A fast and simple heuristic for metro map path simplification

Tim Dwyer* Nathan Hurst† Damian Merrick‡
Monash University Monash University University of Sydney and National ICT Australia§

Abstract

We give a heuristic for simplifying a path defined by a given sequence of points. The heuristic seeks to fit the points with a set of line segments aligned with a limited set of possible directions. Such "schematic path simplification" has application in automatically drawing simplified metro maps from geographic data. We show that a simple version of our algorithm produces reasonable results against real-world data and runs in linear time with respect to the number of points (assuming a small fixed number of possible directions). We then give some refinements to the algorithm which may improve the quality of the results but which significantly increase the worst-case time complexity.

Keywords: Metro map layout, Optimisation.

Index Terms: I.3.3 [Computer Graphics]: Picture/Image Generation—Line and Curve Generation; I.3.3 [Computer Graphics]: Picture/Image Generation—Display Algorithms

1 Introduction

In recent years there has been considerable interest from the information visualisation and graph drawing communities in the problem of automatically generating schematic metro maps from geographic railway maps [5, 9, 10]. As well as the obvious application of dynamically generating useful maps for commuters, metro map style drawing of abstract relational network data may be useful in a number of information visualisation applications. Examples of applications that have already been explored are project plans [2], molecular pathways [4] and visualisation of abstract "trains of thought" [7].

A number of aesthetic criteria have been generally accepted by researchers in the field as useful drawing conventions for metro map layout — mostly from observation of metro maps created by graphic designers. Of particular interest for this paper is the requirement that line segments in each railway line's path should be aligned with a limited set of directions. For example, all lines might be drawn strictly parallel to the horizontal or vertical axis and 45° and 135° diagonals. This "schematic path" requirement may be treated as a constraint in the overall layout problem, that is, as part of the process of placing nodes (stations) to satisfy other criteria such as regular spacing. However, attempts at such a combined approach have met with mixed success. For example, Hong et al. [5] tried a modified force-directed approach with extra forces to align the paths. This approach gives a roughly correct solution but constraint optimisation techniques are required to obtain strictly aligned paths. Nöllenberg and Wolff [9] were able to generate high quality solutions to small metro maps using mixed-integer programming techniques. The running time of this method makes it unsuitable for interactive applications or large networks, however. Stott and Rodgers [10] also generated maps of reasonable quality, by using a hill-climbing optimiser to minimise a multicriteria objective function. This approach has similarly prohibitive running time.

Recently, Merrick and Gudmundsson [6] proposed a path simplification approach, which can either be applied directly to the geographic layout or used as a post-processing step after placing nodes roughly to improve other aesthetic criteria. A relatively simple, fast method may then be adequate for obtaining the initial layout, for example, the force-directed approach of Hong et al. or the constrained stress-majorization approach of Dwyer et al. [3]. The Merrick and Gudmundsson path simplification algorithm finds, for each path, a minimal number of linked line segments of constrained alignment such that a line segment passes within a disc of some constant radius ε around each input point, and these discs are encountered in order. The algorithm produces reasonable results and runs in $O(|C|^3 n^2)$ time, where C is the (typically small) set of directions against which lines must be aligned and n is the number of points in the input path.

Neyer [8] also presented an algorithm that minimises the number of links in a path given a restricted number of directions and an ε error bound. This algorithm runs in time $O(nk^2 \log n)$, where k is the number of links in the output. The output is a path which remains within a given distance of the original path according to the Fréchet distance metric [1]. Utilising Fréchet distance in this respect, however, can produce undesirable "zig-zags" in some paths when the set of allowed orientations is small. In addition, Neyer required ε to be chosen such that if a circle of radius ε is drawn around every input point, no two of these circles intersect.

A significant drawback to the approaches of Neyer and of Merrick and Gudmundsson is that the quality of the results is sensitive to the input choice of ε. It is possible that a value of ε that produces good results for one part of the path may not be the best choice for another part. For example, in many metro maps there may be a great deal of detail near the centre of the map, where a small ε may be appropriate, but it may be desirable to allow a larger ε—and hence greater straightening—in outlying paths. In Section 3, we present a new algorithm that tries to fit line segments close to node centres based on a least-squares regression approach. The algorithm generates good results on real-world input data and runs in worst-case time linear in the size of the input, i.e. $O(|C|n)$.

