

A Hamilton Path Heuristic with Applications to the Middle Two Levels Problem

Ian Shields

IBM

P.O. Box 12195

Research Triangle Park, North Carolina 27709, USA

`ishields@us.ibm.com`

Carla D. Savage *

Department of Computer Science

North Carolina State University, Box 8206

Raleigh, North Carolina 27695-8206, USA

`savage@cayley.csc.ncsu.edu`

December 16, 1999

Abstract

The notorious middle two levels problem is to find a Hamilton cycle in the middle two levels, M_{2k+1} , of the Hasse diagram of \mathcal{B}_{2k+1} (the partially ordered set of subsets of a $2k + 1$ -element set ordered by inclusion). Previously, the best known result, due to Moews and Reid [11] in 1990, was that M_{2k+1} is Hamiltonian for all positive k through $k = 11$. We show that if a Hamilton path between two distinguished vertices exists in a reduced graph then a Hamilton cycle can be constructed in the middle two levels. We describe a heuristic for finding Hamilton paths and apply it to the reduced graph to extend the previous best known results. This also improves the best lower bound on the length of a longest cycle in M_{2k+1} for any k .

1 Introduction

Let \mathcal{B}_n be the n -atom Boolean lattice, i.e., the partially ordered set of subsets of $[n] = \{1, 2, \dots, n\}$, ordered by inclusion. The Hasse diagram,

*Research supported in part by NSF grant DMS9622772

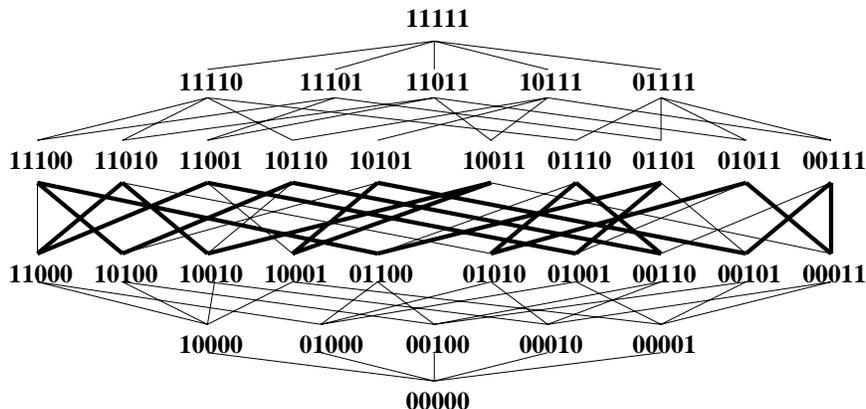


Figure 1: The Hasse diagram of \mathcal{B}_5 showing a Hamilton cycle through the middle two levels.

$H(\mathcal{B}_n)$, is the covering graph of \mathcal{B}_n with two subsets adjacent when they differ in only one element. We define the i th level $\mathcal{B}_n(i)$ of \mathcal{B}_n as the set of i -element subsets of $[n]$, so there are $\binom{n}{i}$ elements in $\mathcal{B}_n(i)$.

The notorious *middle two levels problem* is to determine if there is a Hamilton path or cycle in the subgraph M_{2k+1} of $H(\mathcal{B}_{2k+1})$ induced by the middle two levels $\mathcal{B}_{2k+1}(k)$ and $\mathcal{B}_{2k+1}(k+1)$. The problem has been variously attributed to Dejter, Erdős, Trotter, Havel, and Kelley (see [4], [5], [7], [8], [13]).

The Boolean lattice is isomorphic to the set of n -bit binary numbers with two numbers being adjacent in the Hasse diagram iff they differ in exactly one bit position. For $n = 2k + 1$, the middle two levels are the $2k + 1$ -bit binary numbers whose representations are permutations of $0^k 1^{k+1}$ or $0^{k+1} 1^k$. We illustrate the Hasse diagram of \mathcal{B}_5 in Figure 1, with the heavier lines defining a Hamilton cycle through the middle two levels graph, M_5 .

Since M_{2k+1} is connected and vertex transitive, the nonexistence of a Hamilton path in M_{2k+1} for some k would provide a counterexample to the Lovász conjecture [9] that every connected, undirected, vertex transitive graph has a Hamilton path. However, both constructive and existential approaches have so far been unsuccessful in establishing the existence of a Hamilton path or cycle in M_{2k+1} in the general case. A result of Babai for vertex transitive graphs gives a lower bound of $(3V_k)^{1/2}$ on the length of the longest cycle, where $V_k = 2\binom{2k+1}{k}$ is the total number of vertices in M_{2k+1} . This lower bound was improved to $0.25V_k$ in [6]. The best lower bound is currently given by the following theorem from [14].

Theorem 1 *For any $\epsilon > 0$, there is an $h \geq 1$ so that if M_{2i+1} has a Hamilton cycle for $1 \leq i \leq h$, then the middle levels graph, M_{2k+1} , has a cycle of length at least $(1 - \epsilon)V_k$ for all $k \geq 1$.*

Since it was shown by David Moews and Mike Reid in 1990 that M_{2k+1} is Hamiltonian for $1 \leq k \leq 11$ [11], it follows from Theorem 1 that M_{2k+1} has a cycle of length at least $0.838V_k$.

In this paper, we extend the result of Moews and Reid to show that M_{2k+1} is Hamiltonian for $1 \leq k \leq 15$. Combining this with Theorem 1 slightly improves the lower bound.

