Discrete Applied Mathematics 43 (1993) 131-153
North-Holland

Dividing and conquering the square

Donna C. Llewellyn and Craig A. Tovey

School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA 30332-0205, USA

Received 19 March 1990
Revised 7 May 1991

Abstract

Llewellyn, D.C. and C.A. Tovey, Dividing and conquering the square, Discrete Applied Mathematics 43 (1993) 131-153.

A local minimum of a matrix is a cell whose value is smaller than those of its four adjacent cells. For an $n \times n$ square matrix, we find a local minimum with at most $2.554n$ queries, and prove a lower bound of $\sqrt{2}n$ queries required by any method. For a different neighborhood corresponding to the eight possible moves of a chess king, we prove upper and lower bounds of $3n + O(\log n)$ and $2n$, respectively.

Keywords. Local optimum, local search, matrix, grid, graph, saddle point.

1. Introduction

A local minimum of a matrix is a cell with value less than or equal to those of its neighboring cells. How hard is it to find a local minimum of an $n \times n$ matrix? It takes zero computation to determine that one exists, since any global minimum is surely one. And any local minimum, once found, can be verified in O(1) time. On the other hand, local improvement, the most natural search method, can require time $\Omega(n^2)$, the same order as enumeration. For example, suppose a local minimum is sought for a matrix with a descending spiral, illustrated for $n = 8$ in Fig. 1 (M is some large value such as n^2). Any local improvement algorithm must traverse a path along that spiral out to the unique local optimum in the corner. But the length of this path will be $\Omega(n^2)$ for most starting points in the square.

The large gap between the obvious $\Omega(1)$ lower bound and $O(n^2)$ upper bound is a characteristic of the local optimization problem. In this paper we narrow the gap

Correspondence to: Professor D.C. Llewellyn, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA.

1	2	3	4	5	6	7	8
M	M	M	M	M	M	M	9
27	28	29	30	31	32	M	10
26	M	M	M	M	33	M	11
25	M	39	M	M	34	M	12
24	M	38	37	36	35	M	13
23	M	M	M	M	M	M	14
22	21	20	19	18	17	16	15

Fig. 1. Local improvement can require $\sim n^2$ work.

between these bounds considerably. There are two natural neighborhoods to consider: (i) in the *king adjacency*, the eight neighbors of a particular cell are those a king could move to from that cell on a chessboard; (ii) in the *grid adjacency* the four neighbors of a cell are the two adjacent in the same row and the two adjacent in the same column. We will investigate both neighborhoods here. Our best strategies for the two adjacencies turn out rather differently.

First we summarize our principal results: let $\tau(n)$ equal the minimum number of matrix lookups required by any valid algorithm that finds a local optimum of a square $n \times n$ matrix. Then

$$2n \le \tau(n) \le 3n + O(\log n) \quad \text{(king adjacency)},$$

$$\sqrt{2}n \le \tau(n) \le 2.554n \quad \text{(grid adjacency)}.$$

Our best strategy for the grid adjacency, which yields the $2.554n$ upper bound, is fairly complicated. It is interesting that the matrix, perhaps the simplest natural discrete structure for local optimization, is not very straightforward to solve.

In the rest of this section we review necessary background on search procedures for local optima, and apply it to our specific case of a matrix. The next sections develop the results stated for the king and grid adjacencies, respectively. We conclude in Section 4 with some remarks and conjectures.

1.1. Divide-and-conquer

We seek a strict local optimum of an $n \times n$ matrix A of distinct numbers $A(i,j)$. (All results apply to the slightly more general problem of seeking a nonstrict local optimum when the $A(i,j)$ values are not necessarily distinct.) Equivalently, form a graph $G=(V,E)$ of the matrix as follows: take the cells of A as the nodes V of G, and take the edge set

$$E = [\{(i,j),(i',j')\}:\ \max(|i-i'|,|j-j'|)=1]$$

for the king adjacency; and

$$E = [\{(i,j),(i',j')\}:\ |i-i'|+|j-j'|=1]$$

for the grid adjacency. Then we seek a local optimum of the function A on the graph G. As in [2,4,5], our computational model employs an oracle to compute the values of A. A call to the oracle is a *query*; the total number of queries is taken as the computational effort.

We now summarize necessary background regarding local optima on graphs, from [4]. Results are in terms of finding a local minimum, without loss of generality.

The following *divide-and-conquer* method will find a local minimum of A:

(1) Query the vertices in a separating set S of G, finding a vertex $v \in S$ with minimum $A(v)$. (Where a separating set is a collection of vertices which disconnects the graph.)

(2) Query the vertices in $N(v)$, i.e., those adjacent to v. If v is a local minimum, stop. Otherwise proceed to (3).

(3) Select $x \in N(v) \cup \{w\}$ with $A(x) < A(v)$ and $A(x) \leq A(w)$; replace G by the connected component of $G \setminus S$ containing x; set $w := x$; return to (1).

Virtually all the work in the algorithm occurs in (1), where the vertices of the separating sets S are queried. The best separators are found by solving the following

Separation game.

Input: Graph $G = (V, E)$.

Two players: Minimizer I, Maximizer II.

Description: Player I removes vertices from G until it is disconnected. Player II selects one of the newly created components to call G, discards the other components, and passes the new G back to I. The game ends when $|V| \leq 1$.

Step 1. $i = 0$, $V^0 = V$; score$(G) = 0$.

Step 2. If $|V| \leq 1$ STOP.

Step 3. Player I chooses $S^i \subseteq V^i$ such that $G^i \setminus S^i$ is not connected or is the empty graph; score$(G) =$ score$(G) + |S^i|$.

Step 4. Player II selects G^{i+1}, a connected component of $G^i \setminus S^i$; $i := i + 1$; go to Step 2.

The *value* of the separation game on G, denoted $v(G)$, is score(G) when each player plays optimally, I to minimize and II to maximize. Let K denote the number of separating sets used by player I, and let $\lambda_{\max}(G)$ denote the maximum degree of any vertex in the graph G. The principal result we employ is

Theorem 1.1.1. *Any algorithm to find a local minimum of G requires at least $v(G)$ queries; the Divide-and-Conquer method requires at most $v(G) + K\lambda_{\max}(G)$ queries.*

1.2. Implications

The value of K in Theorem 1.1.1 is typically logarithmically small, and for our graph of the matrix the maximum degree $\lambda_{\max}(G) = 8$ or 4. Therefore, the implication of Theorem 1.1.1 is to *transform our problem into an analysis of the separation game on G.*

Solving the separation game is unfortunately NP-complete in general but we can employ the following partial characterization [4]:

Lemma 1.2.1. *In the separation game S^i can always be taken to be a minimal separating set of G.*

The minimum separating sets, in turn, are partially characterized in the following lemma by an interesting dual relationship between the two adjacencies:

Lemma 1.2.2. *A minimal separating set for the separation game under the king adjacency must be connected with respect to the grid adjacency; a minimal separating set under the grid adjacency must be connected with respect to the king adjacency.*

Proof. Let S be a minimal separating set of $G=(V,E)$. S separates some set $U\subset V$; $U\cap S=\emptyset$; $U\neq\emptyset$ from $V-S-U$ in the graph G. The set U may be taken to be connected for if not we can replace it with any connected component. Here "connected" means under the adjacency for which a local optimum is sought.

