IS-ZC444: ARTIFICIAL INTELLIGENCE

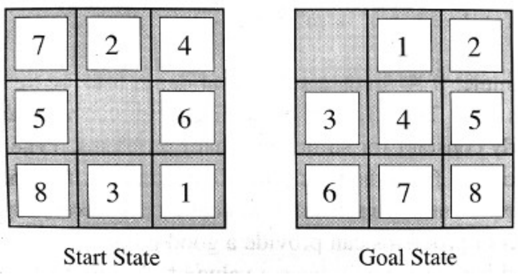
Lecture-07: Beyond Classical Search

Dr. Kamlesh Tiwari
Assistant Professor
Department of Computer Science and Information Systems,
BITS Pilani, Pilani, Jhunjhunu-333031, Rajasthan, INDIA

Sept 27, 2020 FLIPPED (WILP @ BITS-Pilani Jul-Nov 2020)

Heuristic Function

Consider 8-puzzle problem



- Average solution cost is 22 steps
- Branching factor is $\sim$3, so $3^{22} = 3.1 \times 10^{10}$ states in tree
- However, graph search would have $9!/2 = 181,440$ states[1]
- To find shortest solution using A*, we need to find appropriate heuristic functions

[1] 15-puzzle has 10^{13} states

Heuristic Function for 8-puzzle

h_1: number of misplaced tiles

h_2: sum of the distances of the tiles from their goal position[a]

[a] Since blocks cannot move along diagonal, distance would involve horizontal and vertical moves (city-block or Manhattan distance)

- For our example $h_1 = 8$, $h_2 = 18$
- **Effective branching factor (b^*):** if the algorithm generates N number of nodes and the solution is found at depth d, then

$$N + 1 = 1 + (b^*) + (b^*)^2 + (b^*)^3 + ... + (b^*)^d$$

you get N and d when you run the algorithm. Substitute N and d in above equation to get b^*

Effective branching factor can characterize the quality of heuristic

Heuristic Function for 8-puzzle

Experiment: A* algorithm is run on 1200 random problems with solution length 2 to 24 (100 for each even number).

	Search Cost (nodes generated)			Effective Branching Factor		
d	IDS	$A^*(h_1)$	$A^*(h_2)$	IDS	$A^*(h_1)$	$A^*(h_2)$
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	3644035	227	73	2.78	1.42	1.24
14	–	539	113	–	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

$h_2(n) \geq h_1(n)$ so h_2 dominates h_1.

From the table it is clear that **h_2 is better than h_1**

It is better to use heuristic function with larger values

Admissible Heuristics from relaxed problem

Consider following relaxations (relaxed 8-puzzle problem)

1. Tile could move anywhere (not only one square towards adjacent empty block). h_1 would give exact number of steps
2. Tile could move one square in any direction, even into an occupied square. h_2 would give exact number of steps

State space graph of relaxed problem is a supergraph of original state space because of removal of restrictions

- Solution in original problem space would work for relaxed setting
- However there could be a better solution in relaxed setting (having lower cost)

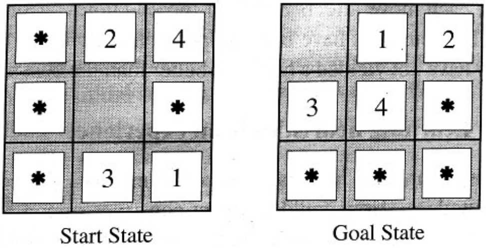
lower cost? means no overestimate on actual cost

A solution in relaxed problem space could be used as heuristic

$$h(n) = max\{h_1(n), h_2(n), .., h_m(n)\}$$

Admissible Heuristics from subproblem

Following is a subproblem (only arrange 1-2-3-4)



- **Pattern database** stores each subproblem and solution cost
- Solution cost could be used as a heuristic
- we could also make database for 5-6-7-8
- But we cannot add the costs of 1-2-3-4 and 5-6-7-8
- **Disjoint pattern database** solution cost is taken as moves involving only 1,2,3,4 in the solution path. Now we can add cost of 1-2-3-4 and 5-6-7-8 database

Learning Heuristics from Experience

- Experience means solving lots of 8-puzzle problem
- Each optimal solution provides examples from which $h(n)$ could be learned
- Using decision tree, neural nets or something else..
- Instead of state n we have to use **feature** say $x_1(n)$ is number of misplaced tiles
- $x_2(n)$ could be number of pairs of adjacent tiles that are not adjacent in the goal state

Heuristic would be obtained by combining $x_1(n)$, $x_2(n)$, ...

