

A Description of Holland's Royal Road Function

Terry Jones
Santa Fe Institute
1660 Old Pecos Trail, Suite A
Santa Fe, NM 87505
terry@santafe.edu

June 30, 1994

Abstract

This note contains a description of John Holland's Royal Road function, which was presented at the Fifth International Conference on Genetic Algorithms in July 1993, and posted to the Internet Genetic Algorithms mailing list in August 1993 [2].

1 Introduction

The Royal Road functions were introduced in [5]. They were designed as functions that would be simple for a genetic algorithm to optimize, but difficult for a hillclimber. However, one form of hillclimbing [6], easily outperformed a genetic algorithm on what had been expected to be the simplest Royal Road function for that algorithm. Recently, Holland [2] presented a revised class of Royal Road functions that were designed to create insurmountable difficulties for a wider class of hillclimbers, and yet still be admissible to optimization by a genetic algorithm.

Holland used seven variables in the description he posted to the genetic algorithms community. In describing the class of functions, I will use his variable names where reasonable, and indicate deviations. Holland suggested a default value for each of these variables and in what follows I will use these values to provide illustration. A summary of these variables and their default values can be found in section 3.

The main description here is of the function posted to the genetic algorithms community. Holland has also suggested further generalizations, which are discussed in section 6.

The following terms will be used consistently in what follows: *block*, *complete*, *gap*, and *region*. These are introduced below and can be relied on to always mean exactly the same thing.

2 Description

The function takes a binary string as input and produces a real value. The function is used to define a search task in which one wants to locate strings that produce high function values. Section 5 provides some more information on fitness values.

The string is composed of 2^k non-overlapping contiguous regions, each of length $b + g$. With Holland's defaults, $k = 4$, $b = 8$, $g = 7$, there are 16 regions of length 15, giving an overall string length of 240. We will number the regions, from the left, as $0, 1, \dots, 2^k - 1$.

Each region is divided into two non-overlapping pieces. The first, of length b , will be called the block. The second, of length g , will be called the gap¹. In the fitness calculation, only the bits in the block part of each region are considered. The bits in the gap part of each region are completely ignored during fitness calculation. Holland's posting called the block part of each region by various names: "building blocks", "elementary building blocks", "schemata", "lower level target schemata", and "elementary (lowest-level) building blocks (schemata)".

The fitness calculation proceeds in two steps. Firstly, there is what Holland calls the PART calculation. Then follows the BONUS calculation. The overall fitness assigned to the string is the sum of these two calculations.

2.1 The PART Calculation

The PART calculation considers each block individually, and each in exactly the same fashion. Each block receives some fitness score, and these are added to produce the total PART contribution to the overall fitness.

The fitness of each block is based entirely on the number of 1 bits it contains². The idea is to reward 1's up to a certain point. Every 1 up to (and including) a limit of m^* adds to the block's fitness by v . Thus, with the default settings, a block with three 1's would have fitness $3 * 0.02 = 0.06$. If a block contains more than m^* 1's, but less than b 1's, it receives $-v$ for each 1 over the limit. With the default settings $m^* = 4$, and so a block with six 1's is assigned a fitness of $(6 - 4) * -0.02 = -0.04$.

Finally, if a block consists entirely of 1's (i.e. it has b 1's), it receives *nothing* from the PART calculation. Such a block will be rewarded in the BONUS calculation. If a block consists entirely of 1 bits, it will be said to be complete.

From the above, we can construct a table of fitness values based on the number of ones in a block. Table 1 gives the values for the default settings.

¹Holland did not give a name for this part of the regions, or give a variable name for its length. These gaps have been called "introns" (borrowing from biology) and have been used with varying degrees of success [1, 4] to alter the effects of crossover in genetic algorithms

²Holland used $m(i)$ to denote the number of 1's in block i . I will not use a variable.

1's in block	0	1	2	3	4	5	6	7	8
Block fitness	0.00	0.02	0.04	0.06	0.08	-0.02	-0.04	-0.06	0.00

Table 1: PART Fitness Values For Holland's Default Settings

2.2 The BONUS Calculation

The idea of the BONUS calculation is to reward completed blocks and some combinations of completed blocks. Holland gives rewards for attaining what he calls "levels". At the lowest level³, 0, rewards are given for complete blocks (i.e blocks that consist entirely of 1's). If such a block exists, it receives u^* as its fitness. Any additional complete blocks receive a fitness of u . Thus, with the default settings, a string that contained three complete blocks would receive a BONUS fitness of $1.0 + (2 * 0.3) = 1.6$ for level 0.