In Section 4 we present some refinements to the basic algorithm that may improve the quality of results, particularly for more chaotic input data, but which increase the worst case run-time to $O(|C| n \log n)$.

2 Definitions

A metro map is defined as a set of stations (nodes) V where each node $v \in V$ has a position in the real plane $pos(v)$, and a set of paths connecting the stations into railway lines. Every path $P \subseteq V$ is a chain of nodes. We wish to find a line to represent the path. The line for the path can have bends but each segment of the line should be straight and parallel to one of the directions given by the set of unit vectors C. A line segment s is represented by a pair of end points (s_1, s_2) and a direction vector $\vec{s}$ where $\vec{s} \cdot c = 1$ for exactly one $c \in C$. If C contains two orthogonal vectors ($|C| = 2$) then the resulting line is said to be rectilinear, four vectors at 0°, 45°, 90°

*e-mail: Tim.Dwyer@infotech.monash.edu.au
†e-mail: Nathan.Hurst@infotech.monash.edu.au
‡e-mail: dmerrick@it.usyd.edu.au

§National ICT Australia is funded through the Australian Government's Backing Australia's Ability initiative, in part through the Australian Research Council.

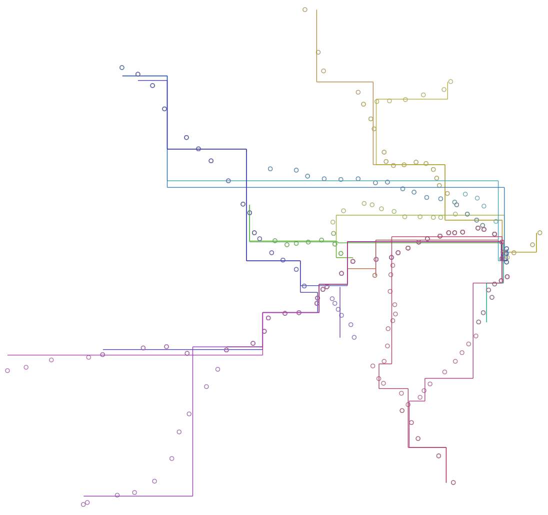
Figure 1: The Sydney Cityrail map with schematic paths produced using $|C| = 2$ and simple merging of sequential line segments that are parallel.

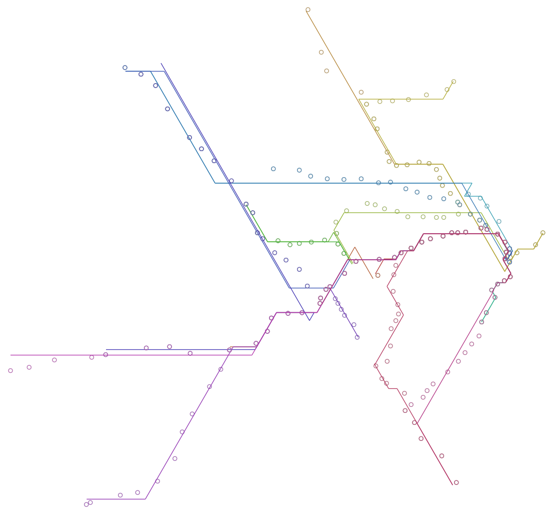
Figure 2: The Sydney Cityrail map with schematic paths produced using $|C| = 3$ and simple merging of sequential line segments that are parallel.

and 135° ($|C| = 4$) gives an octilinear line and so on[1].

Ideally, we would like the distance between each point $pos(v)$ in the path P ($v \in P$) and the line representing that path to be minimal. Preferably, the closest points on the line to each point in the path should be in the same order (encountered in the same order if the line is traversed) as the order defined over the path. In practice we do not enforce this last constraint on the assumption that node markers can be drawn in the correct order along the line regardless.

We call the process of fitting such a simplified line to a path *path schematisation*.

