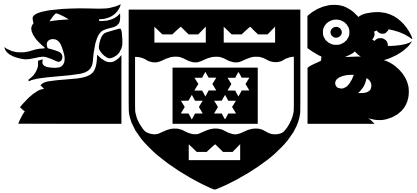
VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui



School of Engineering and Computer Science

COMP 422

Building Blocks and Crossover

Mengjie Zhang

mengjie@ecs.vuw.ac.nz

Outline

- Crossover and building blocks
- Theoretical basis for building blocks in GP
- Preservation vs disruption of building blocks
- Empirical evidence of crossover effects
- Improving crossover – from biology
- Brood recombination
- Other approaches:
  - Intelligent crossover
  - Context sensitive crossover
  - Explicit multiple gene systems
  - Explicit defined introns

Crossover and Building Blocks

- In must GP systems, crossover is the predominant search operator
- GP crossover has been used as the basis for the claim that GP search is more effective than systems based on random transformations/mutations, like simulated annealing.
- Koza argues that a GP population contains **building blocks**.
- A building block can be any subtree or tree occurs in a fraction of the population.
- Building block hypothesis (in GA): [Goldberg]
  - Good building blocks improve the fitness of individuals.
  - Individuals with good building blocks are most likely to be selected for ...
  - Good building blocks are likely to spread.

Crossover – The Controversy

- According to this hypothesis, GP works faster than systems just based on mutations, because good building blocks get combined into even large and better blocks to form better individuals.
- Is this true in GP? Two central questions:
  - Does the GP crossover operator outperform mutation-based systems by locating and combining good building blocks, or is it just a form of macro mutation?
  - What sorts of measures can be taken to the crossover operator to improve the performance?

Holland's Schema Theorem on GA

- Schema theorem:
  - For bitstrings with $\{0, 1\}$,
  - A schema is a template string $\{0, 1, *\}$,
  - Bit strings can be instances of many different similarity template (or, *schemata*).
  - Example 1: the bit string 001 is an instance of each schema in the set $\{***, **1, *01, 0*1, 0**, 00*, *0*, 001\}$
  - Example 2: the schema $\{0**1\}$ represents the sets of bit strings $\{0001, 0011, 0101, 0111\}$.

Schema Theorem on GA

- Order and Length of a Schema
  - Order: the number of defined bits in the schema.
  - Length: the distance between the first and the last defined bits of the schema.
  - Example: For a schema $1**010***10**$, the order is 6 and the length is 10.
- When a GA explicitly searches $\{0, 1\}$, it implicitly searches the much larger space of the schemata $\{0, 1, *\}$. In this way, GA can be said to be inherently parallel and enable every large solution spaces to be searched.
- Schema theorem: The expected number of copies of *very fit*, *short*, *low order* schemata increases exponentially to form even more highly fit, higher order schemata giving the powerful learning ability through the recombination.

Theoretical Basis for Building Blocks

- Attempts have been made to extend the schema theorem from GA to GP.
- But GP case is much more complex due to the varying length representation and more flexible combination.
- A crucial issue: To what extent crossover tends to preserve or to disrupt good schemata?
- Schema theorem analyses:
  - Koza's schema theorem analysis
  - O'Reily's schema theorem analysis
  - Whigham's schema theorem analysis
  - Other schema theorems

Koza's Schema Theorem Analysis

- In 1992, Koza first addressed this issue in GP.
- A schemata is a set of subtrees containing one or more subtrees from a special schema defining set.
- If the schema defining set is $H_1 = \{(- 2 y), (+ 2 3)\}$, all the subtrees that contain (- 2 y) or (+ 2 3) are instances of H_1.
- He did not give the order or the length definition
- His conclusion: GP crossover tends to preserve, rather than disrupt, good schemata. Smaller but good schemata are combined into bigger schemata and, ultimately, good overall solutions.

O'Relly's Schema Theorem Analysis

- O'Relly [1994,95] formalized and extended Koza's idea by defining a schema as a multiset of subtrees under proportional selection and tree based crossover.
- A *don't care symbol (#)* was introduced, such as H_1 = {(- # y), (+ 2 #)}.
- The *order* is the number of nodes which are not # symbols and the *length* is the number of links in the tree fragments plus the number of links connecting them. ?
- The sum of the links is variable and the maximum probability of disruption varies with the size of the tree.
- No conclusion was drawn on whether crossover preserves or disrupts good schemata.

