An application of the Bayesian Stackelberg model to Web Application Security

Artificial Intelligence Project - 2012

Federico De Meo - VR355974
Oscar Maraia - VR353026

1 Introduction

In a class of games known as Stackelberg games, one agent (the leader) must commit to a strategy that can be observed by the other agent (the follower or adversary) before the adversary chooses its own strategy. We consider Bayesian Stackelberg games, in which the leader is uncertain about the types of adversary it may face. This paper presents a possible application of this game to web application security, taking the Google Gruyere web application as a reference for the attacks. We want to compute the optimal mixed strategy for the leader to commit to. We report one of the first efficient exact algorithms to compute this strategy, called DOBSS. We first define the problem of solving a Stackelberg game in its most intuitive form, i.e. as a mixed-integer quadratic program (MIQP); we then show the analogue representation for a Bayesian Stackelberg game. Finally, we show its conversion into a mixed-integer linear program (MILP). We use this form to efficiently compute the objective values with the aid of the IBM ILOG CPLEX Optimizer.

2 Bayesian Stackelberg game model

We consider a game model with two players, each of which has a set of possible actions. A player can choose a strategy, which is a probability distribution over actions and represents the probability that a certain action is chosen by that player. A "pure strategy" is a strategy in which only one action can be chosen, i.e. has probability 1. In a "mixed strategy" every action can be chosen with a probability $0 \leq p < 1$.

A Stackelberg game is a noncooperative, hierarchical decision making game, in which a player named *leader* commits to a strategy first and then a second player named *follower* selfishly optimizes its reward, considering the strategy chosen by the leader. Although the follower in a Stackelberg game is allowed to observe the leader's strategy before choosing its own strategy, there is often an advantage for the leader over the case where both players must choose their moves simultaneously. To see the advantage of being the leader in a Stackelberg game, consider a simple game with the payoff table as shown in Table 1, adapted from [CS06]. The leader is the row player and the follower is the column player. The first number in a cell is the reward of the leader, the second one is the reward of the follower.

The only pure-strategy Nash equilibrium for this game is when the leader plays a and the follower plays c which gives the leader a payoff of 2. However, if the leader can commit to playing b before the follower chooses its strategy, then the leader will obtain a payoff of 3, since the follower would then play d to ensure a higher payoff for itself. If the leader

	c	d
a	2,1	4,0
b	1,0	3,2

Table 1: Payoff table for a Stackelberg game

commits to a uniform mixed strategy of playing a and b with equal (0.5) probability, then the follower will play d, leading to a payoff for the leader of 3.5. Our goal is to determine the optimal strategy for a leader to commit to. In the previous example, the optimal mixed strategy for the leader is playing a with probability $\frac{2}{3}$ and b with probability $\frac{1}{3}$, which leads the follower to play d, resulting in a payoff for the leader of $3.\bar{6}$, which is its maximum expected reward.

A Bayesian Stackelberg game is an incomplete information Stackelberg game in which each player must be one of a given set of types. Types lead to different sets of actions and payoff tables. We assume that there is only one leader type, although there are multiple follower types. The leader does not know the follower's type but knows the probability distribution over his types. Unfortunately, the problem of choosing an optimal strategy for the leader to commit to in a Bayesian Stackelberg game is NP-hard [CS06].

3 DOBSS algorithm

Different approches have been proposed to solve Bayesian Stackelberg games. We adopted one of the first exact methods existing in the literature, DOBSS (Decomposed Optimal Bayesian Stackelberg Solver), first introduced in [PPM$^+$08]. This method has three key advantages. First, the method allows for a Bayesian game to be expressed compactly without requiring conversion to a normal-form game via the Harsanyi transformation. Second, the method requires only one mixed-integer linear program (MILP) to be solved, rather than a set of linear programs as in [CS06], thus leading to a further performance improvement. Third, it directly searches for an optimal leader strategy, rather than a Nash (or Bayes-Nash) equilibrium, thus allowing it to find high-reward non-equilibrium strategies (thus exploiting the advantage of being the leader).

We first define the problem of solving a Stackelberg game in its most intuitive form, i.e. as a mixed-integer quadratic program (MIQP); we then show the analogue representation for a Bayesian Stackelberg game. Finally, we show its conversion into a mixed-integer linear program (MILP).

