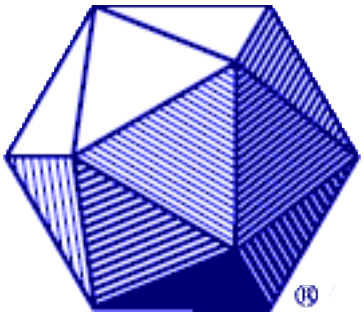

Finding Saddlepoints of Two-Person, Zero Sum Games

DONNA CRYSTAL LLEWELLYN[1] AND CRAIG TOVEY[2],
School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta

MICHAEL TRICK[3], *Carnegie-Mellon University*

DONNA CRYSTAL LLEWELLYN: During my undergraduate years at Swarthmore College, I became interested in combinatorics and graph theory and their applications. Following these interests, I went to Stanford University and then Cornell University, receiving my doctorate in operations research in 1984 under the direction of L. E. Trotter, Jr. I had the honor of being awarded a National Science Foundation Mathematical Sciences Postdoctorate Fellowship, which I used to pursue my research interests at the University of Bonn, West Germany. Currently, I am an assistant professor at Georgia Institute of Technology in the Department of Industrial and Systems Engineering.

CRAIG TOVEY: As an undergraduate in applied mathematics at Harvard College, I wrote a thesis under the direction of Christos Papadimitriou. Since then my principal research interests have remained in the area of design and analysis of algorithms, particularly probabilistic analysis of combinatorial algorithms and computational complexity. I received my Ph.D. in operations research from Stanford University in 1981, supervised by George Dantzig. Since then I have taught and done research at Georgia Tech, where I am now an associate professor.

MICHAEL TRICK: I received my Ph.D. in Industrial and Systems Engineering from Georgia Tech in 1987. I spent the 1987–88 academic year at the Institute for Mathematics and Its Applications in Minneapolis as a postdoctoral fellow. My research interests include integer programming, design and analysis of algorithms, and applications of operations research in the biological and social sciences. I am now affiliated with the Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh.

1. Introduction. A two-player, zero sum game, as defined by von Neumann and Morgenstern in their classical work [**13**], is completely specified by its payoff matrix. Let player I have m possible "moves" or "plays" (called *pure strategies*) and player II have n pure strategies. Suppose that player I pays player II a_{ij} if player I chooses his ith strategy and player II chooses her jth. Then the *payoff matrix* of the game is the $m \times n$ matrix $A = [a_{ij}]$.

[1]Supported by NSF Postdoctoral Fellowship DMS84-14104
[2]Supported by NSF Grant ECS-8451032
[3]Supported by ONR Grant N00014-86-K-0173

A *saddlepoint* (SP) of a matrix A is an entry a_{ij}, where a_{ij} is the largest element in row i and the smallest in column j; that is

$$a_{ij} \geqslant a_{ik} \quad \text{for all } k$$

$$a_{ij} \leqslant a_{kj} \quad \text{for all } k.$$

If both inequalities are strict, then we call a_{ij} a *strict saddlepoint* (strict SP).

If there exists a strict saddlepoint of the payoff matrix, then it is the unique optimum solution to the game. A saddlepoint of the payoff matrix (if it exists) corresponds to a (not necessarily unique) optimum solution. Notice that if all of the entries of the payoff matrix are distinct, then a saddlepoint is a strict saddlepoint. (In general, finding the optimal solution vectors for the two players is equivalent to solving an $m \times n$ linear programming problem [3].)

While these properties of saddlepoints are well known, it appears that the question of how to find these solutions efficiently has not been carefully studied. For instance, in [4] it is stated that "a [saddlepoint] is easy to find, even in a very large matrix." Other standard texts, such as [1], [2], [5]–[10], [12] and [14]–[17] also mention that saddlepoints are easy to locate but fail to discuss finding an efficient algorithm.