Now S must contain $B(U)$, the *boundary* of U, i.e., $S\supseteq B(U)\equiv\{v\in V\colon v\notin U, \exists u\in U, (v,u)\in E\}$ for otherwise there would be a path from U to some vertex in $V-S-U$ that did not pass through S. But also $B(U)$ is a separating set, thus $S=B(U)$ by the minimality of S.

We also may take U to be topologically simple. If U is not simple, there are two cases: (i) if G contains a vertex not encircled by U and not in $B(U)$, then let U' be U together with all vertices encircled by U (thus including some members of $B(U)$). In this case, $B(U')$ is strictly contained in $B(U)$ and therefore $S=B(U)$ was not minimal. Otherwise, (ii) there must exist a nonempty connected component U' of G, encircled by U. Then U' is simple by induction, therefore $B(U')\subseteq B(U)$, and so we can replace U by U'.

It remains to show that when U is connected under the king (respectively grid) adjacency, then $B(U)$ is connected with respect to the grid (respectively king) adjacency. The idea is demonstrated in Fig. 2.

The boundary of a cell under the king adjacency is connected with respect to the grid adjacency; the boundary of a cell under the grid adjacency is connected with respect to the king adjacency. For a formal proof, we employ this observation in an induction on $|U|$. Remove c, the rightmost of the uppermost cells of U. Referring to Fig. 2, cell $i\notin U$: $i=1,\dots,4$. By induction, the boundary of $U-c$ is connected as claimed. Again referring to Fig. 2 $(5,6,7,8)\cap U\neq\emptyset$ (king adjacency); $(6,8)\cap U\neq\emptyset$ (grid). When c is added to $U-c$, the boundary gains all neighbors of c not in U,

	X	
X	c	X
	X	

X	X	X
X	c	X
X	X	X

1	2	3
8	c	4
7	6	5

Fig. 2. Cell c and its king and grid boundaries.

and loses c since U is connected. Checking all the possible cases, it is generally easy to see that if $B(U-c)$ is connected as claimed, so is $B(U)$. The only nontrivial cases occur when the removal of c disconnects the boundary. For example (king adjacency), if $6 \in U$, $7 \notin U$, $8 \in U$, then $5 \in B(U)$, $7 \in B(U)$, and it is possible that 5 and 7 are only connected through c. But then U is not simple, and we have a contradiction. $\square$

Before proceeding with the analysis, it is interesting to see what upper and lower bounds can be derived directly from known results. In [4] it is shown (Corollary 4.11) that a local optimum for any planar graph may be found in $13.35\sqrt{n} + \lambda(\log n/(\log 3 - 1))$ queries. Since $\lambda = 8$ or 4 here, the logarithmic term is negligible, and we can take $13.35\sqrt{n}$ as an upper bound on the necessary number of queries, for both the king and grid adjacencies. For a lower bound, we appeal to the following results ([4, Corollary 4.5] and [3, Theorem 11], respectively):

Theorem 1.2.3. *For any graph and integer t,*

$$\upsilon(G) \geq \min\left\{t, \max_k \min\{|B(S)|:\ k-t \leq |S| \leq k\}\right\}.$$

Theorem 1.2.4. *Let $G=(V,E)$ be an $n \times n$ grid graph, and let $A \subset V$ satisfy $|V|/3 \leq |A| \leq 2|V|/3$. Then $|B(A)| \geq n/3$.*

Theorem 1.2.4 applies to the king adjacency as well, because all edges in the grid graph are edges in the king graph. Letting $t = n/3$ and considering $k = |V|/2 = n^2/2$, we find that $n/3$ is a lower bound on the number of queries needed to find either a king or grid local optimum. Thus we have

Theorem 1.2.5. *Let $\tau(n)$ denote the least number of queries required by a valid algorithm to find a local optimum in a matrix (king or grid adjacency). Then*

$$n/3 \leq \tau(n) \leq 13.35n.$$

We sharpen these bounds considerably in the following.

2. The king adjacency

We consider the problem of finding a local optimum of a matrix where the neighborhood structure is defined by the king adjacency. By Lemma 1.2.2, a minimal separating set here must be connected with respect to the grid adjacency. We call any such set a *region* of the matrix. Whenever we consider a region of a matrix, we will think of it as being embedded within a sufficiently large square matrix. For any element in this embedding structure that is not in the original region, define its distance to the region as the length of the shortest path (using grid adjacency) to the region. Then give each of these embedding entries value equal to its distance $+n^2$. The easiest way to think of this is to think of dropping the region into the embedding structure and hence the values of the surrounding region will be strictly larger than the values within the region and will gradually climb as one moves further away from the region. This will prove useful later when we approximate the indices of a matrix to be queried and may by chance query an entry of a region which does not exist.

2.1. Upper bounds

Our divide-and-conquer algorithms will have two major types of steps: query and check. For ease of presentation we first define these steps and give the parameters for each. Then we present each procedure, first in words, and then using these generic steps. We also give a pictorial view of each procedure.

A *query* step takes as input a description of a region of A and gives as output the minimum entry in that region. This step requires a number of queries equal to the size of the input set. This input will be given in one of two ways:

- Column set (called a *Column Query*): a pair made up of a column index, j, and a pair of row indices, (i_0, i_1) with $i_0 < i_1$. Here the query step should be performed over rows $i_0, i_0+1, \ldots, i_1$ in column j. (We will use the notation Column Query($j, (i_0, i_1)$: a) where the output of the query is a.)
- Row set (called a *Row Query*): a pair made up of a row index, i, and a pair of column indices, (j_0, j_1) with $j_0 < j_1$. Here the query step should be performed over columns $j_0, j_0+1, \ldots, j_1$ in row i.

A *check* step takes as input an entry in the matrix A and gives as output the smallest element among the input and its neighbors.

Our procedure will take as input matrix A and a range of rows and a range of columns, and give as output a local minimum of A within the given ranges.

Procedure Row-Column(Rows(1, n), Columns(1, n): a^*) (see Fig. 3). This algorithm is the divide-and-conquer algorithm defined in Section 1 with a specific, natural choice of separators. The first separator is the central column of the matrix. If the minimum of this column is not a local optimum then it is assumed without loss of generality that the left neighbor is smaller, and the next separator is the central row of the left submatrix of A. If more separators are needed, the procedure is repeated on the remaining square submatrix (taken without loss of generality to be the upper left submatrix of A).

Step 1. Column Query($\lceil n/2 \rceil, (1, n)$: a^1).

Step 2. Check(a^1: $\bar{a}^1$).

Step 3. If $a^1 = \bar{a}^1$ then STOP: $a^* = a^1$. Otherwise, without loss of generality $a^1 = a_{i,j}$ and $\bar{a}^1 = a_{i,j-1}$.

Step 4. Row Query($\lceil n/2 \rceil, (1, \lceil n/2 \rceil - 1)$: a^2).

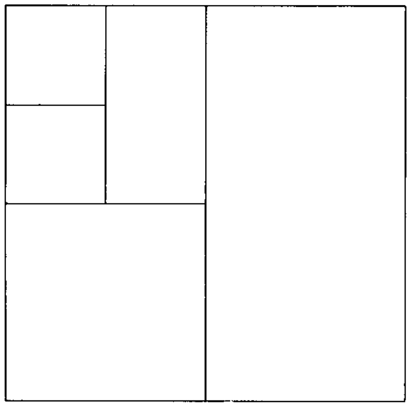

Fig. 3. Illustration of Procedure Row-Column.