Other Schema Theorems

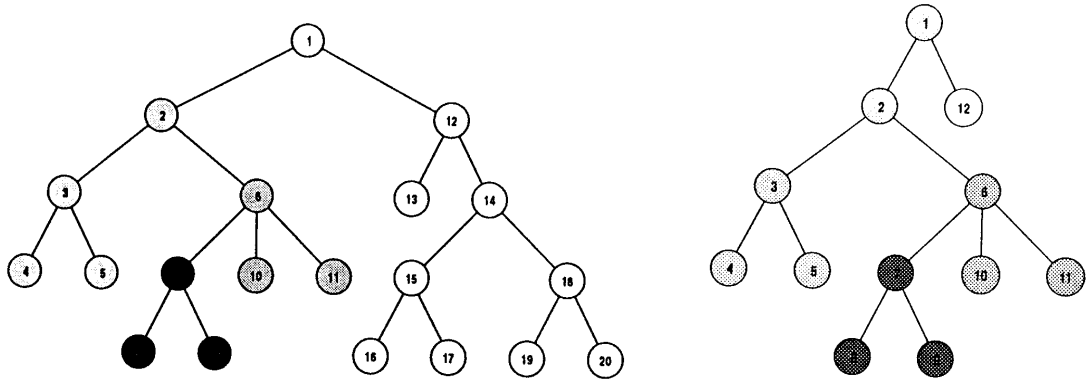
- Whigham's schema analysis:
  - Whigham formulated a definition of schemata in a grammar-based GP system [1995].
  - Similarly to O'Relly, he also stated that the probability of disruption depends on the size of the program.
- Poli and Langdon's schemata theorem:
  - Poli and Langdon [1997] proposed a new schema theorem that converges to the GA schema theorem.
  - Results: there might be two different phases in a GP run – the first phase completely depending on fitness, and the second depending on fitness and structure of the individual (e.g. schema defining length).

Theoretical Basis for Building Blocks

- Rosca's schema theorem:
  - Rosca [1997] concentrated on the structural aspect and proposed a *rooted-tree schemata*.
  - A rooted-tree schema is a subset of the program trees that matches an identical tree fragment including the root.
- Conclusions:
  - None of these GP schema theorems predicts with any certainty that good schemata will preserve and propagate during a GP run.
  - The question of whether GP crossover is just a macromutation or more than that was not answered.
  - There might be two different phases in a GP run – the first phase completely depending on fitness, and the second depending on fitness and structure of the individual.

Preservation vs Disruption of Building Blocks

- A Gedanken experiment



(a) (b)

- Small building blocks:
  - Assume the dark nodes (7-9) constitutes a good block.
  - In this program tree, there are 19 crossover points.
  - If crossover is distributed randomly among these nodes, then the probability of the good building block being disrupted by crossover is 2/19 (10.5%)

Preservation vs Disruption of Building Blocks

- Large building blocks
  - Assume crossover finds a new, good building block containing the original one: node 6-11.
  - The probability of this new block being disrupted by crossover is 5/19 (26.3%).
  - If another new larger block is found, say, nodes 1-11, then the disruptive probability will be 10/19, or 52.6%.
  - Consider a case: assume the larger block is almost a perfect program, and all needed to be done is to get rid of the code represented by the white node. If Figure (b) is obtained from Figure (a), then there is only one node to get rid of before the solution is perfect. However, the disruptive probability becomes 10/11, or 90.91%.

Preservation vs Disruption of Building Blocks

- Thus crossover is a disruptive force as well as a constructive force – putting building blocks together and then tearing them apart.
- Reproduction and crossover
  - Reproduction operator takes the fittest individuals and duplicates them.
  - Argument: good building blocks in those duplicated individuals will have more chances to find crossovers that are not disruptive.
  - Sometimes YES, other times NO.
  - Knowing the reproduction parameter and how to adjust it during a run to improve the performance – a new direction.
- Conclusion: Is crossover just a macromutation or more? – Still inconclusive from the Gedankan experiment!

Empirical Evidence of Crossover Effects

- At least two parents and one or more children for crossover.
- How to measure the effects of crossover?
  - The average fitness of all parents is compared with the average fitness of all children.
  - The fitness of children and parents is compared on individuals. Which one, the best one or the worst one or one by one?