3 Path Schematisation Algorithm

The basic strategy of our algorithm is to partition the path into m blocks of points and use least-squares regression to find a line of best fit to the set of points in each block. A block B is a set of consecutive points in a path, and each block corresponds to a line segment fitting those points. Given three consecutive blocks B_{i-1}, B_i and B_{i+1} $(1 < i < m)$ the first point of B_i will also be the last point of B_{i-1} and the last point of B_i will be the first point of B_{i+1}. Clearly, the first point of the first block in a path B_1 and the last point of the last block B_m are not shared with any neighbour.

The line segment for a particular block $B_i (1 < i < m)$ is determined by the line of best fit for its constituent points, as described below, and the end points of the line segment are simply the intersections between the lines for B_i and B_{i-1} and B_i and $B_i + 1$. For B_1 we take the start of the line to be the closest point on the line to the first point of the path. The end of the line for B_m is deduced similarly from the last point in the path.

The algorithm begins with a partitioning of the path into blocks of two points and proceeds to merge adjacent blocks if doing so results in an improved fit of the line to the points according to some metric. The first metric we consider is the least-squares regression of a line at a fixed orientation. Using the following theorem we can test all the possible orientations for lines for a particular block in $O(|C|)$ time.

Theorem 3.1 *Given a block incorporating a set of points V, a set of simple statistics S can be computed as the block is constructed such that the cost of fitting a line of any orientation to V can be calculated in constant time.*

Specifically, the cost of fitting a line of least-squares best fit, orthogonal to the unit vector $\mathbf{n} = (n_x, n_y)$ to V is:

$$2|V|(\mathbf{b}\cdot\mathbf{n})^2 - 2(\mathbf{b}\cdot\mathbf{n})(n_x S_y + n_y S_y) + 2n_x n_y S_{xy} + {n_x}^2 S_{xx} + {n_y}^2 S_{yy}$$

where:

$$\begin{aligned} S &= (S_x, S_{xx}, S_{xy}, S_y, S_{yy}) \\ &= (\sum_{a\in V} a_x, \sum_{a\in V} a_x^2, \sum_{a\in V} a_x a_y, \text{etc}\ldots) \end{aligned}$$

and $\mathbf{b} = \frac{1}{|V|}(S_x, S_y)$.

Proof The vector $\mathbf{b}$ gives us the barycenter of V through which any line of best fit must pass. For a particular point $\mathbf{a} \in V$, the distance from that point to the line of best fit orthogonal to $\mathbf{n}$ is $|\mathbf{a}\cdot\mathbf{n} - \mathbf{b}\cdot\mathbf{n}|$. The total cost of the fit of this line for all points in V is then:

$$regression\_cost(V) = \sum_{a\in V} (\mathbf{a}\cdot\mathbf{n} - \mathbf{b}\cdot\mathbf{n})^2 \qquad (1)$$

This equation can be expanded in a straightforward manner to the cost equation of Theorem 3.1 through application of the dot-product rule for 2-dimensional vectors. ∎

Suppose two blocks B_1 and B_2 incorporating respective point sets V_1 and V_2 are to be merged into a new block B'. The line of best fit needs to be recalculated for the combined point set $V' = V_1 \bigcup V_2$. To do this, we calculate the statistics for B', which are simply the sums of the statistics for B_1 and B_2. Thus, each merge operation, including computing the statistics for the new block and the line of best fit, takes $O(|C|)$ time.

So when do we merge? We discuss more sophisticated strategies for reducing an overall cost function in Section 4, but at the very least we must merge two adjacent blocks if their lines of best fit are parallel. Otherwise, additional linkages are required to connect all line segments into a continuous path. Results of using this simple merge strategy on real-world railway data from the Sydney Cityrail network with $|C| = 2$ (rectilinear), $|C| = 3$ (hexilinear) and $|C| = 4$ (octilinear) are shown in Figures 1, 2 and 3 respectively. Figure 4

[1]Note that our definition of the set C differs from that of Merrick and Gudmundsson [6], who use $|C| = 4$ for rectilinear drawings, $|C| = 8$ for octilinear, and so on.

