Fast and readable layered network visualizations using large neighborhood search

Connor Wilson, Tarik Crnovrsanin, Eduardo Puerta, and Cody Dunne

Abstract—Layered network visualizations assign each node to one of several parallel axes. They can convey sequence or flow data, hierarchies, or multiple data classes, but edge crossings and long edges often impair readability. Layout algorithms can reduce edge crossings and shorten edges using quick heuristics or optimal methods that prioritize human readability over computation speed. This work uses an optimization metaheuristic to provide the best of both worlds: high-quality layouts within a predetermined execution time. Our adaptation of the large neighborhood search metaheuristic repeatedly selects fixed-sized subgraphs to lay out optimally. We conducted a computational evaluation using 450 synthetic networks to compare five ways of selecting candidate nodes, four ways of selecting their neighboring subgraph, and three criteria for determining subgraph size. Large neighborhood search generally halved the number of crossings vs. the barycentric heuristic while maintaining a reasonable runtime. Our best approach randomly selected candidate nodes, used degree centrality to pick cluster-like neighborhoods, and chose smaller neighborhoods that could be optimally laid out in 0.6 or 1.2 seconds (vs. 6 seconds). In a case study visualizing 13 control flow graphs, most with over 1000 nodes, we show that our method can be employed to create visualizations with fewer crossings than Tabu Search, another metaheuristic, and vastly outperforms an ILP solver on meaningfully large graphs. A free copy of this paper and all supplemental materials are available at https://osf.io/w3fev/.

Index Terms—Network visualization, graph drawing, layered network, optimization, heuristic, metaheuristic, computational experiment

1 Introduction

Network visualizations are widely used to explore and explain relational data. This data consists of a set of nodes representing entities and a set of edges denoting the connections between them. In node-link network visualizations, the nodes are drawn with marks such as circles, and the edges are drawn as straight or curved lines joining them.

This paper focuses on a subclass of node-link visualizations called layered networks. In these networks, each node is assigned to a layer, for example, to show a hierarchy or a sequence of events over time. A layered network visualization generally uses parallel lines to represent the layers, and nodes within a layer are positioned along their assigned line [36].

Creating highly readable layered network visualizations is critical for user performance in several domains, including medicine [2], [56] and sociology [5], [62]; another more specific example is the need for readable schematic diagrams for industry machinery, which may be too complicated to create by hand but need to be easily understandable by maintenance technicians [76].

We will use reverse engineering as a case study to motivate and evaluate this work. Control flow graphs created by reverse engineering compiled code are often visualized as layered networks [12]. Reverse engineers use these visualizations for several crucial tasks, including understanding the structure of binary code and the optimizations performed by the compiler [9], [11], which benefit from higher-quality visualizations [13].

User task performance with node-link visualizations is highly dependent on the spatial layout of the nodes. For example, the tasks of finding the shortest paths between two nodes or finding the minimal set of nodes that disconnect two nodes when removed [45], [57], [70]. We can partially quantify a spatial layout's impact on user task performance using aesthetic or readability criteria such as the number of edge crossings [58]. Other criteria we know affect readability include the bendiness of edges and the angle formed by crossing edges [16].

Many layered network layout algorithms have been proposed to improve readability directly using optimization models or indirectly with layout heuristics. All algorithms for this task present a tradeoff between computation speed and scalability to large networks. On one hand, heuristics such as the popular Sugiyama [64] and *dot* [26] frameworks are fast but suboptimal. On the other, algorithms that find optimal solutions produce perfect layouts but may take a long time to compute. Models minimizing edge crossings in particular have existed since 1997 [41]. These have gained attention in recent years due to increasing computational power and advancements in solver software, making crossing-optimal layouts feasible for non-trivially sized networks [14]. Researchers have proposed models combining this with additional criteria such as edge length reduction [14], restrictions on bends in long edges [30], and maximizing the size of the planar subgraph [24].

However, large networks remain challenging, especially for approaches that stack together multiple aesthetic criteria. Additionally, the exponential runtime of the optimization algorithm causes these approaches to fail to find high-quality solutions along the way [28]. In this sense, these techniques are "boom or bust" in that they either find the optimal solution quickly or hang indefinitely, failing to even

All authors are with Northeastern University
E-mail: [wilson.conn | t.crnovrsanin | puerta.e | c.dunne]@northeastern.edu

