CS 581 Notes

Intelligent (Autonomous) Agents

Agent

An agent is just something that acts (from the Latin agere, to do).

Of course, we would prefer "acting" to be:

- autonomous
- situated in some environment (that could be really complex)
- adaptive
- create and goal-oriented

Rational Agent

A rational agent is one that acts to achieve the best outcome, or when there is uncertainty, the best expected outcome[1].

AI: Constructing Agents

You can say that: AI is focused on the study and construction of agents that do the right thing.

Percepts and Percept Sequences

Percept – content/information that agent's sensors are perceiving / capturing currently

Percept Sequence – a complete history of everything that agent has ever perceived

- any practical issues that you can see here?
- what can a percept sequence be used for?

[1] no worries, we will make it a little less vague soon

Percepts, Knowledge, Actions, States

- Agent's choice of action / decision at any given moment:
  - CAN depend on:
    - built-in knowledge
    - entire percept sequence
  - CANNOT depend anything it hasn't perceived
- Agent's action CAN change the environment state

Knowledge is power, right?

Agent Function/Program

Specifying an action choice for every possible percept sequence would define an agent

- Action <-> percept sequence mapping IS the agent function.
- Agent function describes agent behavior.
- Agent function is an abstract concept.
- Agent program implements agent function.

Vacuum Cleaner Agent Example

Algorithm 1.1 Vacuum Cleaner Agent Example

```
1: function TABLE-DRIVEN-AGENT(percept) returns an action
2:     percepts: a sequence, initially empty
3:     table, a table of actions, indexed by percept sequences, initially fully specified
4:     append percept to the end of percepts
5:     action ← LOOKUP(percepts, table)
6:     return action
7: end function
8:
9: function REFLEX-VACUUM-AGENT([location, status]) returns an action
10:    if status = Dirty then return Suck
11:    else if location = A then return Right
12:    else if location = B then return Left
13:    end if
14: end function
```

Actions Have Consequences

- An agent can act upon its environment, but how do we know if the end result is "right"?
- After all, actions have consequences: either good or bad.
- Recall that agent actions change environment state!
- If state changes are desirable, and agent performs well.
- Performance measure evaluates state changes.

Performance Measure

A Tip

It is better to design performance measures according to what one actually wants to be achieved in the environment, rather than according to how one thinks the agent should behave.

A Warning

If it is difficult to specify the performance measure, agents may end up optimizing a wrong objective. Handle uncertainty well in such cases.

Rationality

Rational decisions at the moment depend on:

- The performance measure that defines success criteria
- The agent's prior knowledge of the environment
- The actions that the agent can perform
- The agent's percept sequence so far

Rational Agent

For each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has.

Rationality in Reality

- An omniscient agent will ALWAYS know the final outcome of its action. Impossible in reality. That would be perfection.
- Rationality maximizes what is EXPECTED to happen.
- Perfection maximizes what WILL happen.
- Performance can be improved by information gathering and learning.

Designing the Agent for the Task

Analyze the Problem

Task Environment — PEAS

In order to start the agent design process, we need to specify/define:

- The Performance measure
- The Environment in which the agent will operate
- The Actuators that the agent will use to affect the environment
- The Sensors

Task Environment Properties

Key dimensions by which task environments can be categorized:

- Fully vs partially observable (can be unobservable too)
- Single agent vs multi-agent
  - multi-agent: competitive vs. co-operative
- Deterministic vs. non-deterministic (stochastic)
- Episodic vs. sequential
  - Sequential requires planning ahead
- Static vs. dynamic
- Discrete vs. continuous
- Known vs. unknown (to the agent)

Select Agent Architecture

Agent = Architecture + Program

Typical Agent Architectures

- Simple reflex agent.
- Model-based reflex agent.
- Goal-based reflex agent.
- Utility-based reflex agent.

