Evolutionary Algorithms

Evolutionary Computation/Algorithms Genetic Algorithms

- Simulate “natural” evolution of structures via selection and reproduction, based on performance (fitness)
- Type of heuristic search to optimize any set of parameters- Discovery, not inductive learning in isolation
- Create "genomes" which represent a current potential solution with a fitness (quality) score
- Values in genome represent parameters to optimize
  - MLP weights, TSP path, knapsack, potential clusterings, etc.
- Discover new genomes (solutions) using genetic operators
  - Recombination (crossover) and mutation are most common

Evolutionary Computation/Algorithms
Genetic Algorithms

- Populate our search space with initial random solutions
- Use genetic operators to search the space
- Do local search near the possible solutions with mutation
- Do exploratory search with recombination (crossover)

1 1 0 2 | 3 1 0 2 2 1 (Fitness = 60)
2 2 0 1 | 1 3 1 1 0 0 (Fitness = 72)

1 1 0 2 1 3 1 1 0 0 (Fitness = 55)
2 2 0 1 3 1 0 2 2 1 (Fitness = 88)

Evolutionary Algorithms

- Start with initialized population $P(t)$ - random, domain-knowledge, etc.
- Typically have fixed population size (type of beam search), large enough to maintain diversity
- Selection
  - Parent_Selection $P(t)$ - Promising Parents more likely to be chosen based on fitness to create new children using genetic operators
  - Survive $P(t)$ - Pruning of less promising candidates, Evaluate $P(t)$ - Calculate fitness of population members. Could be simple metrics to complex simulations.
- Survival of the fittest - Find and keep best while also maintaining diversity

Evolutionary Algorithm

Procedure EA

$t = 0$;

Initialize Population $P(t)$;

Evaluate $P(t)$;

Until Done{ /*Sufficiently "good" individuals discovered or many iterations passed with no improvement, etc.*/

$t = t + 1$;

Parent_Selection $P(t)$;

Recombine $P(t)$;

Mutate $P(t)$;

Evaluate $P(t)$;

Survive $P(t)$;}

EA Example

- Goal: Discover a new automotive engine to maximize performance, reliability, and mileage while minimizing emissions
- Features: CID (Cubic inch displacement), fuel system, # of valves, # of cylinders, presence of turbo-charging
- Assume - Test unit which tests possible engines and returns integer measure of goodness
- Start with population of random engines

Individual	CID	Fuel System	Turbo	Valves	Cylinders
1	350	4 Barrels	Yes	16	8
2	250	Mech. Inject.	No	12	6
3	150	Elect. Inject.	Yes	12	4
4	200	2 Barrels	No	8	4

We now evaluate each individual with the engine simulator. Each individual receives a fitness score (the higher the better):

Individual	CID	Fuel System	Turbo	Valves	Cylinders	Score
1	350	4 Barrels	Yes	16	8	50
2	250	Mech. Inject.	No	12	6	100
3	150	Elect. Inject.	Yes	12	4	300
4	200	2 Barrels	No	8	4	150

Parent selection decides who has children and how many to have. For example, we could decide that individual 3 deserves two children, because it is so much better than the other individuals. Children are created through recombination and mutation. As mentioned above, recombination exchanges information between individuals, while mutation perturbs individuals, thereby increasing diversity. For example, recombination of individuals 3 and 4 could produce the two children:

Individual	CID	Fuel System	Turbo	Valves	Cylinders
3'	200	Elect. Inject.	Yes	8	4
4'	150	2 Barrels	No	12	4

Note that the children are composed of elements of the two parents. Further note that the number of cylinders must be four, because individuals 3 and 4 both had four cylinders. Mutation might further perturb these children, yielding:

Individual	CID	Fuel System	Turbo	Valves	Cylinders
3'	250	Elect. Inject.	Yes	8	4
4'	150	2 Barrels	No	12	6

We now evaluate the children, giving perhaps:

Individual	CID	Fuel System	Turbo	Valves	Cylinders	Score
3’	250	Elect. Inject.	Yes	8	4	250
4’	150	2 Barrels	No	12	6	350

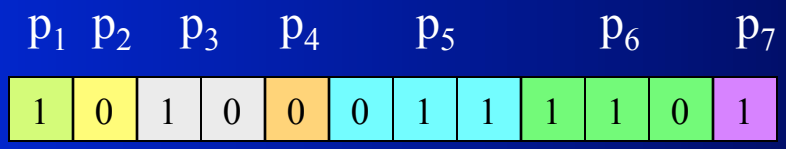
Finally we decide who will survive. In our constant population size example, which is typical of most EAs, we need to select four individuals to survive. How this is accomplished varies considerably in different EAs. If, for example, only the best individuals survive, our population would become:

Individual	CID	Fuel System	Turbo	Valves	Cylinders	Score
3	150	Elect. Inject.	Yes	12	4	300
4	200	2 Barrels	No	8	4	150
3’	250	Elect. Inject.	Yes	8	4	250
4’	150	2 Barrels	No	12	6	350

This cycle of evaluation, selection, recombination, mutation, and survival continues until some termination criterion is met.

