The Quadratic Assignment Problem in Code Optimization for a Simple Universal Turing Machine

Paul W. Rendell

Department of Computer Science
University of the West of England
Bristol BS6 Y

The author's simple universal Turing machine requires the description of a specific Turing machine as data. In this article the coding of a specific Turing machine for unary multiplication is described. The description is a list of transitions with unary coded links between them. The order of the transitions in the list therefore makes a large difference in the size of the resulting code and the speed at which it runs. Finding the optimum ordering is a quadratic assignment problem. A simple neighborhood search procedure starting from random initial samples is described. It was sufficient to locate the optimum solution for a Turing machine with 23 transitions. The reason for the success was found to be due to the large basins of attraction for good solutions.

1. Introduction

In 2000, the author constructed a Turing machine in Conway's Game of Life [1]. Figure 1 shows an image of this machine with the magnified portion showing details of one of the stack cells that represent the Turing machine's tape. In 2009, this was extended to a simple Game of Life universal Turing machine (GoL-UTM) [2]. Figure 2 shows an image of this together with an image of the original at the same scale. In 2011, growing stacks were added to make a fully universal Turing machine; an image is shown in Figure 3.

In this paper we describe aspects of two universal Turing machines in Conway's Game of Life. The first of these is Wolfram's two state three symbol machine [3]. Section 2 shows the operation of this machine in a cut-down version of the author's Turing machine [1]. In subsequent sections we describe the coding of a unary multiplication Turing machine to run in the GoL-UTM. In particular, we describe the procedure used to optimize the order of the transitions that provide the universal Turing machine with the description of the unary multiplication Turing machine. This optimization is a quadratic assignment problem.

https://doi.org/10.25088/ComplexSystems.20.1.1

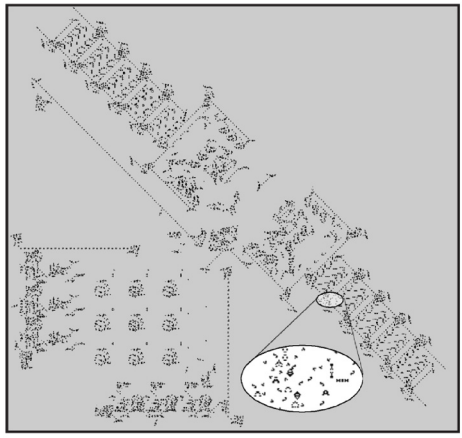

Figure 1. The Game of Life Turing machine [1].

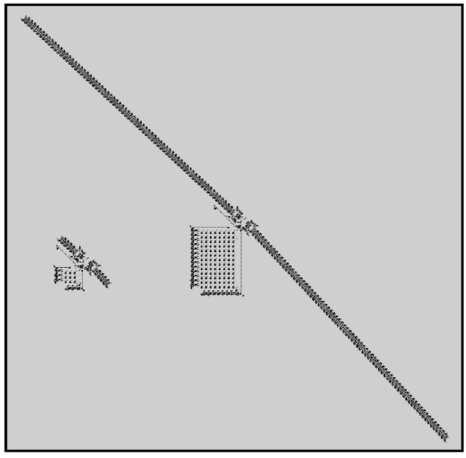

Figure 2. Size comparison of the Game of Life universal Turing machine and the original Turing machine.

https://doi.org/10.25088/ComplexSystems.20.1.1

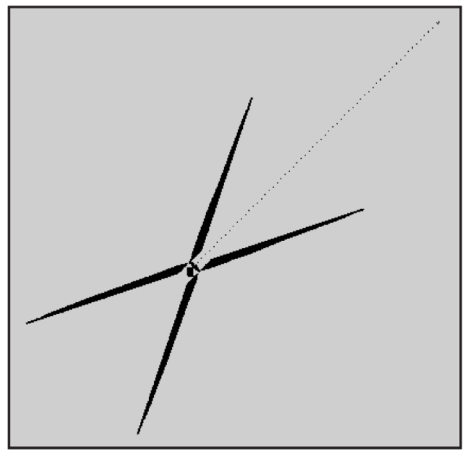

Figure 3. The full universal Turing machine with growing stacks. The data to program the stack is provided externally and can be seen as the line of dots from top right to the center.

The Game of Life is a cellular automaton invented by J. H. Conway [4]. It has an infinite universe divided into cells, where each cell takes a state from a binary set and updates its state according to certain strict rules. All cells change their states simultaneously in discrete time. For Conway's Game of Life the cells have two states, usually called live and dead, and the rules are based on the number of neighboring cells that are alive.

A Turing machine is a finite state machine that interacts with an infinite data storage medium. The data storage medium takes the form of an unbounded tape on which symbols can be written and read back via a moving read/write head.

A universal Turing machine is a Turing machine that can emulate any another Turing machine. It has on its tape a description of that machine and its tape. A universal Turing machine can perform any calculation simply by emulating a more dedicated machine. Turing first described his universal Turing machine in 1936 [5].