Select Internal Representations

Typical Agent Architectures

- Simple reflex agent: uses condition-action rules
- Model-based reflex agent: keeps track of the unobserved parts of the environment by maintaining internal state:
  - "how the world works": state transition model
  - how percepts and environment is related: sensor model
- Goal-based reflex agent: maintains the models of the world and goals to select decisions (that lead to goal)
- Utility-based reflex agent: maintains the model of the world and utility function to select PREFERRED decisions (that lead to the best expected utility: avg (EU * p))

State and Transition Representations

- Atomic: state representation has NO internal structure
- Factored: state representation includes fixed attributes (which can have values)
- Structured: state representation includes objects and their relationships

Problem-Solving / Planning Agent

- Context / Problem:
  - correct action is NOT immediately obvious
  - a plan (a sequence of actions leading to a goal) may be necessary
- Solution / Agent:
  - come up with a computational process that will search for that plan
- Planning Agent:
  - uses factored or structured representations of states
  - uses searching algorithms

Defining Search Problem

- Define a set of possible states: State Space
- Specify Initial State
- Specify Goal State(s) (there can be multiple)
- Define a FINITE set of possible Actions for EACH state in the State Space
- Come up with a TRANSITION model which describes what each action does
- Specify the Action COST Function (a function that gives the cost of applying action a to state s)

Measuring Searching Performance

Search algorithms can be evaluated in four ways:

- Completeness: Is the algorithm guaranteed to find a solution when there is one, and to correctly report failure when there is not?
- Cost optimality: Does it find a solution with the lowest path cost of all solutions?
- Time complexity: How long goes it take to find a solution? (in seconds, actions, states, etc.)
- Space complexity: How much memory is needed to perform the search?

Informed Search and Heuristics

Informed search relies on domain-specific knowledge / hints that help locate the goal state.

$$h(n) = h(\text{State } n)$$
$$h(n) = n(\text{relevant information about State } n)$$

Romanian Roadtrip: Heuristics h(n)

For this particular problem, the heuristic function $h(n)$ is defined by a straight line (Euclidean) distance between two states (cities). As the crow flies in other words.

A* Algorithm: Evaluations Function

Calculate/obtain:

$$f(n) = g(\text{State}_n) + h(\text{State}_n)$$

A* Evaluation Function

$$f(n) = g(\text{State}_n) + h(\text{State}_n)$$

where:

- $g(n)$ – initial node

Admissible Heuristic: Proof

An admissible heuristics $h()$ is guaranteed to give you the optimal solution. Why? Proof by contradiction:

- Say: the algorithm returned a suboptimal path ($C > C*$)
- So: there exists a node n on $C*$ not expanded on C:

If so:

$$\begin{aligned} & f(n) > C* \\ & f(n) = g(n) + h(n) && \text{(by definition)} \\ & f(n) = g*(n) + h(n) && \text{(because } n \text{ is on } C*) \\ & f(n) \leq g*(n) + h*(n) && \text{(if } h(n) \text{ admissible: } h(n) \leq h*(n) \end{aligned}$$

What Made A* Work Well?

- Straight-line heuristics is consistent: its estimate is getting better and better as we get closer to the goal
- Every consistent heuristics is admissible heuristics, but not the other way around

But that would mean that:

$$f(n) \leq C*$$

A*: Search Contours

How does A* "direct" the search progress?

Dominating Heuristics

We can have more than one available heuristics. For example $h_1(n)$ and $h_2(n)$. $h_2(n)$ dominates $h_1(n)$ iff[2] $h_2(n) > h_1(n)$ for every n.
If you have multiple admissible heuristics where none dominates the other:

$$\text{Let } h(n) = \max\left(h_1(n), h_2(n), \ldots, h_m(n)\right)$$

Domination $\rightarrow$ Efficiency: Why?

With

$$f(n) < C* \text{ and } f(n) = g(\text{State}_n) + h(\text{State}_n),$$

we get

Domination $\rightarrow$ Efficiency: But?