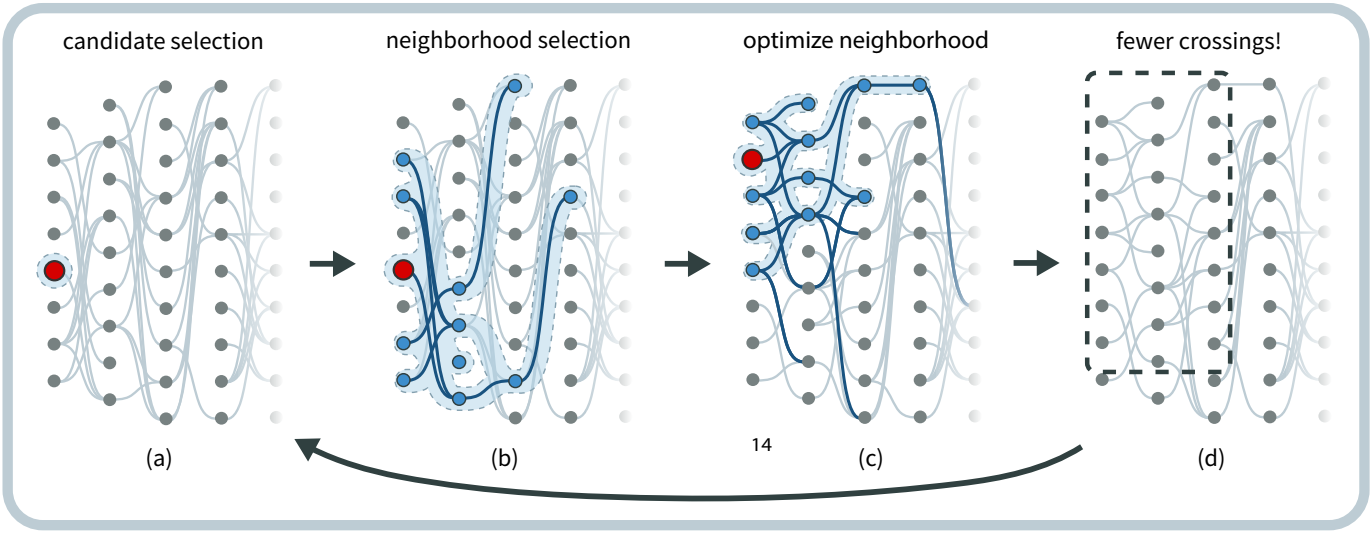


Fig. 1. We adapt *Large Neighborhood Search* to the layered network layout problem by executing the following steps. First, in (a), a candidate node is selected. This candidate is then used as a root node to collect a subgraph of nearby nodes and edges in (b). The vertical positions of the nodes within their layer are rearranged in (c) by an exact optimization model, which can be calculated quickly for the small subgraph selected. In (d), the result is an optimal solution for this specific subgraph according to the optimization objective, which, in this case, is edge crossing minimization. This process is repeated from (a) as many times as possible within a predefined time limit, varying the selected candidate and neighborhood. For an animated version of this process, click here.

find a valid layout for larger networks when one could have been determined by heuristics in mere milliseconds [30]. Compounding the problem further is the difficulty of estimating the time it will take to optimize one of these models, often making global optimization techniques unreliable.

In this work, we analyze how a known metaheuristic approach for optimization models can be applied to the problem of creating layouts for layered network visualizations. Our goal is to provide the best of both worlds: more readable network visualizations than those produced by layout heuristics while keeping execution time within predetermined bounds. The metaheuristic we use is called Large Neighborhood Search [61], popular in other domains and well-suited to layered network layouts. The intuition behind our approach (fig. 2) is that subgraphs can be quickly optimized many times, even in the context of the entire network, in cases when the network is too large for optimization models to handle in a reasonable time—or at all! We have several open questions to address to apply large neighborhood search principles to this problem. Most importantly, how do you select neighborhoods (which we interpret as subgraphs of the input network) to optimize? Specifically, how do you choose a candidate node, and how do you then find a local neighborhood surrounding it? Also, how do you determine the neighborhood size to use, and which variables in the model to optimize?

Specifically, this paper contributes:

1) A useful framework for applying Large Neighborhood Search to the problem of creating layered network layouts,
2) The results of a computational evaluation showing the effect on readability and computation time of five candidate selection methods, four neighborhood selection methods, three neighborhood sizes, and three restrictions on neighborhood drawing regions,
3) An improved network size criteria for better estimating the runtime of solving the ILP model for layout optimization,
4) A case study justifying and demonstrating the use of Large Neighborhood Search for creating high-quality diagrams of large software control flow networks, and
5) Free and open-source code and data for all our techniques and evaluations, available at https://osf.io/w3fev/.

2 Background and Related Work

In this section, we discuss heuristic and optimal approaches to compute network layouts as our work explores the trade-off between them. Similarly, we also provide background to LNS in optimization literature and similar metaheuristics applied to this problem.

The domains utilizing layered network visualization are too numerous to list here, spanning control flow diagrams in reverse engineering [11], machine learning model explanation [47], [72], and schematic diagrams for industrial machinery [76]. These visualizations are broadly used for Sankey diagrams [75], timeline-based data [10], [33], and hierarchical data such as trees or even more broadly any network that can be classified as directed [25].

2.1 Layered Network Visualization

A *layered network* or *layered graph* G is comprised of a set of vertices or nodes V and edges E, and a layering function L. For a given network with K layers, the *layering* $L : V \rightarrow \{1, 2, 3, \ldots, K\}$ is an assignment of nodes to layer values. Each edge connects two nodes in adjacent layers, meaning the layer value of the two nodes differs by one. Layered networks with longer edges can be modified by adding a dummy node to each layer spanned by the

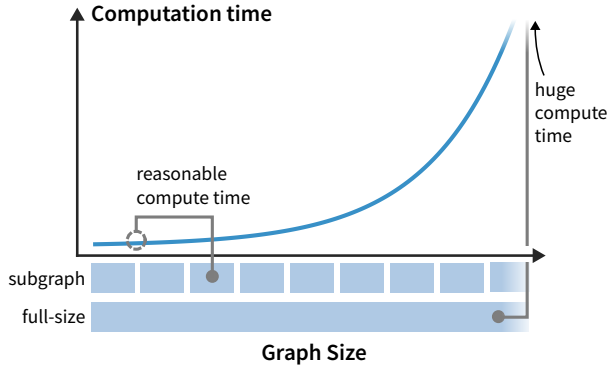


Fig. 2. An illustration of the intuition for using large neighborhood search. Optimizing the model globally is an exponential-time operation, but we can quickly find a high-quality solution by breaking the input down into smaller independent subgraph optimizations performed in sequence.