Corollary 1 *For every $k > 1$, the middle two levels graph, M_{2k+1} has a cycle of length at least $0.86V_k$.*

Our efforts to find Hamilton cycles in these graphs with up to millions of vertices focus on (1) transforming M_{2k+1} to a *reduced graph* R_{2k+1} , smaller than M_{2k+1} by a factor of $2(2k + 1)$, (2) identifying *distinguished vertices* u and v in R_{2k+1} with the property that any Hamilton path in R_{2k+1} from u to v is *guaranteed* to lift to a Hamilton cycle in M_{2k+1} , and (3) devising a Hamilton path/cycle heuristic which is fast enough and powerful enough to succeed in step (2).

The reduction of step (1) was used by Moews and Reid and part of it, the reduction to *necklaces*, was suggested earlier by Dejter in [4]. Steps (2) and (3) are new.

In Section 2 we describe the reduction and in Section 3 the algorithm. The performance of our implementation of the heuristic on the middle levels graph is discussed in Section 4. In fact, the Hamilton path/cycle heuristic was surprisingly successful. Since it is general enough to apply to any family of graphs, we describe it in enough detail to allow others to try it on graphs of interest.

2 Reducing the Middle Two Levels Problem

2.1 Definitions

An undirected graph G consists of a set of *vertices* $V(G) = \{v_1, v_2, \dots, v_n\}$ and a set of *edges* $E(G) = \{e_1, e_2, \dots, e_m\}$ where each edge is an unordered pair of vertices. We will write uv for the edge $\{u, v\}$ and we say that u and v are the endpoints of this edge. If $uv \in E(G)$ then we say that u and v are *adjacent*. We also write $u \leftrightarrow v$. If there is at most one edge uv for any pair of vertices u and v and there are no edges uu for any vertex u we say

that the graph is a *simple* graph. If an edge uu exists for any vertex u we say that such an edge is a *loop*.

Given a graph G , a list $P = (v_1, v_2, \dots, v_l)$ of vertices of $V(G)$ is a path of length $l - 1$ in G if $v_i \neq v_j$ for all $i \neq j$ and $(v_i, v_{i+1}) \in E(G)$ for $1 \leq i < l$. A path P in G is a *Hamilton path* if $V(P) = V(G)$ where $V(P) = \{v_1, v_2, \dots, v_l\}$. It is a *Hamilton cycle* if it is a Hamilton path and $(v_1, v_l) \in E(G)$.

Given a string X of n symbols, $X = x_1x_2 \dots x_n$, we define a rotation function $\sigma(X)$ by $\sigma(x_1x_2 \dots x_n) = x_2x_3 \dots x_nx_1$. Evidently $\sigma^n(X) = X$ for any n -element string X and indeed $\sigma^i(X) = \sigma^j(X)$ whenever $i \equiv j \pmod{n}$. For convenience, we may think of $\sigma^i(X)$ as rotating to the left when $i > 0$ and rotating to the right when $i < 0$.

For the remainder of this paper we will restrict our attention to strings where the elements are taken from $\{0, 1\}$. We will refer to n -element strings as n -bit binary numbers.

Given a set S of n -bit binary numbers we define a relation \sim on S such that $X \sim Y$ iff $Y = \sigma^i(X)$ for some integer i . The relation \sim is an equivalence relation and we refer to the equivalence classes as *necklaces*. We denote the equivalence class of X by $\nu(X)$. Now suppose $X = \sigma^i(X)$ for some $i < n$. Then it follows easily that if $g = \gcd(i, n)$ then it is also true that $X = \sigma^g(X)$ and X is simply the concatenation of n/g copies of a substring of length g .

Lemma 1 *The necklace equivalence classes for the middle two levels for $n = 2k + 1$ each have n members.*

Proof. Clearly there can be at most n distinct members of each necklace equivalence class. If there are fewer in some class C , then for $X \in C$ there is some $i < n$ for which $X = \sigma^i(X)$. Let $g = \gcd(i, n)$. Then X is composed of n/g copies of some substring Y of length g . Thus n/g is a factor of both k and $k + 1$ so $g = n$ contradicting the assumption of $i < n$. \square

For a binary digit x we define the *complement* \bar{x} as

$$\bar{x} = \begin{cases} 1 & \text{if } x = 0 \\ 0 & \text{if } x = 1 \end{cases}$$

Now suppose $X = x_1x_2 \dots x_n$ is an n -bit binary number. Define \bar{X} in the obvious way as $\bar{X} = \bar{x}_1\bar{x}_2 \dots \bar{x}_n$. We define the complement of a necklace by $\nu(\bar{X}) = \nu(X)$. Note that this is well-defined since $Y = \sigma^i(X)$ iff $\bar{Y} = \sigma^i(\bar{X})$.

2.2 The Reduction

As mentioned in Section 1, for $n = 2k + 1$ we will denote by M_n or M_{2k+1} the graph corresponding to the middle two levels problem where the vertices are the n -bit binary numbers with k or $k + 1$ ones and $u \leftrightarrow v$ if u and v differ in exactly one bit position. Evidently M_n is $k + 1$ -regular and has $2\binom{n}{k} = 2\binom{n}{k+1}$ vertices.