The quadratic assignment problem is an optimization problem involving allocation. Typically it involves allocating facilities to locations in such a way as to minimize cost. This might be the cost of moving things between locations given the requirements of facilities and the transport costs between locations. In this case, the simple universal Turing machine requires a description of the specific Turing machine in the form of a list of transitions. The length of a coded transition depends on the relative position of related transitions and the processing speed depends on both the location of the transition in the list and the relative location of the next transition.

https://doi.org/10.25088/ComplexSystems.20.1.1

2. Wolfram's Two State Three Symbol Turing Machine

The smallest known universal Turing machine is Wolfram's two state three symbol machine [3]. This is small enough to fit in the author's original Turing machine in Conway's Game of Life [1], which has three states and three symbols. Wolfram's two state three symbol machine was proved to be universal by Alex Smith in 2007 [6]. The coding of the Turing machine tape for universal Turing machine behavior creates a tape much larger than can be demonstrated in Conway's Game of Life.

Wolfram's two state three symbol machine was coded into a cut-down version of the Game of Life Turing machine. The cut-down version runs slightly faster than the original three state version at 10560 life generations per Turing machine cycle. The three symbols are coded on the Turing machine stacks as: no gliders, one glider at the bottom of the stack cell, and one glider in the middle of a stack cell. Figure 4 shows a snapshot of the machine. Figure 5 shows the two stacks after one complete cycle with a black tape. The stack contents are highlighted. Figure 6 shows the stack contents after 13 cycles. Note that the symbol under the read/write head is not visible as it is cycling through the finite state machine. Table 1 shows the first 13 cycles diagrammatically with the nonblank symbols shown as `L` for low glider and `M` for middle glider.

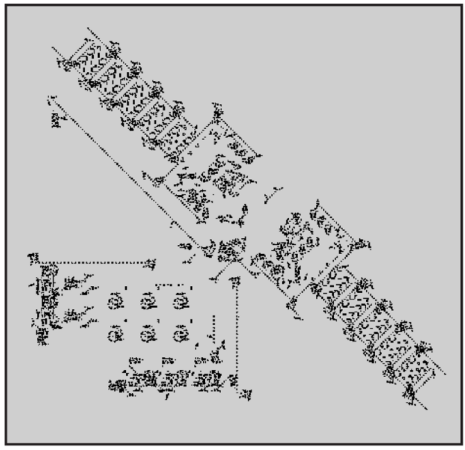

Figure 4. Game of Life two state three symbol Turing machine.

https://doi.org/10.25088/ComplexSystems.20.1.1

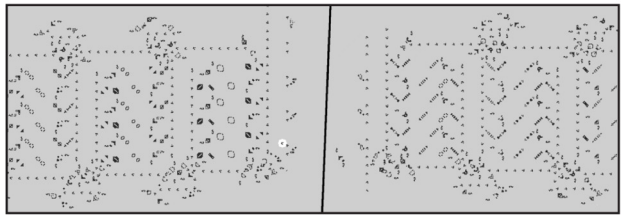

Figure 5. The tape after one cycle at generation 21 130.

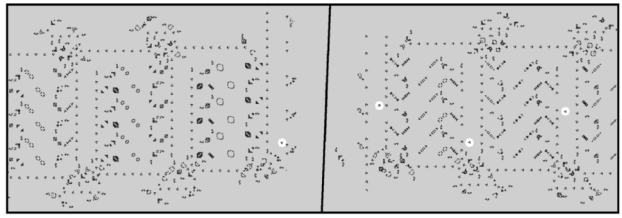

Figure 6. The tape after 13 cycles at generation 147 850.

Cycle 1			L	[ ]					
Cycle 2				[L]	M				
Cycle 3				[ ]	M	M			
Cycle 4			L	[M]	M				
Cycle 5		L		[M]		□			
Cycle 6			L	[ ]	L				
Cycle 7		L	L	[L]					
Cycle 8	L	L	M	[ ]					
Cycle 9		L	L	[M]	M				
Cycle 10			L	[L]	L	M			
Cycle 11				[L]	M	L	M		
Cycle 12				[ ]	M	M	L	M	
Cycle 13			L	[M]	M	L	M		

Table 1. The first 13 cycles of Wolfram's two state three symbol Turing machine. The symbol under the read/write head is shown in brackets [ ].

https://doi.org/10.25088/ComplexSystems.20.1.1

3. The Game of Life Simple Universal Turing Machine

The GoL-UTM [2] was required to meet practical considerations as well as theoretical. One such consideration is speed, the ability to demonstrate the complete operation of the universal Turing machine in a reasonable time. Universal Turing machines with low state/symbol counts such as Wolfram's (two states with three symbols) [3, 6] and Rogozhin's (four states with six symbols) [7] use tag systems that run in exponential time. In common with the Neary and Woods [8] design for small fast universal Turing machines, the design for the GoL-UTM is a direct simulation of a Turing machine. It uses relative indexing to locate transitions between states.

The Game of Life Turing machine had a capability of eight symbols and 16 states. The design of the GoL-UTM made use of the extra symbols and states to simplify the processing cycle—it required all eight symbols and 13 of the states. One simplification utilized the definition of a transition that Minsky [9] used in his 2-tag Post machine as described later in Section 5. This removed the need to maintain a separate record of the current state or symbol. The extra states were also used to speed up the processing. It was found possible to have just one traverse of the GoL-UTM read/write head between the Turing machine's current transition and its current read/write head position per GoL-UTM cycle.

4. The Example Turing Machine

The initial example Turing machine used to demonstrate the universal Turing machine [2] was the same as used in the original Game of Life Turing machine [1]. This was deliberately a very modest machine in order to keep the Game of Life pattern as small as possible so that it could be displayed by the tools then available. The tools we have today—for example, Golly [10], which uses the hashlife [11] algorithm—are much more powerful and can display much larger patterns. It was therefore decided to use a more complex Turing machine in the GoL-UTM in order to demonstrate its capabilities more fully.

