Construction of Covering Arrays (CA) Using Simulated Annealing (SA)

Dinesh Kumar Karri (G01103757)

Introduction

Search problems increase in the complexity with the amount of assigned search space. These problems are very where and needs to be solved. BFS, DFS, A* search algorithms are widely prevalent to solve search problems where the search is carried out one node at a time. But there is a completely different class of search problems that exists which the above-mentioned techniques cannot be used. These problems are called Combinatorial Search problems. In these problems given a set of parameters, we want the find all the possible combinations for these parameters.

The complexity of these kind of search problems increases with the search space for the parameters. For example, we want to test a never-before-seen robot and we are tasked to test the motion functionalities of the robot. There are 2 actuators on the robot which are responsible for the motion. We are tasked to create a glossary of how different combination of actuator inputs to fully study the robot motion as given in the table below. For simplicity's sake, each actuator can only take two inputs ON and OFF which can be represented in binary as 1 and 0. This is a simple enough problem that we can make a list of all possible combinations. The combinations will be (A1, A2) belongs to the set {(0,0), (0,1), (1,0), (1,1)}. We test all possible combinations and understand how the robot works. But let suppose there are *k* actuators, and each actuator can take *v* values. This becomes a complex problem as the range of *k* and *v* are large. To verify if all possible combinations of these actuator input scheme are tested, we can use something called a Covering Array (CA).

A CA is mathematical object that is often used for testing purposes. For problems with many possible configurations, each configuration can give a different output or if we are doing fault detection, at least one of the combinations could be faulty. We can always test all possible configurations, but this is highly intractable as there will be too many configurations to test. In this case we can use Covering Arrays (CA). A CA is used to quantify how well a set of sample configurations covers the set of all configurations.

But making a covering array is also a tedious task. To find the optimal CA that covers all possible configurations we can use optimization algorithms like Simulated Annealing (SA) which is a modified version of hill climbing algorithm, Genetic Algorithms (GA), Evolutionary Algorithms (EA) etc. which will give a CA which is optimal or near optimal. The objective for use these algorithms is to avoid being stuck in local minima and get a near optimal solution dependent on the proposed cost function.

In this project we will use a SA to obtain an optimal CA for given set of parameters. We will also perform experiments to show our results and give our observations on it.

Background

A CA is represented as CA(N;k,t,v) where *k* is the number of parameters we are testing so it gives the number of columns in the array, *t* is the strength i.e. number of parameters we are testing at once which gives number of columns we pick at once, *v* is the number of symbols/values each parameter can take and N is the number of rows in the array. So technically the order of the CA matrix is Nx*k*. N is found by the equation $N = v^t$. N can also be defined as the minimum number of rows required to achieve all possible combinations $\binom{k}{t}$ in the covering array.

A neighborhood function gives all possible sets of neighbors for the current state from which we pick one of these neighboring states which gets us closer to the solution just like in search tree. Then we place the chosen neighboring state as the current state and repeat this process until we reach the solution.

An objective function is the function that determines quantitatively how far or close we are to the optimal solution. This is used to choose the next state using the neighborhood function.

In Simulated Annealing SA, as like in the physical annealing process, there is a temperature which is called synthetic temperature T. First, we must have an objective function and decide whether it's a minimization or a maximization problem. Then, in this process we pick random states in the domain of the states as the children/neighbor for the current state. If the random state pick improves the current state, we choose the random state as the current state. Is the random state does not improve the current state, then we accept the random state with some probability less than 1. The probability of the next state is determined by the equation $\rho = e^{-\Delta E/T}$. If ρ is less than the random probability between (0,1) on a normal distribution, then the new state is accepted. As we can see from the equation of ρ, as T decreases and $\Delta E > 0$ then the probability ρ tends to 0. So, the smaller the value of T the lesser the value of ρ. When T is higher, we accept not just he good moves but also some bad moves. With this method, the objective function will reach the global optimal value and the corresponding global optimal state is reached.