Let us now consider the graph N_n whose vertices are the necklace equivalence classes of the vertices of M_n . If u is a vertex of M_n then $\nu(u)$, the necklace containing u , is a vertex of N_n and $\nu(u)$ is adjacent to $\nu(v)$ iff there is an i such that u differs from $\sigma^i(u)$ in exactly one bit. By Lemma 1 it follows that N_n has $2\binom{n}{k}/n$ vertices which makes it smaller by a factor of n than M_n . (Note, in fact, that the number of vertices of N_{2k+1} is twice the Catalan number $\binom{2k+1}{k}/(2k+1)$.)

We further reduce the problem size by observing that in N_n , $\nu(X) \leftrightarrow \nu(Y)$ iff $\overline{\nu(X)} \leftrightarrow \overline{\nu(Y)}$. We define another equivalence relation \diamond on $V(N_n)$ by $X \diamond Y$ if either $X = Y$ or $X = \overline{Y}$ and denote the equivalence class of \diamond containing X by $\rho(X)$. Since every string in X has odd length, $X \neq \overline{X}$, so every equivalence class $\rho(X)$ has exactly 2 elements. We construct a reduced graph R_n whose vertices are the equivalence classes $\{\rho(X) \mid X \in V(N_n)\}$ with adjacency defined by $\rho(X) \leftrightarrow \rho(Y)$ in R_n if, in N_n , either $X \leftrightarrow Y$ or $X \leftrightarrow \overline{Y}$. Observe that $\overline{\nu(0^k 1^{k+1})} = \nu(1^k 0^{k+1})$ and therefore $\nu(0^k 1^{k+1}) \leftrightarrow \nu(1^k 0^{k+1})$ in N_n so that there is a loop in R_n from $\rho(\nu(0^k 1^{k+1}))$ to itself. More generally, there is a loop in R_n iff $\rho(Y) = \rho(\overline{Y})$ for Y in N_n or equivalently $Y \leftrightarrow \overline{Y}$ in N_n . Thus R_n is usually not a simple graph.

2.3 Lifting paths from the reduced graphs

In Section 2.2 we saw how to reduce the size of the middle two levels problem by constructing smaller graphs from the original. We now show that if a suitably constructed Hamilton path in R_n exists then it can be lifted to construct a Hamilton cycle in N_n and then show how that cycle in N_n can be lifted to construct a Hamilton cycle in M_n .

We observed earlier that R_n is not a simple graph because some edges are loops.

Lemma 2 *If there is a Hamilton path $P_R = (r_1, r_2, \dots, r_l)$ in R_n and the loops $r_1 r_1$ and $r_l r_l$ are edges in R_n then there is a Hamilton cycle in N_n*

Proof. We construct such a cycle P_N . First choose s_1 in N_n such that $r_1 = \rho(s_1)$ and let $P_{N,1} = (s_1)$ be the path in N_n consisting of this single

vertex. Now suppose we have constructed a path $P_{N,k} = (s_1, s_2, \dots, s_k)$ of $k < |N_n|$ vertices in N_n . and suppose $r_{k+1} = \rho(t) = \rho(\bar{t})$ for some t in N_n . Now set

$$s_{k+1} = \begin{cases} t & \text{if } s_k \leftrightarrow t \\ \bar{t} & \text{if } s_k \leftrightarrow \bar{t} \end{cases}$$

Proceeding in this manner we construct the path $P_{N,l} = (s_1, s_2, \dots, s_l)$ of l vertices, where $r_l = \rho(s_l)$. The desired cycle is then

$$(s_1, s_2, \dots, s_l, \overline{s_l}, \overline{s_{l-1}}, \dots, \overline{s_1})$$

which is a cycle because $\rho(v) \leftrightarrow \rho(v)$ in R_n iff $v \leftrightarrow \bar{v}$ in N_n and, by assumption, $r_1 \leftrightarrow r_1$ and $r_l \leftrightarrow r_l$ in R_n . \square

For the special case where $r_1 = \rho(\nu(0^k 1^{k+1}))$ and $r_l = \rho(\nu(0(01)^k))$ we now show that we can lift the Hamilton cycle in the necklace graph to a Hamilton cycle in the full middle two levels graph M_n ,

Lemma 3 *If there is a Hamilton path $P_R = (r_1, r_2, \dots, r_l)$ in R_n and $r_1 = \rho(\nu(0^k 1^{k+1}))$ and $r_l = \rho(\nu(0(01)^k))$ then there is a Hamilton cycle in M_n*

Proof.

We first choose a sequence of vertices (Y_1, Y_2, \dots, Y_l) in N_n , as representatives in N_n of the equivalence classes (r_1, r_2, \dots, r_l) in R_n . Now choose a sequence of vertices (X_1, X_2, \dots, X_l) in M_n , as representatives in M_n of l of the equivalence classes $(N_1, N_2, \dots, N_{2l})$ in N_n . By the construction of N_n and R_n and the assumption of the lemma these vertices may be chosen subject to the following constraints.