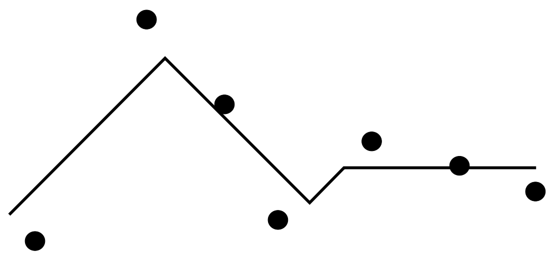

Figure 5: An example where simple merging of adjacent parallel lines leads to a poor fit

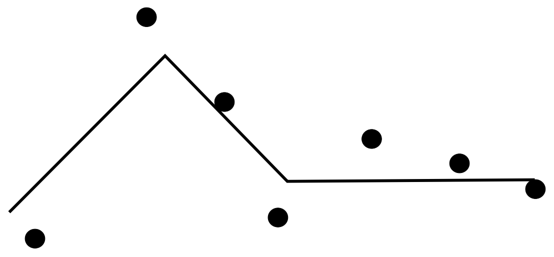

Figure 6: Merging the third and fourth line segments from Figure 5 may be considered a better schematisation.

shows a drawing of the Greater London rail map with $|C| = 4$. In these drawings, the positions of stations in the original geography are shown with small circles. The larger map, London, has 29 paths, each with up to 43 nodes; 747 nodes in total (a station has one node for each path in which it appears). Each map was schematised in less than 0.01 seconds on a standard Pentium 4 laptop.

We can eliminate all adjacent blocks with parallel lines by repeated merging in $O(|C||V|)$ time (where V is the set of vertices in the path). This is done by placing the sequence of blocks in a doubly linked list. Pairs of adjacent blocks are checked and merged if their line segments are parallel. Merging means removing the original blocks from the list and inserting the new block at the same position. Any block preceding the new block must then be checked against the new block to see if it is now parallel to the line of the new block, and so on. A backtrack (checking the preceding block) only occurs after a merge and at most $|V|$ merges are possible. Thus, the $O(|C||V|)$ time complexity is clear.

4 Refining the Algorithm

The simple strategy of merging adjacent blocks if their lines are parallel produces reasonable results, as shown by the examples. However, there are situations where a user may want more aggressive simplification. For example, Figure 5 shows an example that may be considered suboptimal. Although each of the lines is a good fit to the data points according to the cost function defined in Eq. 1, the distance between the actual end points of the line segment and the data points in a block is not considered. Thus, the third line segment (from the left) could be considered too short to actually improve the fit of the line. Figure 6 shows the result of merging the third and fourth blocks. The net distance between the points and the actual line segments is not greatly increased yet the drawing is simpler.

We want to emphasise at this point that, in practice, we have found that the simple heuristic of merging subsequent parallel line segments is sufficient for real geographical railway data. However, the block structure affords a degree of flexibility in optimising against more complicated goal functions. To address the situation described above we may define an overall cost function with, in addition to the basic regression cost for each block from Eq. 1, a term

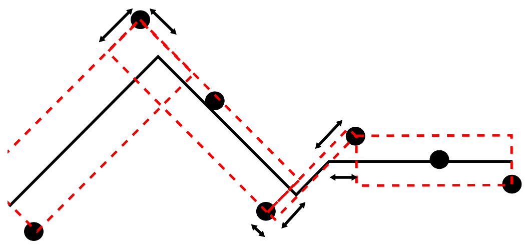

Figure 7: The path schematisation from Figure 5 shown again with bounding boxes (oriented with fit line) for the points in each block and the new error terms (shown by arrows).

to penalise solutions with more line segments, and another to penalise line segments which are not actually long enough to be close to their points. The following expression gives such a cost function for a path P with n points, partitioned into blocks $B_1, B_2, \ldots, B_m$.

$$cost(P) = \frac{\alpha m}{n} + \sum_{i=1}^{m} (\beta regression\_cost(B_i) + \gamma bb\_cost(B_i)) \quad (2)$$

The parameters α, β and γ are user defined weights that may be used to control the simplification. The $bb\_cost(B)$ function returns the squared distance between the first and last points in block B and closest edges of the bounding box around the points, aligned with the line. The error terms returned by the bb_cost function are illustrated in Figure 7.