Data Representation (Genome)

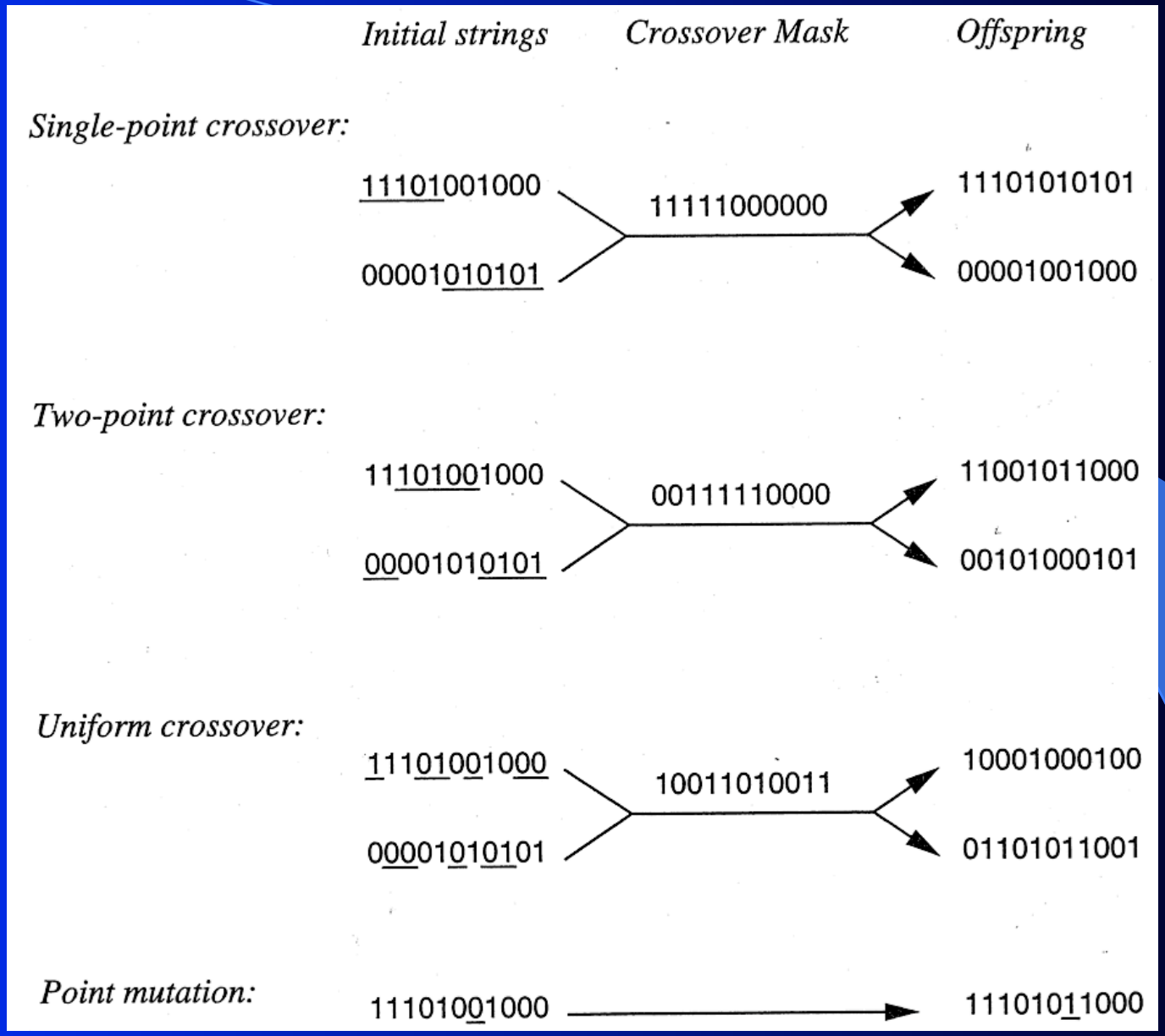
- Individuals are represented so that they can be manipulated by genetic operators
- Simplest representation is a bit string, where each bit or group of bits could represent a feature/parameter
- Assume the following represents a set of parameters



- Could do crossovers anywhere or just at parameter breaks
- Can use more complex representations including real numbers, symbolic representations (e.g. programs for genetic programming), etc.

