Gregory Taylor and Luke Perlowski

12/19/2018

Cyber Physical Systems Final Paper

Colonel Blotto Games in CPS: Translating Matlab Solver to C Solver

Introduction: A Colonel Blotto Game is a famous game in Game Theory that has two colonels using soldiers to try and win battles on a battlefield. The game poses a scenario that each of the two generals has a certain number of soldiers, and wants to win more battles than their opponent. The puzzle is to figure out how to play the situation optimally if the way to win a battle is to solely have more soldiers at a battle than the other colonel. There are many different intricacies in this game such as what if the colonels value certain battles more than others due to their position on the battlefield or what if one colonel has less soldiers to play with than the other.

The Colonel Blotto Game can have applications in a number of different technical fields and scenarios, but perhaps the most interesting is how it affects Cyber Physical Systems (CPS). Cyber Physical Systems are series of physical sensors and actuators that communicate with and are controlled by corresponding cyber systems. These physical sensors or actuators can be targets for attackers who want to cause harm to a system or the surrounding environment. These attackers can launch attacks on the cyber components of these systems to manipulate and control the physical components.

This can potentially lead to serious harm, as it almost did in the STUXNET attack. The STUXNET attack targeted nuclear facilities in Iran to slow down their nuclear development program. Failing nuclear reactors have the potential to cause serious damage to an entire city.

This demonstrates the potential damage that can be done in the physical world through attacks in the cyber world.

The Colonel Blotto Game can be used to help conceptualize one of these attacks, where the cyber nodes or components are battles that correspond to several physical nodes which can be thought of as the battlefields. The resources that the system attacker and the system defenders have can potentially be computing power or knowledge of their opponent's strategies. Defenders and attackers can value physical nodes differently based on the perceived harm associated with a successful attack on that physical node. This is one potential way that the Colonel Blotto Game can be used to conceptualize a real world scenario in CPS.

We can use game theory and computer code to try and gain a better understanding of the Colonel Blotto Game and its CPS applications. Using the MatLab code provided to us, we attempted to recreate it in C, which would speed up its execution and allow for its use as an analytical tool for this purpose.

The matlab code is split into four main pieces: ChangingresourcesMain.m, resourcecombos.m, Gamebuild.m, and npg2.m. ChangingresourcesMain, as the name implies, is the main function of this code. Here, the initial parameters for the Colonel Blotto game are set. These include the number of cyber nodes, the connections between the cyber and physical nodes, the resources available to each player, the costs associated with each physical node for each player, and the number of cyber nodes required to take down the physical nodes. Once the parameters are set, the main function calls resourcecombos and passes on the number of cyber nodes and the resources available for the players as parameters. Resourcecombos then takes those parameters, enumerates every possible distribution of the given resources over the given number of nodes, and returns those distributions back to the main function. Next, the main function calls Gamebuild, passing it the number of cyber nodes, the resource distributions from resourcecombos, the connection matrices between cyber and physical nodes, the physical node costs, and the required number of cyber nodes to take down the physical nodes. Gamebuild takes all of this information and compares the players' strategy sets to create a payoff matrix for the entire game. This matrix is then returned to the main function, and from there is passed into npg2, which is a Nash Equilibrium solver. Npg2 uses the input parameters to solve for the first Nash Equilibrium that it finds, meaning that games with multiple Nash equilibria will only be analyzed based on the first one found. Finally, the main function then is used to plot relevant data in matlab figures.

