Searching by Panning and Zooming

Prosenjit Bose[*] John Howat[†] Pat Morin[*]

Abstract

We consider the problem of searching through visualizations of large amounts of data by performing *panning* and *zooming* operations. We give algorithms to travel from a source to an unknown target using a number of pans and zooms that is competitive with the optimal number of pans and zooms depending on certain parameters, and consider both the one-dimensional and two-dimensional versions of this problem.

1 Introduction

The ability to navigate through large amounts of data has become an important skill with the proliferation of visualization tools. Imagine navigating financial data to evaluate the performance of a stock: initially, data for the current week might be visible, but the user can zoom out to get a broader picture of long-term trends and find a particular week of interest. A similar task is that of navigating a digital map. Perhaps the user might begin with the screen centered and fully zoomed-in on his or her home, and must find the location of some buried treasure that is marked elsewhere on the map. To complete the search, the user must center his or her fully-zoomed screen on the treasure. In both cases, one can simply pan around the data until successful, but it is likely much faster to zoom out so that subsequent pans move over a much larger area.

We will consider two operations: *panning* (*i.e.*, translating the view so that it is centered elsewhere), and *zooming* (*i.e.*, increasing the detail of the view but decreasing the total area viewed). We wish to minimize the total number of pans and zooms to get from a *source* location to a *target* location. Note that in the examples in the preceding paragraph, the financial data problem is one-dimensional, since the user can pan forward and backward in time, while the digital map example is two-dimensional, since the user can pan horizontally and vertically.

Such navigation tasks have received attention in the human-computer interaction literature (*e.g.*, [1, 2, 3, 5]). In particular, van Wijk and Nuij [5] consider optimal zooming and panning, but in a continuous setting and with the requirement of smoothness, for the sake of animation. To our knowledge, a rigorous examination of the number of zooms and pans required (and how to apply them) has not been undertaken in the discrete case.

Results. We consider the number of pans and zooms required in both the one- and two-dimensional cases and offer algorithms that are competitive with the optimal number of pans and zooms depending certain parameters.

2 Definitions and Model

In this section, we formally define the model for searching by panning and zooming. The model consists of an undirected graph $G_{n,d}$. The parameter n refers to the number of *world vertices*, which correspond to what the user sees when he or she is fully zoomed-in, and the parameter d indicates that pans may be performed along one of d axes, for a total of $2d$ possible pans. For simplicity, we assume that n is a power of $2d$.

We construct the graph $G_{n,d}$ over $\log_{2d} n$ *zoom levels*. At the bottom level (level 0), are the n world vertices arranged in a d-dimensional $n^{1/d} \times n^{1/d} \times \cdots \times n^{1/d}$ grid. At level $0 < i < \log_{2d} n$ are $n/(2d)^i$ *zoom vertices*, which correspond to what the user sees when zoomed out. These vertices are arranged in a d-dimensional $n^{1/d}/(2d)^{i/d} \times n^{1/d}/(2d)^{i/d} \times \cdots \times n^{1/d}/(2d)^{i/d}$ grid. There are two types of edges in $G_{n,d}$: *pan edges* and *zoom edges*. A vertex at level $0 \leq i < \log_{2d} n$ has $2d$ pan edges (or fewer, at the extremes of the grid) to adjacent vertices in the grid on level i. On level $0 \leq i < \log_{2d} n - 1$, every vertex in each contiguous $(2d)^{1/d} \times (2d)^{1/d} \times \cdots \times (2d)^{1/d}$ block of vertices has a zoom edge to the vertex on level $i+1$ that is above that block if the two grids were overlayed (and scaled appropriately). Traversing a zoom edge from a lower level to a higher one is called *zooming out*, while traversing a zoom edge from a higher level to a lower one is called *zooming in*. Figures 1 and 2 show $G_{16,d}$ for $d = 1$ and $d = 2$.