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  inputs: problem, a problem
          schedule, a mapping from time to "temperature"

  current ← MAKE-NODE(problem.INITIAL-STATE)
  for $t$ = 1 to $\infty$ do
      $T$ ← schedule($t$)
      if $T$ = 0 then return current
      next ← a randomly selected successor of current
      $\Delta E$ ← next.VALUE – current.VALUE
      if $\Delta E > 0$ then current ← next
      else current ← next only with probability $e^{\Delta E/T}$
```
Algorithm 1 - The algorithm for simulated annealing is as given above.

Problem

In this project, we will implement simulated annealing (SA) to try to build a CA s.t. the number of rows (N) is the lower bound size that tentative can cover the required combinations in each N × t sub-array. To simplify the problem, for this implementation, fixed assignments of t=2 and v=2 will be used, that is, only the number of parameters (k) that will be the input varies, being its domain [5,7]

Proposed Approach

1. First we find the N for the covering CA(N;k,t,v) using $N = v^t$.
2. We initialize a candidate solution as a matrix of order Nxk and each element in the matrix is initialized with a random value from possible value/symbol *v* can take in the domain of *v*. This is taken as the current state.
3. Then an objective function is created which will give the total number of missing combinations in the current state and set it up as a minimization problem.
4. Then we initialize the temperature T=k as the starting temperature.
5. We use a geometric cooling scheme, which is decremented at each round by a factor α=0.99 using the relation $T_x = \alpha T_{x-1}$. This will determine the number of searches done during the period of a single run.
6. We create a $\Delta E =$ Obj_func(current state) - Obj_func(next possible state)
7. We create a neighborhood function for the states such that function randomly selects a column j($1 \leq j \leq k$). After that, the function swaps the symbol of each row in the matrix $M_{i,j}$ to create N neighbors (the swap of a cell is a neighbor) then evaluates the objective function i.e. the number of missing combinations in each of them. We pick the neighbor with the lowest objective function and make it the next possible state. If there is an improvement in the objective function that is if $\Delta E > 0$ then we accept the state as current state. But if $\Delta E < 0$, then we use the probability ρ as described in the background section.
8. As T decreases and if step 7 fails to give us a neighbor we repeat step 7 with a new neighbor set.
9. If the objective function reaches a steady state i.e., if the value of the objective does not change for several runs we want to stop the run. For this we use something called the frozen factor ϕ. if after ϕ (frozen factor) consecutive temperature decrements the best-so-far solution is not improved, the algorithm stops its execution, for this $\Phi = v^t \times \binom{k}{t}$. If the frozen factor is not satisfied, the simulations keep going.
10. Then we retrieve the current state which is obtained after several mutations and is near optimal or optimal.

Experimental Results

We run experiments with the above proposed approach to find CA(N;k,t,v) for given k in domain [5,7], t =2, v in domain [0,1]. We perform 30 experimental runs for each value of k and provide results for the runs in this section.

Creating covering array using simulated annealing for k = 5, t = 2, v = 2

Table given below consists of the results obtained from simulated annealing (SA) process to make a covering array (CA); CA(N;k,t,v), where k = 5, t = 2, v = 2. To find the lower bound of the number of rows N required in the array that will have all possible combinations we do –