Polynomial time optimal methods such as dynamic programming are inappropriate for minimising Eq. 2 because the problem is not decomposable into independent subproblems. The first term, which captures the total number of line segments, requires knowledge about the global situation. The bounding box term requires knowledge of the length of the line segment, which is based on the intersection points of the ends of the line with the neighbouring line segments; so we require information about the current situation of those neighbouring segments as well. Luckily, the block-merge process described in Section 3 gives us a useful framework for a local search to minimise Eq. 2.

The resulting optimisation process is simple. We place each of the $m-1$ pairs of adjacent blocks into a priority queue keyed by the change to the cost function (Eq. 2) if the blocks were merged. Using fibonnacci or pairing heaps [11], this queue can be built in linear time. We then merge the pair of blocks at the top of the priority queue (the pair for which merging would result in the most significant improvement), removing the entry in the queue. The *remove-max* operation takes $O(\log n)$ time in a fibonnacci heap. The pairs preceding and succeeding the old blocks are updated in the priority queue (using constant time *increase-key*, *decrease-key* operations). This process is repeated until no further improvement is possible. Note that pairs of blocks with parallel lines should be assigned infinite cost so that they are processed immediately.

The optimisation process resulting from this simple greedy heuristic is $O(n \log n)$ time and it should be noted that the result is by no means optimal. Further refinement is possible by searching blocks for potential split points. That is, points across which the block may be split into two new blocks in order to further reduce the total cost. Such a search would probably take at least time linear in the number of points in the block and there is no way to know how many splits would be required in order to converge to a local minimum.

5 Conclusion and Further Work

In Figures 1 to 4, no effort was made to force sections of different paths that share common points to be in any way aligned. A clear

