Vol. 13, No. 4 (1987)

Edited by
George J Klir

International Journal of

GENERAL SYSTEMS

Methodology • Applications • Education

Gordon and Breach Science Publishers
New York London Paris Montreux Tokyo Melbourne

ISSN 0308-1079

IJGS 13(4) 289–372 (1987)

International Journal of General Systems

CONTENTS Volume 13, Number 4 (1987)

Int. J. General Systems, Vol. 13, pp. 323–332
Photocopying permitted by license only

PRISONER'S DILEMMA REVISITED: A NEW STRATEGY BASED ON THE GENERAL SYSTEM PROBLEM SOLVING FRAMEWORK

FRANÇOIS E. CELLIER
Department of Electrical and Computer Engineering, University of Arizona, Tucson, Arizona 85721, U.S.A.

(Received 21 January 1987; in final form 13 April 1987)

In this paper, the famous Prisoner's Dilemma Problem is reexamined by using the methodology of the General System Problem Solving (GSPS) Framework. It is shown that this framework provides us with a powerful means to take into account the secondary effects that were ignored by previous strategies. In a classroom contest, the new strategy proved far superior to previously employed techniques.

INDEX TERMS: Prisoner's dilemma, game theory, General System Problem Solving framework, forecasting, computer software.

INTRODUCTION

The Prisoner's Dilemma Problem[1] is a famous example of a game characterized by very simple rules that appears to be unsolvable by mathematical logic.

Two misdemeanants have been arrested by a police officer who books them into the city jail, and keeps them in two separate cells without means of communication. Evidence suffices to keep both behind bars for a duration of two years. However, each of the defendants is offered the same deal by the prosecutor. If he confesses to their jointly committed crimes while his alleged accomplice keeps quiet, he will go free whereas his mate-in-crime will "do time" for a total of five years. However, if both confess to their crimes each of them will be sentenced to four years in jail. Thus, the game is characterized by the truth-table shown in Figure 1 where the numbers denote years of imprisonment that each of them can save in comparison with his maximum sentence.

To make the game a little more interesting, we shall now repeat the same game with the same two players over a fixed number of steps, say 200 steps. After each step, each of the players gets to know how his adversary decided in the previous step. Each of the players tries to maximize his own "profit" over the 200 step period.

In the best of all cases, one of the players will make 1000 points while the other

A_{INPUT}	B_{INPUT}	A_{OUTPUT}	B_{OUTPUT}
stays mum	stays mum	3	3
stays mum	confesses	0	5
confesses	stays mum	5	0
stays mum	confesses	1	1

Figure 1 Truth table for the Prisoner's Dilemma problem.

goes empty handed. However, this is very unlikely to happen since, already after the first step, the other player will learn what his chum did to him, and is most likely going to retaliate. This will encourage the first player to take revenge again, and both players will fall into a pattern of defecting, in which case neither of them is going to make a large profit. The only way how any of the players can hope to make profit is by seeking means of cooperation, that is: by settling for 600 points knowing that his adversary is going to make 600 points, too.

This is obviously a very important problem, because, if we can solve this problem, we can teach Washington and Moscow, and thereby save the world, as the only way how we got a change to survive the nuclear age is by developing a pattern of mutual trust.

SOLVING PRISONER'S DILEMMA THE HEURISTIC WAY

In the past, many different algorithms have been proposed to tackle this problem. Several contests were made to determine the best possible strategy. Thereby, each algorithm played against each other algorithm by accumulating points over all the games. Thus, $n*(n-1)/2$ games were played where n denoted the number of participating algorithms. Most of the algorithms submitted were of a strictly heuristic nature.

It turned out that one algorithm performed particularly well. This algorithm obtained the name TIT FOR TAT. It is amazingly simple. During the first step, the algorithm cooperates. Thereafter, it simply repeats what the adversary algorithm did in the previous step. The algorithm enjoys a number of very appealing characteristics:

1) The algorithm cannot be exploited. Whatever the adversary does, he will never make more than five points over those collected by TIT FOR TAT. Thus, the responsibility is totally with the enemy. If he wants to make profit, he must allow TIT FOR TAT to prosper, too.
2) The algorithm is extremely adaptive and always willing to cooperate. If the adversary decided to defect during the first 100 steps and cooperate during the second 100 steps, TIT FOR TAT will cooperate during the first step only, then defect during steps 2 to 101, then immediately cooperate again during steps 102 to 200. Thus, old sins are forgotten and forgiven after only one step of the game.
3) The algorithm is extremely benign. It will never probe the adversary, and thereby risk a breakdown of cooperation.