Step 5. Check($a^2 : \bar{a}^2$).

Step 6. If $a^2 = \bar{a}^2$ then STOP: $a^* = a^2$. Otherwise, without loss of generality $a^2 = a_{i,j}$ and $\bar{a}^2 = a_{i-1,j}$.

Step 7. Procedure Row-Column(Rows($1, \lceil n/2 \rceil - 1$), Columns($1, \lceil n/2 \rceil - 1$): a^*).

Theorem 2.1.1. *Procedure Row-Column finds a local minimum of an $n \times n$ matrix in less than $3n + O(\log n)$ queries.*

Proof. Let $f(n)$ be the number of queries that this procedure requires for an $n \times n$ matrix. Then clearly, $f(n) = n + \lceil n/2 \rceil + 12 + f(\lceil n/2 \rceil - 1)$. This leads to the solution $f(n) = n + 2(n/2 + n/4 + \cdots) + O(\log n)$ which converges to $3n + O(\log n)$. □

Corollary 2.1.2. *A local optimum of an $m \times n$ matrix, with $m < n$, can be found in less than $m(2 + \alpha) + n/(2^{\alpha}) + O(\log n) \le 2m + n + O(\log n)$ queries, where $\alpha \equiv \lfloor \log_2 n/m \rfloor$.*

Proof. Slightly altering Procedure Row-Column to always bisect the longer direction (and hence use the lesser number of queries) in place of alternating between column and row queries gives this result immediately. □

2.2. *Lower bounds*

We now find lower bounds for the number of queries needed to find a local minimum of an $n \times n$ matrix. The main result of this section is a lower bound of $2n$ queries.

In the discussions that follow it will be helpful to have the notion of the top, bottom, left and right of a region of a matrix. Suppose that R is a region within $n \times n$ matrix A.

Define

$$a=\min\{i\colon (i,j)\in R \text{ for any } j,\ 1\le j\le n\},$$

$$b=\min\{j\colon (a,j)\in R\},$$

$$c=\max\{j\colon (a,j)\in R\},$$

$$d=\max\{i\colon (i,j)\in R \text{ for any } j,\ 1\le j\le n\},$$

$$e=\min\{j\colon (d,e)\in R\},$$

$$f=\max\{j\colon (d,f)\in R\}.$$

Then define the following "corners" of region R:

UL("upper left") $\equiv (a,b)$,

UR("upper right") $\equiv (a,c)$,

LL("lower left") $\equiv (d,e)$,

LR("lower right") $\equiv (d,f)$.

Now, in order to define the sides, consider the region R and define an entry of R to be *interior* if it has four grid neighbors within R and *frontier* otherwise. We will think of traveling along the frontier entries from one corner to another in the clockwise direction (using the grid adjacency to define this path). The collection of frontier entries that one encounters while traveling from UL to UR, inclusive, is called the *top*, those met while traveling from UR to LR, inclusive, are called the *right*, those hit while in transit from LR to LL, inclusive, form the *bottom*, and finally the others, that set lying on the path from LL to UL, inclusive, is the *left*. It should be clear that if R is the whole matrix A then the top is row 1, the right is column n, the bottom is row n, and the left is column 1. It will not hurt our arguments to have an entry in more than one side.

Now, let the minimum diameter of R, MinD(R), be the length of a simple grid-connected left-right path (i.e., a path of matrix entries from any element of the left to any element of the right) of minimum length in R, where the length of a path is measured by the number of entries in the path. Analogously define the maximum diameter, MaxD(R). Then, let the minimum height, MinH(R) be the length of a simple grid-connected top-bottom path of minimum length in R; and analogously define the maximum height, MaxH(R).

We will define a strategy for player II, the maximizer in the Separation game, to get a lower bound on $v(G)$.

Player II strategy: Before player I plays, we define R to be the subset of matrix A consisting of all unqueried entries. By definition of separation here, this forms a region. Therefore, after player I's turn, the unqueried entries form two (disconnected) regions, say R_1 and R_2. If one of these R_i, $i=1$ or 2, touches all four sides (top, bottom, left and right) of R, then choose that component, R_i.

Otherwise, choose that component among R_1 and R_2 which has the largest maximum of its maximum diameter and maximum height. That is, let $d_i = \max\{\mathrm{MaxD}(R_i), \mathrm{MaxH}(R_i)\}$, and let $i^* = \operatorname{argmax}\{d_i\}$. Then choose component R_{i^*}.

Lemma 2.2.1. *It requires at least n queries to find a local minimum of an $n \times n$ matrix.*

Proof. Let the sequence of regions chosen by player II be given by $A = R^0, R^1, \ldots, R^k$. Then, using the strategy above, for some j, $1 \leq j \leq k$, region R^j will not touch all four sides of region R^{j-1} (since at the end this is true). Hence, either the top and bottom of R^{j-1} are disconnected or the left and right of R^{j-1} are disconnected by queried elements (or both). Without loss of generality, assume that the left and right are disconnected. Then, by piecing together the earlier queries, the left and right of the original matrix, A, are also disconnected. Hence, there exists a path from the top to the bottom of A made up of queried entries. Clearly, these entries alone have used n queries. □

Theorem 2.2.2. *It requires at least $2n - 1$ queries to find a local minimum of an $n \times n$ matrix.*

Proof. First note that the theorem is true in the case of $n = 1$. Consider the above strategy for player II. Suppose, without loss of generality (as guaranteed by Lemma 2.2.1), that eventually player I queries a top–bottom path. Consider the first time such a path has been queried (i.e., this is the first time player II must choose a component that does not touch all four sides). Suppose that up to this time $n + H$ queries have been made. If $H > n$, then clearly the theorem is proved. So suppose that $H < n$. Now, in the chosen component, there exists a square of side length equal to the minimum of the minimum diameter and minimum height of the component. How can this value be made as small as possible? By using all of the extra H queries to shorten one of them, say the minimum diameter. This is done by using all of these queries in "horizontal" queries, and centering them so that the minimum diameter is exactly $(n - H)/2$. Hence, in the chosen component, there exists a square matrix of size at least $(n - H)/2 \times (n - H)/2$. By Lemma 2.2.1 above, this requires at least $(n - H)/2$ queries. Thus, the total number of queries so far is at least $n + H + (n - H)/2 = 1.5n + 0.5H$. Now, "bootstrapping" with this result in place of the bound given in the lemma gives that the enclosed square requires at least $1.5((n - H)/2)$ and hence the total lower bound is $1.75n + 0.25H$. Iterating this procedure gives a lower bound of $2n$ queries. □

Now we give an alternative way to arrive at the same lower bound of $2n$ queries. We include this because the method is quite different and we believe it provides some additional insight into the geometry of the problem.

First we need the following lemma. In this method, it is best to think of the matrix, A, as being placed in the $\mathbb{R}^2$ plane in the following way. Each entry takes up a unit square, so that the outer edge of the left of A is the y-axis, the outer edge of the bottom of A is the x-axis, the outer edge of the right of A is the line $x=n$ and the outer edge of the top of A is the line $y=n$. Then, given any region R, its area Area(R) is the actual (continuous) area of the enclosed region; its perimeter Per(R) is the sum of the Euclidean lengths of all the straight lines that make up the outer edges of the frontier of the region.