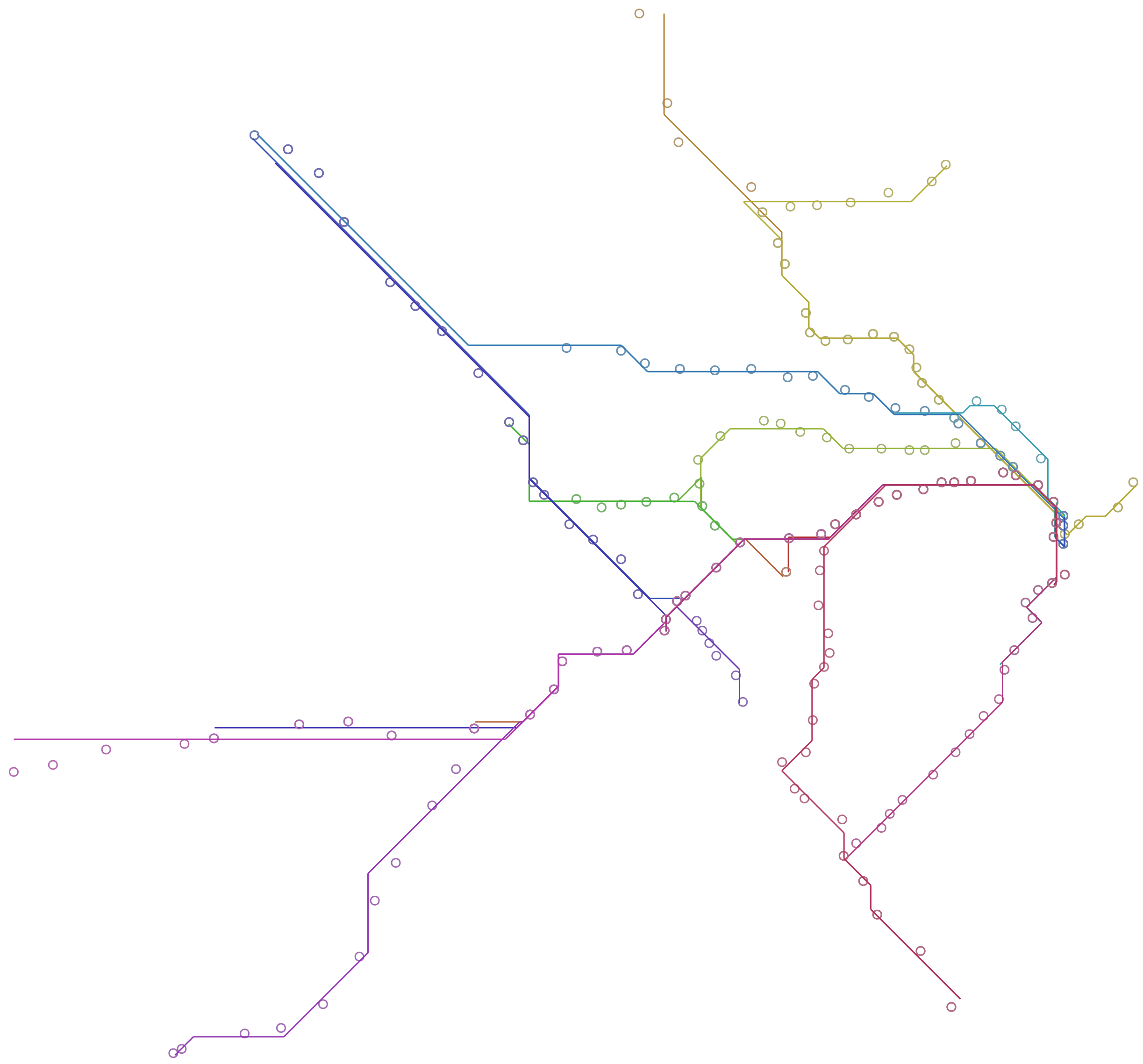

Figure 3: The Sydney Cityrail map with schematic paths produced using $|C| = 4$ and simple merging of sequential line segments that are parallel. The original geographic positions of stations are shown with small circles.

