DGPC – DAVE'S GENETIC PROGRAMMING CODE IN C (BY DAVID ANDRE)

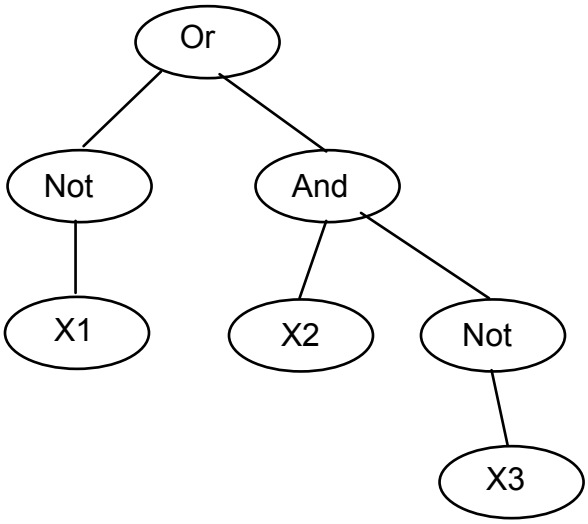
GOALS OF DGPC

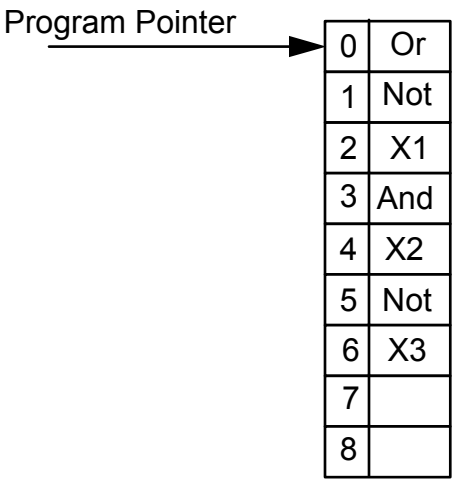
- **Use Minimal Memory - 2 (or 1) Bytes per point of program tree**
- **Constant memory use**
- **Allow easy use of automatically defined functions**
- **Written in C**
- **Have highly portable code. DGPC has been tested on Macs, PCs, Windows, Suns, and Alpha's (64-bit)**
- **Implement steady state evolution**
- **Allow non-standard individuals**
- **Support for priming the population**
- **Constant tables (easy change)**

REPRESENTATION

- **Polish notation representation**
- **Example**
  - **Function set F = {AND, OR, NOT} (with arity of all functions being fixed and known)**
  - **Terminal set T = {X1, X2, X3}**

REPRESENTATION





(Or (Not x1) (And x2 (Not x3)))

Lisp Representation (Tree based)

Array based Representation

FOUR TYPES OF NODES

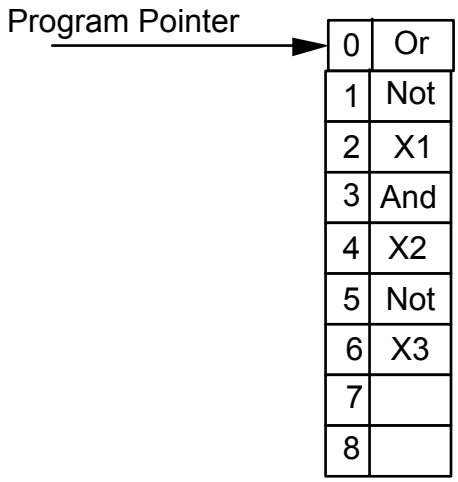
Terminals	**These nodes have no arguments and are 0-arity functions.**
Functions	**Nodes for *N* -arity functions have *N* arguments.**
Constants	**These nodes are constants that obtain a value once and only once during the existence of the node over all the different fitness cases.**
Macros	**These nodes are 'macros', and thus do not evaluate their arguments prior to being called. Instead a user-defined function is responsible for the evaluation of macros. Macros may be 0-arity (as in the case of ADF's with no arguments).**

RANDOM CONSTANTS

- **If, for example, there are 50 functions and terminals, there is room for 206 random constants**

Evaluation of an node


```
eval(branch * br)
{
int □□□temp;
int * args[MAXNUMARGS];
temp = g_index_ptr;
if (ISCONSTANT(br[g_index_ptr]))
  return(br[g_index_ptr++]->value);
if (ISMACRO(br[g_index_ptr]))
  return(  GETCodeName(br[g_index_ptr]));
else
  g_index_ptr++;
   for (i=0; i < ARITY(br[temp□□]); i++)
     args[i] = eval(br);
  return(  GETCodeName(br[temp])(args));
}
```



