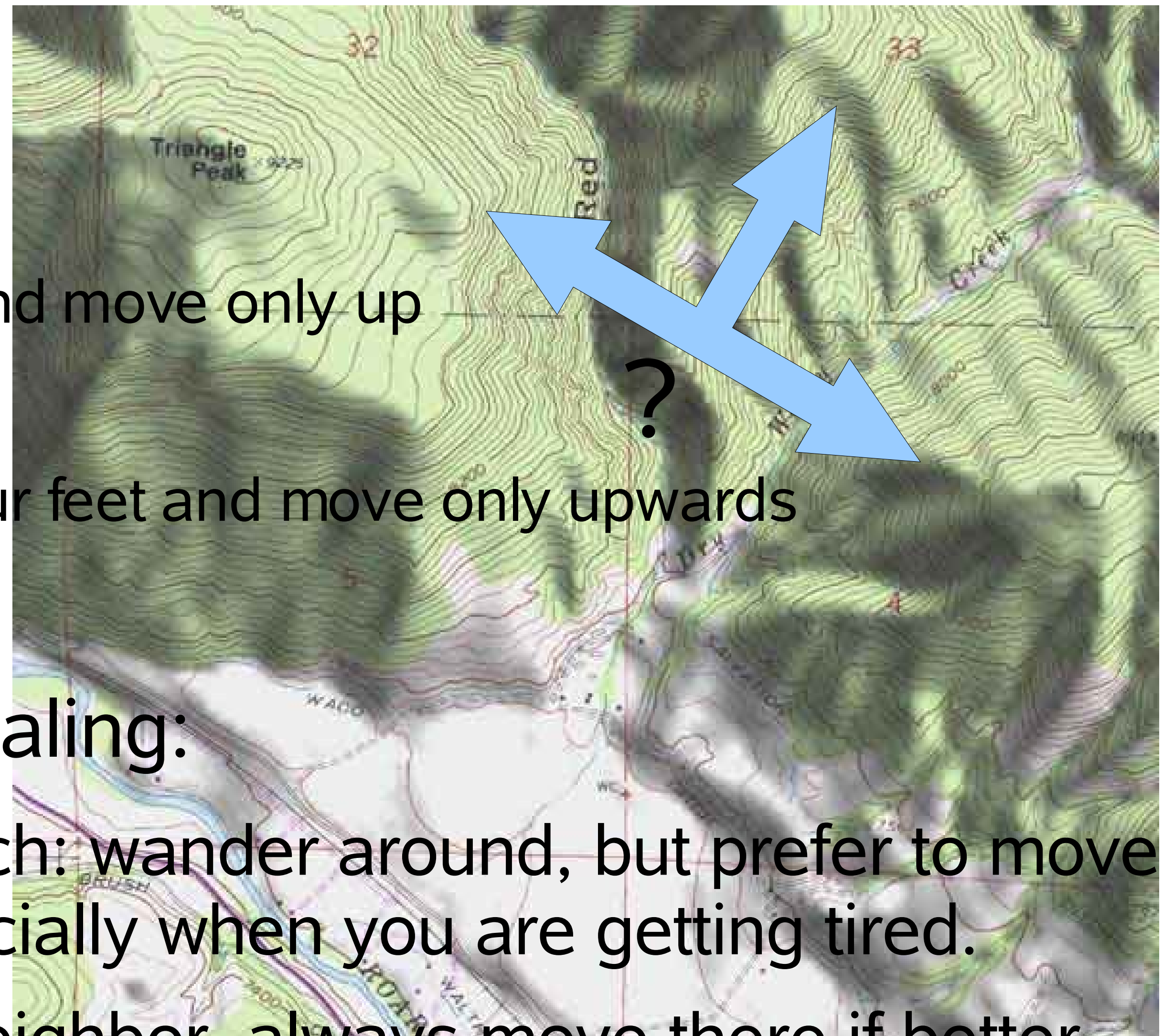
Some ideas on Evolutionary Algorithms

randomized search, or optimization, algorithms,
with evolution-inspired heuristics

Sometimes useful for...