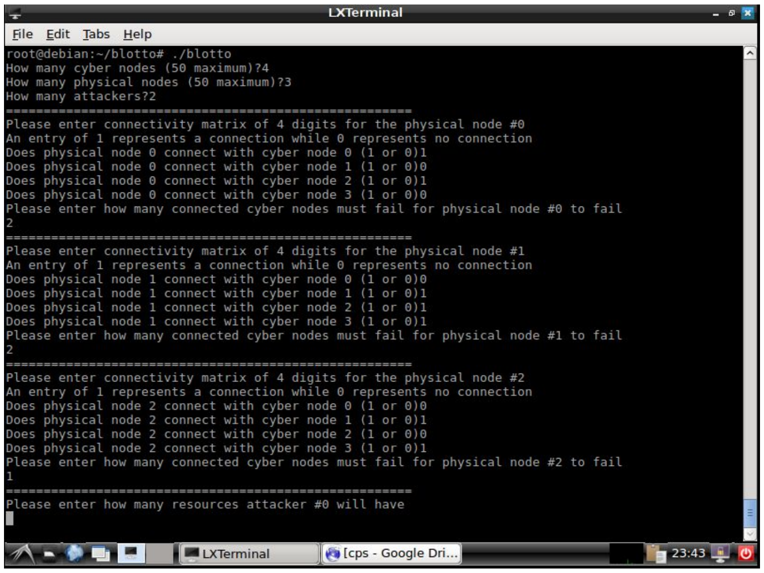
Our intention in converting this to C code was to preserve as much of the structure as we could. However, we wanted to code the project in such a way that all parameters could be modified. In the matlab version of this code many parameters can be changed in the source code, but the way that part of the code is structured does not allow a user to change the number of physical nodes very easily, or change the number or attackers and defenders. In our main function, we created structures to represent defenders, attackers, and physical nodes. The player structures include the number of resources available to each specific player and their personal payoff distribution. The physical node structures include the number of hitpoints, or cyber nodes required to take the physical node down, as well as the cyber node connection matrix for the specific physical node. The main function prompts the user for the same input parameters as the matlab code, plus the number of attackers, defenders, and physical nodes. Once the user has entered this information, the attacker and physical node structures are then dynamically allocated.

Similarly to the matlab code, the main function now needs the strategy sets for each player, represented by the resource combinations. Aside from allowing a variable number of players and physical nodes, the resource combination code was the main focus of this project. In the matlab code, it runs very slowly. Our goal was to rewrite it in C to improve the speed at which it enumerates all of the resource combinations. After examining the algorithms used for the combos in matlab, it was clear that all of the library functions would make it difficult to implement in C if we tried to use the exact same structure. There is no equivalent to the matlab nchoosek function in C, which creates matrices containing all k length combinations of the values in matrix n. There is also no equivalent to the diff function, which calculates the difference between adjacent elements in a matrix. Both of the functions were integral in the matlab version of resourcecombos, so we decided to take a different approach. Iteration and recursive functions are much faster in C than in matlab, so we created a recursive function to enumerate all resource combinations given the number of cyber nodes and resources. Distribution of r resources over n different nodes is the same as trying to distribute r indistinguishable items into n distinguishable buckets, which led us to use the "stars and bars" approach. This is a method used to prove combinatorics theorems, and we found a few different examples of how to implement this in python code online. Our implementation is a recursive function that works by fixing the position of the first "bar," then calling itself with modified parameters to repeat this process until all resources have been allocated. Then, it will repeat this process with the first "bar" fixed in the next possible location, until all possible sets have been enumerated. In terms of resources and nodes, this means that it fixes the first node at 0 resources before calling itself with 1 less node in the parameters. Then after all recursive iterations where node 1 has 0 resources, it will repeat this process with 1 resource allocated to node 1. This loops until all available resources are allocated to node 1, at which point the function returns. This method completely enumerates every possible way that the given resources can be allocated over the given nodes, and still runs well.

After the resource combinations for each player have been enumerated, they then must be packed into a single matrix for the Gamebuild. This is where our functioning C code stops, as we ran into issues that we did not have enough time to solve while trying to code the Gamebuild. Following the structure of the Gamebuild in matlab, there is no way to allow for a variable number of attackers and defenders. Because the strategy sets are compared in nested loops, one for each player, we could not use the structure. Based on the research we did, it is possible to implement the same functionality in a more scalable way, but we did not have enough time to get it close to functional. However, we formatted all the information in the main function so that it would be easy to integrate the gamebuild component, and then the Nash Solver.

