Discrete Applied Mathematics 23 (1989) 157-178
North-Holland

LOCAL OPTIMIZATION ON GRAPHS

Donna Crystal LLEWELLYN*, Craig TOVEY** and Michael TRICK***

Carnegie-Mellon University, Graduate School of Industrial Administration, Pittsburg, PA 15213, U.S.A., and Georgia Institute of Technology, School of Industrial and Systems Engineering, Atlanta, GA 30332-0205, U.S.A.

Received 18 June 1987
Revised 29 February 1988

The complexity of finding local optima is an open problem for many neighborhood structures. We show how to derive close lower and upper bounds on the minimum number of function evaluations needed to find a local optimum in an arbitrary graph. When these bounding techniques are applied to the hypercube, the results give insights into the class PLS and the gap between the average and worst-case behavior of local search.

1. Introduction

Finding local optima is at the heart of most heuristic algorithms for difficult combinatorial optimization problems. Hence, the efficiency of these heuristics is heavily dependent upon the effectiveness of the local optimization procedures. However, the complexity of finding local optima remains an open problem for most neighborhood structures. A classical example of this anomaly occurs with the traveling salesman problem. The "2-opt" neighborhood structure is very popular, but, as pointed out in [10], the complexity of finding a "2-optimal" tour is not known. Further, the most widely used local optimization method, local improvement, has been shown in certain cases to have exponential worst-case behavior [11].

In this paper we study the complexity of finding a local optimum of an arbitrary function, f, over an arbitrary neighborhood graph. As in [9, 14] our computational model employs a (not necessarily compactly representable) oracle to compute the values of f. We consider several simple structures—a path, a grid and a hypercube. The analysis of these structures is nontrivial and the results are often counter-intuitive. For example, we show that the problem of finding an entry in a square matrix that is minimum in its row and column requires, asymptotically, the examination of between $\frac{1}{2}$ and $\frac{2}{3}$ of the matrix entries. Further, for a d-dimensional cube, at least $2^d/2\sqrt{d}$ vertices must be examined.

We find it interesting that the problem of determining the complexity of local optimization proves so rich even for such regular structures. Moreover, the analysis

* Supported by NSF DMS84-14104 and ONR N00014-88-K-0349.
** Supported by NSF 84-ECS-8451032 and matching funds from the Digital Equipment Corporation.
*** Supported by NSF N00014-86-K-0173.

here yields insight into the class PLS (polynomial time local search - the class of local optimization problems, roughly speaking, where the first two iterations of the local improvement algorithm are guaranteed to be polynomial time (see [10])) and the relationship between worst and average case performance of local improvement algorithms. This wealth of problems arising from "simple" graphs leads us to study the general question of discrete local optimization in some detail.

Our analysis uses various techniques. We show how to derive close lower and upper bounds on the minimum worst-case number of function evaluations needed to find a local optimum in a graph. The upper bounds arise from a general divide and conquer algorithm while the lower bounds result from an adversarial argument that uses a structure similar to a spanning tree. Determining each of these bounds is ultimately tied to computing the value of a separation game on the graph of neighbors. Unfortunately, we also prove that this value is NP-hard to compute exactly. We discuss different ways to get effective estimates of this value in certain cases.

The paper is organized as follows: in Section 2 we derive the divide and conquer algorithm and the adversarial argument, using the path as a motivating example. Then we define the separation game. In Section 3 we analyze this game for a matrix. Section 4 contains the proof that determining the value of the game in general is NP-hard as well as a discussion of our methods for determining estimates of this value. In Section 5 we analyze the special case of the d-dimensional hypercube and investigate the implications and relations of this analysis to the class PLS, the expected performance of local improvement algorithms, and other results. We conclude with some remarks in Section 6.

2. Adversarial divide and conquer

2.1. Divide and conquer algorithm

In a typical combinatorial optimization problem, each instance has a finite set of feasible solutions and each solution is assigned a subset of these as its "neighbors". A (strict) local minimum has an objective function value which is less than or equal to (strictly less than) the values of each of its neighbors. We represent the solutions as vertices in a graph, $G=(V,E)$, and place edges between each vertex and its assigned neighbors, and define a local minimum to be a vertex v such that $(v,w)\in E\Rightarrow f(v)\leq f(w)$, where $f: V\rightarrow \mathbb{R}$ is the objective function to be minimized. Given a function evaluation oracle for function f we are interested in determining the number of calls to the oracle needed to find a local optimum of f in G. We will assume, without loss of generality, that we are looking for a local minimum.

The most widely applied solution method is local search, using either a best or better neighbor selection rule (see, e.g., [15, 18]). As an example, let the graph $G=(V,E)$ be a simple path; that is, $V=\{v_1,\ldots,v_n\}$ and $E=\{(v_i,v_{i+1})\,|\,1\leq i<n\}$.

It is clear that any variation of local search can require asking the function value of O(n) vertices. However, a divide and conquer strategy can find a local optimum with only O($\log n$) queries. To see this, first ask the function value of $v_{\lfloor n/2 \rfloor}$, followed by its neighbors, $v_{\lfloor n/2 \rfloor - 1}$ and $v_{\lfloor n/2 \rfloor + 1}$. If $f(v_{\lfloor n/2 \rfloor})$ is the minimum of these three values, then we may stop. Otherwise, a local minimum is located within the half of the path which contains the smaller valued neighbor. Iterate this procedure on this (smaller) graph. Clearly, this will require at most $3 \log n$ queries.

We now generalize the above divide and conquer scheme to all graphs. Whereas in the path example a single vertex sufficed to split the graph into smaller cases. In general we will need a more complex separator set. This is the only change in the procedure below.

Divide and conquer algorithm.

Input. Graph $G=(V,E)$ and function $f: V \rightarrow \mathbb{R}$.
Output. $v^* \in V$ such that $f(v^*) \leq f(w)$ for all $(v^*, w) \in E$.

```
Step 0. Let $i=0$, $G_0=G$.
Step 1. Select vertices one at a time to be submitted to the oracle until this collection of vertices separates $G_i$.[1] Call this collection $S$.
Step 2. Find $v' \in S$ that minimizes $f(v)$ for all $v \in S$, breaking ties arbitrarily.
Step 3. Query the neighbors of $v'$. Call these vertices $N$.
Step 4. If $f(v') \leq f(w)$ for all $w \in N$, return $v^* = v'$ and stop.
Step 5. Let $w \in N \setminus S$ be such that $f(w) < f(v')$. Let $G_{i+1}$ = the connected component of $(G_i \setminus S)$ containing $w$ and let $i = i+1$.
Step 6. Go to Step 1.
```

Lemma 2.1. *The divide and conquer algorithm finds a local minimum of f, in G.*

Proof. If local search were carried out starting with w (chosen in Step 5) then it could never select a vertex in S because $f(w) < f(v') \leq f(u)$ for all $u \in S$. Hence the algorithm could never leave G_i and so must find a local minimum in G_i. Thus there exists a local minimum in the component chosen in Step 5 and the desired result now follows immediately. □

Let the number of queries required by our divide and conquer algorithm for graph G and function f be denoted $\mathrm{dc}(G,f)$. It is clear that $\mathrm{dc}(G,f)$ provides an upper bound on the number of queries needed in the worst case by any method to solve this problem. Given function f, let $L(G,f,A)$ be the minimum number of queries that algorithm A requires to find a local optimum of f in graph G (in the worst

[1] We say a set of vertices $S \subseteq V$ separates $G=(V,E)$ if there exist two vertices u and $v \in V$ such that every path connecting u and v passes through S. A path $P=(v_1, \ldots, v_l)$ passes through a set of vertices S if $\{P \setminus \{v_1, v_l\}\} \cap S \neq \emptyset$.

case). Then let

$$L(G,A)=\max_f L(G,f,A) \quad \text{and} \quad L(G)=\min_A L(G,A).$$

Similarly, let

$$\mathrm{dc}(G)=\max_f \mathrm{dc}(G,f).$$

It turns out that in general, dc(G) provides a very close upper bound on $L(G)$. To see this it is first necessary to turn to the problem of determining a lower bound on $L(G)$. We will show that this lower bound is usually close to dc(G) and hence it must follow that in these cases dc(G) is close to $L(G)$.