Empirical Evidence of Crossover Effects

- Results of measuring the effects of crossover:
  - Crossover has an overwhelmingly *negative* effect on the fitness of the offspring.
  - In tree based and linear genomes, the fitness of the children is less than half of the fitness of parents in about 75% of all the crossover events. [Nordin 1996]
  - In graph-based representations, less than 10% of the crossover events result in an improvement in fitness of offspring relative to their parents. [Teller 1996]
- Conclusion: Crossover routinely reduces the fitness of offspring substantially relative to their parents in almost every GP system. This stands in stark contrast to biological crossover.

Empirical Evidence of Crossover Effects

- Crossover vs Hill climbing or annealing
- Headless chicken crossover:
  - Only one parent is selected from the preexisting learned solutions.
  - An entirely new individual is created randomly.
  - Do crossover for the above two individuals
  - The offspring is kept if it is better than or equal to the parent; otherwise, discarded.
- Headless chicken crossover is a form of hill climbing.
- Lang [1995] argued that headless chicken crossover was much better than GP crossover.

Empirical Evidence of Crossover Effects

- Is hill climbing better than GP? Lang's results showed so, but the problem was the boolean 3-multiplexer where there were no strict local minima [Jules, 1995]
- Other studies [O'Relly, 1994,95] [Angeline, 1997] also suggest that macromutation techniques, such as mutation-simulated annealing and crossover-hill climbing, may perform as well as and sometimes slightly better than traditional GP crossover.

Empirical Evidence of Crossover Effects

Conclusions from the empirical measurements:

- These empirical evidence did not show GP crossover is more effective or better than mutation based techniques.
- There is no serious support for Lang's claim – Hill climbing outperforms GP.
- Based on the evidence existing today, **the standard GP crossover acts primarily as a macromutation operator**. [Banzhaf]
- The standard GP crossover performs as well as, or almost as well as techniques based on long established and successful algorithms, such as simulated annealing and hill climbing.
- Crossover performs quite well even if it gives highly disruptive effect on offspring.
- There may be room for substantial improvement of crossover operator.

Improving Crossover – From Biology

- Crossover can be seen as the result of the evolution of evolvability. [Altenberg, 1994]
- Principal constraints on biological crossover:
  - Biological crossover takes place only between members of the same species.
  - Biological crossover occurs with remarkable attention to preservation of "semantics". For example, the hair color gene does not get swapped with the tallness gene.
  - Biological crossover is homologous. Two DNA strands can line up identical so that crossover is accurate almost down to the molecular level.

Improving Crossover – From Biology

- GP crossover:
  - GP crossover is unconstrained and uncontrolled. In GP, any subtree may be crossed over with other subtrees. There is no requirement that the two subtrees fulfill similar functions.
  - Crossover points are selected randomly in both parents. There is no requirement that a subtree, after being swapped, is in a context in the new program relative to the context to the old program.
  - There are predefined building blocks (genes). If GP got a good subtree building block, it would be very likely to be disrupted rather than preserved and spread.
- These differences provide heuristic guidance to improve GP crossover.