Two vertices u and v have a lowest common ancestor $\mathrm{lca}(u, v)$ that is defined as the lowest common ancestor of u and v in the tree induced by the zoom edges. The height of a vertex u is denoted $\mathrm{h}(u)$ and is defined as the

[*]School of Computer Science, Carleton University, Ottawa, Canada. Email: `{jit,morin}@scs.carleton.ca`. Research partially supported by NSERC.

[†]School of Computing, Queen's University, Kingston, Canada. Email: `howat@cs.queensu.ca`.

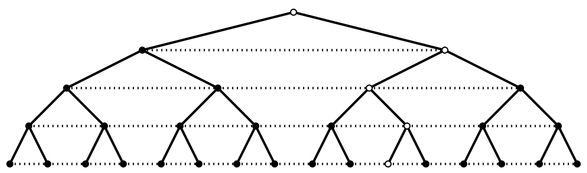

Figure 1: The graph $G_{16,1}$. Solid edges are zoom edges and dotted edges are pan edges. White vertices are marked.

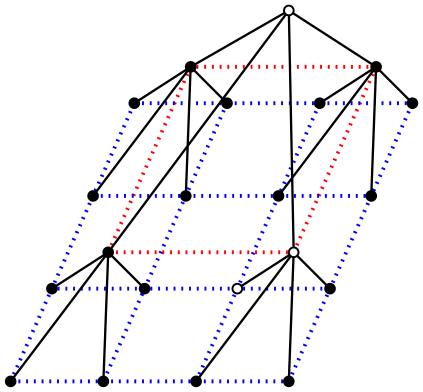

Figure 2: The graph $G_{16,2}$. Solid edges are zoom edges and dotted edges are pan edges. For clarity, pan edges on different zoom levels are coloured differently. White vertices are marked.

height of the vertex u in the tree induced by the zoom edges.

The problem is to start at a world vertex s and reach a different world vertex t by traversing a sequence of pan and zoom edges, and the cost of the traversal is the number of pan and zoom edges used. Our goal is to use a number of pans and zooms as close as possible to the length of a shortest path between s and t in $G_{n,d}$, denoted $\delta(s,t)$. To assist in this task, all vertices that are reachable from t using a sequence of only zoom edges are *marked*. Note that the marked vertices are precisely those on the path in the tree induced by the zoom edges from the target (which is a leaf) to the root.

We also consider a parameter r which represents the number of vertices viewable at once. At a given vertex, a traversal algorithm knows the location of any marked vertex that can be reached by traversing a sequence of at most r pan edges (and no zoom edges). Additionally, if the traversal algorithm is currently at a marked vertex, it knows which child of that vertex is marked.

3 Searching in One Dimension

In this section, we study the one-dimensional case, when $d = 1$ and the graph $G_{n,1}$ looks similar to a level-linked tree [4]. We begin with two observations about shortest paths in this graph.

Observation 1 *For any two vertices u and v on the same zoom level of the graph $G_{n,1}$, if there is a path of $k \leq 3$ pan edges between u and v, then $\delta(u,v) = k$.*

Proof. Since u and v are connected by a path of k pan edges, we know that $\delta(u,v) \leq k$. We must now show that $\delta(u,v) \geq k$. Since moving to a child of u or v produces a strictly longer path, we need only consider paths using vertices on the level of u and v and above.

Any shortest path between u and v that is not the unique path consisting only of pan edges must use zoom edges at least twice: once to zoom out and once to zoom in. If $k \leq 2$, then any path that uses zoom edges will have length at least 2 and so $\delta(u,v) \geq 2 = k$. Otherwise, if $k = 3$, u and v zoom out to different vertices, which means there is at least one pan edge between them, so the total path length is at least 3 and therefore $\delta(u,v) \geq 3 = k$ □

Observation 1 tells us that if we wish to move to another vertex at the same level and it is within 3 pan edges, then the shortest path to that vertex is simply to walk directly to it along those pan edges, *i.e.*, there are no savings to be had by additional zooming out.

For the next observation, we will use $p(v)$ to denote the unique vertex reached by traversing the zoom edge at vertex v to the zoom level above it.

