4$:a. If the feedback = $f(c_1, c_2)$, the third condition in both classifiers will be ignored and both classifiers can be combined into c_1c_2:a.

In the model, the number of lobsters fishers catch is the principal feedback from the environment. It reflects the lobster distribution and the fishing effort, which is a function of time, bottom type, depth, ecological zone, and trap distribution. But sometimes, it is not sufficient. For example, the purpose of Area CS is to differentiate which ecological zone is more productive in each month. The total catch in ecological zones with deep water is much less than catches with shallow water, but in the winter, catchability in deep water is better than in shallow water. In this case, feedbacks that only consider the number of lobsters caught are not correct. Instead, how good the seasonal catch is in each ecological zone at a certain time needs to be considered. The feedback used in each CS is as follows:

$$F_{\text{Strategy}} = n - C / N, \quad [1]$$

$$F_{\text{Area}}(a) = n - (C_{\text{not in } a} / N_{\text{not in } a} - C_{\text{in } a} / N_{\text{in } a}), \quad [2]$$

$$F_{\text{Location}}(d, b) = n - (C_{\text{not in } d \text{ or not in } b} / N_{\text{not in } d \text{ or not in } b} - C_{\text{in } d \text{ and } b} / N_{\text{in } d \text{ and } b}), \quad [3]$$

where n is the number of lobsters caught by the current trap, C is the total catch, N is the total number of trap hauls, and a is the ecological zone, d is the depth, and b is the bottom type.

3. The design of the trap-cutting classifier system

In addition to the trap-placement decision, fishers make another daily decision about how to deal with competitors' traps that are located near theirs. They can cut those traps or they can ignore them. When they make this decision, the information they need to consider is whether the owner of the trap is their neighbor, whether the ecological zone they are in is their own territory, and whether the zone is productive. Incorporation of this decision in the model opens the door for fishers to compete by directly interfering in the actions of their competitors. The design of the Trap-Cutting CS is illustrated in SI Table 12.

SI Table 12. The design of the trap-cutting classifier system

Conditions	Description	Results	# of bits
(1) Current frequency of encounters	The frequency of encounters. Indicates whether the fisher who is encountered is a neighbor or not. Because information flow among fishers is based on frequency of encounters, fishers know the performance of those who are close to them much better. They tend to imitate them, which increases even more the number of their encounters. We use this condition to determine if the other fisher is a neighbor.	0~1	1
(2) Area	Includes information about territory and productivity of the area	0~23	5
(3) Action to me	Did the other fisher cut my trap the last time we met?	0~1	1
(4) Action to neighbors	Did the other fisher cut the trap of one of my neighbors last time he/she met one of them?	0~1	1
	Total # of bits		8

Actions	
(1) Cut	
(2) Ignore	
Total # of bits	1

The feedback of the CS is discussed in the article. Retaliation is implicitly built into the pay-off matrix. SI Tables 13a1 and 13a2 give the description of the pay-off matrix, SI Table 13b gives the example values for variables that could be used for the model, and SI Tables 13c1 and 13c2 show the instantiate matrix used in the model.

SI Table 13. Trap-cutting pay-off matrix

a1. Feedback when trap is located in OWN neighborhood

My action	Other fisher is	Other fisher cuts	Other fisher is	Other fisher ignores
Cut	**neighbor**	benefits = minor decrease in competition costs = high probability of retaliation, loss of cooperation, loss of trap	**neighbor**	benefits = minor decrease in competition costs = high probability of retaliation, loss of cooperation
	nonneighbor	benefit = larger competitive benefit because my action is likely to be reinforced by my colleagues costs = loss of trap, lower probability of retaliation	**nonneighbor**	benefit = larger competitive benefit because my actions are likely to be reinforced by colleagues costs = lower probability of retaliation
Ignore	**neighbor**	benefits = maintenance of cooperation costs = minor increase in competition, loss of trap	**neighbor**	benefits = maintenance of cooperation costs = minor increase in competition
	nonneighbor	benefits = none costs = minor increase in competition, loss of trap	**nonneighbor**	benefits = none costs = minor increase in competition

