Requirements

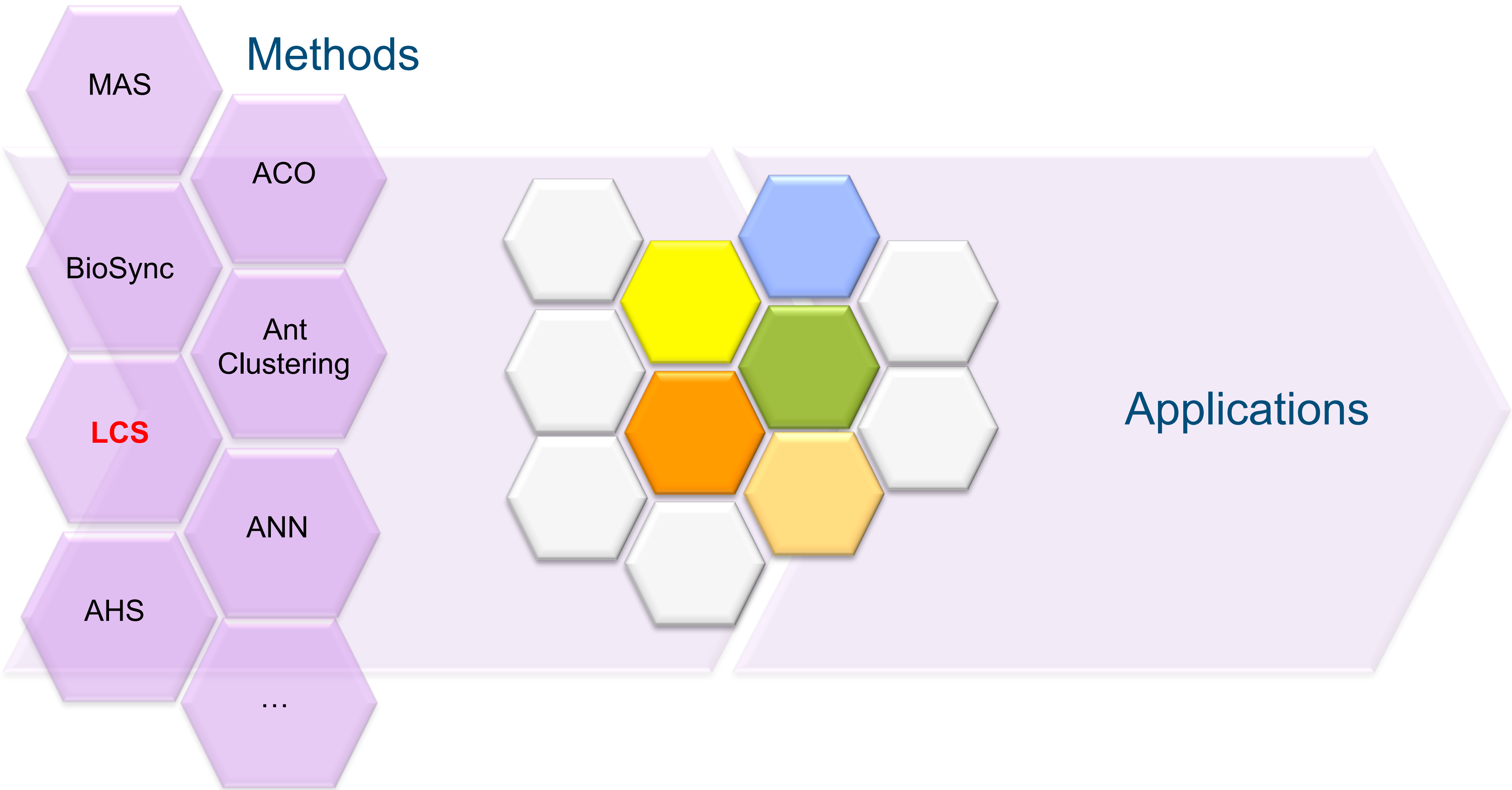
...

Goals

Insight into an example of machine learning used in OC systems

Content

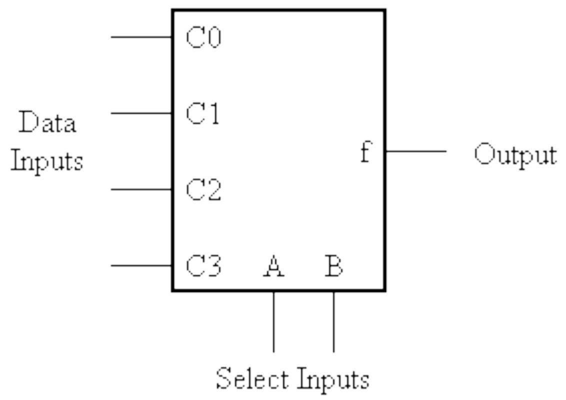
- ❑ Examples
- ❑ LCS overview
- ❑ ZCS: Zeroth-level classifier system
- ❑ XCS: Accuracy-based classifier system
- ❑ Application example: Organic traffic control (OTC)


Methods
MAS
ACO
BioSync
Ant Clustering
LCS
ANN
AHS
…
Applications

- Machine learning techniques are a promising approach for self-optimization in organic computing systems.
- How can computers be programmed so that problem solving capabilities are built up by specifying "what is to be done" rather than "how to do it"? (Holland, 1975)
- Major issues:
  - How can the system react in unforeseen situation?
  - How can the system automatically improve its performance (if possible) at runtime?
  - Overall: Flexible and autonomous reaction to changes of the environment and/or the system itself are desirable.
- One example: Learning Classifier Systems

Example 1: k-multiplexer

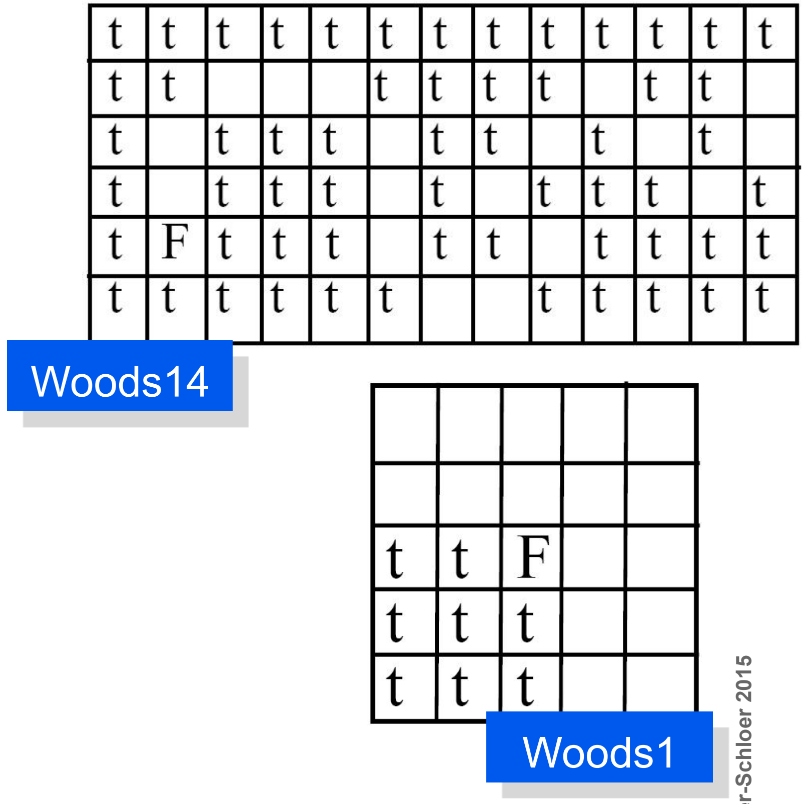
- The k-multiplexer is a Boolean function with k variables.
- The k variables consist of m address bits and 2^m data bits.
- Number of inputs: $k = m + 2^m$
- Function: Return the data bit (out of 2^m) that is specified by the address (m bits).
- Example for m = 2, k = 6
- Input: 01 1001
- Address 01b points to bit 1 of the data bit-string (starting to count from the right to the left; bits numbered $0..2^m-1$)
- Output: 0



- Question: Can we build an agent that – given the MUX as a black-box – learns the underlying Boolean function?
- One idea to build such an agent:
  - Agent proposes output for a given input.
  - Agent proposal is based on an internal set of rules, e.g. one rule could look like 01 1001 : 0
  - After the proposal, agent receives feedback about the proposal.
  - Feedback is usually called reward; high/low reward for good/bad proposals
  - Agent aims at maximizing its reward.

ISE SRA

- Example of an *Animat* problem
- Basis: Rectangular toroidal regular (n x m)-grid
- Each grid cell may contain a tree (t), food (F), or it may be empty.
- Food and trees fixed per instance
- Animat/agent/robot is initially randomly placed on empty cell.
- Walks around, looking for food.
- In each step agent can go to one of the eight neighboring cells but to empty and food cells only.



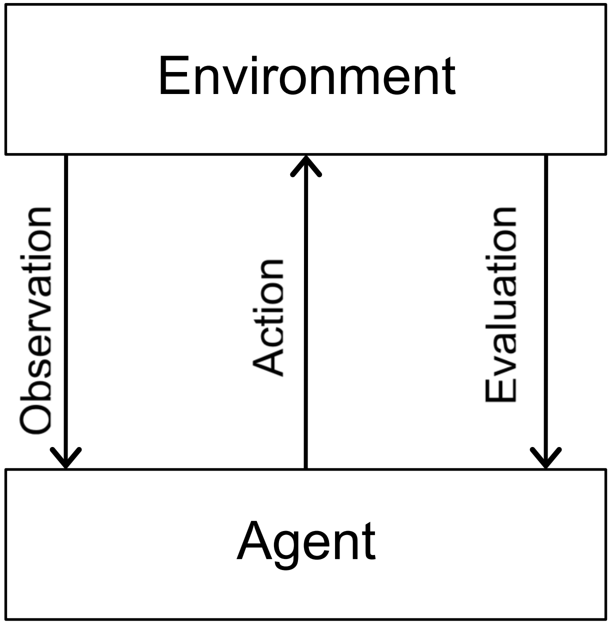
Woods1: Optimal average number of steps to reach food: 1.7 steps (Bull & Hurst, “ZCS redux”)

- Question: Can we build an agent that can efficiently find food “in the Woods” without global knowledge?
- One idea to build such an agent:
  - Suppose the agent can “see” the eight surrounding cells
  - Based upon this perception it has to decide where to go next.
  - Reward is paid once the food is found.
- One approach to building such agents is the use of a Learning Classifier System.

- The k-multiplexer and the Woods-scenario are representing two important problem classes:
  - Single-step problems, e.g., the k-multiplexer
    - On each input/action there is an immediate feedback: right or wrong
  - Multi-step problem, e.g., the Woods scenario
    - Feedback about the quality of actions may be delayed.
    - Multiple steps are necessary to reach the goal.