Observation 2 *For any two vertices u and v on the same zoom level of the graph $G_{n,1}$, if there is a path of $k \geq 4$ pan edges between u and v, then $\delta(u,v) = 2 + \delta(p(u), p(v))$.*

Proof. Since $k \geq 4$, there is a shortest path from u to v that zooms out from the level that u and v are on. Furthermore, any pan edges on a shortest path must be between on a path from u to v, or $p(u)$ and $p(v)$, or $p(p(u))$ and $p(p(v))$, and so on. Therefore, one shortest path from u to v consists of a sequence of $k_1 \geq 0$ pan edges from u towards v ending at u', followed by a zoom edge to $p(u')$. Symmetrically, this shortest path from u to v contains a sequence of $k_2 \geq 0$ pan edges from v towards u ending at v', preceded by a zoom edge from $p(v')$.

We argue that a shortest path from u to v has $k_1 = k_2 = 0$, so that $u = u'$ and $v = v'$, which shows that $\delta(u,v) \geq 2 + \delta(p(u), p(v))$. To see this, observe that the path from u to u' to $p(u')$ has length $k_1 + 1$. However, the path from u to $p(u)$ and then onto $p(u')$ has length at most $1 + \lceil k_1/2 \rceil \leq 1 + k_1$. Symmetrically, the path from $p(v')$ to v' to v can be replaced by a path of length at most $1 + \lceil k_2/2 \rceil \leq 1 + k_2$. Therefore, a shortest path between u and v zooms out from u to $p(u)$, takes the shortest path from $p(u)$ to $p(v)$, and then zooms in to v, which has length $2 + \delta(p(u), p(v)) = \delta(u,v)$. □

Observation 2 tells us that if we wish to move to another vertex at the same level and it is more than 3 pan edges away, then the shortest path to that vertex involves zoom edges (at least one to zoom out from the current zoom level and at least one to zoom back into it).

We are now ready to present an algorithm for traversing from s to t. If we are currently at a marked vertex, we zoom in by following a zoom edge down to the marked child. Otherwise, if we are not at a marked vertex but one is visible and within distance 3, then we pan towards the marked vertex. Otherwise, we zoom out.

Theorem 1 *In traversing from s to t, the strategy outlined above uses a number of pans and zooms that is at most:*

$$\begin{cases} \delta(s,t) & \text{if } r \geq 3 \\ \delta(s,t)+1 & \text{if } r = 2 \\ \delta(s,t)+2 & \text{if } r = 1 \\ 2\,\mathrm{h}(\mathrm{lca}(s,t)) & \text{if } r = 0 \end{cases}$$

Proof. If $r \geq 3$, then the traversal algorithm can always determine if there is a marked vertex within three pan edges, and will therefore zoom out until there is a marked vertex within 3 pan edges, pan to it, and then zoom in to t. The zooming out and zooming in is paid for by Observation 2, and the panning is paid for by Observation 1. The total length of the path is $\delta(s,t)$. However, if $r < 3$, the traversal algorithm can no longer always determine if there is a marked vertex within 3 pan edges, and so it can no longer necessarily follow a shortest path.

When $r = 2$, the algorithm will perform one additional zoom-out when there are three pan edges between the current vertex and the marked vertex. This zoom out will cause either one or two pan edges to be crossed on the next zoom level, as illustrated in Figure 3. If only one pan edge is crossed, then we will have performed two zooms and a pan, which is the same as the cost of traversing the three pan edges below. However, if two pan edges are crossed, we will have performed two zooms and two pans, which is one more than the cost of traversing the three pan edges below. Therefore, the total length of the path is at most $\delta(s,t)+1$.