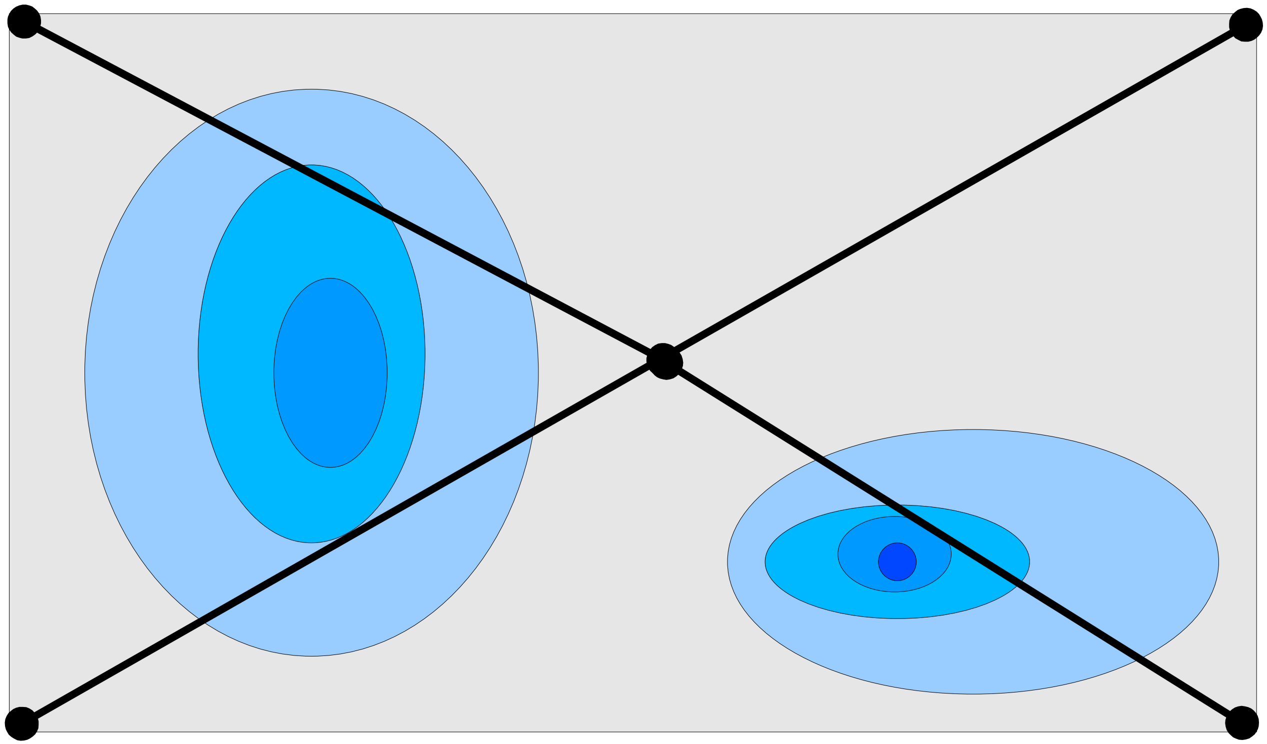
- global optimization: finding the best solution ever
- multimodal optimization, exploring the search space: finding many alternative solutions / localizing many optima
- finding sufficient solutions (not necessarily optimal) of hard problems
- optimizing objects with continuous, discrete and structural components

What is global optimization?