Lemma 2.2.3. *For any region*, R,

$$\frac{\sqrt{\mathrm{Area}(R)}}{\mathrm{Per}(R)} \le \frac{1}{4}.$$

Proof. First consider a rectangle with width w and length $w+\delta$, where $\delta \ge 0$. Then $\mathrm{Per}(R)=2w+2(w+\delta)$ and hence $(\mathrm{Per}(R))^2=16w^2+4\delta^2+16w\delta$. Further, $\mathrm{Area}(R)=w(w+\delta)$, so $16(\mathrm{Area}(R))=16w^2+16w\delta$. Since $\delta\ge 0$, it is clear then that $(\mathrm{Per}(R))^2 \ge 16\,\mathrm{Area}(R)$ so the lemma follows for rectangles.

Now consider any region R. Let the tightest circumscribing rectangle be T. It is clear that $\mathrm{Area}(T)\ge\mathrm{Area}(R)$. Hence it is sufficient to prove that $\mathrm{Per}(T)\le\mathrm{Per}(R)$ since then we will have $(\mathrm{Per}(R))^2\ge(\mathrm{Per}(T))^2\ge\mathrm{Area}(T)\ge 16\,\mathrm{Area}(R)$. Think of traveling around the boundary of R and notice that this boundary will agree with the boundary of T at least for some stretch on each side of T (top, bottom, left and right). Where the boundary of R differs from the boundary of T, call it a *journey*. Any journey either originates and ends on the same side or else it originates on one side and ends on an adjacent side of T. We will consider each of these cases separately.

(1) Suppose the journey originates and terminates on the same side. Without loss of generality, suppose it is the top or bottom. Consider the origin as a point in the $\mathbb{R}^2$ plane, (a,b). Then the terminus is another point (c,b). Without loss of generality assume that $c>a$. Then the amount of the perimeter of T between these two points is exactly $c-a$. The amount of the perimeter of R that lies between these points is at least $c-a$ since the boundary follows the grid adjacency (it might also have a vertical component and hence could be greater).

(2) Now without loss of generality, suppose that the journey originates on the left and ends on the top. Then the origin is some point (a,b) and the terminus is another point (c,d). We know that $d>b$. Then the perimeter of T between these points is $(d-b)+|(c-a)|$ and the perimeter of R must be at least this by the same reasoning as above (it must travel at least a $d-b$ vertical distance and at least a $|c-a|$ horizontal distance).

Since the two regions clearly have the same perimeter between journeys, this completes the proof. $\square$

Now consider the region R before player I queries during some iteration of the separation game. The queries that follow before another turn of player II can be combined into some separating set, C. This causes R to be divided up into two regions, R_1 and R_2. For ease of presentation, call Area$(R) \equiv A$, Per$(R) \equiv P$ and similarly, Area$(R_i) \equiv A_i$ and Per$(R_i) \equiv P_i$ for $i = 1, 2$. We will abuse notation slightly and use C also to represent the number of entries in the set C. We now need the following result.

Lemma 2.2.4. *Let*

$$\frac{A_1}{P_1} = \max_{i=1,2} \left\{ \frac{A_i}{P_i} \right\},$$

and suppose that

$$A_1 = \lambda A \quad \textit{for some } 0 < \lambda < 1.$$

Then,

$$P_1 - \lambda P \leq 2\lambda C.$$

Proof. First we will need the following inequality: $P_1 + P_2 \leq 2C + P$. To see this rigorously, we need the following notation. Let the frontier of region R_i for $i = 1, 2$ be denoted F_i. Let F_i be the (not necessarily disjoint) union of F_{i1} and F_{i2}, where F_{i1} is the part of F_i which is adjacent to the frontier of C, and F_{i2} is the part of F_i which is a subset of the frontier of R. We can break up $P_1 + P_2$ into the part which arises from tracing along $F_{12} \cup F_{22}$ and the part which comes from tracing along $F_{11} \cup F_{21}$. The first part is clearly less than or equal to P. Our goal is therefore to show that the second part is at most $2C$. To this end, C can be assumed to be a minimal separating set by Lemma 1.2.1, whence simple case analysis verifies that no cell of C has more than two sides adjacent to $F_{11} \cup F_{21}$. Lemma 1.2.1 also ensures that C has no interior. Therefore, the second part is at most $2C$ and the inequality holds.

Thus,

$$2C + P \geq P_1 + P_2$$

and also

$$P_1 + P_2 - 2C \leq P.$$

This implies that

$$\begin{aligned} P_1 - \lambda P &\leq P_1 - \lambda(P_1 + P_2 - 2C) \\ &\leq (1-\lambda)P_1 - \lambda P_2 + 2\lambda C. \end{aligned}$$

So, if we can show that

$$\lambda P_2 \geq (1-\lambda)P_1$$

then the lemma is proved.

By assumption,

$$\frac{A_1}{P_1} \geq \frac{A_2}{P_2} \quad \Rightarrow \quad \frac{\lambda A_1}{P_1} \geq \frac{(1-\lambda)A}{P_2}$$

$$\Rightarrow \quad \lambda P_2 A \geq (1-\lambda) A P_1$$

$$\Rightarrow \quad \lambda P_2 \geq (1-\lambda) P_1. \qquad \square$$

Now we are ready for our main result.

Theorem 2.2.5. *The number of queries needed to find a local optimum in a region of area A and perimeter P is at least*

$$\frac{8A}{P} - 1.$$

Proof. The proof is inductive on A. First note that if $A=1$ then it must be that $P=4$, and clearly it requires exactly one query to solve the problem, so the result holds.

In general, it is sufficient to show that

$$\frac{8A}{P} \leq C + \frac{8A_1}{P_1}$$

where as in Lemma 2.2.4, $\max_{i=1,2}\{A_i/P_i\} = A_1/P_1$. By Lemma 2.2.3 we know that

$$P \geq 4\sqrt{A}$$

and that

$$P_1 \geq 4\sqrt{A_1} = 4\sqrt{A\lambda}.$$

So, $P \times P_1 \geq 16A\sqrt{\lambda}$. This implies that $C\sqrt{\lambda}(P \times P_1) \geq 16AC\lambda$. Rearranging this gives

$$\frac{C\sqrt{\lambda}}{8A} \geq \frac{2C\lambda}{P \times P_1}.$$

Using Lemma 2.2.4 gives the righthand side as

$$\geq \frac{P_1 - \lambda P}{P \times P_1} = \frac{1}{P} - \frac{\lambda}{P_1}.$$

So,

$$\frac{C\sqrt{\lambda}}{8} \geq \frac{A}{P} - \frac{A\lambda}{P_1} = \frac{A}{P} - \frac{A_1}{P_1}.$$

Hence

$$\frac{8A}{P} \leq C\sqrt{\lambda} + \frac{8A_1}{P_1}.$$

We know that $\lambda < 1$ so the result follows. $\square$

Corollary 2.2.6. *The number of queries needed to find a local minimum of an $n \times n$ matrix is at least $2n$.*

Proof. For an $n \times n$ square $A = n^2$ and $P = 4n$. Using this in Theorem 2.2.5 above gives the result immediately. $\square$

3. The grid adjacency

In this section we consider the problem of finding a local optimum of a matrix where the neighborhood structure is defined by the usual adjacency in grids. By Lemma 1.2.2, a minimal separating set need only be connected with respect to the king adjacency. Thus in this section a *region* of a matrix will be a subset of entries so connected. As in Section 2 we will continue to think of our matrix as being embedded within a larger square matrix. Of course, we must update our definition of distance to be consistent with king adjacency connected paths here.

