1 Two-player zero-sum game

We model the problem as a two-person zero-sum game (a matrix game). The first player is called the *searcher* or "the guy" (T. Szabó), and his task is to find the sink. The other player, depending on the point of view, is called Nature, chance, the *adversary*, or the gal.

The searcher has the first turn. The two players play alternate moves. The searcher moves by picking a vertex of the 3-cube, and the adversary responds by giving the out-map at the chosen vertex. The game is over when adversary announces that the searcher has chosen the sink. The "payoff" that the searcher has to pay is the number of turns that he has played.

The *game tree* T encodes all possible plays of the game. It consists of alternating levels when it is the searcher's turn and the adversary's turn. A searcher node has up to eight children, numbered from 0 to 7, corresponding to the eight vertices of the cube. Since it does not make sense to ask the same vertex twice, deeper search nodes have fewer children, and in particular, the tree is finite.

An adversary node has also up to eight children, numbered from A to H, corresponding to the possible out-maps. Not all out-maps may be possible, they have to be consistent with the answers given so far and the existence of a u.s.o. that has all those out-maps. The leaves of the tree are the searcher nodes whose label indicates the out-map $\emptyset$.

A node in the tree can be identified uniquely by the string of labels leading to it, like S0A5B4A for a searcher node, or S0A5B4A3 for an adversary node. The root is called S.

Terminology. The search tree has *nodes* and *arcs*, whereas the 3-cube has *vertices* and *edges*.

Strategies. A *pure strategy* S (in the language of game theory) for the searcher is simply a selection of one outgoing arc for each searcher node in the tree. (Depending on arcs chosen close to the root, some nodes deeper in the tree become unreachable, and it becomes irrelevant which successor the searcher chooses for those nodes. However, for simplicity, we define a strategy as above.)

A pure strategy is a deterministic algorithm.

A pure strategy A for the adversary is simply a choice of one of the 744 u.s.o.s.

The payoff is a function $f(S, A)$ of the two strategies.

When a player plays a *mixed strategy*, she chooses randomly among her pure strategies according to some specified probabilities.

The expected payoff is given by

$$f((p_S), (q_A)) := \sum_{S \in \mathcal{S}} \sum_{A \in \mathcal{A}} p_S q_A f(S, A), \tag{1}$$

where $(p_S)_{S \in \mathcal{S}}$ and $(q_A)_{A \in \mathcal{A}}$ are the probability vectors that define the mixed strategies.

From game theory we know the following.

Theorem 1 *There is a unique value*

$$v = f((\hat{p}_S), (\hat{q}_A)),$$

called the value *of the game, and two optimal mixed strategies* $(\hat{p}_S)$ *and* $(\hat{q}_A)$ *for the two players, with the following property: For all mixed strategies* (p_S) *and* (q_A)*, we have*

$$f((p_S), (\hat{q}_A)) \geq v \geq f((\hat{p}_S), (q_A)).$$

This means in particular that the pair of strategies $(\hat{p}_S), (\hat{q}_A)$ is *stable* in the following sense: If the adversary plays according to the strategy $(\hat{q}_A)$, the searcher cannot gain by choosing a different strategy than $(\hat{p}_S)$, even if he knows the probabilities $\hat{q}_A$. The same statement is true for the adversary with respect to an optimal strategy $(\hat{p}_S)$ of the searcher, but this fact is less interesting for us.

An optimal strategy is defined by the above properties: a strategy $(\hat{p}_S)$ for the first player is *optimal* if

$$\max_{(q_A)} f((\hat{p}_S), (q_A)) = v,$$

and similarly for the second player. (in fact, "$\geq$" holds for any strategy (p_S))

The concept of a mixed strategy is somehow related to the notion of a randomized searching algorithm. We can define this in the following, more natural way. A *randomized strategy* R is an assignment of a probability distribution to the outgoing arcs of every searcher node in the game tree. The searcher follows this algorithm by making at every node at which he arrives during the game a new, independent random choice.

There is a conceptual difference to mixed strategy as defined above. In a mixed strategy, the searcher makes a single random choice in the beginning, and then follows that (deterministic) strategy.

However, we shall show that this difference has no consequence for the game. Intuitively, random choices in different branches of the tree have nothing to do with each other; thus, any dependence between choices in different branches of the tree have no effect on the outcome. We have introduced mixed strategies because they allow us to apply the powerful basic results of game theory. Randomized strategies are the more natural setting in which one can think of a randomized algorithm.