A Turing machine to perform unary multiplication was chosen. It was written by D. Boozer at the California Institute of Technology and does not require any code modifications to run on the universal Turing machine and does not have large input or output. This machine has 16 states and two symbols. The state transitions are listed in Table 2. The initial tape is shown in Table 3. The final tape is shown in Table 4. This Turing machine takes 443 cycles to multiply four by four.

https://doi.org/10.25088/ComplexSystems.20.1.1

State/ Symbol	Next State	Write	Move	State/ Symbol	Next State	Write	Move
01/0	01	0	L	09/0	10	0	R
01/1	02	0	L	09/1	09	1	R
02/0	03	0	L	10/0	12	0	R
02/1	02	1	L	10/1	11	1	R
03/1	04	0	L	11/0	01	1	L
04/0	05	0	L	11/1	11	1	R
04/1	04	1	L	12/0	13	0	R
05/0	06	1	R	12/1	12	0	R
05/1	05	1	L	13/0	13	0	L
06/0	07	0	R	13/1	14	0	L
06/1	06	1	R	14/0	15	0	L
07/0	09	1	R	14/1	14	0	L
07/1	08	1	R	15/0	16	0	L
08/0	03	1	L	15/1	15	1	L
08/1	08	1	R	16/0	Halt	0	R

Table 2. State transitions of the unary multiplication Turing machine.

0	0	0	0	0	0	1	1	0	1	1	0
											$\widehat{1}$

Table 3. The unary multiplication Turing machine's initial tape consists of two strings of 1s, the length of which represents the value. The Turing machine starts in state 1 with the read/write head just to the right of the rightmost 1.

0	1	1	1	1	0	0	0	0	0	0	0
$\widehat{16}$											

Table 4. The unary multiplication Turing machine's final tape consists of one string of 1s, the length of which represents the value. The Turing machine stops in state 16 with its read/write head just to the left of the leftmost 1.

5. Coding the New Example Turing Machine

The transitions of the example Turing machine are coded onto the GoL-UTM tape in the format used by Minsky [9] in his Post machine. The machine cycles for processing these transitions start by writing a symbol on the tape, moving the read/write head, and reading the symbol on the tape at the new position. Then there is a choice of the next transition depending on the value read. The cycle then continues with

https://doi.org/10.25088/ComplexSystems.20.1.1

the processing of the next transition overwriting the symbol used to select it.

Table 5 shows the list of transitions for the unary multiplication Turing machine in this format. The last column of Table 5 shows the flow information that is used as the frequency data in the optimization in Section 6. Finding the optimum ordering of these transitions is discussed in Section 6.

				Next Transition for Symbol		
No.	State From - To	Symbol Written	Move Direction	0 Read	1 Read	Flow
1	11 - 11	1	R	2	1	12
2	11 - 10	0	R	4	3	8
3	10 - 9	1	R	22	3	3
4	10 - 12	0	R	7	4	4
5	13 - 14	0	L	10	5	4
6	13 - 13	0	L	6	5	10
7	12 - 13	0	R	6	5	1
8	15 - 15	1	L	9	8	16
9	15 - 16	0	L	H	H	1
10	14 - 15	0	L	9	8	1
11	1 - 2	0	L	14	15	4
12	1 - 1	0	L	12	11	0
13	3 - 4	0	L	19	18	16
14	3 - 3	0	L	14	13	0
15	2 - 2	1	L	14	15	10
16	5 - 5	1	L	17	16	120
17	5 - 6	1	R	21	17	16
18	4 - 4	1	L	19	18	144
19	4 - 5	0	L	17	16	16
20	7 - 8	1	R	23	20	12
21	6 - 7	0	R	1	20	28
22	9 - 1	1	L	12	11	3
23	8 - 3	1	L	14	13	12

Table 5. The unary multiplication Turing machine transitions in the format required by the GoL-UTM.

Table 6 shows the optimally ordered list of transitions for the unary multiplication. The GoL-UTM's tape shown in Table 7 consists of the tape of the example machine followed on the right by the list of

https://doi.org/10.25088/ComplexSystems.20.1.1

transitions of the example machine. The initial transition is twelfth in the list; it happens to be transition 12 in Table 5 as well. The transition numbers are not coded on the tape. The movement direction is coded as "0" for left and "1" for right. The links to the next transition are a unary coding of the number of transitions to skip either forward, with a list of 1s or backward, with a list of 0s. A zero length list is used to repeat the same transition and the special string "10" represents halt. A separator *C* separates the two links and each transition starts and ends with the separator *M*. In addition, the GoL-UTM codes its working position in the current transition and the position of the example Turing machine's read/write head by using a marked form of the tape coding between these positions. The marked form of the symbols {0, 1, *C*, *M*} are {*A*, *B*, *D*, *X*}. Thus, the first transition on the tape is coded *XBABDX* because it is to the left of the GoL-UTM's working position. It is transition 16 in Table 5, the symbol to write is 1, the direction is left, and the next transitions are one forward and this one again.