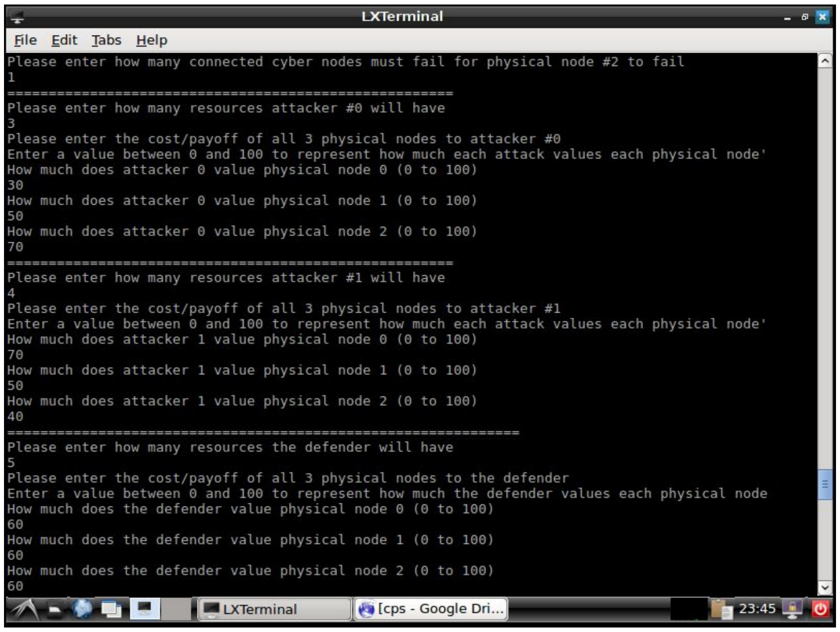
Despite not being able to completely rewrite the matlab code in C, we have made the main function capable of altering any possible input parameter to the Blotto game, which will allow a greater range of scenarios to be analyzed if the code is completed. In addition, we have completely converted the resource combo code into C, allowing it to run much faster and use a less redundant algorithm. This C version of the resource combos could also be modified to write the combinations out to a file, which could then be read into the existing matlab code in an effort to speed it up.

Below are several screenshots of the code being executed in a terminal. It first prompts the user for the number of cyber nodes, physical nodes, and attackers. We decided to cap it at 50 max nodes for each as this allowed for us to allocate memory easier. This number can be larger if needed. It is not so much about the size of the number but more about the fact that there is a number there. It then asks the user to assign connectivity matrices and their hitpoint value. Hitpoint value is what we called the value of the number of connected cyber nodes that would have to fail before a physical node would fail. It then prompts the user for player specific values: the resource value and the cost matrix for each attacker and the defender. From there it returns all the values put into the code by the user. This may seem trivial, but represents that the values can be stored and manipulated. The code then also outputs the resource combinations for each attack and the defender. In the code, these matrices for the resource combinations are concatenated together with flags to mark the dividers, essentially preparing the matrix for future use in game builds.