Genetic Operators

- Crossover variations - multi-point, uniform, averaging, etc.
- Mutation - Random changes in features, adaptive, different for each feature, etc.
- Others - many schemes mimicking natural genetics: dominance, selective mating, inversion, reordering, speciation, knowledge-based, etc.


Initial strings
Crossover Mask
Offspring
Single-point crossover:
11101001000
00001010101
11111000000
11101010101
00001001000
Two-point crossover:
11101001000
00001010101
00111110000
11001011000
00101000101
Uniform crossover:
11101001000
00001010101
10011010011
10001000100
01101011001
Point mutation:
11101001000
11101011000

Fitness Function Evaluation

- Each individual in the population should have a fitness based on the fitness function
- Fitness function will depend on the application
  - Learning system will usually use accuracy on a validation set for fitness (note that no training set needed, just validation and test)
  - Solution finding (path, plan, etc.) - Length or cost of solution
  - Program - Does it work and how efficient is it
- Cost of evaluating function can be an issue. When expensive can approximate or use rankings, etc. which could be easier.
- Stopping Criteria - A common one is when best candidates in population are no longer improving over time

Parent Selection

- In general want the fittest parents to be involved in creating the next generation
- However, also need to maintain diversity and avoid *crowding* so that the entire space gets explored (local minima vs global minima)
- Most common approach is *Fitness Proportionate Selection* (aka roulette wheel selection)

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

- Everyone has a chance but the fittest are more likely

Parent Selection

- There are other methods which lead to more diversity
- Rank selection
  - Rank order all candidates
  - Do random selection weighted towards highest rank
  - Keeps actual fitness *value* from dominating
- Tournament selection
  - Randomly select two candidates
  - The one with highest fitness is chosen with probability p, else the lesser is chosen
  - p is a user defined parameter, $.5 < p < 1$
  - Even more diversity
- Fitness scaling - Scale down fitness values during early generations. Scale back up with time. Equivalently could scale selection probability function over time.

Tournament Selection with $p = 1$

Biagio D'Antonio, b. 1446, Florence, Italy - *Saint Michael Weighing Souls* - 1476

Survival - New Generation

- Population size - Larger gives more diversity but with diminishing gain, small sizes of ~100 are common
- How many new offspring will be created at each generation (what % of current generation will not survive)
  - Keep selecting parents without replacement until quota filled
- New candidates replace old candidates to maintain the population constant
- Many variations
  - Randomly keep best candidates weighted by fitness
  - No old candidates kept
  - Always keep a fixed percentage of old vs new candidates
  - Could keep highest candidate seen so far in separate memory since it may be deleted during normal evolution

GA($Fitness$, $Fitness\_threshold$, p, r, m)

Fitness: A function that assigns an evaluation score, given a hypothesis.

$Fitness\_threshold$: A threshold specifying the termination criterion.

p: The number of hypotheses to be included in the population.

r: The fraction of the population to be replaced by Crossover at each step.

m: The mutation rate.

- *Initialize population:* $P \leftarrow$ Generate p hypotheses at random
- *Evaluate:* For each h in P, compute $Fitness(h)$
- While $[\max_h Fitness(h)] < Fitness\_threshold$ do

  Create a new generation, P_S:

  1. *Select:* Probabilistically select $(1-r)p$ members of P to add to P_S. The probability $\Pr(h_i)$ of selecting hypothesis h_i from P is given by

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

  2. *Crossover:* Probabilistically select $\frac{r \cdot p}{2}$ pairs of hypotheses from P, according to $\Pr(h_i)$ given above. For each pair, $\langle h_1, h_2 \rangle$, produce two offspring by applying the Crossover operator. Add all offspring to P_s.
  3. *Mutate:* Choose m percent of the members of P_s with uniform probability. For each, invert one randomly selected bit in its representation.
  4. *Update:* $P \leftarrow P_s$.
  5. *Evaluate:* for each h in P, compute $Fitness(h)$

- Return the hypothesis from P that has the highest fitness.