1. $r_i = \rho(Y_i) = \rho(\overline{Y_i})$
2. $Y_i = \nu(X_i)$
3. $X_1 = 0^k 1^{k+1}$.
4. X_i is a permutation of $0^k 1^{k+1}$.
5. $X_l = \sigma^j(0(01)^k)$ for some $j < n$.
6. $X_i \leftrightarrow \overline{X_{i+1}}$ in M_n for $1 \leq i < l$

Suppose first that l is odd. By Lemma 2 we may construct a Hamilton cycle

$$\nu(X_1), \nu(\overline{X_2}), \nu(X_3), \dots, \nu(X_l), \nu(\overline{X_l}), \dots, \nu(\overline{X_3}), \nu(X_2), \nu(\overline{X_1})$$

in N_n . Now X_l was chosen so that $X_l = \sigma^j(0(01)^k)$ for some $j < n$ and so we have $\sigma(\overline{X_l}) = \sigma(\overline{\sigma^j(0(01)^k)}) = \sigma(\sigma^j(\overline{0(01)^k})) = \sigma^{j+1}(\overline{0(01)^k}) = \sigma^{j+1}(1(10)^k) = \sigma^j(1(01)^k)$. This differs in one bit from $\sigma^j(0(01)^k) = X_l$ so that $\sigma(\overline{X_l}) \leftrightarrow X_l$ in M_n . We may thus construct a path

$$q = X_1, \overline{X_2}, X_3, \dots, X_l, \sigma(\overline{X_l}), \sigma(X_{l-1}), \dots, \sigma(\overline{X_3}), \sigma(X_2), \sigma(\overline{X_1})$$

of length $2l$ in M_n . Now

$$\sigma(\overline{X_1}) = \sigma(\overline{0^k 1^{k+1}}) = \sigma(1^k 0^{k+1}) = \sigma(\sigma^k(0^k 1^k 0)) = \sigma^{k+1}(0^k 1^k 0)$$

which differs in exactly one bit from $\sigma^{k+1}(0^k 1^k 1) = \sigma^{k+1}(X_1)$. Thus $\sigma(\overline{X_1}) \leftrightarrow \sigma^{k+1}(X_1)$ in M_n . Now $k+1$ and n are relatively prime so we may extend our partial path q in M_n to a Hamilton path:

$$q, \sigma^{k+1}(q), \sigma^{2(k+1)}(q), \dots, \sigma^{(n-1)(k+1)}(q),$$

that is:

$$\begin{aligned} & X_1, \dots, \sigma(\overline{X_1}), \\ & \sigma^{k+1}(X_1), \dots, \sigma^{k+2}(\overline{X_1}), \\ & \vdots \\ & \sigma^{(n-1)(k+1)}(X_1), \dots, \sigma^{(n-1)(k+1)+1}(\overline{X_1}). \end{aligned}$$

Since $\sigma(\overline{X_1}) \leftrightarrow \sigma^{k+1}(X_1)$ we have that

$$\sigma^{(n-1)(k+1)+1}(\overline{X_1}) \leftrightarrow \sigma^{(n-1)(k+1)+k+1}(X_1) = X_1$$

and our Hamilton path is a Hamilton cycle.

If l is even we use Lemma 2 to construct a Hamilton cycle

$$\nu(X_1), \nu(\overline{X_2}), \nu(X_3), \dots, \nu(\overline{X_l}), \nu(X_l), \dots, \nu(\overline{X_3}), \nu(X_2), \nu(\overline{X_1})$$

in N_n .

Since $X_l = \sigma^j(0(01)^k)$ for some $j < n$ we have $\overline{X_l} = \overline{\sigma^j(0(01)^k)} = \sigma^j(\overline{0(01)^k}) = \sigma^j(1(10)^k) = \sigma^{j+1}(011(01)^{k-1})$. This differs in one bit from $\sigma(\sigma^j(0(01)^k)) = \sigma(X_l)$ so that $\overline{X_l} \leftrightarrow \sigma(X_l)$ in M_n and we may construct a path

$$q' = X_1, \overline{X_2}, X_3, \dots, \overline{X_l}, \sigma(X_l), \sigma(\overline{X_{l-1}}), \dots, \sigma(\overline{X_3}), \sigma(X_2), \sigma(\overline{X_1})$$

As in the odd case, we may extend our path q' in M_n to a Hamilton cycle in M_n :

$$q', \sigma^{k+1}(q'), \sigma^{2(k+1)}(q'), \dots, \sigma^{(n-1)(k+1)}(q'). \square$$

3 The Hamilton Path Heuristic

Given a graph G and vertices $s, t \in V(G)$ we would like to find, if possible, a Hamilton path starting at s and ending at t . The heuristic we propose is a refinement of the path reversal technique of Pósa [10] that is the basis of several Hamilton path algorithms for random graphs. The heuristic delays performing a reversal until it finds a sequence of one or more reversals guaranteed to result in a path which can be further extended. If no such sequence exists, the algorithm terminates. The worst-case running time is polynomial in the number of vertices.

A standard backtrack search (see for example [12]) attempts to construct such a path by starting at s and extending the path to a new vertex as long as possible (avoiding t until all other vertices have been included). Whenever it is impossible to further extend the path from a vertex x , the search “backs up” to the predecessor, y , of x on the path and the path is extended (if possible) from y to one of its other neighbors.

A backtrack search is guaranteed to find a Hamilton path from s to t if such a path exists. The drawback is that it is an exhaustive search, exponential time in the worst case, and it quickly becomes too slow to be effective for anything but relatively small problems. Pósa [10] observed that reaching a dead end in backtrack search could be used as an opportunity to modify the current path by a *reversal* and thus possibly to continue. Figure 2 illustrates a path P from a starting vertex s to a vertex u whose only neighbors in G are v and x , which are already on P . In this case, backtrack search would back up to y .