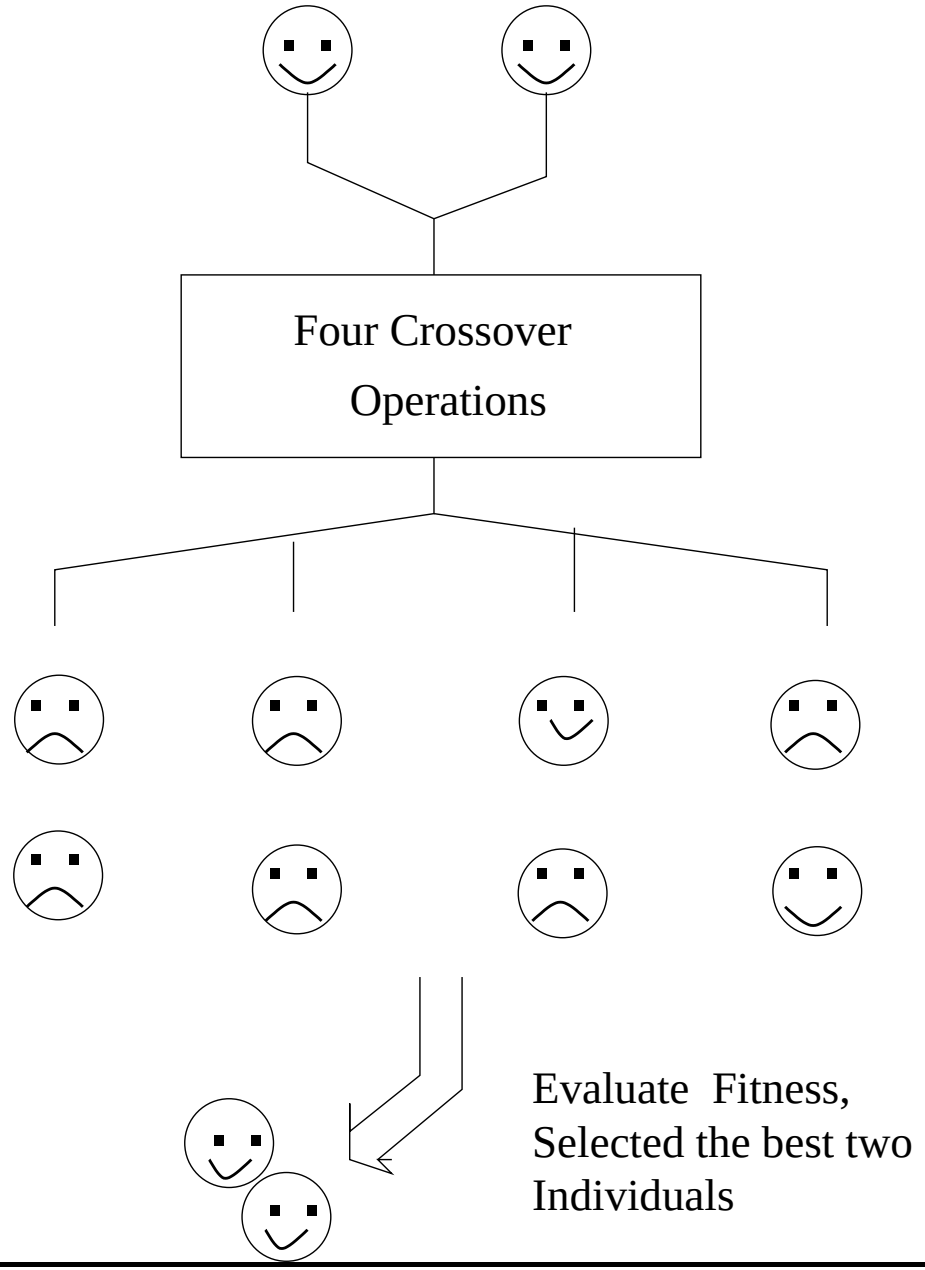
Improving Crossover

- Most effects to improve crossover have focused on preservation of good building blocks, rather than preservation of homology.
- Main methods:
  - Brood recombination [Tackett, 1994]
  - “Intelligent” crossover [Teller, 1996]
  - Context-sensitive crossover [D’haeseler 1994]
  - Explicit multiple gene systems [Altenberg, 1995]
  - Explicit defined introns [Nordin, 1996]
  - ...

Brood Recombination

- Main idea: Reduce the destructive effect of crossover.
- Many animal species produce far more offspring than are expected to live. Although there are many different mechanisms, the excess offspring die. So should GP crossover.
- The approach:
  - create a "brood" each time crossover is performed.
  - Key parameter: brood size N
  - Pick two parents from the population
  - Perform random crossover on the parents N times, each time creating a pair of children.
  - Evaluate each child for fitness. Sort them by fitness. Select the best two, which are considered the "real" children of the parents. Discard other children.

Brood Recombination



Brood Recombination

- Big problem: evaluate more individuals and make GP slow down.
- Time-saving evaluation: Evaluation is performed on only a small portion of the training set.
- Is brood recombination effective?
  - Brood recombination performed significantly better than standard GP crossover.
  - There was only a small reduction in performance by using 30 out of 360 training cases to evaluate the brood.
  - It is possible to reduce the population size when this method is used.

Brood Recombination

- How does the brood recombination improve the performance?
  - by not disrupting building blocks?
  - or by adding a different form of search process to the GP algorithm – in machine learning terms, giving GP the ability to look ahead when adjusting the beam/population?
  - The results were consistent with either of them.
- An interesting suggestion: a dynamic form of brood recombination, where size of the brood grows as evolution proceeds, might yield better results.