The purpose of this article is to show that a strict saddlepoint, when it exists, can be found very quickly. In fact, we give an algorithm that requires looking at only $\simeq n^{0.58}m$ entries of the payoff matrix (where $m \geqslant n$). Hence as the number of strategies increases, *the proportion of the payoff matrix examined approaches zero.* This algorithm also determines when a matrix has no strict SP. Further, we will prove that finding a (nonstrict) saddlepoint requires examination of the entire payoff matrix.

Given a matrix, we refer to the problem of finding a (strict) saddlepoint or determining that one does not exist as the (strict) SP problem. For ease of notation, we will denote $\log_2 x$ by $\lg x$.

We distinguish between two different types of computations. One kind is the ordinary arithmetic operation (i.e., comparison, addition, multiplication and division by integers). The other kind is the identification of the value of a matrix entry; this might involve a lengthy function evaluation. Therefore, we adopt the following model of computation. Assume that each matrix query contributes α units to the running time and each arithmetic operation requires β time; where α and β are some constants.

II. Finding Strict Saddlepoints. We motivate our algorithm for finding a strict saddlepoint with a couple of observations.

Observation 1. If two entries of A, a_{ij} and a_{kl} are known, then at least one of them can be eliminated as a possible strict SP with a single matrix look-up and a constant number of arithmetic operations.

Proof. Clearly this follows if either $i = k$ or $j = l$. Hence assume that neither of these holds. Without loss of generality let $a_{ij} \leqslant a_{kl}$. Look up a_{il}. Clearly, if a_{ij} is a strict SP then $a_{ij} > a_{il}$. Further, if a_{kl} is a strict SP then $a_{il} > a_{kl}$. Since $a_{ij} \leqslant a_{kl}$, it is clear we can not have $a_{ij} > a_{il} > a_{kl}$. Hence, knowing a_{il} will eliminate either a_{ij} or a_{kl} or both from consideration.

Notice that in the proof of observation 1 it is assumed that $a_{ij} \leqslant a_{kl}$. Hence, the entry a_{kj} cannot be a strict SP since $a_{ij} \leqslant a_{kl}$ implies that it is impossible to have $a_{ij} > a_{kj} > a_{kl}$. This leads to our second observation.

Observation 2. If the diagonal terms of a square matrix, A, are known and are such that $a_{11} \leqslant a_{22} \leqslant \cdots \leqslant a_{nn}$, then no term below the diagonal can be a strict SP.

Note that to apply observation 2 we will first have to rearrange the rows and columns of A so that the diagonal is ordered (which will not affect the existence or nonexistence of a strict SP). Our algorithm uses observations 1 and 2 together with proper manipulation of rearrangement pointers. The idea of the algorithm is illustrated in FIGURES 1 and 2.

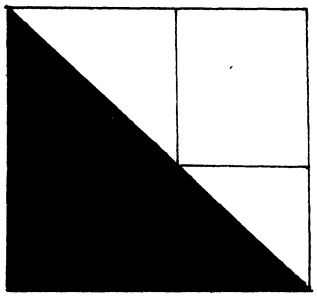

FIG. 1

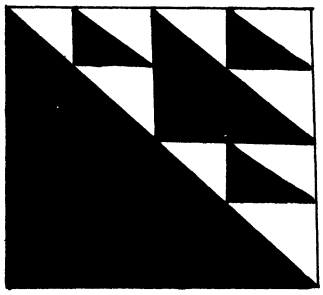

FIG. 2

The main diagonal is looked up and then sorted to eliminate the subdiagonal portion of the matrix from consideration. The remaining region (i.e., the upper triangle) is divided into two triangles and one square which are studied recursively (see FIGURE 1 for the first step of this recursion and FIGURE 2 for the next two steps). These pieces are then put back together using observation 1. In the diagrams, the shaded regions are the parts of the matrix that have been ruled out by our observations and hence do not have to be searched.

We now state the algorithm precisely.

Procedure Square
Input: $A, n, r(1), \ldots, r(n), c(1), \ldots, c(n)$
Output: candidate

This procedure takes as input a matrix A, and a subset of n of its rows and a subset of n of its columns. These rows and columns define an $n \times n$ submatrix $\bar{A}$ of A. The procedure then returns candidate, a location in $\bar{A}$ which might be a strict SP of A. No other element of $\bar{A}$ is a strict SP of A.