Similarly, when $r = 1$, the algorithm will perform one or two additional zoom-out operations when there are three pan edges between the current vertex and the marked vertex. If after one zoom-out, the marked vertex is within distance one, then we will perform two zooms and one pan, when either two or three pans were sufficient, which is at most one more than the length of the shortest path. If two zoom-outs are needed before the marked vertex is within distance one, then we performed four zooms and one pan when three pans were sufficient, which is at most two more than the length of the shortest path. See Figure 4 for an illustration of

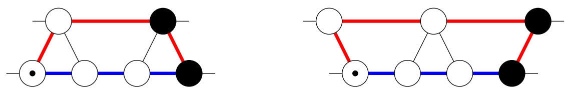
Figure 3: The possible cases when traversing $G_{n,1}$ with radius $r = 2$. The initial vertex is dotted and the marked vertices are shaded. The shortest path consists of the blue edges, while the actual path consists of red edges.

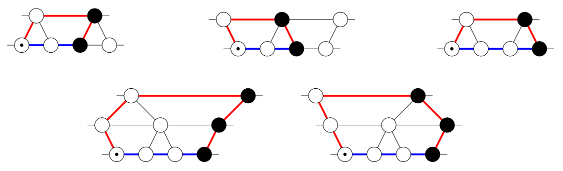
Figure 4: The possible cases when traversing $G_{n,1}$ with radius $r = 1$. The initial vertex is dotted and the marked vertices are shaded. The shortest path consists of the blue edges, while the actual path consists of red edges.

these cases. In any case, the total length of the path is at most $\delta(s,t)+2$.

Finally, if $r = 0$, the algorithm has no choice but to zoom out until it is at a marked vertex and then zoom back in to t. Therefore, $\mathrm{h}(\mathrm{lca}(s,t))$ zoom-outs are performed, followed by $\mathrm{h}(\mathrm{lca}(s,t))$ zoom-ins, for a total cost of $2\,\mathrm{h}(\mathrm{lca}(s,t))$. □

Note that $\mathrm{h}(\mathrm{lca}(s,t))$ can be large compared to $\delta(s,t)$. To see this, consider the two centermost world vertices: their lowest common ancestor is the root of the tree induced by the zoom edges, but $\delta(s,t) = 1$.

4 Searching in Two Dimensions

In this section, we study the two-dimensional case. $G_{n,2}$ is drawn as a $\sqrt{n} \times \sqrt{n}$ grid of world vertices along with smaller and smaller grids of zoom vertices for each zoom level above, as in Figure 2. We first note that the proof of Observation 1 applies equally well to the case when $d = 2$.

Observation 3 *For any two vertices u and v on the same zoom level of the graph $G_{n,2}$, if there is a path of $k \leq 3$ pan edges between u and v, then $\delta(u,v) = k$.*

Unfortunately, Observation 2 no longer holds true in two dimensions: it is possible for two vertices on the same level to have a path of 4 pan edges between them while a shortest path that involves zoom edges has length 5. See Figure 5 for an example of this.

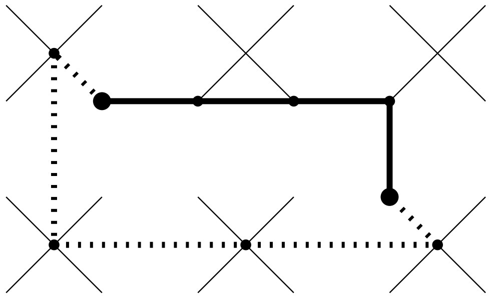

Figure 5: It is possible for two vertices (drawn heavy) on the same level to have a path of 4 pan edges (drawn heavy) between them while a shortest path that involves zoom edges (drawn dotted) has length 5.

Observation 4 *For any two vertices u and v on the same zoom level of the graph $G_{n,2}$, if there is a path of 4 pan edges between u and v, then $\delta(u,v) = \min\{4, 2 + \delta(p(u), p(v))\}$.*

Proof. If $p(u)$ and $p(v)$ are connected by a pan edge, then there is a shortest path of length 3 between u and v: zoom out from u to $p(u)$, cross the pan edge between $p(u)$ and $p(v)$, then zoom into v. As before, no shorter path is possible. Since $\delta(p(u), p(v)) = 1$, we have $\delta(u,v) = \min\{4, 2 + \delta(p(u), p(v))\}$.