If $h_2(n)$ dominates $h_1(n)$, should you always use $h_2(n)$? Generally yes, but $h_2(n)$ vs $h_1(n)$ heuristic computation time may be a deciding factor here.

Heuristic and Search Performance

- Consider an 8-puzzle game and two admissible heuristics:
  - $h_1(n)$ – number of misplaced tiles (not counting blank)
  - $h_2(n)$ – Manhattan distance

[2] if and only if

$h()$ Quality: Effective Branching

Can We Make A* Even Faster? (Sometimes at a cost!)

Weighted A* Evaluation Function

$$f(n) = g(\text{State}_n) + W * h(\text{State}_n)$$

where:

- $g(n)$ – initial node to node n path cost
- $h(n)$ – estimated cost of the best path that continues from node n to a goal node
- $W > 1$

Here, weight W makes $h(\text{State}_n)$ (perhaps only "sometimes") inadmissible. It becomes potentially more accurate = less expansions!

Perfect Information

Two Player Games: Env Assumptions

Defining Zero Sum Game Problem

- Defining a set of possible states: State Space
- Specify how will you track Whose Move / Turn it is
- Specify Initial State
- Specify Goal State(s) (there can by multiple)
- Define a FINITE set of possible Actions (legal moves) for EACH state in the State Space
- Come up with a Transition Model which describes what each action does
- Come up with a Terminal Test that verifies if the game is over
- Specify the Utility (Payoff/Object) Function: a function that defines the final numerical value to player p when the game ends in the terminal state $s4$

MinMax Algorithm: The Idea

I don't know what move my opponent will choose, but I am going to ASSUME that it is going to be the best/optimal option for them.

α the value of the best (highest value)

β

Constraint Satisfaction Problem

The goal is to find an assignment (variable = value):

$$X_1 = v_1, \ldots, X_n = v_n$$

- If NO constraints violated: consistent assignment
- If ALL variables have a value: complete assignment
- If SOME variables have NO value: partial assignment
- SOLUTION: consistent and complete assignment
- PARTIAL SOLUTION: consistent and partial assignment

Local Search Algorithms

"Hill Climbing" (Greedy Local)

Assumption: We don't go to a repeated state

Do we always need to care about the path to the goal?

Informed Search: the Idea

When traversing the search tree, use domain knowledge / heuristics to avoid search paths (moves/actions) that are likely to be fruitless

Informed Search and Heuristics

Informed search relies on domain-specific knowledge/hints that help locate the goal state.

Hard Problems

- Many important problems are provably not solvable in polynomial time (NP and harder)
- Results based on the worst-case analysis
- In practice: instances are often easier
- Approximate methods can often obtain good solutions

Local Search

- Moves between configurations by performing local moves
- Works with complete assignments of the variables
- Optimization problems:

  - Start from a suboptimal configuration
  - Move towards better solutions
- Satisfaction problems:
  - Start from an infeasible configuration
  - Move towards feasibility
- NO guarantees
- Can work great in practice!

Local Search Algorithms

If the path to the goal does not matter, we might consider a different class of algorithms. Local Search Algorithms

- do not worry about paths at all.
- Local Search

Selecting Neighbor

- How to select the neighbor?
  - exploring the whole or part of the neighborhood
- Best neighbor
  - Select "the" best neighbor in the neighborhood
- First neighbor
  - Select the first "legal" neighbor
  - Avoid scanning the entire neighborhood
- Multi-stage selection
  - Select one "part" of neighborhood and then
  - select from the remaining "part" of the neighborhood
- Hill-climbing search
  - Gradient descent in continuous state spaces
  - Can use e.g. Newton's method to find roots
- Simulated annealing search

- Tabu search
- Local beam search
- Evolutionary/genetic algorithms

Although local search algorithms are not systematic, they have two key advantages:

- they use very little memory – usually a constant amount; and
- the can often find reasonable

Local search algorithms are useful for search pure optimization problems, in which the aim is to find the best state according to the objective function

Simulated Annealing

What Is It?