3.1. Upper bounds

Of course, Procedure Row-Column can still be used here since any set of entries that is connected with respect to grid adjacency will also be connected with respect to king adjacency. Hence we immediately get an upper bound of $3n + O(\log n)$ on the number of queries needed. Here, though, we can show that it is not optimal. To improve on it, we employ diagonal queries. The intuition here is that while the Euclidean length of a diagonal of an $n \times n$ matrix is $n\sqrt{2}$, there are only n entries in the matrix on the diagonal, so diagonal queries are more efficient by a factor of $\sqrt{2}$.

In this section we will need one further way to call a query step:

• Line segment (called a *Line Query*): a pair made up a line definition, $bx + cy = d$, and a range on x, $x_0 \le x \le x_1$. Here it should be interpreted that the matrix, A, is placed in the $\mathbb{R}^2$ plane with the $(n, 1)$ entry at the origin and the $(1, n)$ entry at the $(1, 1)$ position in the plane. The query should be performed at the intersections of the matrix and the line segment.

Notice that the Row and Column Queries can be described as special cases of the Line Query. However, for technical reasons we leave them with their own descriptions.

To make the following procedures easier to understand, we must first take care of rotated matrices. We will call a square matrix that has been rotated 45 degrees a *diamond matrix*. Let $A = [a_{ij}]$ be our $n \times n$ matrix and let $B = [b_{ij}]$ be the inscribed diamond matrix. Note that B has Euclidean side lengths equal to $n/\sqrt{2}$, but when counting queries the side length is effectively only $n/2$. For this reason we will refer to B as an $n/2 \times n/2$ diamond matrix inscribed in the $n \times n$ matrix A. For ease, let n be even. Take $b_{11} = a_{n/2,1}$, $b_{1,n/2} = a_{1,n/2}$, $b_{n/2,1} = a_{n,n/2}$, and $b_{n/2,n/2} = a_{n/2,n}$. Then to find a local minimum of B, we will perform Procedure Row-Column on it, but querying the appropriate diagonal segments of A in place of rows and columns. The input will be the matrix A, but it is understood that only the elements of B must be known, and the check queries should only be performed over elements of B.

Procedure Diamond(Rows(1, n**), Columns(1,** n**):** a^***)** (see Fig. 4).

Step 1. Line Query($x + y = 1$, $\frac{1}{4} \le x \le \frac{3}{4}$: a^1).

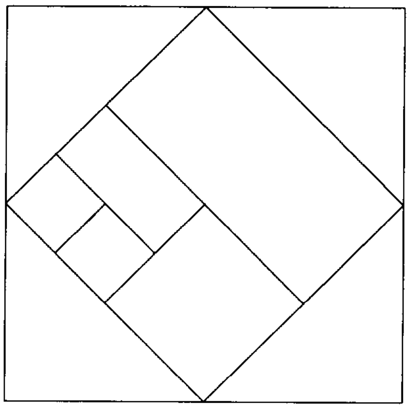

Fig. 4. Procedure Diamond.

Step 2. Check($a^1 : \bar{a}^1$).

Step 3. If $a^1 = \bar{a}^1$ then STOP: $a^* = a^1$. Otherwise, without loss of generality $a^1 = a_{i,j}$ and $\bar{a}^1 = a_{i,j-1}$ or $a_{i+1,j}$.

Step 4. Line Query($x = y$, $\frac{1}{2} \leq x \leq \frac{1}{2}$: a^2).

Step 5. Check($a^2 : \bar{a}^2$).

Step 6. If $a^2 = \bar{a}^2$ then STOP: $a^* = a^2$. Otherwise, without loss of generality $a^2 = a_{i,j}$ and $\bar{a}^2 = a_{i+1,j}$ or $a_{i,j+1}$.

Step 7. Procedure Diamond(Rows($\lceil n/2 \rceil + 1, n$), Columns($\lceil n/4 \rceil, \lceil 3n/4 \rceil$): a^*).

Corollary 3.1.1. *Procedure Diamond finds a local minimum of an $n/2 \times n/2$ diamond matrix in less than $1.5n + O(\log n)$ queries.*

Proof. This follows immediately from Theorem 2.1.1 and the discussion above on diamond matrices. □

Corollary 3.1.2. *A local minimum of an $m \times n$ diamond matrix, with $m \leq n$, can be found in less than $m(2 + \alpha + n/(2^\alpha m)) + O(\log n) \leq 2m + n + O(\log n)$ queries, where $\alpha \equiv \lfloor \log_2 n/m \rfloor$.*

To make the presentation easier, when we call Procedure Diamond we will refer to this more general form of Corollary 3.1.2 which can take as input an oblong diamond matrix. We will call the procedure by giving as input the four corners of the matrix rather than the rows and columns of the embedding square (or rectangular) matrix.

One last result we need before we can use this as a subroutine is the principle of containment: If one region, B, is a subset of another region, A, then it can require no more queries to find a local minimum of B than to find one of A. This is clear, since to find a local optimum of B we could just consider B to be embedded with

A as discussed above (of course A in turn is embedded within another large matrix), and then find a local optimum of A. We will sometimes call a procedure on a row and column set that imply that the whole diamond matrix does not exist (it would require rows or columns with negative indices or indices greater than n). When we do this we are actually relying on this containment principle, and are considering the existing region to be embedded within the called diamond matrix.

Procedure Diagonal(Rows(1, n), Columns(1, n): a^*) (see Fig. 5). This algorithm first queries and checks along the NE-SW diagonal of the square matrix. Failing to find a local optimum here, it queries and checks along half of the NW-SE diagonal. Then if it still hasn't found a local optimum it queries a diagonal paralled to the NE-SW line halfway down the triangle. After this it is either left with another triangle which it treats with Procedure Diamond, or it forms a diamond and a triangle out of the resulting shape. Each of these can be taken care of with Procedure Diamond.

Step 1. Line Query($x=y$, $0\le x\le 1$: a^1).

Step 2. Check($a^1:\bar{a}^1$).

Step 3. If $a^1=\bar{a}^1$ then STOP: $a^*=a^1$. Otherwise, without loss of generality $a^1=a_{i,j}$ and $\bar{a}^1=a_{i-1,j}$ or $a_{i,j-1}$.

Step 4. Line Query($x+y=1$, $0\le x\le\frac{1}{2}$: a^2).

Step 5. Check($a^2:\bar{a}^2$).

Step 6. If $a^2=\bar{a}^2$ then STOP: $a^*=a^2$. Otherwise, without loss of generality $a^2=a_{i,j}$ and $\bar{a}^2=a_{i,j-1}$ or $a_{i,j+1}$.

Step 7. Line Query($y=x+\frac{1}{2}$, $0\le x\le\frac{1}{4}$: a^3).

Step 8. Check($a^3:\bar{a}^3$).