$$N = v^t$$

So, in this case N = 4.

Table 1. The table consists of CA created for 30 separate runs and the statistics for each run. The following convention was followed for this table. M(0) is the initial state, M(t) is the final state, O(0) is the initial objective function, O(t) is the final objective function, Frozen Factor Value (FFV), Difference between O(0) and O(t) is denoted by ∆O, Frozen Factor stopping condition achieved or not is represented as FFSC, Number of times the loop was executed in a single run (LC).

No.	Initial State of M(0)	Final State of M(t)	O(0)	O(t)	∆O	FFV	FFSC	LC
0	$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \end{bmatrix}$	9	6	3	6	Yes	238
1	$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix}$	17	6	11	6	Yes	223
2	$\begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix}$	19	6	11	6	Yes	207
3	$\begin{bmatrix} 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix}$	13	6	7	6	Yes	174
4	$\begin{bmatrix} 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{bmatrix}$	6	6	0	6	Yes	241

5	$\begin{bmatrix}1&1&1&0&0\\0&1&1&1&1\\1&1&1&1&0\\0&0&1&0&1\end{bmatrix}$	$\begin{bmatrix}1&1&0&0&0\\0&0&0&1&0\\1&1&1&1&1\\0&0&1&0&1\end{bmatrix}$	13	6	7	6	Yes	295
6	$\begin{bmatrix}1&1&1&0&0\\0&0&1&0&1\\1&1&1&1&1\\0&0&1&1&0\end{bmatrix}$	$\begin{bmatrix}0&0&0&0&1\\1&1&1&0&1\\1&0&0&1&0\\0&1&1&1&0\end{bmatrix}$	10	6	4	6	Yes	216
7	$\begin{bmatrix}0&1&0&1&0\\1&0&0&1&0\\1&1&1&1&0\\1&1&0&1&0\end{bmatrix}$	$\begin{bmatrix}0&0&1&1&0\\0&0&0&0&1\\1&1&0&1&1\\1&1&1&0&0\end{bmatrix}$	19	6	13	6	Yes	177
8	$\begin{bmatrix}1&0&0&0&1\\0&0&0&0&0\\0&1&1&1&1\\0&0&0&0&1\end{bmatrix}$	$\begin{bmatrix}1&0&0&1&1\\1&1&1&1&0\\0&0&1&0&0\\0&1&0&0&1\end{bmatrix}$	13	6	7	6	Yes	281
9	$\begin{bmatrix}1&1&1&0&1\\0&1&0&1&0\\0&1&1&0&0\\0&1&0&0&1\end{bmatrix}$	$\begin{bmatrix}1&1&0&0&1\\1&1&1&1&0\\0&0&1&1&1\\0&0&0&0&0\end{bmatrix}$	13	6	7	6	Yes	124
10	$\begin{bmatrix}0&1&1&1&1\\1&1&0&1&0\\0&1&1&1&1\\1&0&1&0&1\end{bmatrix}$	$\begin{bmatrix}0&1&0&0&1\\0&1&1&1&0\\1&0&1&1&1\\1&0&0&0&0\end{bmatrix}$	12	6	6	6	Yes	298
11	$\begin{bmatrix}0&0&0&0&1\\0&0&1&0&0\\0&1&0&1&0\\1&1&0&1&0\end{bmatrix}$	$\begin{bmatrix}1&0&0&1&1\\0&0&1&1&0\\1&1&1&0&1\\0&1&0&0&0\end{bmatrix}$	11	6	5	6	Yes	205
12	$\begin{bmatrix}0&1&1&1&0\\0&1&1&1&0\\0&0&1&0&0\\1&0&0&0&1\end{bmatrix}$	$\begin{bmatrix}0&0&0&0&1\\1&0&0&1&0\\0&1&1&1&0\\1&1&1&0&1\end{bmatrix}$	14	6	8	6	Yes	245
13	$\begin{bmatrix}0&0&0&1&0\\1&0&0&1&0\\1&1&0&1&0\\0&0&1&0&0\end{bmatrix}$	$\begin{bmatrix}1&0&1&0&1\\0&0&0&1&1\\1&1&1&1&0\\0&1&0&0&0\end{bmatrix}$	15	6	9	6	Yes	134
14	$\begin{bmatrix}1&0&1&0&0\\1&0&0&1&1\\1&0&0&1&1\\0&1&0&1&1\end{bmatrix}$	$\begin{bmatrix}1&1&0&0&1\\1&1&1&1&0\\0&0&0&0&0\\0&0&1&1&1\end{bmatrix}$	14	6	8	6	Yes	220
15	$\begin{bmatrix}1&0&0&0&0\\0&0&0&1&1\\0&1&1&0&1\\1&0&0&1&0\end{bmatrix}$	$\begin{bmatrix}0&0&1&1&0\\1&0&0&1&1\\0&1&0&0&1\\1&1&1&0&0\end{bmatrix}$	9	6	3	6	Yes	189

16	$\begin{bmatrix}1&1&0&1&0\\0&0&0&0&0\\0&1&0&1&1\\1&0&0&0&1\end{bmatrix}$	$\begin{bmatrix}1&1&1&1&0\\0&0&1&1&1\\0&1&0&0&0\\1&0&0&0&1\end{bmatrix}$	10	6	4	6	Yes	195
17	$\begin{bmatrix}1&0&1&1&0\\1&1&1&1&1\\1&0&1&1&1\\0&1&1&1&0\end{bmatrix}$	$\begin{bmatrix}0&1&1&0&1\\0&0&0&1&0\\1&1&0&1&1\\1&0&1&0&0\end{bmatrix}$	17	6	11	6	Yes	495
18	$\begin{bmatrix}0&1&1&0&0\\0&0&0&0&0\\0&0&0&1&0\\0&1&0&0&1\end{bmatrix}$	$\begin{bmatrix}1&0&0&1&0\\1&0&1&0&1\\0&1&1&0&0\\0&1&0&1&1\end{bmatrix}$	14	6	8	6	Yes	252
19	$\begin{bmatrix}1&0&0&0&1\\0&0&0&1&1\\1&1&1&1&1\\0&1&1&1&1\end{bmatrix}$	$\begin{bmatrix}0&0&0&1&0\\1&0&0&0&1\\1&1&1&1&0\\0&1&1&0&1\end{bmatrix}$	13	6	7	6	Yes	170
20	$\begin{bmatrix}1&0&1&0&0\\1&1&0&0&0\\1&0&0&1&0\\0&1&0&0&0\end{bmatrix}$	$\begin{bmatrix}0&1&0&1&0\\1&1&1&1&1\\0&0&0&0&1\\1&0&1&0&0\end{bmatrix}$	14	6	8	6	Yes	196