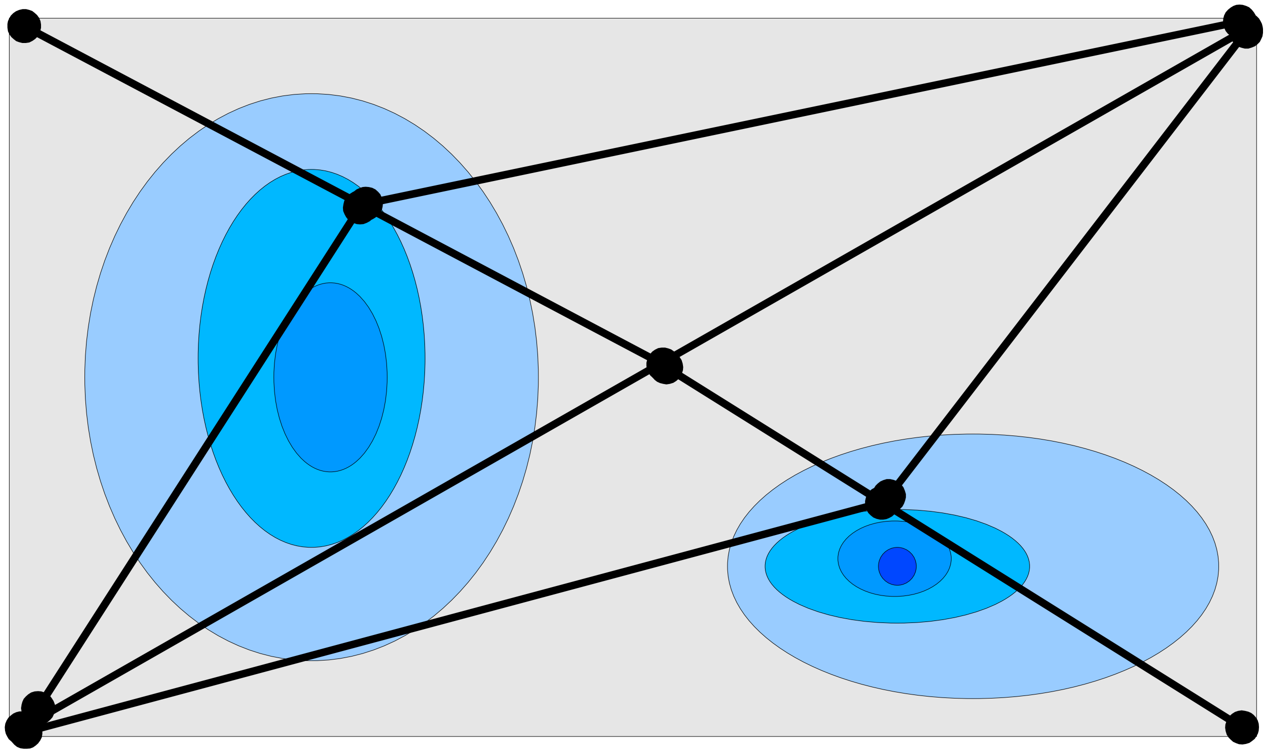


- Greedy search:
  - hill climbing
    - look around and move only up
  - gradient based
    - look under your feet and move only upwards
- doesn't work.
- Simulated annealing:
  - Hiker's approach: wander around, but prefer to move upwards, especially when you are getting tired.
  - Pick random neighbor, always move there if better, and sometimes also if a bit worse.