Step 0: If $n = 1$, return candidate $= (r(1), c(1))$
Step 1: Look up and sort $A(r(i), c(i))$, $i = 1, \ldots, n$ in ascending order of size.

Suppose σ is the permutation that gives this sorting, i.e., suppose that the set $\{A(r(\sigma(i))), c(\sigma(i))\}_{i=1}^{n}$ is an increasing sequence.

Set

$$r(i) = r(\sigma(i))$$
$$c(i) = c(\sigma(i)) \quad \text{for } i = 1, \ldots, n.$$

Step 2: Call Triangle ($\overline{A}$, n, $r(1), \ldots, r(n), c(1), \ldots, c(n)$; candidate)
Step 3: If candidate = $\varnothing$, return $\varnothing$. Otherwise, take σ^{-1} (candidate) and return it.

Procedure Triangle
Input: A, n, $r(1), \ldots, r(n), c(1), \ldots, c(n)$
Output: candidate

This procedure takes as input a matrix A and a subset of n of its rows and a subset of n of its columns. These rows and columns define an upper triangle submatrix T of A. The procedure then returns a location in T which might be a strict SP of A. No other element of T is a strict SP of A.

Step 0: If $n = 1$, return candidate = $(r(1), c(1))$
Step 1: Call Triangle ($A, \lfloor \frac{n}{2} \rfloor, r(1), \ldots, r(\lfloor \frac{n}{2} \rfloor), c(1), \ldots, c(\lfloor \frac{n}{2} \rfloor)$; candidate1)
Step 2: Call Square ($A, \lceil \frac{n}{2} \rceil, r(1), \ldots, r(\lceil \frac{n}{2} \rceil), c(\lceil \frac{n}{2} \rceil), \ldots, c(n)$; candidate2)
Step 3: Call Triangle ($A, \lfloor \frac{n}{2} \rfloor), r(1 + \lceil \frac{n}{2} \rceil, \ldots, r(n), c(1 + \lceil \frac{n}{2} \rceil), \ldots, c(n)$; candidate3)
Step 4: Using Observation 1, discard at least two of the candidates. If candidate$i$ remains ($i = 1$, 2 or 3), let candidate equal candidate$i$. Otherwise, candidate = $\varnothing$.
Step 5: Return candidate.

Program Strict Saddlepoint
This program solves the strict saddlepoint problem on an $n \times n$ matrix A.
Step 1: Call Square $(n, 1, \ldots, n, 1, \ldots, n$; candidate)
Step 2: If candidate $\neq \varnothing$, check to see if candidate is a strict SP; if so, then report the strict SP. Otherwise, then conclude there does not exist a strict SP.

PROPOSITION. *Program Strict Saddlepoint solves the strict saddlepoint problem for a square matrix.*

Proof. Observation 2 implies that candidate1, candidate2, and candidate3 are the only possible positions of a strict saddlepoint in the upper triangle matrix spanned by the row and column indices which are passed to Procedure Triangle. Let candidate be the output from Procedure Square. Then by the above remark and Step 3 of Procedure Square, this position is the only possible position in the square matrix defined by the indices passed to this procedure by the central program. Thus all that is left for the Program Saddlepoint to do is to check if this position is indeed a strict saddlepoint; which it does in Step 2.

THEOREM. *Program Strict Saddlepoint terminates in time* $O(n^{\lg 3})$.

Proof. Let $F(n)$ be the time required to run Procedure Square $(n, r(1), \ldots, r(n), c(1), \ldots, c(n)$; candidate) and let $G(n)$ be the analogous time for

procedure Triangle $(n, r(1), \ldots, r(n), c(1), \ldots, c(n)$; candidate). Hence, the work required to run Program Saddlepoint is $F(n) + O(\beta n)$; (due to the last check required to see if candidate is really a strict SP in step 2 of Program Saddlepoint). Notice that $G(n) = F(n) - c_1 \beta n \log(\beta n)$ for some constant, c_1, because the only difference in the two algorithms is that the sorting step of Procedure Square does not occur in Procedure Triangle. By analyzing the steps of Program Saddlepoint,

$$F(n) = c_1 \beta n \lg(\beta n) + F\left(\frac{n}{2}\right) + 2G\left(\frac{n}{2}\right) + c_2\beta + c_3\beta n,$$

where the $c_3\beta n$ term is the cost of passing the row and column pointers and $c_2\beta$ is the cost of making the final decision between candidates; that is, step 4 of the triangle procedure (using observation 1).