2.2. *An adversarial argument*

Suppose the function evaluation oracle is an adversary who attempts to stymie the optimizer. This adversary will provide function values that cause the person trying to find a local minimum to continue to query for a long time. If the oracle can force the minimizer to ask for $g(n)$ values, then $g(n)$ must be a lower bound on the worst case number of queries for the graph. We give below a specific adversarial strategy that we will apply to get a lower bound.

First we show how the adversary would work on the path example. When the optimizer queries a vertex, v, this vertex automatically splits the path into two subpaths, say P_a and P_b, each containing v. The oracle then selects one of these, say subpath P_a, as the part of the graph where the local optimum will be located. In choosing P_a, she has decided not to have any local minima of f within P_b. To insure this, she sets function values for each of the nodes in P_b that decrease along the path to v. Now whenever the optimizer queries a node in P_b, the oracle simply responds with the preset value; thus the optimizer is wasting time with any queries within P_b. For any queries in P_a, the oracle can now repeat this process since any selected vertex in P_a breaks this path into two subpaths.

In this example, it is a simple exercise to see that the adversarial strategy gives a bound of $\log n$ queries for any choice of first vertex. This follows from the observation that the oracle will pick P_a so that it is at least as long as P_b. Hence, P_a will have at least $\lfloor \frac{1}{2}n \rfloor$ vertices. Iterating this process clearly gives a lower bound of $\lfloor \log n \rfloor$ queries.

To generalize the path example, note that as long as the graph is connected, the oracle can form a "valley" whose deepest point is at the chosen minimum. When the graph becomes disconnected, this valley must be generalized to a network of deepening ravines. This is the core of our adversarial strategy given below. We need the following definition. A *bare tree* of G is an acyclic connected subgraph, T, of G such that all vertices of G are in or adjacent to T. (In our implementation a bare tree will be a spanning tree without its leaves, hence the name.) Then our strategy follows.

Adversarial strategy.

Step 0. Let $i=0$ and $G_0=G$. Let w_0 be any integer.

Step 1. Give the default value w_i to any vertex asked until there is no bare tree of G_i using only unqueried vertices. Let the last vertex queried be v^i. Let the newly formed components of the subgraph induced by the unqueried vertices be denoted $C_{i1},\dots,C_{ik}$. In G_i, v^i is adjacent to each of these components.

Step 2. Choose a bare tree T of G_i, using only v^i and unqueried vertices. Removing v^i disconnects T into $T_1,\dots,T_k$, where T_r is a bare tree of C_{ir}. Choose some j, $1\le j\le k$. In C_{ir}, $r\ne j$, consider bare tree T_r. For vertex $u\notin C_{ij}$, $u\notin T_r$ for any r, and u not queried, let $f(u)=w_i$. Along each branch of each T_r, pick function values less than w_i decreasing toward v^i. Let

$$f(v^i)=\min\{f(u)\mid u\in C_{ik} \text{ for any } k\}-1.$$

Let $w_i=f(v^i)-1$.

Step 3. Let $i=i+1$. Let $G_i=C_{i-1,j}$.

Step 4. Go to Step 1.

Lemma 2.2. *If the oracle responds according to the above strategy, then as long as G has more than one vertex, it cannot be determined where G has a local minimum.*

Proof. This follows directly from our comment above that as long as a graph is connected, the oracle can devise a network of deepening ravines toward any selected vertex. □

2.3. *The separation game*

In order to obtain a good lower bound on $L(G)$ from Lemma 2.2 we need to analyze the adversarial strategy. It is interesting that analyzing this strategy is very similar to analyzing the divide and conquer algorithm. This is because each procedure has the following two person game embedded within its structure.

Separation game

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

Two players. minimizer I, maximizer II.

Description. Player I removes vertices from G until G is disconnected and then passes play to II. Player II chooses one of the newly created components to call G, discards the other components and passes G back to I. The game ends when $|V|\le 1$.

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

Step 2. If $|V^i|\le 1$ go to Step 6.

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 chooses G^{i+1}, a connected component of $G^i \setminus S^i$; $i=i+1$.
Step 5. Go to Step 2.
Step 6. Stop.

We define the value of the separation game on a graph G, denoted $v(G)$, to be score(G) when each player plays optimally, I to minimize score(G) and II to maximize.

Lemma 2.3. *The value of the separation game provides a lower bound on the amount of work required by any algorithm to find a local minimum on graph G; while the amount of work required by the divide and conquer algorithm provides an upper bound.*

Proof. This lemma is equivalent to claiming that $v(G) \leq L(G) \leq \mathrm{dc}(G)$. The second inequality follows from Lemma 2.1 and the definitions. Further, in any application of the adversarial strategy described in Section 2.2, the oracle can select the component of G that is chosen in the optimal play of the separation game. Hence, $v(G)$ provides a lower bound on the number of function queries forced by the oracle in the adversarial strategy, which is less than $L(G)$ by Lemma 2.2. □

This lemma shows that if the gap between $v(G)$ and dc(G) is small then either one will provide a good estimate of $L(G)$. In the path example, we saw that one implementation of the divide and conquer algorithm (choosing the midpoint of the path at each iteration) gives a $3 \log n$ upper bound on dc(G) and that the adversarial strategy gives a lower bound of $\log n$. The difference between these two can be viewed in two ways. First, the upper bound is three times the lower bound; second, the gap is logarithmically small. In general, the latter view is the correct one. In particular, the gap is equal to the maximum degree of a vertex (2) times the number of times Step 3 of the divide and conquer algorithm is needed ($\log n$). Actually, it is foolish for the adversary to select either subpath P_a or P_b immediately. Rather, she should wait for the next query and then select the subpath that does not contain this queried vertex. Hence the optimal strategy is not to naively pick the midpoint, but rather to slightly unbalance the two subpaths by doing Fibonacci search. It is straightforward to prove that this gives the optimal value of the separation game to be $\log_\phi n + 2$; where ϕ is the golden mean ($\frac{1}{2}(\sqrt{5}-1)$) and the 2 arises from the extra queries to test the local optimality of the final selection.

In the next section we investigate another example where we are able to establish fairly tight bounds on $L(G)$. We will use the following result.

Proposition 2.4. *In the separation game S^i can always be taken to be a minimal separating set of G. That is, for any $S \subset S^i$, $G^i \setminus S$ is connected.*

Proof. The removal of any nodes not in a minimal separating set can be delayed one iteration, at which time they can be removed (if in the component chosen by player II) keeping score(G) the same, or ignored (otherwise) decreasing score(G). □

3. Finding a row-column minimum in a matrix

We now apply the ideas of Section 2 to a specific local optimization problem. Suppose A is an $n \times n$ matrix: how hard is it to find an entry that is less than or equal to every other entry in its row and column? It is clear that one can play the adversary against a local improvement algorithm so as to force every entry of the matrix to be queried. We show that asymptotically, one needs only to query between one half and two thirds of the matrix, and we conjecture that the upper bound of $\frac{2}{3}$ is asymptotically tight.