Transition Number	Symbol Written	Move Direction	Next Transition for 0 Read	Next Transition for 1 Read
16	1	L	17	16
17	1	R	21	17
19	0	L	17	16
18	1	L	19	18
13	0	L	19	18
21	0	R	1	20
20	1	R	23	20
23	1	L	14	13
14	0	L	14	13
15	1	L	14	15
11	0	L	14	15
12	0	L	12	11
22	1	L	12	11
1	1	R	1	1
3	1	R	22	3
2	0	R	4	3
4	0	R	7	4
7	0	R	6	5
6	0	L	6	5
5	0	L	10	5
10	0	L	9	8
8	1	L	9	8
9	0	L	H	H

Table 6. The unary multiplication Turing machine transitions in the format required by the GoL-UTM.

https://doi.org/10.25088/ComplexSystems.20.1.1

..	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	1	1	1	0	1	1	1	1
0	A	A	X	B	A	B	D	X	B	B	B	B	B
B	D	X	A	A	A	D	A	A	X	B	A	A	D
X	A	A	A	A	D	A	X	A	B	B	B	B	B
B	B	B	B	D	B	X	B	B	B	D	X	B	A
B	D	A	A	A	X	A	A	D	A	A	A	A	X
B	A	A	D	X	A	A	A	A	D	A	X	0	0
											$\widehat{1}$		
C	0	M	1	0	0	C	0	0	M	1	1	1	1
C	M	1	1	0	0	C	M	0	1	1	C	0	M
0	1	1	C	M	0	1	1	C	1	1	M	0	0
C	1	M	0	0	1	C	M	0	0	1	1	C	1
M	1	0	1	C	M	0	0	1	0	C	1	0	M
0	0	0	0	0	0	0	0	0	0	0	0	0	..

Table 7. GoL-UTM initial tape for unary multiplication.

It takes 56 561 transitions of the GoL-UTM to convert the initial tape shown in Table 7 to the final tape, the first part of which is shown in Table 8. Each GoL-UTM cycle takes 23 040 Game of Life generations giving just over 1300 million generations to complete the program. This took less than 10 minutes on the author's laptop with the fixed length stack version of the GoL-UTM running in Golly [10].

1	A	B	B	B	B	B	B	B	B	B	B	B	B	B
B	B	B	A	A	A	A	A	A	A	A	A	A	A	A
A	X	B	A	B	D	X	B	B	B	B	B	B	D	..

Table 8. Part of the GoL-UTM final tape for unary multiplication.

https://doi.org/10.25088/ComplexSystems.20.1.1

6. Optimizing the Order of Transitions

6.1 Problem Definition

The GoL-UTM will work with the transitions in any order. However, the order makes a great deal of difference in the size of the list and the speed of operation.

The optimization for size involves minimizing the lengths of the unary links between transitions. In order to maximize speed the links used most frequently should be shorter and transitions used most often should be closer to the left to minimize the GoL-UTM's read/write head movement to and from the Turing machine's tape.

This problem can be formulated as the classic quadratic assignment problem. This is an NP-hard optimization problem. It was first proposed by Koopmans and Beckmann in 1957 [12] as a mathematical model for maximizing profit when production is distributed over a number of sites. The objective is to find the optimum location for each plant to maximize profit and minimize transportation costs.

In our case, the equivalent is the allocation of transitions to positions in the list. We require the more general form with a linear component as proposed by Koopmans and Beckmann in order to cope with the "closest to the left" requirement for speed optimization. Most authors discarded this linear term as it is easy to solve [13].

The task is to find an allocation $O = [o_x]$ which is a permutation of the integers 1 to n, where n is the number of allocations that must be made and o_x is the activity allocated to location x.

The cost of an allocation is calculated using three matrices, $F = [f_{ij}]$, $D = [\mathrm{d}_{xy}]$, and $C = [\mathrm{c}_{ix}]$. The flows between activities is F, in our case how often a link between two transitions is processed. The cost related to the distance between two locations is D, in our case the distance between the two transitions in the list. The cost of allocating an activity to a location is C, which in our case is the frequency of use of the transition multiplied by how far this location is from the left.

The cost function which must be minimized is:

$$\sum_{x=1}^{n} \sum_{y=1}^{n} f_{o_x o_y} . d_{xy} + \sum_{x=1}^{n} c_{o_x x}. \tag{1}$$

The values for these matrices for the unary multiplication transitions in Table 6 are shown in Table 9 for F, equation (2) for D, and Table 10 for C:

$$D = [d_{xy}] \quad \text{where} \quad d_{xy} = |y - x|. \tag{2}$$

https://doi.org/10.25088/ComplexSystems.20.1.1

$$\begin{bmatrix}
10072, 10048, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 \\
0, 0, 10034, 10046, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 \\
0, 0, 10015, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 10015, 0 \\
0, 0, 0, 10032, 0, 0, 10008, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 \\
0, 0, 0, 0, 10032, 0, 0, 0, 0, 10008, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 \\
0, 0, 0, 0, 10029, 10071, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 \\
0, 0, 0, 0, 10003, 10017, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 \\
0, 0, 0, 0, 0, 0, 0, 10151, 10009, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 \\
0, 0, 0, 0, 0, 0, 0, 0, 20010, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 \\
0, 0, 0, 0, 0, 0, 0, 10009, 10000, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 \\
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 10000, 10040, 0, 0, 0, 0, 0, 0, 0, 0 \\
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 10000, 10000, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 \\
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 10144, 10016, 0, 0, 0, 0 \\
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 10000, 10000, 0, 0, 0, 0, 0, 0, 0, 0, 0 \\
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 10000, 10100, 0, 0, 0, 0, 0, 0, 0, 0 \\
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 11059, 10141, 0, 0, 0, 0, 0, 0 \\
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 10058, 0, 0, 0, 10102, 0, 0 \\
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 11296, 10144, 0, 0, 0, 0 \\
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 10141, 10019, 0, 0, 0, 0, 0 \\
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 10060, 0, 0, 10060 \\
10140, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 10140, 0, 0, 0 \\
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 10030, 10000, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 \\
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 10120, 10000, 0, 0, 0, 0, 0, 0, 0, 0, 0
\end{bmatrix}$$