Lemma 1 *For any mixed strategy* (p_S)*, there is an equivalent randomized strategy* R*; and for any randomized strategy* R*, there is an equivalent mixed strategy* (p_S)*, in the sense, that for any (pure or mixed) strategy* A *of the adversary, the expected payoffs are equal:*

$$\sum_{S \in \mathcal{S}} p_S f(S, A) = g(R, A)$$

$g(R, A)$ for a randomized strategy R is defined as follows. For every leaf node $\ell \in T$, let $P(\ell, R)$ denote the probability of reaching node ℓ under the randomized strategy R, assuming that the adversary's moves always move towards ℓ. This is the product of the edge probabilities on the path from the root to ℓ. Let $L(A) \subseteq T$ be the set of leaves ℓ that can be reached when the adversary plays strategy A, assuming that the searcher's moves always move towards ℓ. Then

$$g(R, A) = \sum_{\ell \in L(A)} P(\ell, R)[1 + d(\ell)/2]$$

where $d(\ell)$ is the depth of ℓ in the tree.

With analogous definitions of $P(\ell, R)$ and $N(A)$ for *all* searcher nodes of the tree, we can also write

$$g(R, A) = \sum_{\ell \in N(A)} P(\ell, R) \tag{2}$$

Proof. (sketch) To get from R to (p_A) we simply take the product distribution of all random choices at all searcher nodes. In other words, we make all random choices in advance before we start.

To get from (p_A) to R, we take conditional probabilities. Let a be a successor node of a searcher node ℓ. Then R chooses a with probability

$$\frac{P(a, R)}{P(\ell, R)}$$

provided that $P(\ell, R) \neq 0$. If $P(\ell, R) = 0$, than the distribution among the children of ℓ can be arbitrary.

... a little computation shows that the two quantities are equal. ... QED.

Worst-case bounds for deterministic algorithms. For investigating the worst-case payoff of a deterministic algorithm for the searcher, we may endow the adversary with an extended set of strategies: The adversary does not have to choose her u.s.o. beforehand, but she may choose a child for every adversary node as she likes, subject only to the condition that the answer is consistent with the previous answers that have led to this node.

In this setting the strategies for the adversary are defined similarly as for the searcher, and the evaluation of the value of the game is simply a min-max computation in the tree from the leaves towards the root.

This is then a different game from the one we are considering.

2 Modeling as a linear program

We model a randomized strategy as a linear programming problem as follows. In each searcher node ℓ, we have a set of probabilities $p(\ell, a)$ adding up to 1. The probabilities $P(R, \ell)$ in (2) are products of these probabilities.

In order to get linear relations, we use the probabilities $P(R, \ell)$ as variables. More precisely, we introduce a variable p_a for every adversary node a. This quantity represents the probability of reaching this node, assuming that the adversary's moves always move towards a. For every searcher node ℓ, we introduce the following inequality: Let b be the parent node of ℓ, and let $K(\ell)$ be the set of children of ℓ, then we have

$$p_b = \sum_{a \in K(\ell)} p_a \tag{3}$$

The root node r does not have a parent; for this node we replace the left-hand side by 1:

$$1 = \sum_{a \in K(r)} p_a \tag{4}$$

The expected payoff (2) can be written as

$$g(R, A) = \sum_{a \in N'(A)} p_a \tag{5}$$

We want to minimize the maximum of this over all u.s.o.'s A, so we write the constraints as

$$v \geq \sum_{a \in N'(A)} p_a, \text{ for all } A \in \mathcal{A} \tag{6}$$

Then the linear programming problem is a follows: minimize v subject to the constraints (3), (4), (6), and the nonnegativity constraints $p_a \geq 0$ for all $a \in N'$. The optimal value v is the value of the game. The optimal dual variables of the inequalities (6) are a set optimal probabilities $\hat{q}(A)$ for the optimal mixed strategy for the adversary.

3 Simplifications. Using Symmetry

It is intuitively obvious that the searcher should make his random choices symmetrical with respect to the symmetries of the cube. For example, the first vertex should clearly be chosen uniformly at random from all 8 vertices of the cube. This means that the optimal strategy will consist of 8 subtrees that are essentially identical. To reduce this duplication, we employ a trick.

Before starting the algorithm, the searcher applies a random automorphism π of the given cube. This means that the first vertex to be queried can be vertex 0, without loss of generality. (Or rather, the vertex which has been renamed as vertex 0.)