So,

$$\begin{aligned} F(n) &= c_1\beta n \lg(\beta n) + 3F\left(\frac{n}{2}\right) + c_2\beta + c_3\beta n - 2c_1\beta\left(\frac{n}{2}\right)\lg\left(\frac{\beta n}{2}\right) \\ &= c_1 n \lg n + 3F\left(\frac{n}{2}\right) + c_2\beta + c_3(\beta n) - c_1(\beta n)\lg(\beta n) + c_1(\beta n)\lg\left(\frac{1}{2}\right) \\ &= 3F\left(\frac{n}{2}\right) + c_2\beta + (c_3 - c_1)(\beta n). \end{aligned}$$

Iterating this expression yields

$$F(n) = 3^{\lg n}F(1) + \sum_{i=0}^{\lg n - 1}\left(3^i c_2\beta + \left(\frac{3}{2}\right)^i (c_3 - c_1)(\beta n)\right).$$

But, $F(1) = \alpha$, so after simplifying this geometric series we get

$$\begin{aligned} F(n) &= 3^{\lg n}\left(\alpha + \frac{1}{6}c_2\beta\right) + 2\left(\frac{3}{2}\right)^{\lg n - 1}(c_3 - c_1)(\beta n) - \left(\frac{1}{2}c_2\beta + 2(c_3 - c_1)\beta n\right) \\ &= 3^{\lg n}\left(\alpha + \frac{1}{6}c_2\beta + \frac{4}{3}(c_3 - c_1)\beta\right) - \left(\frac{1}{2}c_2\beta + 2(c_3 - c_1)\beta n\right). \end{aligned}$$

So,

$$F(n) = O(3^{\lg n}).$$

But, $3^{\lg n} = n^{\lg 3}$, and hence

$$F(n) = O(n^{\lg 3}).$$

Hence, by the remark at the beginning of the proof, the Program Strict Saddlepoint runs in time $O(n^{\lg 3})$ as claimed.

Note that this analysis shows that the program saddlepoint requires both order $n^{1.58}$ matrix look-ups and order $n^{1.58}$ total running time. That is, even if a matrix look-up takes significantly more time than an arithmetic operation, that is, $\alpha \gg \beta$, the algorithm will remain $O(n^{1.58})$.

The reader familiar with fractals will note that the unshaded region in FIGURE 2, corresponding to the portion of the matrix that must be searched, is an illustration

of a fractal of dimension lg 3; and hence has area $n^{\lg 3}$. This provides a geometrically intuitive argument for the running time of Program Strict Saddlepoint [**11**, pp. 141–142].

COROLLARY. *The strict SP problem on an $m \times n$ matrix, with $m \geqslant n$, can be solved in $O((\frac{m}{n})n^{\lg 3})$ time.*

Proof. Divide the matrix into $\lceil \frac{m}{n} \rceil$ possibly overlapping $n \times n$ square matrices, $A_1, \ldots, A_{\lceil m/n \rceil}$. Apply the theorem to each A_i to get at most $\lceil \frac{m}{n} \rceil$ candidate strict SP's, from which the real strict SP (if it exists) can be found in work $O(\frac{m}{n} + m + n)$. The total work is thus bounded by

$$\left\lceil \frac{m}{n} \right\rceil O(n^{\lg 3}) + O(m) = O\left(\left(\frac{m}{n}\right)n^{\lg 3}\right)$$

as claimed.