Step 9. If $a^3=\bar{a}^3$ then STOP: $a^*=a^3$. Otherwise,

Case 1: $a^3=a_{i,j}$ and $\bar{a}^3=a_{i,j-1}$ or $a_{i-1,j}$, then Procedure Diamond((0, 1), (0, $\frac{1}{2}$), ($\frac{1}{4}$, $\frac{3}{4}$): a^*) or

Case 2: $a^3=a_{i,j}$ and $\bar{a}^3=a_{i,j+1}$ or $a_{i+1,j}$, then go to Step 10.

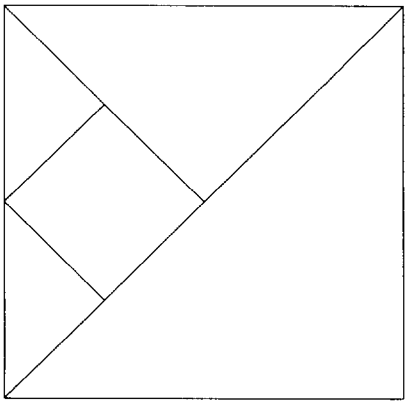
Fig. 5. Procedure Diagonal.

Step 10. Line Query($y=\frac{1}{2}-x$, $0\leq x\leq\frac{1}{4}$: a^4).

Step 11. Check($a^4:\bar{a}^4$).

Step 12. If $a^4=\bar{a}^4$ then STOP: $a^*=a^4$. Otherwise,

Case 1: $a^4=a_{i,j}$ and $\bar{a}^4=a_{i,j-1}$ or $a_{i+1,j}$, then Procedure Diamond($(0,\frac{1}{2}),(\frac{1}{4},\frac{1}{4}),(0,0)$: a^*) or

Case 2: $a^4=a_{i,j}$ and $\bar{a}^4=a_{i,j+1}$ or $a_{i-1,j}$, then Procedure Diamond($(0,\frac{1}{2}),(\frac{1}{4},\frac{3}{4}),(\frac{1}{4},\frac{1}{4}),(\frac{1}{2},\frac{1}{2})$: a^*).

Theorem 3.1.3. *Procedure Diagonal finds a local optimum of an $n\times n$ matrix in less than $2.75n+\mathrm{O}(\log n)$ queries.*

Proof. This procedure terminates in a Procedure Diamond iteration in either Case 1 of Step 9 or Case 1 or 2 of Step 12. We will consider each of these in turn. The Procedure Diamond iteration of each of these steps has as input a region within an $n/4\times n/4$ diamond matrix (within an $n/2\times n/2$ square matrix) and hence by Corollary 3.1.1 requires no more than $3n/4+\mathrm{O}(\log n)$ queries. The number of queries up to Step 9 is $n+n/2+n/4+12$. Hence if the procedure terminates here then the total number of queries is less than $2.5n+\mathrm{O}(\log n)$. If, however, the procedure reaches Step 12 then it already has performed $n+n/2+n/4+n/4+16$ queries and hence the total number is less than $2.75n+\mathrm{O}(\log n)$ if the termination occurs in either Case 1 or 2 of Step 12. □

The problem with this algorithm is that it is not "balanced". That is, one sees that if termination occurs in Step 9 then the total number of queries is significantly less than if termination occurs in Step 12. Intuitively, we should balance the regions to be explored, so that the number of queries is approximately the same regardless of where the algorithm is led. The place where we have the leeway to do this in Procedure Diagonal is when we choose the third query (Step 7). It was rather arbitrary that we decided to query the halfway diagonal. With this in mind we now introduce a "generic" or parametrized algorithm that leaves as parameters where the third (and later) diagonals should be queried. Then we optimize over possible values of these parameters in order to balance the resulting regions and so minimize the total number of queries.

We will need one subroutine that we haven't seen yet, a procedure to deal with triangles that doesn't use Procedure Diamond directly. Instead, this algorithm iterates the ideas within Procedure Diagonal with the parametric argument described above. Its input will be the three corners of the triangle. It is assumed that the triangle is an isosceles right triangle.

Procedure Triangle($(0,1),(0,0),(\frac{1}{2},\frac{1}{2})$: a^*) (see Fig. 6). This algorithm first queries the diagonal that is α of the way down the triangle (Procedure Diagonal uses $\alpha=\frac{1}{2}$). This diagonal divides the triangle into a triangle and a trapezoid. If the diagonal does not turn up a local optimum then the algorithm will either iterate on the new

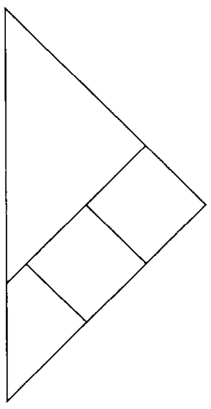
Fig. 6. Procedure Triangle.

triangle portion or it needs to deal with the trapezoid. Recall that Procedure Diagonal took care of the trapezoid by dividing it into a triangle and a diamond. Here, it first cuts it into two with a NW-SE diagonal β of the way from the first cut (Procedure Diagonal uses $\beta = \frac{1}{2}$). Now, since β doesn't necessarily equal $\frac{1}{2}$, this diagonal cut divides the trapezoid into a region that can be handled by Procedure Diamond and another trapezoid. This new trapezoid is again divided by a NW-SE diagonal, this time γ of the way down. Finally, this results in two regions, one of which can be taken care of with Procedure Diamond and the other by iterating Procedure Triangle.

Step 1. Line Query($y = x + (1-\alpha)$, $0 \le x < \alpha/2$: a^1).

Step 2. Check($a^1 : \bar{a}^1$).

Step 3. If $a^1 = \bar{a}^1$ then STOP: $a^* = a^1$. Otherwise,

Case 1: $a^1 = a_{i,j}$ and $\bar{a}^1 = a_{i,j-1}$ or $a_{i-1,j}$, then Procedure Triangle($(1,0), (0, 1-\alpha)$, $(\alpha, 1-\alpha)$: a^*) or

Case 2: $a^1 = a_{i,j}$ and $\bar{a}^1 = a_{i,j+1}$ or $a_{i+1,j}$, then go to Step 4.

Step 4. Line Query($y = -x + (1-2\beta)$, $-\beta + \alpha/2 \le x < \frac{1}{2} - \beta$: a^2).

Step 5. Check($a^2 : \bar{a}^2$).

Step 6. If $a^2 = \bar{a}^2$ then STOP: $a^* = a^2$. Otherwise,

Case 1: $a^2 = a_{i,j}$ and $\bar{a}^2 = a_{i,j+1}$ or $a_{i-1,j}$, then Procedure Diamond($(\alpha/2 - \beta$, $1 - \alpha/2 - \beta), (\alpha, 1-\alpha), (\frac{1}{2} - \beta, \frac{1}{2} - \beta), (\frac{1}{2}, \frac{1}{2})$: a^*) or

Case 2: $a^2 = a_{i,j}$ and $\bar{a}^2 = a_{i,j-1}$ or $a_{i+1,j}$, then go to Step 7.

Step 7. Line Query($y = -x + (1 - 2\beta - 2\gamma)$, $\min(0, -\frac{1}{4} - \beta/2 - \gamma/2 + \alpha/2) \le x < \frac{1}{2} - \beta - \gamma$: a^3).

Step 8. Check($a^3 : \bar{a}^3$).