- Obviously we could solve the k-multiplexer by letting the agent enumerate all input states (k bits).
- When *k* becomes large: Space and time complexity are too high.
  - Enumeration is exponential in time w.r.t. *k.*
  - Enumeration is exponential in space w.r.t. *k.*
- Problematic: Reward may not be received immediately, if agent interacts with the real world: Enumeration possible?
- So, we are looking for a heuristic that
  - Produces high quality results (yet possibly suboptimal) in much less time…
  - and requires much less space.

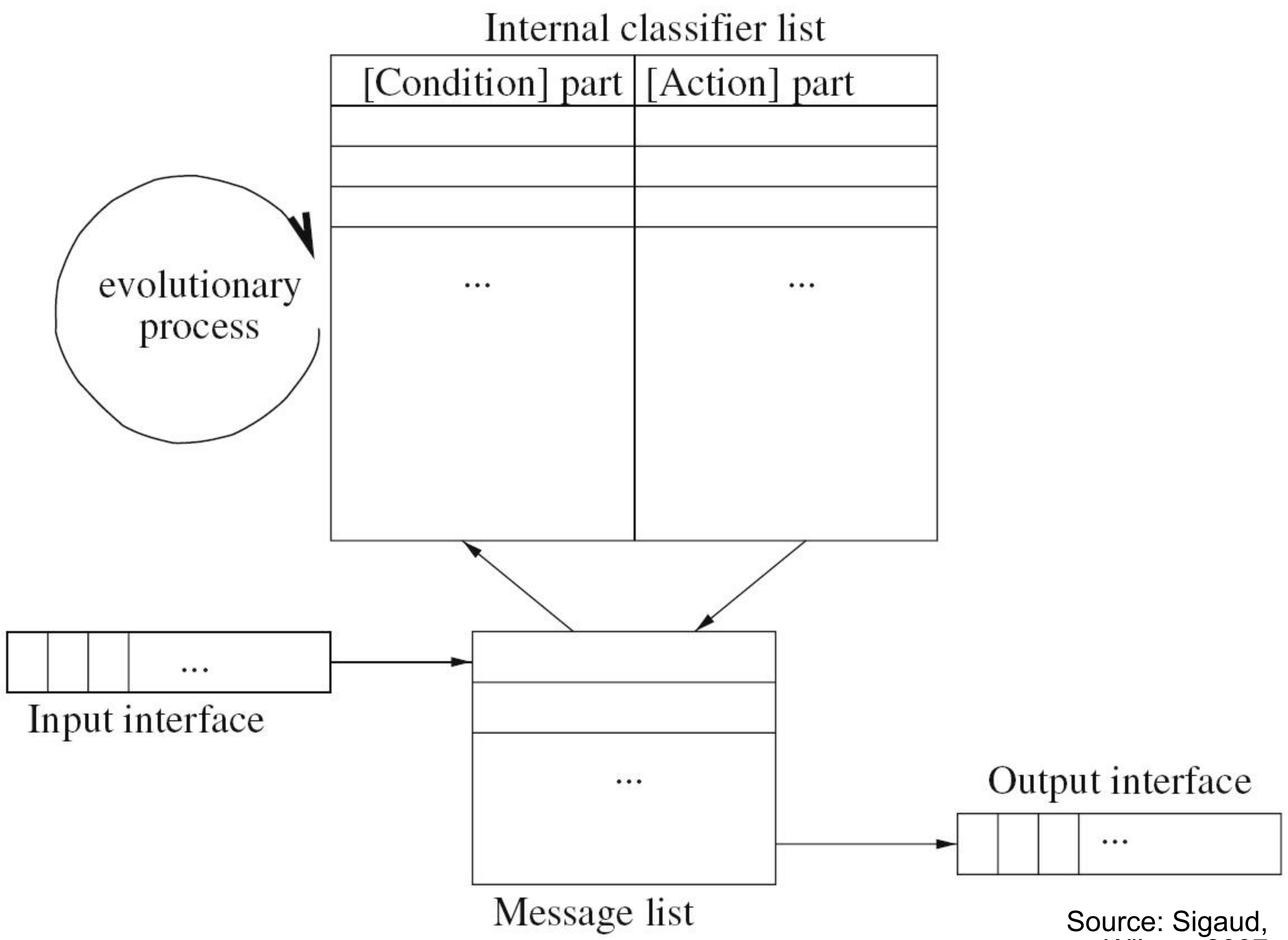
- Abstract view
  - Observation: Obtain (sensory) input about the current state of the environment
  - Action: After reasoning about the current state, the agent decides on an action that impacts the environment.
  - Evaluation: The agent observes the effect of its action and evaluates it (good/bad; reward)



- Initial LCS was introduced by John H. Holland in 1975.
- He was (and still is) interested in complex adaptive systems.
- How can computers be programmed so that problem-solving capabilities are built up by specifying "what is to be done" rather than "how to do it"? (Holland, 1975)
- An important development in LCS was done by Stewart W. Wilson in 1995.
- Based on the initial approach by Holland, Wilson proposed a simplified and more efficient classifier system called XCS.
- XCS is today one of the most studied classifier systems.
- Many extensions have been proposed.

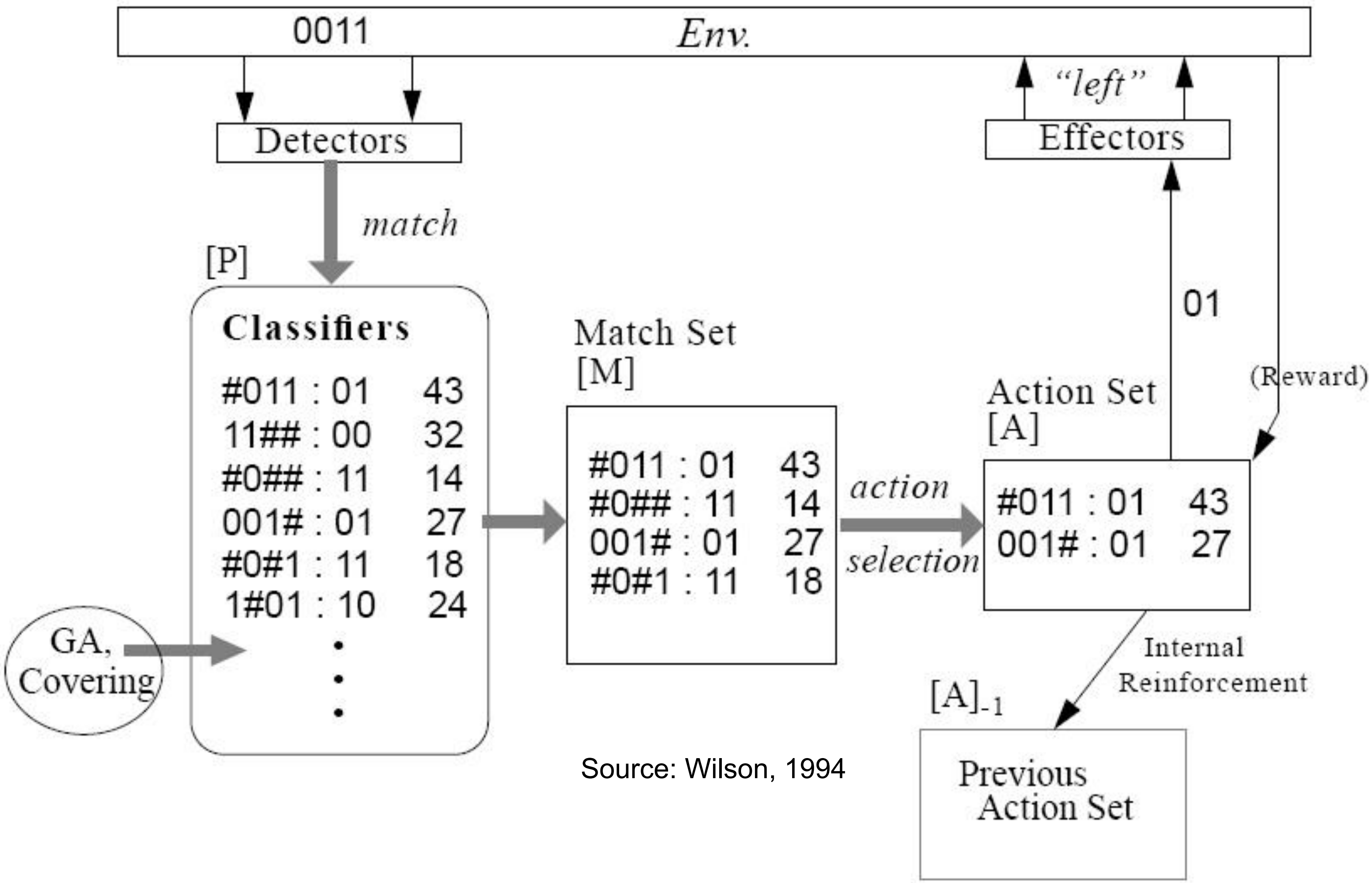
- Initially, Holland designed a system called CS1 (published in 1978).
- System contains
  - Set of classifiers (condition/action) pairs
  - Input interface to receive state from the environment
  - Output interface to apply actions to the environment
  - Internal message list as an internal “workspace” for I/O
  - Evolutionary process (genetic algorithm) to generate new classifiers

Initial approach by Holland (2)