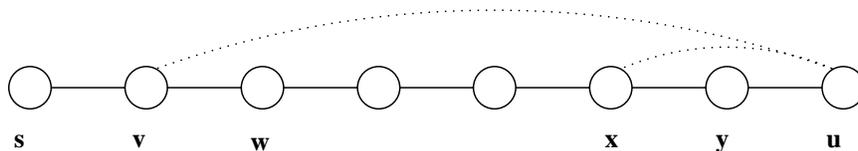


Figure 2: Backtrack search from s has reached u and cannot continue.

Instead, we can transform the current path P into a path of equal length by a *reversal at v*, which inserts the edge uv into P and removes the existing edge vw from P as illustrated in Figure 3. Vertex w becomes the new endpoint of P . More generally, if $P = (u_1, u_2, \dots, u_k)$ is a path in G and there is an edge $u_k u_j$ for some $0 < j < k$, then the *reversal of p at u_j* , namely $P' = (u_1, u_2, \dots, u_j, u_k, u_{k-1}, \dots, u_{j+1})$, is also a path in G . If $j \neq k - 1$ then P' and P share a common starting point but have different endpoints.

Define $\text{PosaSearch}(G, s, t)$ to be the Hamilton cycle search heuristic

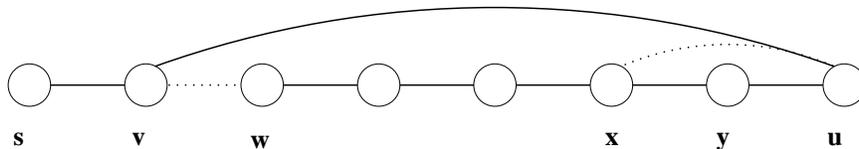


Figure 3: A path reversal at v : add edge uv and remove edge vw .

which constructs a path P by starting at s and extending the path until no longer possible (avoiding t until the end). When the path cannot be extended from a vertex u , the procedure selects a neighbor v of u and performs a path reversal at v by calling $\text{ReversePath}(v, P)$. PosaSearch does not backtrack. See Figure 4 for the pseudo-code,

Of course, there are drawbacks to the PosaSearch . It may not succeed in finding a path even if one exists. Furthermore, it may run indefinitely, even if we eliminate the obvious case where the reversal candidate chosen is the immediate predecessor of the endpoint on the path. Such an example is shown in Figure 5 where the algorithm will alternate between reversing at v a path with endpoint w and reversing at v a path with endpoint u . Even if the edge uv were not present, using PosaSearch , the path shown would never be extended to a Hamilton path as it can never include the vertex x .

Nevertheless, the path reversal approach has proven useful in studying random graphs. Angluin and Valiant [1] devised a fast variant that was always terminating by removing edges from the graph once they had been included in a path. This was shown to almost surely find a Hamilton cycle in random graphs with n vertices and $cn \log n$ edges. The middle two levels graph, although certainly not random, has a relationship of vertices to edges of approximately this form and we did try the algorithm on it, but were never successful in finding a Hamilton cycle.

To overcome the potential non-termination of the Pósa search and to increase the chances that a path reversal would lead to an extension of the path, we improved the way in which candidates were chosen for the reversal operation whenever all neighbors of the endpoint are already on the path.

Reversals transform a path and alter the order of vertices on it. We therefore define several functions to handle arbitrary paths. Let $P = (u_1, u_2, \dots, u_n)$ be an arbitrary path of length $n - 1$ in a graph G . We denote the last vertex u_n as $\text{end}(P)$. We denote the i th vertex u_i by $P(i)$ and the position of u_i as $\text{pos}(P, u_i) = i$. We denote the number of vertices on P as $|P|$ and thus $\text{end}(P) = P(|P|)$. The successor of a vertex u on P is defined for all vertices other than $\text{end}(P)$ as $\text{succ}(P, u) = P(1 + \text{pos}(P, u))$. We may then rewrite a reversal of P at u_j , denoted

```

PosaSearch(G,s,t)
  P ← (s)
  while (|P| < |V(G)| - 1)
    do u ← end(P)
      ▷ Extend P from u if possible
      extension_candidates ← neighbors of u, other than t, not on P
      if extension_candidates not empty
        then select x from extension_candidates
             P ← Px    ▷ Add x to the end of P
      else ▷ Do a reversal, if possible
            reversal_candidates ← neighbors of u on P
            if reversal_candidates not empty
              then select v from reversal_candidates
                   P ← ReversePath(v, P)
            else return FAIL
    ▷ Now P is missing only t so perform reversals until t is a neighbor
    ▷ of end(P)
    while t is not a neighbor of end(P) do
      if end(P) has no neighbor
        then return FAIL
      else ▷ all neighbors are on P
            select a neighbor v of end(P)
            P ← ReversePath(v, P)
    P ← Pt    ▷ Add t to the end of P
  return P

```

Figure 4: The Posa path reversal heuristic

$r(P, j)$, as $(P(1), P(2), \dots, P(j), P(n), \dots, P(j+1))$. Evidently $r(P, j)$ is a path iff $end(P) \leftrightarrow P(j)$. Let $P' = r(P, j)$ be a reversal of P at $P(j)$ and let u be a vertex on P . Let $p = pos(P, u)$ be the position of u on P . Then the position p' of u on P' is

$$p' = pos(P', u) = \begin{cases} p & \text{if } p \leq j \\ |P| - p + j + 1 & \text{otherwise} \end{cases} \quad (1)$$

Rewriting Equation 1 we may determine the vertex of P that is now in position p' on P' as

$$P'(p') = \begin{cases} P(p') & \text{if } p' \leq j \\ P(|P| - p' + j + 1) & \text{otherwise} \end{cases} \quad (2)$$

The successor of $u = P(p')$ on P' is defined iff $p' < |P|$ and is obtained by substituting $p' + 1$ for p' in Equation 2.