Following our development in Section 2, the graph of neighborliness, G, is formed by defining a node for each matrix entry with an edge between two nodes if and only if they are either in the same row or column. Using Lemma 2.3, we will attempt to find $\upsilon(G)$ in order to obtain a good estimate of $L(G)$. However, G is dense and difficult to visualize. Therefore, we move to an auxiliary graph. Consider the complete bipartite graph $H=(R \cup C, E)$, where $R=\{1,\dots,n\}$ and $C=\{1,\dots,n\}$. In this graph we have one node for each row and for each column, with an edge connecting each row and column pair. Here, each edge corresponds to an element in the matrix, and so we move to an edge version of our separation game where player I removes edges from the auxiliary graph rather than deleting vertices from the original graph. Other than this change, the game is played in the same manner as the original (vertex) separation game.

Proposition 3.1. *The value of the* (*vertex*) *separation game played on the original graph is the same as the value of the edge separation game played on the auxiliary graph.*

Proof. We noted above that edges in the auxillary graph correspond to vertices of the original graph. This implies that a separating set of vertices in G is a disconnecting collection of edges in H. Further a disconnecting set of edges in H must correspond to a separating set of vertices in G unless one of the resulting components in H is an isolated vertex. However, this case will never arise in our application since player II would never choose such a component before the last iteration of the game. □

We now analyze this game. For ease of presentation, we will relax the assumption that $|R|=|C|$; but will always assume that $m=|R| \leq |C|=n$. Suppose that player I chooses subset F of the edges to separate graph $H=(R \cup C, E)$. By Proposition 2.4, we can assume that F is a minimal separating set. Then clearly, since F is a set of edges, it must disconnect H into exactly two components, each a complete bipartite graph. The components can be represented by (J, K) and $(m-J, n-K)$, where in each case the ordered pair corresponds to the number of row vertices and the number of column vertices in that subgraph. Clearly, player II will return to player I the component which requires more work to finish disconnecting. Let $f(m,n)$ be the value of the edge separation game on the complete bipartite graph H. Where, for general r and s, $f(r,s)$ is to be read as $f(\min(r,s), \max(r,s))$. The discussion above implies

$$f(m,n) = \min_{1\le J\le m} \min_{1\le K\le n} \{[\max(f(J,K), f(m-J, n-K))] + J(n-K) + K(m-J)\}.$$

To simplify this, let $K(J,m)$ be the smallest integer such that

$$f(J, K(J,m)) \ge f(m-J, n-K(J,m)).$$

Thus, we can assume without loss of generality that F is chosen in such a way that $K \ge K(J,m)$ and hence that player II returns the (J,K) component. This implies

$$\begin{aligned} f(m,n) &= \min_{1\le J\le m} \min_{K(J,m)\le K\le n} \{f(J,K) + J(n-K) + K(m-J)\} \\ &= \min_{1\le J\le m} \min_{K(J,m)\le K\le n} \{f(J,K) + Jn + K(m-2J)\}. \end{aligned}$$

However, if we begin with $m \le n$ then in order to maintain this condition of no more row vertices than column vertices, we must assume that $J \le \lfloor \frac{1}{2}m \rfloor$. Now it is clear that for $J > \lfloor \frac{1}{2}m \rfloor$, switching the roles of R and C will give a symmetric argument. Hence we now assume $J \le \lfloor \frac{1}{2}m \rfloor$, giving

$$f(m,n) = \min_{1\le J\le \lfloor m/2 \rfloor} \min_{K(J,m)\le K\le n} \{f(J,K) + Jn + K(m-2J)\}.$$

Further, it follows from the definition that for $J \le \lfloor \frac{1}{2}m \rfloor$, $f(J,K)$ is nondecreasing in K. Hence,

$$f(m,n) = \min_{1\le J\le \lfloor m/2 \rfloor} \{f(J, K(J,m)) + Jn + K(J,m)(m-2J)\}.$$

We now need to determine some effective bounds on $K(J,m)$.

Proposition 3.2. *If $m \le m'$ and $n \le n'$ then $f(m,n) \le f(m', n')$.*

Proof. This is clear by definition. □

Proposition 3.3. $K(\lfloor \frac{1}{2}m \rfloor, m) \ge \lceil \frac{1}{2}n \rceil$.

Proof. By definition, $f(\lfloor \frac{1}{2}m \rfloor, K(\lfloor \frac{1}{2}m \rfloor, m)) \ge f(\lceil \frac{1}{2}m \rceil, n - K(\lfloor \frac{1}{2}m \rfloor, m))$. There are two cases.

(1) $\lceil \frac{1}{2}m \rceil \le n - K(\lfloor \frac{1}{2}m \rfloor, m)$. Now, $\lfloor \frac{1}{2}m \rfloor \le \lceil \frac{1}{2}m \rceil \Rightarrow$ by Proposition 3.2 that

$$K(\lfloor \tfrac{1}{2}m \rfloor, m) \ge n - K(\lfloor \tfrac{1}{2}m \rfloor, m) \Rightarrow K(\lfloor \tfrac{1}{2}m \rfloor, m) \ge \tfrac{1}{2}n.$$

(2) $\lceil \frac{1}{2}m \rceil \geq n - K(\lfloor \frac{1}{2}m \rfloor, m)$. Then, by definition $f(\lfloor \frac{1}{2}m \rfloor, K(\lfloor \frac{1}{2}m \rfloor, m) \geq f(n - K(\lfloor \frac{1}{2}m \rfloor, m), \lceil \frac{1}{2}m \rceil)$.

(a) $n - K(\lfloor \frac{1}{2}m \rfloor, m) \leq \lfloor \frac{1}{2}m \rfloor \Rightarrow K(\lfloor \frac{1}{2}m \rfloor, m) \geq n - \lfloor \frac{1}{2}m \rfloor \geq \frac{1}{2}n$.

(b) $n - K(\lfloor \frac{1}{2}m \rfloor, m) > \lfloor \frac{1}{2}m \rfloor$. Then, by Proposition 3.2,

$$K(\lfloor \tfrac{1}{2}m \rfloor, m) \geq \lceil \tfrac{1}{2}m \rceil \Rightarrow n - K(\lfloor \tfrac{1}{2}m \rfloor, m) < \lfloor \tfrac{1}{2}m \rfloor$$

which is a contradiction.

In each of these cases, the integrality of $K(J, m)$ gives the desired result. □

Using a similar argument, one can also prove the following upper bound.

Proposition 3.4. $K(\lceil \frac{1}{2}m \rceil, m) \leq \lfloor \frac{1}{2}n \rfloor + 1$.

Corollary 3.5. *If m is even then $K(\frac{1}{2}m, m) = \lceil \frac{1}{2}n \rceil$.*

Proof. Using Proposition 3.3 and Proposition 3.4, clearly $\lceil \frac{1}{2}n \rceil \leq K(\frac{1}{2}m, m) \leq \lfloor \frac{1}{2}n \rfloor + 1$. Consider the case when n is odd. This gives $\lceil \frac{1}{2}n \rceil \leq K(\frac{1}{2}m, m) \leq \lceil \frac{1}{2}n \rceil$ and we are done.