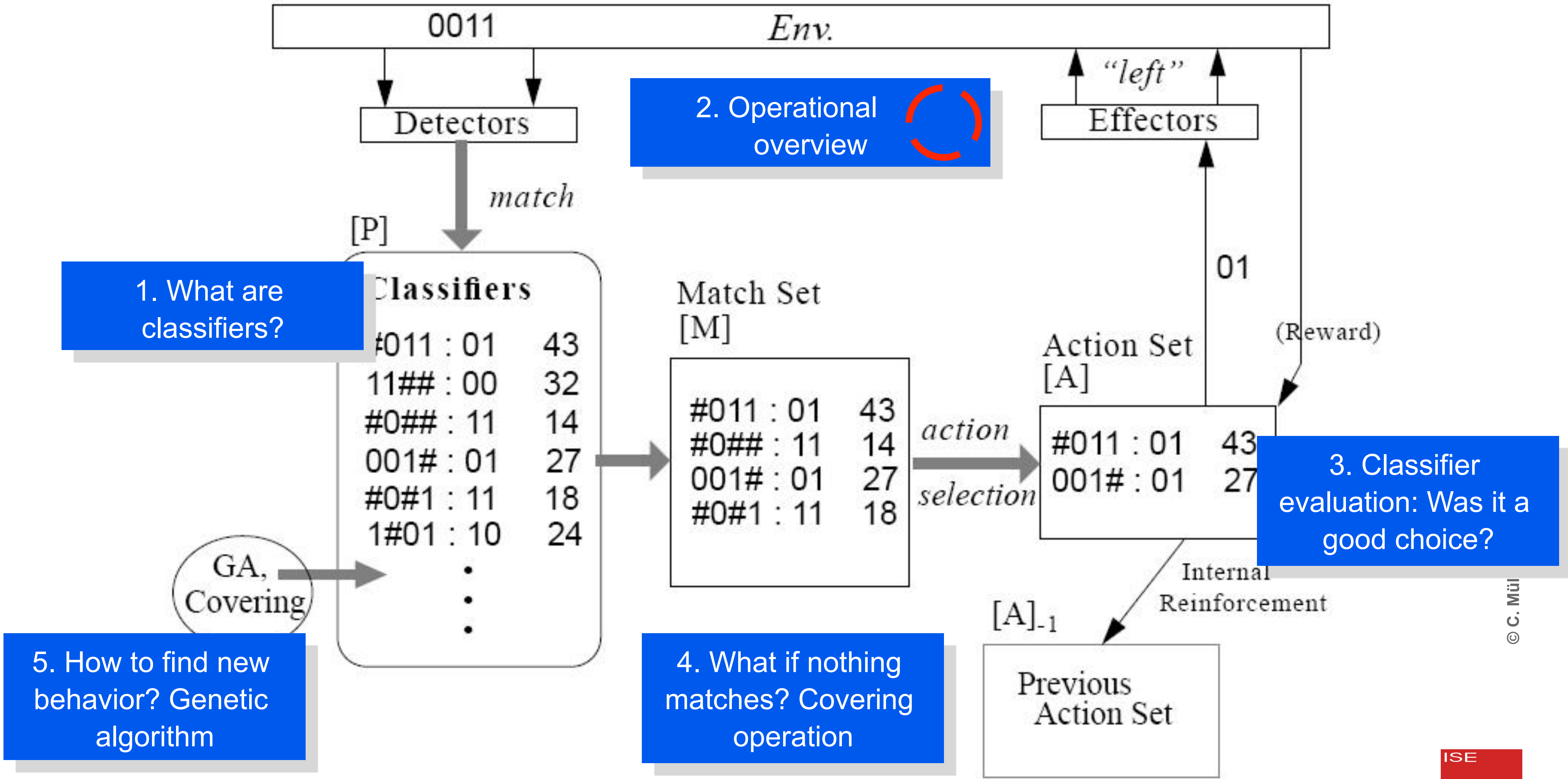
Source: Sigaud, Wilson, 2007

- ZCS was introduced by Wilson in 1994.
- Simplified in comparison to the original LCS proposed by Holland.
- The message list in Holland's approach was rather complicated to use.
- Also a mechanism called "rule bidding" was removed.



Source: Wilson, 1994

In the following we will take a closer look at the different components.



1. What are classifiers?

`if condition then action : fitness`

- Initially a classifier is a triple (Condition, Action, *fitness*).
- Condition: Represents a state of the environment
- Action: Action for effecting the environment
- Fitness (= strength): to “the best of the agent’s knowledge”, applying the classifier’s action under the given condition will return the expected reward.
- To match a classifier: Test condition against environmental state.
- Condition may contain wildcards (typically abbreviated by #).
- Initially, only binary representation was used.

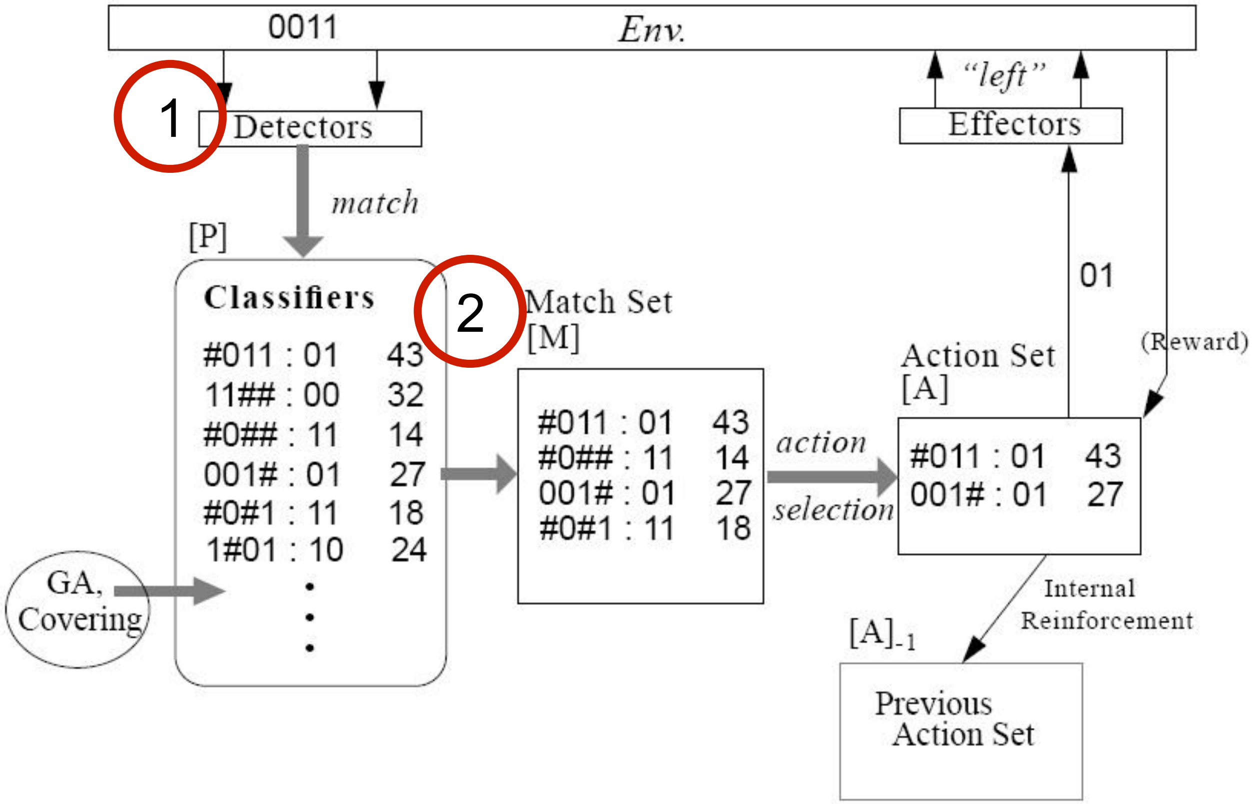
1. What are classifiers?

`if condition then action : fitness`

- Example:
  - (011001, 0, 123): exact match required
  - (01#001, 1, 70): matches 010001 and 011001
- Classifiers have been extended to deal with other representations.
- For example, real values:
  - Use of “interval predicates” instead of binary variables
  - Input attribute x_i matches, if $i_i \leq x_i \leq u_i$
  - Concatenation of (i_i, u_i) pairs for the i-th entry of the condition
- Different representation impacts operations of the genetic algorithm used (in a minute).

- First glance: What is happening at runtime?

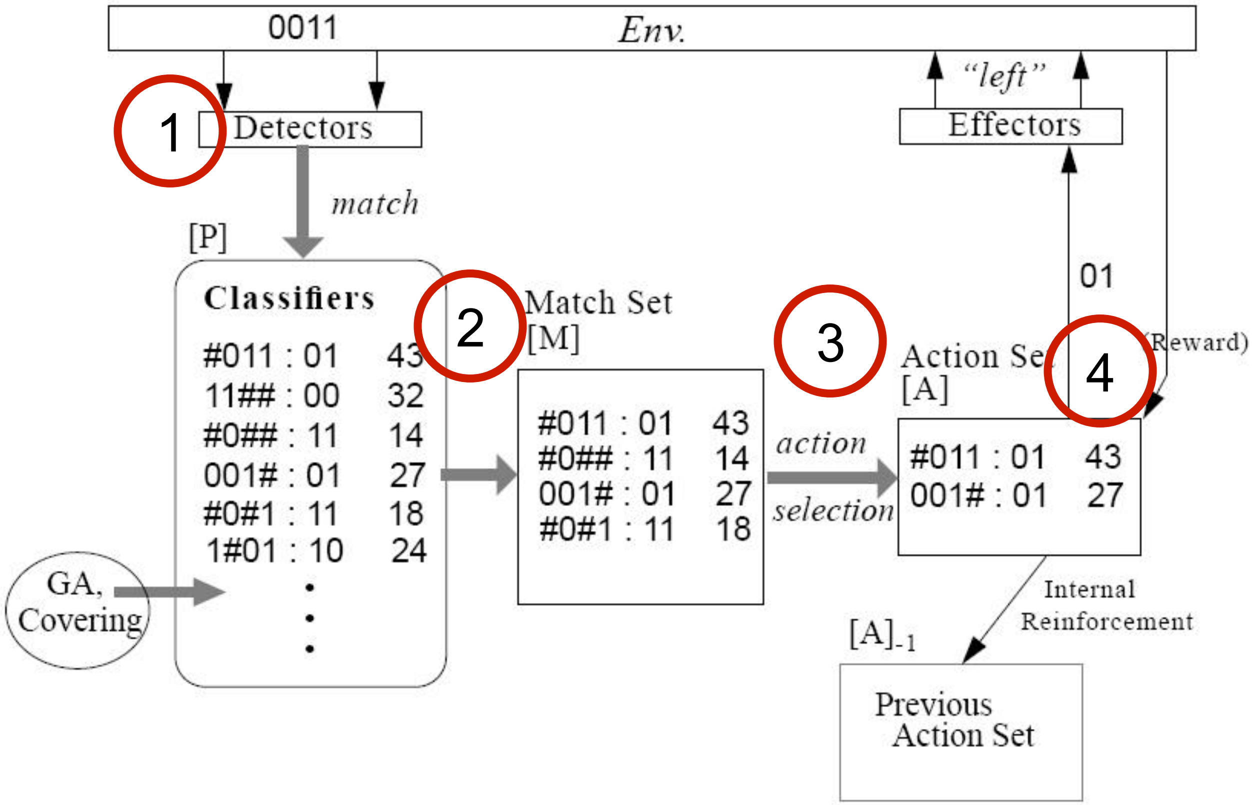
1. Agent periodically receives information about environment through detectors.
2. State is used to calculate the match set: Select all classifiers that match the current situation.



2. Operational overview

3. Action selection through roulette-wheel selection according to fitness (high fitness → high probability of being selected).