For the dual variables, i. e., for the probabilities chosen by the adversary, this means that all u.s.o.'s that are in the same automorphism class are chosen with the same probability. So the constraints (6) can be replaced by

$$v \geq \frac{1}{24} \sum_{\pi \in S} \sum_{a \in N'(\pi(A))} p_a, \text{ for all } [A] \in \mathcal{A}_0 \tag{7}$$

$\mathcal{A}_0$ denotes the set of 19 isomorphism classes of u.s.o.'s of the 3-cube, and S is the set of 24 automorphisms of the 3-cube. (Some isomorphism classes contain fewer than 24 u.s.o.'s because not all $\pi(A)$ are different. In those cases the sum over all $\pi(A)$ will in practice be reduced to all distinct u.s.o.'s in one class, and the denominator 24 adapted accordingly.)

As a consequence, if the partial answers to an initial set of queries give some information which is symmetric with respect to some subgroup G of symmetries of the cube, then one needs to consider only one query vertex in each set of query vertices that lie in the same orbit of G. The choice of 0 as the initial vertex is one instance of this rule. I have only carried out this simplification to the second round: I the first out-map at vertex 0 is 3-out, then all query vertices at the same distance from 0 are equivalent. So I only need to query 1 (from the set 1,2,4), 3 (from the set 3,5,6), and 7. I the first out-map is 2-out and 1-in, for example (0,1), (0,2), and (4,0), then 1 and 2 are equivalent, and so are 5 and 6. So 2 out of 7 choices are eliminated, and the same is true for the case 1-out and 2-in.

This simplification was sufficient to make the problem small enough to be computed with reasonable effort.

One could also require that, of the three out-maps with 2-out and 1-in in the first round, only one branch is considered, the others being identical. The model would have to be adapted. I haven't done this simplification. (with the unpleasant consequence that the optimal strategy is difficult to recognize in the form in which the algorithm produces it.)

Correctness proof for symmetry. "Symmetry" here means that all isomorphic u.s.o.'s have equal probabilities.

The searcher may enforce symmetry by applying a random automorphism before he starts. This means that symmetry cannot increase the value of the game. The adversary can also enforce symmetry, by simply choosing all probabilities of isomorphic u.s.o.'s equal. This means that symmetry cannot decrease the value of the game.

Here is a more formal argument.

Lemma 2 *The function*

$$h((q_A)) := \min_R g(R, (q_A)) \tag{8}$$

is a concave function of (q_A) over set the of all probability distributions over the u.s.o.'s A. Moreover, it is symmetric under automorphisms of the cube.

Proof. $g(R, (q_A)) = g(\pi(R), (q_\pi(A)))$. Define $\pi(R)$.

Corollary 3 *Restricting $q(A)$ to symmetric distributions does not hurt the adversary.*

It is clear that it does not help the adversary.

Computing the optimal strategy for a given set of probabilities. If some probability distribution q_A for the adversary is given, an optimal strategy against *that* mixed strategy (i. e., the function $h((q_A))$ given in (8)) can be computed bottom-up in the tree by dynamic programming. For each adversary node, one can compute the expected cost as the weighted average of its children, according to the weight implied by the probabilities q_A. For each searcher node, one selects the successor of minimum expected cost and adds 1 to this cost. This defines also the optimal strategy, or even the set of all optimal strategies.

The bottom-up procedure defines a subtree of the game tree. *Any* pure or mixed or randomized strategy in that subtree is optimal against (q_A). If the given probabilities q_A are an optimal strategy for the adversary that has been computed by linear programming, some searcher nodes will in general have several optimal successors. One can perhaps check whether the optimal solution is unique, or find the set of all optimal strategies for the game, by looking at the linear program for that smaller subtree, but without the simplifications due to symmetry.

Representation In the recursive procedure for generating the game tree, I have chosen to keep for each node $\ell \in T$ the *set* of all u.s.o.'s A that are compatible with the path leading to that vertex. This set represents the partial information that has been acquired in the process. The edge orientations are not stored explicitly.

Shortcut: If the partial knowledge allows the sink to be identified, I will directly query the sink.

A game graph instead of a game tree? If two different paths in the tree lead to the same partial information, for example if two nodes are reached by querying the same two vertices in different order[1] (and getting the same answers), one might consider to merge these two nodes into one node. Indeed, one can prove(???)[2] that there is an optimal randomized algorithm which treats the two branches identically. (This follows from the bottom-up procedure for computing the optimal strategy.)

This however destroys the possibility to model the problem as a linear problem.

Questions. Is the optimal primal/dual solution unique? Is the query vertex always a possible sink?

[1]interesting question: does this ever happen in an optimal strategy?
[2]Ist das wahr?