Secluded Path via Shortest Path

Matthew P. Johnson[1], Ou Liu[2], and George Rabanca[2]

[1] Department of Math and Computer Science, Lehman College, CUNY
[2] PhD Program in Computer Science, The Graduate Center, CUNY

Abstract. We provide several new algorithmic results for the secluded path problem, specifically approximation and optimality results for the static algorithm of [3, 5], and an extension (h-Memory) of it based on de Bruijn graphs, when applied to bounded degree graphs and some other special graph classes which can model wireless communication and line-of-sight settings. Our primary result is that h-Memory is a PTAS for degree-Δ unweighted, undirected graphs, providing a $\left\lceil \sqrt{\frac{\Delta+1}{h+1}} \right\rceil$-approximation in time $O(h^h n \log n)$; in particular, 0-Memory (i.e., static) provides a $\sqrt{\Delta+1}$-approximation (i.e., $\epsilon = \sqrt{\Delta+1} - 1$), tightening the previous analysis of this algorithm, and Δ-Memory is optimal (i.e., $\epsilon = 0$), and is faster than the known optimal algorithm for this setting [3].
We also show that 0-Memory and 1-Memory give constant approximations for unit-disk graphs and planar graphs, and that an extension of h-Memory solves many other tessellation graphs. Finally, we prove that the problem is NP-hard on node-weighted graphs of degree 3.

1 Introduction

Let the *neighborhood* $N[P]$ *of a path* P be the set of all nodes within distance at most 1 from some node on the path. Equivalently (for paths of length greater than 1), this is the union of the neighborhoods of all the nodes on the path. The *secluded path problem* be defined as follows: given a graph on n nodes and two specified nodes s and t, find a path from s to t of minimum size neighborhood.

Earlier work showed the problem is very hard to approximate in general and gave some approximation results where the factor typically was not constant but depended on parameters of the graph. Recently, an optimal dynamic programming algorithm [3] was given for the (unweighted, undirected) setting where the degree is bounded by a constant Δ, with a running time of $O(n^2)$. The DP sub-instances represent being at a node and implicitly "remembering" the Δ previous nodes you have visited. If Δ is not treated as a constant, however, then the $O(\Delta^\Delta)$ previous node sequences yield a running time of $O(\Delta^\Delta \cdot n^2)$.

Consider the situation where Δ is fixed, but is large enough that the $O(\Delta^\Delta \cdot n^2)$ running time is prohibitive. If the Δ^Δ factor comes from "remembering" Δ previous steps, then a natural question is whether remembering a smaller number of steps would give us some approximation guarantee. That is, despite the fixed-Δ problem being polynomial solvable, because of the sort of polynomial it is, there may nonetheless be value in a PTAS. In this paper we provide such

a PTAS, providing the usual tradeoff between approximation guarantee and running time, except in this case providing a $(1+\epsilon)$-approximation for $\epsilon \geq 0$ rather than $\epsilon > 0$.

Our algorithm relates to de Bruijn graphs [4], specifically constructing de Bruijn subgraphs based on the input graph G and on a memory parameter h. Each node in the constructed graph G_h corresponds to a length-h (or, for nodes closed to the source, length$\leq h$) walk within G. That is, these fixed-length walks in G are interpreted as "k-mers" on symbols V generating an h-dimensional de Bruijn subgraph.

Intuitively, many of our arguments in this paper take the following form. We wish to optimize for some function $cost(\cdot)$, but this is hard. Instead we optimize for some alternative function $cost'(\cdot)$, which is easy. Although $cost'(\cdot)$ may differ from $cost(\cdot)$ on many inputs (i.e., it may be inaccurate), it will match $cost(\cdot)$ on optimal solutions, or nearly matches it on optimal solutions, or nearly match it on *some* optimal solutions, and so by optimally solving according to $cost'(\cdot)$ we will find a good solution under $cost(\cdot)$.

We also apply this algorithm (and extensions) to certain special graph classes that correspond to two application settings. The first is wireless communication, where nodes of the graph correspond to unit disks located in the plane and two nodes share an edge if their disks intersect. For problem instances in which the nodes may occur at arbitrary locations in the plane the corresponding graph class is that of *unit disk graphs*. Alternatively, nodes may be placed in a regular arrangement, corresponding to a *grid graph*. A natural family of such graphs is yielded by considering tilings or tessellations of the plane, which partition the area of the plane into (regular) polygons. In this case, each tile can be interpreted as the unit disk kissing the tile's corners, in which case tiles share and edge iff the corresponding disks overlap. (This graph is the dual graph of the tessellation interpreted as a planar graph drawing.) Two fundamental types of tessellations are a) *regular* or *Platonic* and b) *semiregular* of *Archimedean* (see [7, 2]). There are three Platonic tessellations: those whose tiles are triangles, squares, and hexagons. A relaxation of the technical geometric definition of such tilings yields the 8 Archimedean tilings, each of which consist of two types of tiles. The resulting disk graph will then have disks of two sizes.

A second application setting is the movement through city streets. Suppose we can travel along streets from intersection to intersection, there is an *observer* at each intersection, and that when we are at an intersection we are *visible to* the observers at the next intersection for each adjoining street. That is visibility is determined by line of sight between intersections, where the city blocks are obstacles and when there are multiple collinear intersections, each is visible only to its neighboring intersections. This setting is modeled by planar graphs, where intersections become nodes, streets become edges, and city blocks become faces. Note that the (non-dual) graph of a tessellation is a special case of planar graphs that can be interpreted in this way.

Related work. The problem considered in this paper, recently introduced by [3], is a variation of the classical shortest path. In the standard version of this

problem, a cost measure is associated with edges, e.g., representing length and the task is to identify a shortest path form s to t. The cost of the path is a *linear* sum of its constituent edges.

In this problem, the cost is on the nodes "seeing" the message rather than on the edges. Selecting an edge (u, v) means that u will transmit to v, but also to its entire neighborhood. The deeper difference, therefore, is that we pay not just for the nodes constituent to the path itself but for neighboring nodes as well, which results in the total cost being a nonlinear function of the chosen nodes' transmission costs.

Chechik et al. [3] gave a number of negative and positive results, including the following. They showed, by reduction from Red-Blue Set Cover [1, 13], the problem is strongly inapproximable on unweighted undirected graphs with unbounded degree (more specifically, is hard to approximate with ratio $O(2^{\log^{1-\epsilon} n})$, where n is the number of nodes in the graph G, assuming $\mathbf{NP} \not\subseteq \mathbf{DTIME}\left(n^{\text{poly}\log n}\right)$). Conversely, they showed that the static algorithm gives a $(\sqrt{\Delta}+3)$-approximation.