III. Saddlepoints. Note that if we are seeking a SP then observation 1 holds only if $a_{11} > a_{22} > \cdots > a_{nn}$ and similarly observation 2 will fail unless the inequality is strict. Thus it is not clear whether the search for a SP can be sped up in a similar way. In fact we show next that the SP problem cannot be solved in time better than cn^2.

Let C be the class of all algorithms that are made up entirely of table look-ups and arithmetic operations.

THEOREM. *There does not exist an algorithm for the SP problem in the class C that can run faster than cn^2. Further, there exists an algorithm in C that does run that fast.*

Proof. We can put a check mark in each matrix cell which is the maximum in its row in time cn^2, and then do the same for the cells which are the minimum in their columns in cn^2 time while checking for a cell with two check marks which would clearly be an SP. Hence, there surely is a quadratic time algorithm in C for this problem.

We now show that no faster algorithm is possible. We do this by playing the adversary against an arbitrary algorithm in C. We argue that the algorithm must look up every entry in A.

Our strategy: For the first $n^2 - 1$ queries of the values of a_{ij} we answer 0, unless all the other entries in row i have been queried, in which case we answer 1. If we answer .5 to the last (n^2) query, then either that cell is the unique SP or there does not exist an SP. This follows since every 0 entry has either a 1 or a .5 in its row and every 1 has either a 0 or a .5 in its column. If instead, we answer -1, then the cell is not a SP, and in fact any zero entry in that last row will be a SP. Hence no algorithm in C can know the answer to the SP problem without n^2 queries.

In conclusion, we have shown that the strict SP problem can be solved in $O(n^{1.58})$ time, but that a similar improvement from $O(n^2)$ for the SP problem cannot be found. Note that our algorithm for the strict SP problem uses $O(n^{1.58})$ matrix queries and has an $O(n^{1.58})$ total running time. We conjecture that it may be possible to solve this problem with less than $O(n^{1.58})$ matrix queries but at the cost of a longer running time.

REFERENCES

1. Jean Pierre Aubin, Mathematical Methods of Game and Economic Theory, North-Holland Publishing Company, Amsterdam, 1979.
2. David Harold Blackwell and Meyer A. Girshick, Theory of Games and Statistical Decisions, Wiley, New York, 1954.
3. George B. Dantzig, Linear Programming and Extensions, Princeton University Press, Princeton, NJ, 1963.
4. Melvin Dresher, L. S. Shapley, and A. W. Tucker, editors, Advances in Game Theory, Princeton University Press, Princeton, NJ, 1964.
5. David Gale, The Theory of Linear Economic Models, McGraw-Hill, New York, 1960.
6. David Gale, The Theory of Matrix Games and Linear Economic Models, Department of Mathematics, Brown University, Providence, RI, 1957.
7. Samuel Karlin, Mathematical Methods and Theory in Games, Programming, and Economics, Addison-Wesley Publishing Company, Reading, MA, 1959.
8. H. W. Kuhn and A. W. Tucker, editors, Contributions to the Theory of Games, Volume I. Princeton University Press, Princeton, NJ, 1950.
9. Richard I. Levin and Robert B. DesJardins, Theory of Games and Strategies, International Textbook Company, Scranton, PA, 1970.
10. Duncan R. Luce and Howard Ráiffa, Games and Decisions: Introduction and Critical Survey, John Wiley and Sons, Inc., New York, 1957.
11. Benoit B. Mandelbrot, The Fractal Geometry of Nature, W. H. Freeman and Company, New York, 1977.
12. Francis B. May, Introduction to Games of Strategy. Allyn and Bacon, Boston, MA, 1970.
13. John von Neumann and Oskar Morgenstern, Theory of Games and Economic Behavior, Princeton University Press, Princeton, NJ, 1944.
14. Guillermo Owen, Game Theory, second edition, Academic Press, New York, 1982.
15. T. Parthasarathy and T. E. S. Raghavan, Some Topics in Two-Person Games, American Elsevier Publishing Company, New York, 1971.
16. Anatol Rapoport, Two Person Game Theory; the Essential Ideas, University of Michigan Press, Ann Arbor, 1966.
17. S. Vajda, The Theory of Games and Linear Programming, Methuen, London, 1967.