Step 9. If $a^3 = \bar{a}^3$ then STOP: $a^* = a^3$. Otherwise,

Case 1: $a^3 = a_{i,j}$ and $\bar{a}^3 = a_{i,j+1}$ or $a_{i-1,j}$, then Procedure Diamond($(0, 1-\alpha)$, $(\alpha/2 - \beta, 1 - \alpha/2 - \beta), (\frac{1}{2} - \beta - \gamma, \frac{1}{2} - \beta - \gamma), (\frac{1}{2} - \beta, \frac{1}{2} - \beta)$: a^*) or

Case 2: $a^3 = a_{i,j}$ and $\bar{a}^3 = a_{i,j-1}$ or $a_{i+1,j}$, then Procedure Triangle($(0, 1-\alpha), (0,0)$, $(\frac{1}{2} - \beta - \gamma, \frac{1}{2} - \beta - \gamma)$: a^*).

Now we will use this procedure as a subroutine to get a parameterized form of Procedure Diagonal. This proceeds exactly like Procedure Diagonal except that once we are left with a triangle to analyze we use Procedure Triangle rather than the unparameterized version (which as mentioned above set α and β equal to $\frac{1}{2}$ and has no γ).

Procedure Para-Diagonal(Rows(1, *n*), Columns(1, *n*): *a)** (see Fig. 7). As mentioned above, this procedure starts out like Procedure Diagonal, querying the NE-SW diagonal and then the half NW-SE diagonal. Now it is left with an isosceles right triangle and so finishes by calling Procedure Triangle.

Step 1. Line Query($y=x$, $0\le x\le 1$: a^1).
Step 2. Check($a^1:\bar{a}^1$).
Step 3. If $a^1=\bar{a}^1$ then STOP: $a^*=a^1$. Otherwise, without loss of generality $a^1=a_{i,j}$ and $\bar{a}^1=a_{i-1,j}$ or $a_{i,j-1}$.
Step 4. Line Query($y=-x+1$, $0\le x\le\frac{1}{2}$: a^2).
Step 5. Check($a^2:\bar{a}^2$).
Step 6. If $a^2=\bar{a}^2$ then STOP: $a^*=a^2$. Otherwise, without loss of generality $a^2=a_{i,j}$ and $\bar{a}^2=a_{i,j-1}$ or $a_{i,j+1}$.
Step 7. Procedure Triangle($(1,0),(0,0),(\frac{1}{2},\frac{1}{2})$: a^*).

Theorem 3.1.4. *Procedure Para-Diagonal finds a local optimum of an* $n\times n$ *matrix in less than* $2.5445n+O(\log n)$ *queries.*

Proof. For this procedure to converge we must restrict the values of the various parameters. We will always require that α is between $\frac{1}{4}$ and $\frac{1}{2}$, β is at least $\frac{1}{2}-\alpha$, and γ is no more than $\frac{1}{2}-\alpha$. To analyze this, we must separately consider the four different ways that this algorithm can terminate. These are:

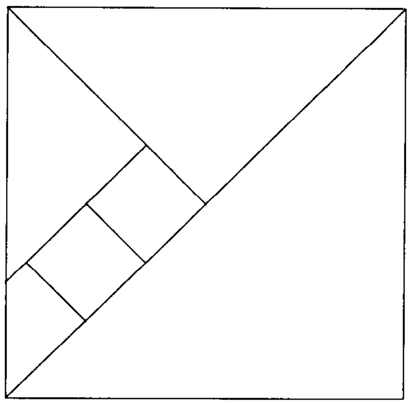
Fig. 7. Procedure Para-Diagonal.

(1) If Case 1 is always chosen in Step 3 of Procedure Triangle;
(2) at any time Case 1 is chosen in Step 6 of Procedure Triangle;
(3) at any time Case 1 is chosen in Step 9 of Procedure Triangle;
(4) Case 2 is chosen in Step 9 of Procedure Triangle.

First, it is clear that the largest number of queries will result in any of the three last choices if they occur at the first possible time, i.e., the first time that step of Procedure Triangle is encountered. Indeed, the number of queries will decrease the later they are chosen. Hence, we will analyze the possibilities above, with options (2)-(4) understood to mean that they actually occur at the first time that step is encountered in the first iteration of Procedure Triangle. Since we wish to optimize over choices of parameters α, β, and γ, we will first symbolically write down the number of queries in parametric form and then discuss possible values of these parameters. Note that the first six steps of Procedure Para-Diagonal require $1.5n+8$ queries. Hence, we will only analyze Procedure Triangle (with the inputs used in Step 7 of our procedure) and at the end will add these extra queries.

(1) Case 1 of Step 3 is always chosen (Procedure Triangle is iterated): Here it is straightforward that the total number of queries is less than $0.5\lceil\alpha/(\frac{1}{2}-\alpha)\rceil n+O(\log n)$.

(2) Case 1 of Step 6 is chosen: The procedure Diamond step will require less than $[2(\frac{1}{2}-\alpha)+\beta]n+O(\log n)$ queries by Corollary 3.1.2. The steps of Procedure Triangle leading up to this step require $[\alpha+(\frac{1}{2}-\alpha)]n+8$ queries. Hence this termination option requires less than $1.5n+[-2\alpha+\beta]n+O(\log n)$ queries.

(4) Case 2 of Step 9 is chosen: Here the triangle left to analyze requires no more than $\tau[\frac{1}{2}-\beta-\gamma]n$ queries where τkn is the number of queries needed by Procedure Triangle on an isosceles right triangle with side length kn. The steps of Procedure Triangle leading up to this step are as given in (3) above. Hence this termination option requires less than $n-\beta n+\gamma n+\tau[\frac{1}{2}-\beta-\gamma]n+O(\log n)$ queries.

As an example, consider the values of $\alpha=\frac{1}{3}$, $\beta=\frac{1}{4}$ and $\gamma=\frac{1}{6}$. The reader can easily verify that these satisfy our convergence requirements stated above. These values give the following results:

(1) $1.0n+O(\log n)$.
(2) $1.08n+O(\log n)$.
(3) $1.08n+O(\log n)$.
(4) Using $\tau n=1.25n$ from Procedure Diagonal (the number of queries less the first two query and check steps) above, gives $0.70n+O(\log n)$.

Using these values, we would have an upper bound of $2.58n+O(\log n)$ queries for the whole problem. Notice that we have almost accomplished complete balancing of the regions here. It is probably too much to ask for to also balance the last triangular region. Next, we discuss how to pick good values for α, β and γ.

In order to optimize the parameters α, β, and γ, we will write them each in terms of 0.5τ. What we will show is that the smallest possible 0.5τ we can get using this procedure is between 1.044 and 1.045.

Since we are trying to balance the different regions, what we will do is set each of the results (1)-(3), above equal to $0.5n\tau$. Then we will require that the quantity in (4) remains no more than $0.5n\tau$. We are suppressing the check steps and their $O(\log n)$ terms for clarity.

