IJCAI–13 Formatting Instructions*

Francesca Rossi
University of Padova
Italy
pcchair13@ijcai.org

Abstract

The *IJCAI–13 Proceedings* will be printed from electronic manuscripts submitted by the authors. The electronic manuscript will also be included in the online version of the proceedings. This paper provides the style instructions.

1 Introduction

2 General Test Game

A *test game* $G = (A, \mathbb{B}, p, v, m, t)$ is a 2-player Bayes game between the tester and the test taker. The tester has a set of potential problems $A = \{a_1, a_2, \ldots, a_n\}$ to test and the test taker has an uncertain type (thus Bayes) characterized by $\mathbb{B} = \{B_1, B_2, \ldots, B_L\}$. Set $B_l \subseteq A$ is the unsolvable problem set of the l-th type test taker. Function $p : \mathbb{B} \to [0, 1]$ characterizes the probability that a particular type test taker occurs. The memorize capacity function $m : \mathbb{B} \to \mathbb{N}$ denotes how many problems a particular test taker can memorize and the value function $v : \mathbb{B} \to \mathbb{R}^+$ denotes how much utility the tester gets if that test taker failed the test (recall that the tester can only fail test takers who cannot solve all problems and he want to fail as many of them as possible). For simplicity, we write $p(B_l), m(B_l), v(B_l)$ as p_l, m_l, v_l when the context is clear. During the test, t out of n problems will be tested and a particular test taker will pass the test if all tested problems are either not in the unsolvable problem set or memorized.

The game only has two outcomes, pass or fail. The tester gets 0 utility if a test taker passes and he gets v_l utility if a type-l test taker fails. On the type-l test taker's side, he gets 0 utility if he passes and $-v_l$ if he fails. From the test taker's perspective, any utility function that has a higher pass utility than fail utility will give him the same incentive to pass the test at his best. We define them as $0, -v_l$ specifically for the zero-sum property of the game.

Our goal is to find the optimal strategy for the tester to maximize his utility under Stackelberg settings. That is, the tester firstly reveals his strategy about how t problems are picked to test, then the test taker chooses the best memorizing strategy. This corresponds to our confidential-free assumption where test takers know what problems might be tested, their answers, and how likely they are tested. The only constraint for test takers is that they cannot memorize too many (greater than m_l) problem answers.

2.1 General Linear Program (LP) Formulation

General 2-player Bayes Stackelberg games are NP-hard in terms of game matrix size so previous work used mixed integer LP (e.g. DOBSS [cite]) to solve them. We model test games as zero-sum so we can bypass the NP-hardness and use the following maximin LP to solve our 2-player zero-sum Bayes Stackelberg games in polynomial time with respect to the game matrix size:

$$\begin{aligned} \max \ & \sum_l p_l V_l \\ s.t. \ & (\forall l, \forall c_l \in C_l), \\ & \sum_r x_r u^l(r, c_l) \geq V_l \\ & \sum_r x_r = 1 \\ & (\forall r \in R) x_r \geq 0 \end{aligned} \tag{1}$$

where

- V_l: utility that the test taker can get from type l test takers
- C_l: action space of type l test takers, i.e. all combinations of problems that they can memorize.
- R: action space of the tester, i.e. all combinations of problems that he can test.
- x_r: probability to test problem set r

The dual of the above LP is:

$$\begin{aligned} \min \ & U \\ s.t. & (\forall r) U \geq \sum_{l, c_l} u^l(r, c_l) y_{l, c_l} \\ & (\forall l) \sum_{c_l} y_{l, c_l} = p_l \\ & (\forall l, c_l) y_{l, c_l} \geq 0 \end{aligned} \tag{2}$$

*These match the formatting instructions of IJCAI-07. The support of IJCAI, Inc. is acknowledged.

which is equivalent to

$$\begin{aligned} \min \ & \max_r(U_r) \\ s.t.(\forall r)U_r &= \sum_{l,c_l} u^l(r,c_l)y_{l,c_l} \\ (\forall l)\sum_{c_l} y_{l,c_l} &= p_l \\ (\forall l,c_l)y_{l,c_l} &\geq 0 \end{aligned} \tag{3}$$

The dual LP is as if that all types of test takers share the same goal to lower the tester's best testing utility ($\max_r(U_r)$ where U_r is the utility of testing r) as much as possible by memorizing problems and revealling their memorizing strategies y_{l,c_l} to the tester.

However, even if the LPs above are in polynomial size of the game matrix size (or the action space size), they are exponential to the input of our test games: the tester's action space is exponential to t (the number of problems to test) and the type-l test taker's action space is exponential to m_l (the number of problems he can memorize). So those LPs only give us polynomial algorithms when m_l, t are constant. Our next question is whether the test game in general can be solved in polynomial time in terms of the input size.