Table 9. Quadratic assignment problem flow matrix for unary multiplication transitions.

$$\begin{bmatrix} 60,40,15,20,20,50,5,80,5,5,20,0,80,0,50,600,80,720,80,60,140,15,60 \end{bmatrix}$$

Table 10. Quadratic assignment problem linear matrix for unary multiplication transitions. This is row 1 for position 1; subsequent rows use these values multiplied by the row number.

6.2 Solution Method

The initial plan was to try a multistart tabu search along the lines proposed by James et al. [14]. Following early results was advice in [14] that "high quality results can be obtained from approaches that capitalize on the strategic use of information learned during the search process." A study of the structure of this particular problem was undertaken. This involved generating random allocation samples and using a neighborhood search to locate the closest local optimum. The number of times each local optima was found was recorded as well as the average number of steps the neighborhood search took to locate the local optima from the random starting points.

https://doi.org/10.25088/ComplexSystems.20.1.1

This approach is also similar to the multistart approach of Boese [15] except that the greedy method was not used because we wished to locate the closest local minimum as part of studying the problem structure.

The neighborhood definition used for the search is all the allocations that can be generated by swapping the positions of any two transitions.

The local search procedure used generated a random allocation and then looked at all neighboring allocations. If a better one was found, a step was taken and the neighborhood search was repeated centered on this sample. This was continued until a local minimum was located.

The neighborhood definition was very successful with this particular problem. It took about 20 steps on average to find the local minima in a random sample. This is a long way considering that any point in the sample space can be reached in 22 steps.

6.3 Analysis of Results

The following analysis is of one run of 2000 random samples.

Figure 7 shows a plot of the number of hits on local minima against the cost function. This clearly shows an upward trend toward the overall minimum value. Analysis of the data shows that the 10 best local minima had 70 hits and all 10 minima were found in the first 27 of these, which also fell in the first half of the total of 2000 samples. That means that this area was hit 43 times without finding another local minima. The rest of the search space received 1930 hits that found 1811 local minima, finding them at a rate of one per two hits right up to the last sample.

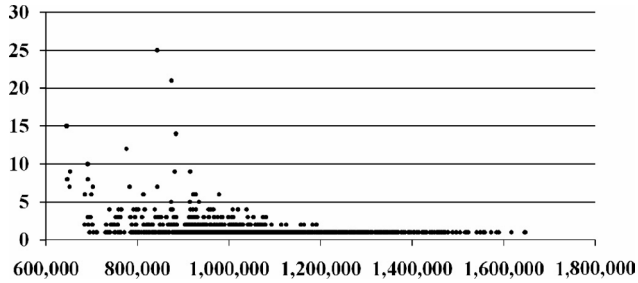

Figure 7. Plot of the number of random allocations that optimized to the same local minima against the cost function value of the local minima.

Figure 8 shows the number of local minima against the cost function value. This looks remarkably symmetrical in comparison to Figure 7.

https://doi.org/10.25088/ComplexSystems.20.1.1

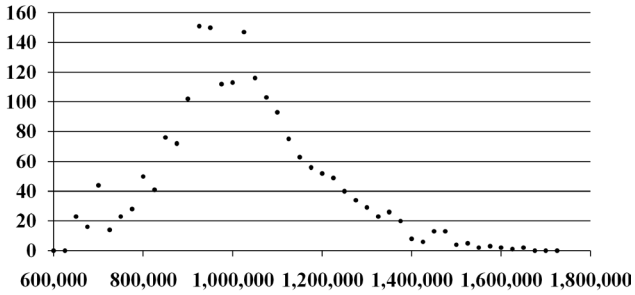

Figure 8. Plot of the number of local minima against the cost function value. The *y* axis is the count of minima in bands 25 000 wide of the cost units.

6.4 Comparison with the Greedy Search

It is noted that neither Reeves [16] nor Merz [17] noticed any evidence for large basins of attraction. Reeves was looking for a "big valley" structure in a flowshop sequencing problem in 1997. Merz was looking at the relationship between local optima rather than how they were found. One reason that they did not see evidence of the large basins of attraction could be that they were using a greedy local search. This search takes a step toward the first better solution found rather than looking at all the solutions in a neighborhood before taking a step. In order to compare the effect of this we collected samples using this search. Figure 9 shows a plot of the number of hits on local minima against the cost function. This is very similar to Figure 7 with the following differences.

- The number of hits on any optima were lower with the greedy search.
- The average number of steps to a local optima was up for the greedy search to 71 steps from 20 steps.
- The lack of local optima in the bottom left corner of the graph is not nearly so obvious.

This suggests a threshold in the size of basins of attraction above which the simple local search is significantly better than the greedy search. The small steps made by the greedy search locate small optima, while the local search makes the biggest possible step and always finds the deepest local optimum.