TABLE 1
Notation used in this paper

$G = (V, E, L)$	A *layered network* G with vertex set V, edge set E, and *layer assignment* on the vertices $L : V \to \{1, 2, \ldots, K\}$.
V_r	All $u \in V$ such that $L(u) = r$.
E_r	All $(u, v) \in E$ such that $L(u) = r$ and $L(v) = r + 1$.
K	Number of layers in G.
$x_{i,j}$	Binary variable denoting the relative position of nodes i and j in the layout.
$c_{(i,k),(j,l)}$	Binary variable denoting if edges (i, k) and (j, l) cross.
D	Drawing function $D : V \to \{1, 2, \ldots\}$. Each node u is drawn in the plane at coordinate $\big(L(u), D(u)\big)$, and edges are drawn as straight lines connecting the nodes.

edge, a process described by Gansner et al. [25]. This is necessary to apply the optimization models discussed in this work. Same-layer edges can be removed by a node promotion heuristic [54] or by including constraints to allow for this [14]. We then consider a *layered network visualization* (LNV), or *drawing*, to be an assignment of nodes to (x, y) coordinates, such that nodes in the same layers are aligned along parallel lines that are ordered consecutively. Hence, assuming the layer assignment is fixed, the problem addressed by layered network visualization is to find a vertical placement of the nodes in each layer that optimizes for some aesthetic criteria, such as edge crossings. All layouts pictured in this paper use vertically oriented layer lines.

Standard heuristics for LNV. Purchase et al. [57] find that crossing reduction is the most important overall metric for graph readability and that edge length reduction is most important for path-related tasks [70]. Therefore, many heuristics for layered network visualization have been proposed to minimize edge crossings in layered node-link diagrams. Among the most popular are the 1981 barycenter heuristic of Sugiyama et al. [64], and its counterpart the median heuristic of Eades and Wormald [20]. The barycenter heuristic places each vertex at the mean index of its neighbors, while the median heuristic uses the median. These heuristics are defined for 2-layer networks but extend to general layered networks by "sweeping" back and forth, applying the heuristic repeatedly.

Many other heuristics originally designed for the bipartite crossing problem have been extended to the k-layered case via this iterated layer sweep scheme. Eades and Kelly [17] propose the insertion heuristic, switching heuristic, and split heuristic. In a later paper, Eades proposes another heuristic, the degree-weighted barycenter [19]. The weighted median heuristic of Gansner et al. [25] refines the original median heuristic with a few additional techniques, and is incorporated into the popular *dot* algorithm (available through the Graphviz software) for drawing directed graphs [26]. Matuszewski et al. introduced the global sifting heuristic [50], one of the few simple layout heuristics to not use the layer sweep approach, which iteratively sifts vertices in decreasing order by node degree. These techniques all find solutions extremely quickly and are thus well-suited to large inputs, but they are bounded in terms of the quality of the solution. For a more comprehensive background, we refer the reader to Tamassia's book on graph drawing [65], of which chapter 13 deals with crossing minimization for layered networks [36].

Other techniques such as Parallel Edge Splatting [7] address LNV-adjacent problems within dynamic or temporal graph visualization. In particular, Linhares et al. [46] propose a community detection technique for temporal visualization similar to our neighborhood methods. We refer the reader to [51] and [3] for a broader overview of dynamic and multilayer network visualization.

Optimal LNV. Jünger et al. introduced the powerful optimal crossing minimization technique [41], though its performance is exponential. In fact, any method designed to find the edge crossing-optimal layout is exponential, unless P=NP [28]. The formulation relies on representing the input networks as an integer linear programming (ILP) problem, which optimizes a set of variables and linear constraints to determine the best possible layout. It is often used even within the space of layered network heuristics because it performs quickly in practice for smaller networks and can provide a baseline to compare results to, as empirically established by Jünger and Mutzel [42].

Other works have adapted this original formulation to account for additional readability criteria. The layout technique of Di Bartolomeo et al. [14] combines crossing minimization and edge length reduction, allowing users to specify the desired level of tradeoff between optimizing for crossings and optimizing for edge bends. This utilizes a model provided by Gansner et al. [25] for the final node placement once a node ordering has been determined. Zarate et al. [75] provide grouping constraints on nodes to allow for crossing area-minimal layouts of Sankey diagrams. Glover et al. [30] apply constraints to long edges to guarantee they are drawn without bends. Finally, Gange et al. [24] combine crossing minimization with a model for the maximal planar subgraph problem.