One such approach could be to use a liner combination

$$h(n) = c_1 x_1(n) + c_2 x_2(n) + \ldots$$

Beyond Classical Search

We have seen **systematic search** algorithms for problems that are observable and deterministic. That may not be for real-world problem.

Beyond Classical Search

We have seen **systematic search** algorithms for problems that are observable and deterministic. That may not be for real-world problem.

- In many problems, path to the goal is irrelevant.
  - 8-queens - order of adding queens is not important
  - Integrated circuit design
  - Factory floor layout
  - Job shop scheduling
  - Automatic programming
  - Telecommunication network optimization
  - Vehicle Routing
  - Portfolio management

Beyond Classical Search

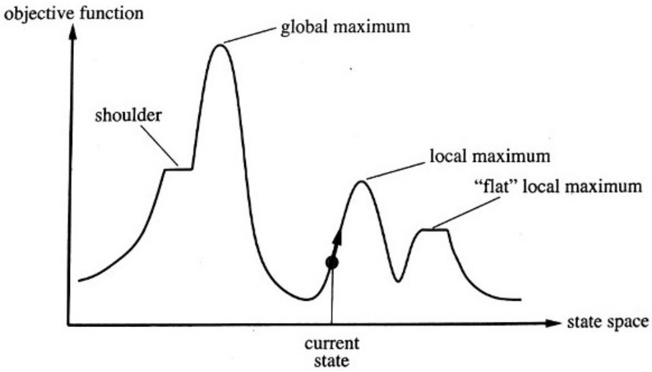
We have seen **systematic search** algorithms for problems that are observable and deterministic. That may not be for real-world problem.

- In many problems, path to the goal is irrelevant.
  - 8-queens - order of adding queens is not important
  - Integrated circuit design
  - Factory floor layout
  - Job shop scheduling
  - Automatic programming
  - Telecommunication network optimization
  - Vehicle Routing
  - Portfolio management
- We go for **local search** algorithms in such scenarios
- These algorithms operate using a single current node (and move to its neighbor). Advantages include
  1. Uses very little memory
  2. Can find reasonable solution in large or infinite (continuous) state space.

Local Search Algorithms

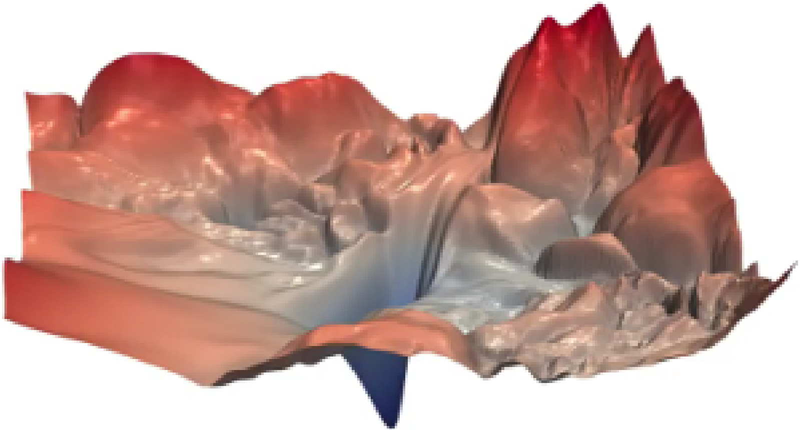
- Local search algorithms can solve pure optimization problems.
- They find optimum state based on some **objective function** [2]

State space landscape can be used to proceed.