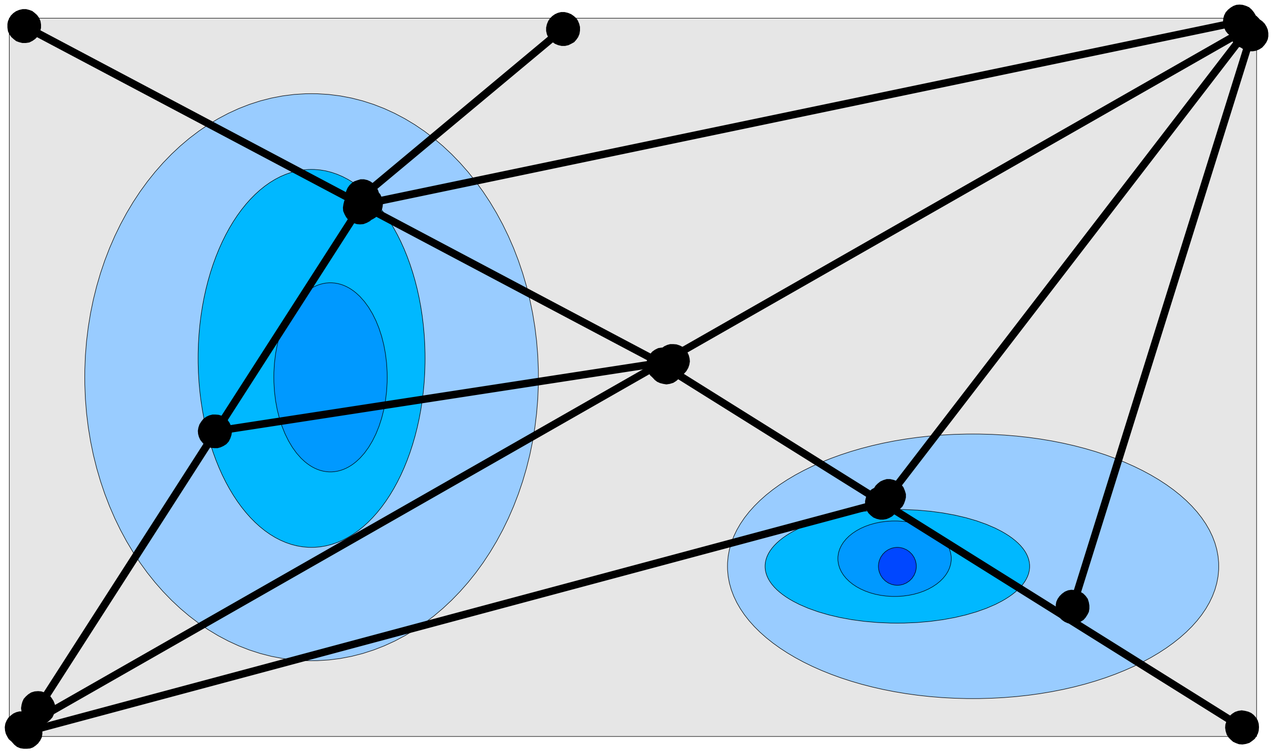
How to „map” the search space?



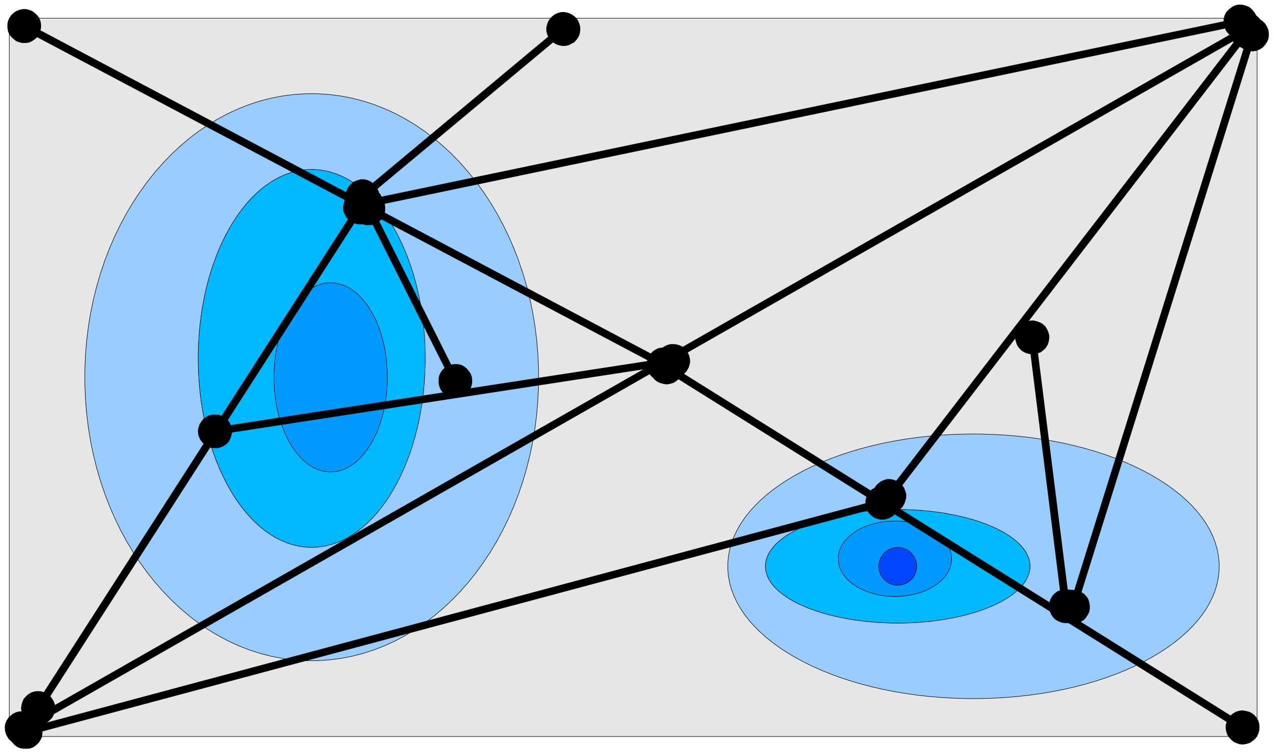
How to „map” the search space?