Note that for a single follower type, we simply take the mixed strategy for the leader that gives the highest payoff when the follower plays a reward-maximizing strategy. We need only to consider the reward-maximizing pure strategies of the followers, since for a given fixed strategy x of the leader, each follower type faces a problem with fixed linear rewards. If a mixed strategy is optimal for the follower, then so are all the pure strategies in the support of that mixed strategy. This allows us to represent the optimal pure strategies using binary variables.

We denote by x the leader's policy, which consists of a vector of the leader's pure strategies. The value x_i is the proportion of times in which pure strategy i is used in the policy. Similarly, q denotes the vector of strategies of the follower. We also denote X and Q the index sets of the leader and follower's pure strategies, respectively. The payoff matrices R and C are defined such that R_{ij} is the reward of the leader and C_{ij} is the reward of the follower when the leader takes pure strategy i and the follower takes pure strategy j.

The leader's MIQP problem is defined as:

$$\begin{array}{ll} \max_{x,q,a} & \sum_{i\in X}\sum_{j\in Q} R_{ij}x_i q_j \\ \text{s.t.} & \sum_{i\in X} x_i = 1 \\ & \sum_{j\in Q} q_j = 1 \\ & 0 \le (a - \sum_{i\in X} C_{ij}x_i) \le (1-q_j)M \\ & x_i \in [0...1] \\ & q_j \in \{0,1\} \\ & a \in \Re. \end{array} \tag{1}$$

To extend this Stackelberg model to handle multiple follower types we follow a Bayesian approach and assume that there is an a priori probability p^l that a follower of type l will appear, with L denoting the set of follower types. In this case, q^l becomes the vector of strategies of follower $l \in L$. We also index the payoff matrices of the leader and each of the follower types l by the matrices R^l and C^l. The leader then solves the following problem:

$$\begin{array}{ll} \max_{x,q,a} & \sum_{i\in X}\sum_{l\in L}\sum_{j\in Q} p^l R^l_{ij}x_i q^l_j \\ \text{s.t.} & \sum_{i\in X} x_i = 1 \\ & \sum_{j\in Q} q^l_j = 1 \\ & 0 \le (a^l - \sum_{i\in X} C^l_{ij}x_i) \le (1-q^l_j)M \\ & x_i \in [0...1] \\ & q^l_j \in \{0,1\} \\ & a \in \Re. \end{array} \tag{2}$$

We now face the final step: eliminating non-linearity of the objective function in the MIQP to generate a MILP. We can linearize the quadratic programming problem 2 through the change of variables $z^l_{ij} = x_i q^l_j$, obtaining the following problem:

$$\begin{array}{ll} \max_{q,z,a} & \sum_{i\in X}\sum_{l\in L}\sum_{j\in Q} p^l R^l_{ij} z_{ij} \\ \text{s.t.} & \sum_{i\in X}\sum_{j\in Q} z^l_{ij} = 1 \\ & \sum_{j\in Q} z^l_{ij} \le 1 \\ & q^l_j \le \sum_{i\in X} z^l_{ij} \le 1 \\ & \sum_{j\in Q} q^l_j = 1 \\ & 0 \le (a^l - \sum_{i\in X} C^l_{ij}(\sum_{h\in Q} z^l_{ih})) \le (1-q^l_j)M \\ & \sum_{j\in Q} z^l_{ij} = \sum_{j\in Q} z^1_{ij} \\ & z^l_{ij} \in [0...1] \\ & q^l_j \in \{0,1\} \\ & a \in \Re. \end{array} \tag{3}$$

This is the final form that we actually implemented through the CPLEX framework. This will be further discussed in section 5.

4 Model applied to Web Applications

We will now describe a possible application of the model to web application security, in which an attacker (follower) tries to exploit some bugs to perform unauthorized actions and a web master (leader) tries to fix those bugs.

The core problem was to populate the payoff matrix in a meaningful way. To do so, we chose a particular web application, Google's Gruyere codelab[LBT11]. The Gruyere application is a deliberately insecure web application which runs on Google's AppEngine

framework in a sandboxed way. This provides a convenient and safe place to practice exploiting (and

fixing) a web application. We selected a subset of the exploits which affect Gruyere. As an useful reference we used *OWASP Top 10 - 2010* [WW10], which reports the ten most critical web application security risks.