I never figured out where and when the name "TIT FOR TAT" was created. Actually, it is a misnomer. The algorithm really should be called "I4I", because it is as old as the Old Testament where it can be looked up under the heading "an eye for an eye, and a tooth for a tooth" (2nd Mose [21, 24]). Interestingly enough, this is one of those statements of the Old Testament that is most criticized by Christianity worldwide, as it seems to reveal an unforgiving and cruel God in place of the kind and forbearing "old man" with the white beard that Christians prefer to envisage. Actually, this perception is entirely wrong. There is hardly any

"algorithm" more forgiving than TIT FOR TAT. After one step only, the algorithm has become reconciled. Somehow, the biblical statement implies the (unfortunately omitted) addition: "a hug for a hug, and a kiss for a kiss".

Does the New Testament offer a better "algorithm" to work with? In Matt. [5, 39], we are told that "whosoever shall smite thee on thy right cheek, turn to him the other also". Indeed, also this "algorithm" is often referred to in the Prisoner's Dilemma literature. It is called TIT FOR TWO TAT, and describes a variation of the previously discussed TIT FOR TAT algorithm, in that retaliation is seeked only after two consecutive offenses. After one "slap", the algorithm shows "its other cheek",...but if it is hit again, then finally it retaliates (or did I misinterpret the biblical message?). However, this algorithm has severe drawbacks when compared to TIT FOR TAT.

1) It can easily be exploited. A repetitive pattern of defecting altered with cooperation will provide the adversary with the same 600 points as uncompromised cooperation, while TIT FOR TWO TAT is left with 100 points only.

2) It does not provide the adversary with a clearcut message about its seriousness, and thereby encourages defecting.

SYSTEMATIC APPROACHES TO PRISONER'S DILEMMA

The aforementioned algorithms were all found in a purely heuristic manner. Are there any systematic ways to crack this problem? A closed solution seems unfeasible. However, the optimality principle might eventually help us to come up with an answer.

From the perspective of a control engineer, this problem is a decentralized optimal control problem with competing performance indices and incomplete knowledge.[2] It may well be worthwhile to investigate whether Control theory can help us tackle this problem. Unfortunately, most of the control algorithms are more into continuous variables than discrete variables. But eventually, an adaptation of dynamic programming[3] might help.

However, we tried another approach, namely by the use of the General System Problem Solving Framework.[4] The above "truth table" looks pretty much like the *behavior relation*[5] of a completely deterministic input/output model. Thus, the probability vector can be omitted, as all probabilities are exactly equal to one. The decision flow of the two players over the 200 steps of one game can be interpreted as a *raw data model* with two component variables (the decisions of the two players). Both variables are binary, as each player has only two alternatives in any one step. The total game can thus be represented by a raw data matrix consisting of two columns and 200 rows.

All of the previously used algorithms had one drawback in common. They did not take into consideration any secondary effects, that is: my current answer will probably affect the adversary's next step which, in turn, will affect my next step, a.s.f. Optimization should consider these secondary effects. Unfortunately, this seems to be a difficult problem as no *a priori* knowledge about the adversary's strategy is being provided.

For this purpose, we decided to decompose the overall problem into two simpler problems:

1) Determination of a model of the adversary's behavior with optimized forecasting power, and
2) Determination of an optimal strategy based on the previously determined forecasting model.

DETERMINATION OF A FORECASTING MODEL

To be able to determine any model of the adversary's behavior, we require measurement data. Thus, the algorithm must start by using any good strategy to collect data about the adversary's behavior. One of the most serious drawbacks of this approach is the fact that the system has to be shaken up in order to provide good data. The optimal strategy would probably be to use a random number generator during the first 10 steps of each game to probe the adversary. Unfortunately, this probing may in itself influence the adversary's behavior, and prevent further cooperation for all future. Thus, we decided to play TIT FOR TAT during the first 10 steps of the game, by leaving eventual probing up to the adversary.

Having gathered raw data over 10 steps, we can now perform an *optimal mask analysis*[5] to obtain the best possible forecasting model of the available data.

DETERMINATION OF THE BEST STRATEGY

Using the previously computed forecasting model, we can predict all possible behavior patterns over a number of steps, say 4 steps. In four steps, there exist 16 different choices leading to 16 different forecasted behaviors. Among those, the one is chosen that optimizes my own profit over the next 4 steps. This determines my next move. In this way, we have been able to devise a strategy that optimally considers secondary effects over four steps.