[2] Many optimization problems do not fit in standard search model. For example Darwinian evolution uses an objective function **reproductive fitness** but there is no goal test and path cost for this.

Real State Space Landscape



Hill Climbing Search

- Steepest ascent version
- Look around and choose the best
- Greedy-local search
- Makes quite rapid progress

Algorithm 1: Hill-Climbing(problem)

Returned a state of *problem* that is local maximum.

current $\leftarrow$ Make-Node (*problem*.Initial-State)

while *not terminate* **do**

  neighbor $\leftarrow$ a highest valued successor of current

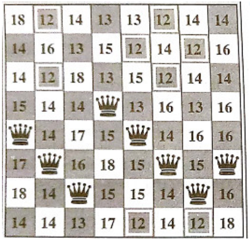
  if *neighbor.Value* $\leq$ *current.Value* **then**

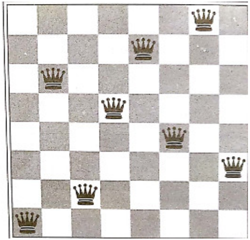
    return current.State

  current $\leftarrow$ neighbor

Hill Climbing Search

- Consider 8-queens problem
- In **complete-state formulation** has 8-queens in each state (one per column)
- Successor of a state are all possible states generated by moving a single queen to another square (each queen have 7 moves) so $8 \times 7 = 56$ successors.
- Heuristic cost function h is the number of pairs of queens that are attacking each other ($h = 17$ to $h = 1$)





Gets Stuck

Sometime algorithm reaches to a state where no further progress could be made.

Due to

1. Local Maxima
2. Ridges
3. Plateaux

Gets Stuck

Sometime algorithm reaches to a state where no further progress could be made.

Due to

1. Local Maxima
2. Ridges
3. Plateaux

Starting from randomly generated 8-queens state, hill climbing gets stuck 86% of time. Taking 4-steps when succeed and 3-steps when stuck (8^8 states in state space)

Gets Stuck

Sometime algorithm reaches to a state where no further progress could be made.

Due to

1. Local Maxima
2. Ridges
3. Plateaux

Starting from randomly generated 8-queens state, hill climbing gets stuck 86% of time. Taking 4-steps when succeed and 3-steps when stuck (8^8 states in state space)

Allow **sideway moves** in this scenario

Gets Stuck

Sometime algorithm reaches to a state where no further progress could be made.

Due to

1. Local Maxima
2. Ridges
3. Plateaux

Starting from randomly generated 8-queens state, hill climbing gets stuck 86% of time. Taking 4-steps when succeed and 3-steps when stuck (8^8 states in state space)

Allow **sideway moves** in this scenario

How many sideway moves? ∞

Gets Stuck

Sometime algorithm reaches to a state where no further progress could be made.

Due to

1. Local Maxima
2. Ridges
3. Plateaux

Starting from randomly generated 8-queens state, hill climbing gets stuck 86% of time. Taking 4-steps when succeed and 3-steps when stuck (8^8 states in state space)

Allow **sideway moves** in this scenario

How many sideway moves? ∞
Put a limit on this (say 100) Now it solves 94% problems. But, takes 21 steps to solve and 64 for failure on an average.

Hill Climbing Variants

Stochastic hill climbing

Instead of the best one, choose a path with probability of its steepness (converge slowly).

Hill Climbing Variants

Stochastic hill climbing

Instead of the best one, choose a path with probability of its steepness (converge slowly).

First-choice hill climbing

Do not explore the state-space when you see the first improvement.

Hill Climbing Variants

Stochastic hill climbing

Instead of the best one, choose a path with probability of its steepness (converge slowly).

First-choice hill climbing

Do not explore the state-space when you see the first improvement.

Random-restart hill climbing

if probability of success is p then run $1/p$ number of times.

- For our previous case [a] it needs 7 iterations.
- and 22 steps

[a] With random start 8-queens state, hill climbing gets stuck 86% of time.