Now consider when n is even. In this case, the two components resulting from the removal of F are $(\frac{1}{2}m, K(\frac{1}{2}m, m))$ and $(\frac{1}{2}m, n - K(\frac{1}{2}m, m))$. Clearly $\frac{1}{2}n$ will suffice to make the first subgraph at least as hard to finish as the second and hence by minimality of $K(J, m)$ we must have $K(\frac{1}{2}m, m) = \frac{1}{2}n = \lceil \frac{1}{2}n \rceil$. □

Now, we can use these bounds to determine bounds on $f(m, n)$.

Proposition 3.6. *The value of the edge separation game played on the complete $m \times n$ bipartite graph is greater than or equal to $\lceil \frac{1}{2}mn \rceil + \lfloor \frac{1}{2}n \rfloor$.*

Proof. To prove this we will first show that $f(m, n) \geq \lceil \frac{1}{2}mn \rceil + 1$. Then, using this iteratively gives the desired result. By definition,

$$\begin{aligned} f(m,n) &= \min_{1 \leq J \leq \lfloor m/2 \rfloor} \{f(J, K(J,m)) + J(n - K(J,m)) + K(J,m)(m - J)\} \\ &= \min_{1 \leq J \leq \lfloor m/2 \rfloor} \{f(J, K(J,m)) + Jn + K(J,m)(m - 2J)\}. \end{aligned}$$

Note that for all $J, K > 0$, $f(J, K) \geq 1$, hence,

$$f(m,n) \geq 1 + \left[\min_{1 \leq J \leq \lfloor m/2 \rfloor} \{K(J,m)(m - 2J) + Jn\} \right].$$

It is clear that $K(J, m)$ is nonincreasing in J, hence

$$f(m,n) \geq 1 + \min_{1 \leq J \leq \lfloor m/2 \rfloor} \{K(\lfloor \tfrac{1}{2}m \rfloor, m)(m - 2J) + Jn\}.$$

Using Proposition 3.3 and treating the four possible cases of parities of m and n separately we get,

$$\begin{aligned} f(m,n) &\geq \min_{1 \leq J \leq \lfloor m/2 \rfloor} \{1 + \lceil \tfrac{1}{2}n \rceil (m-2J) + Jn\} \\ &\geq 1 + \lceil \tfrac{1}{2}n \rceil m + 1(n - 2\lfloor \tfrac{1}{2}n \rfloor) \geq 1 + \lceil \tfrac{1}{2}n \rceil m \\ &\geq 1 + \lceil \tfrac{1}{2}mn \rceil \quad \text{as desired.} \end{aligned}$$

Now, plugging this in for $f(J, K(J,m))$ instead of the term of 1 used above and using the fact as above that in this range $K(J,m) \geq \lceil \frac{1}{2}n \rceil$ and $J \geq 1$ gives the next result that $f(m,n) \geq 1 + \lceil \frac{1}{2}mn \rceil + \lceil \frac{1}{4}n \rceil$. Iterating then yields the result. □

Applying induction on m and n, it is straightforward to obtain effective upper bounds on the value of the edge separation game in some special cases.

Proposition 3.7. *If m and n are powers of 2, with $m \leq n$, then the value of the edge separation game played on the complete $m \times n$ bipartite graph has an upper bound of* $\lfloor \frac{2}{3}mn + \frac{1}{3}n/m \rfloor$. □

Proposition 3.8. *The bound given in Proposition 3.7 does not necessarily hold if m or n is not a power of 2.*

Proof. First, consider $f(2,3)$. From Corollary 3.5, $K(1,2) = 2$, and clearly $f(1,n) = n$ for all $n \geq 1$. Thus,

$$f(2,3) = f(1,2) + 1(3-2) + 2(2-1) = 2 + 1 + 2 = 5.$$

But, $\lfloor \frac{2}{3}mn + \frac{1}{3}n/m \rfloor = \lfloor \frac{2}{3} \cdot 6 + \frac{3}{6} \rfloor = 4 < 5$.

Next, consider $f(3,16)$. It is straightforward to prove that $K(1,3) \geq \lceil \frac{3}{5}n \rceil$. Hence, here $K(1,3) \geq 10$. Thus,

$$\begin{aligned} f(3,16) &\geq f(1,10) + 1(16-10) + 10(3-1) \\ &= 10 + 6 + 20 = 36. \end{aligned}$$

But,

$$\lfloor \tfrac{2}{3}mn + \tfrac{1}{3}n/m \rfloor = \lfloor \tfrac{2}{3} \cdot 48 + \tfrac{16}{9} \rfloor = 33 < 36. \quad \square$$

However, with careful application of divide and conquer, and induction, we can obtain an upper bound on the value for all m and n. The proof entails several cases. The nature of the proof in each individual case depends only on whether certain components (arising from the divide and conquer status) have an even number of vertices.

Proposition 3.9. *For all m and n, with $m \leq n$, the value of the edge separation game played on the complete $m \times n$ bipartite graph has an upper bound of* $\lfloor \frac{2}{3}mn + \frac{1}{3}n \rfloor$.

Proof. We will show that $f(m,n) \le \frac{2}{3}mn + \frac{1}{3}n$ and the result will follow by integrality of f.

We need to break this into several cases. The idea in each is the same. We try to decompose each vertex set of the bipartite graph into equal parts and analyze the total work involved. We will be getting an upper bound by analyzing the work needed to solve the problem using the following specific strategy: Let $R = A \cup B$ and $C = S \cup T$. (Hence $|A| + |B| = m$ and $|S| + |T| = n$.) Then we will first disconnect A from T which requires $|A|\,|T|$ edges removed, then we will disconnect B from S which requires $|B|\,|S|$ edges removed. Then we finish by choosing the component that requires the most work, among (A, S) and (B, T). The total work needed then is

$$|A|\,|T| + |B|\,|S| + \max\{f(A,S), f(B,T)\}.$$

Using induction on m and n implies

$$f(m,n) \le |A|\,|T| + |B|\,|S| + \max[\tfrac{2}{3}|A|\,|S| + \tfrac{1}{3}|S|, \tfrac{2}{3}|B|\,|T| + \tfrac{1}{3}|T|].$$

Here, if one analyzes the cases m and n are both even, m is odd and n is even, m is even and n is odd, and m and n are both odd, using our results above, it is clear that the desired result will follow. □

Putting together Propositions 3.6 and 3.9, we have proved the following theorem.

Theorem 3.10. *The value of the edge separation game played on a complete $m \times n$ bipartite graph has a lower bound of $\lceil \frac{1}{2}mn \rceil + \lfloor \frac{1}{2}n \rfloor$ and an upper bound of $(\frac{2}{3} + o(1))mn$.*

Corollary 3.11. *The amount of work required to find an element that is smallest in its row and its column in an $m \times n$ matrix, with $m \le n$, is greater than $\lceil \frac{1}{2}mn \rceil$. Further, our divide and conquer algorithm requires no more than $\frac{2}{3}mn + \frac{1}{3}n + 2n \log m$ work to solve this problem.*

Proof. From Theorem 3.10, we see that $L(G) > \lceil \frac{1}{2}mn \rceil$ for all $m \le n$. The other result will follow from Theorem 4.7 in the next section. □

Conjecture 3.12. *$v(G) \ge \frac{2}{3}mn + \frac{1}{3}n/m$ for all $m \le n$.*

We have verified this conjecture for all $m \le n \le 64$.