They showed that the problem is NP-hard already on node-weighted graphs of degree 4 graphs and directed graphs of degree 3;[3] for the unweighted undirected setting with any constant maximum degree Δ, they gave a polynomial-time (albeit exponential in Δ) optimal dynamic programming algorithm. They also gave a result implying a 6-approximation for planar graphs as a special case, and noted without proof that a 3-approximation can be obtained. (We provide a proof here for completeness.)

A variant of the problem was also recently introduced as the Thinnest Path Problem [6], which involves hypergraphs and subsumes the secluded path problem. They gave a $\sqrt{\frac{n}{2}}$-approximation result for the static algorithm (see Sec. 2) applied to the hypergraph setting and a 19-approximation analysis of this algorithm for the case of unit-disk graphs.

Finally, turning to geometric settings, similarly motivated problems have been studied in the networking and sensor networks communities, where sensors are often modeled as unit disks. For example, the Maximal Breach Path problem [12] is defined in the context of traversing a region of the plane that contains sensor nodes at predetermined points, and its objective is to maximize the minimum distance between the points on the path and the the sensor nodes.

Similarly motivated problems have been studied in the context of path planning in AI "stealth" path planning problems, in which the task is to find a minimum "visibility" path from source to destination, have been considered in [8, 10, 11]. Although the motivation is similar, such problems are technically quite different from the graph-based problems studied here; nonetheless, our unit-disk and planar settings can be viewed as simplified, stylized version of their geometric settings, where visibility may typically be defined in terms of line-of-sight. Finally, a dual problem studied extensively is *barrier coverage*, i.e., the (deterministic or stochastic) placement of sensors in order to make it difficult for an adversary to cross the region unseen (see [9] and the references therein).

[3] Degree 4 is stated but the construction in fact requires only degree 3.

Contributions. Our main result is a PTAS for Δ-degree unweighted undirected graphs, which provides a $\left\lceil \sqrt{\frac{\Delta+1}{h+1}} \right\rceil$-approximation, where h is the memory parameter, and its two special cases. First, when $\epsilon = 0$ it provides an optimal solution in $O(n \log n)$ time, faster than the known $O(n^2)$ optimal algorithm of [3] for this setting (treating h as constant for both), and with a much simpler analysis. Second, when $\epsilon = \sqrt{\Delta + 1} - 1$, the algorithm collapses to the static algorithm, which we show provides a $\sqrt{\Delta + 1}$ approximation, which is a tighter than the known analysis of [3] for this algorithm.

Our other positive results are showing that: 1) static is an 8-approximation for unit-disk graphs, improving on the 19-approximation analysis of [5] for this case, and that 1-Memory is a 4-approximation; 2) 1-Memory is a 2-approximation algorithm for (directed) planar graphs (see App. B); and 3) (1,6)-Memory is optimal for the hexagon grid graph, and (2,12)-Memory is optimal for the square grid graph (both node-weighted); various other tessellation graphs can be solved similarly (see App. C). Finally, we prove that the problem is NP-hard on node-weighted graphs of degree 3, improving on the known degree-4 hardness result [3].

2 Preliminaries

Definition 1. *For a path P in the given graph $G = (V, E)$, let $v \in P$ indicate that v is a node on the path. Let $N[P] = \bigcup_{v \in P} N[v_i]$, where $N[v]$ indicates the neighborhood of v. Let $cost(P) = |N[P]|$ denote the cost of P, P^* indicate an optimal path, and $cost_0(P) = \sum_{v \in P}(deg(v) - 1)$ be the static cost of P. Observe that $cost(P) \leq cost_0(P)$. Let the* static graph *$G_0 = (V_0, E_0)$ be a directed weighted graph where $V_0 = V$ and for each $\{u, v\} \in E$ there exist $(u, v), (v, u) \in E'$ with weights $\deg(u) - 1$ and $\deg(v) - 1$, respectively.*

We adopt the convention that a path P from s to t is treated as a path from s to some neighbor of t, so that every node on the path *transmits*. Thus $t \in N[P]$ but $t \notin P$. We call nodes that transmit (i.e., nodes on the path) *transmitters*, we call other nodes *nontransmitters* or *non-path* nodes; and we call nodes that receive, i.e., nodes in $N[P]$, *receivers*. Notice that $N[P]$ includes P itself (unless P is a single node), so all transmitters are also receivers. The input graph is assumed to be undirected with unweighted edges and nodes, unless otherwise stated. The extended graphs we construct will be directed and edge-weighted. We use OPT and ALG throughout to represent the cost of an optimal solution and the cost of the (current) algorithm's solution, respectively. Without loss of generality, we ignore the cost of the source node (trivially) receiving when convenient.

Definition 2. *A path Q is* shifted successor *of a path P if $P = (v_1, v_2, ..., v_h)$ and $Q = (v_2, v_3, ..., v_h, u)$, for some node u.*

To motivate the algorithm used in this paper, consider the incremental costs of each edge e_i added, sequentially, to a partial path P. Let $e_i = (v_i, v_{i+1})$,

with vertex indices renumbered accordingly, and let P_i indicate the subpath $e_1, e_2, ..., e_i$ of P. Then the cost of e_1 is $1 + \deg(v_1)$ and for each subsequent edge e_i it is the number of v_i's neighbors being encountered for the first time, i.e., $|N[v_i] - N[P_{i-1}]|$.

Given a graph G, consider the graph G' constructed as follows: for each node v_i of G, introduced a "column" of $O(n!)$ nodes, corresponding to the different possible histories of v_i (i.e., simple paths from the source to v_i). For each edge (v_i, v_j) of G, we introduce in G' two collections of directed edges. First, we introduce edges of the form $v_i^a \to v_j^{a'}$ for all possible histories for v_i and v_j that are *consistent* in the sense that a indicates a simple path from s to a neighbor of v_i and a' indicates that extended to v_i, i.e., to a neighbor of v_j. Similarly, for edges $v_j^a \to v_i^{a'}$. The cost of edge $v_i^a \to v_j^b$ is set to $|N[v_i] - N[P_{i-1}^a]|$. Also, add to G' a new source node and destination node, connected with cost 1 (respectively, 0) edges to all the nodes of the source (respectively, destination) column.

By construction, we have that 1) each node v_i^a in G' corresponds to a simple path in G from source to node v_i, and 2) the cost of each edge $v_i^a \to v_j^b$ of G' corresponds to the the *secluded cost* of the transmission of node v_i in G *after reaching it by taking subpath a from source to* v_i. Thus we state:

Observation 1 *For each path P in G there will be a corresponding path P' in G' whose cost (i.e., sum of edge weighs) equals P's secluded cost, and vice versa.*

Of course, the graph G' constructed above is larger than G by a factor of $O(n!)$. We will find that for several graph classes, we can restrict ourselves to a constant-size memory h, and hence ensure that our expanded graph is only a constant factor (specifically, $O(h^h)$) larger than the original. That is, each node v in G now maps to a column of $O(h^h)$ nodes in G_h. Each of these nodes corresponds to some possible length-h subpath reaching a neighbor of v (or shorter subpath reaching v_i in the case of v near to the source, or a null path if it *is* the source (see Fig. 3.2). Directed edges in G_h now correspond to "recent history" of length $\leq h$ and a shifted successor of it, rather than a monotonic extension as above. The graph G_h can equivalently be viewed as a subgraph of a certain h-dimensional de Bruijn graph whose "symbols" are the nodes of G.