In metallurgy, annealing is the process used to temper or harden metals and glass by heating them to a high temperature T and then gradually cooling them, thus allowing the material to coalesce into a low-energy E crysalline state (less or no defects).
Key Idea:

- Use Metropolis algorithm but adjust the temperature dynamically
- Start with a high temperature (random moves)
- Decrease the temperature
- When the temperature is low, becomes a local search

Metropolis Heuristics

Basic Idea

- Accept a move if it improves the objective value
- Accept "bad moves" as well with some probability
- The probability depends on how "bad" the move is
- Inspired by statistical physics

How to choose the probability?

- t is a scaling parameter (called temperature)
- Δ is the difference $f(n) - f(s)$

Fixed T

- What happens for a large T?
  - Probability of accepting a degrading move is large
- What happends for a small T?
  - Probability of accepting a degrading move is small

Algorithm 2.1 Simulated Annealing Pseudocode

```
1: function SIMULATED-ANNEALING(problem, schedule) returns a solution state
2:     current ← problem.INITIAL
3:     t ← 1
4:     while True do
5:         T ← SCHEDULE(t)
6:         if T == 0 then return current
7:         end if
8:         next ← a randomly selected successor of current
9:         ΔE ← VALUE(current) − VALUE(next)
10:        if ΔE > 0 then
11:            current ← next
12:        else
13:            current ← next only with probability e^{−ΔE/T}
14:        end if
15:    end while
16: end function
```

Temperature/Cooling Schedule

Idea: start with

Summary

- Converges to a global optimum
  - connected neighborhood
  - slow cooling schedule

        * slower than the exhaustive search

- In practice
  - can give excellent results
  - need to tune a temperature schedule
  - default choice: $t_{k+1} = \alpha t_k$
- Additional tools
  - restarts and reheats

Applications

- Basic Problems
  - Traveling salesman
  - Graph partitioning
  - Matching problems
  - Graph coloring
  - Scheduling
- Engineering
  -

Heuristics and Metaheuristics

- Heuristics
  - how to choose the next neighbor?
  - use local information (state and its neighborhood)
  - direct the search towards a local min/maximum
- Metaheuristics
  - how to escape local minima?
  - direct the search towards a global min/maximum
  - typically include some memory or learning

Local Beam Search

The local beam search algorithm:

- keeps track of k states rather than just one
- begins with k randomly generated states
- at each step, all the successors of k states

The Idea

In a local beam search useful information is passed among the k parallel search threads. For example, if one state generates several good successors and the other $k-1$ states all generate bad successors, then the effect is that

Tabu Search

Key Features

- Always move to the best available neighborhood solution, even if it is worse than the current solution
- Maintain a list of solution points that must be avoided (not allowed) or a list of move features that are not allowed:
  - This is the Tabu List.
- Update the Tabu List based on some memory structure (short-term memory):
  - Remove Tabu moves after some time period has elapsed (Tenure).
- Allow for exceptions

Memory Structures

The memory structures used in Tabu Search can be divided into three categories:

Short-term The list of solutions recently considered. If a potential solution appears on this list, it cannot be revisited until it reaches an expiration point (Tenure).

Intermediate-term A list of rules intended to bias the search towards promising areas of the search space.

Long-term Rules that promote diversity in the search process (i.e. regarding resets when the search becomes stuck in a plateau or a suboptimal dead-end).

Aspiration Criteria

A criteria which allows a Tabu move to be accepted under certain conditions.
Most common aspiration criterion: If the move finds a new best solution, then accept the move even if the move is Tabu.

Tenure

The Tabu Tenure is the number of iterations that a move stays in the Tabu List

too small risk of cycling

too large may restrict the search too much

Intensification

Search parameters can be locally modified in order to perform intensification and/or diversification
Intensification – usually applied when no configurations with a quality comparable to that of stored elite configuration(s), have been found in the

Stopping Criteria

Potential Stopping Criteria:

- Number of iterations
- Number of iterations without improvement
-
-

Evolutionary Algorithms

Evolutionary Algorithms [Wikipedia]

An evolutionary algorithm (EA) is a subset of evolutionary computation, a generic population-based metaheuristic optimization algorithm.