However, the greedy version was completed in less than a quarter of the time and is therefore the faster method.

https://doi.org/10.25088/ComplexSystems.20.1.1

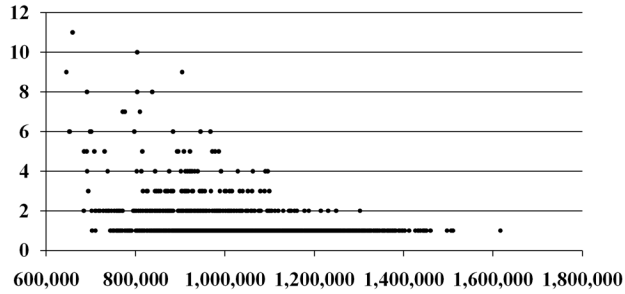

Figure 9. Plot of the number of random allocations that optimized to the same local minima found by the local neighborhood greedy search against the cost function value.

7. Expected Basin of Attraction Size

A possible explanation for the large basins of attraction for good local minima is the structure created by the neighborhood definition. It was decided to establish a baseline expected size for basins of attraction. A simulation was performed treating the cost function as a mapping for random allocation and assuming that the actual scores for neighbors are independent.

7.1 Simulation for Expected Size of the Basin of Attraction

The simulation used the standard normal distribution for the distribution of allocation scores. For a quadratic assignment problem with 23 items to allocate and a neighborhood defined by swapping allocations, there are 253 different neighboring allocations to each allocation.

For each trial i of the simulation, a local minima L_i was given a score S_i from the range of interest. The 253 neighbors N_{ij} of L_i were given scores V_{ij} for all j in the range [0:252]. The values V_{ij} were drawn from the standard normal distribution with the additional constraint that $V_{ij} > S_i$.

The set B_i of members of the basin of attraction of the local minima L_i was initialized with all V_{ij}. The simulation then examined each member B_{ik} of set B_i in turn. Each B_{ik} had 253 neighbors, B_{ikl} for l in the range [0:252]. Each B_{ikl} had B_{ik} as one neighbor and 252 others. If all these other neighbors had scores larger than the score of B_{ik} then B_{ikl} was added to B_i. All the scores were drawn from the standard normal distribution.

The process continued until all members of B_i were examined. The result of one run is shown in Figure 10. The points plotted are the average of 40 samples. The average number of steps followed the same

https://doi.org/10.25088/ComplexSystems.20.1.1

curve rising to three steps. This shows a change in the expected size of the basin of attraction from the minimum of 253 at a standardized score of −2 to just over 1000 at a standardized score of −4 and not much change for lower values.

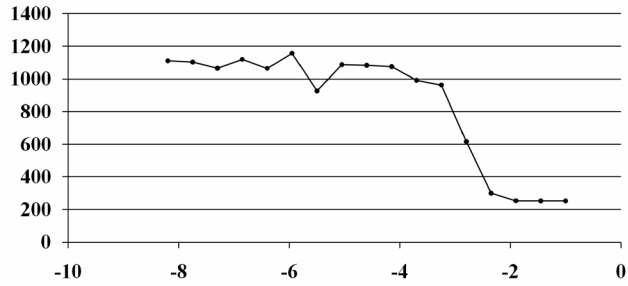

Figure 10. Plot of the simulated size of basins of attraction for standard normal distribution samples. Each point is the average of 40 samples.

The average local optima for our example in Figure 7 was 1 010 000 with deviation 173 700, which translates to −6.6 when standardized. The absolute minimum was at 645 192, which translates to −8.0. There is not much difference in the expected size of the basins of attraction for these values in Figure 10.

7.2 Simulation for Expected Number of Neighbors in the Same Basin of Attraction

A simpler simulation was performed that shows a similar effect. We looked at a local optima to determine how many of its immediate neighbors are in its basin of attraction. This simulation can be run more quickly by allocating 254 random scores and picking the smallest to be the local minima. Figure 11 shows the result having the same change between −2 and −4 but with much less deviation.

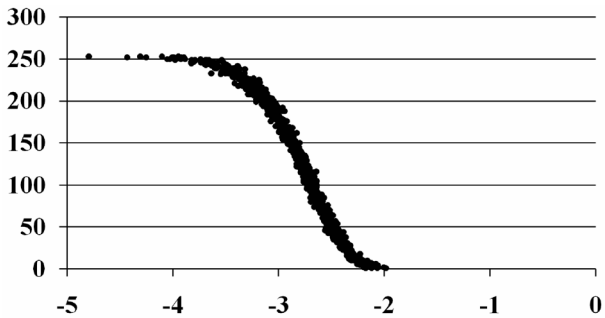

Figure 11. Plot of a simulation of 1000 samples. The number of immediate neighbors in the basin of attraction of an allocation against the allocation cost function value.

https://doi.org/10.25088/ComplexSystems.20.1.1

7.3 Good Neighbor Assumption

We can introduce the common assumption that the neighbors of the optimum solution are drawn from an atypical population. It is not unreasonable to assume that this atypical population is also the source of other local optima. We know the mean and deviation for local optima (1 010 000 and 173 700 respectively). If we use this for the distribution of the neighbors for a good solution then the optimum 645 192 will translate to −2.1 when standardized. This is just the point where the graphs in Figures 10 and 11 indicate a major change in the expected size of the basin of attraction. Figures 12 and 13 show the normalized versions of the data from the local search.