4. All classifiers that propose the same action as the selected one are put into the action set.



3. Classifier evaluation: Was it a good choice?

- As the system is supposed to learn, the usefulness of the classifiers has to be "monitored": Idea is to change the fitness according to performance.
- Fitness modification in current and previous action set [A] and $[A]_{-1}$ as well as [M].
- Algorithm parameter: $\beta \in (0, 1]$
- Fixed fraction β of the fitness of all classifiers in [A] is put into a common "bucket".
- cont. …

ZCS: Fitness assignment (2)

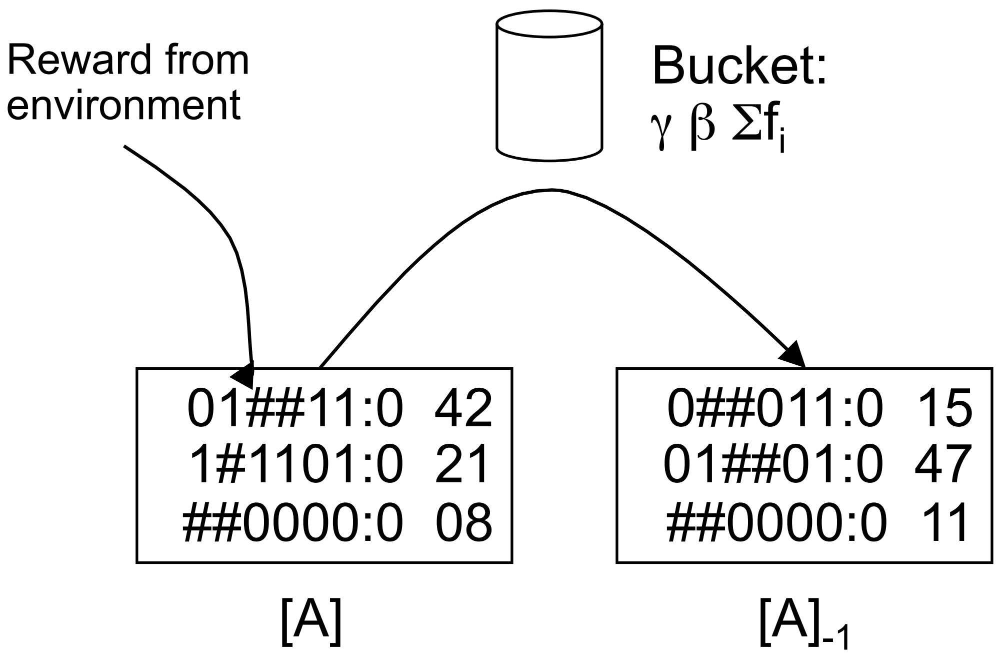
3. Classifier evaluation: Was it a good choice?

- Content of bucket is discounted by $\gamma \in (0, 1]$.
- Bucket content is distributed evenly among classifiers in $[A]_{-1}$.
- Reward paid by the environment: Fraction β of reward distributed evenly among members of [A].
- Remaining members in [M] (and not in [A]) are “taxed” with τ.

ISE SRA

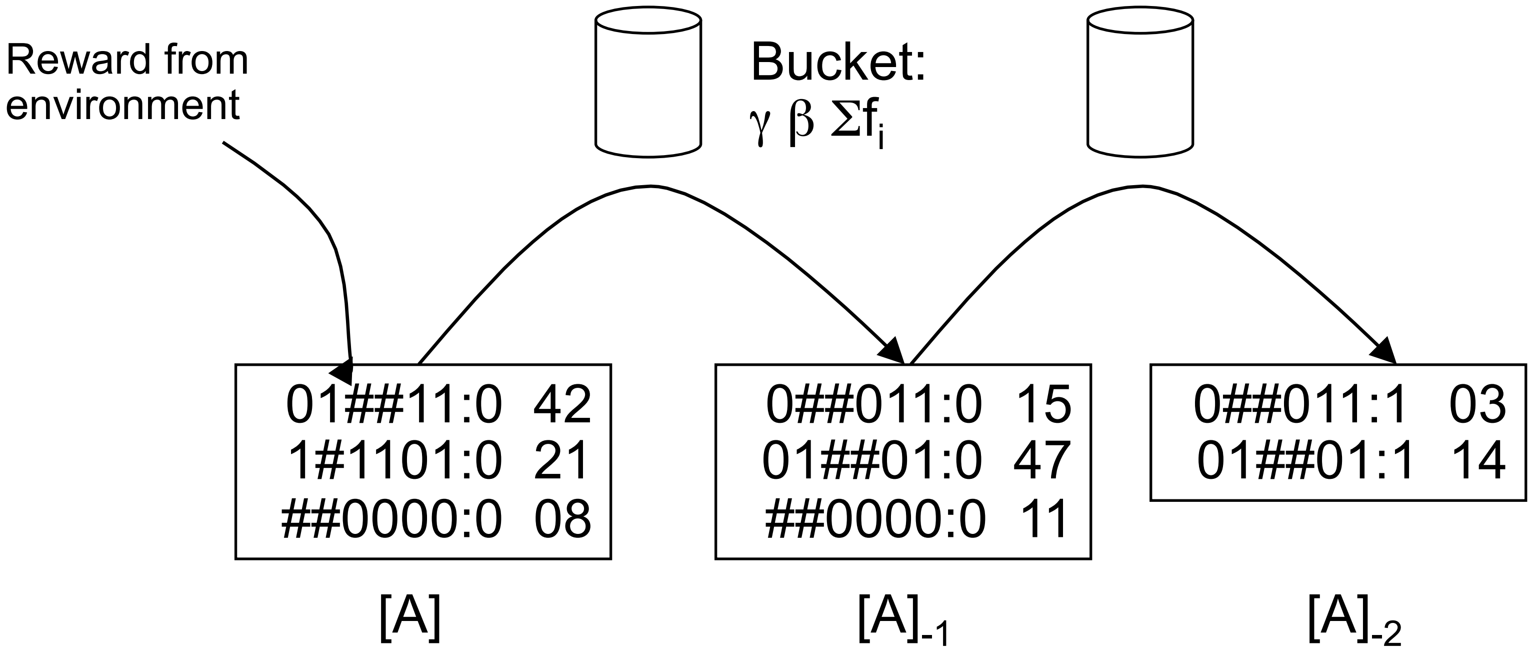
3. Classifier evaluation: Was it a good choice?

- Reward received for current action is paid to action set.
- But: Some fitness is passed on to previous action set.
- Bucket brigade metaphor



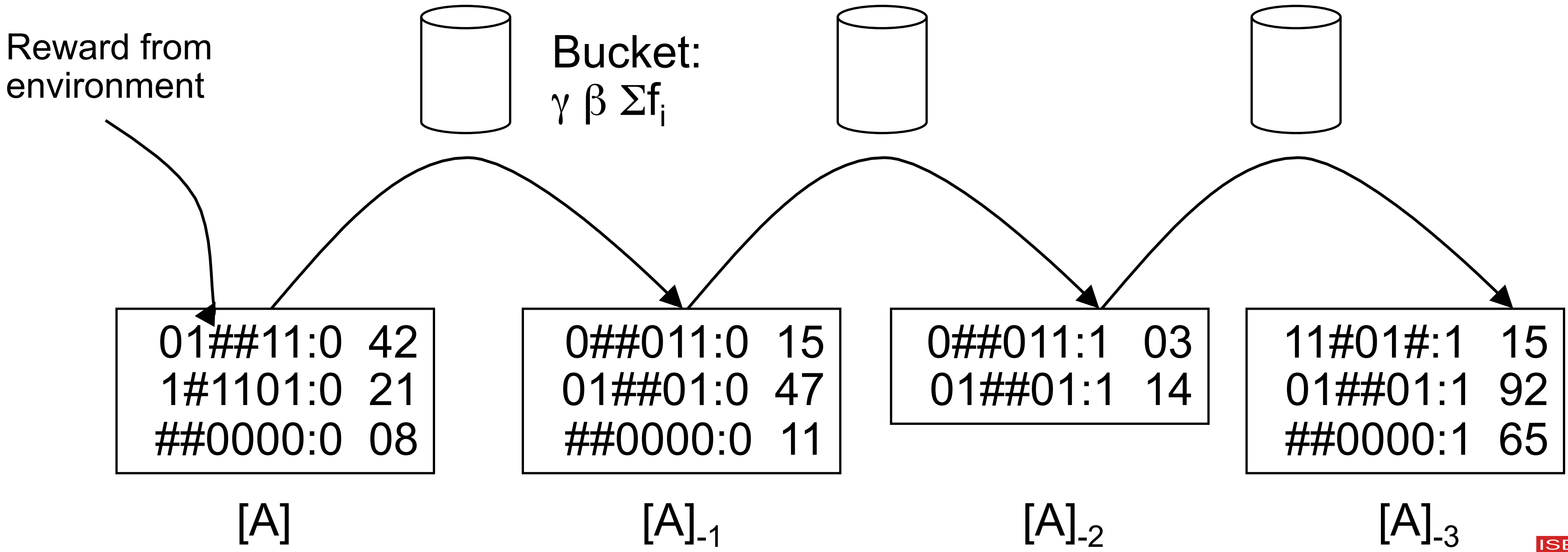
3. Classifier evaluation: Was it a good choice?

- Reward received for current action is paid to action set.
- But: Some fitness is passed on to previous action set.
- Bucket brigade metaphor



3. Classifier evaluation: Was it a good choice?

- Reward received for current action is paid to action set.
- But: Some fitness is passed on to previous action set.
- Bucket brigade metaphor



4. What if nothing matches?
Covering operation

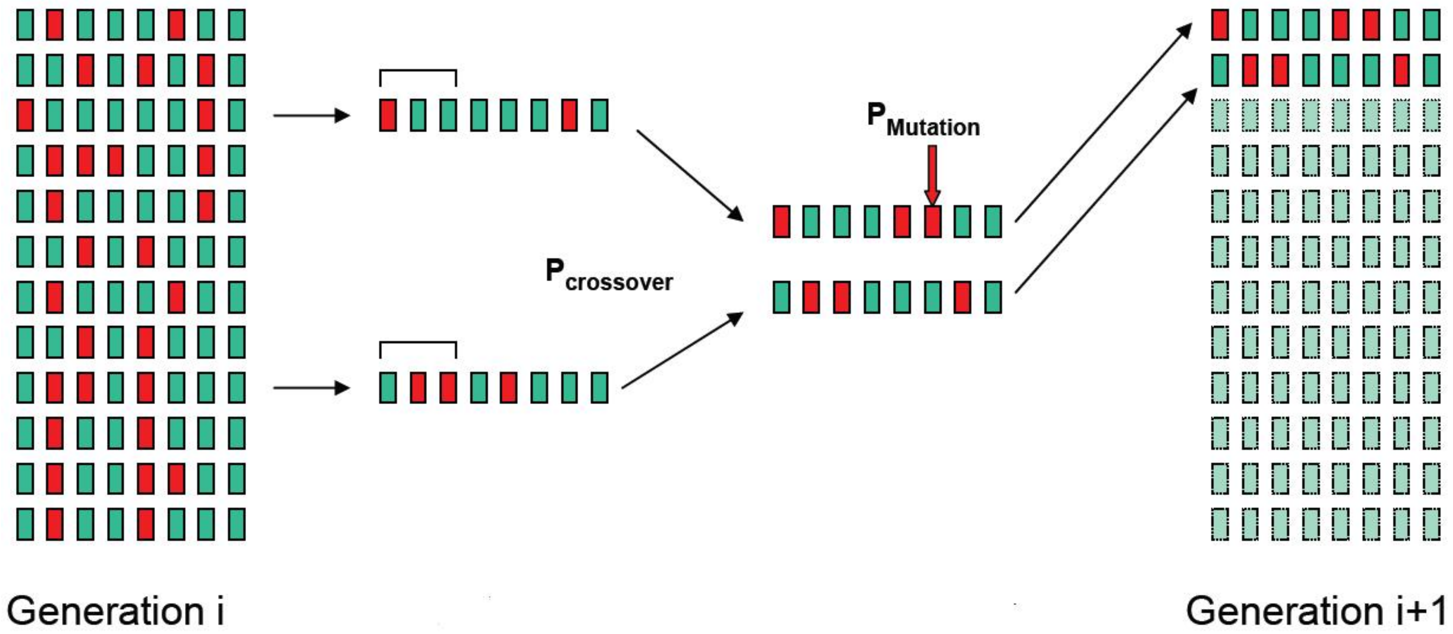
- When input from environment is received:
  - There may be no matching classifier, or
  - the match set is “poor”, i.e., combined fitness is less than Φ times the population’s average fitness.
- Covering mechanism
  - Used to create a new rule matching the current situation
  - Action of new rule is chosen randomly.
  - New rule is generalized: With probability ⅓ each bit of the condition is replaced by # (wildcard).
  - Fitness is initialized with population average.