But that is not the end of the BONUS story. At the next level, *pairs* of blocks are rewarded. Suppose we label the blocks from left to right as $B_0, B_1, \dots, B_{2^k-1}$. The function then rewards any completed pair (B_{2i}, B_{2i+1}) for $0 \leq i < 2^{k-1}$. The rewards are calculated in the same way as they were done at level 0. The first completed pair of blocks receives u^* , and additional completed pairs receive u .

Notice that not all pairs of completed blocks are so rewarded. Only those whose indices are of the form $(2i, 2i + 1)$.

We can now calculate rewards for the next level, that of quadruplets of completed blocks, and this is done in the same fashion. In general, there are $k + 1$ levels, and at level $0 \leq l \leq k$, we are looking for contiguous sets of 2^l complete blocks, whose first block is labeled B_{i2^l} with $0 \leq i < 2^{k-l}$. At all levels, the first such set of complete blocks receives fitness u^* , and additional sets of completed blocks receive fitness u . The total fitness for the level is the sum of these fitnesses.

To make this more concrete, consider the function with Holland's default values. With $k = 4$ we have 16 regions and each contains a block of length $b = 8$. The BONUS fitness calculation rewards completed single blocks (this is level 0), and rewards the completion of the sets $\{B_0, B_1\}, \{B_2, B_3\}, \{B_4, B_5\}, \{B_6, B_7\}, \{B_8, B_9\}, \{B_{10}, B_{11}\}, \{B_{12}, B_{13}\}, \{B_{14}, B_{15}\}$, (level 1) $\{B_0, B_1, B_2, B_3\}, \{B_4, B_5, B_6, B_7\}, \{B_8, B_9, B_{10}, B_{11}\}, \{B_{12}, B_{13}, B_{14}, B_{15}\}$ (level 2) $\{B_0, B_1, B_2, B_3, B_4, B_5, B_6, B_7\} \{B_8, B_9, B_{10}, B_{11}, B_{12}, B_{13}, B_{14}, B_{15}\}$ (level 3) and $\{B_0, B_1, B_2, B_3, B_4, B_5, B_6, B_7, B_8, B_9, B_{10}, B_{11}, B_{12}, B_{13}, B_{14}, B_{15}\}$ (level 4) of completed blocks.

The total BONUS contribution to the fitness is computed by adding the fitness at each of the $k + 1$ levels.

³Holland numbered his lowest level 1 not 0.

Variable	Default
k	4
b	8
g	7
m^*	4
v	0.02
u^*	1.0
u	0.3

Table 2: Holland's Default Variable Settings

3 Summary Of Variable Defaults And Meanings

Table 2 shows the default settings suggested by Holland. Here is a brief summary of each of these.

k This determines the number of level 0 blocks. There will be 2^k of them. Equivalently, it determines the number of levels in the function, $k + 1$.

b The number of bits in each block. These bits are examined when the fitness is calculated.

g The number of bits in the gap following each block⁴. These bits are ignored when the fitness is calculated.

m^* This is used in calculating the PART fitness. It is the number of 1's that a block may have before being penalized for having too many. The penalty applies to blocks that have in excess of m^* 1's but not to those blocks that are complete.

v This is also used in calculating the PART fitness. If the number of 1's in a block is m^* or less, then each receives v . If the number of ones is greater than m^* but less than b , then each 1 over m^* *decreases* the fitness by v . If the block is complete, it receives no PART fitness.

u^* The fitness acquired in the BONUS calculation by the first (if any) completed set of blocks at each level.

u The fitness acquired in the BONUS calculation by the second and subsequent (if any) completed set of blocks at each level.

⁴The name is a slight misnomer, since the last block is followed by g bits which are not followed by another block.

4 Relationships Between Variables

The seven variables used to define this class of functions should be well understood before they are altered. They may be roughly divided into three groups:

- The variables k , b and g can be thought of as structural variables. These can be changed at will, each adjusts a specific aspect of the function (see above) and they do not interact in any subtle way with any other variables.
- The PART variables are m^* and v .
- The BONUS variables are u^* and u .

The PART and BONUS variables *do* interact subtly, and various settings can radically alter the nature of the function thus defined. The function was originally supposed to reward up to a certain number of ones in each block, then penalize any number greater than this, except for a complete block. If, for example, $u^* < vm^*$, this qualitative objective will not be satisfied. Here an optimal string (there are

$$2g2^k \binom{b}{m^*}^{2^k}$$

of them) will have m^* 1's in each block, since any more will result in a decreased fitness. Similarly, if $u < vm^*$ an optimal string will contain a single completed block and all others will have m^* 1's. This becomes even more odd when $u < vm^* < u^*$ since in this situation, the optimal bit pattern for a block depends on the contents of the other blocks – i.e. before any block is completed, the best a block can do is get completed, if there is exactly one completed block, the others will maximize the function by setting m^* bits, and finally, if there are several completed blocks, each of them would increase the string's fitness by moving to m^* 1's, but if they all do this, the pressure to return to a completed block returns!