2.2 Optimization Metaheuristics

Many commercial solvers can efficiently optimize a wide array of models [35], [40], and research on model optimiza-

tion has progressed by leaps and bounds through the years [29], [52]. However, Raidl et al. [59] describe the divide in the literature between this work on global optimization and the study of metaheuristics. Metaheuristics are generalized strategies for finding high-quality solutions to optimization problems [4]. In this paper, we focus on **Large Neighborhood Search** (LNS), which was first described by Shaw [61] for the vehicle routing problem. LNS refers to a broad category of techniques that iteratively improve an initial solution, examining large, often exponentially-sized neighborhoods of candidate solutions at each iteration [4]. It has been applied to several hard optimization problems, including the pickup and delivery problem [60] and has been generalized to any ILP [63]. Ropke et al. [60] apply LNS to a transportation routing problem but use a heuristic to search the neighborhoods, while our approach uses an optimal solution as in [61], [63]. In their survey, Ahuja et al. [1] show that the benefits of Very Large-Scale Neighborhood Search (a subcategorization of LNS) are often dependent on problem-specific knowledge informing the neighborhood selection process. We apply this recommendation to layered network layouts, following Raidl's insight that there is much potential progress to be made with hybrid approaches integrating ILP into the neighborhood exploration process [59].

Controlling optimization time. One potential issue with globally optimizing ILP models is the difficulty of predicting their computation time—their exponential nature means there may not be large differences between networks that take one minute and networks that take one month to optimize. Hurley et al. [38] find that solver-inherent randomization significantly affects runtime, showing that the top 3 solvers for the 2014 SAT Competition could have ranked in any order. Di Bartolomeo et al. [14] provide results for optimal crossing minimization displaying variation spanning orders of magnitude when using the number of nodes in the network as a predictor for runtime.

That said, predicting the runtime of a solver has had a lot of attention in the literature because of the need for solver portfolios that can decide which optimization algorithm will perform best for a given problem [48]. Machine learning-based solutions [69] and statistical methods [39] have been shown to be effective for this task. In our work, we do not require a perfect predictor, but we do incorporate these guidelines to standardize runtime for instances of specific sizes.

Existing metaheuristic approaches for LNV. The most similar work to our own is a series of papers applying **Tabu Search**, a different optimization metaheuristic, to the layered crossing minimization problem. Tabu search, introduced by Glover et al. [31], was first adapted to layered network visualization by Laguna et al. [44]. The metaheuristic's principle is to maintain a queue of recently explored solutions (the "taboo" list) to prevent the process from cycling on the same solutions. More recently, Cavero et al. [8] combine the Tabu search framework with a strategic oscillation technique, improving the performance of layered crossing minimization. Other recent works have adapted the approach to adjacent problems, such as the incremental graph drawing problem [53], [55], which bear some similarity to our large neighborhood approach. van den Elzen et al. [67] use **simulated annealing**, a metaheuristic, to reorder nodes for dynamic graph visualization. Another metaheuristic, **GRASP**, has been studied for network layouts in the bipartite case [43]. However, Tabu search, GRASP, and simulated annealing differ from LNS in that they focus on a small number of local moves.

Our work in this paper takes a different approach to optimizing the layered network layout, with a much broader scope and more extensive use of the ILP formulation.

3 Methodology

In the following section, we describe our adaptation of Large Neighborhood Search to the layered graph drawing problem, starting with a standard optimization model and then focusing on how we define and optimize neighborhood subsets of the input network.

3.1 ILP Formulation

We first present the ILP formulation of Jünger et al. [41] for crossing minimization on layered networks. While formulations that incorporate other aesthetic criteria have been proposed [14], [24], Jünger et al.'s constraints are typically still included in some form due to the importance of crossing minimization with respect to readability [57]. Hence, though LNS and our application of it are optimization model-independent, we perform our evaluation using this standard model:

$$
\begin{aligned}
&\text{minimize} \sum c_{(u_1,v_1),(u_2,v_2)} \quad \text{subject to:} \\
&x_{u_1,u_2} + x_{v_2,v_1} + c_{(u_1,v_1),(u_2,v_2)} \geq 1 \\
&x_{u_2,u_1} + x_{v_1,v_2} + c_{(u_1,v_1),(u_2,v_2)} \geq 1 \\
&\qquad\qquad \forall i,\ \ \forall (u_1,v_1),(u_2,v_2) \in E_i \\
&x_{u_1,u_2} + x_{u_2,u_3} - x_{u_1,u_3} \leq 1 \\
&x_{u_1,u_2} + x_{u_2,u_3} - x_{u_1,u_3} \geq 0 \\
&\qquad\qquad \forall i,\ \ \forall u_1,u_2,u_3 \in V_i \\
&x_{u_1,u_2},\ c_{(u_1,v_1),(u_2,v_2)} \in \{0,1\}
\end{aligned}
\tag{1}
$$

See table 1 for definitions of these quantities. For a more thorough explanation of each of the inequalities, see Wilson et al. [71].