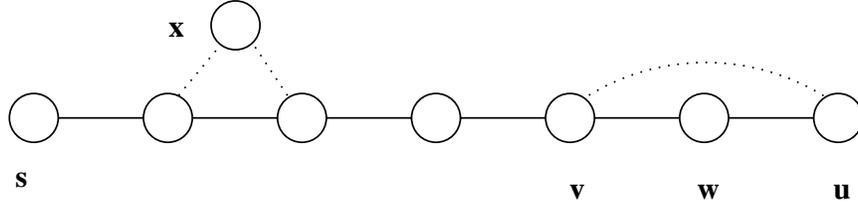


Figure 5: Problems with the PosaSearch heuristic

If P is a path with endpoint $u = \text{end}(P)$ and v is a vertex on P adjacent to u we may reverse P at v to arrive at a new path $P' = r(P, \text{pos}(P, v))$. We use Equation 1 to find the position of v on P' and then use Equation 2 to determine the successor w on P' of v . If the edge uv is on P then $P = P'$, otherwise we say that P' is *derivable* from P in one reversal and we say that the new endpoint w of P' is *reachable in one reversal of P* . Inductively, we say that a path Q is derivable from P in $n > 1$ reversals if it is derivable in one reversal from a path R which is derivable from P in $n - 1$ reversals. We define a vertex of P as *reachable in $n > 1$ reversals* if it is reachable in one reversal from a path R which is derivable from P in $n - 1$ reversals.

Our approach is to extend the path as is done in PosaSearch until all neighbors of the endpoint are already on the path and we cannot extend further. Whenever we reach such a point we start building a tree of the endpoints that are reachable from the current path P . If u is in the tree and if y is the parent of u , then u and y are endpoints of paths P_u and P_y reachable from P and, furthermore, $P_u = r(P_y, j)$ for some index j . We save this value j as $J[u]$ and save the parent of u as $\pi[u]$. The root of the tree is $\text{end}(P)$. $\pi[\text{end}(P)]$ is set to NIL and $J[\text{end}(P)]$ is set to 0. The sequence of values $J[\pi[u]], J[\pi[\pi[u]]], \dots, J[\text{end}(P)]$ along the path in the tree from u to the root define the sequence of reversals required to derive P_u from P .

The search proceeds using a standard breadth-first search (BFS). Initially, all vertices on P are marked unvisited. When a reachable endpoint is first discovered, it is placed on a first-in first-out queue, Q . To get the search started, the endpoint of P (the root of the tree) is marked visited and put on Q . The following is repeated as long as Q is not empty: remove the front vertex, u , from Q . Note that because u is on Q , $u = \text{end}(R)$ for some path R derivable from P . When u is dequeued each neighbor v of u in G is examined. If v is on P we use the stored π and J values to determine the endpoint w that would result from a reversal of R at v . If w has not yet been visited then w is reachable from P and we add it to the tree, set its π and J values, mark it visited, and insert it into Q . If v is not on P our

search has succeeded. We terminate the search process and perform the sequence of reversals required to derive R with endpoint u from P . Again, the stored π and J values will be sufficient to find R . Since $u \leftrightarrow v$ in G and v is not on P we are now guaranteed to be able to extend the path R by at least one edge. If the BFS search fails to find a vertex having a neighbor not on P then the algorithm terminates without having found a Hamilton path.

The pseudo-code for `BFSExtend` is shown in Figure 6, using a style similar to the BFS algorithm in Cormen, Leiserson and Rivest [3].

```

BFSExtend( $P, s, t$ )
  for each vertex  $u$  on  $P$ 
    do  $visited[u] \leftarrow false$ 
   $z \leftarrow end(P)$ 
  Enqueue( $Q, z$ )
   $visited[z] \leftarrow true$ 
   $J[z] \leftarrow 0$ 
   $\pi[z] \leftarrow NIL$ 
  while  $Q$  is not empty
    do  $u = Head(Q)$ 
      Dequeue( $Q$ )
      for each vertex  $v$  in  $Adj(u)$ 
        do if  $v \neq t$ 
          then if  $v \in P$ 
            then  $p \leftarrow FindPosition(P, u, Pos[v])$ 
               $w \leftarrow FindVertex(P, u, p + 1)$ 
              if  $\neg visited[w]$  and  $w \neq t$ 
                then  $J[w] \leftarrow p$ 
                   $\pi[w] \leftarrow u$ 
                   $visited[w] \leftarrow true$ 
                  Enqueue( $Q, w$ )
              else  $P \leftarrow DoReversals(P, u)$ 
            return success
  return fail

```

Figure 6: The Hamilton path BFS heuristic

We do not assume any particular representation of the graph, only that a function $Adj(u)$ is available to return a list of vertices adjacent to u . We maintain several data structures for each vertex in the tree. For each vertex u a Boolean variable $visited[u]$ indicates whether u has been visited or not. The ancestor of u in the tree is stored in $\pi[u]$ and the position of the vertex about which a reversal is done to make u the new endpoint

of a path currently ending in $\pi[u]$ is stored in $J[u]$. A first-in, first-out queue Q , initially empty, is used to manage the list of vertices that have been discovered but whose neighbors have not yet been examined. We also assume that the current position of a vertex u on the path P is available as $Pos[u]$.

