SCS-C: A C-language Implementation of a Simple Classifier System

K.P.Williams and S.A.Williams

RUCS/98/TR999/X

University of Reading

Department of Computer Science

Parallel, Emergent and Distributed Architectures Laboratory (PEDAL) Group

June 5, 1998

SCS-C: A C-language Implementation of a Simple Classifier System

KENNETH P. WILLIAMS AND SHIRLEY A. WILLIAMS
Department of Computer Science,
University of Reading,
PO Box 225, Whiteknights, Reading, UK, RG6 6AY

June 5, 1998

1 Introduction

SCS-C[1] is a pure ANSI C-language translation of the original Pascal SCS code presented by Goldberg[1]. It is a direct translation and contains no enhancements as such, it's operation is essentially the same as that of the original, Pascal version. The report is included as a concise introduction to the SCS-C distribution. It is presented with the assumptions that the reader has a general understanding of Goldberg's original Pascal code, and a good working knowledge of the C programming language. The report begins with an outline of the files included in the SCS-C distribution, followed by a discussion of the points of SCS-C where it differs from those of the Pascal version.

2 Files Distributed with SCS-C

The following is a list of files distributed with SCS-C, the routines contained within those files and the simple `#include` structure of the SCS-C distribution. The source files that make up SCS-C essentially correspond one-to-one with those presented in the Pascal version, with the addition of a single header file `scs.h` as described below.

`scs.h` This is the only header file used in SCS-C and is `#include`'ed into all other source files. It contains several `#define` definitions, notably: `LINELENGTH`, which determines the column width of printed output and

[1]**Disclaimer:** SCS-C is distributed under the terms described in the file `NOWARRANTY`. These terms are taken from the GNU Public License. This means that SCS-C has no warranty implied or given, and that the authors assume no liability for damage resulting from its use or misuse.

can be set to any desired positive value. It also contains all `typedef`'s, all `struct` definitions as well as prototypes of all functions used in the source files.

`advance.c` contains code to advance all variables by one time step.

`advance()` advance winner record.

`aoc.c` contains routines for the apportionment of credit.

`initaoc()` initialize clearing house record.

`initrepaoc()` initial report of clearinghouse parameters.

`auction()` auction among currently matched classifiers - returns the winner.

`clearinghouse()` distribute payment from recent winner to oldwinner.

`taxcollector()` collect existence and bidding taxes from population members.

`reportaoc()` report who pays to whom.

`aoc()` apportionment of credit coordinator.

`detector.c` convert environmental states to environmental message.

`detectors()` convert environmental state to environmental message.

`writemessage()` write message in bit-reverse order.

`reportdetectors()` write out environmental message.

`initdetectors()` dummy detector initialization routine.

`initrepdetectors()` dummy initial detectors report.

`effector.c` contains the single effector routine.

`effector()` set action in classifier output as dictated by auction winner.

`environ.c` controls multiplexor environment.

`generatesignal()` generate random signal.

`decode()` decode substring into a single unsigned binary value.

`multiplexoroutput()` calculate correct output of the 6-input/1-output multiplexor function (described in Goldberg [1]) being learned by SCS-C.

`initenvironment()` initialize the multiplexor environment.

`initrepenvironment()` write initial environmental report.

`writesignal()` write signal in bit-reverse order.

`reportenvironment()` write current multiplexor info.

`ga.c` contains genetic algorithm routines for SCS.

`initga()` initialize ga parameters.

`initrepga()` initial ga report.

`select()` select a single individual according to strength.

`mutation()` mutate a single position with specified probability.

`bmutation()` mutate a single bit with specified probability.

`crossover()` cross a pair at a given site with mutation on the ternary bit transfer.

`worstofn()` select worst individual from random subpopulation of size n.

`matchcount()` count number of positions of similarity.

`crowding()` replacement using modified De Jong crowding.

`statistics()` contains population statistics routines - computes max, avg, min, sum of strength.

`ga()` coordinate selection, mating, crossover, mutation and replacement.

`reportga()` report on mating, crossover and replacement.