(1) $$0.5n\left[\frac{\alpha}{\frac{1}{2}-\alpha}\right]=0.5n\tau \;\Rightarrow\; \alpha=\left[\frac{0.5\tau}{1+2(0.5\tau)}\right].$$

(2) $$[1.5n-2\alpha+\beta]n=0.5n\tau \;\Rightarrow\; \beta=\left[\frac{2(0.5\tau)^2-1.5}{2(0.5\tau)+1}\right].$$

(3) $$[1.5-\alpha-\beta+\gamma]n=0.5n\tau \;\Rightarrow\; \gamma=\left[\frac{4(0.5\tau)^2-0.5\tau-3}{2(0.5\tau)+1}\right].$$

(4) $$1-\beta-\gamma-\tau[0.5-\beta-\gamma]\le 0.5\tau \quad\text{and}\quad 0.5\tau\ge 0.$$

This gives a third order equation to solve. The solution is $0.5\tau=1.0445$. Note that we must also make sure that our convergence ranges on the parameters are enforced ($0.25\le\alpha\le 0.5$, $\beta\ge 0.5-\alpha$, and $\gamma\le 0.5-\alpha$). Checking these with the above gives the information that $1\le 0.5\tau\le 1.075$. Hence we are within the necessary bounds. Using this value of $0.5\tau=1.0445$ gives

$$\alpha\cong 0.33813532,$$

$$\beta\cong 0.22077064,$$

$$\gamma\cong 0.10340596.$$

As these satisfy all of our requirements, this is the solution. The total number of queries for the $n\times n$ matrix is less than $[1.5+1.0445]n+O(\log n)=2.5445n+O(\log n)$, completing the proof of Theorem 3.1.4. □

3.2. Lower bounds

Here notice that we cannot use the results in Section 2.2 directly since more sophisticated query sets may be chosen by the player I with this adjacency structure. However, from our discussion on diagonals in Section 3.1, Theorem 2.2.2 does immediately give the following result.

Theorem 3.2.1. *It requires at least $n\sqrt{2}$ queries to find a local minimum of an $n\times n$ matrix.*

4. Conclusions

4.1. Conjectures

The bounds found in Sections 2 and 3 are summarized in Fig. 8 (logarithmic terms

Adjacency	Lower Bound	Upper Bound
King	$2n$	$3n$
Grid	$\sqrt{2}n$	$2.5445n$

Fig. 8. Queries to find a local optimum in an $n \times n$ square.

are disregarded here). We have substantially improved on the bounds of Theorem 1.1.1, but a gap persists for both adjacencies. In particular, we conjecture a lower bound of $3n$ for the king adjacency (this would imply $3/\sqrt{2}$ for the grid). Neither proof of Theorem 2.2.2 applies directly to this conjecture. The first proof, at the least, would require a new strategy for player II. (Against the given strategy, player I can achieve close to $2n$ by separating an $(n-2)\times(n-2)$ noncentered square from the rest of the $n \times n$ square.) To prove the conjecture with the second method, one would show that at least $12A/P$ queries are required to find a local optimum in a region of area A and perimeter P. However, this is false: consider a $1 \times \sqrt{2}$ rectangle. By Corollary 2.1.2, a local minimum can be found in $2+\sqrt{2}$ queries. Now $A=\sqrt{2}$, $P=2\sqrt{2}+2$; so $(2+2\sqrt{2})P/A=(2+\sqrt{2})^2=11.656\ldots<12$. We do conjecture that at least $8\sqrt{2}A/P$ queries are required to find a local optimum under the king adjacency. We also conjecture a lower bound of $2.5n$ for a square matrix under the grid adjacency.

Since completing an earlier draft of this paper, we have found that Althöfer and Koschnick [1] have independently studied the problem of local optimization on an m-dimensional grid. For $m=2$ (our grid adjacency case), their results reduce to

$$\frac{n^2}{4n+1}=\frac{n}{4}-\frac{1}{16}+\mathrm{o}(n)\le \tau(n)\le 4n+\mathrm{O}(\log n).$$

It would be interesting to see if Theorems 3.1.4 and 3.2.1 could be extended to $m=3$ or more dimensions to strengthen the bounds in [1].

4.2. Related problems

Local saddlepoints are related to local optima with the grid adjacency. We say a point (i,j) is a local saddlepoint iff

$$A(i\pm 1,j)<A(i,j)<A(i,j\pm 1),$$

i.e., $A(i,j)$ is larger than its two horizontal grid neighbors and smaller than its two vertical grid neighbors. Unlike local optima, local saddlepoints need not exist. Finding them turns out to be more costly, as the following theorem shows.

Theorem 4.2.1. *Any valid local saddlepoint algorithm requires at least $nm/4$ queries in the worst case for an $n \times m$ matrix.*

Proof. We play the adversary against an arbitrary valid algorithm. We let A be made of identical 2×2 submatrices as shown in Fig. 9. Each blank cell will have

	1		1		1
4	5	4	5	4	5
	1		1		1
4	5	4	5	4	5

Fig. 9. Adversary's matrix.

value either 3 or 7, but we do not decide which until it is queried. With this strategy, none of the fixed cells can be a local saddlepoint, and each unfixed cell is a local saddlepoint iff its value is 3. Now, as the algorithm makes queries, we respond with the fixed value for fixed cells, and with 7 for the unfixed cells, until the last unfixed cell is queried. Then we randomly decide on either 3 or 7. Obviously the algorithm must query all $nm/4$ unfixed cells to determine whether or not a local saddlepoint exists. □

Corollary 4.2.2. *It requires $\theta(n^2)$ queries to find a local saddlepoint.*

Proof. Obviously there exists a valid $O(n^2)$ algorithm, and the result follows. □

In a broader context, Theorem 4.2.1 displays a nice asymmetry between grid and row-column adjacencies. In the row-column adjacency, a cell is adjacent to all other cells in its row or column. That is, the graph has edge set

$$\{(i,j),(i',j')\} \in E \quad \Leftrightarrow \quad \min\{|i-i'|, |j-j'|\} = 0.$$

A (row-column) saddlepoint is the largest in its row and smallest in its column. Bounds for the different adjacencies are displayed in Fig. 10. For the grid adjacency, saddlepoints are more costly to find; but for the row-column adjacency, local optima are more costly. It would be interesting to find a general reason for the opposing behavior of the two neighborhoods.

	saddlepoint	local optimum
row-column	$\theta(n)$	$\theta(n^2)$
grid	$\theta(n^2)$	$\theta(n)$

Fig. 10. Comparison between grid and row-column adjacencies.

Acknowledgement

The authors gratefully acknowledge support from ONR grant N00014-88-K-0349 (Llewellyn) and NSF grant 84-ECS-8451032 (Tovey). The authors also thank Adam Rosenberg and Michael Trick for helpful discussions.

References

[1] I. Althöfer and K.-U. Koschnick, On the deterministic complexity of searching local maxima, Manuscript, Universität Bielefeld (1989); also: Discrete Appl. Math. 43 (1993) 111–113.
[2] D. Hausmann and B. Korte, Lower bounds on the worst-case complexity of some oracle algorithms, Discrete Math. 24 (1978) 261–276.
[3] R. Lipton, D. Rose and R. Tarjan, Generalized nested Dissection, SIAM J. Numer. Anal. 16 (1979) 346–358.
[4] D. Llewellyn, C. Tovey and M. Trick, Local optimization on graphs, Discrete Appl. Math. 23 (1989) 157–178.
[5] G. Nemhauser and L. Wolsey, Best algorithms for approximating the maximum of a submodular set function, Math. Oper. Res. 3 (1978) 177–188.