The pseudo-code, `FindPosition`, implementing the algorithm of Equation 1 to find the position of the vertex u on a path R derivable from path P such that u is currently in position p on path P and $u = \text{end}(R)$ is shown in Figure 7.

```

FindPosition ( $P, u, p$ )
  if  $J[u] \geq 0$ 
    then  $p \leftarrow \text{FindPosition}(P, \pi[u], p)$ 
    if  $J[u] < p$ 
      then  $p \leftarrow |P| - p + J[u] + 1$ 
  return  $p$ 

```

Figure 7: Finding the new position of a vertex

The pseudo-code, `FindVertex`, implementing the algorithm of Equation 2 to find the vertex v that will be in position r on the path R derivable from P such that $u = \text{end}(R)$ is shown in Figure 8.

```

FindVertex ( $P, u, r$ )
  while  $J[u] \geq 0$ 
    if  $J[u] < r$ 
      then  $r \leftarrow (|P| - r) + J[u] + 1$ 
     $u \leftarrow \pi[u]$ 
  return  $P[r]$ 

```

Figure 8: Finding the new vertex in a position

The function `DoReversals` shown in Figure 9 performs the reversals once a suitable new endpoint has been found.

Our algorithm can be made non-deterministic by allowing choices from the adjacency list to be made randomly. This allows additional attempts to find a Hamilton path to be made with a new random number seed if the algorithm fails on a particular attempt.

Running Time

If V and E are the number of vertices and edges respectively in G and H is the maximum height of the BFS tree then we may determine the worst-

```

DoReversals( $P, u$ )
  if  $J[u] \geq 0$ 
    then  $DoReversals(P, \pi[u])$ 
       $l \leftarrow J[u] + 1$ 
       $h = |P|$ 
      while  $l < h$ 
         $t = P[h]$ 
         $Pos[P[h]] \leftarrow l$ 
         $Pos[P[l]] \leftarrow h$ 
         $P[h] \leftarrow P[l]$ 
         $P[l] \leftarrow t$ 
         $h \leftarrow h - 1$ 
         $l \leftarrow l + 1$ 

```

Figure 9: Performing the reversals

case running time as follows. The running time for each of FindPosition and FindVertex is $O(H)$. BFSExtend performs some initial housekeeping for each vertex on P and then examines each edge in G that is incident with a reachable vertex of P and computes a reachable endpoint for each such edge, so the time spent building the (partial) BFS tree is at worst $O(V) + O(E)O(H)$. The path reversals performed in BFSExtend are $O(H)O(V)$ at worst, so the running time of BFSExtend is at worst $O(E)O(H)$. The main program performs some initial housekeeping for each vertex in G and then examines each edge in G until a path is found or the algorithm terminates. Since a call to BFSExtend either results in termination of the algorithm or extension of the current path, the total running time for the algorithm is at worst $O(V)O(E)O(H)$. In an extreme case this is $O(V^4)$. However, if the number of edges were, say $O(V \log V)$ and the maximum height of the BFS tree was $O(\log V)$ the worst case would become $O(V^2 \log^2 V)$, a significant improvement.

For comparison, the running time of the deterministic algorithm of Bollobás, Fenner and Frieze [2] for finding paths almost surely in random graphs is $O(V^{4+\epsilon})$.

4 The Performance of the Heuristic on the Middle Two Levels Graphs

Our BFS heuristic was applied to search the reduced graphs R_{2k+1} , $k = 1, 2, \dots, 15$, to try to find a Hamilton path from vertex $\rho(\nu(0^{k+1}1^k))$ to vertex

$\rho(\nu(0(01)^k))$. A Hamilton path meeting these requirements was found for each $k \leq 15$. The case $k = 15$, which has about 9.6M vertices in the reduced graph and 155M vertices in the full graph, took almost 3 weeks on 400MHz P-II with 192MB RAM. The timing results are summarized in Table 1. For the middle two levels graph, the number of edges, E is $O(V \log V)$ where V is the number of vertices. For $k = 15$ the maximum height of the BFS tree was ≤ 6 for all searches except possibly the final one to add t for which our instrumentation did not check the tree height. This is much better than the theoretical worst case of $O(V)$. If the maximum height were bounded by $O(\log V)$ then the worst case running time on this class of graph would be $O(V^2 \log^2 V)$ rather than the general worst case of $O(V^4)$. In contrast to the small height actually seen, the largest BFS tree that was generated had over 6M of the 9.6M graph vertices enqueued at once.

k	$n = 2k + 1$	# vertices in R_n	# vertices in M_n	Time (Secs)
8	17	1,430	48,620	0
9	19	4,862	184,756	1
10	21	16,796	705,432	2
11	23	58,786	2704,156	18
12	25	208,012	10,400,600	235
13	27	742,900	40,116,600	2,941
14	29	2,674,440	155,117,520	36,859 (10 hrs)
15	31	9,694,845	601,080,390	1,756,134 (3 weeks)

Table 1: Running time for the heuristic to find a Hamilton path in R_n (and thus a Hamilton cycle in the middle two levels of B_n .)