Simulated Annealing (Metallurgy approach)

- Simulated annealing is hill climbing combined with **random walk**
- Step size is gradually reduced
- First Applied around 1980, for VLSI layout problem

Algorithm 2: Simulated-Annealing(problem, schedule)

current $\leftarrow$ Make-Node (*problem*.Initial-State)
for $t = 1$ *to* ∞ **do**
    T $\leftarrow$ *schedule(t)*
    if $T = 0$ **then return** current
    next $\leftarrow$ a randomly selected successor of current
    $\Delta E \leftarrow$ next.Value - current.Value
    if $\Delta E > 0$ **then** current $\leftarrow$ next
    else current $\leftarrow$ next with probability $e^{\Delta E/T}$

Genetic Algorithms

Evolution

- Recall Darwin's theory of evolution: "*Survival of the fittest*" [3]

Genetic Algorithms

Evolution

- Recall Darwin's theory of evolution: "*Survival of the fittest*" [3]





Genetic Algorithms

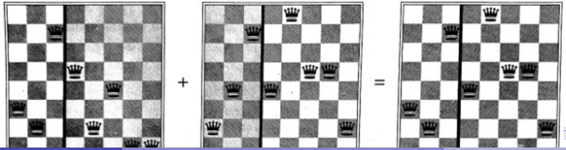
Evolution

- Recall Darwin's theory of evolution: "*Survival of the fittest*" [3]





Is it true? Let's formulate and try..



Genetic Algorithms

Learning approach of Genetic algorithms is based on simulated evolution (appeared in 1975)

Genetic Algorithms

Learning approach of Genetic algorithms is based on simulated evolution (appeared in 1975)

- State are represented using fixed length bit strings (chromosome)

Genetic Algorithms

Learning approach of Genetic algorithms is based on simulated evolution (appeared in 1975)

- State are represented using fixed length bit strings (chromosome)
- Search for a goal state begins with a population of initial states

Genetic Algorithms

Learning approach of Genetic algorithms is based on simulated evolution (appeared in 1975)

- State are represented using fixed length bit strings (chromosome)
- Search for a goal state begins with a population of initial states
  - Members of the current population give rise to the next generation population using random **mutation** and **crossover**

Genetic Algorithms

Learning approach of Genetic algorithms is based on simulated evolution (appeared in 1975)

- State are represented using fixed length bit strings (chromosome)
- Search for a goal state begins with a population of initial states
  - Members of the current population give rise to the next generation population using random **mutation** and **crossover**
  - States are evaluated using some **fitness** measure

Genetic Algorithms

Learning approach of Genetic algorithms is based on simulated evolution (appeared in 1975)

- State are represented using fixed length bit strings (chromosome)
- Search for a goal state begins with a population of initial states
  - Members of the current population give rise to the next generation population using random **mutation** and **crossover**
  - States are evaluated using some **fitness** measure
  - Most fit state act as a seeds for producing next generation

Genetic Algorithms

Learning approach of Genetic algorithms is based on simulated evolution (appeared in 1975)

- State are represented using fixed length bit strings (chromosome)
- Search for a goal state begins with a population of initial states
  - Members of the current population give rise to the next generation population using random **mutation** and **crossover**
  - States are evaluated using some **fitness** measure
  - Most fit state act as a seeds for producing next generation
- Applied a variety of learning tasks and optimization problems (like robot control and learning parameters for ANN)
- Search can move abruptly. Crowding can happen

Genetic Algorithms

Learning approach of Genetic algorithms is based on simulated evolution (appeared in 1975)

- State are represented using fixed length bit strings (chromosome)
- Search for a goal state begins with a population of initial states
  - Members of the current population give rise to the next generation population using random **mutation** and **crossover**
  - States are evaluated using some **fitness** measure
  - Most fit state act as a seeds for producing next generation
- Applied a variety of learning tasks and optimization problems (like robot control and learning parameters for ANN)
- Search can move abruptly. Crowding can happen

Without **guarantee**, GA often finds an object of high fitness

Genetic Algorithms

Algorithm 3: GA (`Fitness`, F_{th}, p, r, m)