These examples indicate that one should choose $vm^* < u \leq u^*$. There are a number of other more obvious inequalities amongst the variables that should be maintained. For example, $v \geq 0$ and $0 \leq m^* < b$.

5 Fitness Values With The Default Settings

With Holland's default settings, a maximally fit string has all 16 blocks complete. The gaps may contain any bit settings. Thus there are 2^{g2^k} maximally fit strings. The fitness of any of these strings is 12.8. This can be calculated as follows. The PART fitness for each block is zero. The BONUS fitness at level 0 is

$$\begin{aligned} u^* + u(2^k - 1) &= 1.0 + 0.3 * 15 \\ &= 5.5 \end{aligned}$$

At level 1, we have 8 completed pairs of blocks, for a total of $u^* + 7u = 3.1$, at level 2 we have 4 completed quadruples, giving $u^* + 3u = 1.9$, at level 3 we have 2 8-tuples, for $u^* + u = 1.3$, and at level 4 we have $u^* = 1.0$. The total fitness is then

$$5.5 + 3.1 + 1.9 + 1.3 + 1.0 = 12.8$$

At the other end of the scale, it is possible for fitness to be negative. If every block contains $b - 1$ 1's, then each will receive $-v(b - 1 - m^*)$ in the PART calculation. The minimal fitness will therefore be the product of this value and 2^k . The BONUS calculation will not add anything since there are no complete blocks. With Holland's defaults, the minimum fitness is -0.9 .

6 Generalizations Of The Function

There are several conceptually simple ways one can generalize the class of functions described above. Some of these are currently in use [3]. This is by no means an exhaustive list, other possibilities, and some of the following, are presented in [1]. Once the basic function is understood, it is relatively simple to come up with plausible variations on the theme.

- The lengths of blocks and gaps need not be constant, one could specify 2^k different block and gap lengths.
- The maximal block bit pattern need not be all 1's. Holland suggests that each block's maximal pattern be randomly generated. To evaluate a string, one could initially XOR each block of the string with the maximal bit pattern for that block and then evaluate as normal.
- Holland has suggested the possibility of defining "potholes at every level" [3]. In this scheme, the BONUS fitnesses would be calculated in a manner akin to the PART fitnesses. At level 0, the PART calculation could proceed as usual. At every higher level, a number of components from the previous level would need to be completed. Suppose, for simplicity, that this number also happened to be b . Then fitness at each level above the lowest would be calculated using m^* and b (and possibly v or some other variable), as for the PART calculation at level 0. This would create a "pothole" at every level of the hierarchy.
- Blocks need not be contiguous. In the most general case, the block and gap bits could be spread randomly through the string. One would imagine that this would make the problem more difficult for a genetic algorithm using one- or two-point crossover (as opposed to uniform crossover) since the building blocks would now not have short defining length⁵.

⁵The expected defining length for a block of length b embedded randomly in a string of length n is $\mathcal{O}(n)$

- The choice of which sets of blocks to reward in the BONUS calculation seems a little arbitrary. For example, there is no reward for getting any set of three blocks completed. Even at the levels that rewards are given, some sets are rewarded and others are not. For example, completing the set $\{B_0, B_1\}$ will increase a string's fitness at level 1, whereas completing the set $\{B_1, B_2\}$ will not. The choice of which sets to reward originates with the choices for R2 in [5] which were made for simplicity.

7 Other Minor Points

Here are some relatively minor points.

- It is not possible to assign a setting to the variables to produce either of the functions R1 or R2 defined in [5]. In fact, the Royal Road functions, as defined in that paper, include the class of functions that Holland proposes. More than this, the class includes *all* functions $f : \{0, 1\}^n \rightarrow \mathfrak{R}$ as one can assign any desired fitness to each of the 2^n schema of order n to produce any desired function. But this is to ignore their intent, which is to provide a useful way to construct, think about and concisely notate a class of functions of interest to people studying genetic algorithms. To these ends, they are most useful when used to express a function in which a relatively small number of low-order schema are being rewarded.
- The intermediate building blocks of R2 are not present in R1 [5], making it possible to compare algorithm performance in the two situations. It is natural to ask how the hierarchical levels of Holland's class of functions are most naturally removed. One reasonable answer to this is to compute the fitness by doing the normal PART calculation and then doing only the lowest level BONUS calculation.
- There doesn't seem to be any good reason for including the last gap on the string. If the purpose of such gaps is to introduce introns that (it is hoped) will alleviate hitchhiking problems, then the last gap could be omitted. Including the last gap does make the definition of the function simpler though.
- Holland [2] mentions "climbing levels" of the hierarchy and of "achieving" levels. These mean the same thing. A string that receives a BONUS fitness at level l (and not at any higher level) can be said to have reached or achieved level l . An algorithm may be said to be climbing to higher and higher levels if, over time, it finds strings that reach higher and higher levels. If one imagines the blocks as forming the base of a triangle, the pairs of blocks as forming the next level and so on up to the apex set (containing all blocks), then the progress of an algorithm, measured in terms of the highest level reached over time, can be thought of as a climb up these levels.