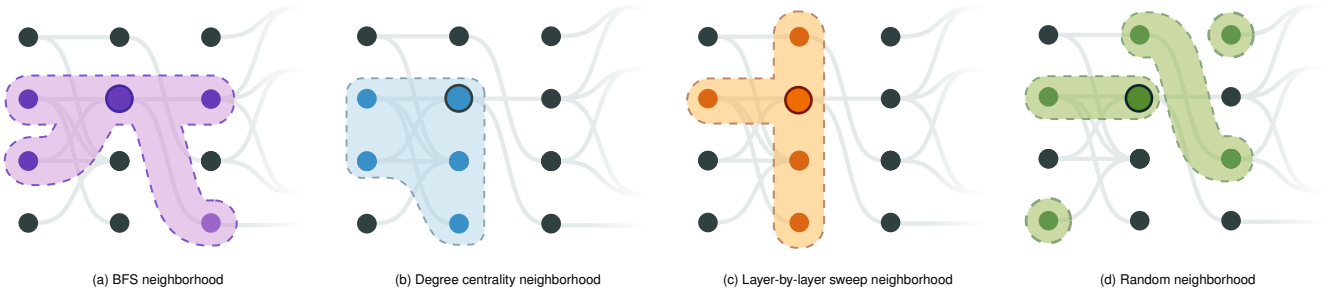
3.2 Large Neighborhood Search

Below we provide a high-level overview of our adaptation of LNS for an input layered network G, an initial layout of its nodes, and a predetermined time limit:

1) Select a subgraph H_i of G
2) Fix in place all positional variables x_{u_1,u_2} in the ILP model of eq. (1) unless $u_1 \in H_i$ or $u_2 \in H_i$
3) Optimize the model, updating node positions accordingly
4) Repeat from step 1 until time runs out

3.2.1 *Initial layout*

LNS, as an iterative technique, requires an initial solution to improve on. Since it examines much larger neighborhoods than other optimization metaheuristics, LNS is more resilient to poor initial layouts [1]. For the sake of standardization, we utilize the iterated barycenter heuristic of



(a) BFS neighborhood (b) Degree centrality neighborhood (c) Layer-by-layer sweep neighborhood (d) Random neighborhood

Fig. 3. A visual example of the differences between the four neighborhood selection techniques for the same network layout and candidate node. The candidate node is larger and outlined, and the neighborhood consists of the nodes bounded by the dashed lines. The edges included in the dotted outline influence the size calculation in eq. (3).

Sugiyama et al. [64] for the initial layout, as it is quick to compute and is perhaps the most widely known and used heuristic for layered network visualization.

3.2.2 Fixing variables

Fixing and unfixing portions of the model are integral to the framework. A major strength of LNS is that the model needs only be loaded once, a computationally expensive operation, while fixing and unfixing variables is very cheap [63]. Fixing a variable is achieved by setting the upper and lower bound on the variable to the same number.

Note that we need only fix and unfix the x-variables (the decision variables of the model) since values the c-variables in eq. (1) take are fully constrained relative to the x-variables.

3.2.3 Iterative improvement

As LNS is an improvement heuristic, we always fix x-variables to maintain the current layout D. x_{u_1,u_2} controls the relative position of u_1 and u_2 in the drawing, so fix $x_{u_1,u_2} = 1$ if $D(u_1) > D(u_2)$ for the current layout and $x_{u_1,u_2} = 0$ otherwise.

The optimization subproblem then considers all constraints in the base model while solving a reduced subset of the model's decision variables. Hence, the optimization model will never backtrack or otherwise decrease the value of the overall objective function. That way, we achieve perfect *intensification*—an important notion that says optimization improvement heuristics ought to use the current best-found solution to inform its search for new solutions [4]. This is another strength of LNS—by examining a large subproblem and solving to optimality, we are more likely to avoid falling into a local minimum, which is a problem for local search techniques [1].

3.3 Candidate Selection

We hypothesize that the solver will have more difficulty untangling the edge crossings in certain areas of the network than others, and these areas should be prioritized. Selecting a candidate node to use as the root node for neighborhood selection is an important consideration when given a fixed, short amount of time to find a high-quality solution. We examine five methods for selecting this candidate.

- **Degree Centrality Candidate.** Networks with high edge density are more challenging for solvers and cause large increases in the number of crossings in the layout when positioned suboptimally. Focusing on nodes with the greatest degree centrality may yield subproblems with large potential for improvement. Additionally, Matuszewski et al. [50] find that their layered graph layout technique, global sifting, is greatly improved when nodes are sifted in order by degree.
- **Betweenness Centrality Candidate.** Betweenness centrality is a widely used technique for scoring how important each node is to the overall network structure [22], which may refine the algorithm's focus to subgraphs with excess edge crossings.
- **Edge Crossings Candidate.** Instead of using centrality metrics as a proxy for improvement potential, we can simply use the number of crossings on edges adjacent to a node. Nodes with many adjacent crossings have a higher potential for moves, which decreases the crossing number.
- **Edge Length Candidate.** Similarly, we can use the average length of edges adjacent to a node. Nodes with very long adjacent edges may need to be repositioned since long edges often cross over many other edges.
- **Random Candidate.** As a candidate selection baseline method, we also study the effect of uniformly randomly choosing a candidate node.

3.3.1 Penalty function

To ensure the same sections of the network are not repeatedly optimized, we apply a penalty function on the candidate score after performing each optimization. This is necessary to achieve global diversification, a crucial quality for optimization heuristics [4].

We loosely borrow the idea of a penalty on node movement from work on incremental, force-directed graph visualization [23], [32]. Each node is initially assigned a numerical candidate score s based on the candidate method. Then for every node in the neighborhood, a penalty w_n is applied to the candidate score, and w_c if the node was the selected candidate for that iteration. If the node swapped positions with another node during the optimization step, the penalty is halved. The node's score s is then multiplied by the reciprocal of the penalty, $s \cdot \frac{1}{w_c}$, and the node chosen next iteration will be the one with the new highest candidate score. In our implementation we use $w_c = 8$ and $w_n = 4$.