We now observe that computing the cost of $v_i^a \to v_j^{a'}$ on *only some of* the preceding nodes can only *increase* the cost:

Observation 2 *The cost of a path in G_h can be only greater than the cost of the corresponding path in G', and thus also the* secluded *cost of the corresponding path in G.*

We note that in both an optimal solution and in the path returned by Static and h-Memory, each transmitter will be adjacent to its successor and/or predecessor but to no other nodes on the path, since otherwise a shortcut would have produced a shorter path. Call a path with no potential shortcuts *minimal*. By definition of the edge costs, in G_h we are charged for a non-source node when it receives the transmission from its predecessor on the path. A path in G_h costs more than the corresponding path in G when a *nontransmitting* or *non-path* node receives multiple times.

Algorithm 1 Static (0-Memory)

1: **for** each edge (u, v) **do**
2: $\quad e_{uv} \leftarrow \deg(u) - 1$
3: **end for**
4: run Dijkstra on the graph with edge weights $\{e_{uv}\}$

Algorithm 2 h-Memory

1: given the input graph, construct the h-dimensional De Bruijn graph G_h, as described in the text
2: run Dijkstra on G_h

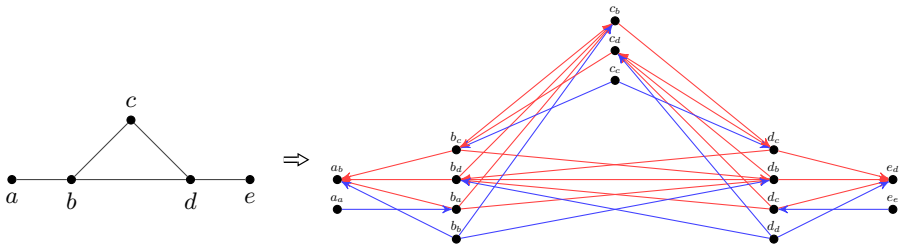


Fig. 1. Construction of the 1-Memory graph G_1 for a simple input graph G. In this example, all b_b's outgoing edges are given cost 3, b_a's are given cost 2, and b_c's and b_d's are given cost 1.

3 Δ-degree graphs

3.1 Static

We begin by proving an approximation guarantee for the memoryless static algorithm, when run on graphs of maximum degree Δ. The proof in this section is a warmup for Sec. 3.3.

Theorem 1. *0-Memory (i.e., static) gives a $\sqrt{\Delta + 1}$-approximation on Δ-degree unweighted undirected graphs.*

See App. A for proof.

3.2 Δ-Memory

Now we consider another extreme, i.e., the h-Memory algorithm with $h = \Delta$. We will use $d_P(a, b)$ to indicate the *distance on the path* P between $a, b \in P$, which we define as the number of edges separating them on P, i.e., the length of subpath $P[a, b]$.

We will first prove a lemma, slightly strengthening a lemma of [3], which gave a similar result for $\Delta + 1$. It tells us there exist optimal solutions in which the length of the subpath between two nodes with a common neighbor is bounded by Δ.

Lemma 1. *There exists an optimal solution P^* in which, if $N[u] \cap N[v] \neq \emptyset$ for $u, v \in P^*$, we have $d_{P^*}(u, v) \leq \Delta$.*

Proof. Suppose two nodes u_0, u_k on a (possibly optimal) path P share a neighbor $x \notin P$. Let $d_P(u_0, u_k) = k$, and name the intermediate nodes so that this subpath consists of the nodes $u_0, u_1, ..., u_{k-1}, u_k$. Let δ be the degree of x, and let P' a modification of P that replaces the subpath $P[u_0, u_k]$ with the subpath u_0, x, u_k.

Let us now consider the costs of these two paths. First we examine the cost of P'. When u_0 transmits, u_1 and x receive; when x transmits, u_k and x's other $\delta - 2$ neighbors (ignoring u_0) receive; when u_k transmits, u_{k-1} receives, for a total of $\delta + 2$ receivers. (We ignore without loss of generality any other neighbors of u_0 and u_k, which will receive from both P' and P.)

Now consider $P[u_0, u_k]$. Its cost is k for nodes $u_1, ..., u_k$ plus 1 for x, plus any other neighbors those nodes may have, for a total of $cost(P[u_0, u_k]) \geq k + 1$. If $k \geq \delta + 1$ then

$$cost(P[u_0, u_k]) \geq k + 1 \geq \delta + 2 = cost(u_0, x, u_k)$$

and so if P is optimal then P' is too. Otherwise, $k < \delta + 1$, and indeed $d_{P^*}(u_0, u_k) = k \leq \delta \leq \Delta$. □

We are now ready to prove the theorem. It shows that while Δ-Memory overestimates some paths' costs—even some optimal paths' costs—it optimally computes the costs of a least *some* optimal path, and so it will find one.

Theorem 2. *Δ-Memory is a polynomial-time optimal algorithm for unweighted, undirected graphs of Δ-bounded degree.*

Proof. Let $cost(P)$ indicate the secluded cost of a path P in the input graph G, and let $cost_h(P)$ indicate the static cost of the corresponding path in G_h.

The lemma tells us that there is always an optimal solution P^* in which the cost of v_i's transmission is determined by (aside from v_i's neighborhood) not the entire subpath from source to v_i but only the last Δ nodes visited prior to reaching v_i. Thus the cost of the corresponding path in G_h will equal the secluded cost of P^*:

$$cost(OPT) = cost_h(OPT) \tag{1}$$

Obs. 2, however, told us that the cost of a path in G_h (for any h, and so in particular for G_Δ) can only be greater than the secluded cost of the corresponding path in G.

Then we have:

$$cost(ALG) \leq cost_\Delta(ALG) \leq cost_\Delta(OPT) = cost(OPT)$$

The first inequality follows from Obs. 2, the second inequality from the fact that the algorithm chooses a path of optimal $c_\Delta(\cdot)$ cost, and the equality from Eq. 1.

