CS 580: Introduction to Artificial Intelligence
Project 1: Using SA to Construct Covering Arrays

Due: Sun, Mar 5 at 11:59 pm ET

This Project is considered individual effort and the code of honor is applied when reviewing the implementation.

Introduction

A Covering Array (CA) is a combinatorial object, denoted by CA(N; t, k, v), which can be described as a matrix with $N \times k$ elements, such that every $N \times t$ subarray contains all possible combinations of v^t symbols at least once. N represents the rows of the matrix, k is the number of parameters, which have v possible values and t represents the strength or the degree of controlled interaction.

To illustrate the use of CAs suppose that we have a system with 3 parameters (k) each with 2 possible values (v) labeled as 0 and 1 respectively as shown in **Table 1**.

Label	OS	Browser	Database
0	Linux	Firefox	MySQL
1	Windows	Chrome	Oracle

Table 1. System with 3 parameters each with 2 possible values

Assume we want to construct a CA with t=2 from the System described above; thus, a matrix of at least 4 rows is needed to cover v^t combinations. A CA that fulfills this this requirement is shown in **Table 2**:

1	**1**	**0**
1	**0**	**1**
0	**1**	**1**
0	**0**	**0**

Table 2. CA (4;2,3,2)

In the example shown in **Table 2** the total of combinations between pairs of parameters is $\binom{k}{t} = \binom{3}{2} = 3$ being these {OS, Browser}, {OS, Database}, {Browser, Database}. Each parameter has 2 settings giving 4 possible combinations for each pair of them.

The definition of a CA implies that every $N \times t$ sub-array contains all possible combinations of v^t symbols at least once, based on this fact, for the tuple {O.S., Web browser} all the combinations {0,0}, {0,1}, {1,0}, {1,1} are covered, the same way for any pair of selected parameters.

Implementation

In this project, you 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 $N \times t$ 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].

The algorithm will return:

(1) The CA (if it could find it) or display that the solution was not achieved
(2) The number of iterations performed in the execution.
(3) What is the stop criterion: solution, frozen, final temperature archived

The implementation will include the following:

- The initial solution will be created randomly, that is, you will initialize the matrix with the corresponding domain of each variable.
- The objective function will count the total of missing combinations in each $N \times t$ subset, thus being a minimization problem.
- The algorithm will include a geometrical cooling scheme, it starts at an initial temperature T_i =k and which is decremented at each round by a factor α=0.99 using the relation $T_x = \alpha T_{x-1}$
- **Frozen factor**: if after ϕ (frozen factor) consecutive temperature decrements the best-so-far solution is not improved, the algorithm stops its execution, for this $\Phi = v^t \times \binom{k}{t}$.
- The objective function will count the number of combinations covered in each $N \times t$ subset, thus being a minimization problem.
- The neighborhood function randomly selects a column j(1≤j≤k). After that, the function swaps the symbol of each row in the matrix $M_{i,j}$ to create the neighbors, then evaluates the number of missing combinations in each of them.
- Propose a formula to determine *N* based on the number of parameters (*k*) and include it in your report.
- Propose a formula to determine Δ*E* and include it in your report.

Deliverables

The SA implementation must be coded with Python. You will carry out an experimentation for *k*=5, *k*=6 and *k*=7. Each value of k must be executed 30 times, for a total of 90 executions. It will then generate a pdf document (written with Calibri 12, single spaced) with at least the following sections:

- Introduction
- Background
- Proposed Approach
- Experimental Results
- Conclusions

Results must contain a table that summarizes the statistics of the 30 runs for each value of k, including the number of times the algorithm was successful and the stopping criteria.

Be sure to explain your conclusions if you think the proposed neighborhood function is appropriate and justify your answer.

Submission

Submit your solution as `P1_<username>.py`, and your report `P1_<username>.pdf`, where `<username>` is your Mason account. Once the link for submission is closed, we do not accept resubmissions, so it is the responsibility of the student to verify that both files are the correct ones and can be extracted.
Multiple submissions are allowed, and the last submission is the one it's scored.