An EA uses mechanisms inspired by biological evolution, such as reproduction, mutation, recombination, and selection. Candidate solutions to the optimization problem play the role of individuals in a population, and the fitness function determines the quality of the

Chromosome

Chromosome – a package of DNA with part of all of the genetic materials of an organism

Artificial Chromosome

In Evolutionary Algorithms, an artificial chromosome is a genetic representation of the task to be solved.

Typically:

$$1 \text{ individual} = 1 \text{ chromosome} = 1 \text{ solution}$$

Also called a genotype.

Evolutionary Algorithms [Wikipedia]

An evolutionary algorithm (EA) is a subset of evolutionary computation, a generic population-based metaheuristic optimization algorithm.

An EA uses mechanisms inspired by biological evolution, such as reproduction, mutation, recombination, and selection. Candidate solutions to the optimization problem play the role of individuals in a population, and the fitness function determines the quality of the solutions

(see also loss function).

Evolution of the population then takes place after the repeated application of the above operators.

Genetic Algorithm

Components

Table 4.1: Simple Genetic Algorithm

Representation	Bit strings
Recombination	1-point crossover
Mutation	Bit flip
Parent Selection	Fitness proportional
Survival Selection	Generational

Chromosome: Representation

Individuals / chromosomes can be represented as a string of values.

Binary Representation Issues

Table 4.2: Chromosome A

Variable/Gene = 512									
1	0	0	0	0	0	0	0	0	0

Table 4.3: Chromosome A

Variable/Gene = 0									
0	0	0	0	0	0	0	0	0	0

Small, one bit, change can lead to drastic fitness changes. Solution: use Gray Code.

Gray Code

The reflected binary code (RBC), also known as reflected binary (RB) or Gray Code (named after Frank Gray – Bell Labs), is an ordering of the binary numeral system such that two successive values differ in only one bit (binary digit).

Table 4.4: Gray Code

Decimal	Binary	Gray	Decimal	Binary	Gray
0	0000	0000	7	0111	0100
1	0001	0001	8	1000	1100
2	0010	0011	9	1001	1101
3	0011	0010	10	1010	1111
4	0100	0110	11	1011	1110
5	0101	0111	12	1100	1010
6	0110	0101	13	1101	1011

Hamming Distance

The Hamming distance between two equal-length strings of symbols is the number of positions at which the corresponding symbols are different.

Gray code: subsequent numbers → Hamming distance of 1

Integer Representation

- Some problems naturally map (8-queens) to integer representations
  - solution is an assignment variable = integer value
- Unrestricted: any value is permissible
- Restricted: only values from a certain set / domain
  - for example $\{0, 1, 2, 3\}$ for {North, East, South, West}
- Consider:
  - is there any relationship between values (e.g. 3 is more like 4 than 789 → ordinal relationship) or not ({North, East, South, West})
  - Choose your recombination / mutation strategy accordingly

Permutation Encoding

In permutation encoding, every chromosome is a string of numbers that represent a position in a sequence.

Permutation encoding is useful for ordering problems. For some types of crossover and mutation corrections must be made to leave the chromosome consistent (i.e. have valid sequence in it) for certain problems.

Genetic Algorithm: Other Selection Mechanisms

Elitism

Elitism (Elitist selection) is a strategy in genetic algorithms (in practice: evolutionary algorithms in general) where one or more most fit individuals (the elites) in every generation, are inserted into the next generation without undergoing any change.
This strategy usually speeds up the convergence of the algorithm.

Steady State Selection

In every generation, few chromosomes are selected (good - with high fitness) for creating a new offspring.

Then some (bad - with low fitness) chromosomes are removed and the new offspring is placed in their place.