Since $\deg(G) = \Delta$, the number of length$\leq \Delta$ subpaths to any node of G is $O(\Delta^\Delta)$. With Δ constant, the size of G_Δ only a constant larger than G, i.e., $|V_\Delta| = O(|V|)$ and $|E_\Delta| = O(|E|)$. Since $|E| = O(|V|)$ (because Δ is constant), Dijkstra can be run on G_Δ in time $O(n \log n)$. □

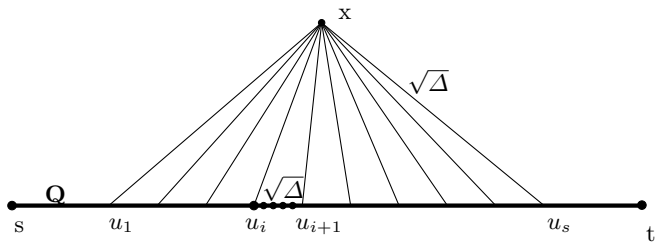


Fig. 2. Approximation guarantee of h-Memory.

3.3 h-Memory for $0 \leq h \leq \Delta$

Finally, we generalize the two previous results for for values of the memory parameter h ranging from 0 to Δ, using an extension of the argument for Theorem 1.

Theorem 3. *h-Memory gives a $\left\lceil \sqrt{\frac{\Delta+1}{h+1}} \right\rceil$-approximation on Δ-degree unweighted undirected graphs in polynomial time.*

Proof. Let $cost_h(P)$ indicate the cost of a path P from the point of view of h-Memory: the cost when node $v_i \in P$ transmits is the number of neighbors receiving who did not receive in the previous h transmissions, i.e., $|N(v_i) - \cup_{j=i-h}^{i-1} N(v_j)|$.

We show that an optimal path P^* can be converted into a path Q of $cost_h(Q) \leq \left\lceil \sqrt{\frac{\Delta+1}{h+1}} \right\rceil \cdot cost(P^*)$. The existence of such a path Q establishes that $ALG \leq \left\lceil \sqrt{\frac{\Delta+1}{h+1}} \right\rceil \cdot OPT$.

Initialize Q to P^* and let all nodes of the graph be *unmarked* (see Fig. 3.3). Consider a node x not on Q. Say that nodes $u_1, ..., u_s$ on Q are h-separated neighbors of x, i.e., they are all in $N(x)$ but for each $2 \leq i \leq s$, the h nodes preceding u_i on Q are not in $N(x)$. That is, between each successive pair of x's neighbors on Q there are at least h nodes on Q that are not neighbors of x.

Now say that a node x not on Q is *bad* if it has more than $\left\lceil \sqrt{\frac{\Delta+1}{h+1}} \right\rceil$ unmarked h-separated neighbors on Q, say, $u_1, ..., u_s$. Let the *length* of x be $d_Q(u_1, u_s)$. Say that x is *worst* if has maximum length among all bad nodes.

Now we iteratively modify Q by repeatedly doing the following: choose a worst node x with unmarked h-separated neighbors $u_1, ..., u_s$ on Q. Then we replace the subpath $Q[u_1, u_s]$ with the subpath (u_1, x, u_s), and we mark x.

Now we examine the final value of $cost_h(Q)$.

Consider a move that adds a node x to the path. This occurs when x has $s > \left\lceil \sqrt{\frac{\Delta+1}{h+1}} \right\rceil$ neighbors on Q. We now examine how this move changes the cost of Q. Prior to the move, the nodes $succ_Q(u_1), ..., u_s$ and x all receive (due to the transmission of $u_1, ..., pred_Q(u_s)$). Recall that $u_1, ..., u_s$ are h-separated, with at least h nodes in between each successive pair u_i, u_{i+1}, and so $|[u_1, u_s]| \geq$

$(s-1)\cdot(h+1)+1$. Therefore the transmissions of $u_1, ..., pred_Q(u_s)$ contribute at least $(s-1)\cdot(h+1) \geq \sqrt{\frac{\Delta+1}{h+1}}\cdot(h+1) = \sqrt{(\Delta+1)\cdot(h+1)}$ to $cost(Q)$.

After the move, the transmission of x (to nodes $u_2, .., u_s$ and possibly others) is of $cost_h$ at most $\Delta - 1$. In addition, there is $cost_h$ 1 each for $succ_Q(u_1)$ and x to both receive from u_1. Altogether, the modified path incurs an h-memory cost of at most $\Delta + 1$ due to the transmitters u_1, x, replacing the true cost s of the nodes they replace.

Thus we charge Δ+1 to the substring $P^*[u_1 u_{s-1}]$, which is less than $\frac{\Delta+1}{\sqrt{(\Delta+1)\cdot(h+1)}} = \sqrt{\frac{\Delta+1}{h+1}}$ times the cost of that substring's transmissions.

Now consider neighbors y of Q that we never make such moves based on, i.e., nodes y that are neighbors of at most $\left\lceil\sqrt{\frac{\Delta+1}{h+1}}\right\rceil$ unmarked h-separated nodes on Q. We pay at most $\left\lceil\sqrt{\frac{\Delta+1}{h+1}}\right\rceil$ in h-memory cost for each such node *due to transmissions of unmarked nodes*, rather than the true cost of 1. Thus the h-memory cost of Q is at most $\left\lceil\sqrt{\frac{\Delta+1}{h+1}}\right\rceil$ times the true cost of P^*.

Since h is bounded by the constant Δ, the number of length-h subpaths is again $O(\Delta^\Delta)$, and so the running time is again $O(n \log n)$. □

Finally, we observe that for any $\epsilon \geq 0$, setting[4] $e = \lfloor\epsilon\rfloor$ and $h = \frac{\Delta+1}{1+e^2} - 1$ yields $\left\lceil\sqrt{\frac{\Delta+1}{h+1}}\right\rceil = \sqrt{\frac{\Delta+1}{h+1}} = 1 + e \leq 1 + \epsilon$. The second equality is obtained by algebra, and the first equality holds because $1 + e$ is an integer. Thus we conclude:

Corollary 1. *h-Memory is a PTAS for the the secluded path problem on unweighted, undirected fixed-Δ degree graphs.*

4 Unit-disk graphs

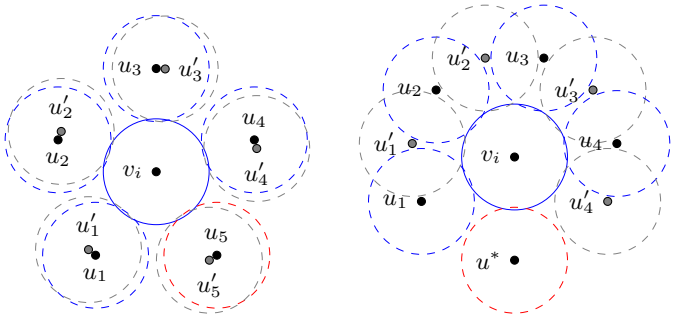
We begin by proving a lemma showing that the cost of every (minimal) secluded path in G is inflated by a factor of a most 8 in G_0. Thus there will exist a path in G_0 of length at most 8 times the optimal secluded path cost, which therefore upper-bounds the cost of the solution returned by the algorithm.