How to „map” the search space?



How to „map” the search space?



How to „map” the search space?

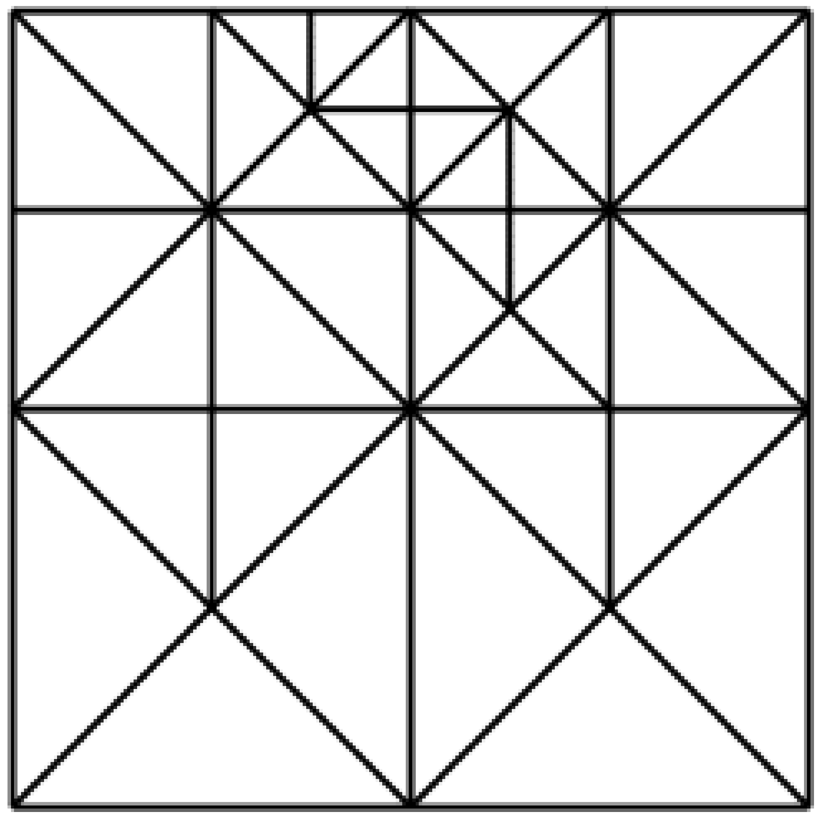


fig. 3: Adaptive triangulation

How to „map” the search space?

- A triangle or polygon estimates the fitness of its area by the mean of fitness at its vertices.
- Select the figure with high fitness (or sometimes with big volume), divide a triangle by adding a vertex on the edge with biggest fitness difference, divide a polygon by a diagonal joining two vertices each between vertices differing in fitness.
- When running short on memory, remove an edge with low fitness or low difference in fitness between figures it separates.

How to „map” the search space?

- This algorithm I've developed yesterday sounds promising, but is somewhat complicated... what is its maximal simplification?
- Keep just the points, and select an edge with high fitness to divide!
- This is a form of „phenotype recombination”.

How to „map” the search space?

- Random moving of a point to a close neighbor as in simulated annealing is „mutation”.
- Recombination and mutation are the basic operators of evolutionary algorithms (EAs).
- We can look at the population processed by an EA as an implicit adaptive map of the fitness in the search space.
- Or as a randomized hypothesis (prob. distribution) of where the true optimum is.
  - This hypothesis is updated with each generation of the EA.