4. Computing the value of the separation game

We have seen in Lemma 2.3 that the value of the separation game on G is useful since it provides a lower bound on the number of function evaluations required by any algorithm, and a fairly close estimate on the work required by the divide and conquer algorithm. However, we have also seen in Section 3 that analyzing even fairly simple graphs can be difficult. In this section, we establish the complexity of determining $\upsilon(G)$ and we show how to obtain bounds on this value.

4.1. Complexity of computation

Despite the apparent simplification provided by Proposition 2.4, it is not easy to determine player I's optimal strategy. It seems plausible that player I wishes to separate G^0 into roughly equal parts at minimum cost, for if G^0 where separated into unequal parts then player II could simply choose the larger component. Unfortunately, the size of a component is a poor measure of the value of the game. For instance, the graphs P_n (a path on n vertices) and K_n (the complete graph on n vertices) have the same number of vertices yet the separation games on these graphs have very different values. To see this, from Section 2 we know that $\upsilon(P_n) \leq \log n$, while in contrast it is easy to establish that $\upsilon(K_n) = n-1$.

This complication is formalized by showing that calculating $\upsilon(G)$ is NP-hard. To do this, we will show that to play optimally during the first two moves, player I must solve an NP-hard problem. The precise objective for player I's first move is to choose S^0 to minimize $|S^0| + \max_i \upsilon(G_i^1)$ where the G_i^1 are the components of $G^0 \setminus S^0$.

Notice that (assuming the complexity result is true) $\upsilon(G_i^1)$ is hard to evaluate! This is one of those curious cases where the complexity of the problem fights against a proof of its complexity. To resolve this, we will find a structure such that the values of the graphs that result after the first few moves are known.

We need an intermediate result on a restricted version of vertex cover: A tripartite graph is a graph $G = \{V_1, V_2, V_3; E_{12}, E_{13}, E_{23}\}$ where V_i, $i = 1,2,3$ are disjoint vertex sets and $(i,j) \in E_{ab} \Rightarrow i \in V_a$ and $j \in V_b$. That is, a tripartite graph is a graph with an explicit 3-coloring.

Recall that the vertex cover problem is to find a minimum cardinality subset of vertices of a graph which contains at least one endpoint of each edge.

Theorem 4.1. *Vertex Cover restricted to tripartite graphs is NP-complete.*

Proof. The reduction is a mofidication of the reduction of [7] from 3-SAT to Vertex Cover. Rather than 3-SAT, we will use another restricted form of SAT, proved NP-complete in [17]: SAT restricted by (1), (2), and (3) below is NP-complete:

(1) Each clause has 2 or 3 literals;
(2) Each variable appears in at most 3 clauses;
(3) Each variable appears complemented exactly once.

If the reduction of [7] for 3-SAT to Vertex Cover is used with the above SAT restriction (where edges replace triangles for clauses with 2 literals) then a 3-colorable graph results. The 3-coloring can be found in a greedy fashion, provided the nodes are colored in the order: clause triangles, clause edges (using just colors 1 and 2), uncomplemented variables, and complemented variables. □

We are about to prove that the separation game value is hard to calculate. Our construction makes use of a structure we call a *spider*. Define an *(h, w, t)-spider* to be a graph with vertex set

$$V=\{v_1,\dots,v_h,u_1,\dots,u_w,x_1,\dots,x_t\}$$

and edge set

$$\begin{aligned}E=\{(v_i,v_j)\colon 1\le i<j\le h\}&\cup\{(u_i,u_j)\colon 1\le i<j\le w\}\cup\{(x_i,x_j)\colon 1\le i<j\le t\}\\&\cup\{(v_i,u_j)\colon 1\le i\le h,\ 1\le j\le w\}\\&\cup\{(u_i,x_j)\colon 1\le i\le w,\ 1\le j\le t\}.\end{aligned}$$

So a spider is three complete graphs K_h, K_w, K_t, which we call the head, waist and tail respectively, where each node in the waist is connected to each node in the head and tail.

Proposition 4.2. *If G is an (h, w, t)-spider then the value of the separation game on G is equal to $w+\max(h,t)-1$.*

Proof. The waist is the only minimal separating set so, by Proposition 2.4, player I chooses it as the first move. This leaves K_h and K_t. Since $v(K_p)=p-1$, player II will choose the larger. □

Theorem 4.3. *For a graph G and integer k, determining if the value of the separation game $v(G)\le k$ is NP-complete.*

Proof. The problem is in NP since we can completely specify player I's strategy with one label per node; where a node receives label i if player I plans to remove it during his ith turn (if player II has played so that this node is in V^i).

To show completeness, we will reduce from the result in Theorem 4.1. We will take three spiders and connect their heads with a "web" of arcs forming a tripartite cover instance. We then show that player I's optimal strategy is to separate the spiders and to use a single spider, for which we know the optimal strategy. Separating the spiders optimally involves solving the tripartite vertex cover problem.

Let $G=(V_1, V_2, V_3, E_{12}, E_{13}, E_{23}; k)$ be an instance of vertex cover restricted to tripartite graphs. Set $T=|V_1|+|V_2|+|V_3|$.

Create these spiders H^i, $i=1,2,3$, where H^i is a $(|V_i|+T, k, 2T)$-spider. Denote the first $|V_i|$ vertices in the head of H^i, v_j^i for $j=1,2,\dots,|V_i|$ for $i=1,2,3$. Put edges between these vertices corresponding to G and call this graph H.

We claim $v(H)\le 2(T+k)-1 \Leftrightarrow G$ has a vertex cover of size $\le k$.

($\Leftarrow$) If G has a vertex cover C, with $|C|\le k$, then player I can choose S^0 as the nodes corresponding to C. This disconnects H into 3 spiders, each with a larger tail than head. By Proposition 4.2, each has value $k+2T-1$ so $\upsilon(H)\leq|S^0|+k+2T-1\leq 2(T+k)-1$ as needed.

($\Rightarrow$) By Proposition 2.4, player I's first move removes either

(i) a waist, leaving a component containing two connected spiders. This component requires at least $k+2T$ deletions because a spider has value $k+2T-1$ and at least one node is needed to disconnect the spiders. This gives $\upsilon(H)\geq 2(T+k)$. Contradiction.

(ii) nodes attached to the web, separating one spider from the other two. Two cases:

(a) if this separates all three spiders then, since a spider requires $k+2T-1$ deletions, no more than k nodes separated the spider, which corresponds to a vertex cover of size $\leq k$.

(b) if two spiders are still connected, then player II can pick that component. Player I's response is either a waist (which leads to the contradiction in (i)) or a separation of the spiders (which leads to the calculation in (ii)(a)) hence proving the claim. □

4.2. *Boundary theorems and lower bounds*

Theorem 4.3 implies that we cannot find the exact value of the separation game for an arbitrary graph by any known efficient algorithm. Hence, we turn to developing tools for estimating $\upsilon(G)$; in this section we illustrate how to apply boundary theorems to obtain lower bounds on $\upsilon(G)$.

Theorem 4.4. *If the separation game is played on graph $G=(V,E)$, then for all integers k and t, $0\leq t\leq k\leq|V|$, player* II *can force player* I *to either*

(i) *delete at least t vertices, or*

(ii) *create a set of components with total cardinality s, with $k-t\leq s\leq k$.*

Proof. Consider the following strategy for player II:

Step 1. Let $B=\emptyset$, $P=G$.