THE ITERATION PROCEDURE

After the adversary has made his next move, the algorithm compares his actual behavior with his forecasted behavior. If they agree, the same forecasting model is used to determine its next move by optimizing again over 4 steps. However, if the actual behavior differs from the forecasted behavior, the last 10 steps are used to evaluate an improved forecasting model prior to the optimization of the next move.

ASSESSMENT OF PRO'S AND CON'S

This algorithm has a number of remarkable properties that make it very appealing for the task at hand:

1) The algorithm is benign by nature. As long as it is not probed by the adversary, it will effectively cooperate, and behaves the same way as TIT FOR TAT does.

2) The algorithm is adaptive to modified behavior of the adversary. Due to the comparison between forecasted and actual behavior, the algorithm is enabled to adapt itself to a changing strategy of the adversary.
3) The algorithm cannot easily be exploited. Even if the adversary has complete knowledge over how the algorithm works, it will not be easy to devise a strategy that defeats it.
4) The algorithm is able to exploit stupidity of the adversary. Contrary to TIT FOR TAT, our system theoretic scheme is able to exploit unintelligent behavior of the adversary in an optimal manner. Algorithms can be divided into benign, malignant, and autistic. Each type of algorithm is confronted in an optimal way, and weaknesses are shamelessly exploited.
5) The algorithm is not purely heuristic, but incorporates optimality both in the determination of the forecasting model, and in the evaluation of the next move.

The only disadvantage of our new algorithm, as we see it, is its lack of transparency. Trust develops much more on the basis of *consequent behavior* rather than simply *benign and tolerant behavior.* This is another important advantage of TIT FOR TAT that was not mentioned earlier in this paper. The behavior of TIT FOR TAT can easily be predicted. It is extremely consequent, and therefore, any adversary knows what he is confronted with. The secrecy behind any fruitful cooperation is not to necessarily cooperate on arbitrary terms at all times, but to convince the adversary that total cooperation is in his own best interest. TIT FOR TAT does precisely that. Our new algorithm is much more complex, and therefore, does not share the beauty of TIT FOR TAT which lies in its simplicity.

IMPLEMENTATION IN SAPS-II

SAPS-II[5] is an interactive programming tool for the analysis and design of systems using the methodology of the GSPS Framework.[4] The above presented algorithm for the Prisoner's Dilemma problem was implemented in SAPS-II, and we want to present the solution in order to show how easily SAPS-II can be utilized for the implementation of rather elaborate new algorithms.

One step of the Prisoner's Dilemma problem is performed by the SAPS-II (that is: CTRL-C[6]) function shown in Figure 2. Function DILEMMA( ) returns the next move to be made on the basis of the previous move of the adversary. With the above given description, the code should be easily understandable. As can be seen, CTRL-C provides us with a fullfledged programming environment. Function DILEMMA( ) actually performs the basic strategy of the algorithm only, whereas the next move is computed inside the user function NEXTMOVE( ) shown in Figure 3, which, on the basis of the previously collected raw data and the optimal mask that was evaluated in function DILEMMA( ), computes both the next move (OUT) and the predicted answer of the adversary (NXT).

FORECAST( ) is a SAPS-II system function. Its first argument is the raw data matrix concatenated from below with space to store the prediction. Thereby, the wanted number of forecasting steps is implicitly determined. Input data must be provided for those variables (columns) for which, according to the mask, no prediction is to be evaluated. In our case, only the second variable needs to be

```
//[out]=DILEMMA(in);
//
//This function plays one step of Prisoner's Dilemma
//
//in:=-1:first step
//in:=  0:adversary defeated during last step
//in:=+1:adversary cooperated during last step
//
//out:=0:algorithm wants to defeat during current step
//out:=1:algorithm wants to cooperate during current step
//
LOAD raw mask nxt<dilemma.dat
IF in=-1,...      //Cooperate on first move
   raw=1;...      //Initialize raw data, and declare RAW as global
   out=1;...
   ELSE...
  [n,m]=SIZE(raw);...
  raw(n,2)=in;...      //Store adversary's last move in raw data
  IF n<10,...          //Play TIT for TAT during 2nd to 10th step
      out=in;...
      ELSE...
      If n=10,...      //Compute forecasting model in 11th step
        mcan=[-1,-1;-1,-1;-1,-1;0,1];...
        mask=OPTMASK(raw,mcan);...
        [out,nxt]=NEXTMOVE(raw,mask);...
      ELSE...
        IF in=nxt,...//If forecast was correct, use old model
          [out,nxt]=NEXTMOVE(raw,mask);...
         ELSE...   //If forecast was wrong, compute new model
          r=raw(n-9:n,:);...
          mcan=[-1,-1;-1,-1;-1,-1;0,1];...
          mask=OPTMASK(r,mcan);...
          [out,next]=NEXTMOVE(raw,mask);...
        END,...
      END,...
    END,...
  raw(n+1,1)=out;...  //Store new move in raw data matrix
  END
$ DELETE/NOCONFIRM dilemma.dat;1
SAVE raw mask nxt >dilemma.dat
RETURN
```