Selfish Gene

- A pattern is a relation between a piece of information and particular objects, its „instances”.
- The information is „genotype”, and the instance is „phenotype”.
- Patterns at „time t+1” are the patterns that managed to put themselves there from „time t”. They „are fit”.
- Patterns „with individuality” are short enough so that they are seldom disrupted. To „be fit”, they often perform some function. Called **genes**.

Evolution and Modularity

- We would like to select for meaningful genes rather than individuals, because the pool of competing individuals is enormous, and for a gene (i.e. feature type), the pool of competing **alleles** (i.e. feature values) is relatively small.
- When the problem is modular (decomposable into features), and genes match to features, EAs with recombination search for alleles in parallel. When genes don't match to features, recombination is just a large-scale mutation.
- Example of a modular problem: TSP.

Evolution and Modularity

- Location of a city en route (i.e. first, or fifth, city) is not a good feature, it doesn't tell anything about the path.
- Edges are better features. But perhaps angles (corners, triples of cities) are even better?
- With a good set of features, it is often very difficult to design a good recombination operator, which should:
  - preserve features of parents, shouldn't introduce features not present in parents
  - allow to balance features from both parents, shouldn't force most features to come from the same parent

Evolution and Modularity

- The above conditions often cannot be met exactly (i.e. for TSP): see „Forma Analysis” by Nicholas J. Radcliffe.
- When recombination tends not to disrupt groups of related genes, it can give exponential speed-up compared to mutation only (and thus to non-EA search strategies).
  - This has been shown for Traveling Salesman Probl.
- Therefore, related genes should lay close in the genome. There are techniques to let this linkage evolve. A recent powerful one is „Bayesian Optimisation Algorithm”.

Genetic Programming (GP) and Modularity

- GP takes programs as the genotype. This sounds as a powerful idea: programs are the „ultimate representation” for solutions.
- But the original GP falls completely short on the issues we discussed so far (genes, recombination, linkage).
- Fortunately, there has been much progress:
  - NeuroEvolution of Augmenting Topologies (NEAT) [1]
  - Meta-Optimizing Semantic Evolutionary Search (MOSES) [2]
- These approaches learn new genes incrementally.

EAs in other mechanisms

- EAs are usually separate tools in a toolbox, but sometimes they are parts of other tools.
- One such tool is also biology inspired: Artificial Immunological Systems (AISs).
  - AISs use fuzzy matching of antibodies to detect pathogens.
  - AISs use *negative selection*, which eliminates detrimential antibodies (detecting organism own cells) in the early phase, and *positive selection* to increase affinity to detected pathogens.
- The other one I want to talk about are Learning Classifier Systems.

Learning Classifier Systems [3]