Otherwise, if $p(u)$ and $p(v)$ are not connected by a pan edge, there must be at least two pan edges between them. Any path that uses zoom edges must use at least two of them, and at least two additional pan edges between $p(u)$ and $p(v)$ (or at least two additional zoom edges) must be used to produce a path of length at least 4. Since u and v are already connected by 4 pan edges, we have $\delta(u,v) = \min\{4, 2 + \delta(p(u), p(v))\}$. $\square$

It is important to note that a traversal algorithm will not be able to distinguish between these two cases without additional exploration, and so this will ultimately lead to a slightly longer path. As we shall see, however, the difference is minimal for sufficiently large r.

Observations 3 and 4 cover all cases with at most 4 pan edges separating u and v. An analogue of Observation 2 applies when there are at least 5 pan edges.

Observation 5 *For any two vertices u and v on the same zoom level of the graph $G_{n,2}$, if there is a path of $k \geq 5$ pan edges between u and v, then $\delta(u,v) = 2 + \delta(p(u), p(v))$.*

Proof. In the proof of Observation 2, we claimed there was a sequence of k_1 pan edges between u to u' and k_2 pan edges between v and v'. We now subdivide k_1 into $k_1 = k_{1,v} + k_{1,h}$ where $k_{1,v}$ counts the number of vertical pan edges (to the north or south) and $k_{1,h}$ counts the number of horizontal pan edges (to the east or west). It is straightforward to see that a path that has both north and south (east and west) pan edges can be replaced by a path with only north or only south (only east or only west) pan edges. Similarly, k_2 can be subdivided into $k_2 = k_{2,v} + k_{2,h}$.

As before, we argue that $k_1 = k_2 = 0$. To see this, observe that the path from u to u' to $p(u')$ has length $k_1 + 1$, while the path from u to $p(u)$ to $p(u')$ has length at most $1 + \lceil k_{1,v}/2 \rceil + \lceil k_{1,h}/2 \rceil \leq 1 + k_{1,v} + k_{1,h} = 1 + k_1$. Symmetrically, the path from $p(v')$ to v' to v can be replaced by a path of length at most $1 + \lceil k_{2,v}/2 \rceil + \lceil k_{2,h}/2 \rceil \leq 1 + k_{2,v} + k_{2,h} = 1 + k_2$. Therefore, as in Observation 2, a shortest path between u and v zooms out from u to $p(u)$, takes the shortest path from $p(u)$ to $p(v)$, and then zooms in to v, which has length $2 + \delta(p(u), p(v)) = \delta(u,v)$. $\square$

Applying the algorithm for the one-dimensional case to the two-dimensional case yields the following result:

Theorem 2 *In traversing from s to t, the strategy outlined in Section 3 uses a number of pans and zooms that is at most:*

$$\begin{cases} \delta(s,t) + 1 & \text{if } r \geq 3 \\ \delta(s,t) + 2 & \text{if } r = 2 \\ 2\,\mathrm{h}(\mathrm{lca}(s,t)) & \text{if } r \leq 1 \end{cases}$$

Proof. If $r \geq 3$, then the traversal algorithm can always determine if there is a marked vertex within three pan edges, and will therefore zoom out until there is a marked vertex within 3 pan edges, pan to it, and then zoom in to t. The zooming out and zooming in is paid for by Observation 5, except possibly on the last zoom level, where one pan might be replaced by two zooms by Observation 4, for a net increase of 1 in the length of the path. In all other cases, pans are fully paid for by Observations 3 and 4. The total length of the path is $\delta(s,t) + 1$.

If $r = 2$, then (as in the proof of Theorem 1), one additional zoom-out and zoom-in may be performed, but one of these is offset by saving a pan. The total length of the path is thus $\delta(s,t) + 2$, since, as in the previous case, an additional pan may be performed as a result of Observation 4.