21	$\begin{bmatrix}1&1&0&1&0\\1&0&1&1&1\\1&0&1&1&1\\1&0&0&0&0\end{bmatrix}$	$\begin{bmatrix}0&1&0&0&0\\0&1&1&1&1\\1&0&1&1&0\\1&0&0&0&1\end{bmatrix}$	15	6	9	6	Yes	242
22	$\begin{bmatrix}0&1&0&1&0\\0&1&1&0&0\\1&0&0&0&0\\1&1&0&1&0\end{bmatrix}$	$\begin{bmatrix}0&0&0&0&1\\0&1&0&1&0\\1&0&1&0&0\\1&1&1&1&1\end{bmatrix}$	13	6	7	6	Yes	179
23	$\begin{bmatrix}0&0&0&1&0\\1&1&1&0&1\\1&1&1&0&1\\1&0&1&1&1\end{bmatrix}$	$\begin{bmatrix}0&0&1&0&1\\1&1&0&0&1\\0&0&0&1&0\\1&1&1&1&0\end{bmatrix}$	14	6	8	6	Yes	158
24	$\begin{bmatrix}1&0&1&0&0\\1&1&0&1&1\\1&0&0&1&1\\0&1&0&0&0\end{bmatrix}$	$\begin{bmatrix}0&1&0&0&1\\1&1&1&1&1\\1&0&0&0&0\\0&0&1&1&0\end{bmatrix}$	9	6	3	6	Yes	173
25	$\begin{bmatrix}0&0&1&0&1\\1&1&1&1&0\\0&1&1&1&1\\0&1&0&0&0\end{bmatrix}$	$\begin{bmatrix}0&1&1&1&1\\1&1&0&1&0\\1&0&1&0&1\\0&0&0&0&0\end{bmatrix}$	9	6	3	6	Yes	220
26	$\begin{bmatrix}1&1&0&0&1\\0&0&0&0&1\\0&1&1&0&1\\1&0&1&0&1\end{bmatrix}$	$\begin{bmatrix}0&0&0&1&0\\1&0&1&0&0\\0&1&1&0&1\\1&1&0&1&1\end{bmatrix}$	15	6	9	6	Yes	245

27	$\begin{bmatrix}0 & 1 & 0 & 1 & 0\\1 & 0 & 1 & 0 & 0\\0 & 0 & 0 & 0 & 1\\1 & 0 & 0 & 0 & 0\end{bmatrix}$	$\begin{bmatrix}1 & 0 & 1 & 1 & 1\\1 & 1 & 1 & 0 & 0\\0 & 1 & 0 & 0 & 1\\0 & 0 & 0 & 1 & 0\end{bmatrix}$	11	6	5	6	Yes	210
28	$\begin{bmatrix}1 & 1 & 0 & 0 & 1\\1 & 1 & 0 & 1 & 1\\0 & 0 & 0 & 0 & 1\\0 & 1 & 0 & 1 & 1\end{bmatrix}$	$\begin{bmatrix}1 & 1 & 0 & 0 & 1\\1 & 0 & 1 & 1 & 0\\0 & 1 & 1 & 0 & 0\\0 & 0 & 0 & 1 & 1\end{bmatrix}$	17	6	11	6	Yes	179
29	$\begin{bmatrix}0 & 1 & 1 & 0 & 0\\1 & 1 & 0 & 1 & 0\\0 & 0 & 1 & 1 & 0\\1 & 1 & 1 & 1 & 0\end{bmatrix}$	$\begin{bmatrix}0 & 0 & 0 & 0 & 1\\0 & 1 & 0 & 1 & 0\\1 & 0 & 1 & 1 & 0\\1 & 1 & 1 & 0 & 1\end{bmatrix}$	14	6	8	6	Yes	330

In this table, we have the initial state matrix and final state matrix. This will only be represented in this table to show the change in the state matrix. From next table these will not be presented in order to save some space.

From the results in this table we can see that the initial Objective Function O(0) gives the number of missing combinations in the initial state matrix M(0) and is different for different runs as for each of the 30 runs presented, M(0) is initialized randomly with each element in the matrix takes either the value 0 or 1 since v = 2. The final state matrix M(t) has final Objective Function O(t) of 6 for all the runs which before hitting the frozen factor condition. This can be interpreted that for the given parameters of the covering there will be at least 6 missing combinations. Note that this conclusion was made from the limited data of 30 runs. To conclusively we can perform Monte Carlo simulations and the result might change. In all the simulations we hit the frozen factor condition which stops the simulations assuming it is the global optimal solution for the covering array. Thus for given parameters the presented final state matrices for 30 runs are the optimal covering arrays for their respective initial states. Experiments were also performed for different k = 6 and k = 7 with N, v, and t remaining the same to see the effects of SA to make CA.

Creating covering array using simulated annealing for k = 6, t = 2, v = 2

Here, the results for 30 runs for simulated annealing process to make a covering array CA(N;k,t,v) for k = 6, t = 2, v = 2 are given.

Table 2. The table consists of CA created for 30 separate runs and the statistics for each run. The following convention was followed for this table. O(0) is the initial state's objective function, O(t) is the final state's objective function, Frozen Factor Value (FFV), Difference between O(0) and O(t) is denoted by ∆O, Frozen Factor stopping condition achieved or not is represented as FFSC, Number of times the loop was executed in a single run (LC).