For the **crossings candidate** and **edge length candidate** methods, the candidate score changes each iteration. Hence, we do not apply the penalties as just described, but instead recalculate the score every iteration and implement a no-repeats policy on the candidate node selection. We also lift the no-repeat limitation on a node if it or one of its neighbors swapped positions in the previous iteration.

3.4 Neighborhood Selection

As noted previously, our LNS adaptation requires at each step a set of variables in the model to optimize, keeping all others fixed. In this work, we explore four methods of aggregating these neighborhoods given some candidate node.

Breadth-First Search (fig. 3a). BFS is a typical algorithm for searching a graph, and is standard for collecting the nodes nearby to a root node. Given some candidate, we use BFS to branch outwards, adding neighboring nodes to the selected subgraph.

Degree Centrality Search (fig. 3b). Based on our hypothesis that clusters of connected nodes should be prioritized, we also utilize a simple heuristic to collect highly connected subgraphs. Cavero et al. [8] describe this technique for the layered graph optimization problem. We use their approach, but tweak the criteria slightly to avoid having to train parameters—at each iteration, add to the subgraph the node u with the highest ratio $\frac{d_{in}(u)}{d_{out}(u)+1}$, where $d_{in}(u)$ is the number of edges connecting u to a node in the subgraph, and $d_{out}(u)$ is the number of edges from u to nodes not in the subgraph. At each iteration, the process considers all nodes connected to the currently selected subgraph.

Layer-by-Layer Search (fig. 3c). The majority of standard heuristics for LNV utilize a layer-by-layer sweep approach wherein a single layer's nodes are reordered, and this process is repeated for increasing and decreasing layer numbers. We use a modification of this sweeping technique, following the same general principle. At each layer, starting from the candidate's, all nodes are added before moving on to the next layer. Nodes from the current layer are selected in the order of highest in-degree calculation as described above, $\frac{d_{in}(u)}{d_{out}(u)+1}$. After adding all nodes, the next layer is chosen at random from the two adjacent layers on the left or right.

Random Search (fig. 3d). As a baseline method, we also consider the process of uniformly selecting random nodes from the network to form a subgraph.

3.4.1 Controlling neighborhood size

We hypothesize that the size of the optimized subproblems will have a large effect on the quality of the final layout. A main challenge of this work is controlling the size of these subproblems, which is necessary in order to compare the neighborhood aggregation and candidate selection techniques. Since the stated goal of our work is to find techniques which produce the highest-quality solutions within a fixed, user-specified period of time, we use solver runtime as a metric to represent subproblem size. Then, we need to control the average number of optimizations performed per a unit amount of time. This requires predicting the runtime of a given ILP model, however—a tricky challenge in optimization research [38], [39].

While previous work in network layout optimization has used the number of nodes in the network as an independent variable for ILP runtime [14], we find that the number of crossing constraints in the model is a better indicator of optimization time:

$$|C| = 2 \cdot \sum_{r=1}^{K-1} \frac{|E_r|\big(|E_r|-1\big)}{2} \tag{2}$$

Previous experimentation, detailed in section , shows this to be more consistent predictor of runtime than counting the number of nodes in the network. Even with this improved metric, controlling the runtime of optimization across the different neighborhood techniques remains challenging.

However, eq. (2) needs to be adapted to our neighborhood selection techniques, since calculating eq. (2) for only the selected subgraph would ignore the other constraints with edges not in the subgraph. These account for the majority of constraints in the model when the subgraph is small and increase the optimization time in practice. Hence, when the next node u is selected for the current neigh, we instead increment a counter by the number of additional crossing variables associated with u, $Inc(u, H_i)$, given by the following calculation:

$$Inc(u, H_i) = \sum_{r \in \{L(u)-1, L(u)\}} \sum_{(u,v) \in E_r : v \in H_i} 2\,|E_r \setminus H_i| \tag{3}$$

where $E_r \setminus H_i$ is the set of edges in E_r that are not already in the currently selected subgraph H_i. This has the effect of counting the number of crossing constraints in the model that could be violated when the subgraph nodes are repositioned, in line with our findings on runtime prediction. The value of the counter also approaches eq. (2) as H_i grows towards the size of the full network. We repeat this process until the counter reaches a predetermined threshold, at which point the process is halted and the selected subgraph H_i is returned.

Note that eq. (3) increments only for edges with both endpoints in the selected subgraph to prevent double-counting. This also means that the random neighborhood method (fig. 3d) will always select at least one connected component. In practice, the technique selects more nodes than the others, however, because it selects many mostly disconnected nodes that do not cause the counter to increment much.