`initial.c` contains routines to coordinate program initialization.

`initrepheader()` write a report header to a specified output file/device.

`interactiveheader()` clear screen and print interactive header.

`safe_malloc()` memory allocation routine which terminates program (with an error message) on failure.

`initialization()` coordinate input and initialization.

`repchar()` repeatedly write a character to stdout.

`skip()` skip lines.

`io.c` contains file input/output routines.

`page()` start a new page.

`open_input()` read filenames in interactive mode.

`open_output()` read filenames in interactive mode.

`perform.c` classifier matching routines.

- `randomchar()` set position at random with specified generality probability.
- `readcondition()` read a single classifier condition.
- `readclassifier()` read a single classifier.
- `countspecificity()` count condition specificity.
- `initclassifiers()` init population of classifier.
- `initrepclassifiers()` initial report on population parameters.
- `writecondition()` convert internal condition format to external format and write to file/device.
- `writeclassifier()` write a single classifier.
- `reportclassifiers()` generate classifiers report.
- `match()` match a single condition to a single message.
- `matchclassifiers()` compare all classifiers against current environmental message and create a list of matches.

`reinfor.c` functions for reinforcement and criterion.

- `initreinforcement()` initialize reinforcement parameters.
- `initrepreinforcement()` initial reinforcement report.
- `criterion()` return true if criterion is achieved.
- `payreward()` pay reward to appropriate individual.
- `reportreinforcement()` report award.
- `reinforcement()` make a payment if criterion is satisfied.

`report.c` report coordination functions.

- `reportheader()` send report header to specified file/device.
- `report()` report coordination routine.
- `consolereport()` write report to console.
- `plotreport()` write plot figures to file.

`scs.c` This contains the `main` SCS function.

- `main()`

`timekeep.c` contains the timekeeping routines.

- `addtime()` increment iterations counter and set carry flag if necessary.

`inittimekeeper()` initialize timekeeper.

`initreptimekeeper()` initialize timekeeper report.

`timekeeper()` keep time and set flags for time-driven events.

`reporttime()` print out block and iteration number.

`utility.c` this file contains various utility routines including the SCS `halt()` routine, and also the random number utility functions, which include;

`randomperc()` returns a single uniformly distributed, real, pseudo-random number between 0 and 1 using the subtractive method, as described by Knuth in [2].

`rnd(low, high)` returns a uniformly distributed integer between `low` and `high`.

`rndreal(low, high)` returns a uniformly distributed real between `low` and `high`.

`warmup_random()` primes the random number generator.

`advance_random()` generates batches of 55 random numbers.

`flip(p)` flips a biased coin, returning 1 with probability `p`, and 0 with probability `1-p`.

`halt()` Terminates SCS-C through prompting for user input.

`initrandomnormaldeviate()` initialization routine for function `randomnormaldeviate`.

`noise(mu, sigma)` generates a normal random variable with mean `mu` and standard deviation `sigma`.

`randomize()` asks user for random number seed.

`randomnormaldeviate()` is a utility used by `noise` - it computes a standard normal random variable.

The following are the test data files. All these are identical to those presented in Goldberg [1].

`environ.dta` This is a sample *efile*.

`ga.dta` This is a sample *gafile*.

`perfect.dta` This is a sample *cfile* for the perfect rule set experiments of Chapter 6.

`lessthan.dta` This is a less than perfect *cfile* for default hierarchy experiments of Chapter 6.

`reinf.dta` This is a sample *rfile*.

`time.dta` This is a sample *tfile*.

`class100.dta` This is a sample *cfile* for the clean-slate experiments of Chapter 6. Only 10 of each type of rule are shown for brevity.

`Makefile` is a simple UNIX makefile for SCS-C.

3 Implementation Features of SCS-C

3.1 Translation Issues and Memory Utilization

Several differences between the Pascal and ANSI-C programming languages forced a number of minor changes in the way SCS-C operates in comparison to the original SCS program. Firstly, in SCS-C all arrays are 1 element larger than their SCS originals. This is because in C all arrays are indexed from 0 rather than 1. Loop bounds were left unchanged to avoid the risk of introducing errors into the C implementation (due to time constraints).