Figure 2 Basic strategy for one step of the Prisoner's Dilemma problem.

```
//[move,guess]=NEXTMOVE(raw,mask);
//
//Computes the best next move out of raw data and optimal mask
//
//move: =next move to be played
//guess: =predicted answer of adversary
//
LOAD profit <nextmove.dat
[n,m]=SIZE(raw);
//
//After 10 steps, compute the initial gain vector. This row vector
//contains the own merit to the left, and the adversary's merit
//to the right.
IF n=10,...
    profit=[0,0];...
    profit=MERIT(raw,profit,n);...
  END
//
//Loop over the 16 possible strategies, concatenate the strategy
//below the raw data matrix, perform the forecast, and evaluate
//its merit
strat=[0,0;0,0;0,0;0,0];
p2=[0,0];
ff=[raw;strat];
FORi=1:16,...
    f1=[raw;strat];...
    f2=FORECAST(f1,mask,n,0);...
    ff=[ff,f2];...
    p1=MERIT(f2,profit,n);...
    p2=[p2;p1];...
    strat=PERMUTE(strat);...
  END
//Find the best possible strategy
best=1;
ownbest=p2(1,1);
advbest=p2(1,2);
FOR i=2:16,...
    IF p2(i,1)>ownbest,...
      best=i;...
      ownbest=p2(i,1);...
      advbest=p2(i,2);...
    END,...
  If p2(i,1)=onwbest,...
    IF p2(i,2)<advbest,...
      best=i;...
      ownbest=p2(i,1);...
      advbest=p2(i,2);...
    END,...
  END,...
END
profit=p2(best,:);
$ DELETE/NOCONFIRM nextmove.dat;1
SAVE profit >nextmove.dat
b1=2*best+1;
b2=b1+1;
f2=ff(:,b1:b2);
move=f2(n+1,1);
guess=f2(n+1,2);
RETURN
```

Figure 3 Evaluation of the next move in the Prisoner's Dilemma problem.

forecasted, thus, data must be provided for the first variable over the next four steps. This is called the *strategy* (STRAT), and there are obviously 16 different strategies possible.

The user function PERMUTE( ), depicted in Figure 4, computes the next strategy out of the current strategy. This is not necessarily the most elegant algorithm for that purpose, but it is easily readable. It produces (0;0;0;0)→(0;0;0;1)→(0;0;1;0)→(0;0;1;1), etc.

The user function MERIT( ), shown in Figure 5, computes the profit of a particular raw data matrix. To save execution time, only the additional profit since the last evaluation is computed. MERIT( ) returns a row vector. Its left element is the own profit, its right element is the profit made by the adversary.

The algorithm is initialized by the CTRL-C "DO function" INITDILEMMA.CTR, given in Figure 6, which can be executed by the command:

DO INITDILEMMA

This function defines the four user functions, initializes SAPS-II, and creates initial data files for the "local" variables of the user functions DILEMMA( ) and NEXTMOVE( ) that must be saved between consecutive calls. In CTRL-C, commands starting with "$" are interpreted as commands of the underlying operating system (in our case: VAX/VMS), and are passed on for execution.

As this example demonstrates, SAPS-II makes the implementation of new algorithms a fairly easy task. The new strategy for the Prisoner's Dilemma problem is quite involved. An implementation of this algorithm in one of the commonly used programming languages (e.g. FORTRAN) would require roughly 2000 lines of code and several weeks of work.

```
//[nstrat]=PREMUTE(ostrat);
//
//User function to compute the next strategy
//
nstrat=ostrat;
IF ostrat(4,1)=0,...
    nstrat(4,1)=1;...
  ELSE...
    nstrat(4,1)=0;...
    IF ostrat(3,1)=0,...
      nstrat(3,1)=1;...
    ELSE...
      nstrat(3,1)=0;...
      IF ostrat(2,1)=0,...
        nstrat(2,1)=1;...
      ELSE...
        nstrat(2,1)=0;...
        IF ostrat(1,1)=0,...
          nstrat(1,1)=1;...
      ELSE...
        nstrat(1,1)=0;...
      END,...
    END,...
  END,...
 END
RETURN
```