P $\leftarrow$ generate p states at random
while $max(\texttt{Fitness}(h_1), \texttt{Fitness}(h_2), ..., \texttt{Fitness}(h_p)) < F_{th}$ **do**
  Select: $(1 - r)p$ members of P
  Crossover: on $(r \times p)/2$ pairs to produce two offspring
  Mutation: randomly invert a bit of m percentage of population
return state h_i having maximum $\texttt{Fitness}(h_i)$

Genetic Algorithms

Algorithm 4: GA ($\texttt{Fitness}$, F_{th}, p, r, m)

P $\leftarrow$ generate p states at random
while $max(\texttt{Fitness}(h_1), \texttt{Fitness}(h_2), ..., \texttt{Fitness}(h_p)) < F_{th}$ **do**
  Select: $(1 - r)p$ members of P
  Crossover: on $(r \times p)/2$ pairs to produce two offspring
  Mutation: randomly invert a bit of m percentage of population
return state h_i having maximum $\texttt{Fitness}(h_i)$

- Fitness function is typically a heuristic
- The fitness function is a criterion for ranking states to select states probabilistically for inclusion in the next generation population

Selection

- **Fitness proportionate.** Probability of selecting a state in next generation is

$$Pr(h_i) = \frac{\texttt{Fitness}(h_i)}{\sum_{j=1}^{p} \texttt{Fitness}(h_j)}$$

Selection

- **Fitness proportionate.** Probability of selecting a state in next generation is

$$Pr(h_i) = \frac{\mathtt{Fitness}(h_i)}{\sum_{j=1}^{p} \mathtt{Fitness}(h_j)}$$

- **Tournament selection.** randomly pick two states and then with some predefined probability p the more fit of these two is then selected, and with probability $(1 - p)$ the less fit state is selected

Selection

- **Fitness proportionate.** Probability of selecting a state in next generation is

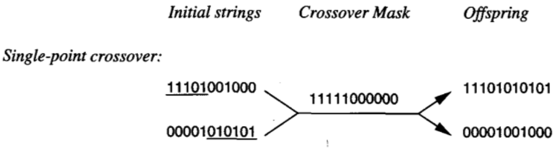
$$Pr(h_i) = \frac{\texttt{Fitness}(h_i)}{\sum_{j=1}^{p} \texttt{Fitness}(h_j)}$$

- **Tournament selection.** randomly pick two states and then with some predefined probability p the more fit of these two is then selected, and with probability $(1 - p)$ the less fit state is selected
- **Rank selection.** states are sorted by fitness and the probability of selection of a state is proportional to its rank in this sorted list, rather than its fitness

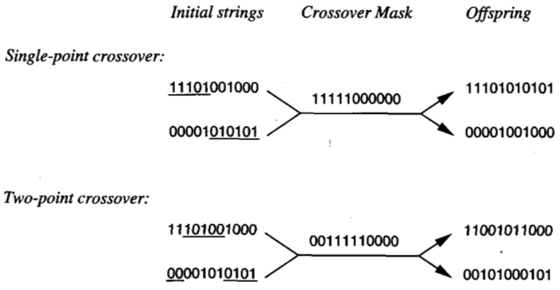
Selection

- **Fitness proportionate.** Probability of selecting a state in next generation is
$$Pr(h_i) = \frac{\texttt{Fitness}(h_i)}{\sum_{j=1}^{p} \texttt{Fitness}(h_j)}$$
- **Tournament selection.** randomly pick two states and then with some predefined probability p the more fit of these two is then selected, and with probability $(1 - p)$ the less fit state is selected
- **Rank selection.** states are sorted by fitness and the probability of selection of a state is proportional to its rank in this sorted list, rather than its fitness
- **Elitist Model.** select a small proportion of the fittest candidates in current population intact into the next generation