Lemma 2. *In the case of a minimal solution, a nontransmitter receives at most 8 times.*

Proof. To obtain the bound, first recall that a unit disk can have at most 5 mutually disjoint neighbors, say $u_1, ..., u_5$ (see Fig. 3(a)). Therefore there can be at most 10 transmitting neighbors of v lying on a minimal subpath.

Consider the (minimal) subpaths $P_{10} = (u_1, u'_1, ..., u_2, u'_2, ..., u_3, u'_3, ..., u_4, u'_4, ..., u_5, u'_5)$ and P_9, with the latter the same except with u_1 omitted (assume u'_5 is not the destination), in the instance of Fig. 3(a). First v has no other neighbors. Then

[4] $e = 0$ when $\epsilon < 1$.



(a) 8-approximation. (b) Factor-8 instance.

Fig. 3. 8-approximation of Static on unit disk graphs.

P_{10} costs more (at least 20: each transmission by one of v's 10 neighbors transmits to v and has cost at least 2) than the subpath $P' = (u_1, v, u_5')$ does (as low as 2+9+2=13, with v being paid for only once), and so P_{10} cannot be a subpath of a shortest path. (Similarly for P_9, with costs 18 versus 12.) In order to be charged for v_i more than once (i.e., in order for it not to lie on the shortest path), having v_i transmit must be prohibitively expensive, i.e., it must have additional neighbors who do not receive the message when path P transmits, co-located at, say, node u^*.

If v *does* have any additional neighbors, they each would necessarily receive the transmission, at least once, from the nodes of P_{10} (or P_9), since the minimality property implies that these disks occupy an arc sweeping out greater than 5/6 of v's, and so there is no space for another disk to intersect v but not P_9.

Combined with the minimality property, the constraint that the path's disks not intersect with u^* limits its number to 8 (see Figs. 3(a) and 3(b)). □

Proposition 1. *Static provides an 8-approximation when run on node-weighted UDGs.*

Proof.

$$cost(ALG) \leq cost_0(ALG) \leq cost_0(OPT) \leq 8 \cdot cost(OPT)$$

The inequalities follow from 1) Obs. 2, 2) the fact that the algorithm returns a path of minimum $cost_0(\cdot)$, and 3) the lemma. □

An example in which the factor-8 approximation obtains is shown in Fig. 3(b). Here $u_1, u_1', ..., u_4, u_4'$ are individual nodes; v_i indicates n co-located nodes (for some large n), and u^* indicates n^2 co-located nodes.

Proposition 2. *1-Memory provides a 4-approximation when run on node-weighted UDGs.*

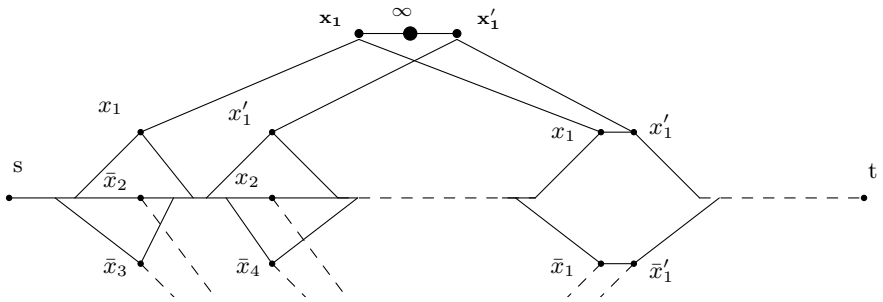


Fig. 4. Hardness reduction.

Proof. The proof proceeds very similarly to the proof of Proposition 1, this time showing that we pay for each nontransmitter at most 4 times. Because of the 1-Memory property, in the instance of Figs. 3(a) and 3(b), path $P_{10} = (u_1, u'_1, ..., u_2, u'_2, ..., u_3, u'_3, ..., u_4, u'_4, ..., u_5, u'_5)$ would be charged for node v_i only once. Arrangements such the one shown in Fig. 3(a) can be extended to paths P' in which v_i receives from the transmissions of multiple *noncontiguous* subpaths. We will then be charged for v_i once for each such maximal subpath. Recall that a unit disk can intersect with at most 5 independent unit disks, meaning at most 5 such subpaths. As before, the fact that v_i is not chosen to transmit means there must be a neighbor of v_i to avoid, which reduces the worst-case total to 4. □

5 NP-hardness

Proposition 3. *The secluded path problem is NP-hard on node-weighted of degree 3.*

Proof. We reduce from the special case of 3-SAT in which each literal appears at most twice. We create a graph in which the optimal secluded path cost equals $2n$ if the formula is satisfiable and is more otherwise (see Fig. 4). For each clause we add a *clause gadget*, in which the path splits into three length-2 subpaths, corresponding to the literals of a clause, which then merge back together. (The first clause gadget from the left in the example shown in the figure is for $(x_1 \vee \bar{x}_2 \vee \bar{x}_3)$.) Each clause gadget's "literal nodes" (x_i) have a neighbor of weight 1 ($\mathbf{x_i}$ or $\mathbf{x'_i}$). Those neighbors have very heavy neighbors themselves, so that they cannot be part of any shortest path. The cost of all other vertices in the graph is 0. A clause gadget thus forces a good solution path to choose one of three alternatives corresponding to the clause's three literals.

To prevent an optimal solution path from visiting both x_i and $\bar{x}_i$, we add *variable gadgets* at the end of the construction (one variable gadget is shown in the figure). Each (for variable x_i) forces a solution path to make one of two choices for a variable, which means paying 2 nodes neighboring x_i, x'_i ($\mathbf{x_i}$ and $\mathbf{x'_i}$) or those neighboring $\bar{x}_i, \bar{x}'_i$ ($\bar{\mathbf{x}}_\mathbf{i}$ and $\bar{\mathbf{x}}'_\mathbf{i}$). Thus the variable gadgets by themselves

therefore give a cost a total of $2n$ Thus a path will cost more than $2n$ iff it corresponds to a non-satisfying assignment.

6 Discussion

Several of the results in this paper used a lemma that on an optimal solution path two nodes with a common neighbor cannot be too far apart, which permitted the use of de Bruijn graphs to implement enough memory to correctly represent transmission costs using edge costs. In a similar spirit, the unit-disk approximation result depended on showing that there cannot be too many nodes on an optimal path sharing a neighbor. In all cases, we could then obtain good results computing shortest paths.

The itchiest open problem is proving the problem to be easy or hard on planar and/or unit-disk graphs. Note that the triangular grid graph, whose status is also open, is a special case of both the planar and unit-disk graphs. It appears that positive results for these settings will require fundamentally different techniques.