Figure 4 Generation of all input patterns available to the adversary.

```
//[newgain] = MERIT(data,oldgain,nrec)
//
//User function to compute the new profit, given the old profit was
//already computed over the first nrec steps.
//
newgain = oldgain;
[n,m] = SIZE(data);
FOR i = nrec + 1:n,...
    IF data (i,1) = 0,...
      IF data (i,2) = 0,...
        newgain = newgain + [1,1];...
      ELSE...
        newgain = newgain + [5,0];...
      END,...
    ELSE...
      IF data (i,2) = 0,...
        newgain = newgain + [0,5];...
      ELSE...
        newgain = newgain + [3,3];...
      END,...
    END,...
  END
RETURN
```

Figure 5 Computation of the merit of a particular strategy.

```
//DO function to initialize prisoner's dilemma
//
DEFF dilemma
DEFF nextmove
DEFF merit
DEFF permute
DO saps:saps
$ PURGE
$ IF F$SEARCH("dilemma.dat"). NES. ""THEN DELETE dilemma.dat;
$ IF F$SEARCH("nextmove.dat"). NES. ""THEN DELETE nextmove.dat;
raw = 0;
mask = 0;
nxt = 0;
SAVE raw mask nxt >dilemma.dat
profit = 0;
SAVE profit >nextmove.dat
RETURN
```

Figure 6 Initialization of the Prisoner's Dilemma algorithm.

The execution of elaborate CTRL-C functions is currently still a rather slow and inefficient process though. It would make sense to provide a "function compiler" that allows to compile rigorously debugged user functions, and add them to the set of standard system functions. Such a feature is currently not yet available.

A CLASSROOM CONTEST

Since the proof of the cake is in the eating, we decided to test our new scheme against other schemes. For that purpose, we asked the students of a senior level class on Artificial Intelligence to come up with algorithms for the Prisoner's

Dilemma Problem. Of course, several of the (more intelligent) students consulted the literature, and came up with all kind of variations of TIT FOR TAT. Others decided on a more personalized route, and designed algorithms of their own. Finally, I ended up with 25 different algorithms to run against each other in the previously described manner.

As was to be expected, most of the individualized algorithms were no good at all, and ranked poorly. TIT FOR TAT and its cousins ranked all about the same, starting with the third rank. However, there were two algorithms doing significantly better (obviously by exploiting the less intelligent ones). One of them was my own algorithm, and the other was an algorithm that based on a pattern recognition approach. That algorithm probed its adversaries during the first 20 steps by playing at random. On the basis of the obtained results, it decided whether the adversary was benign, malignant, or autistic, and thereafter used three different (but not adaptive) schemes to deal with its enemies.

The contest was repeated several times with different random number streams for those algorithms working with some sort of randomness. My own algorithm usually worked best except for one case where it ranked second behind the pattern recognizer.

CONCLUSIONS

In this paper, a new scheme was developed for tackling the Prisoner's Dilemma Problem. This scheme bases on the GSPS Framework. The overall problem has been decomposed into two subproblems. One is the determination of a forecasting model to predict the adversary's behavior, the other is a scheme to determine the next move on the basis of the previously determined forecasting model. For both subproblems, we were able to devise an optimal scheme.

REFERENCES

1. D. R. Hofstadter, "Computer tournaments of the prisoner's dilemma suggest how cooperation evolves." *Scientific American*, **248**(5), 1983, pp. 14–20.
2. Y.-C. Ho and K.-C. Chu, "Team decision theory and information structures in optimal control problems." *IEEE Trans. Automatic Control*, **AC-17**, 1972, pp. 15–28.
3. R. E. Bellman, *Applied Dynamic Programming*, Princeton University Press, Princeton, N.J., 1962.
4. G. J. Klir, *Architecture of Systems Problem Solving*. Plenum Press, New York, 1985.
5. F. E. Cellier and D. W. Yandell, "SAPS-II: A new implementation of the systems approach problem solver," *Inter. J. General Systems*, **13**, 4, pp. 307–322.
6. Systems Control Technology, Inc. *CTRL-C. A Language for the Computer-Aided Design of Multivariable Control Systems*. Systems Control Technology, Inc., Palo Alto, CA 94304, 1984.