5. How to find new behavior? Genetic algorithm

- Inspired by Darwinian evolution: Survival of the fittest
- On each invocation two classifiers are selected as parents: roulette-wheel based on fitness
- With fixed probabilities, these "parents" are subject to "genetic" operations
  - Crossover (recombination)
  - Mutation
- Resulting two classifiers get half of their parents' fitness (which is deducted from them).
- Rate at which GA is invoked is application dependent.
  - Too frequent: noisy fitness
  - Too seldom: slow development
  - Classifiers should have been evaluated a "couple of times" before becoming a parent.

Genetic operations: Example illustrated

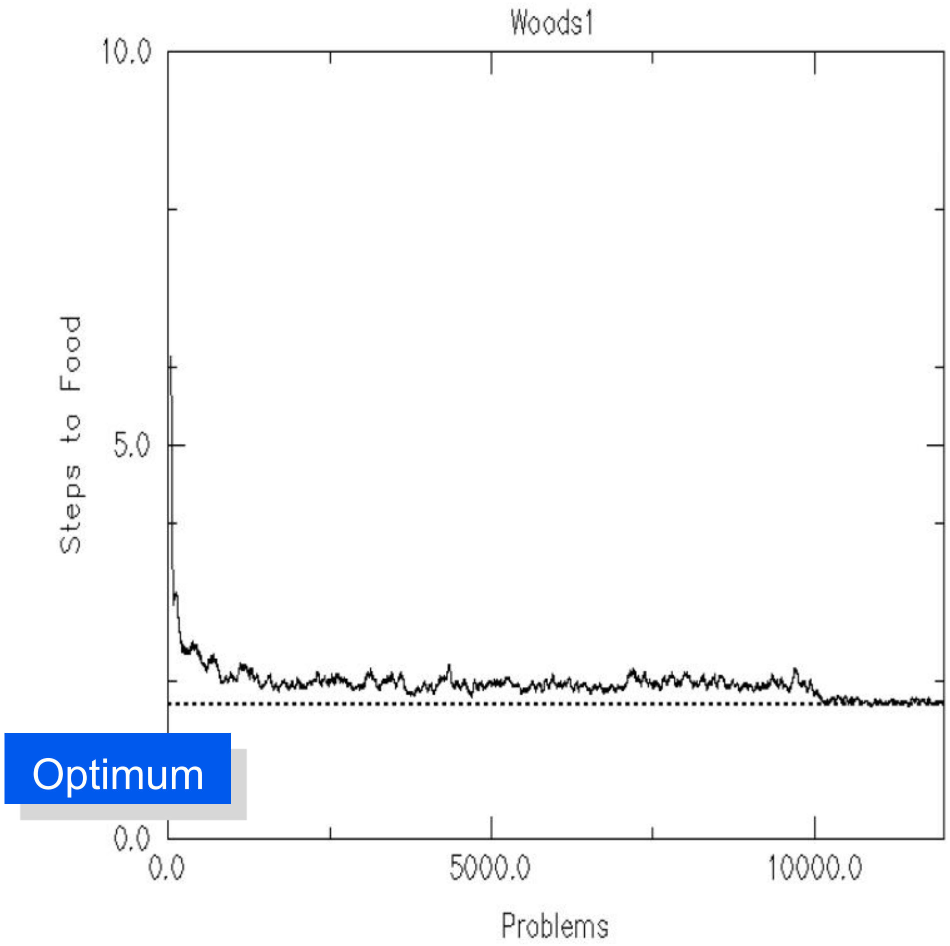
5. How to find new behavior? Genetic algorithm

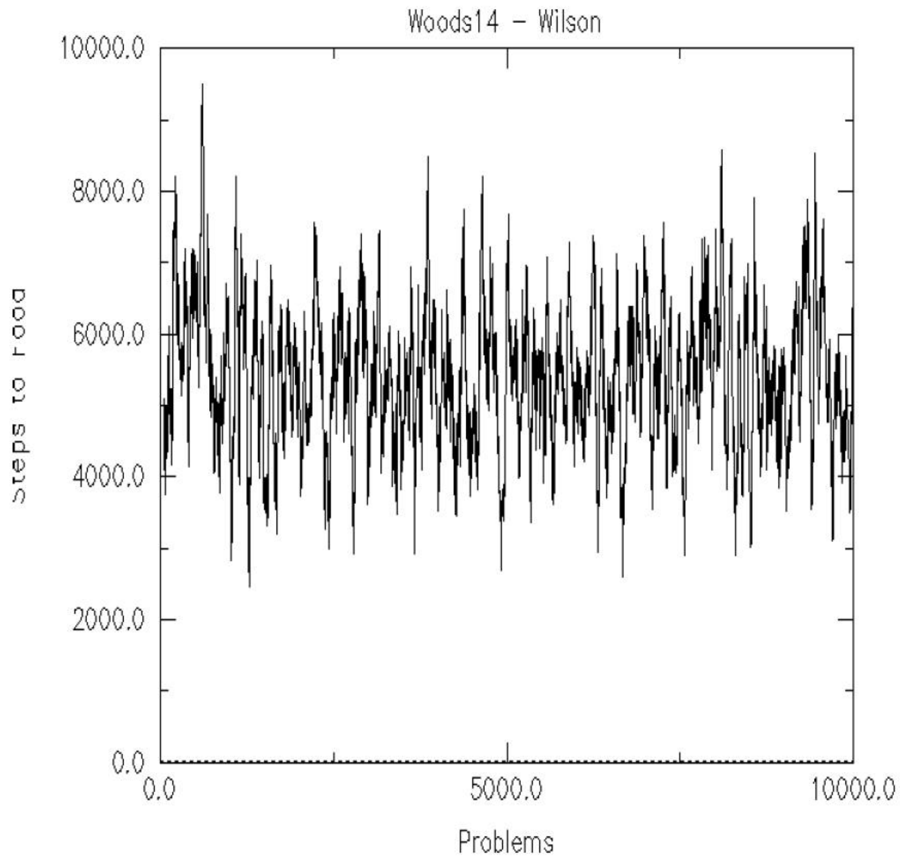


- One-point crossover in the example: 3/5 bits
- Multi-point crossover: More than one “cut point”
- Mutation: Randomly flips a bit (typically very small probability).

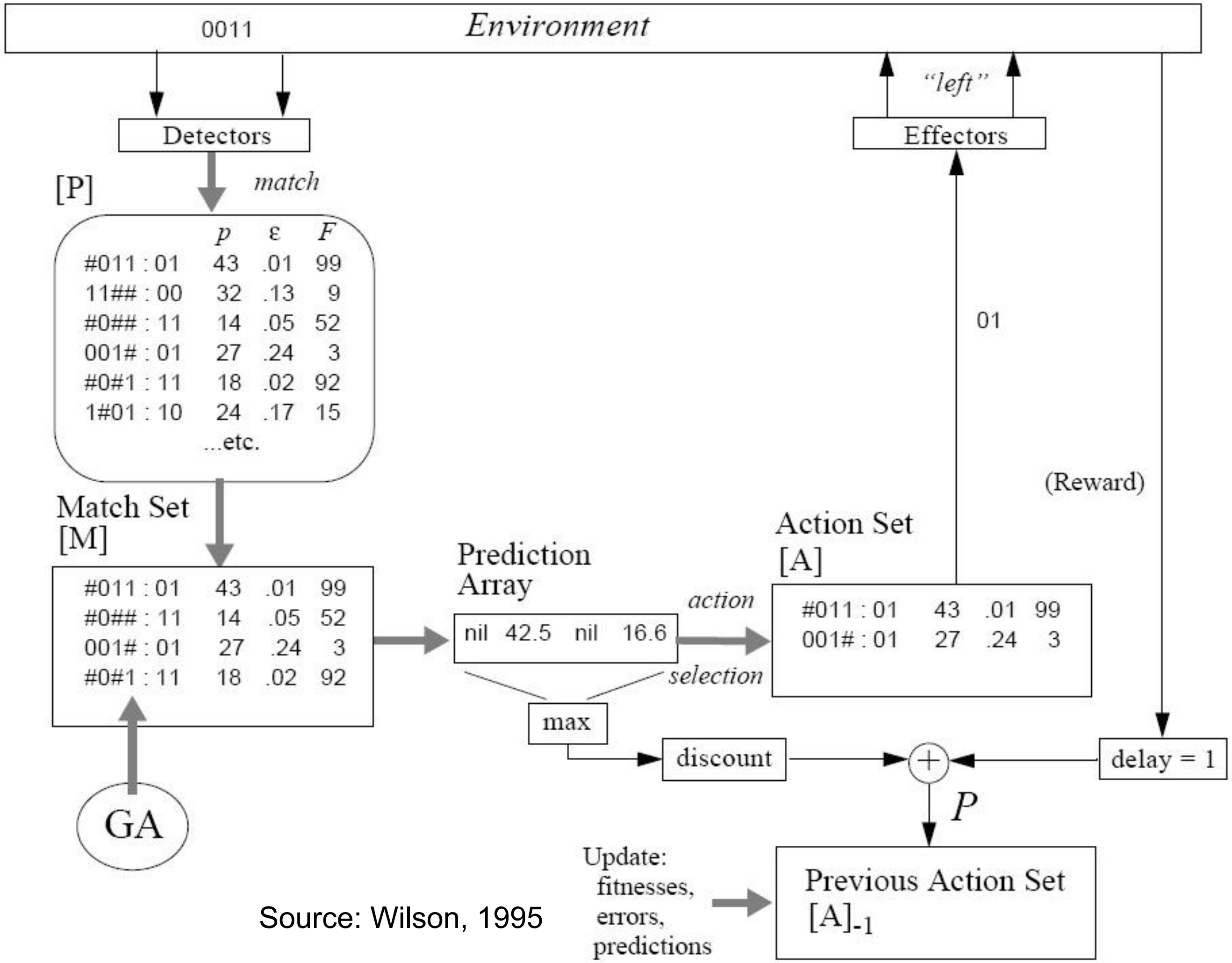
- Typical parameter values according to Wilson:
  - Rule-base (population, [P]) contains 400 classifiers
  - Initial fitness of classifiers set to 20
  - Discount factor $\gamma = 0.71$
  - Tax $\tau = 0.1$
  - Covering trigger $\Phi = 0.5$
  - GA rate per time step $p = 0.25$
  - Crossover rate = 0.5
  - Per bit mutation rate $\mu = 0.002$

Performance examples





- In ZCS (and other LCSs): Fitness of a classifier is measured by its expected reward.
- Wilson suggested that in some situations a classifier with low pay-off may be best suited.
- XCS introduces separate attributes for expected reward and fitness.
- In some situations the best reward may be lower than in other situations.
- Fitness of a classifier is consequently measured by the accuracy of the reward *prediction*.
- Coined the term accuracy-based classifiers in contrast to strength-based (fitness-based).