Step 2. Let player I separate P.

Step 3. Pick an order of the resulting components, say $P_1, P_2, \ldots, P_p$ and let $P_\emptyset=\emptyset$ and $P_{p+1}=P$.

Step 4. Let $i\geq 1$ be such that

$$|B\cup P_0\cup P_1\cup\cdots\cup P_{i-1}|\leq k \quad\text{and}\quad |B\cup P_0\cup P_1\cup\cdots\cup P_i|>k.$$

Step 5. Let $B'=B\cup P_0\cup P_1\cup\cdots\cup P_{i-1}$. If $k-t\leq|B'|\leq k$ go to Step 6; if $i=p+1$ go to Step 7; otherwise let $P=P_i$, $B=B'$ and go to Step 2.

Step 6. Stop, B' is a subset of vertices with $k-t\leq|B'|\leq k$.

Step 7. Stop, player I has chosen t vertices.

The termination in Step 7 is correct due to the following:

By construction, at Step 2 for each iteration $|B| \leq k-t \leq k < |B \cup P|$ and, for $i=p+1$ in the final Step 5, $|B'| \leq k-t$. But

$$P_0 \cup P_1 \cup \cdots \cup P_p = P \setminus S,$$

where S is the separating set chosen by player I in Step 2. So

$$|S| = |B \cup P| - |B'| > k-(k-t) = t.$$

This means player I has removed at least t vertices as required. □

Let $B(S)$ be the *boundary* of $S \subseteq V$. That is,

$$B(S) = \{v \in V \setminus S : \exists e = (v, w),\ w \in S\}.$$

By fixing t, we get the following corollary:

Corollary 4.5. *For any graph G and integer t,*

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

Proof. From Theorem 4.4, for any k, either t nodes are removed or a set of components with total cardinality s, $k-t \leq s \leq k$, is created. To create such a set, the boundary of that set must have been among the nodes chosen by player I. Hence, in this case, player I removes at least $\min\{|B(S)| : k-t \leq |S| \leq t\}$ nodes. Since Theorem 4.4 holds for all k, the maximum over all of these values is a lower bound on the value of the game. □

One useful value for t in the Corollary 4.5 is as follows. Let

$$\beta(G) = \max_i \min\{|B(S)| : |S| = i\}.$$

Setting $t = \beta(G)$ gives the following corollary.

Corollary 4.6. *For any G,*

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

4.3. Separator theorems and upper bounds

The strategy in Theorem 4.4 also leads to a feasible implementation for Step 1 of the divide and conquer algorithm in Section 2.1. This accounts for all of the queries of the algorithm except that Step 3 examines the neighbors of one node in each separating set. Therefore, for graphs where each node is of small degree, upper bounds on $\upsilon(G)$ provide useful upper bounds on $L(G)$. Let $\delta_{\max}(G)$ be the maximum degree of any node in G and let K be the number of separating sets used by player I.

Theorem 4.7. $\upsilon(G) \le L(G) \le \upsilon(G) + \delta_{\max}(G)K$.

Proof. The first inequality comes from Lemma 2.3. The second inequality is a result of player I using optimal separators for the separation game in a divide and conquer algorithm. Step 3 of the divide and conquer algorithm requires the additional inquiry of at most $\delta_{\max}(G)$ nodes for one node in each separating set. □

We now show how we can use separator theorems to determine upper bounds on $\upsilon(G)$. Suppose G belongs to a class of graphs $\mathbf{G}$ closed under the subgraph operation. $\mathbf{G}$ satisfies an $s(n)$-separator theorem with constants α and β, $\alpha \in [\frac{1}{2}, 1)$, $\beta > 0$ if any $G \in \mathbf{G}$ with n vertices can be separated into two sets A and B with $|A|, |B| \le \alpha n$ and the separating set has $\le \beta s(n)$ vertices. A feasible, though not necessarily polynomially computable, strategy for player I is to separate the graph by such a separator, forcing player II to choose a smaller graph to work with. This algorithm gives the following bound:

Corollary 4.8. *If G belongs to a class of graphs* $\mathbf{G}$ *closed under the subgraph operations such that* $\mathbf{G}$ *satisfies an* $s(n)$*-separator theorem with constants* α *and* β*, then*

$$\upsilon(G) \le \beta \sum \{s(\alpha^i n) \colon 0 \le i \le -\log n / \log \alpha\}.$$

Proof. This follows from the definition of player I's algorithm. □

There are several separator theorems in the literature for special types of graphs. Here, we give two examples which illustrate how to use these to derive bounds on the value of the separation game.

Theorem 4.9 [4]. *Any planar graph satisfies a* $\sqrt{n}$*-separator theorem with* $\alpha = \frac{2}{3}$ *and* $\beta = \sqrt{6}$.

Theorem 4.10. *For any planar graph on n vertices, the value of the separation game on G is at most* $13.35\sqrt{n}$.

Corollary 4.11. *A local optimum on a planar graph with n vertices and maximal degree* δ *can be found in*

$$13.35\sqrt{n} + \delta\left(\frac{\log n}{\log 3 - 1}\right)$$

function evaluations.

Proof. This follows from Theorems 4.7 and 4.10 and Corollary 4.8. □

This theorem for planar graphs can be generalized to graphs of fixed genus.

Theorem 4.12 [8]. *Any graph of genus g satisfies a separator theorem with $\alpha = \frac{2}{3}$ and $\beta s(n) = (6\sqrt{g} + 2\sqrt{2})\sqrt{n} + 1$.*

Theorem 4.13. *For any graph G of genus g on n vertices, the value of the separation game on G is at most $(6\sqrt{g} + 2\sqrt{2})(3 + \sqrt{6})\sqrt{n} + O(\log n)$.*

Corollary 4.14. *A local optimum on a graph of genus g with n vertices and maximum degree δ can be found in*

$$(6\sqrt{g} + 2\sqrt{2})(3 + \sqrt{6})\sqrt{n} + O(\log n) + \delta\left(\frac{\log n}{\log 3 - 1}\right)$$

function evaluations.

Proof. This follows from Theorems 4.7 and 4.13 and Corollary 4.8. □

5. The hypercube

We now apply the machinery developed in the previous sections to the problem of finding a local optimum on the vertices of the hypercube. Letting $G_d = (V, E)$ denote the graph of the d-dimensional cube, and f a function $V \to \mathbb{Z}$, how hard is it to find a local optimum of f on G_d?

A straightforward application of our divide and conquer algorithm gives:

Theorem 5.1. *A local optimum on the d-cube can be found with $c(2^d \log d/\sqrt{d})$ function evaluations where $c = \sqrt{2/\pi} + o(d)$.*

Proof. We prove this theorem by illustrating that dc(G) has the given value as an upper bound. Hence by Lemma 2.3, the proof will be complete.