- **Injection (SQLi)**: Injection flaws, such as SQL injection, occur when untrusted data is sent to an interpreter as part of a command or query. Fixed by *Escaping Routines* (ER);
- **Cross-Site Scripting (XSS)**: XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation and escaping. Fixed by *Escaping Routines* (ER);
- **Broken Authentication and Session Management (BASM)**: Application functions related to authentication and session management are often not implemented correctly. Fixed by *Session Securify* (SS);
- **Insecure Direct Object References (IDOR)**: A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Fixed by *Access Control* (AC);
- **Cross-Site Request Forgery (CSRF)**: A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web application. Fixed by *CSRF Guard* (CSRFG);
- **Security Misconfiguration (SM)**: Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform. All these settings should be defined, implemented, and maintained as many are not shipped with secure defaults. Fixed by *Environment Securing* (ES);
- **Failure to Restrict URL Access (FRUA)**: Applications need to perform URL access control checks each time pages are accessed, or attackers will be able to forge URLs to access hidden pages anyway. Fixed by *Access Control* (AC);
- **Insufficient Transport Layer Protection (ITLP)**: Applications frequently fail to authenticate, encrypt, and protect the confidentiality and integrity of sensitive network traffic. Fixed by *SSL* (SSL).

Based on our experience we tried to evaluate the gravity of these attacks and relative fixes. We estimated the rewards for both leader (Table 2) and follower (Table 3), considering actions' costs too.

Leader's Fixes	Impact	Cost/Failure	Success
Escaping Routines	8	-1	7
Session Securify	4	-2	2
Access Control	2	-2	0
CSRFGuard	4	-2	2
Environment Securing	3	-1	2
SSL	4	-2	2

Table 2: Leader's rewards

Follower's Exploits	Impact	Cost/Failure	Success
SQLi	4	-1	3
XSS	4	-1	3
Broken Authentication and Session Management	4	-2	2
Insecure Direct Object References	1	-0,5	0,5
CSRF	4	-2	2
Security Misconfiguration	3	-1	2
Failure to Restrict URL Access	1	-0,5	0,5
Insufficient Transport Layer Protection	4	-3	1

Table 3: Follower's rewards

Table 4 derives from the merge of the values of Tables 2 and 3. More specifically:

- if a fix is effective against an exploit:
  - the reward of the leader is the impact of the leader's action, minus the cost of the fix;
  - the reward of the follower is the cost of the attack;
- if a fix is not effective against an exploit:
  - the reward of the leader is negative, due to the cost of the fix and the impact of the follower's action;
  - the reward of the follower is the impact of the follower's action, minus the cost of the attack.

The follower can also choose to do no action at all (NOP).

	SQLi	XSS	BASM	IDOR	CSRF	SM	FRUA	ITLP	NOP
ER	7, -1	7, -1	-5, 2	-2, 0.5	-5, 2	-4, 2	-2, 0.5	-5, 1	-1, 0
SS	-6, 3	-6, 3	2, -2	-3, 0.5	-6, 2	-5, 2	-3, 0.5	-6, 1	-2, 0
AC	-6, 3	-6, 3	-6, 2	-2, -0.5	-6, 2	-5, 2	-2, -0.5	-6, 1	-2, 0
CSRFG	-6, 3	-6, 3	-6, 2	-3, 0.5	0, -2	-5, 2	-3, 0.5	-6, 1	-2, 0
ES	-6, 3	-6, 3	-6, 2	-3, 0.5	-6, 2	3, -1	-3, 0.5	-6, 1	-2, 0
SSL	-6, 3	-6, 3	-6, 2	-3, 0.5	-6, 2	-5, 2	-3, 0.5	0, -3	-2, 0

Table 4: Payoff Matrix

5 Implementation

We implemented the problem so far discussed using IBM ILOG CPLEX Optimizer which, among the others, solves very large linear programming problems using either primal or dual variants of the simplex method. The CPLEX Optimizer has a modeling layer called Concert that provides interfaces to C++, Java, MATLAB and other languages. We chose Java for our case. The complete source code has been attached in Appendix A.

We defined two fictional follower types, Hacker and Lamer, which have the same payoff values but different possible actions. The Hacker type will always do something (he cannot select the NOP action). The Lamer type can't select Cross-Site Request Forgery and Insufficient Transport Layer Protection.