2.2 Hardness on Test Size

In order to formally characterize *test game*'s hardness, define

Definition 1 (OPTIMAL TEST STRATEGY Problem)**.** *Given a test game G and a value u, the* OPTIMAL TEST STRATEGY *problem is to decide whether the tester has a Stackelberg strategy with at least u utility.*

It's easy to show

Theorem 1. *Even if test takers cannot memorize any problems ($m_l = 0$),* OPTIMAL TEST STRATEGY *is NP-hard when test size t is non-constant.*

Proof. Reduce a VERTEX COVER instance to a OPTIMAL TEST STRATEGY instance as follows. Given a graph $G = (V, E)$, construct a test game G' with problem set $A = V$. For each edge $e = \{i, j\} \in E$, add one type of test takers l_e whose unsolvable problem set $B_{l_e} = e$. Let $p_{l_e} = 1/|E|, v_{l_e} = 1$ and $m_{l_e} = 0$ for all l_e. Graph G has a vertex cover of size k if and only if the tester has a strategy to test $t = k$ problems and gets $u = 1$ utility (i.e. all test takers will fail for sure) in test game G'. □

2.3 Hardness on Memorize Capacity

Theorem 2. *Even if test size is 2,* OPTIMAL TEST STRATEGY *is coNP-hard when memorize capacity is non-constant.*

Proof. We reduce INDEPENDENT SET instances to test games with test size $t = 2$ and ask the complementary question: is u the highest utility that the tester can get. That question has a yes answer if and only if the dual minimax LP [reference] has a feasible solution with objective at most u. For a graph $G = (V, E)$, let V be our test problem set A. Add $L = |E| + |V| + 2$ types of test takers as follows:

- For each edge $e = (i, j) \in E$, add one type of test takers l_e with $v_{l_e} = 1$, $B_{l_e} = \{i, j\}$ and $m_{l_e} = 0$.
- For each vertex i, add one type of test takers l_i with $v_{l_i} = d_G - d(i), B_{l_i} = \{i\}$ and $m_{l_i} = 0$. Here d_G is the maximum vertex degree in G and $d(i)$ is the degree of vertex i.
- Add one auxiliary type of test takers l_α with $v_{l_\alpha} = \alpha\varepsilon, m_{l_\alpha} = 2$ and $B_{l_\alpha} = V$ where $\alpha = \binom{|V|}{2} - |E| - \binom{k}{2}$ and $\varepsilon \ll 1$.
- Finally, add one type of test takers l_k with $v_{l_k} = \varepsilon, B_{l_k} = V$ and $m_{l_k} = k$.

Let all types of test takers occur with uniform probability $p = 1/L$. Then we show that deciding whether $u = (2d_G + a \cdot \varepsilon)/L$ is an upper-bound for the tester's utility, or equivalently whether we can find a dual solution with objective as low as $u = (2d_G + a \cdot \varepsilon)/L$ in our dual minimax LP, is equivalent to finding a size-k independent set in G. Recall that a dual solution is a strategy for test takers to memorize problems. If no one memorizes, the tester's utility U_r to test a problem set r is $U_r = (2d_G + a \cdot \varepsilon + \varepsilon)/L$ if $r \notin E$ and $U_r = (2d_G - 1 + a \cdot \varepsilon + \varepsilon)/L$ if $r \in E$. To lower the objective, test takers l_α, l_k should memorize problem sets that bring the maximum of U_r down. No matter what the strategy is (as long as they only memorize unsolvable problems), $\sum_r U_r$ is a constant. And because U_r for $r \notin E$ is much larger than U_r for $r \in E$ as $\varepsilon \ll 1$, it's better to only decrease U_r for $r \notin E$. The only way to do that is to let l_α test takers only memorize pairs of problems that are not edges and let l_k test takers only memorize size-k indepedent sets of problems. If there's a size-k independent set, let l_k test takers memorize one size-k independent set all the time and let l_α test takers memorize all pairs of problems that are not covered by that independent set with uniform probability. That will make $U_r = (2d_G + a \cdot \varepsilon)/L$ for $r \notin E$ and $U_r = (2d_G - 1 + a \cdot \varepsilon + \varepsilon)/L$ for $r \in E$. This is the best they can achieve and this can only be achieved when a size-k independent set exists in G.