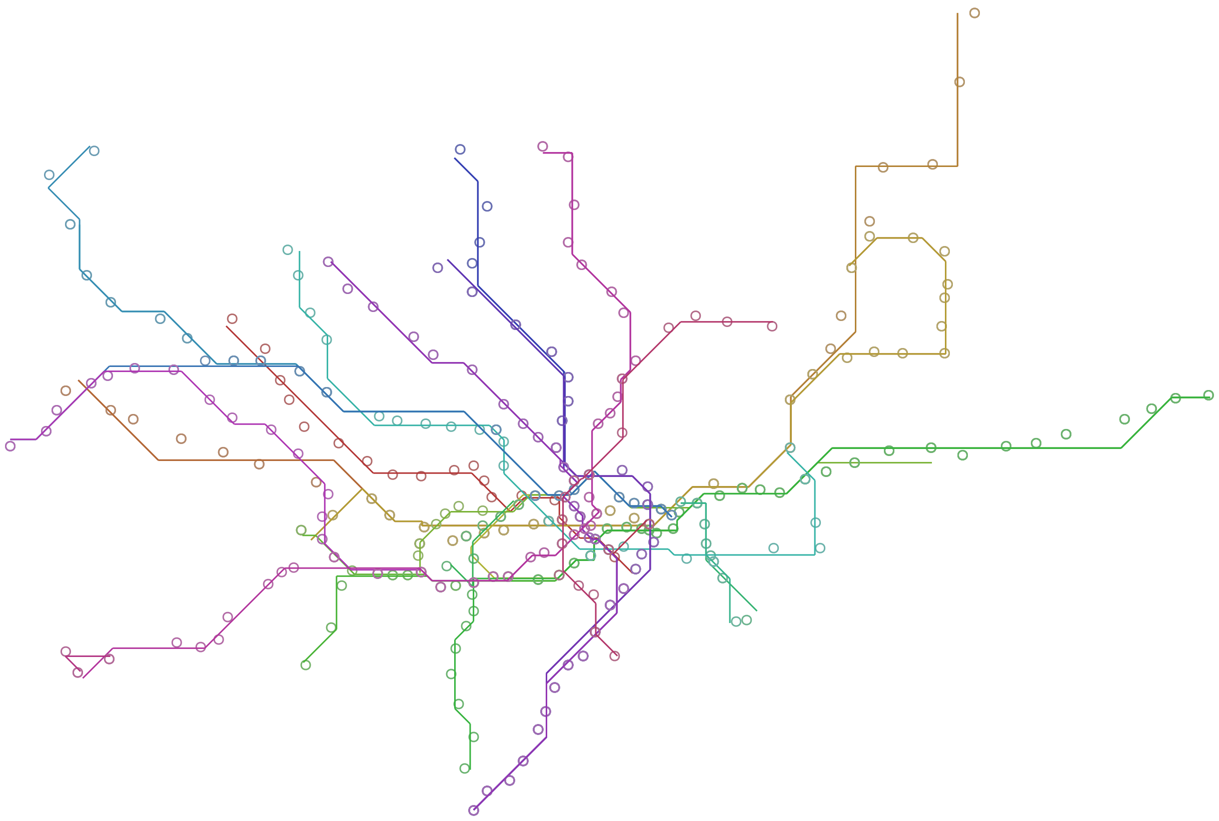

Figure 4: The Greater London rail map with schematic paths produced using $|C| = 4$ and simple merging of sequential line segments that are parallel. The map has 29 paths, each with up to 43 nodes; 747 nodes in total (a station has one node for each path in which it appears). It took less than 0.01 seconds to schematise the map on a standard Pentium 4 laptop.

improvement for these examples would be to somehow share the block structures between paths. However, it is not entirely clear what is required in this case. At the very least there should probably be some guarantee of topology preservation. That is, paths which do not cross in the geographic map should not be allowed to cross in the simplified map.

Clearly, the refinements suggested in the previous section should be implemented and tested in some empirical evaluation. Also, the algorithms outlined in this paper should be tested against the bounded error approach of Merrick and Gudmundsson. However, it is not clear at this stage exactly what constitutes a "good" path simplification. In this paper we have shown that altering the definition of quality of fit is one way to create simpler and faster algorithms. Perhaps before any in-depth empirical evaluation is conducted some more concrete requirements need to be harvested from experts in graphic design, or even from users of metro maps.

REFERENCES

[1] H. Alt and M. Godau. Computing the fréchet distance between two polygonal curves. *International Journal of Computational Geometry and Applications*, 5:75–91, 1995.

[2] R. Burkhard, M. Meier, P. Rodgers, M. Smis, and J. Stott. Knowledge visualization: a comparative study between project tube maps and gantt charts. In K. Tochtermann and H. Maurer, editors, *Proceedings of the 5th International Conference on Knowledge Management*, pages 388–395. Know-Center, Austria, June 2005.

[3] T. Dwyer, Y. Koren, and K. Marriott. Stress majorization with orthogonal ordering constraints. In *Proceedings of the 13th International Symposium on Graph Drawing (GD'05)*, volume 3843 of *LNCS*, pages 141–152. Springer, 2006.

[4] W. C. Hahn and R. A. Weinberg. A subway map of cancer pathways. *Nature Reviews Cancer*, 2(5), May 2002.

[5] S. H. Hong, D. Merrick, and H. A. D. do Nascimento. The metro map layout problem. In *Proceedings of the 12th International Symposium on Graph Drawing (GD'04)*, volume 2912 of *LNCS*, pages 482–491. Springer, 2005.

[6] D. Merrick and J. Gudmunsson. Path simplification for metro map layout. In *Proceedings of the 14th International Symposium on Graph Drawing (GD'06)*, LNCS. Springer, (to appear) 2006.

[7] K. V. Nesbitt. Getting to more abstract places using the metro map metaphor. In *Proceedings of the 8th International Conference on Information Visualisation*, pages 488–493, July 2004.

[8] G. Neyer. Line simplification with restricted orientations. In *Proceedings of the 6th International Workshop on Algorithms and Data Structures*, pages 13–24, 1999.

[9] M. Nöllenburg and A. Wolff. A mixed-integer program for drawing high-quality metro maps. In *Proceedings of the 13th International Symposium on Graph Drawing (GD'05)*, volume 3843 of *LNCS*, pages 321–333. Springer, 2005.

[10] J. Stott and P. Rodgers. Metro map layout using multicriteria optimization. In *Proceedings of the 8th International Conference on Information Visualisation (IV'04)*, pages 355–362. IEEE Computer Society, 2004.

[11] M. A. Weiss. *Data Structures and Algorithm Analysis in Java*. Addison Wesley Longman, 1999.