No.	O(0)	O(t)	∆O	Frozen Condition Stop Condition	Number of loop cycles before stop

0	17	9	8	Yes	128
1	19	9	10	Yes	205
2	26	9	17	Yes	228
3	17	9	8	Yes	188
4	12	9	3	Yes	176
5	13	9	4	Yes	177
6	15	9	6	Yes	188
7	21	9	12	Yes	170
8	21	9	12	Yes	213
9	14	9	5	Yes	147
10	19	9	10	Yes	157
11	15	9	6	Yes	187
12	22	9	13	Yes	166
13	21	9	12	Yes	182
14	19	9	10	Yes	146
15	15	9	6	Yes	226
16	11	9	2	Yes	214
17	17	9	8	Yes	229
18	21	9	12	Yes	186
19	24	9	15	Yes	156
20	21	9	12	Yes	199
21	20	9	11	Yes	160
22	17	9	8	Yes	221
23	17	9	8	Yes	176
24	19	9	10	Yes	221
25	24	9	15	Yes	185
26	15	9	6	Yes	165
27	17	9	8	Yes	154
28	14	9	5	Yes	162
29	25	9	16	Yes	131

Similar to the previous experiment with k = 5, all the experiments for k = 6 reach the frozen factor stopping condition and the total missing combinations at the end for SA for all the runs are 9 with different initial states and respective initial objective functions. The results for all the 30 runs are presented in the table above.

Creating covering array using simulated annealing for k = 7, t = 2, v = 2

Here, the results for 30 runs for simulated annealing process to make a covering array CA(N;k,t,v) for k = 7, t = 2, v = 2 are given.

Table 3. The table consists of CA created for 30 separate runs and the statistics for each run. The following convention was followed for this table. O(0) is the initial state's objective function, O(t) is the final state's objective

function, Frozen Factor Value (FFV), Difference between O(0) and O(t) is denoted by ∆O, Frozen Factor stopping condition achieved or not is represented as (FFSC), Number of times the loop was executed in a single run (LC).

No.	O(0)	O(t)	∆O	Frozen Condition Stop Condition	Solution Reached?	Number of loop cycles before stop
0	22	15	7	Yes	Yes - FFSC	128
1	30	14	16	No	No - $O(t) \neq 0$	205
2	23	14	9	No	No - $O(t) \neq 0$	228
3	20	12	8	No	No - $O(t) \neq 0$	188
4	17	14	3	No	No - $O(t) \neq 0$	176
5	25	14	11	No	No - $O(t) \neq 0$	177
6	22	14	8	No	No - $O(t) \neq 0$	188
7	23	12	11	No	No - $O(t) \neq 0$	170
8	32	12	20	No	No - $O(t) \neq 0$	213
9	34	14	20	No	No - $O(t) \neq 0$	147
10	23	14	9	No	No - $O(t) \neq 0$	157
11	30	15	15	Yes	Yes - FFSC	187
12	31	12	19	No	No - $O(t) \neq 0$	166
13	23	12	11	No	No - $O(t) \neq 0$	182
14	30	12	18	No	No - $O(t) \neq 0$	146
15	40	12	28	No	No - $O(t) \neq 0$	226
16	35	14	21	No	No - $O(t) \neq 0$	214
17	26	14	12	No	No - $O(t) \neq 0$	229
18	17	14	3	No	No - $O(t) \neq 0$	186
19	29	14	15	No	No - $O(t) \neq 0$	156
20	40	12	28	No	No - $O(t) \neq 0$	199
21	19	12	7	No	No - $O(t) \neq 0$	160
22	34	14	20	No	No - $O(t) \neq 0$	221
23	26	12	14	No	No - $O(t) \neq 0$	176
24	27	12	15	No	No - $O(t) \neq 0$	221
25	25	14	11	No	No - $O(t) \neq 0$	185
26	32	12	20	No	No - $O(t) \neq 0$	165
27	30	14	16	No	No - $O(t) \neq 0$	154
28	29	12	17	No	No - $O(t) \neq 0$	162
29	27	14	13	No	No - $O(t) \neq 0$	131

From the table above, for k = 7, t = 2, and v = 2, we can see that we mostly did not reach the frozen factor condition or the solution where O(t) = 0, which means there are no missing combinations after the simulated annealing process is performed. That means we cannot say if we reached an optimal solution which may either require change in temperature or the change in N. This will be validated in the next section but for now the this is the conclusion for these set of experiments.

Observations

a) For CA with k = 5, t = 2, and v = 2:

From the results for the given CA parameters, we calculated that N = 4. But with this value of N, even though the frozen factor stopping condition is met, the number of missing combinations in the solution output is not 0. That means there is still something that can be done to get all possible combinations in the CA.