□

2.4 Summary on General Test Games

Theorem 3. *The* OPTIMAL TEST STRATEGY *for a* test game *can be computed in polynomial time when test size and memorize capacity are constant. It's NP-complete when test size is non-constant and coNP-complete when memorize capacity is non-constant. Hence it's both NP-hard and coNP-hard when both test size and memorize capacity are non-constant.*

Proof. When test size and memorize capacity are both non-constant, OPTIMAL TEST STRATEGY is both NP-hard and coNP-hard by Theorem 1 and 2. LP (1) shows that OPTIMAL TEST STRATEGY can be computed in polynomial time when test size and memorize capacity are constant. When memorize capacity is constant or test size is constant, either the primal LP (1) or the dual LP (3) has constant number of constraint. Such LP has a polynomial size optimal solution even if the number of variables is exponential. That optimal solution's feasibility and objective can be checked in polynomial time. Therefore OPTIMAL TEST STRATEGY is in NP when memorize capacity is constant and it is in coNP when test size

is constant. Hence they are NP-complete and coNP-complete respectively by Theorem 1 and 2. □

[HARDNESS TABLE]

3 One-Problem Test Game

We showed the hardness of test games when either one of test size or memorize capacity is non-constant. When the test size is non-constant, there is little we can improve because it is NP-hard even if the memorize capacity is zero. When the memorize capacity is non-constant, our proof for Theorem 2 only showed that NP-hardness for testing 2 problems. Therefore an efficient algorithm to compute optimal one-problem test may exist.

We firstly investigated such tests by conducting experiments using LP (1). Surprisingly, no matter how input varies, the strategy we get always has the following format: among a subset T of problems A, test any one of them with uniform probability. That is counter-intuitive as some problems in T seem to be obviously more superior than others so intuitively they should be tested with more probability. For example, when n problems can be sorted by difficulty, a test taker who can solve a harder problem can always solve an easier problem. In that case, it seems that the hardest problem should be tested strictly more likely than the second hardest problem. But in most cases, the optimal strategy will test one of them with equal probability.

[EXAMPLE TABLE]

Nevertheless, such uniform test property looks to be a good news for us to find an efficient algorithm for solving one-problem test. Inspired by that uniform test property, we present an algorithm to compute optimal strategies and use it to prove uniform test property consequently.

The algorithm consists of two parts. It firstly uses binary search and max flow to determine the optimal tester's utility U. This part corresponds to the following LP formulation (let U_a^0 be the utility of testing a without memorizing as we defined in Algorithm 3):

$$\begin{aligned} &\min\ U \qquad (4)\\ &s.t.\ (\forall a \in A)\ U \geq U_a^0 - \sum_{B_l \ni a} v_l z_{l,a}\\ &\qquad (\forall l)\ \sum_{a \in B_l} z_{l,a} \leq m_l p_l\\ &\qquad (\forall l, a)\ 0 \leq z_{l,a} \leq p_l \end{aligned}$$

This LP modifies LP (2) by:

1. Change joint probability y_{l,c_l} to marginal probability $z_{l,a}$ of memorizing one problem a. We did this modification because marginal probability is more meaningful for one-problem test and we can always restore a joint probability from a marginal probability[reference].
2. Relax equality $\sum_{a \in A} z_{l,a} = m_l p_l$ to inequality $\sum_{a \in A} \leq m_l p_l$. We can do this because $v_l \geq 0$ (thus $u^l, V_l \geq 0$) and this modification is essentially adding $V_l \geq 0$ to the primal LP (1).

Algorithm 1 Compute the optimal one-problem test strategy with an acceptable objective error ϵ (for binary search)