Using the probability distribution over types <40% for Hacker, 60% for Lamer>, the output of the program execution is the following:

```
Total (root+branch&cut) =    0.05 sec.

Solution status = Optimal
Leader's Maximum Expected Utility = -0.09499999999999931
Hacker's Maximum Expected Utility: 1.3076923076923077
Lamer's Maximum Expected Utility: 1.1

Hacker's Pure Strategy
Action: XSS

Lamer's Pure Strategy
Action: SQLi

Leader's Mixed Strategy
Probability of action Escaping Routines: 0.4542307692307692
Probability of action Session Securify: 0.20423076923076922
Probability of action Access Control: 0.0
Probability of action CSRF Guard: 0.06923076923076923
Probability of action Environment Securing: 0.2723076923076923
Probability of action SSL: 0.0
```

This values confirm the information gathered from the OWASP report, which puts SQL Injection and Cross-Site Scripting at the first places. Therefore, the best action for the webmaster is to sanitize the web application input forms through escaping routines.

Using a different probability distribution over types, 60% for Hacker and 40% for Lamer, the Leader's Maximum expected utility decreases to -0.22999999999999932 (as the Hacker type is more dangerous than the Lamer one).

Let's compare the previous results with a totally uninformed strategy, i.e. the leader chooses randomly its action with uniform probability. In this case the follower would choose the SQL Injection or Cross-Site Scripting pure strategies (the ones with the maximum reward for the follower). This would lead to a leader's expected utility of $-3,833333333$, which is far worse than the utility resulting from the informed strategy.

Let's consider another case: if the leader chooses to implement all possible defenses the follower would no longer attack (as it would lose for every attack, except for the NOP action) and the leader will suffer a total cost of -10.

6 Conclusion

This paper presents our study on Bayesian Stackelberg games, a new possible approach to web application security analysis and the results of a high-performance tool, such as CPLEX, to solve the problem. DOBSS proved to be pretty powerful for our case, although more efficient algorithms already exist [JKT11].

Future work could focus on:

- defining more realistic follower types;
- refining the computation of payoff values for both leader and follower;
- introducing the value of the data being protected as a new variable in the computation;
- implementing realtime update of those values through information gathering tools such as intrusion detection systems.

References

[CS06] Vincent Conitzer and Tuomas Sandholm. Computing the optimal strategy to commit to. In *Proceedings of the 7th ACM conference on Electronic commerce*, EC '06, pages 82–90, New York, NY, USA, 2006. ACM.

[JKT11] Manish Jain, Christopher Kiekintveld, and Milind Tambe. Quality-bounded solutions for finite bayesian stackelberg games: scaling up. In *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 3*, AAMAS '11, pages 997–1004, Richland, SC, 2011. International Foundation for Autonomous Agents and Multiagent Systems.

[LBT11] Bruce Leban, Mugdha Bendre, and Parisa Tabriz. Google's Gruyere Codelab. `http://google-gruyere.appspot.com/`, 2011.

[PPM+08] Praveen Paruchuri, Jonathan P. Pearce, Janusz Marecki, Milind Tambe, Fernando Ordonez, and Sarit Kraus. Playing games for security: an efficient exact algorithm for solving bayesian stackelberg games. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems - Volume 2*, AAMAS '08, pages 895–902, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.

[WW10] Jeff Williams and Dave Wichers. OWASP Top 10 2010. `https://www.owasp.org/index.php/Top_10_2010`, 2010.