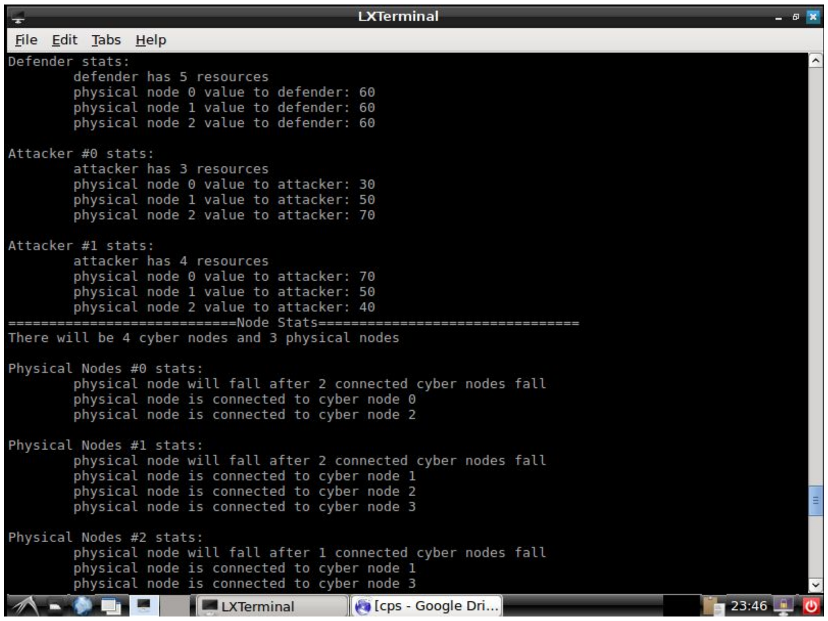
```
root@debian:~/blotto# ./blotto
How many cyber nodes (50 maximum)?4
How many physical nodes (50 maximum)?3
How many attackers?2
=======================================================
Please enter connectivity matrix of 4 digits for the physical node #0
An entry of 1 represents a connection while 0 represents no connection
Does physical node 0 connect with cyber node 0 (1 or 0)1
Does physical node 0 connect with cyber node 1 (1 or 0)0
Does physical node 0 connect with cyber node 2 (1 or 0)1
Does physical node 0 connect with cyber node 3 (1 or 0)0
Please enter how many connected cyber nodes must fail for physical node #0 to fail
2
=======================================================
Please enter connectivity matrix of 4 digits for the physical node #1
An entry of 1 represents a connection while 0 represents no connection
Does physical node 1 connect with cyber node 0 (1 or 0)0
Does physical node 1 connect with cyber node 1 (1 or 0)1
Does physical node 1 connect with cyber node 2 (1 or 0)1
Does physical node 1 connect with cyber node 3 (1 or 0)1
Please enter how many connected cyber nodes must fail for physical node #1 to fail
2
=======================================================
Please enter connectivity matrix of 4 digits for the physical node #2
An entry of 1 represents a connection while 0 represents no connection
Does physical node 2 connect with cyber node 0 (1 or 0)0
Does physical node 2 connect with cyber node 1 (1 or 0)1
Does physical node 2 connect with cyber node 2 (1 or 0)0
Does physical node 2 connect with cyber node 3 (1 or 0)1
Please enter how many connected cyber nodes must fail for physical node #2 to fail
1
=======================================================
Please enter how many resources attacker #0 will have
```

