Homework 2

Implementation of the AETG algorithm that generates combinatorial test suites.

AETG is a commercial tool that generates combinatorial test suites. Essentially, the AETG algorithm constructs a covering array, or mixed level covering array that is used to represent the test suite. A *covering array*, $CA_\lambda(N;t,k,v)$, is an N x *k* array. In every N x *t* subarray, each *t*-tuple occurs at least λ times. In our application, *t* is the *strength* of the coverage of interactions, *k* is the number of factors, and *v* is the number of levels. A *mixed level covering array*, $MCA_\lambda(N;t,k,(v_1,v_2,...v_k))$, is an N x *k* array. Let $\{i_1,...,i_t\} \subseteq \{1,...,k\}$, and consider the subarray of size N x *t* obtained by selecting columns $i_1,...,i_t$ of the MCA. There are $\prod_{i=1}^{t} v_i$ distinct *t*-tuples that could appear as rows, and an MCA requires that each appear at least once. This variant of the covering array provides the flexibility to represent test suites for systems in which components are not restricted to having the exact same number of parameters. (In the application to combinatorial testing, we treat only the case when λ = 1.)

This literature on AETG will help you to implement the algorithm:

[1] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton. The AETG system: an approach to testing based on combinatorial design. IEEE Transactions on Software Engineering (1997), 23(7):437-44.

[2] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton. Method and system for automatically generating efficient test cases for systems having interacting elements United States Patent, Number 5,542,043, 1996.

[3] D. M. Cohen, S. R. Dalal, J. Parelius, and G. C. Patton. The combinatorial design approach to automatic test generation. IEEE Software (1996), 13(5):83-8.

Additional notes:

1. The AETG algorithm will generate test suites for any strength. In this assignment, you are only required to implement the algorithm for 2-way and 3-way coverage.
2. For this assignment, you are not required to take strings as input. You may simply give every factor and every level a unique numerical identifier. For instance, the input in Table 1 may be represented as shown in Table 2:

Hardware	**Operating System**	**Network Connection**	**Memory**
PC	Windows XP	Dial-up	64MB
Laptop	Linux	DSL	128MB
PDA	Solaris	Cable	256MB

Table 1 - A component based system with four factors that each have three levels

f_0	f_1	f_2	f_3
0	3	6	9
1	4	7	10
2	5	8	11

Table 2 - A component based system with four factors that each have three levels. (Factor f_0 has levels 0, 1, 2; factor f_1 has levels 3, 4, 5; factor f_2 has levels 6, 7, 8; factor f_3 has levels 9, 10, 11).

3. You should provide sample input files with a description so that the grader can create a file to test your program, <u>or</u> prompt the user for the number of factors and levels.
4. You may implement the program in your choice of programming language.
5. Your program may end up using too much time or memory for larger inputs, so you will need to be careful in your implementation. The papers do not describe the exact implementation details for the exact data structures that they use. It is expected that you can effectively make decisions that lead to a fast and efficient implementation.
6. Since there is randomization in your algorithm, you will run your program 100 times for each input and report the average size and execution time for each input. You should also use 50 candidates as done in the AETG literature.
7. You will report the results of your algorithm for several inputs for which AETG has published results. Of course, it is possible that your results will slightly vary from their reported results since there is randomization in the algorithm. However, if your results are off by more than 10% for a couple of inputs, you likely have a bug in your program.
8. You should strive to implement an efficient solution. To receive credit for this assignment, your algorithm cannot run for longer than 20 minutes to find a solution.
9. You are only required to implement a solution for up to 3-way coverage. However, if you implement a solution for n-way coverage and report results for up to 5-way coverage, you will receive 25% extra credit on this assignment.

Submission

Part 1 – Implementation and results (95%)

1. **Code** – You should provide your code and if applicable, your makefile. Please zip the files before e-mailing them to our course e-mail address.
2. **Results** – your results should be close to those published in the AETG literature. You should provide output files for each input such that each output file contains the best covering array generated for that input. You will also complete the following table with the results from your program:

Input	Your best result	Your average result	Your worst result	Average execution time
10^{20}				
3^{13}				
2^{10}				
4^{40}				
3^{4}				
$10^1 9^1 8^1 7^1 6^1 5^1 4^1 3^1 2^1$				

3. **Grading -** No credit will be given for code that does not compile or if the size of the solutions are not within 25% of the results reported in the AETG literature. No credit will be given to code that takes longer than 20 minutes to run on any of the above inputs. Your code must be object-oriented and use well named methods and variables. There will be a 10% penalty if the code is not commented. You can receive 50% credit if you program fully works only for covering arrays, but not mixed-level covering arrays.

Part 1 – Written question (5%)
The AETG algorithm does not actually have a logarithmic guarantee on the size of the test suites. Explain why.