Source: Wilson, 1995

- Classifiers are extended to a 5-tuple
  (Condition, Action, Prediction (*p*), Prediction error (ε), Fitness (*F*))
- Condition, Action: same as in ZCS
- Prediction: A measure for the pay-off received when this classifier's action controlled the system
- Prediction error: A measure for the classifiers miss-prediction
- Fitness: A function of the inverse prediction error
- Other parameters can be used depending on the implementation, e.g., the number of times a classifier has been used (experience).
- Prediction and prediction error are used in action selection.
- Fitness used by the genetic algorithm (and also in action selection)

- Match set calculated as before
- Many policies may be used for action selection, e.g.
  - Deterministic selection: Highest prediction wins.
  - Probabilistically based on fitness-weighted average of predictions suggesting an action (contained in the architecture's prediction array)
- Selected action is sent to the effectors.
- Reward may (or may not) be received in return.
- Classifiers are updated (differently for single- and multi-step problems).

ISE SRA

- Reinforcement consists of updating prediction, prediction error, and fitness.
- Let's consider the (simpler) update process in single-step problems:
- Update prediction for each classifier C_j in [A]
  - $p_j \leftarrow p_j + \beta\,(P - p_j)$
  - P is the current reward.
  - β is called the learning rate.
- Update prediction error accordingly
  - $\varepsilon_j \leftarrow \varepsilon_j + \beta\,(|P - p_j| - \varepsilon_j)$
- Update fitness accordingly:
  - $F_j \leftarrow F_j + \beta\,(\kappa_j' - F_j)$

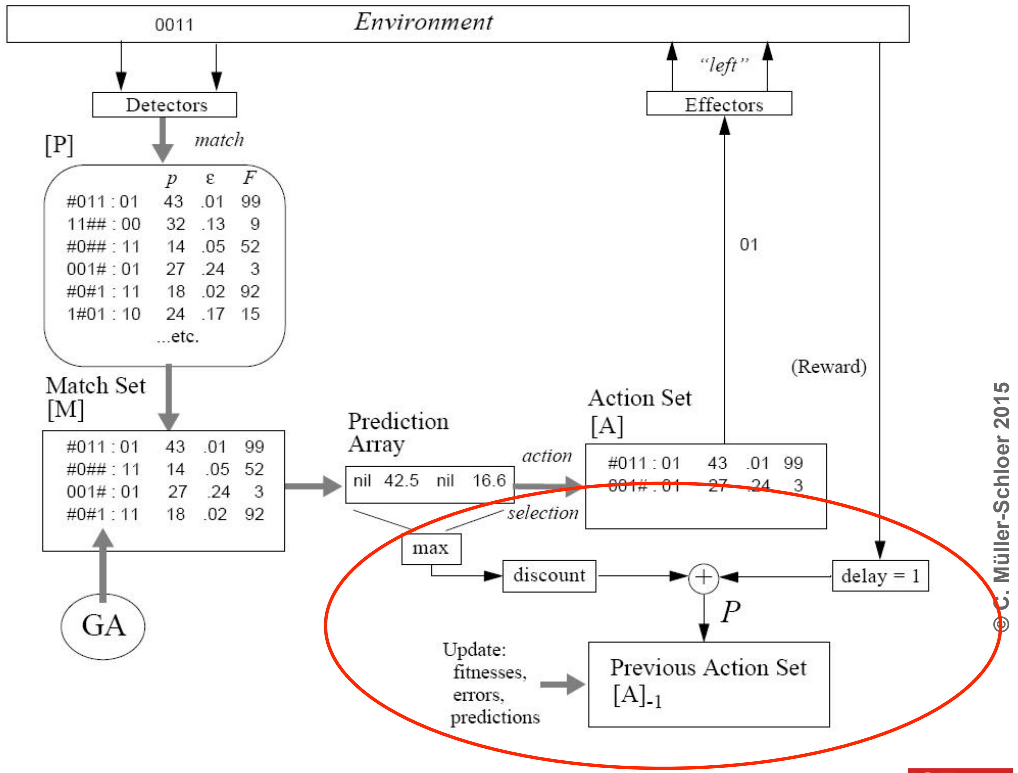
low prediction error → high fitness!

  - κ_j' is the relative accuracy across [A], $\kappa_j \cong 1/\varepsilon_j$

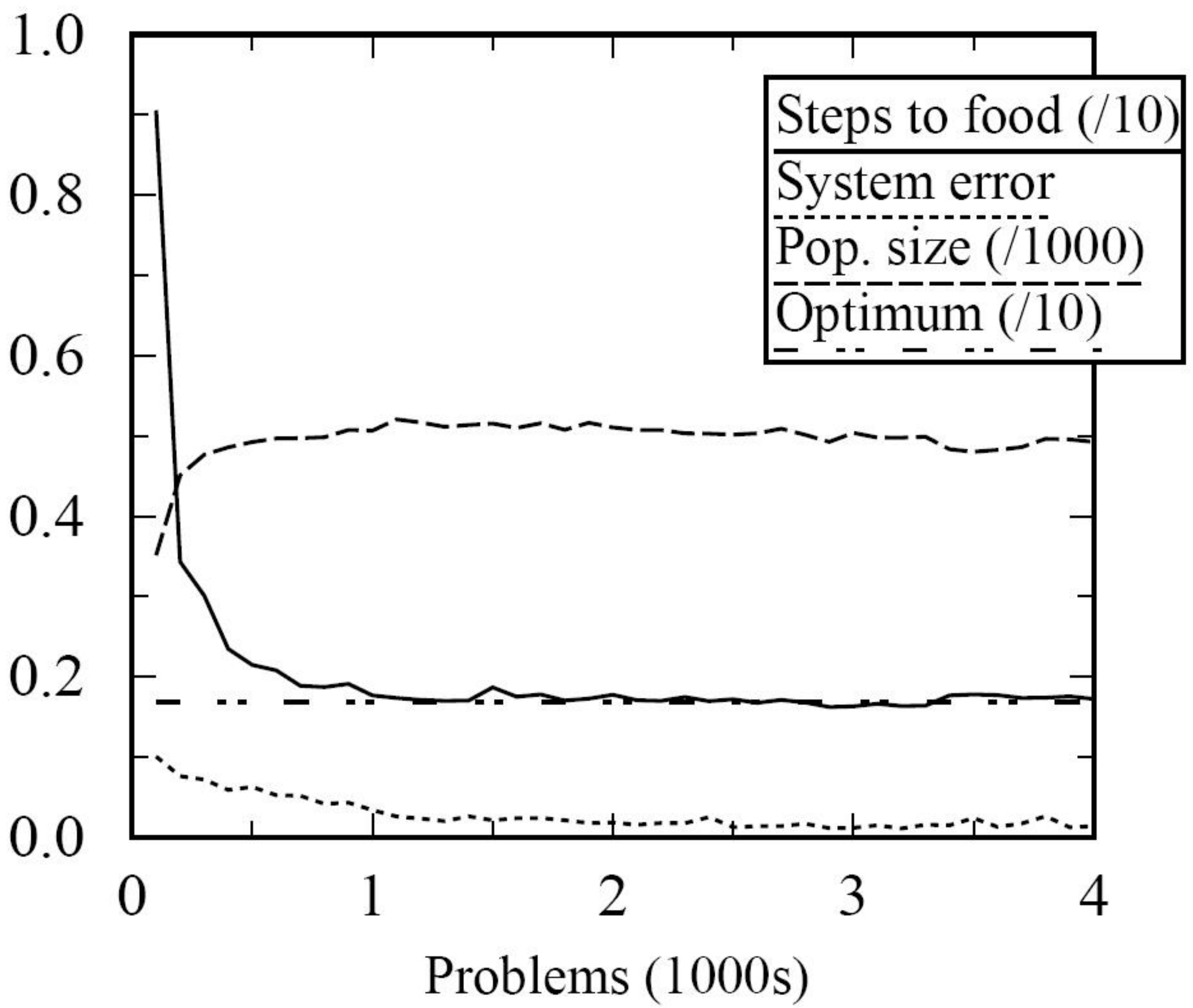
- For multi-step problems updating is done based on $[A]_{-1}$.
- Delayed update allows to retrieve “information from the future”.
- Inspired by Q-Learning (reinforcement learning technique)
- Here reward P is calculated differently:
  $P = r + \gamma * \max P(a)$