Figure of the input prompts in terminal



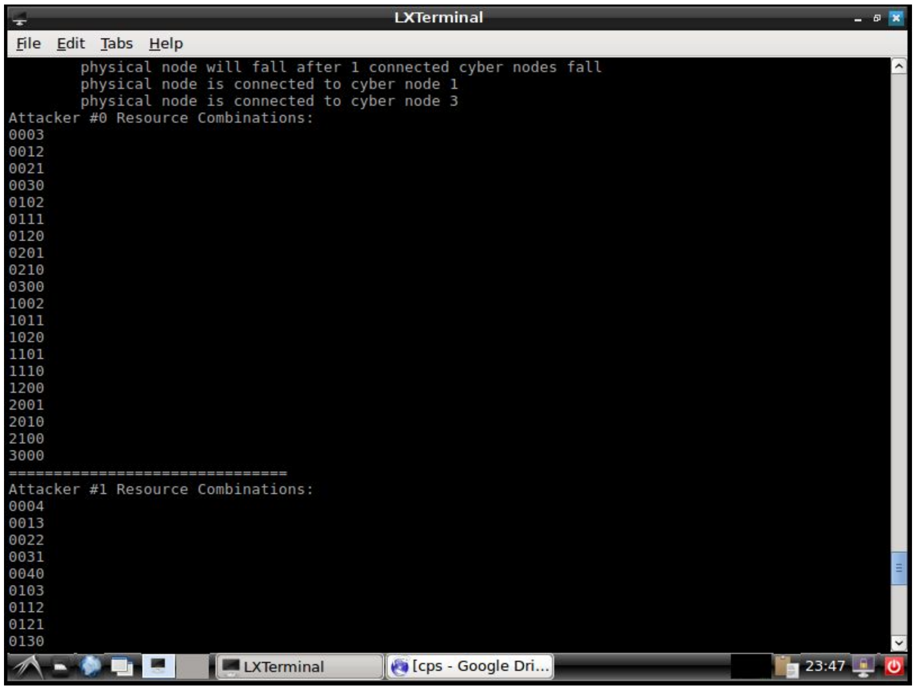
```
Please enter how many connected cyber nodes must fail for physical node #2 to fail
1
=======================================================
Please enter how many resources attacker #0 will have
3
Please enter the cost/payoff of all 3 physical nodes to attacker #0
Enter a value between 0 and 100 to represent how much each attack values each physical node'
How much does attacker 0 value physical node 0 (0 to 100)
30
How much does attacker 0 value physical node 1 (0 to 100)
50
How much does attacker 0 value physical node 2 (0 to 100)
70
=======================================================
Please enter how many resources attacker #1 will have
4
Please enter the cost/payoff of all 3 physical nodes to attacker #1
Enter a value between 0 and 100 to represent how much each attack values each physical node'
How much does attacker 1 value physical node 0 (0 to 100)
70
How much does attacker 1 value physical node 1 (0 to 100)
50
How much does attacker 1 value physical node 2 (0 to 100)
40
===============================================================
Please enter how many resources the defender will have
5
Please enter the cost/payoff of all 3 physical nodes to the defender
Enter a value between 0 and 100 to represent how much the defender values each physical node
How much does the defender value physical node 0 (0 to 100)
60
How much does the defender value physical node 1 (0 to 100)
60
How much does the defender value physical node 2 (0 to 100)
60
```

Next Set of input prompts



```
Defender stats:
        defender has 5 resources
        physical node 0 value to defender: 60
        physical node 1 value to defender: 60
        physical node 2 value to defender: 60

Attacker #0 stats:
        attacker has 3 resources
        physical node 0 value to attacker: 30
        physical node 1 value to attacker: 50
        physical node 2 value to attacker: 70

Attacker #1 stats:
        attacker has 4 resources
        physical node 0 value to attacker: 70
        physical node 1 value to attacker: 50
        physical node 2 value to attacker: 40
============================Node Stats================================
There will be 4 cyber nodes and 3 physical nodes

Physical Nodes #0 stats:
        physical node will fall after 2 connected cyber nodes fall
        physical node is connected to cyber node 0
        physical node is connected to cyber node 2

Physical Nodes #1 stats:
        physical node will fall after 2 connected cyber nodes fall
        physical node is connected to cyber node 1
        physical node is connected to cyber node 2
        physical node is connected to cyber node 3

Physical Nodes #2 stats:
        physical node will fall after 1 connected cyber nodes fall
        physical node is connected to cyber node 1
        physical node is connected to cyber node 3
```

Output of Player and Node Values



```
        physical node will fall after 1 connected cyber nodes fall
        physical node is connected to cyber node 1
        physical node is connected to cyber node 3
Attacker #0 Resource Combinations:
0003
0012
0021
0030
0102
0111
0120
0201
0210
0300
1002
1011
1020
1101
1110
1200
2001
2010
2100
3000
===============================
Attacker #1 Resource Combinations:
0004
0013
0022
0031
0040
0103
0112
0121
0130
```

Output of the Resource Combinations