References

1. R. D. Carr, S. Doddi, G. Konjevod, and M. V. Marathe. On the red-blue set cover problem. In *SODA*, pages 345–353, 2000.
2. D. Chavey. Tilings by regular polygons: A catalog of tilings. *Computers & Mathematics with Applications*, 17(1–3):147–165, 1989.
3. S. Chechik, M. P. Johnson, M. Parter, and D. Peleg. Secluded connectivity problems. In *ESA*, 2013.
4. N. G. de Bruijn. A combinatorial problem. *Koninklijke Nederlandse Akademie v. Wetenschappen*, 49(49):758–764, 1946.
5. J. Gao, Q. Zhao, and A. Swami. The thinnest path problem for secure communications: A directed hypergraph approach. In *Allerton Conference on Communications, Control, and Computing*, 2012.
6. J. Gao, Q. Zhao, and A. Swami. The thinnest path problem for secure communications: A directed hypergraph approach. In *Allerton Conference on Communication, Control, and Computing*, pages 847–852, 2012.
7. B. Grunbaum and G. C. Shephard. *Tilings and Patterns*. W. H. Freeman and Company, 1987.
8. A. Johansson and P. Dell'Acqua. Knowledge-based probability maps for covert pathfinding. In *Proceedings of the Third international conference on Motion in games*, MIG'10, pages 339–350, Berlin, Heidelberg, 2010. Springer-Verlag.
9. B. Liu, O. Dousse, J. Wang, and A. Saipulla. Strong barrier coverage of wireless sensor networks. In *MobiHoc*, pages 411–420, 2008.
10. M. Marzouqi and R. Jarvis. New visibility-based path-planning approach for covert robotic navigation. *Robotica*, 24(6):759–773, 2006.
11. M. A. Marzouqi and R. A. Jarvis. Robotic covert path planning: A survey. In *RAM*, pages 77–82, 2011.
12. S. Meguerdichian, F. Koushanfar, M. Potkonjak, and M. B. Srivastava. Coverage problems in wireless ad-hoc sensor networks. In *INFOCOM*, pages 1380–1387, 2001.
13. D. Peleg. Approximation algorithms for the label-cover$_{\max}$ and red-blue set cover problems. *J. Discrete Algorithms*, 5(1):55–64, 2007.

A Proof of Theorem 1

Proof. We show that an optimal path P^* can be converted into a path Q of static cost $cost_0(Q) \leq \sqrt{\Delta+1} \cdot cost(P^*)$ by a modification of the argument of [3]. The existence of such a path Q establishes that $ALG \leq \sqrt{\Delta+1} \cdot OPT$.

Initialize Q to P^* and let all nodes of the graph be *unmarked*. Say that a node x not on Q is *bad* if it is the neighbor of more than $\sqrt{\Delta+1}$ unmarked nodes on Q, say, $u_1, ..., u_s$. Let the *length* of x be $d_Q(u_1, u_s)$. Say that x is *worst* if has maximum length among all bad nodes.

Now we iteratively modify Q by repeatedly doing the following: choose a worst node x with unmarked neighbors $u_1, ..., u_s$ on Q. Then we replace the subpath $Q[u_1, u_s]$ with the subpath (u_1, x, u_s), and we mark x.

Now we examine the final value of $cost_0(Q)$.

Consider a move that adds a node x to the path. This occurs when x has $s > \sqrt{\Delta+1}$ neighbors on Q. We now examine how this move changes the cost of Q. Prior to the move, the nodes $u_2, ..., u_s$ and x all receive (due to the transmission of nodes $u_1, ..., u_{s-1}$), which contributes $(s-1)+1 = s$ to $cost(Q)$. After the move, the transmission of x (to nodes $u_2, .., u_s$ and possibly others) is of $cost_0$ at most $\Delta - 1$. In addition, there is $cost_0$ 1 each for u_2 and x to both receive from u_1. Altogether, the modified path incurs a static cost of $\Delta + 1$ due to the transmitters u_1, x, replacing the true cost s of the transmissions they replace.

Thus we charge $\Delta+1$ to the substring $P^*[u_1, u_{s-1}]$, which is less than $\frac{\Delta+1}{\sqrt{\Delta+1}} = \sqrt{\Delta+1}$ times the cost of that substring's transmissions.

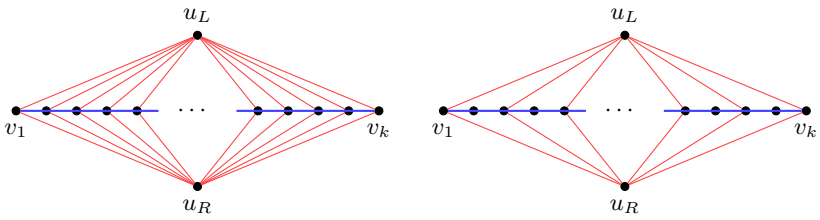
Now consider neighbors y of Q that we never make such moves based on, i.e., nodes y that are neighbors of at most $\sqrt{\Delta+1}$ unmarked nodes on Q. We pay at most $\sqrt{\Delta+1}$ for each such node *due to transmissions of unmarked nodes*, rather than the true cost of 1. Thus the static cost of Q is at most $\sqrt{\Delta+1}$ times the true cost of P^*. □

B Planar graphs

Proposition 4. *Static provides a 3-approximation when run on (unweighted, directed) planar graphs.*

Proof. We claim that any path P returned by Static—let us say this path has minimum *static cost*—will have $cost_0(P) \leq 3 \cdot cost(P)$. The static cost and true cost differ because in the former we are charged each time a given node receives the message. Consider a shortest static-cost path $P = (v_1, ..., v_k)$. Consider how much worse the static cost $cost_0(P)$ can be than the true cost $cost(P)$. No node on P can be a neighbor of a preceding node on P other than its predecessor, since otherwise P is not an optimal static-cost path.

Nodes on P may jointly be neighbors of the one or more nodes not on P, however (Fig. 5(a)). We claim that if there are multiple such nodes on the same "side" of the path P instances can be constructed with one fewer node on that side, with only higher ratio $cost_0(P)/cost(P)$. If there are two such nodes u_a, u_b



(a) 3-approximation by Static. (b) 2-approximation by 1-Memory.

Fig. 5. Approximation on planar graphs.

whose neighbors on P are disjoint, then u_a, u_b can be merged into a single node u. (A sufficiently large number of new neighbors of u can be added to prevent it from being possible to to take a "shortcut" via u, thus not changing P's status as optimal static-cost path.) If u_a, u_b share neighbors among the nodes of P (note that they can share at most two such neighbors), an adjacent edge of P can be split to introduce at most 2 new nodes adjacent to the merged node u. Thus by merging u_a, u_b we hold $cost_0(P)$ fixed, while decreasing $cost(P)$. Without loss of generality, therefore, we may assume that there is at most one such node on either side of P. Similarly, any case of a node that neighboring the path "from both sides" can be made worse by splitting the node into two, and repeating the above. Therefore the static cost of each transmission is at most 3 times the true cost.