Reward from t-1

Prediction array of t

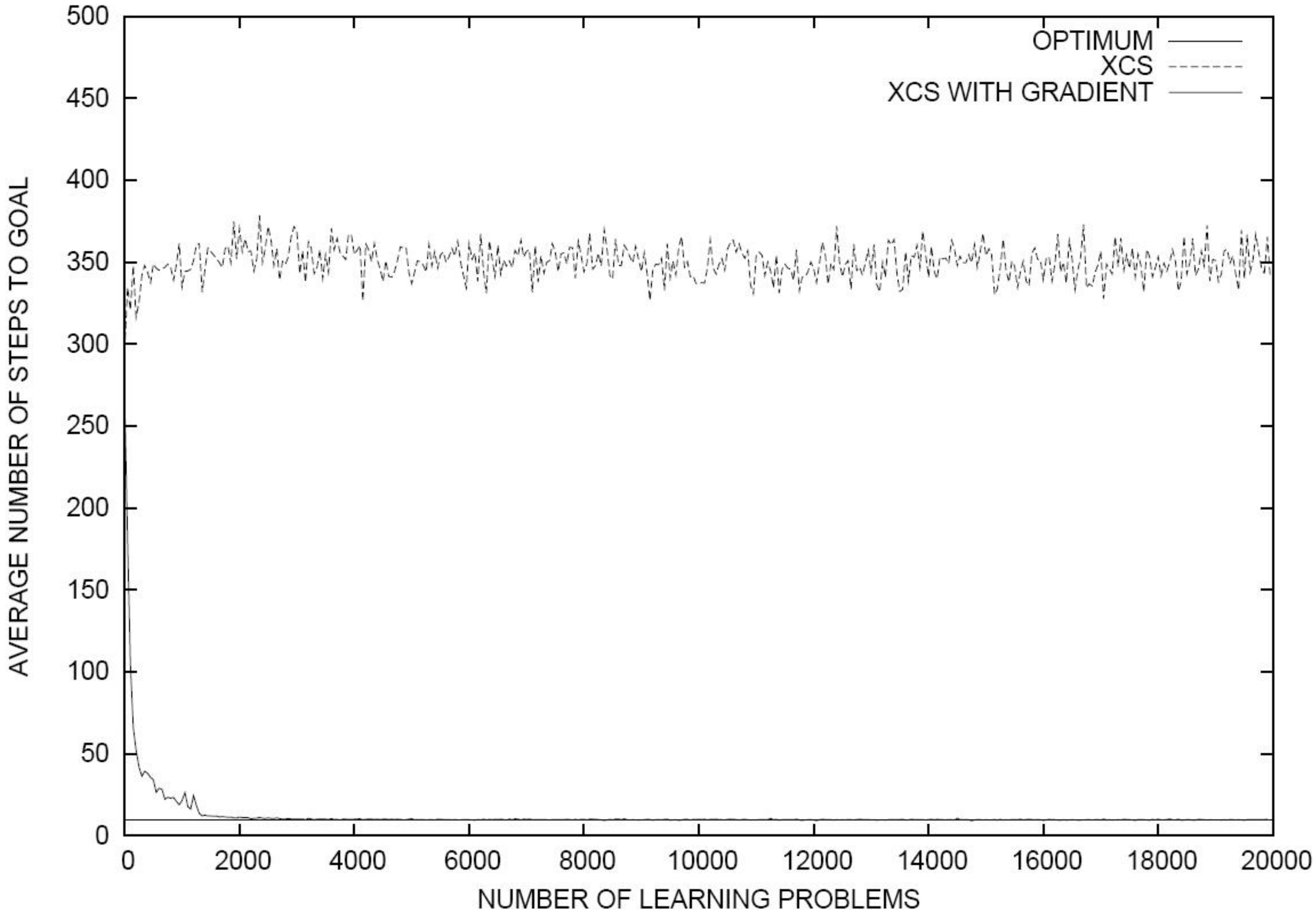


- Experiments done on the Woods2 scenario (complicated Woods1)



Source:
Wilson 1995

- Further improvements of XCS suggested by Butz, Goldberg and Lanzi, (2003) show great improvements for multi-step problems



- Machine learning techniques are applied in OC application.
- One example is the Organic Traffic Control (OTC) project.
- Overall goal: Online adaptation of traffic light controllers to changing traffic situations
- Requirements
  - Adapt autonomously to the environment
    - Long term changes
    - Short term fluctuations, incidents
    - Re-use knowledge
  - Safety: Limit effects of possible errors of learning component!
  - Comprehensible behavior
  - Limit necessary manual intervention and effort for setup!

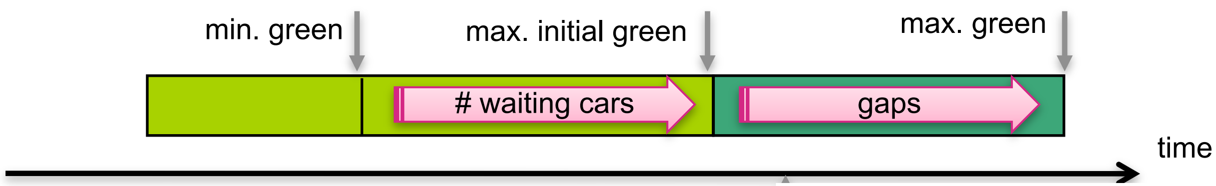
- Input: Traffic situation, vehicles per hour (flow) per relation in the junction (turning)
- Output: Parameter set modifying the program for traffic light controller (fixed time or traffic responsive control)
- Objective function: Level of Service (LoS, average delay time per vehicle); used in Germany (HBS) and the US (HCM)
  - Goal: Minimize LoS value
  - Best: LoS = 0 (no delay for anyone)

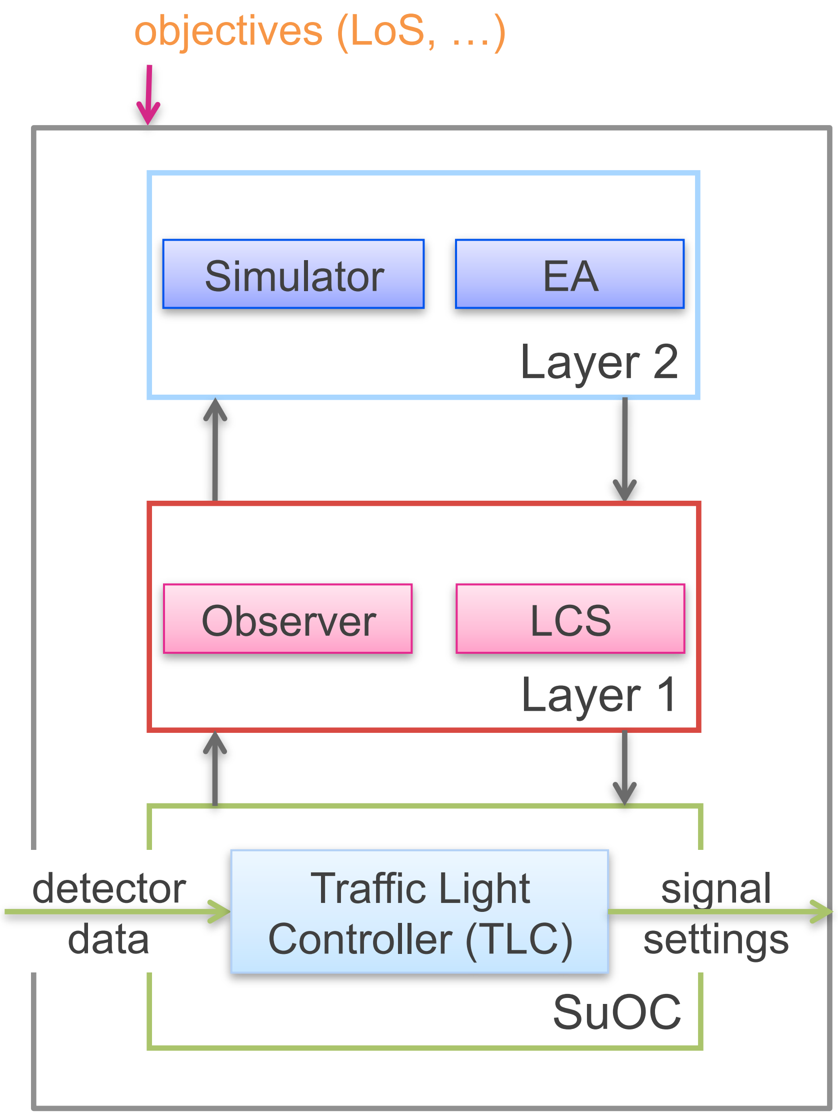
Flows weighted with delay

$$LoS = \frac{\sum flow \cdot t_{delay}}{\sum flow}$$

Each flow: vehicles/hour

- Example: NEMA Controller (traffic responsive controller)
- Cars in each lane are counted by detectors.
- Detector data represents (typically incomplete) traffic situation.
- Once set to green, it stays there until at least "min. green".
- Extension up to "max. initial green" depending on number of waiting cars
- After that: When gap between incoming cars becomes large: leave green
- Complete parameter set: 9 parameters
- Goal of OTC: LCS selects TLC controller settings based on current traffic situation.





User interface
- User defines system objectives

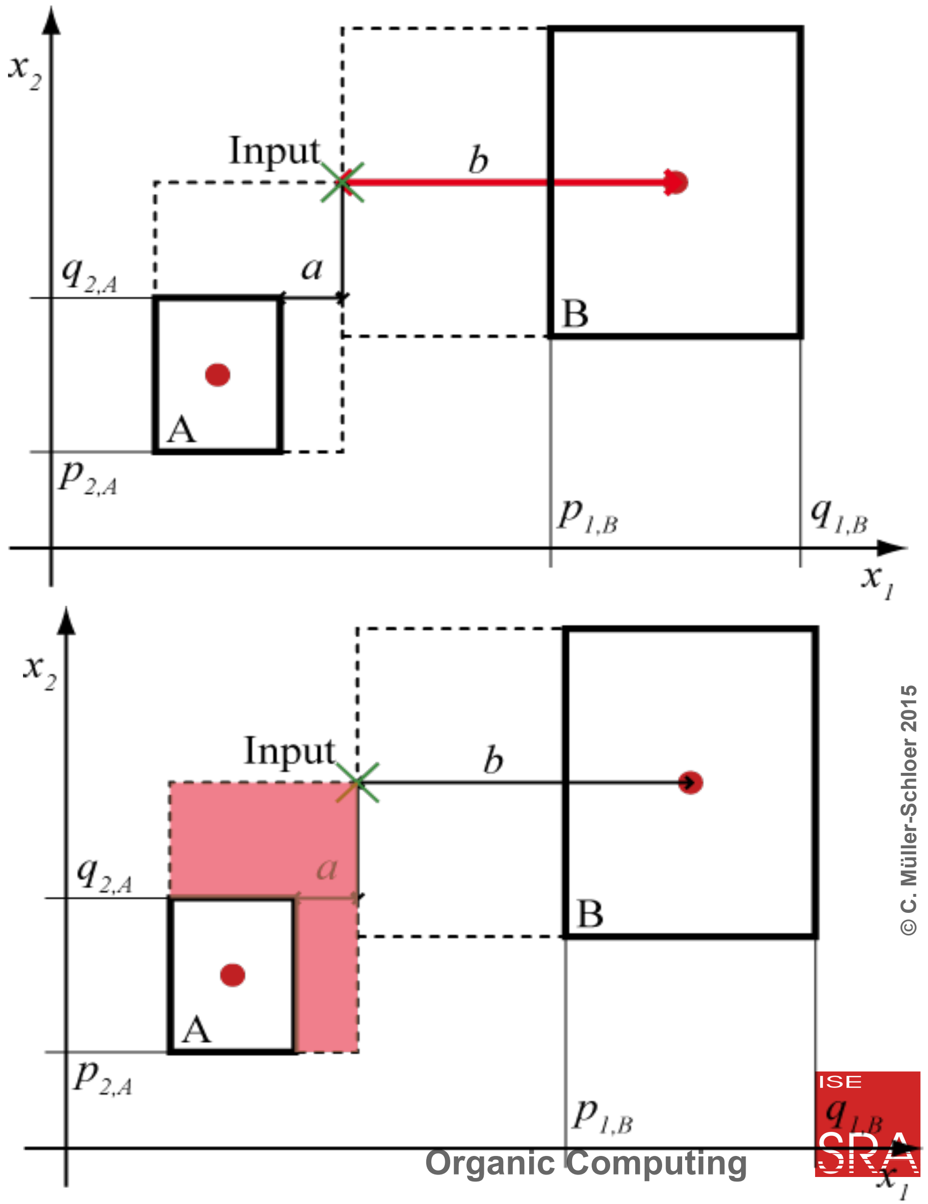
Layer 2
- Extend behavioral repertoire of Layer 1
- Offline learning (TLC parameters)