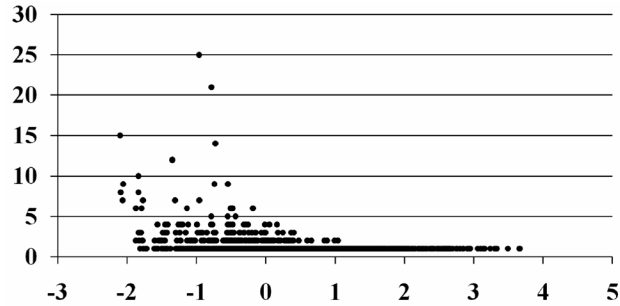

Figure 12. Figure 7 normalized to the distribution of local optima. The number of random allocations that optimized to the same local minima against the cost function value of the local minima generated by the local search algorithm.

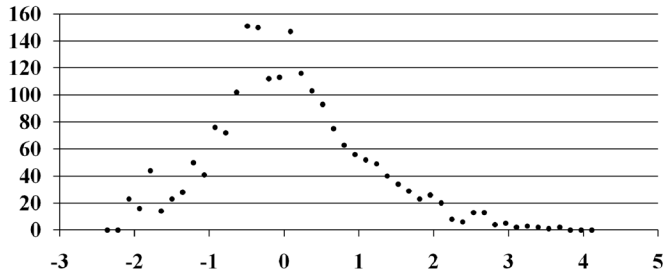

Figure 13. Figure 8 normalized to the distribution of local optima. The number of local minima against the cost function value generated by the local search algorithm. The y axis is the count of minima in bands 25 000 wide of the cost units.

The expected size of the basins of attraction increases by a factor of four over the range where we see no small basins of attraction in the data. It is also noted that only one in 10 local optima have scores

https://doi.org/10.25088/ComplexSystems.20.1.1

below −1 in the normalized data which is in the neighborhood of the best solutions. This is probably only part of the story as the total number of optima is very large and the number of steps to reach the optima is very different.

7.4 Time to Discovery

Figure 14 shows a plot of the number of the first trial to discover each local optima found in a series of 10 000 trials. It clearly shows two groups of optima on the left. The best group of four were all discovered within 658 trails and the second group of 19 were discovered in 5522 trials. In both cases additional members of these groups would be very unlikely.

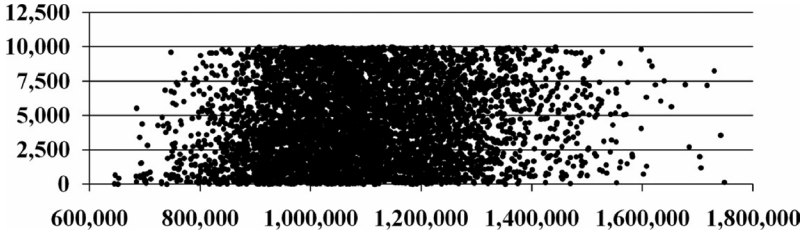

Figure 14. The number of trials it took to discover each local optima using the local neighborhood. The total number of trials was 10 000.

8. Conclusion

8.1 The Game of Life Simple Universal Turing Machine

The optimum ordered description of the unary multiplication Turing machine took 152 symbols on the simple universal Turing machine tape, which works out as an average of 6.6 over the 23 transitions. Each transition requires two separator symbols and two symbols for the symbol to write and the move direction. This leaves an average of just 1.3 symbols for each of the two links to the next transitions. By way of comparison, the worst possible ordering produces a description requiring 439 symbols.

Coding the Turing machine using a list of quintuplets as shown in Table 6 requires 30 quintuples. If it were possible to code each quintuple with just five symbols this comes to 150 symbols before adding any formatting symbols. This demonstrates that the GoL-UTM [2] can have a very compact description of a Turing machine with a little care over the ordering of the transitions.

The unary multiplication Turing machine takes 443 cycles to multiply 4×4. The GoL-UTM took 56 561 cycles to perform the same calculation. That is just less than 128 cycles of the GoL-UTM per cy-

https://doi.org/10.25088/ComplexSystems.20.1.1

cle of the Turing machine. The Turing machine's tape was 31 symbols long, giving a total tape length of 183 for the GoL-UTM. This shows how effective the optimization was for speed as the average cycle is less than 70% of the average distance the GoL-UTM read/write head has to move before taking into account changing state. This demonstrates the speed of the GoL-UTM [2].

Figure 15 shows a size comparison of the Game of Life Turing machine, the universal Turing machine with the small doubling program, and the universal Turing machine with the unary multiplication program.

8.2 The Quadratic Assignment Problem Solution

This example quadratic assignment problem of size 23 was solved with 2000 samples taking half an hour on a modern laptop. The optimum solution was found in the first 200 samples and 1000 samples would have been sufficient to be confident that it was indeed the optimum.

The comparison between a simple local search and the greedy search in Section 6.4 showed that, for this problem, the greedy search requires twice as many samples to give a similar level of hits on the optima with large basins of attraction, and thus the same level of confidence that the true optima has been found. However, the greedy search is much quicker, with the 2000 samples for Figure 9 taking just less than 7 minutes.

For this problem there were no local optima with low scores and small basins of attraction creating a void in the bottom left of Figure 7 and the top left of Figure 14. These voids generate confidence that the true optima has been found.