a2. Feedback in OTHERS' neighborhood

My action	Other fisher is	Other fisher cuts	Other fisher is	Other fisher ignores
Cut	neighbor	benefits = minor decrease in competition costs = high probability of retaliation, loss of cooperation, loss of trap	neighbor	benefits = minor decrease in competition costs = high probability of retaliation, loss of cooperation
	nonneighbor	benefit = minor decrease in competition costs = loss of trap, high probability of retaliation, members of other group will reinforce his/her actions	nonneighbor	benefit = minor decrease in competition costs = high probability of retaliation, members of other group will reinforce his/her actions
Ignore	neighbor	benefits = none costs = minor increase in competition, loss of trap	neighbor	benefits = maintenance of cooperation costs = minor increase in competition
	nonneighbor	benefits = none costs = minor increase in competition, loss of trap	nonneighbor	benefits = none costs = minor increase in competition

b. The following values to the variables determining the pay-off matrix

Costs of gear destruction	20
Change in competition	±30
Cost of retaliation	20
Change in cooperative benefits	±30
Group reinforcement	40

c1. Feedback when trap is located in OWN neighborhood

	Other fisher is	Cut	Ignore
Cut	neighbor	-64	-44
	nonneighbor	6	26
Ignore	neighbor	4	24
	nonneighbor	-26	-6

c2. Feedback in OTHERS' neighborhood

	Other fisher is	Cut	Ignore
Cut	**neighbor**	-64	-44
	nonneighbor	-54	-54
Ignore	**neighbor**	-26	24
	nonneighbor	-26	-6

4.Design issues

Particular aspects of the lobster model required modifications in Wilson's ZCS mechanism.

Changes in the method of covering

Covering occurs when there are no existing strong rules that match the current environment. For all CSs in the model, the action of new rules that are generated by covering is from the most similar existing rules. By similar, we mean a rule for which the bits reserved for the environment most closely match the current environment, which mimics a human being's cognitive process (10). When we are in a new situation, we connect the new features to the similar ones that we met before and we will try what we did before.

Cross-over on match list

In Wilson's ZCS, the GA does cross over on the rule-base [N] (SI Fig. 10). However, in the model, cross-over is done on the match list [M] in order to allow fishers to try new, closely related rules in every circumstance. In this case, rules in the CS converge much more quickly. When there are a large number of environmental variables and rules, [M] is more likely to contain relevant/related rules, leading to quicker convergence, whereas rules chosen from [N] are not likely to be relevant because of the breadth of environmental conditions in the model. Applying cross-over only to rules that appear in the match list limits the diversity of rules developed by cross-over. However, the process of imitation, not included in the ZCS, continually introduces new rules and tends to maintain diversity.

A variable number of rules

In Wilson's ZCS, whenever new rules are added to the rule-base, the same number of old, weak rules will be deleted. But in our model, the total number of rules in [N] is not constant during a simulation year. New rules generated by the GA and social learning are simply added into the system. At the end of the simulation year, the rules in the system are sorted by weight, and the system only keeps the first n strong rules, where n is the total number of rules allowed by the CS. The approach addresses a problem that arises because of the seasonal changes built into the model and the need for rules to be adapted to those changes. At the beginning of the simulation year, only those rules for summertime are used. During that time they become relatively stronger than wintertime rules. If we delete weak rules at that time, i.e., in the summer, useful wintertime rules will tend to be deleted and would have to be relearned every year. Similarly, summertime rules would tend to be deleted in the winter and would require relearning at the start of the next year. Consequently, we allow the number of rules to vary over the course of a model year and only delete weak rules at the end of the year.