One would expect that on any given run, the algorithm would eventually get stuck and fail to find a Hamilton path; the search for a Hamilton path would then be re-started, from the same vertex in our case, but making different random choices of neighbors. The hope would be that if a Hamilton path exists, it would be found without too many re-starts.

It was striking, however, that in our trials on the reduced middle two levels graph, the algorithm always found a Hamilton path on the first run. It would be interesting to experiment further on various classes of graphs to see if this success is typical.

For comparison, consider a less drastic modification of the PosaSearch algorithm which we tried when we first started using it to find a path from a starting vertex s to an ending vertex t . We eliminated the possibility of a trivial reversal but we did not further restrict the algorithm by excluding the desired termination vertex.

Instrumenting this modified algorithm showed that the bulk of our time was spent in processing the final few vertices and then performing reversals to make t the actual endpoint. We made two simple enhancements. The first was to exclude t as a candidate for extension until it was the only vertex not on the path. The second was to choose reversals by giving preference to a reversal in which the new endpoint had a neighbor not already on the path. These two changes alone reduced the running time for $k = 12$ by a factor of four. Furthermore, the second enhancement was the nucleus of what would eventually become the BFS search heuristic.

Even with these enhancements the PosaSearch algorithm slowed considerably as larger graphs were searched. Initially we were using a Pentium Pro system with only 128MB of RAM and the PosaSearch algorithm ran for about 8 days on the $k = 14$ case and had still not finished. The first version of the heuristic found a path for $k = 14$ in only four days.

As k grows, the number of vertices, V in M_n grows exponentially. The memory used for the data structures in our implementation is linear in V so our memory requirement grows exponentially, along with V . For $k = 15$ even the reduced graph has almost 10 million vertices. We stored most items as 32-bit integers so that the original vertex list, the path itself and the data structures we used in the BFS search each required about 40MB of storage. For smaller values we were able to generate adjacency lists once and store them in a large array. For larger values we computed adjacency lists each time they were required as this proved to be faster than allowing the operating system to swap our large arrays. Even so, swapping on the $k = 15$ case caused significant slowdown for the last two of the three weeks of the run. Extending the algorithm to handle cases larger than $k = 15$ would mean processing bit strings of at least 33 bits and would probably be easier to do on a 64-bit machine with much greater memory than the 192MB on the 32-bit system that we used.

Our program printed the path in order as output. Each output line included vertex number, the bit representation of the vertex and the complement of that representation. Even this relatively small amount of information resulted in an output file of around 800MB for $k = 15$. Obviously, such large output cannot reasonably be verified manually, so we wrote a separate verification program not reusing any of the logic from the original program to process the output file and verify that the claimed path was indeed a proper Hamilton path.

Late note: We have recently discovered that the idea of building a tree during the extension process was used by Shamir [15], albeit in a somewhat different fashion, for establishing a tight threshold for Hamiltonicity of random graphs.

References

- [1] D. Angluin and L. G. Valiant. Fast probabilistic algorithms for Hamiltonian circuits and matchings. *J. Comput. System Sci.*, 18(2):155–193, 1979.
- [2] B. Bollobás, T. I. Fenner, and A. M. Frieze. An algorithm for finding Hamilton paths and cycles in random graphs. *Combinatorica*, 7(4):327–341, 1987.
- [3] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.
- [4] I. J. Dejter. Hamilton cycles and quotients of bipartite graphs. In *Graph theory with applications to algorithms and computer science (Kalamazoo, Mich., 1984)*, pages 189–199. Wiley, New York, 1985.
- [5] Dwight Duffus, Bill Sands, and Robert Woodrow. Lexicographic matchings cannot form Hamiltonian cycles. *Order*, 5(2):149–161, 1988.
- [6] S. Felsner and W. T. Trotter. Colorings of diagrams of interval orders and α -sequences. *Discrete Math.*, 144:23–31, 1995.
- [7] Glenn Hurlbert. The antipodal layers problem. *Discrete Math.*, 128(1-3):237–245, 1994.
- [8] H. A. Kierstead and W. T. Trotter. Explicit matchings in the middle levels of the Boolean lattice. *Order*, 5(2):163–171, 1988.
- [9] L. Lovász. Problem 11. In *Combinatorial Structures and their Applications*. Gordon and Breach, 1970.
- [10] L. Pósa. Hamiltonian circuits in random graphs. *Discrete Math.*, 14(4):359–364, 1976.
- [11] Mike Reid. E-mail to J Gallian. Correspondence describing work performed by David Moews and Mike Reid on middle levels problem, 1990.
- [12] Edward M. Reingold, Jurg Nievergelt, and Narsingh Deo. *Combinatorial Algorithms: Theory and Practice*. Prentice–Hall, Englewood Cliffs, New Jersey, 1977.
- [13] Carla D. Savage. Long cycles in the middle two levels of the Boolean lattice. *Ars Combin.*, 35(A):97–108, 1993.
- [14] Carla D. Savage and Peter Winkler. Monotone Gray codes and the middle levels problem. *J. Combin. Theory Ser. A*, 70(2):230–248, 1995.

- [15] Eli Shamir. How many random edges make a graph Hamiltonian?
Combinatorica, 3(1):123–131, 1983.