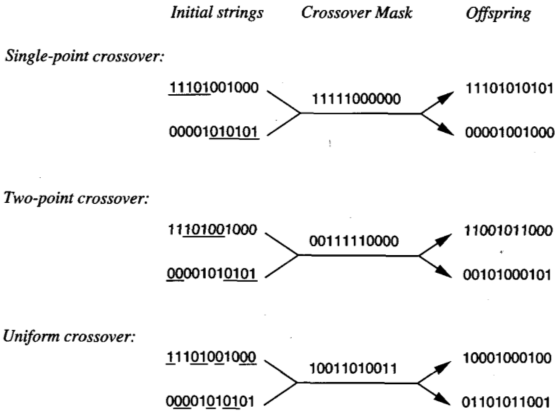
Genetic Operators (crossover and mutation)



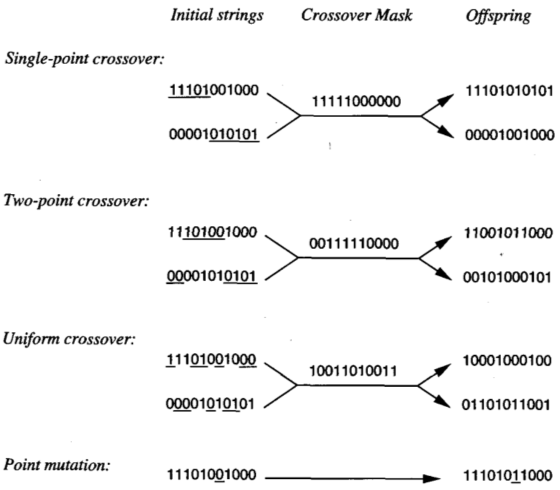
Genetic Operators (crossover and mutation)



Genetic Operators (crossover and mutation)



Genetic Operators (crossover and mutation)



Does GA works?

- Can we mathematically characterize the evolution

Does GA works?

- Can we mathematically characterize the evolution
- **Schema:** string of 0, 1 or $*$

Does GA works?

- Can we mathematically characterize the evolution
- **Schema:** string of 0, 1 or $*$ like $0 * 1$

Does GA works?

- Can we mathematically characterize the evolution
- **Schema:** string of 0, 1 or $*$ like $0 * 1$ denoting set $\{001, 011\}$
- String 1011 is **representative** of 2^4 schema

Does GA works?

- Can we mathematically characterize the evolution
- **Schema:** string of 0, 1 or $*$ like $0 * 1$ denoting set $\{001, 011\}$
- String 1011 is **representative** of 2^4 schema
- Let $m(s, t)$ be number of instances of schema s in generation t

Does GA works?

- Can we mathematically characterize the evolution
- **Schema:** string of 0, 1 or $*$ like $0 * 1$ denoting set $\{001, 011\}$
- String 1011 is **representative** of 2^4 schema
- Let $m(s, t)$ be number of instances of schema s in generation t
- Consider *selection*, let fitness of individual h be $f(h)$

Does GA works?

- Can we mathematically characterize the evolution
- **Schema:** string of 0, 1 or $*$ like $0*1$ denoting set $\{001, 011\}$
- String 1011 is **representative** of 2^4 schema
- Let $m(s, t)$ be number of instances of schema s in generation t
- Consider *selection*, let fitness of individual h be $f(h)$ and average fitness of whole n size population

Does GA works?

- Can we mathematically characterize the evolution
- **Schema:** string of 0, 1 or $*$ like $0 * 1$ denoting set $\{001, 011\}$
- String 1011 is **representative** of 2^4 schema
- Let $m(s, t)$ be number of instances of schema s in generation t
- Consider *selection*, let fitness of individual h be $f(h)$ and average fitness of whole n size population at time t

Does GA works?

- Can we mathematically characterize the evolution
- **Schema:** string of 0, 1 or $*$ like $0 * 1$ denoting set $\{001, 011\}$
- String 1011 is **representative** of 2^4 schema
- Let $m(s, t)$ be number of instances of schema s in generation t
- Consider *selection*, let fitness of individual h be $f(h)$ and average fitness of whole n size population at time t be $\bar{f}(t)$

Does GA works?

