Fall 2003 BMI 226 / CS 426 Notes QQQQQ-1

ASSEMBLY CODE GENETIC PROGRAMMING

MOTIVATION FOR IMPLEMENTATION OF GP IN ASSEMBLY CODE (NORDIN 1994)

• The division function must be coded as a multi-step function in order to protect against division by 0.

• In fact, <u>all</u> arithmetic functions (division, multiplication, subtraction, and addition) can potentially produce overflows and underflows. In processing large populations over many generations, these overflows and underflows are inevitable. Therefore, in practice, <u>all</u> arithmetic functions must be coded as a multi-step function.

• Exponential functions are another example of a function that is especially prone to producing overflows and underflows. It, too, must be protected.

• Thus, there is a potentially big opportunity for speeding up GP by evolving sequences of assembly code.

GP ASSEMBLY CODE — CONTINUED

• All general-purpose computers perform certain very basic operations, such as addition, subtraction, multiplication, division, and conditional operations.

• Nordin (1994) started by identifying a small set of primitive functions that are useful for large group of problems. One might choose the arithmetic functions (+, -, *, %), a conditional operation (IFLTE), and some logical functions (AND, OR, XOR, XNOR). One might also choose some shift and other instructions (e.g., SRL and SLL and SETHI from Sun 4).

GP ASSEMBLY CODE — CONTINUED

• A run starts at generation 0 by creating a population consisting of random sequences of assembly code instructions from the chosen small set of machine code instructions (strictly limiting references to certain registers).

• Perform crossover so as to preserve the integrity of subtrees.

• If ADFs are involved, perform crossover so as to preserve the integrity of the header and footer of the called function and the integrity of the function call.

COMPILER FOR GP

• In this approach, one writes a machinespecific compiler to dynamically compile the S-expression for an individual in the evolving population at the moment when is it going to be evaluated in the run of GP.

• The compiler can be extremely limited and only be able to handle common GP functions such as the arithmetic functions (+, -, *, %), some conditional operations (IFLTE), and perhaps some logical functions (AND, OR, XOR, XNOR).

• This approach is especially advantageous when there is an extremely large number of fitness cases (e.g., the evolving individual must be exposed to 50,000 amino acid residues in a set of protein sequences).