3.4.2 Neighborhoods with boundaries

Up until this point, for a given subgraph H we have unfixed and optimized all variables controlling to the position of

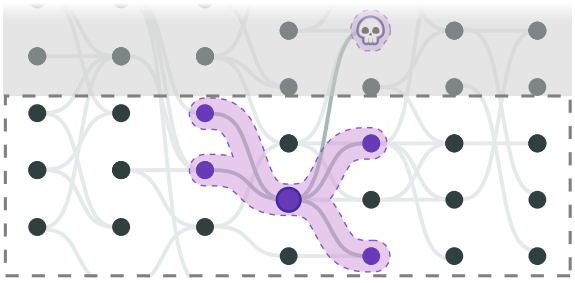


Fig. 4. The process of selecting a neighborhood when boundaries are involved (indicated by the dashed rectangle window). Nodes outside the boundary region (in the grayed-out area) cannot be added to the neighborhood subgraph, and their respective ILP variables are also kept fixed. Therefore, optimizing the nodes' position is also restricted to this region.

nodes in H. While intuitive, there is no particular reason why we must unfix *all* of the variables. Due to the constraints in the model (eq. (1)), contiguous sections of nodes in a layer can all swap positions when unfixed, but two islands of nodes separated by a fixed node cannot swap between the two islands. Hence, we can reduce the size of the region in which the unfixed nodes are able to swap so long as the region is contiguous. Since the variables in the model scales polynomially with the number of nodes in a layer, reducing this swappable area may improve performance.

Now, we describe how to isolate a contiguous region of the network around a selected candidate—see fig. 4 for visual reference. Given some candidate node c and a restriction percentage R, we define the boundary width to be $R \cdot |L_{max}|$, where L_{max} is the largest layer in the network. For each layer r, the swappable area is then the set of nodes $u \in V_r$ such that $|u_y - c_y| \leq \frac{R}{2} \cdot |L_{max}|$, where u_y is u's y-coordinate in the current drawing. This window is shifted if c is close to the top or bottom of the layer so that the swappable area is always of size $R \cdot |L_{max}|$. Then, step 2 of our technique overview (section 3.2) is modified to unfix only variables x_{u_1,u_2} if $u_1 \in H_i$ or $u_2 \in H_i$, and both u_1 and u_2 are in the swappable area.

A clarification of the term "neighborhood." Throughout this work we use "neighborhood" to refer to the subgraph selected for optimization, due to the intuition of collecting a neighborhood of nodes surrounding a candidate. However, in the context of optimization metaheuristics including large neighborhood search, neighborhood has a specific, and slightly different meaning we refer to as the "true" neighborhood: it is the potential *moves* considered at each iteration [1]. In our adaptation, we use ILP models that optimize with respect to *all possible node moves*. This means the true neighborhood in our case is the exponentially-many possibilities for where the selected subgraph nodes can move to, which is represented by the values each unfixed ILP variable can take. Hence this true neighborhood changes when we use boundaries as previously described, even though the subgraph nodes are unchanged.

4 Evaluation

In this section we describe the datasets and experimental procedures for our two computational evaluations: an experiment analyzing the candidate and neighborhood techniques, and an experiment analyzing the efficacy of neighborhood boundaries.

4.1 Datasets

We use three randomly generated datasets in our experiments.

The first dataset is comprised of rectangular-shaped networks, following the experimental design of Matuszewski et al. [50]. The number of layers K and number of nodes per layer n were generated with a ratio of $3 : 2$ as this represents a standard aspect ratio when the nodes are laid out orthogonally. 5 different $K \times n$ sizes of this aspect ratio were generated—18×12, 24×16, 30×20, 36×24, and 42×28—with 50 networks sampled at each size. Edges were sampled randomly and uniformly between adjacent-layer nodes until the network contained $\frac{3}{2}n(K-1)$ edges, to reach an average density of approximately 3 edges per node. Disconnected nodes were removed, and any networks that remained disconnected were resampled, since disconnected networks can and should be optimized piecewise. These are sparse networks. Note that we use edge count instead of more traditional notions of edge density following Matuszewski's recommendation [50]—edge count provides a better metric for visual edge density on layered networks. The choice to not examine different edge densities was made to simplify the dataset as much as possible to help with runtime estimation, and because optimal models are better-suited for sparse networks while many heuristics perform better on denser networks [42]. Hence large, sparse networks are in more dire need of optimized layouts. They also lend themselves better to node- and edge-level reading tasks that optimal layered layouts are meant to improve. We leave study of this technique for networks with varied densities as future work.

For the second experiment looking at node movement restrictions, we generate two additional datasets changing the shape of the networks, since we hypothesize that restricting the node movement will have more impact on networks with large layers. The first is the same as the previously described rectangular-shaped networks, but with 10% of the layers (randomly sampled, but in a contiguous group) containing $3\times$ the number of nodes. The other dataset consists of triangle-shaped networks, selected because hierarchical data, such as evolutionary trees and file hierarchies [37], lends itself well to layered visualization. The layer size generally increases with hierarchy depth for these networks, which we use the triangular shape to model. For these we use the same base size ratios as the rectangular networks, but modify them to start with one node in the first layer and $2n$ nodes in the last, linearly interpolating the layer counts in between. The random sampling scheme for all of these networks, including the edge sampling and base sizes, is the same as the initial dataset, though we examine 20 networks of each of the five sizes.