Note that as k grows large, the approximation factor of the instance shown in Fig. 5(a) approaches the worst case of 3. □

Unfortunately, this approximation ratio does not hold in the case of node-weighted graphs, since we can modify the example as follows. First, delete u_R, and let the destination be a new node v_{k+1} adjacent only to v_k. Let u_L have just one other neighbor besides v_i (in which case $\Delta = k + 1$). Let the weight of u_L be some value $w >> k$, let u_L's other neighbor have cost w^2, and let all other nodes have unit weight. Then the static cost of the path from v_1 to v_{k+1} is approximately kw, whereas the true cost is approximately w, yielding an approximation factor of $k = \Delta - 1$.

Consider what happens when 1-Memory is run on the worst-case example above. Each transmission after the first remembers that u_L, u_R have received, and the path's cost is computed with no error. The algorithm is vulnerable to problem instances with gaps, although only at a cost of a factor-2 approximation rather than 3.

Proposition 5. *1-Memory provides a 2-approximation when run on (unweighted, directed) planar graphs.*

Proof. As in the earlier proof, again consider an optimal path P, from some node v_1 to some node v_k, and think about how to make an arbitrary situation worse.

Algorithm 3 (h, i)-Memory

1: First draw the h-memory graph as before.
2: Now, create 2^i copies, called *layers*, of this graph, each L_S corresponding to different subset of S of a set $D = \{d_1, ..., d_i\}$ of distinguished nodes having so far received. The source points to layer $L_\emptyset$ and all layers point to the target.
3: Starting with $L_\emptyset$, modify the edges at any layer S this way: for each node u with *distinguished neighbors* $D' \subseteq D$ redirect u's outgoing edges to the corresponding vertices in layer $L_{S \cup D'}$. The cost of these edges will include the cost of $D' \setminus S$, but edges in this layer and in all its superset layers will not.
4: Run Dijkstra on this graph.

Clearly no single node o should neighbor two successive nodes $u, v \in P$, since 1-Memory will remember o received from u when v transmits. Then arguments as above will conclude that a series of modifications making things worse will converge to a situation where all *alternating* nodes—the odd nodes v_{2i+1}, say—should share a common neighbor on one side. Similarly, alternating nodes on the other side—again, either the odds or the evens—should share a common neighbor.

Thus we can assume the worst-case instance given in the figure above (or else one in which u_L neighbors the nodes that u_R does not) *without loss of generality.* The cost for 1-Memory of two successive transmissions will be $|N[u_i, u_{i+1}]| = |\{v_{i+1}, v_{i+2}, u_L, u_R\}| = 4$, whereas the true cost is 2. Overall, the total static cost will be approximately $(k/2) \cdot 4$ for static while the true cost is $k + 2$. □

Unfortunately, extending to h-Memory for $h > 2$ improves the performance no further, since two far-apart nodes could share many neighbors.

C Tessellation graphs

A tessellation of the plane can be interpreted as a network in which tiles become nodes and adjacent tiles can communicate, which corresponds to the dual graph of the tessellation. There are three regular tessellations, based on triangles, squares, and hexagons. The hexagon and triangle tilings are dual to one another, and the square tiling is self-dual.

Although we show that the problem is hard on the class of node-weighted graphs of degree 3, we find in this section that some node-weighted grid graphs (of degree ≥ 3) are optimally solvable. (If the nodes are not weighted, then since such graphs have constant degree, we already know they are solvable.)

We start by showing that in hexagon and square grid graphs, in general two nodes $u, v \in P^*$ with a common neighbor cannot be very apart—apart from a few exceptions, which we call *distinguished nodes* D. To handle these nodes that could have very far-apart neighbors, we also simply keep track of which subset of these nodes have received so far. This $|D|$ bits of information is stored by multiplying the constructed graph by a factor of $2^{|D|}$, as described in Algorithm C.

First we consider hexagons.

Lemma 3. *In the hexagon grid graph, let o be a nontransmitter in $N[P^*]$ for some optimal path P^*, where $o \notin N[s]$. Then if o receives multiple times, it does so from either 2 or 3 consecutive nodes in P^*.*

Proof. We prove the stated property for all such nodes o by showing that if o does violate the property, then we can modify P^* in such a way that only improves the solution—that is, the neighborhood of the modified path is a *subset* of the neighborhood of the original path—and does not cause any other node to newly violate the property.

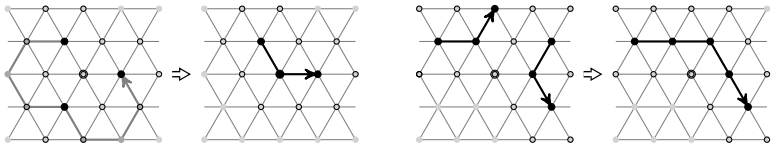
First, we claim that no such node o will receive from 3 of its 6 neighbors, say m_1, m_2, m_3 (in order) unless those nodes form a path, because if o does receive from 3 such nodes, then the subpath from m_1 to m_3 can be replaced with the path (m_1, o, m_3), which renders o a transmitter, and which does not result in any new node violating this property. The transformation for such case, the one in which o's 3 transmitting neighbors are independent, is shown in Fig. 6(a). (One possible path connecting these 3 nodes is shown, for illustrative purposes only, in light gray.) The other possible cases of 3 receives are similar, as are the cases of a) more than 3 receives and b) receives for o from 2 neighbors in *opposite* directions.

The only remaining case is when o receives from 2 neighbors that are neither consecutive nor opposite, which is thus 2 transmitting neighbors, say m_1, m_2 (in order) of o that share a (nontransmitting) neighbor with o. Since $o \notin N[s]$, we know that m_1 is not s, and so must receive from some other node, say u_a. It must transmit to some *transmitting* node u_b, which cannot be a neighbor of o because o's transmitting neighbors are independent. This leaves 3 possible neighbors of m_1 which could be u_a, u_b. Those two nodes cannot themselves be neighbors, however, because otherwise we could replace (u_a, m_1, u_b) with (u_a, u_b). This fully determines the locations of u_a, u_b relative to m_1 except that there are two cases for which is which.