Do error checking

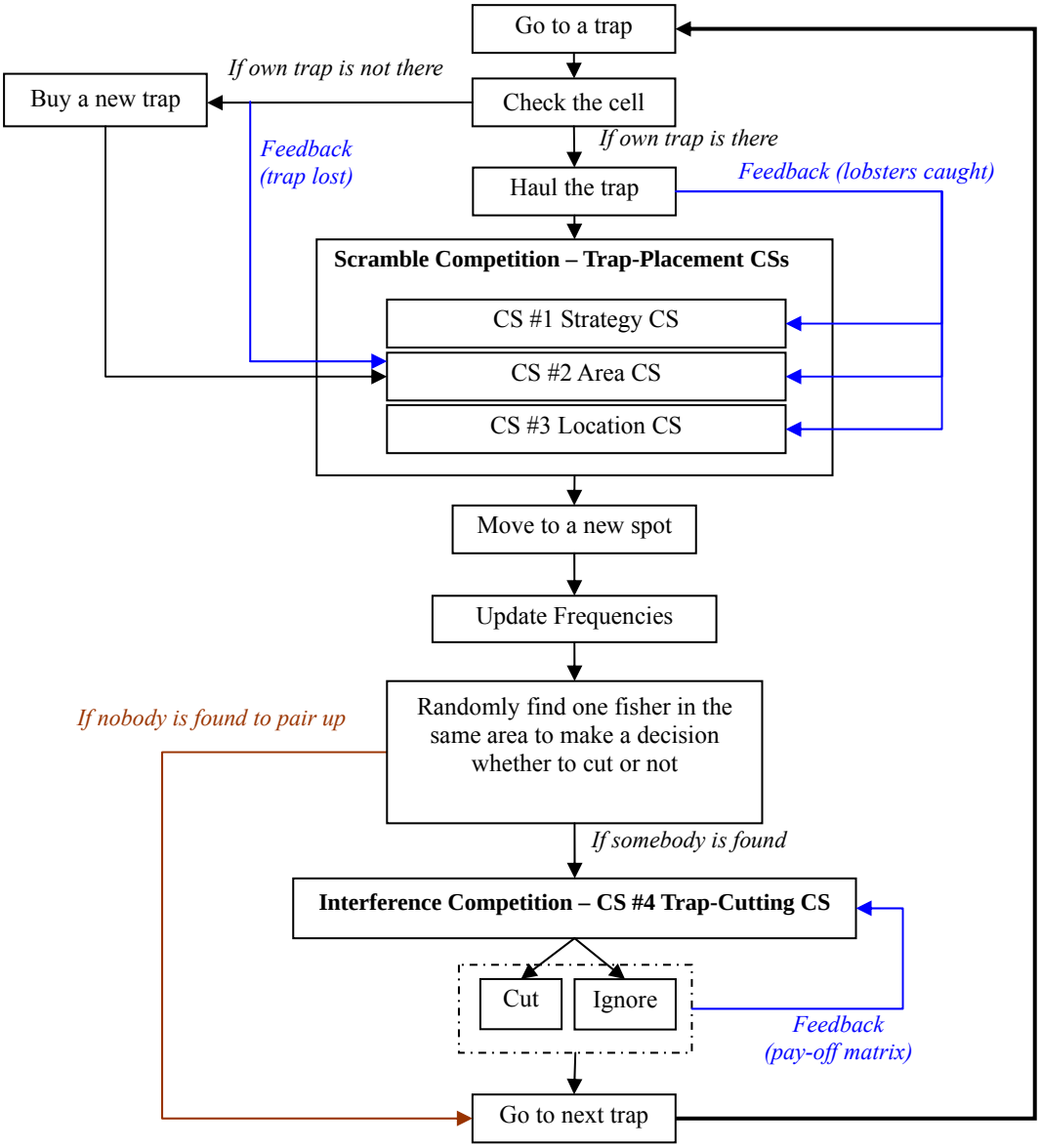
All numbers in the bit strings of the CS are encoded into binary numbers, which yields a lot of invalid rules. For example, we use 5 bits to identify ecological areas. Five bits can be used to represent up to 32 areas but we only use 24 ecological zones plus a # sign, so there are possibly 7 invalid expressions. For this reason, the CS does error checking every time new rules are generated. At the beginning of the simulation, all rules are randomly generated, and the system only keeps valid, new random rules in the rule-base. Invalid rules generated by the GA are considered a miscarriage, and are discarded by the system. The covering process is repeated until it gets a valid rule.

Remember bad experiences

Usually, rules in a traditional CS are reinforced in only one direction, i.e., the stronger the better. But in the Trap-Cutting CS, not only strong rules but also weak rules need to be remembered. For example, a rule allowing fishers to cut their neighbors' traps always gets negative feedback because of later costly retaliation. In the traditional CS, rules like this will be eliminated and later, if the same situation happens again, a new rule, possibly the same bad rule, will be added into the system. In effect, agents repeatedly forget relevant experience and repeatedly make mistakes they learned to avoid earlier. Consequently, when the Trap-Cutting CS is updated at the end of a simulation year, trap-cutting rules with very negative weights are still kept in the system but their action component is switched to ignore. Thus, fishers learn from bad experiences.