Secondly, parameter passing in Pascal is achieved by reference rather than by value as in C. Consequently, it seemed easiest to make the variables in SCS-C dynamic and to pass parameters to/from functions so as to achieve call-by-reference parameter passing. This introduction of dynamic variables will allow easier extension of SCS-C in future experimentation since variables, such as population sizes, can be more easily varied.

The introduction of dynamic variables also has the effect of moving responsibility for memory management onto the programmer. This can be controlled via the use of the `safe_malloc()` function (defined in `initial.c`) and the *free* function (used in `scs.c`).

3.2 Input / Output

Most of the input *readln()* statements in Pascal were translated into equivalent *scanf()* calls in SCS-C. Program termination in the original SCS version however is achieved in the `halt()` function (defined in `utility.c`) via the use of Pascal `kbhit()` function. Since no equivalent function exists in ANSI-C the solution implemented in SCS-C is to initially prompt the user to enter a number of generations the genetic algorithm is to run for and then simply query the user whether they wish the run to continue or terminate. This leads to the following input screen (see Fig.1).

Output from SCS-C is essentially identical to that of the original SCS implementation. Two files (`plot.out` and `rep.out`) are produced, sample outputs as in Figs. (2, 3, 4, 5).

The SCS code may be easily adapted to operate in interactive mode (as indicated at points in `initialize.c` and `scs.c`).

```
prompt% ./scs
How many generations ? : 2000
 Enter random number seed, 0.0 to 1.0 -> 0.3333



|------------------------|
     Iteration = 50
     P correct = 0.960000
   P50 correct = 0.960000
|------------------------|



|------------------------|
     Iteration = 100
     P correct = 0.980000
   P50 correct = 1.000000
|------------------------|

    . . . . . . . . . .

|------------------------|
     Iteration = 2000
     P correct = 0.999000
   P50 correct = 1.000000
|------------------------|
Halt (y/n) ? >> y
```

Figure 1: Example SCS-C Input Screen

```
------------------------------------------------------------
|       SCS-C (v1.0) - A Simple Classifier System          |
|    (c) David E. Goldberg 1987, All Rights Reserved       |
|    C version by Kenneth P. Williams, U. of Reading       |
------------------------------------------------------------


Population Parameters
---------------------
Number of classifiers   = 10
Number of positions   = 6
Bid coefficient = 0.100000
Bid spread = 0.075000
Bidding tax = 0.010000
Existence tax = 0.000000
Generality probability = 0.500000
Bid specificity base = 1.000000
Bid specificity mult. = 0.000000
Ebid specificity base = 1.000000
Ebid specificity mult. = 0.000000


Environmental Parameters (Multiplexor)
--------------------------------------
Number of Address Lines = 2
Number of Data Lines = 4
Total Number of Lines = 6
```

Figure 2: SCS-C Parameters Report (cont...)

```
Apportionment of Credit Parameters
----------------------------------
Bucketbrigadeflag   =  False


Reinforcement Parameters
------------------------
Reinforcement reward  =  1.000000


Timekeeper Parameters
---------------------
Initial iteration        = 0
Initial block  = 0
Report period = 2000
Console report period = 50
Plot report period = 50
GA period = -1


Genetic Algorithm Parameters
----------------------------
Proportion to select/gen  = 0.200000
Number to select  = 1
Mutation Probability = 0.020000
Crossover Probability = 1.000000
Crowding Factor = 3
Crowding Subpopulation = 3
```