Evolutionary Algorithms

- There exist mathematical proofs that evolutionary techniques are efficient search strategies
- There are a number of different Evolutionary algorithm approaches
  - Genetic Algorithms
  - Evolutionary Programming
  - Evolution Strategies
  - Genetic Programming
- Strategies differ in representations, selection, operators, evaluation, survival, etc.
- Some independently discovered, initially function optimization (EP, ES)
- Strategies continue to “evolve”

Genetic Algorithms

- Representation based typically on a list of discrete tokens, often bits (Genome) - can be extended to graphs, lists, real-valued vectors, etc.
- Select *m* parents probabilistically based on fitness
- Create *m* (or *2m*) new children using genetic operators (emphasis on crossover) and assign them a fitness - single-point, multi-point, and uniform crossover
- Replace *m* weakest candidates in the population with the new children (or can always delete parents)

Evolutionary Programming

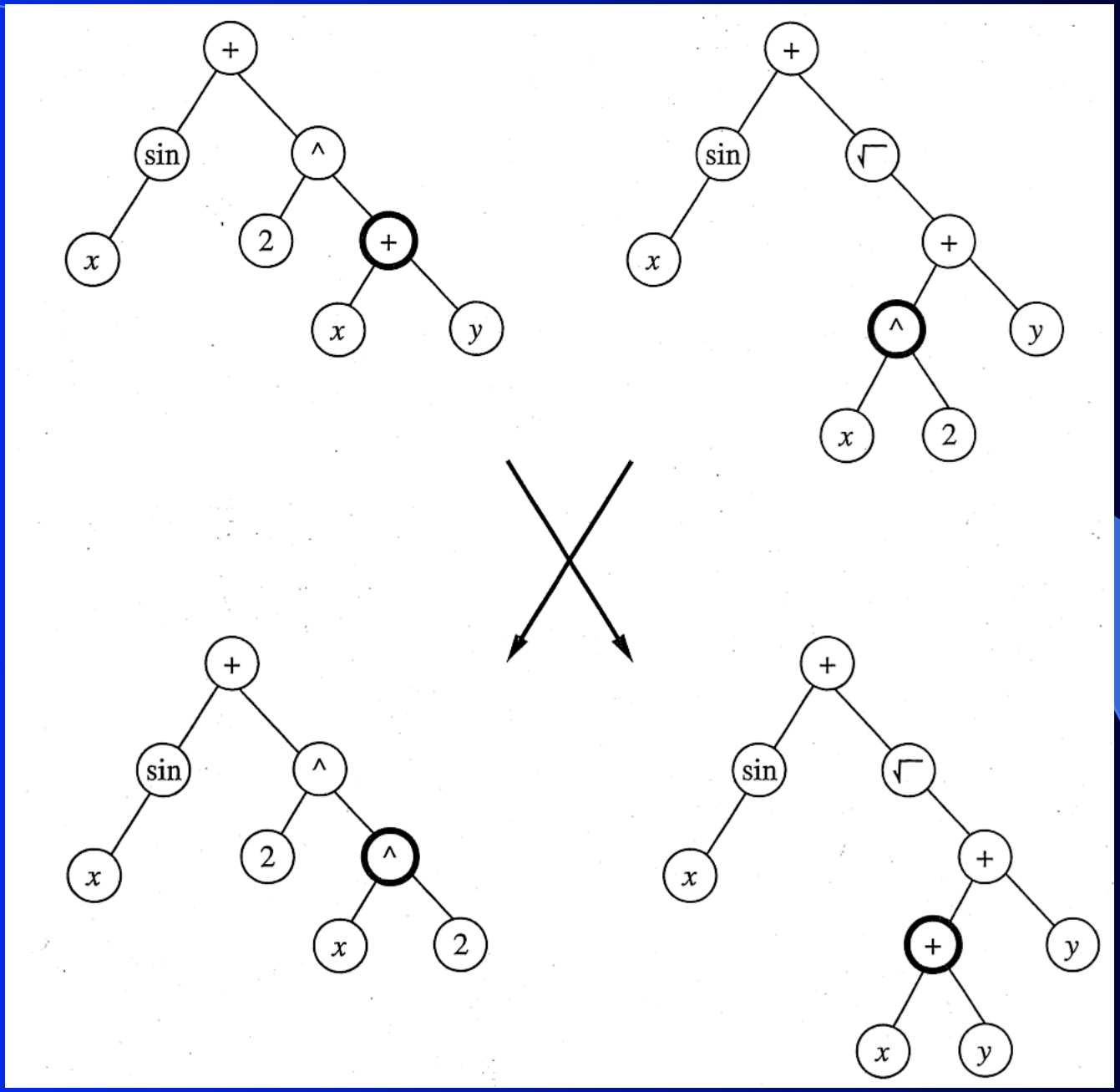
- Representation that best fits problem domain
- All *n* parents mutated (no crossover) to create *n* new children - total of *2n* candidates
- Only *n* most fit candidates are kept
- Mutation schemes fit representation, varies for each variable, amount of mutation (typically higher probability for smaller mutation), and can be adaptive (i.e. can decrease amount of mutation for candidates with higher fitness and based on time - form of simulated annealing)