The rest of population survives to new generation.

Fitness Proportional Selection: Issues

Fitness Proportional Selection approach has issues:

- outstanding individuals take over the entire population quickly; this is called premature convergence
- potential solutions
- when fitness values are all very close together, there is almost no selection pressure [= selection is almost uniformly random] → performance increases slowly
- Potential solutions:
  - windowing (slide and subtract some value based on history)
  - Goldberg's sigma scaling: $f'(x) = \max(f(x) - (f_{avg} - 2 \times \sigma_f), 0.0)$
  - ranking

Rank Based Selection

In rank selection, the selection probability does not depend directly on the fitness, but on the fitness rank of an individual within the population. The exact fitness values themselves do not have to be available, but only a sorting of individuals according to quality.

- Linear ranking:

$$P(R_i) = \frac{1}{n}\left(sp - (2sp-2)\frac{i-1}{n-1}\right) \quad 1 \leq i \leq n, \;\; 1 \leq sp \leq 2 \text{ with } P(R_i) \geq 0, \;\; \sum_{i=1}^{n} P(R_i) = 1$$

  sp – selection pressure (the degree to which the better individuals are favored: the higher the selectrion pressure, the more the better individuals are favored) which can take values between 1.0 (no selection pressure) and 2.0 (high selectrion pressure)

- Exponential ranking

Replace worst

- In this scheme, the worst k members of the population are selected for replacement
- This can lead to rapid improvements in the average population fitness, but can also cause premature convergence
  - it will focus on most fit individual
- Typically used with large populations

Age-Based Replacement

- Rather than look at fitness of the individual, pick the oldest (in iterations) first to replace
- A FIFO queue will be needed

Other Recombination Methods

"Cut and Crossfill"

1. Pick a random crossover point
2. Cut both parents in two segments after this position
3. Copy the first segment of Parent 1 into Child 1 and the first segment of Parent 2 into Child 2
4. Scan Parent 2 from left to right and fill the second segment of Child 1 with values from Parent 2, skipping those that are already contained in it
5. Do the same for Parent 1 and Child 2

Multiparent Recombination

- The idea: recombine genes of more than 2 parents
- Some strategies:
  - Allele frequency-based: ρ-sexual voting
  - segmentation and recombination of parents-based: the diagonal crossover
  - Based on numerical operations on real-values alleles

Other Recombination Options

- Integer representation:
  - same approaches as for binary
- Floating-point representation:
  - simple recombination
  - simple arithmetic recombination
  - whole arithmetic recombination
    - Child 1: $\alpha \times x + (1 - \alpha) \times y$ and Child 2: $\alpha \times y + (1 - \alpha) \times x$

Integer representation

Same approaches as for binary

Floating-point representation:

- Simple recombination
- Simple arithmetic recombination
- Whole arithmetic recombination

Child 1: αx

Other Mutation Mechanisms

Scramble mutation

A subset of genes is chosen and their values randomly shuffles/scrambled.

Inversion mutation

A subset of genes, a substring is selected and inverted.

Interchage (order changing) mutation

Randomly select two positions of the chromosome and interchange the values.

Selected Problems

8-Queens Problem

Artificial Neuron (Perceptron)

A (single-layer) perceptron is a model of a biological neuron. It is made of the following components:

- inputs x_i - numerical values representing information
- weights w_i

Genetic Programming

Traditional Programming vs . . .