5. The architecture of the lobster model

SI Fig. 17 shows the broad decision and action flows of the model. When a fisher goes to haul a trap, he/she checks the cell to see if the trap is still there. If it is there, he/she will haul it and consult the trap-placement CSs to decide where to put it next. If the fisher finds the trap has been cut by others, he/she will buy a new trap and consult the Area CS and Location CS to get a spot to drop the new trap. After placing the trap, the fisher will look around. If there is any other fisher's trap around, he/she will decide whether to cut or ignore it. Then the fisher goes to haul the next trap.



SI Fig. 17. The architecture of the model

6. The exit and entry of fishers

Fishers' economic behavior is simulated as if they were a simple firm. They face fixed costs, daily operating costs, and steaming (travel) costs. These costs are uniform among fishers in the model. Revenues are a function of catch; product (lobster) prices are held constant.

When the model addresses long-run issues, at the end of each year, fishers in the model are allowed to quit and new fishers are allowed to enter. Because learning is a significant problem for fishers and is time-consuming, each fisher is given a bank account that initially buffers entry costs and allows him/her to weather the costs of learning. Over the longer term, the fisher's account tracks profits for the last three years. If, at any time, the account goes to zero, the fisher goes bankrupt and exits the fishery.

If the fishery is profitable, more fishers enter the fishery. At the end of year, the profits of all fishers are summed up. If they are greater than a threshold, a parameter of the model, fishers will enter the fishery. The number of new fishers entering is decided by how big the difference between the threshold and the total profits is. New fishers incorporate rules of their parents, which are randomly picked from the old fishers, into their own CSs. This system mimics apprentice processes and becomes another way that fishers acquire decision rules.

7. The parameters that are important to the model

In SI Table 14, all parameters that are important to the model are listed.

SI Table 14. Parameters in the model

Parameters	Description
fisherNum	Total number of fishers if the population of fishers is a constant, or the initial number of fishers if entry and exit are allowed.
maxLobsterPerCell	Carrying capacity of lobsters in each cell
trapCapacity	The maximum number of lobsters each trap can catch
shiftPercent	The annual maximum range of variation in lobster numbers in each ecological area
numStringPerFisher	The number of traps each fisher has
numStrMovePerDay	The number of traps each fisher moves every day
unitRevenue	The ex-vessel price of lobster
maxStringPerCell	The maximum number of traps each cell can hold
birthThreshold	If the total year profits of the fishery exceed the birthThreshold, new fishers will enter fishery
dayFixedCost	The fixed cost for each day
maxNumCycle	The maximum number of years the simulation will run
gasPrice	The gas cost of moving a trap one cell on the map
info	Indicates different types of social information
startupFund	The initial amount of money in the fisher's bank account

References

1. Bousquet F, Le Page C (2004) *Ecol Model* 176:313–332.

2. Axelrod R (1997) *The Complexity of Cooperation: Agent-Based Models of Competition and Collaboration* (Princeton Univ Press, Princeton, NJ).

3. Holland JH (1976) in *Progress in Theoretical Biology*, vol 4, eds Rosen R, Snell FM (Plenum, New York) pp 263–293.

4. Goldberg DE (1989) *Genetic Algorithms in Search, Optimization and Machine Learning* (Addison Wesley, New York).

5. Holland JH (1986) in *Machine Learning: An Artificial Intelligence Approach*, vol 2, eds Michalski RS, Carbonell JG, Mitchell TM (Morgan Kauffman, San Mateo, CA) pp 48–78.

6. Wilson SW, Goldberg DE (1989) in *Proceedings of the Third International Conference on Genetic Algorithms*, ed Schaffer JD (Morgan Kaufmann, San Mateo, CA) pp 244–255.

7. Wilson SW (1994) *Evol Comput* 2:1–18.

8. Flake GW (2000) *The Computational Beauty of Nature: Computer Explorations of Fractals, Chaos, Complex Systems, and Adaptation* (MIT Press, Cambridge, MA).

9. Holland JH (1975) *Adaptation in Natural and Artificial Systems* (Univ of Michigan Press, Ann Arbor).

10. Holland JH, Holyoak KJ, Nisbett RE, Thagard PR (1986) *Induction: Processes of Inference, Learning, and Discovery* (MIT Press, Cambridge, MA).

11. Clark CW (1976) *Mathematical Bioeconomics* (Wiley, New York).