Evolution Strategies

- Similar to Evolutionary Programming - initially used for optimization of complex real systems - fluid dynamics, etc. - usually real vectors
- Uses both crossover and mutation. Crossover first. Also averaging crossover (real values), and multi-parent.
- Randomly selects set of parents and modifies them to create > *n* children
- Two survival schemes
  - Keep best *n* of combined parents and children
  - Keep best *n* of only children

Genetic Programming

- Evolves more complex structures - programs, Lisp code, neural networks
- Start with random programs of functions and terminals (data structures)
- Execute programs and give each a fitness measure
- Use crossover to create new programs, no mutation
- Keep best programs
- For example, place lisp code in a tree structure, functions at internal nodes, terminals at leaves, and do crossover at sub-trees - always legal in Lisp


+
sin
^
x
2
+
x
y
+
sin
√
x
+
^
y
x
2
+
sin
^
x
2
^
x
2
+
sin
√
x
+
+
y
x
y

Genetic Algorithm Example

- Use a Genetic Algorithm to learn the weights of an MLP. Used to be a lab.

Genetic Algorithm Example

- Use a Genetic Algorithm to learn the weights of an MLP. Used to be a lab.
- You could represent each weight with *m* (e.g. 10) bits (Binary or Gray encoding), remember the bias weights
- Could also represent Neural Network Weights as real values - In this case use Gaussian style mutation
- Walk through an example comparing both representations
  - Assume wanted to train MLP to solve Iris data set
  - Assume fixed number of hidden nodes, though GAs can be used to discover that also

Evolutionary Computation Comments

- Much current work and extensions
- Numerous application attempts. Can plug into many algorithms requiring search. Has built-in heuristic. Could augment with domain heuristics.
- If no better way, can always try evolutionary algorithms, with pretty good results ("Lazy man's solution" to any problem)
- Many different options and combinations of approaches, parameters, etc.
- Swarm Intelligence – Particle Swarm Optimization, Ant colonies, Artificial bees, Robot flocking, etc.
- More work needed regarding adaptivity of
  - population size
  - selection mechanisms
  - operators
  - representation

Classifier Systems

- Reinforcement Learning - sparse payoff
- Contains rules which can be executed and given a fitness (Credit Assignment) - Booker uses bucket brigade scheme
- GA used to discover improved rules
- Classifier made up of input side (conjunction of features allowing don't cares), and an output message (includes internal state message and output information)
- Simple representation aids in more flexible adaptation schemes

Bucket Brigade Credit Assignment

- Each classifier has an associated strength. When matched the strength is used to competitively bid to be able to put message on list during next time step. Highest bidders put messages.
- Keeps a message list with values from both input and previously matched rules - matched rules set outputs and put messages on list - allows internal chaining of rules - all messages changed each time step.
- Output message conflict resolved through competition (i.e. strengths of classifiers proposing a particular output are summed, highest used)

Bucket Brigade (Continued)

- Each classifier bids for use at time *t*. Bid used as a probability (non-linear) of being a winner - assures lower bids get some chances

$$B(C,t) = bR(C)s(C,t)$$

where *b* is a constant << 1 (to insure prob<1), *R(C)* is specificity (# of asserted features), *s(C,t)* is strength

- Economic analogy - Previously winning rules (t-1) "suppliers" (made you matchable), following winning rules (t+1) "consumers" (you made them matchable - actually might have made)

Bucket Brigade (Continued)

- $s(C,t+1) = s(C,t) - B(C,t)$ - Loss of Strength for each consumer (price paid to be used, prediction of how good it will pay off)
- $\{C'\}$ = {suppliers} - Each of the suppliers shares an equal portion of strength increase proportional to the amount of the Bid of each of the consumers in the next time step

$$s(C',t+1) = s(C',t) + B(C_i,t)/|C'|$$

- You pay suppliers amount of bid, receive bid amounts from consumers. If consumers profitable (higher bids than you bid) your strength increases. Final rules in chain receive actual payoffs and these eventually propagate iteratively. Consistently winning rules give good payoffs which increases strength of rule chain, while low payoffs do opposite.