- Can we mathematically characterize the evolution
- **Schema:** string of 0, 1 or $*$ like $0 * 1$ denoting set $\{001, 011\}$
- String 1011 is **representative** of 2^4 schema
- Let $m(s, t)$ be number of instances of schema s in generation t
- Consider *selection*, let fitness of individual h be $f(h)$ and average fitness of whole n size population at time t be $\bar{f}(t)$
- $h \in s \cup p_t$ means 1) h is representative of s and 2) it is present in the population at time t

Does GA works?

- Can we mathematically characterize the evolution
- **Schema:** string of 0, 1 or $*$ like $0 * 1$ denoting set $\{001, 011\}$
- String 1011 is **representative** of 2^4 schema
- Let $m(s, t)$ be number of instances of schema s in generation t
- Consider *selection*, let fitness of individual h be $f(h)$ and average fitness of whole n size population at time t be $\bar{f}(t)$
- $h \in s \cup p_t$ means 1) h is representative of s and 2) it is present in the population at time t
- Let $\hat{u}(s, t)$ be average fitness of instances of s at time t

Does GA works?

- Can we mathematically characterize the evolution
- **Schema:** string of 0, 1 or $*$ like $0 * 1$ denoting set $\{001, 011\}$
- String 1011 is **representative** of 2^4 schema
- Let $m(s, t)$ be number of instances of schema s in generation t
- Consider *selection*, let fitness of individual h be $f(h)$ and average fitness of whole n size population at time t be $\bar{f}(t)$
- $h \in s \cup p_t$ means 1) h is representative of s and 2) it is present in the population at time t
- Let $\hat{u}(s, t)$ be average fitness of instances of s at time t

$$\hat{u}(s, t) = \frac{\sum_{h \in s \cup p_t} f(h)}{m(s, t)}$$

Does GA works?

- Can we mathematically characterize the evolution
- **Schema:** string of 0, 1 or $*$ like $0 * 1$ denoting set $\{001, 011\}$
- String 1011 is **representative** of 2^4 schema
- Let $m(s, t)$ be number of instances of schema s in generation t
- Consider *selection*, let fitness of individual h be $f(h)$ and average fitness of whole n size population at time t be $\bar{f}(t)$
- $h \in s \cup p_t$ means 1) h is representative of s and 2) it is present in the population at time t
- Let $\hat{u}(s, t)$ be average fitness of instances of s at time t

$$\hat{u}(s, t) = \frac{\sum_{h \in s \cup p_t} f(h)}{m(s, t)}$$

- We know $Pr(h) = f(h)/(\sum f(h)) = f(h)/(n\bar{f}(t))$

Does GA works? (contd..)

- Probability that we will select a representative of schema s is

$$Pr(h \in s) = \sum_{h \in s \cup p_t} Pr(h)$$

Does GA works? (contd..)

- Probability that we will select a representative of schema s is

$$Pr(h \in s) = \sum_{h \in s \cup p_t} Pr(h) = \sum_{h \in s \cup p_t} f(h)/(n\bar{f}(t))$$

Does GA works? (contd..)

- Probability that we will select a representative of schema s is

$$
\begin{aligned}
Pr(h \in s) &= \sum_{h \in s \cup p_t} Pr(h) = \sum_{h \in s \cup p_t} f(h)/(n\bar{f}(t)) \\
&= \frac{\hat{u}(s,t)}{n\bar{f}(t)} m(s,t)
\end{aligned}
$$

Does GA works? (contd..)

- Probability that we will select a representative of schema s is

$$
\begin{aligned}
Pr(h \in s) &= \sum_{h \in s \cup p_t} Pr(h) = \sum_{h \in s \cup p_t} f(h)/(n\bar{f}(t)) \\
&= \frac{\hat{u}(s,t)}{n\bar{f}(t)} m(s,t)
\end{aligned}
$$

- Expected number of instances of s resulting from the n independent selection steps that create the entire new generation is just n times this probability.

Does GA works? (contd..)