BASIC EVALUATION TECHNIQUE

EXAMPLE OF FUNCTIONS

A Node:

Index	Value
13	0

In function table:

#13	Arity	Constant	Macro	Print Name	Function Name
	3	0	1	IfTrue	"IfTrue"
#14	Arity	Constant	Macro	Print Name	Function Name
	0	1	0	"%d"	"RandomInt"

STRUCTURE OF AN INDIVIDUAL

Individual #17,891	
Standardized_fitness	12
Normalized_Fitness	.32
Hits	4
Penalties	2
Branches 0	→ tree
1	→ tree
ADFs 0	→ tree
1	→ tree

Function to 'Branch+ADF' Mapping

```
Branch_0
    NumFunctions =3;
    FunctionAssignment[0][0] = 1;
    FunctionAssignment[0][1] = 2;

    etc.


 ADF_0
    NumFunctions = 10;
    FunctionAssignment[2][0] = 9;
    FunctionAssignment[2][1] = 9;
    FunctionAssignment[2][2] = 3;



    etc....
```

STEPS FOR USING DGPC FOR A PROBLEM

(1) Decide on Architecture of individuals
    change NumBranches and NumADF's **(dgp_param.h)**

(2) Create Functions
    create function table **(problem.c)**
    create globals and problem-specific variables **(problem.c)**
    function headers **(problem.h)**
    function-branch mapping **(problem.c)**
    function code **(problem.c)**

(3) Evaluation of individuals
    eval_fitness_of_dude **(problem.c)**

(4) Termination conditions **(dgpc.c)**
    (if s_fitness >0, continue)

(5) Specification of parameters
    Run Parameters **(default.in)**
    General Parameters **(dgp_param.h)**

```
/*      dgpc -- Dave's Genetic Programming in C – v1.0
         Copyright © 1994, David Andre
                All rights reserved.
        dgp.h
Main headers and data structure definitions  --
Uses only 2 bytes per node, which is
2 bytes x 1000 nodes =
2K per Individual == 500 Individual per MEG == 5,000 individuals
per 10 MEG
*/

#if __STDC__ || defined(__cplusplus)
#define P_(s) s
#define ANSI_FUNC
#else
#define P_(s) ()
#ifdef ANSI_FUNC
#undef ANSI_FUNC
#endif
#endif
#include "dgp_param.h"

#define FULL   1
#define GROWTH   2
#define RAMPED 3
#define TOURNAMENT 4
#define FITNESSPROP 5

#include "dgp_param.h"

#define MaxNumFunctions 256
#define GTYPE int
#define GENERIC int


#define max(x,y) ((x)>(y)?(x):(y))
#define min(x,y) ((x)>(y)?(y):(x))

/* Macros for accessing data structures...
*/
#define function_arity(n)
gpop.func_table[(n).index].arity
#define function_is_macro(n)
     gpop.func_table[(n).index].macro
#define function_is_constant(n)
     gpop.func_table[(n).index].constant
#define function_printname(n)
     gpop.func_table[(n).index].PrintName
#define function_code(n)
gpop.func_table[(n).index].code
#define MAXSTRING 100

typedef struct {
                unsigned char index;
```

```
                char value;
                } snode;  /* short node type*/

typedef struct fte {
  int           arity;
  int           macro;
  int           constant;
  char          *PrintName;
  GTYPE (*code) P_((  ));
} function_table_entry;

typedef struct branch {
                short branchnum;
                short NumNodes;
                snode *tree;
                }branch;

typedef struct {
                struct branch   branches[max(1,NumBranches)];
                struct branch   adfs[max(1,NumADFs)];
                float s_fitness;
                float n_fitness;
                float a_fitness;
                short hits;
                short penalties;
                } individual;  /* Structure for an individual */

typedef struct {
                short  pop_size;
                individual * members;
                individual child1;
                individual child2;
                int * sorted_by_fitness;
                individual best_so_far;
                short best_of_run_num;
                float best_fitness;
                function_table_entry func_table[MaxNumFunctions];
                short num_general_functions;
                short NumFunctions[NumBranches+NumADFs];
                short

FunctionAssignment[NumBranches+NumADFs][MaxNumFunctions];
                char ReadFiles[NumBranches+NumADFs][MAXSTRING];
                FILE * OutFile;
                } population; /* Structure for a population */


snode * create_tree(short n);
float random_float(float x);
```

STEADY STATE EVOLUTION