Traditional Programming:

$$\text{Input Data} + \text{Program (Rules)} = \text{Output}$$

. . . :

$$\text{Input Data} + \text{Output} = \text{Program (Rules)}$$

Genetic Algorithm

Fitness Function

- Has to be clearly

Evolutionary Algorithms: Speciation

- **Nature** – speciation occurs when two similar reproducing beings evolve to become too dissimilar to share genetic information effectively or correctly.
  - they are incapable of mating to produce offspring.
    - Example: a horse and a donkey mating to produce a mule. However, in this case, the Mule is usually infertile, and so the genetic isolation of the two parent species is maintained.
- **Implementation** – speciation→

Genetic Programming

- Genetic programming (GP) is an automated method for creating a working computer program from a high-level problem statement of a problem.
- Genetic programming starts from a high-level statement of "what needs to be done" and

Goal: find a program that generates the correct solution

- based on input – correct output data

Genotype:

Preparation

- Determine the set of terminals.
- Select the set of primitive functions.
- Define the fitness function.
- Decide on the parameters for controlling the run.
- Choose the method for designating a result of the run.

Problem Description

Table 7.1:

Objective	Find a computer program with one input X for which the output Y

Mutation

Randomly choose :

Other (Key) Evolutionary Approaches

Evolutionary Programming

- Similar to genetic programming, but the solution is a set parameters for a predefined fixed computer program, not a generated computer program
- Solution fitness is determined by how well the fixed

Fuzzy Logic

Imagine load application processing rules

If credit score good then risk is low
If credit score bad then risk is high
If credit score medium then risk is average

What does it mean: low, high, good bad?

Genetic Fuzzy Systems

Swarm Intelligence/Optimization

Swarm Intelligence (SI) is the collective behavior of decentralized, self-organized systems, natural or artificial. The concept is employed in work on artificial intelligence.

Emergence

In philosophy, systems theory, science, and art, emergence occurs when

Ant Colony Optimization

Stigmergy

Stigmergy (coined by French biologist Pierre-Paul Grasse) = interation through the environment

Pheromone Trail

TSP with Ant Colony Optimization

1. Initialization (ants, pheromone trails)
2. Randomly place ants at nodes
3. Build tours / paths
4. Deposit pheromone, update trail, solution
5. Repeat or exit

Select Next Node For Each Ant

Each ant will pick its next destination:

- unvisited node
- choice will be based on:
  - pheromone intensities d_{ij} on all available paths
  - heuristic value h_{ij} for all available paths (a distance between nodes)

$$P(move\ i \rightarrow j) = \frac{d_{ij}^{\alpha} \times \frac{1}{h_{ij}}^{\beta}}{\sum_{k=1}^{N\ possible\ destinations} d_{ik}^{\alpha} \times \frac{1}{h_{ik}}^{\beta}}$$

where

$\alpha =$ pheromone weight "intensity"

$\beta =$

Update Pheromone Trail

Update pheromone trail values for each edge:

- decrease due to pheromone evaporation (evaporation rate, for example 0.5)

$$* = 0.5$$

-

Random Experiment

- The agent needs reason in an uncertain world
- Uncertainty can be due to
  - Noisy sensors (e.g., temperature, GPS, camera, etc.)
  - Imperfect data (e.g., low resolution image)
  - Missing data (e.g., lab tests)
  - Imperfect knowledge (e.g., medical diagnosis)
  - Exceptions (e.g., all birds fly except ostriches, penguins, birds with injured winds, dead birds, . . .)
  - Changing data (e.g., flu seasons, traffic conditions, etc.)
  - . . .
- The agent still must act (e.g., step on the breaks, diagnose a patient, order a lab test, . . .)

Random Experiment is a process by which we observing something uncertain.

An outcome is a result of a random experiment.

The set of all possible outcomes is called the sample space S (frequently labeled Ω).

Probability In AI: Selected Application

- Classification
  - Naïve Bayes, logistic regression, neural networks
  - Maximum likelihood estimation, Bayesian estimation, gradient optimization, back-propagation
- Decision making

  - Episodic decision making, Markov decision processes, multi-armed bandits
  - Value of information, Bellman equations, value iteration, policy iteration, UCB1, ϵ-greedy
- Reinforcement learning
  - Prediction, control, Monte-Carlo methods, temporal difference learning, Sarsa, Q-learning

Outcomes / Sample Space / Event

Outcome – A result of a random experiment

Sample space S – The set of all possible outcomes

Event – A subset of the sample space S

Events

Union and Intersection

If A and B are events, then

$$A \cap B$$

and

$$A \cup B$$

are also events.