"Intelligent" Crossover

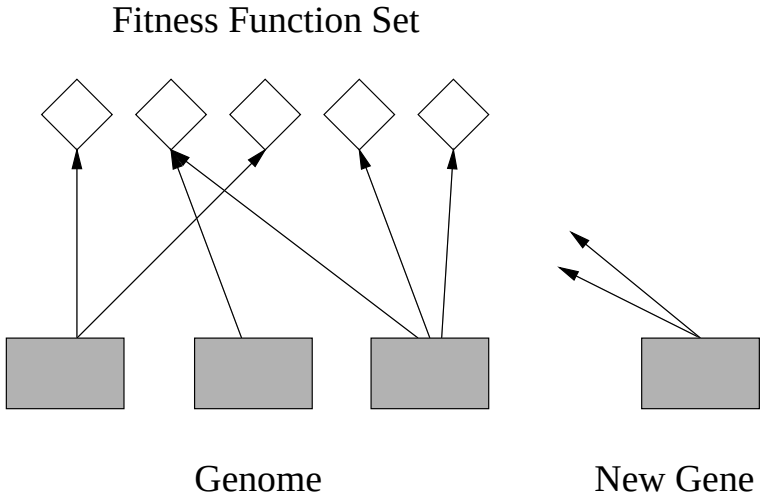
- Main idea: Add intelligence to the crossover operator by letting it **select good crossover points**.
- Heuristic guidelines:
  - Let crossover operator access to information about the execution path in an evolved program [Teller 1996]
  - Compare the performancefor subtrees to decide which subtrees are potential building blocks [Iba 1996]
- Interesting suggestions:
  - There are blocks of code that are best left together – perhaps these are building blocks.
  - These blocks of code can be identified by heuristics or a learning algorithm.
  - GP produces higher constructive crossover and better results when these blocks of code are probabilistically kept together.
- Brood recombination and smart/intelligent crossover attack the argument that crossover is a black box.

Context-Sensitive Crossover

- Most crossover does not preserve the content of the code – which is crucial to the meaning of computer code.
- D'haeseleer [1994] devised an operator – strong context preserving crossover (SCPC) that only permitted crossover between nodes that occupied **exactly the same position** in the two parents.
- An idea of homology, from biology point of view.
- Mixing regular crossover and SCPC could lead to improvement in results.

Explicit Multiple Gene Systems

- A constructional selection system improves crossover
- Fitness components are affected by all or some of the genes.
- Fitness of the individual is just the sum of the fitness components.



- During evolution, a gene is periodically added. If it improves the fitness of the individual, it is kept. Otherwise discarded.
- Having multiple fitness functions allows the genes to be more independent.
- This system is highly theoretical.

Explicit Defined Introns

- Nordin introduced explicitly defined introns (EDI) into GP. [1996]
- The value of EDI is defined as an integer stored between every two nodes in the GP individual.
- The probability that crossover occurs between any two nodes in the program is proportional to the value of EDI.
- Consideration: Allow the EDI vector to evolve during a GP run to identify and protect the building blocks in the program as an emergent phenomenon.
- The EDI value is designed as low within a good building block, and high outside th blocks.
- The results are positive in linear genomes but inconclusive in tree based genomes.

Explicit Defined Introns

- Angeline [1996] devised a better version of EDI
  - Use real values rather than integers, and
  - Constrain changes in the EDIs by a Gaussian distribution of permissible mutation.
- The result of this new EDI method to GP crossover was a substantial improvement in performance.
- Suggestion: GP crossover might be improved substantially by allowing GP to protect some groups of code from crossover preferentially over other groups of code.
- Again, there exist building blocks in GP individuals, which can create better fitness of the programs.
- EDI is a new direction!

Summary

- Improving crossover is a tradeoff
  - Crossover can be improved by reducing the probability of disrupting building blocks, and duplicating homologous crossover is probably worth trying.
  - Improvement of crossover generally carries additional digital overhead such as less effective use of memory and CPU time.
  - We probably should not expect the benefits of homologous crossover at any less cost than is paid in nature.
- Even if crossover is powerful, it is not a perfect operator in the current state of art.
- Recent researches suggest that it will become a much more powerful and robust operator over the next few years.
- How about the mutation operator?