Similarly, m_2 cannot be t (since it is transmitting), and must receive from a transmitter $u_c \notin N[o]$ and must transmit to a node $u_d \notin N[o]$, although u_d may be nontransmitting (because it may be t). Combined with the two possible assignments of u_c and u_d, this yields 4 subcases for m_2, for a total of 8 subcases. The transformation for one of these 8 subcases in shown in Fig. 6(b), which results in o now receiving from 3 consecutive neighbors, and does not result in any new node violating the property. The other subcases are similar. □

Proposition 6. *(1,6)-Memory is optimal for the hexagon grid graph.*

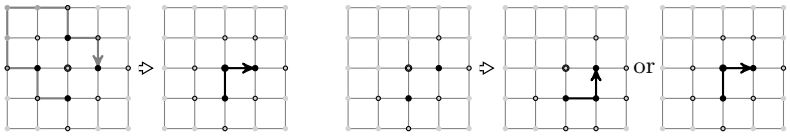
Proof. By the lemma, for all nontransmitters $o \notin N[s]$, any multiple receives for o must be consecutive. Therefore for such transmissions, whether o should be charged depends only on the previous transmission. Hence to determine the cost of a transmission it suffices to remember the previous transmission *and* which subset of $N[s]$ have yet received. □



(a) Transformation for 3 or more receives.

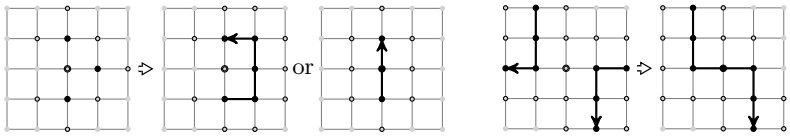
(b) Transformation for 2 diagonal receives.

Fig. 6. Local transformations ensuring that the center node o either a) receives from either 2 or 3 transmissions by nodes on the path, where such transmissions are interspersed with exactly 1 transmission in which o does *not* receive or b) becomes a transmitting node on the path. In each case, the set of transmitting or receiving nodes on the right is a *subset* of those on the left.



(a) Transformation for 4 receives.

(b) Transformation for 2 diagonal receives.



(c) Transformation for 3 receives.

(d) Transformation for 2 opposite receives.

Fig. 7. Local transformations ensuring that the center node o either a) receives from either 2 or 3 transmissions by nodes on the path, where such transmissions are interspersed with exactly 1 transmission in which o does *not* receive or b) becomes a transmitting node on the path. In each case, the set of transmitting or receiving nodes on the right is a *subset* of those on the left.

We begin by considering the (weighted) square grid graph. We will show that a version of the Memory algorithm is optimal for such graphs. We begin by showing that, aside from nodes that are neighbors of s or *diagonal* neighbors of either s or t, when nontransmitters hear the message multiple times from an optimal path, there are either two or three such transmitters, each of which after the first comes exactly two transmissions after the most recent one.

Lemma 4. *In the square grid graph, let o be a nontransmitter in $N[P^*]$ for some optimal path P^*, where $o \notin N[s] \cup N_/[s] \cup N_/[t]$. Let $M(o) = \{m_1, ..., m_k\}$ be the nodes in P^* (in order) that transmit to o. Then $|M(o)|$ is either 2 or 3, $d_{P^*}(m_1, m_2) = 2$, and (in the latter case) $d_{P^*}(m_2, m_3) = 2$.*

Proof. Let o be such a node. First, we observe that in an optimal solution, we can never have $|M(o)| = 4$. Any solution in which the (nontransmitting) node

o hears 4 times, hence from all four of its neighbors, can be transformed into a solution of only lower cost in which that node is now transmitting, i.e., on the path, and hence receives exactly twice. Suppose o receives from its neighbors in order (say) $(0,-1),(-1,0),(0,1),(1,0)$. (See Fig. 7(a), in which these nodes are shaded dark, nodes receiving from them are ringed, o is marked with a large ring, placed at the center, and made the origin, and one possible path through o's neighbors is shown in gray.) Then we can replace the subpath from $(0,-1)$ to $(1,0)$ with the path $((0,-1),o,(1,0))$.

Second, if $|M(o)| = 2$, with the transmitting neighbors *not* on opposite sides of o, located at (say) $(0,-1)$ and $(1,0)$. Then a case analysis shows that path can be transformed by substituting the subpath from $(0,-1)$ to $(1,0)$ with either the path $((0,-1),o,(1,0))$ or $((0,-1),(1,-1),(0,1))$ (see Fig. 7(b)). The transmissions to o (or, in the second case, to $(1,-1)$, are indeed two transmissions apart.

Third, a case analysis shows that when $|M(o)| = 3$, with the neighbors located at (say) $(0,-1),(1,0),(0,1)$, an optimal solution must circle around o or pass through o, in both of which cases there is exactly one transmission separating each such transmission from the most recent. The transformation for the case in which the original path's three transmissions occur *in order* $(0,-1),(1,0),(0,1)$ is shown in Fig. 7(c). Transformations for the cases of other orders of these three transmissions are similar.

Finally, by a much more elaborate case analysis, it can be shown that for such o, if the two transmitters *are* on opposite sides of o, at say $(-1,0)$ and $(1,0)$, then the path can be transformed into one of the cases already handled above, in such a way that the solution only improves. The case analysis proceeds by consider 2 possible ways for the path to pass through $(-1,0)$ and 3 possible ways for it to pass through $(1,0)$. The transformation for one of the 12 cases is shown in Fig. 7(d). □

Proposition 7. *(2,12)-Memory is optimal for the square grid graph.*

Proof. By the lemma above, aside from handling the nodes in $N[s] \cup N_/[s] \cup N_/[t]$, it suffices to "remember" the last two moves in determining the cost of any given transmission. We also must remember what subset of the 12 nodes in the set mentioned have received the message. Thus a "memory" graph in which each node is represented by $2^{12} \cdot 4 \cdot 4$ nodes will suffice. □

Interestingly, this approach does not apply to the triangular tiling, however, since in an optimal path P^* may transmit to a node (triangular face) o twice from nodes arbitrarily far apart on P^* without o's third neighbor receiving (which could have arbitrarily high cost), and hence without the path being improvable by shortcuts, either through o or through other nodes (see Fig. 8).

The approach does apply, though, to several of the 8 semiregular tilings or their duals (details omitted):

Proposition 8. *For appropriately chosen constants, (k_1,k_2)-Memory is optimal for the grid graphs resulting from the following semiregular tilings: truncated*

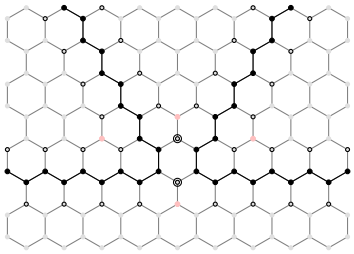


Fig. 8. Triangular grid graph example in which two node (marked with larger rings) receives from two transmissions on the path at arbitrarily large separation. All non-receivers, in particular the four colored faint red, could have high cost, precluding shortcuts.

square (dual), truncated hexagon (dual), trihexagonal (primal), rhombotrihexagonal (primal), and truncated hexagonal (primal).