$\cup$ union ("or")

$\cap$ intersection ("and")

Simple event – An event that cannot be decomposed

Complementary Event

The complement of any event A is the event A' ("not A"), i.e. the event that A does not occur.

Emergence [Wikipedia]

In philosophy, systems theory, science, and art, emergence occurs when a complex entity has properties or behaviors that its parts do not have on their own, and emerge only when they interact in a wider whole.

Emergence plays

Flock and Schools

Socio-cognitive Underpinnings

Universal individual behavior principles:

Evaluate tendency to evaluate stimuli (positive/negative, attractive/repulsive) is shared among all kinds of living organisms

Compare Comparing to others is a driver for learning and change

Imitate Effective strategy for learning, through not many living organisms are capable of full imitation

Rules

Emergent behavior in flocks and schools can be reduced to simple rules:

Separation an individual should avoid crowding or colliding with its neighbors

Alignment an individual should steer in the average heading of its neighbor

Cohesion

Particle Swarm Optimization (PSO)

In computational science, particle swarm optimization (PSO) is a computational method that optimizes a problem by iteratively

Particle Properties

Current position

Best position

Velocity

Parameters

- Number of particles K
- c_1 significance of personal particle experience (trust in individual knowledge)
- c_2 significance of swarm experience (trust in social knowledge)
- Inertia weight w
  - Sort of like a learning rate
- $V_{\max}$ velocity cap
- N_i neighborhood of particle i

Cycle

Create a swarm (population) of K particles initialized with:

- random position (point) in search space
  - best if uniformly distributed over space
- velocity set to:

  - zero or
  - random in $[-V_{\max}, V_{\max}]$

- Use fitness function to determine how fit (how well it solves the problem) each particle is initially.
- Update particle position (point in search space) and velocity while balancing exploitation and exploration.

Where Next?

- $x^t_{G,\ best}$=swarm's (global) best position
- $x^t_{i,\ best}$=Particle i's best position

Particle Position Update

Next position:

$$x_i^{t+1} = x_i^t + \vec{v}_i^{t+1}$$

where

$\mathbf{x_i^{t+1}}$ i's next position

$\mathbf{x_i^t}$ i's current position

$\mathbf{\vec{v}_i^{t+1}}$ i's next velocity

Next velocity:

$$\vec{v}_i^{t+1} = w \times \vec{v}_i^t + c_1 \times a \times (x^t_{i,\ best} - x_i^t) + c_2 \times b \times (x^t_{G,\ best} - x_i^t)$$

where

$a =$ random number in $[0; 1]$

$b =$ random number in $[0; 1]$

Also common:

$$\vec{v}_i^{t+1} = \vec{v}_i^t + c_1 \times a \times (x^t_{i,\ best} - x_i^t) + c_2 \times b \times (x^t_{G,\ best} - x_i^t)$$

Psychosocial Compromise

Check the termination condition

Controlling the Search

- If velocity is too low → algorithm too slow
- If velocity is too → algorithm too unstable

c_1 and c_2 Parameters

Should add up to 4.

$V_{\max}$ Velocity Cap

To better control particle trajectory and prevent stochastic velocity change to have uncontrolled

Diversification

Particles need time to

Characteristics

PSO has a memory:

- "where" the best solution was (as opposed to "what")

Quality population respond

Advantages / Disadvantages

Advantages

- Insensitive to scaling of design variables
- Simple implementation
- Easily parallelized

Variants / Modifications

- 2-D Otsu PSO
- Active Target PSO
- Adaptive PSO

Heuristics and Metaheuristics

Heuristics

- how to choose the next neighbor?
- use local information (state and its neighborhood)
- direct the search towards a local maximum

Other Swarm Algorithms

- Bat Algorithm
- Artificial Fish Swarm
- Cuckoo Swarm