Appendices

A Source Code

```
package BSG;

import ilog.concert.*;
import ilog.cplex.*;

public class BSG {

  static int inf = Integer.MIN_VALUE+1000;
  static int M = Integer.MAX_VALUE;

  // Distribuzione di probabilita' sui tipi di attaccante {Hacker, Lamer}
  static double[] p = { 0.4, 0.6 };

  // Matrice dei reward del Leader
  static double[][] A = { { 7, 7, -5, -2, -5, -4, -2, -5, -1 },
                          { -6, -6, 2, -3, -6, -5, -3, -6, -2 },
                          { -6, -6, -6, -2, -6, -5, -2, -6, -2 },
                          { -6, -6, -6, -3, 0, -5, -3, -6, -2 },
                          { -6, -6, -6, -3, -6, 3, -3, -6, -2 },
                          { -6, -6, -6, -3, -6, -5, -3, 0, -2 }};

  // Matrice dei reward del Follower
  static double[][][] B = {{ //Matrice Hacker
                           { -1, -1, 2, 0.5, 2, 2, 0.5, 1, inf },
                           { 3, 3, -2, 0.5, 2, 2, 0.5, 1, inf },
                           { 3, 3, 2, -0.5, 2, 2, -0.5, 1, inf },
                           { 3, 3, 2, 0.5, -2, 2, 0.5, 1, inf },
                           { 3, 3, 2, 0.5, 2, -1, 0.5, 1, inf },
                           { 3, 3, 2, 0.5, 2, 2, 0.5, -3, inf } },
                          { //Matrice Lamer
                           { -1, -1, 2, 0.5, inf, 2, 0.5, inf, 0 },
                           { 3, 3, -2, 0.5, inf, 2, 0.5, inf, 0 },
                           { 3, 3, 2, -0.5, inf, 2, -0.5, inf, 0 },
                           { 3, 3, 2, 0.5, inf, 2, 0.5, inf, 0 },
                           { 3, 3, 2, 0.5, inf, -1, 0.5, inf, 0 },
                           { 3, 3, 2, 0.5, inf, 2, 0.5, inf, 0 }}};

  // Dimensioni delle matrici
  static int I, J, L;

  static String[] types = { "Hacker", "Lamer" };
  static String[] actionsLeader = { "Escaping Routines", "Session Securify", "Access Control", "CSRF Guard", "Environment Securing", "
        SSL" };
  static String[] actionsFollower = { "SQLi", "XSS", "Broken Authentication and Session Management", "Insecure Direct Object
        References", "CSRF", "Security Misconfiguration", "Failure to Restrict URL Access", "Insufficient Transport Layer Protection",
         "No Action" };

  public static void main(String[] args) {

    IloCplex cplex = null;
    try {
      cplex = new IloCplex();
      L = B.length;
      I = B[0].length;
      J = B[0][0].length;

      // vettore di variabili, serve solo per la stampa finale
      IloNumVar[][] var = new IloNumVar[L * (I + 1) + 1][];

      // aggiungiamo l'obbiettivo e i vincoli
      populateByRow(cplex, var);

      // risolviamo e stampiamo il risultato
      if (cplex.solve()) {
        double[] values;
        System.out.println("######## Solution status = "+ cplex.getStatus());
        System.out.println("######## Leader's Maximum Expected Utility = "+ cplex.getObjValue());
        values = cplex.getValues(var[L * I + L]);
        for (int l = 0; l < L; l++) {
          System.out.println("######## " + types[l] + "'s Maximum Expected Utility: " + values[l]);
        }
        System.out.println();
        int[] choices = new int[L];
        for (int l = 0; l < L; l++) {
          values = cplex.getValues(var[L * I + l]);
          System.out.println("######## " + types[l] + "'s Pure Strategy ######## ");
          System.out.print("Action: ");
          for (int j = 0; j < values.length; j++) {
             if (values[j] == 1) {
              System.out.println(actionsFollower[j]);
              choices[l] = j;
              break;
             }
          }
          System.out.println();
        }
```

```
      System.out.println("######## Leader's Mixed Strategy ######## ");
      double suml = 0;
      for (int i = 0; i < I; i++) {
        suml = 0;
        for (int l = 0; l < L; l++) {
          values = cplex.getValues(var[l * I + i]);
          suml += values[choices[l]]*p[l];
        }