- Probability that we will select a representative of schema s is

$$
\begin{aligned}
Pr(h \in s) &= \sum_{h \in s \cup p_t} Pr(h) = \sum_{h \in s \cup p_t} f(h)/(n\bar{f}(t)) \\
&= \frac{\hat{u}(s,t)}{n\bar{f}(t)} m(s,t)
\end{aligned}
$$

- Expected number of instances of s resulting from the n independent selection steps that create the entire new generation is just n times this probability. Therefore,

$$
E[m(s,t+1)] = \frac{\hat{u}(s,t)}{\bar{f}(t)} m(s,t)
$$

Does GA works? (contd..)

$$E[m(s, t+1)] = \frac{\hat{u}(s, t)}{\bar{f}(t)} m(s, t)$$

Expected number of representative instances of a schema s in the generation at time $t + 1$ is

Does GA works? (contd..)

$$E[m(s,t+1)] \quad = \quad \frac{\hat{u}(s,t)}{\bar{f}(t)} m(s,t)$$

Expected number of representative instances of a schema s in the generation at time $t+1$ is

1. Proportional to the average fitness $\hat{u}(s,t)$ of instances of this schema at time t

Does GA works? (contd..)

$$E[m(s, t+1)] = \frac{\hat{u}(s,t)}{\bar{f}(t)} m(s,t)$$

Expected number of representative instances of a schema s in the generation at time $t+1$ is

1. Proportional to the average fitness $\hat{u}(s,t)$ of instances of this schema at time t , and
2. Inversely proportional to the average fitness $\bar{f}(t)$ of all members of the population at time t

Does GA works? (contd..)

$$E[m(s, t+1)] \;=\; \frac{\hat{u}(s,t)}{\bar{f}(t)} m(s,t)$$

Expected number of representative instances of a schema s in the generation at time $t+1$ is

1. Proportional to the average fitness $\hat{u}(s,t)$ of instances of this schema at time t , and
2. Inversely proportional to the average fitness $\bar{f}(t)$ of all members of the population at time t

Thus, we can expect schema with <u>above average fitness</u> to be represented with increasing frequency on successive generations

Does GA works? (contd..)

Also consider negative effects of single point crossover and mutation

Does GA works? (contd..)

Also consider negative effects of single point crossover and mutation

- Let p_c represents the probability of **crossover** on an individual. $d(s)$ be the distance between left most and right most defined bit of s and l be the length of individual bit string

Does GA works? (contd..)

Also consider negative effects of single point crossover and mutation

- Let p_c represents the probability of **crossover** on an individual. $d(s)$ be the distance between left most and right most defined bit of s and l be the length of individual bit string
- Let p_m represents the probability of **mutation** on an individual and $o(s)$ be number of defined bits in s

Does GA works? (contd..)

Also consider negative effects of single point crossover and mutation

- Let p_c represents the probability of **crossover** on an individual. $d(s)$ be the distance between left most and right most defined bit of s and l be the length of individual bit string
- Let p_m represents the probability of **mutation** on an individual and $o(s)$ be number of defined bits in s

Full schema theorem thus provides a lower bound on the expected frequency of schema s, as follows

$$E[m(s,t+1)] \geq \frac{\hat{u}(s,t)}{\bar{f}(t)} m(s,t)\left(1 - p_c \frac{d(s)}{l-1}\right)(1-p_m)^{o(s)}$$

Does GA works? (contd..)

Also consider negative effects of single point crossover and mutation

- Let p_c represents the probability of **crossover** on an individual. $d(s)$ be the distance between left most and right most defined bit of s and l be the length of individual bit string
- Let p_m represents the probability of **mutation** on an individual and $o(s)$ be number of defined bits in s

Full schema theorem thus provides a lower bound on the expected frequency of schema s, as follows

$$E[m(s,t+1)] \geq \frac{\hat{u}(s,t)}{\bar{f}(t)} m(s,t)\left(1 - p_c \frac{d(s)}{l-1}\right)(1-p_m)^{o(s)}$$

Similar expression. More fit schemas will tend to grow in influence.

Thank You!

Thank you very much for your attention!

Queries ?

(Reference[4])

[4] 1) Book - *AIMA*, ch-03/04, Russell and Norvig. 2) Book - Machine Learning, ch-09, *Tom Mitchell*