Firstly, we have tried increasing the Temperature. This did not affect the solution as there still exists the same number for missing combinations in the solution and frozen factor stopping condition is invoked. So next we tried different values of N in an incremental order i.e., after a set of experiments with current N, it is changed to N+1. As N increased, the number of missing combinations decreased. For the given parameters, when N = 7 we started observing that for most runs the number of missing combinations at the final state O(t) is going to 0. So, for us to reach the most optimal solution, for the given parameters, N must be higher. So, $N = v^t$ would not work for the given parameters. We will need find a new equation for N such that we can reach the global optima.

b) For CA with k = 6, t = 2, and v = 2:

Similar to part(a), using $N = v^t$ to find N did not give the optimal solution but getting stuck at local minima and invoking the frozen factor stopping condition. Changing the temperature to a higher value did not have any effect on the solution. But following the same method of changing in from part(a), we were able to minimize the number of missing combinations in the solution O(t) to 0 for almost all the runs for N = 8.

c) For CA with k = 7, t = 2, and v = 2:

We have observed the same behavior for these parameters as well where the default given function to find N is not giving solution. Here we do not event achieve local minima. There is no effect on the solution with change in temperature. The only possible change that we can make is to N. For N = 7, we have observed that O(t) reaches 0.

Overall observations:

- The value of alpha should always be 1>α>0. The higher the alpha is the faster we converge to the solution and more accurately.
- The neighborhood function given in the project description to select the column randomly flip each element in the column individually to create neighboring states is not just the

only way to mutate the current state, we can try mutation as in the evolutionary algorithm to get to the solution faster.

- To solve the problem of N we are having, if $N \in \{k, k+1, k+2\}$, for at least one of the values in for the proposed N, the covering array for all the 30 runs has a solution where the number of missing combinations for any random initial state after SA process is 0. This was found by performing several runs of 30 with different values of N as explained in this section's parts (a), (b), and (c).

Conclusion

Simulated annealing is a reliable process to find global optimal solution for combinatorial problems like solving Sudoku, Rubik's Cube and designing VLSI systems. One of the applications presented in this project report in the form of creating Covering Arrays CA(N;k,t,v). We have performed Simulated Annealing technique to find the global optimal solution where the covering array has all possible combinations given the parameters k, t, and v. Using these parameters, we decided on value of N using the equation $N = v^t$. We later concluded that this method to find the value of N does not give the optimal solution or gets stuck in local minima by performing several experimental simulations. Note that the equation given for N is only tested for parameters k = {5,6,7}, t=2 and v=2. So, we cannot conclude if the equation for N is not valid globally. But for the given parameters we proposed a different N such that the objective function reaches 0 as temperature T decreases. Further investigation is required for testing SA process to conclusively say global optimal solution will be obtained for changes in different parameters of the covering array.