Define the pth shell of the hypercube to be the vertices with exactly p "1"s, so that the pth shell has cardinality $\binom{d}{p}$. Each shell separates the d-cube, so we can narrow down our search to a single shell by binary search on the $d+1$ shells. This binary search takes at most $\log(d+1)+1$ "steps", each step involving the inspection of a shell where no shell has cardinality exceeding $\binom{d}{\lfloor d/2 \rfloor}$. Further, by the extended Stirling approximation [6], the size of the center shell satisfies:

$$C_1\sqrt{\frac{2}{\pi}}\frac{2^d}{\sqrt{d}} \le \binom{d}{\frac{1}{2}d} \le C_2\sqrt{\frac{2}{\pi}}\frac{2^d}{\sqrt{d}},$$

where

$$C_1 = \frac{e^{1/(12d+1)}}{e^{2/6d}}; \qquad C_2 = \frac{e^{1/12d}}{e^{2/(6d+1)}}.$$

Therefore the binary search requires at most

$$\frac{2^d(1+\log_2(d+1))}{\sqrt{d}}\left(\sqrt{\frac{2}{\pi}}+o(d)\right)$$

lookups. There are at most $d\log d$ "additional" queries so they do not affect this bound. Once the location of a local minimum has been narrowed down to a single shell, an additional $\binom{d}{\lfloor d/2\rfloor}$ queries will certainly suffice to find a local optimum. This gives a total of

$$\left(\sqrt{\frac{2}{\pi}}+o(d)\right)\left(\frac{2^d}{\sqrt{d}}(2+\log_2(d+1))\right),$$

proving the theorem. □

We remark that the computation of the bound in this theorem is fairly tight. This might seem surprising since all of the shells are not the same size and hence a more efficient algorithm might, for instance, use a sequence of p's (querying the pth shell) more like $\frac{1}{2}d, \frac{1}{3}d, \dots$ rather than $\frac{1}{2}d, \frac{1}{4}d, \dots$. Further, $\binom{d}{d/4}$ is much smaller than $\binom{d}{d/2}$, and in the theorem all shell sizes are bounded only by $\binom{d}{d/2}$. However, note that when $0\le i\le \frac{1}{2}\sqrt{d}$.

$$\frac{\binom{d}{\lfloor\frac{1}{2}d\rfloor-i}}{\binom{d}{\lfloor\frac{1}{2}d\rfloor}}\ge\frac{\binom{d}{\lfloor\frac{1}{2}d-\frac{1}{2}\sqrt{d}\rfloor}}{\binom{d}{\lfloor\frac{1}{2}d\rfloor}}\ge\frac{\frac{1}{2}d(\frac{1}{2}d-1)(\frac{1}{2}d-2)\dots((\frac{1}{2}d-\frac{1}{2}d)+1)}{\lfloor\frac{1}{2}d+\frac{1}{2}\sqrt{d}\rfloor\lfloor\frac{1}{2}d+\frac{1}{2}\sqrt{d}-1\rfloor\dots(\frac{1}{2}d+1)}$$

$$>\left(\frac{\frac{1}{2}d-\frac{1}{2}\sqrt{d}+1}{\frac{1}{2}d+1}\right)^{\sqrt{d}/2}\ge\left(\frac{1-\frac{1}{2}\sqrt{d}}{\frac{1}{2}(d+2)}\right)^{\sqrt{d}/2}\ge 1-\frac{\sqrt{d}}{d+2}\frac{\sqrt{d}}{2}>\frac{1}{2}.$$

This means that there is a stretch of $\sqrt{d}$ shells each at least half the size of the largest shell. Clearly any shell by shell search just in this region would cost at least

$$\frac{1}{2}\binom{d}{\lfloor\frac{1}{2}d\rfloor}\log\sqrt{d}=\frac{1}{4}\binom{d}{\frac{1}{2}d}\log d.$$

so we are not off by more than a factor of 4.

The hypercube is one of the few families of graphs whose genus has been determined: Beineke and Harary [2] and Ringel [16] show it equals $(d-4)2^{d-2}+1$. By Corollary 4.14, a local optimum can be found in

$$\begin{aligned}&(6\sqrt{(d-4)2^{d-3}+1}+2\sqrt{2})(3+\sqrt{6})2^{d/2}+O(d)+\frac{d^2}{\log 3-1}\\&>(10)2^d\sqrt{d-4}\ (\gg 2^d)\end{aligned}$$

function evaluations. This is much weaker than the bound given by Theorem 5.1, because the $\approx 6\sqrt{d}2^d$ separator from Theorem 4.12 is obviously too large for the d-cube.

We can also combine a result about boundary minimization on the d-cube with our Corollary 4.6 to get a lower bound on $\upsilon(G)$ and hence a lower bound on the number of function evaluations required. Let

$$B(G,m)=\min\{|B(S)|: |S|=m\}.$$

That is, the size of the smallest boundary any set of m vertices can have. A bound on the minimum boundary size $B(G,m)$ is given by [18]:

Let P_i denote $\sum_{j=0}^{i}\binom{d}{j}$. If $m=P_i$, then $B(G_d,m)=\binom{d}{i+1}$. Otherwise let i be such that $P_{i-1}\le m\le P_i$. Then

$$B(G_d,m)\ge\begin{cases}\binom{d}{i} & i\le\frac{1}{2}(d-1),\\ \binom{d}{i+1}, & i\ge\frac{1}{2}(d-1).\end{cases}$$

Applying the above bounds gives $\beta(G_d)=\binom{d}{\lfloor d/2\rfloor}$ where $\beta(\ )$ is as defined in Section 4. Moreover, for m in the range

$$P_{\lfloor (d-1)/2\rfloor -1}\le 2^{d-1}-\binom{d}{\lfloor\frac{1}{2}d\rfloor}=2^{d-1}-\beta(G_d)\le m\le 2^{d-1}$$

we have a lower bound of

$$B(G_d,m)\ge\binom{d}{\lfloor\frac{1}{2}(d-1)\rfloor}\ge 2^d/\sqrt{d}\left(\sqrt{\frac{2}{\pi}}-\mathrm{o}(d)\right).$$

Applying Corollary 4.6, we have proved the following:

Theorem 5.2. *The value of the separation game played on the d-dimensional hypercube is at least* $(\sqrt{2/\pi}-\mathrm{o}(d))2^d/\sqrt{d}$.

Corollary 5.3. *Any algorithm which finds a local minimum on the d-cube requires at least* $2^{d-1}/\sqrt{d}$ *function evaluations.*

In this case the gap between the lower and upper bounds is more than a constant factor, but still quite small: $\mathrm{O}(\log\log|V|)$.

We make some comments comparing this result to others concerning local optimization on the d-cube. The closest result we know of is due to Aldous [1]. He considers a game where player I, a minimizer, selects an algorithm A to find a local minimum on the d-cube, while player II, a maximizer, selects the function f. The outcome of play of the game is the number of function evaluations required by A to find a local minimum of f. Aldous shows that the value of the game is roughly $\mathrm{O}(2^{d/2})$. In these terms, we have analyzed the value of a modified game when player I has to play before player II (player II sees A before choosing f). It is interesting that the modification causes such a large increase in the value of the game.

The problem we have analyzed, when f is computable by a polylog (in n, i.e. polynomial in d) width and depth circuit, is essentially FLIP, the canonical PLS-complete problem of [10]. So if PLS = P, the polynomial algorithm for FLIP must make strong use of limitations on f, since Theorem 5.2 implies there is no polynomial algorithm that works for arbitrary functions f (or even arbitrary functions f with single local optimum). On the other hand, if PLS is not in P, it might be possible to modify the adversarial argument so as to create a function f which was polylog computable, while retaining a superpolynomial bound. Such a proof would imply NP $\neq$ P and would undoubtedly be very difficult! Even a weaker result of the following form would be interesting (this reflects the generally sorry state of lower bound technology): if f is computed by a circuit of fixed width and depth $p(n)$, then the number of function evaluations required to find a local optimum is at least $q(p(n))$, where $q(\ )$ and $p(\ )$ are polynomial functions (q of quadratic or higher order).