To say that an algorithm achieves a level therefore does not mean that it has found *all* the sets of blocks at a certain level. It means that it has found a single one. It is not possible to find all the sets of blocks at any level without simultaneously finding

all sets at all levels and thereby solving the entire problem. If the problem is not yet solved, there must exist at least one set of blocks at every level that is not complete.

8 Implementation

A C language implementation of Holland's function class, and other information (including a postscript version of this document) is available for anonymous ftp from ftp.santafe.edu in the directory pub/Users/terry/jhrr/jhrr.tar.Z.

The following illustrates how the file may be retrieved. Your prompt is a percent sign. What the system prints is shown in a *slanted* font, and what you type is in sans serif.

```
% ftp ftp.santafe.edu
Connected to santafe.edu.
220- *****
220-
220- Anonymous access to the FTP area at SantaFe.edu is available:
220-
220- ftp ftp.santafe.edu
220- Login: anonymous
220- Password: (Your email address)
220-
220- *****
220-
220 sfi FTP server (Version 2.0WU(10) Mon Apr 12 10:49:51 MDT 1993) ready.
Name (santafe.edu:terry): anonymous
331 Guest login ok, send your complete e-mail address as password.
Password: Enter your email address.
230-
230- SFI FTP - SFI Anonymous FTP root directory.
230-
230- Directory: ~ftp Path: ~ftp
230-
230- Welcome to the FTP area at SantaFe.edu...
230-
230- Everything useful is in the pub directory. Type "cd pub" ...
230-
230- If you have any questions or problems with this service,
230- please send email to <ftp@santafe.edu>.
230-
230-Please read the file README.Z
```

```
230- it was last modified on Fri Mar 19 15:29:44 1993 - 329 days ago
230 Guest login ok, access restrictions apply.
ftp> cd pub/Users/terry/jhrr
250 CWD command successful.
ftp> binary
200 Type set to I.
ftp> get jhrr.tar.Z
200 PORT command successful.
150 Opening BINARY mode data connection for jhrr.tar.Z (NNNN bytes).
226 Transfer complete.
local: jhrr.tar.Z remote: jhrr.tar.Z
NNNN bytes received in NNNN seconds (NNNN Kbytes/s)
ftp> quit
```

You have just retrieved a number of files which were archived into a single file that was then compressed. The next job is to reverse these steps to recover the originals. To do this:

```
% uncompress jhrr.tar.Z
% tar xf jhrr.tar
```

This should result in the creation of a new directory, called **jhrr**. Check that that directory has been created. If so, and you have received no error messages, it is safe to remove the bundled distribution file with

```
% rm jhrr.tar
```

You can now consult the file README in the **jhrr** directory.

9 Acknowledgements

Thanks to Derek Smith for useful conversations and for comments on the text and to Stephanie Forrest, John Holland and Melanie Mitchell for answering questions about the precise nature of their functions.

References

- [1] Forrest, S. and Mitchell, M. [1993] "Relative Building Block Fitness and the Building Block Hypothesis." In *Foundations of Genetic Algorithms 2*, edited by L. D. Whitley. Los Altos, CA: Morgan Kaufmann, 1993.
- [2] Holland, John H. [1993], "Royal Road Functions" Internet Genetic Algorithms Digest vol. 7, issue 22, Aug 12th 1993.

- [3] Holland, John [1994], Personal communication.
- [4] Levenick, James R. [1991] “Inserting Introns Improves Genetic Algorithm Success Rate: Taking a Cue From Biology.” In Richard K. Belew and Lashon B. Booker (Eds.) *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA, pp. 123–127.
- [5] Mitchell, M., Forrest, S. and J. H. Holland [1992] “The Royal Road for Genetic Algorithms: Fitness Landscapes and GA Performance.” In *Proceedings of the First European Conference on Artificial Life*. Cambridge, MA: MIT Press/Bradford Books, 1992.
- [6] Palmer, R. [1992], Personal communication.