Layer 1
- Adapt SuOC parameters
- Online learning (rule quality)

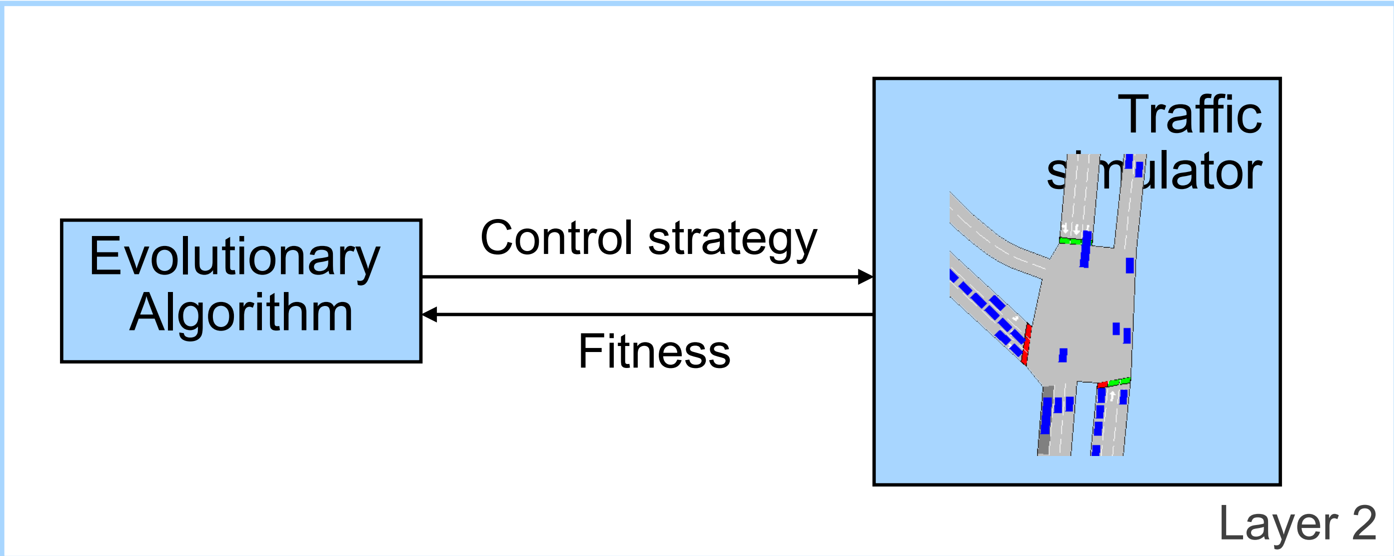
System under Observation and Control
- Control traffic signals

- Exploring the classifier space online using GA can be dangerous!
  - System could try the rule: Set all traffic lights to green!
- Consequence: Generation of new rules done in a separate (isolated) component: “Sandbox”
  - Learning is done offline in a simulator (layer 2).
  - LCS updates classifiers, performs covering, applies changes to TLC (layer 1).
- In classifiers: Representation of input as real-valued intervals
- Building of match set: Trade-off between “use only matching solutions” and creative competition needed for learning

- In the original XCS: Covering creates new classifier for the current situation randomly.
- OTC: Application specific widening of existing classifiers
- Select "closest" rule, copy, widen condition
- Trade-off between "use only tested solutions" and quick reaction time
- Additionally: Threshold used to trigger layer 2 GA

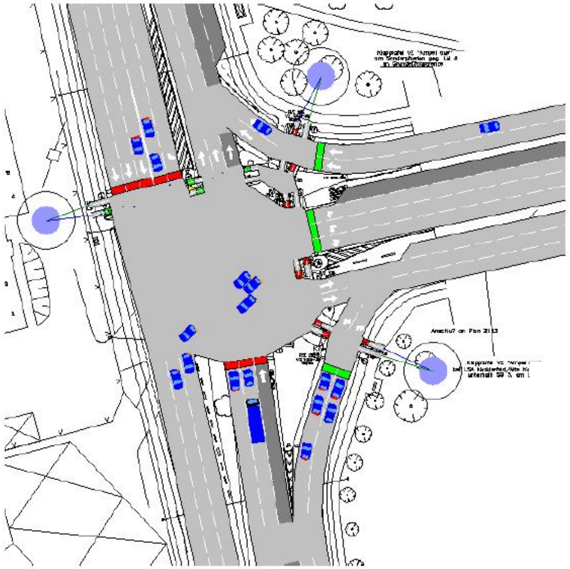


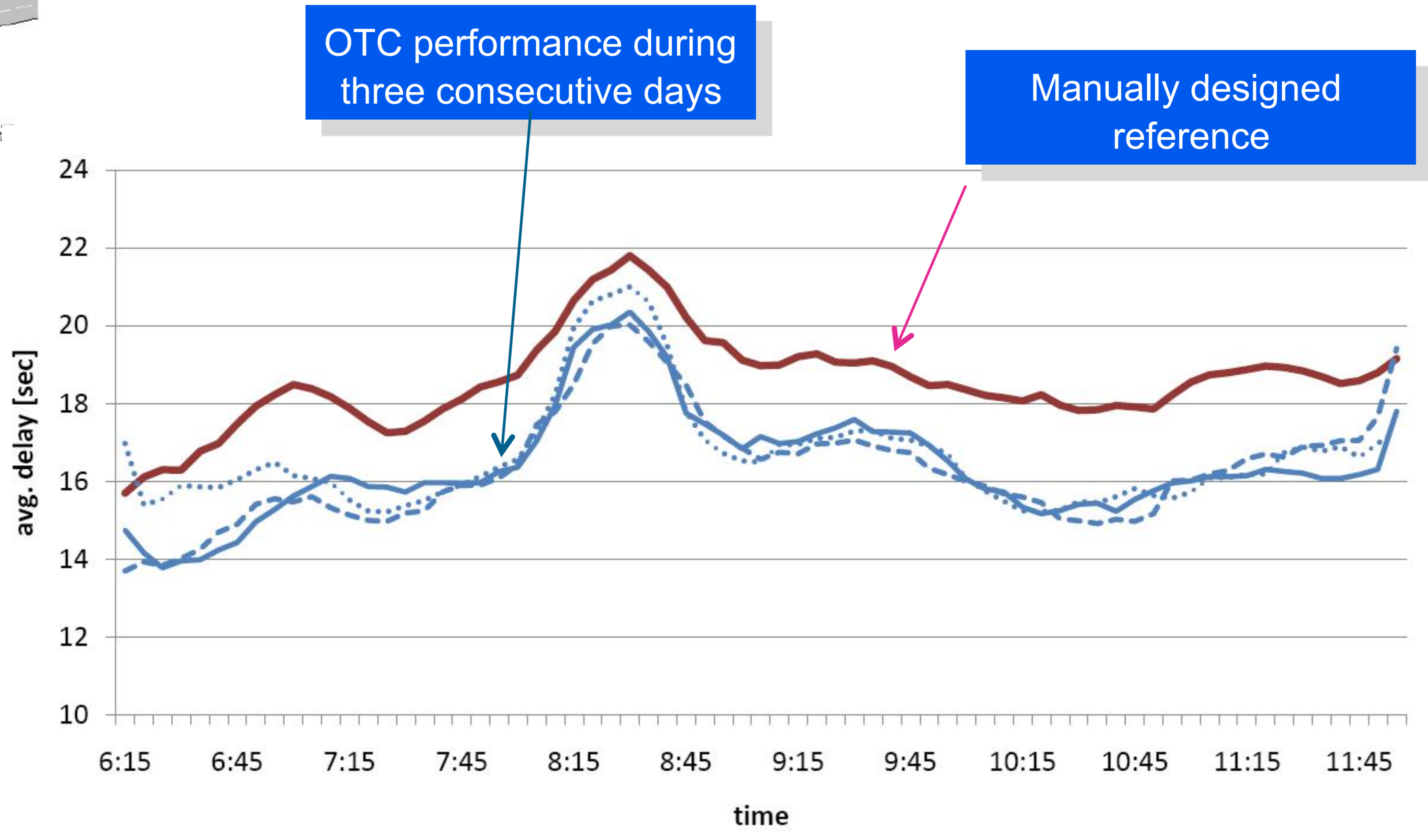
- Generates control strategies for traffic signals (layer 2)
- Quality of strategy is tested using a traffic simulator.
- Fitness metrics: e.g. mean travel time, average number of stops, …



- Strategies are evaluated under different traffic conditions.
- Optimized if-then-rule (condition + strategy) is added to rule set of LCS.

OTC: Performance





Summary

- Machine learning techniques show promising results for self-adaptation in organic computing systems.
- Learning classifier systems are a good example of such techniques.
- The OTC project makes heavy use of LCSs in a practical technical system.
- Modifications have been proposed by the project
  - Increased safety through offline learning (layer 2)
  - Modified operations adapted to the specific problem

- Oliver Sigaud, Stewart W. Wilson: Learning classifier systems: a survey. Soft Computing, pp. 1065-1078, 2007
- John H. Holland: Adaptation in natural and artificial systems. MIT Press, 1992 (contains the text of the 1975 edition with some extensions)
- Stewart W. Wilson: ZCS: A zeroth-level classifier system. Evolutionary Computing Vol. 2, No. 1, 1994
- Larry Bull, Jacob Hurst: ZCS Redux. ???
- Stewart W. Wilson: Classifier Fitness Based on Accuracy. Evolutionary Computing Vol. 3, No. 2, 1995 (original XCS reference)
- Butz, Goldberg, Lanzi: Gradient Decent Methods in Learning Classifier Systems: Improving XCS Performance in Multistep Problems. IlliGAL Report No. 2003028, 2003