Figure 3: SCS-C Parameters Report

```
Snapshot report
---------------

[ Block : Iteration] = [ 0 : 0 ]


Current Multiplexor Status
--------------------------
Signal               =  0  0  0  0  0  0
Decoded address      = 0
Multiplexor output   = 0
Classifier output    = 0

Environmental message:   000000

  No.   Strength       bid   ebid    M   Classifier
--------------------------------------------------


     1  10.00000  0.00000  0.00000    ###000: [ 0 ]
     2  10.00000  0.00000  0.00000    ###100: [ 1 ]
     3  10.00000  0.00000  0.00000    ##0#01: [ 0 ]
     4  10.00000  0.00000  0.00000    ##1#01: [ 1 ]
     5  10.00000  0.00000  0.00000    #0##10: [ 0 ]
     6  10.00000  0.00000  0.00000    #1##10: [ 1 ]
     7  10.00000  0.00000  0.00000    0###11: [ 0 ]
     8  10.00000  0.00000  0.00000    1###11: [ 1 ]
     9  10.00000  0.00000  0.00000    ######: [ 0 ]
    10  10.00000  0.00000  0.00000    ######: [ 1 ]


New winner [1] : Old winner [1]


Reinforcement Report
--------------------
Proportion correct (from start) = 0.000000
Proportion correct (last 50) = 0.000000
Last winning classifier number  = 0
```

Figure 4: Report on SCS-C After 0 Generations

```
Snapshot report
---------------

[ Block : Iteration] = [ 0 : 2000 ]


Current Multiplexor Status
--------------------------
Signal               =  0  0  1  1  0  1
Decoded address      = 1
Multiplexor output   = 1
Classifier output    = 1

Environmental message:   001101

  No.   Strength       bid   ebid    M   Classifier
--------------------------------------------------


     1  9.09091  0.90909  1.03956    ###000: [ 0 ]
     2  9.09091  0.90909  0.82836    ###100: [ 1 ]
     3  9.09091  0.90909  0.91054    ##0#01: [ 0 ]
     4  9.09091  0.90909  0.96905  X ##1#01: [ 1 ]
     5  9.09091  0.90909  0.84879    #0##10: [ 0 ]
     6  9.09091  0.90909  0.97096    #1##10: [ 1 ]
     7  9.09091  0.90909  0.95323    0###11: [ 0 ]
     8  9.09091  0.90909  0.95478    1###11: [ 1 ]
     9  0.00000  0.00000  0.06038  X ######: [ 0 ]
    10  0.00000  0.00000  0.02542  X ######: [ 1 ]


New winner [4] : Old winner [5]


Reinforcement Report
--------------------
Proportion correct (from start) = 0.999000
Proportion correct (last 50) = 1.000000
Last winning classifier number  = 4
```

Figure 5: Report on SCS-C After 2000 Generations

4 Final Comments

SCS-C is intended to be a simple program for first time classifier system experimentation. It is not intended to be definitive in terms of its efficiency or the grace of its implementation. As such it's release may be seen as complimentary to the distribution of the C implementation of the Simple Genetic Algorithm (`SGA-C`) by Smith, Goldberg and Earickson in [3].

The authors are interested in the comments, criticisms, and bug reports from SCS-C users, so that the code can be refined for easier use in subsequent versions. Please email your comments to **K.P.Williams@reading.ac.uk**, or write to:

Ken Williams,
University of Reading,
Department of Computer Science,
Parallel, Emergent and Distributed
Architectures Laboratory (PEDAL),
PO Box 225, Whiteknights, Reading,
Berkshire, ENGLAND RG6 6AY

Acknowledgments

Ken Williams gratefully acknowledges the support of the Engineering and Physical Sciences Research Council (EPSRC) of Great Britain under grant number 94701308, and also of support from the Department of Computer Science, University of Reading.

References

[1] Goldberg, D.E., *Genetic Algorithms in Search Optimization and Machine Learning*, Addison-Wesley, Reading: MA, (1989).

[2] Knuth, D.E., *The Art of Computer Programming*, Vol. 2, (2nd Edition), Addison-Wesley, Reading: MA, (1981).

[3] Smith R.E, Goldberg D.E., and Earickson, J.A, *SGA-C: A C-language Implementation of a Simple Genetic Algorithm*, TCGA Report No. 91002, The Clearing House for Genetic Algorithms, Dept. of Engineering Mechanics, U. of Alabama, Tuscaloosa, AL 35487, (1994).