Theorem 5.2 yields a snake-in-box result. Define a snake in a graph as a simple path such that any two vertices in the path are adjacent in the graph iff they are adjacent in the path. Thus the snake's coils stay a hamming distance of at least 2 from each other. (A snake-in-box in [3] is defined similarly except that the snake bites its tail, i.e., it is a cycle.) How long can a snake in the d-cube be? Notice that a local improvement algorithm which iteratively selects the *best* adjacent vertex must follow a path which is a snake, since a better adjacent vertex further down the path would previously have been selected. Such an algorithm takes at most $d-1$ function evaluations per iteration (after the first). But Theorem 5.2 says that in the worst case, *any* algorithm must use $2^d/2\sqrt{d}$ function evaluations. Hence, the local improvement algorithm in the worst case takes at least $((2^{d-1}/\sqrt{d})-1)/(d-1)$ iterations, so there is a snake that long (of order $2^d d^{-3/2}$). This is not quite as strong a result as the best known [3] of order $2^d d^{-1/2}$.

We get a surprising observation if we compare the proof of Theorem 5.2 with the analysis of average performance in [18]. Recall that

$$B(G,m)=\min\{|B(S)|: |S|=m\}.$$

The driving force behind the exponentially large lower bound is the large size of $B(G_d,m)$. In particular, the fact that

$$B(G_d,m)\geq \binom{d}{\frac{1}{4}d}, \quad \tfrac{1}{3}2^d\leq m\leq \tfrac{2}{3}2^d$$

is sufficient to imply a large exponential lower bound on worst-case performance of any algorithm. On the other hand, the $O(d\log d)$ and $O(d^2)$ upper bounds on average performance of local improvement algorithms in [18] are derived by proving inequalities such as

$$I(d)\leq 2+\sum_{m=2}^{2^d-1} ek/B(G_d,m),$$

where $I(d) \equiv$ the expected number of iterations of a local improvement algorithm for problems with a single local optimum (for any k regular graph). Hence, here the large magnitude of $B(G_d, m)$ leads to a low order polynomial bound on $I(d)$. Thus we have the peculiar situation that *the same graphical property* of the hypercube, namely the large magnitude of $B(G_d, m)$, appears to be responsible for *both* the exponentially bad worst-case and the polynomially good average-case performance of local improvement algorithms.

6. Remarks

Our analysis shows that local search may not be the most efficient method (in the worst case) to find a local optimum of an arbitrary function on a graph. Instead, a divide and conquer algorithm tends to give better performance. On a path, this algorithm is essentially Fibonacci search. Fibonacci search can be extended to higher dimensions but this does not give the right way to find local optima in arbitrary graphs. Binary search has also been extended to higher dimensions by Dobkin and Lipton [5] and Wood [19]: in [5] the function values (data) are allowed to be rearranged in any desired way: in [19] the generalization is highly geometric. The proper generalization to arbitrary graphs, for our problem, is closer to other divide and conquer algorithms on graphs, such as nested dissection for solving a system of linear equations [12]. Like generalized nested dissection, our algorithm depends heavily on finding a sequence of graph separators; as in [12] Corollary 4.5 shows that no efficient algorithm for finding local optima exists if the graph does not have a good separator.

Theorem 4.3 shows that for general graphs it is NP-hard to find the best set of separators. On the other hand, the analysis of the separation game, though hard, also provides a close lower bound, assuring us of the near-optimality of our algorithm.

It is interesting to see how complicated it can be to design an efficient algorithm that does something very simple, such as finding an entry in a matrix which is smallest in its row and column. The analysis of the d-cube also gave us a new insight into the relationship between the average and worst case behavior of local improvement algorithms.

The exponential lower bound on the work required by any algorithm to find a local optimum on the d-cube suggests that, to find polynomial algorithms for local optimization in combinatorial problems (e.g., integer programming, travelling salesman problem) we need to make heavy use of the restrictions on the objective function implied by the combinatorial problem. Even though it is easy to know which points to check to test for local optimality, and it is easy to know that at least one local optimum must exist, it is hard to find a local optimum for arbitrary f. Our construction employs an f which (we think) cannot be computed by a polynomial (in d) width and depth circuit. If it is not easy to know that at least one local optimum must exist, then finding it can be difficult even with a simple f. For example, if we seek a strict local optimum, the trivial adversarial functions, that are constant everywhere, or constant except at one point, ensure that any algorithm is slow. It can also be hard to know how to test for local optimality: Murty [13] shows that it is NP-hard to test for local optimality in *continuous* quadratic programming, even if function evaluations are assumed to cost unit time.

Acknowledgment

The term "bare tree" was suggested by Kurt Anstreicher.

References

[1] D. Aldous, Minimization algorithms and random walk on the d-cube. Tech. Rept., University of California, Berkeley, CA (1981).

[2] L.W. Beineke and F. Harary, The genus of the n-cube, Canad. J. Math. 17 (1965) 494-496.

[3] L. Danzer and V. Klee, Lengths of snakes in boxes, J. Combinat. Theory 2 (1967) 258-265.

[4] H. Djidjev, On the problem of partitioning planar graphs, SIAM J. Alg. Dis. Meth. (1982) 229-240.

[5] D. Dobkin and R. Lipton, Multidimensional searching problems, SIAM J. Comput. 5 (2) (1976) 181-186.

[6] W. Feller, An Introduction to Probability Theory and Its Applications 1 (Wiley, New York, 1971).

[7] M. Garey and D. Johnson, Computers and Intractibility: A Guide to the Theory of NP-Completeness (Freeman, San Francisco, CA, 1979).

[8] J.R. Gilbert, J.P. Hutchinson and R.E. Tarjan, A separator theorem for graphs of bounded genus, J. Algorithms 5 (3) (1984) 391-407.

[9] D. Hausmann and B. Korte, Lower bounds on the worst-case complexity of some oracle algorithms, Discrete Math. 24 (1978) 261-276.

[10] D. Johnson, C. Papadimitriou and M. Yannakakis, How easy is local search?, in: Proceeding IEEE Symposium on the Foundations of Computer Science (1985) 39-42.

[11] G. Leuker, Unpublished manuscript, Princeton University, Princeton, NJ (1976).

[12] R. Lipton, D. Rose and R. Tarjan, Generalized nested dissection, SIAM J. Numer. Anal. 16 (2) (1979) 346-358.

[13] K.G. Murty and S.N. Kabadi, Some NP-complete problems in quadratic and nonlinear programming, Math. Programming 39 (2) (1987) 117-130.

[14] G. Nemhauser and L. Wolsey, Best algorithms for approximating the maximum of a submodular set function, Math. Oper. Res. 3 (1978) 177-188.

[15] C. Papadimitriou and K. Steiglitz, Combinatorial Optimization: Algorithms and Complexity (Prentice-Hall, Englewood Cliffs, NJ, 1982).

[16] G. Ringel, Das Geschlecht des Vollständigen Paaren Graphen, A.B.H. Math. Sem. Univ. Hamburg 28 (1965) 139-150.

[17] C.A. Tovey, A simplified NP-complete satisfiability problem. Discrete Appl. Math. 8 (1984) 85-89.

[18] C.A. Tovey, Low order polynomial bounds on the expected performance of local improvement algorithms, Math. Programming 35 (2) (1986) 193-224.

[19] G.R. Wood, Multidimensional bisection and global minimization, Tech. Rept., University of Canterbury, 1985.