The group of local minima with large basins of attraction and low cost function values that includes the optimum local minima may be a general feature of quadratic assignment problems. It is reasonable that the best solution should have the most neighbors with good scores and thus have a large basin of attraction. This may have some relationship with the "big valley" structure found by Boese [15]. However, our analysis does not attempt to show how close together these deep optima are in the search space. We do find however that the basins of attraction of the optimum allocation for some smaller problems is large enough that random allocations can locate the optima with a simple local search.

The quadratic assignment problem has been described as one of the most difficult problems in the NP-hard class [13]. This work shows that Moore's law has caught up with some of the more modest examples of this class of problem and that they can now be solved convincingly with simple methods.

https://doi.org/10.25088/ComplexSystems.20.1.1

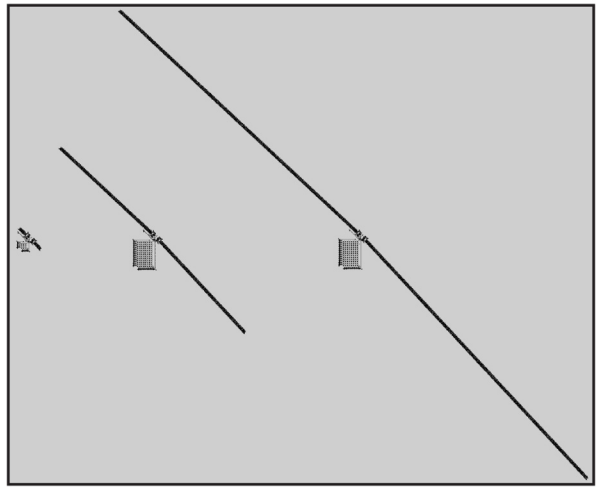

Figure 15. Size comparison of the Game of Life Turing machine, the universal Turing machine with the small doubling program, and the universal Turing machine with the unary multiplication program.

References

[1] P. Rendell, "Turing Universality of the Game of Life," in *Collision-Based Computing*, London: Springer-Verlag, 2001 pp. 513–539.

[2] P. Rendell, "A Simple Universal Turing Machine for the Game of Life Turing Machine," in *Game of Life Cellular Automata*, London: Springer-Verlag, 2010 pp. 519–545.

[3] S. Wolfram, *A New Kind of Science*, Champaign, IL: Wolfram Media, Inc., 2002.

[4] E. R. Berlekamp, J. H. Conway, and R. K. Guy, *Winning Ways for your Mathematical Plays*, Vol. 1–3, 2nd ed., Natick, MA: A K Peters, Ltd., 2001–2004.

[5] A. M. Turing, "On Computable Numbers, with an Application to the Entscheidungsproblem," *Proceedings of the London Mathematical Society*, **42**, 1937 pp. 203–265. doi:10.1112/plms/s2-42.1.230.

[6] A. Smith. "Universality of Wolfram's 2, 3 Turing Machine." (Oct 24, 2007) http://www.wolframscience.com/prizes/tm23/TM23Proof.pdf.

[7] Y. Rogozhin, "Small Universal Turing Machines," *Theoretical Computer Science*, **168**, 1996 pp. 215–240.

[8] T. Neary and D. Woods, "Small Fast Universal Turing Machines," *Theoretical Computer Science*, **362**(1), 2006 pp. 171–195.

[9] M. L. Minsky, *Computation: Finite and Infinite Machines*, Englewood Cliffs, NJ: Prentice-Hall, 1967.

https://doi.org/10.25088/ComplexSystems.20.1.1

[10] A. Trevorrow, T. Rokicki, T. Hutton, D. Greene, J. Summers, and M. Verver. "Golly, a Game of Life Simulator." (June 22, 2011) http://golly.sourceforge.net.

[11] Wikipedia. "Hashlife." (June 29, 2011) http://en.wikipedia.org/wiki/Hashlife.

[12] T. C. Koopmans and M. Beckmann, "Assignment Problems and the Location of Economic Activities," *Econometrica*, **25**(1), 1957. http://cowles.econ.yale.edu/P/cp/p01a/p0108.pdf.

[13] E. M. Loiola, N. M. M. Abreu, P. O. Boaventura-Netto, P. Hahn, and T. Querido, "An Analytical Survey for the Quadratic Assignment Problem," *European Journal of Operational Research*, **176**(2), 2007 pp. 657–690.

[14] T. James, C. Rego, and F. Glover, "Multistart Tabu Search and Diversification Strategies for the Quadratic Assignment Problem," *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans*, **39**(3), 2009 pp. 579–596. doi:10.1109/TSMCA.2009.2014556.

[15] K. D. Boese, A. B. Kahng, and S. Muddu, "A New Adaptive Multi-start Technique for Combinatorial Global Optimizations," *Operations Research Letters*, **16**(2), 1994 pp. 101–113. doi:10.1016/0167-6377(94)90065-5.

[16] Colin R. Reeves, "Landscapes, Operators and Heuristic Search," *Annals of Operations Research*, **86**(0), 1999 pp. 473–490. doi:10.1023/A:1018983524911.

[17] P. Merz and B. Freisleben, "Fitness Landscape Analysis and Memetic Algorithms for the Quadratic Assignment Problem," *IEEE Transactions on Evolutionary Computation*, **4**(4), 2000 pp. 337–352. doi:10.1109/4235.887234.

https://doi.org/10.25088/ComplexSystems.20.1.1