function OPTIMALONEPROBLEMTEST($A, \mathbb{B}, m, v, p$)
  for all $a \in A$ **do** ▷ Utility without memorizing
    $U_a^0 \leftarrow \sum_{B_l \ni a} v_l p_l$
  end for
  $U_{lower} \leftarrow 0,\ U_{upper} \leftarrow \max_{a \in A} U_a^0$
  while $U_{upper} - U_{lower} > \epsilon$ **do** ▷ Binary search
    $U \leftarrow (U_{upper} + U_{lower})/2$ ▷ Objective utility
    $V \leftarrow \{s\} \cup \mathbb{B} \cup A \cup \{t\}$ ▷ Vertex set
    $E \leftarrow \emptyset$ ▷ Initialize edge set
    for all $a \in A$ **do**
      if $U_a^0 > U$ **then**
        $E \leftarrow E \cup \{(a, t)\}$
        $c_{(a,t)} \leftarrow U_a^0 - U$ ▷ Edge capacity
      end if
    end for
    for all $B_l \in \mathbb{B}$ **do**
      $E \leftarrow E \cup \{B_l\} \times B_l$
      $c_{\{B_l\} \times B_l} \leftarrow p_l v_l$
      $E \leftarrow E \cup \{(s, B_l)\}$
      $c_{(s,B_l)} \leftarrow p_l m_l v_l$
    end for
    $f \leftarrow$ MAXFLOW$(G = (V, E), c)$
    if f saturates all edges in $A \times \{t\}$ **then**
      $U_{upper} \leftarrow U$
    else
      $U_{lower} \leftarrow U$
    end if
  end while
  $T \leftarrow \{a \mid U_a^0 \geq U\}$
  $Q \leftarrow$ all B_l such that (s, B_l) is not saturated in f
  $G(B_l) \leftarrow c_{(s,B_l)} - f_{(s,B_l)}$ ▷ Potential flow
  while Q has an unmarked element **do**
    $B_l \leftarrow$ get and mark an unmarked element from Q
    for all $a \in B_l$ and (B_l, a) is not saturated in f **do**
      $T \leftarrow T \backslash \{a\}$
      for all $B_{l'} \ni a$ and $f_{(B_{l'},a)} > 0$ **do**
        $G(B_{l'}) \leftarrow \min(f_{(B_{l'},a)}, \frac{G(B_l)}{|B_l|L})$
        Push $B_{l'}$ into Q
      end for
    end for
  end while
  return Strategy that test T uniformly
end function

With a given U, we use max flow to check its feasibility. Intuitively, a U is feasible if and only if there is a strategy $z_{l,a}$ such that for every a where $U_a^0 > U$, that strategy $z_{l,a}$ decrease it down to U. This is achievable if and only if the max flow can saturate all edges from A to t. The max flow f can also compute that strategy by setting $z_{l,a} = f_{(B_l,a)}/v_l$.

The second part of the algorithm is to compute the test strategy (the optimal primal solution) from the memorize strategy (the optimal dual solution) we computed in the first part by max flow. We prove that our test strategy is optimal by complementrary slackness theorem, which in our context is essentially: 1) the tester only tests problems (non-zero primal variables) that gives him best utility (tight dual constraints); 2) the test taker only memorizes problems (non-zero dual variables) that gives him best utility (tight primal constraints).[1] We will need two more lemmas as follows to use the complementary slackness theorem.

Lemma 1. *Let $z_{l,a} = f_{(B_l,a)}/v_l$ be the dual solution computed by max flow and T be the final subset that Algorithm 3 returns. Then for each type of test takers B_l, either $\sum_{a\in B_l} z_{l,a} = m_l p_l$ (they memorize a subset of T all the time) or $\forall a \in B_l \cap T, z_{l,a} = p_l$ (they memorize unsolvable problems in T all the time so they will pass the test for sure).*

Proof. If there is a B_l such that $\sum_{a\in B_l} z_{l,a} < m_l p_l$, either f_{s,B_l} is not saturated or there is an $a \in B_l$ with $f_{(B_l,a)} > 0$ that is deleted from T later on line 35 of Algorithm 3. In any case, B_l will be pushed into Q and all problems in B_l that does not satisfy $z_{l,a} = p_l$ will be deleted from T. Hence $\forall a \in B_l \cap T, z_{l,a} = p_l$ must hold. □

Lemma 2. *T returned by Algorithm 3 is non-empty.*

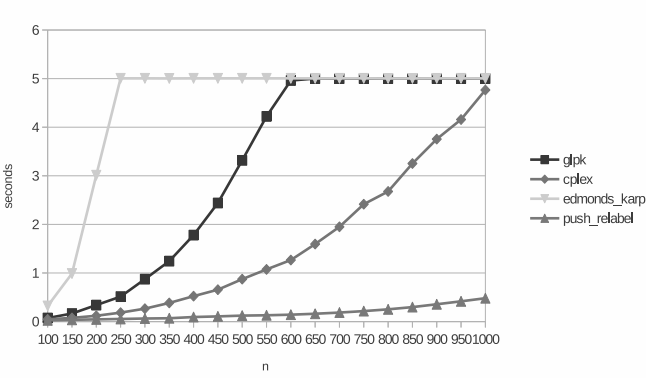
Proof. T is non-empty before we delete problems from T on line 35 of Algorithm 3. Denote that initial T as T_0. If all problems in T_0 are deleted, then for every a in T_0 there is a B_l that is in Q and $f_{(B_l,a)}$ is not saturated. Note that for every B_l in Q, it has a positive potential flow $G(B_l)$ that can additionally flow out of vertex B_l, either because (s, B_l) is not saturated (on line 31) or because there is a positive flow (B_l, a) that can be reduced (on line 37). This means that we can bring U down by sending these potential flow to every vertex in T_0, a contradiction to that U is minimized. □