If $r = 1$, it is possible to force the algorithm to zoom out from s to $\mathrm{lca}(s,t)$ and back to t: consider two centermost world vertices that are two pan edges apart (*i.e.*, opposite corners of the centermost world grid cell). When $r = 0$, the same is true for any two vertices s and t. $\square$

The application of Theorem 2 to the case $r = 1$ is somewhat disappointing, since it is no longer a function of $\delta(s,t)$ as it is in Theorem 1. This can be overcome by checking for a marked vertex at each one of these "diagonally adjacent" vertices, which can be accomplished by panning north, checking for a marked vertex, panning south twice, checking for a marked vertex, and panning north back to the original vertex. This uses a total of 4 additional pans. If a marked vertex is seen from any of these vertices and it is sufficiently close, pan towards it. Otherwise, return to the original vertex, zoom out, and repeat. We therefore have:

Corollary 3 *In traversing from s to t, the strategy outlined above uses a number of pans and zooms that is at most:*

$$\begin{cases} \delta(s,t)+1 & \text{if } r \geq 3 \\ \delta(s,t)+2 & \text{if } r = 2 \\ 4\,\delta(s,t)+8 & \text{if } r = 1 \\ 2\,\mathrm{h}(\mathrm{lca}(s,t)) & \text{if } r = 0 \end{cases}$$

5 Conclusion and Future Work

We have studied the problem of searching by panning and zooming in both the one- and two-dimensional cases. Our results are summarized in the following table.

r	$d=1$	$d=2$
0	$2\,\mathrm{h}(\mathrm{lca}(s,t))$	$2\,\mathrm{h}(\mathrm{lca}(s,t))$
1	$\delta(s,t)+2$	$4\,\delta(s,t)+8$
2	$\delta(s,t)+1$	$\delta(s,t)+2$
≥ 3	$\delta(s,t)$	$\delta(s,t)+1$

Whenever $r > 0$, our traversal algorithm produces a path that is within length at most 2 of the shortest path, except for the case when $r = 1$ and $d = 2$.

Returning to the motivation for this problem mentioned in Section 1, it is perhaps interesting to note that Furnas and Bederson [1] mention several strategies for panning and zooming in the context of usability. One such strategy was to zoom-out until a marked vertex is visible, pan to the marked vertex, and then zoom in. As our results show, this strategy is not too far from optimal, except that the panning should only be done when the marked vertex is sufficiently close by.

There are at least three possible directions for future research:

1. Is it possible to improve the case when $d = 2$ and $r = 1$? The length of the path produced is at most $4\,\delta(s,t)+8$, which is an outlier when compared to the other results. The reason for this is because we simulate having $r = 2$ by panning around before zooming out; can this be avoided or reduced?

2. We have only considered the cases $d = 1$ and $d = 2$. What is a good traversal algorithm for the case when $d > 2$? How do the paths produced by such an algorithm compare to a shortest path?

3. What happens in different models? For example, what if the algorithm can pan directly to any vertex that is within r pan edges? What if only the existence of a marked vertex can be determined, rather than its location? What if the distance r is measured in the L_∞ metric (instead of the L_1 metric considered here)?

References

[1] George W. Furnas and Benjamin B. Bederson. Space-scale diagrams: Understanding multiscale interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '95, pages 234–241, 1995.

[2] Michael Glueck, Tovi Grossman, and Daniel Wigdor. A model of navigation for very large data views. In *Proceedings of the 2013 Graphics Interface Conference*, GI '13, pages 9–16, 2013.

[3] Yves Guiard and Michel Beaudouin-Lafon. Target acquisition in multiscale electronic worlds. *International Journal of Human-Computer Studies*, 61(6):875–905, 2004.

[4] Scott Huddleston and Kurt Mehlhorn. A new data structure for representing sorted lists. *Acta Informatica*, 17(2):157–184, 1982.

[5] J.J. van Wijk and W.A.A. Nuij. Target acquisition in multiscale electronic worlds. *IEEE Transactions on Visualization and Computer Graphics*, 10(4):447–458, 2004.