        System.out.println("Probability of action " + actionsLeader[i] + ": " + suml);
      }
    }

    try {
      cplex.exportModel("mps.mps");
    } catch (Exception el) {
    }
    cplex.end();
  } catch (IloException e) {
    System.err.println("Concert exception caught '" + e + "' caught");
  }
}

static void populateByRow(IloMPModeler model, IloNumVar[][] var)
    throws IloException {

  // variabili z_lij ( z_lij = x_i * q_lj)
  IloNumVar[][][] z = new IloNumVar[L][I][J];
  for (int l = 0; l < L; l++) {
    for (int i = 0; i < I; i++) {
      for (int j = 0; j < J; j++) {
        z[l][i][j] = model.numVar(0.0, 1.0, "z" + l + "" + i + "" + j);
      }
      var[l * I + i] = z[l][i];
    }
  }

  // variabili q_lj
  IloIntVar[][] q = new IloIntVar[L][J];
  for (int l = 0; l < L; l++) {
    for (int j = 0; j < J; j++) {
      q[l][j] = model.intVar(0, 1, "q" + l + "" + j);

    }
    var[L * I + l] = q[l];
  }

  // variabile a (massimo reward del follower)
  IloNumVar[] a = new IloNumVar[L];
  for (int l = 0; l < L; l++) {
    a[l] = model.numVar(Double.MIN_VALUE, Double.MAX_VALUE, "a" + l);
  }
  var[L * I + L] = a;


  // FUNZIONE OBIETTIVO
  IloNumExpr sum = model.numExpr();
  for (int l = 0; l < L; l++) {
    for (int i = 0; i < I; i++) {
      for (int j = 0; j < J; j++) {
        sum = model.sum(
            model.prod(
              model.prod(A[i][j], z[l][i][j]),
              p[l]),
            sum);
      }
    }
  }
  model.addMaximize(sum);

  // VINCOLO Sommatoria[z_Lij] = 1
  IloNumExpr sumz = null;
  for (int l = 0; l < L; l++) {
    sumz = model.numExpr();
    for (int i = 0; i < I; i++)
      for (int j = 0; j < J; j++)
        sumz = model.sum(z[l][i][j], sumz);
    model.addEq(sumz, 1.0, "Somma z_"+l+"ij = 1");
  }

  // VINCOLO Sommatoria[z_LIj] <= 1
  for (int l = 0; l < L; l++) {
    for (int i = 0; i < I; i++) {
      sumz = model.numExpr();
      for (int j = 0; j < J; j++)
        sumz = model.sum(z[l][i][j], sumz);
      model.addLe(sumz, 1, "Somma z_"+l+""+i+"j <= 1");
    }
  }

  // VINCOLO Sommatoria[q_Lj] = 1
  IloNumExpr sumq = null;
  for (int l = 0; l < L; l++) {
    sumq = model.numExpr();
    for (int j = 0; j < J; j++)
      sumq = model.sum(q[l][j], sumq);
```

```
      model.addEq(sumq, 1, "Somma q"+l+"j = 1");
    }

    // VINCOLO q_LJ <= Sommatoria z_LiJ <= 1
    for (int l = 0; l < L; l++) {
      for (int j = 0; j < J; j++) {
        sumz = model.numExpr();
        for (int i = 0; i < I; i++)
          sumz = model.sum(z[l][i][j], sumz);
        model.addLe(q[l][j], sumz, "q_"+l+""+j+" <= Somma z_"+l+"i"+j);
        model.addLe(sumz, 1.0, "Somma z_"+l+"i"+j+" <= 1");
      }
    }

    // VINCOLO 0 <= (a - Sommatoria B_LiJ * (Sommatoria z_Lih)) <= (1 - q_LJ)*M
    for (int l = 0; l < L; l++) {
      for (int j = 0; j < J; j++) {
        IloNumExpr sumj = a[l];
        IloNumExpr sumc = model.numExpr();
        for (int i = 0; i < I; i++) {
          IloNumExpr sumh = model.numExpr();
          for (int h = 0; h < J; h++) {
            sumh = model.sum(z[l][i][h], sumh);
          }
          sumc = model.sum(model.prod(B[l][i][j], sumh), sumc);
        }
        sumj = model.diff(sumj, sumc);
        model.addLe(0, sumj, "0 <= a["+l+"] - Sommatoria B_"+l+"i"+j+" * Somma z_"+l+"ih");
        model.addLe(sumj, model.prod(model.diff(1.0, q[l][j]), M), "a["+l+"] - Sommatoria B_"+l+"i"+j+" * Somma z_"+l+"ih <= (1 - q_"+
              l+""+j+")*M");
      }
    }

    // VINCOLO Sommatoria z_LIj = Sommatoria z_1Ij
    IloNumExpr sumz1 = null;
    for (int i = 0; i < I; i++) {
      sumz1 = model.numExpr();
      for (int j = 0; j < J; j++)
        sumz1 = model.sum(q[0][j], sumz1);
      for (int l = 1; l < L; l++) {
        sumz = model.numExpr();
        for (int j = 0; j < J; j++)
          sumz = model.sum(q[l][j], sumz);
        model.addEq(sumz, sumz1, "Somma z_"+l+""+i+"j = Somma z_0"+i+"j");
      }
    }
  }
}
```