Theorem 4. *Uniformly testing T returned by Algorithm 3 is optimal.*

Proof. First of all, T is non-empty by Lemma 2 so we can test T uniformly. Then since the max flow makes U_a (the utility to test a problem a) decrease to U if $U_a^0 > U$ and we only pick T from problems that $U_a^0 > U$, every problem that we might test gives the tester best utility. That is the first condition for the complementary slackness theorem. Finally, Lemma 1 show that one type of test takers either pass the test for sure or memorize a subset in T all the time which is optimal for uniform test, the second condition of complementary slackness theorem holds. □

[1] This is essentially saying that in a zero-sum game, a minimax, or maximin state is reached, if and only if the Nash equilibrium is reached.

Figure 1: Average running time for different one-problem test algorithms over the number of problems n. For each n, we set the other three parameters $L = n, m_{max} = \lfloor\sqrt{n}\rfloor$ and $b_{max} = 2m_{max}$ (see Table 1).



4 Experiment

In this section, we conduct experiments to see how different algorithms scale on test games.

EXPERIMENTS FOR MULTI-PROBLEM TEST?

For one-problem test, our original LP (1) is exponential to the memorize capacity so we only compare modified LP (4) and max flow based Algorithm 3. During each experiment, we specify four parameters n, L, m_{max}, b_{max}, the number of problems, the number of test taker types, the maximum memorize capacity and the maximum size of B_l. Given those parameters, a test game instance is generated as follows: for each $1 \le l \le L$, generate m_l uniformly from 1 to m_{max}, generate $|B_l|$ uniformly from m_l to b_{max}, generate B_l by selecting $|B_l|$ elements from A uniformly and generate $w_l = v_l p_l$ (weight of type-l) from $[0, 1]$ uniformly. Here we only generate w_l because v_l and p_l always appear together as $v_l p_l$.

For each parameter setting, we generate 5 test game instances and compute the average running time for each algorithm. During each run, we set timeout to be 5 seconds. For the LP, we ran it on glpk and cplex. For the max flow, we tried both Edmonds-Karp and Push-Relabel implementation in C++ boost library.

In our benchmark, we increase n and set the remaining three parameters L, m_{max}, b_{max} as following: $L = n, m_{max} = \lfloor\sqrt{n}\rfloor$ and $b_{max} = 2m_{max}$. Note that our L, b_{max} are quite small (they can be as large as $b_{max} = n/2, L = \binom{n}{b_{max}}$). That's becuase for large L, b_{max}, our uniform random input will almost always make the trivial strategy to uniformly test among all n problems optimal. So we make them small to avoid such trivial solutions. The realized parameter settings are shown in Table 1 (we show them in the step of 50 for the sake of space). The benchmark is shown in Figure 1.

Table 1: Parameter settings for the benchmark

n	L	m_{max}	b_{max}
100	100	10	20
200	200	14	28
300	300	17	34
400	400	20	40
500	500	22	44
600	600	24	48
700	700	26	52
800	800	28	56
900	900	30	60
1000	1000	31	62

5 Conclusion

Acknowledgments

The preparation of these instructions and the LaTeX and BibTeX files that implement them was supported by Schlumberger Palo Alto Research, AT&T Bell Laboratories, and Morgan Kaufmann Publishers. Preparation of the Microsoft Word file was supported by IJCAI. An early version of this document was created by Shirley Jowell and Peter F. Patel-Schneider. It was subsequently modified by Jennifer Ballentine and Thomas Dean, Bernhard Nebel, and Daniel Pagenstecher. These instructions are the same as the ones for IJCAI–05, prepared by Kurt Steinkraus, Massachusetts Institute of Technology, Computer Science and Artificial Intelligence Lab.

A LaTeX and Word Style Files

The LaTeX and Word style files are available on the IJCAI–13 website, `http://www.ijcai-13.org/`. These style files implement the formatting instructions in this document.

The LaTeX files are `ijcai13.sty` and `ijcai13.tex`, and the BibTeX files are `named.bst` and `ijcai13.bib`. The LaTeX style file is for version 2e of LaTeX, and the BibTeX style file is for version 0.99c of BibTeX (*not* version 0.98i). The `ijcai13.sty` file is the same as the `ijcai07.sty` file used for IJCAI–07.

The Microsoft Word style file consists of a single file, `ijcai13.doc`. This template is the same as the one used for IJCAI–07.

These Microsoft Word and LaTeX files contain the source of the present document and may serve as a formatting sample.

Further information on using these styles for the preparation of papers for IJCAI–13 can be obtained by contacting `pcchair13@ijcai.org`.

References