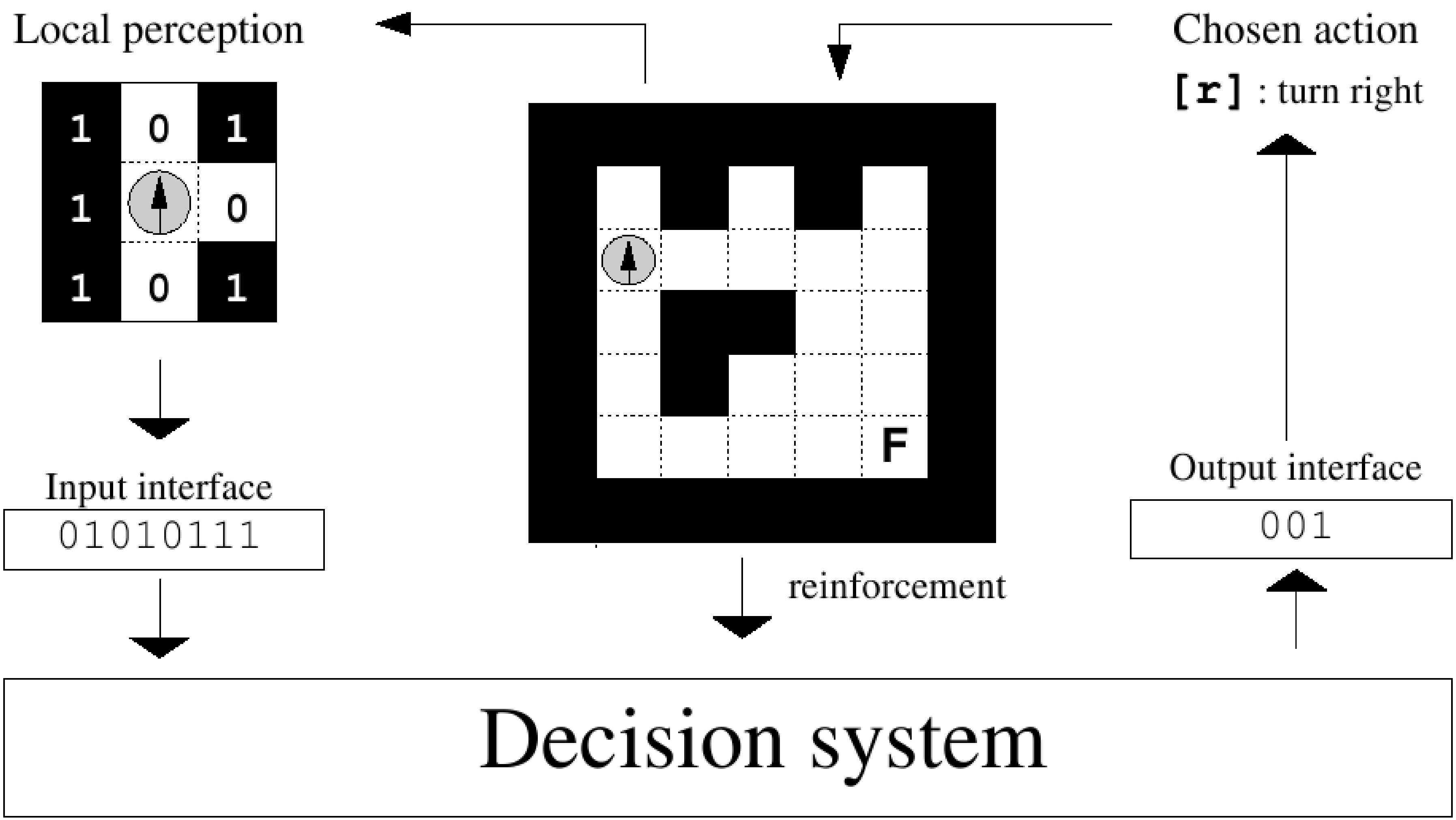
- Abstract Learning Classifier Systems (LCSs) are rule-based systems that automatically build their ruleset using Genetic Algorithm (GA).
- Reinforcement Learning (RL) learns what actions to pick in given states, knowing only the experienced rewards. LCS rules represent state-action(-reward) table in a compact way.
- RL must balance
  - exploration: trying out new actions to see what rewards they give
  - exploitation: following the action with biggest expected rewards (based on experience so far)

Learning Classifier Systems

There are several flavors of LCSs now:

- the „**Pittsburgh approach**” used GA on populations of whole systems; currently (after „Michigan approach”) a single LCS is evolved, using GA inside its learning mechanism
- **strength-based** systems (i.e. ZCS) use the expected reward of a rule (which matches state and produces action) as its fitness
- **accuracy-based** systems (i.e. XCS) represent expected reward in a rule and use accuracy of reward prediction as fitness; GA works only on classifiers matching current situation
- **anticipation-based** systems (ALCSs) don't use condition -> action rules, but condition, action -> effect rules: they build a state transition model; they favor actions bringing more information; often don't use GA but sophisticated specialization and generalization heuristics

Learning Classifier Systems



References and Sources

1. „*Evolving Neural Networks through Augmenting Topologies*”, Kenneth O. Stanley, Risto Miikkulainen, 2002
2. „*Competent Program Evolution*”, Moshe Looks, 2006
3. „*Learning Classifier Systems: A Survey*”, Olivier Sigaud, Stewart W. Wilson, 2007
4. Some pictures I grabbed from the internet.