4.2 Procedure

We now detail the evaluation conducted for candidate functions, neighborhood aggregation functions, neighborhood sizes, network sizes, and movement restriction on nodes,

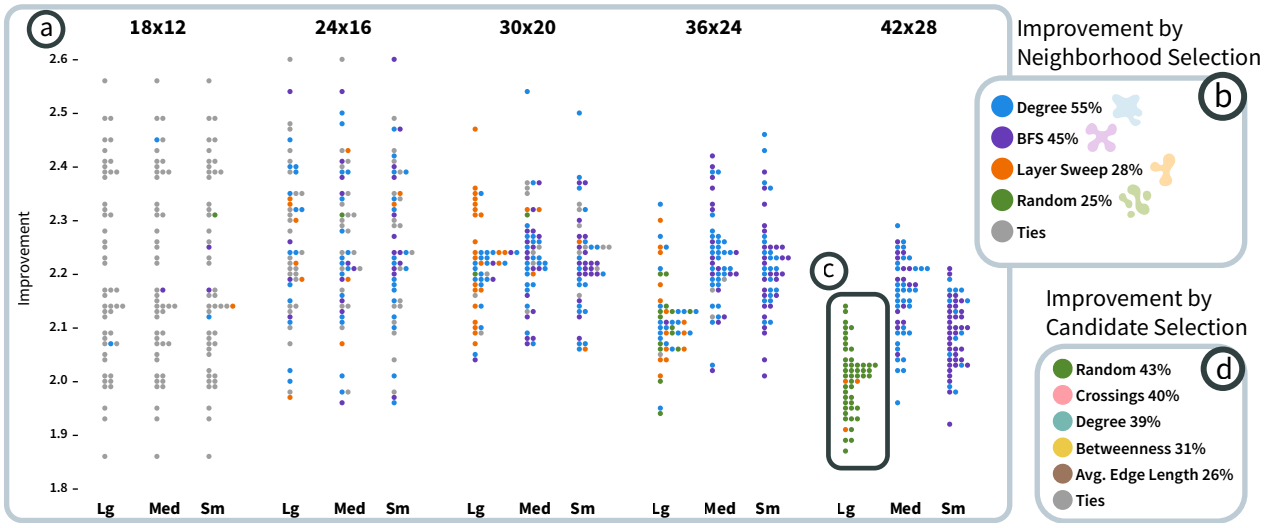


Fig. 5. (a) shows the winning neighborhood method over all 250 networks in terms of crossing improvement. The results are broken up by increasing network size (18×12–42×28), followed by neighborhood size (large, medium, small). The color indicates which neighborhood aggregation technique gave the largest improvement and is gray if there are ties. (b) shows the winning percentage for each technique, including ties. While most techniques reached the same solution on the small networks, BFS (purple) and Degree Centrality (blue) dominated for most larger networks. (c) highlights how most of the random search's wins (green) are from large graphs and large neighborhoods, where the other methods are unable to converge in time. (d) shows the same data filtered by the winning candidate selection method. We found that no technique stands out as a clear winner for candidate selection.

starting with our procedure for controlling optimization time.

4.2.1 Learning the neighborhood size calculation

Since the neighborhood techniques of section 3.4 are so varied in the way they collect subgraph nodes, we must correct for variance between the different methods by learning a unique size calculation for each neighborhood technique according to our formula in eq. (2). The purpose of this is to standardize the number of optimizations performed per minute across the different techniques, as described in section 4.2.1. For each target optimizations per minute in $\{10, 50, 100\}$ and each neighborhood method and input network size, we run all networks in the dataset for 1 minute using a randomly selected neighborhood size. Then we calculate the average optimizations performed per minute and repeat this process nine more times, using binary search to zero in on a number such that when input into eq. (2), the LNS procedure on that specific network size and neighborhood aggregation technique will perform the desired number of mean optimizations per minute. We did not repeat this process for the each candidate method, since candidate selection does not effect size or shape of the subgraph to optimize. This resulted in (4 neighborhood techniques)$\times$(5 input network sizes)$\times$(3 subgraph sizes) = 60 learned size parameters for the first experiment. We henceforth refer to runs of the experiment which performed an average 10 optimizations per minute as using large subgraphs, 50 as medium, and 100 as small.

4.2.2 Candidate + Neighborhood evaluation

We first seek to evaluate the combinations of candidate and neighborhood methods.

For every combination of the two, we ran all 250 input networks for 5 minutes each, using the size constraint to aggregate neighborhood nodes learned in section 4.2.1. This resulted in (5 candidate methods)$\times$(4 neighborhood methods)$\times$(3 neighborhood sizes) = 60 runs of the experiment. At each optimization step we recorded for each network the wall-clock time and number of crossings. Each network was loaded and optimized using the Gurobi optimization software [35].

4.2.3 Restricted movement evaluation

Next, we performed an evaluation limiting the vertical mobility of nodes during optimization, also as a full cross product of the experiment variables. Fixing the candidate method to be randomly selected in order to reduce the number of runs of the experiment, we tested each dataset detailed in section 4.1 for each neighborhood aggregation function at each neighborhood size and network size. For the vertical restriction parameter, we tested each of $\{100\%, 75\%, 50\%\}$ as a percent of maximum layer count. The parameters learned in section 4.2.1 were relearned for the new networks and then each network was run for 5 minutes, as with the first experiment. This resulted in (4 neighborhood methods)$\times$(3 neighborhood sizes)$\times$(3 restriction levels) = 36 runs of the experiment.

4.2.4 Technical Specifications

We ran the experiments in CentOS Linux using 1 CPU with 8 GB RAM and Gurobi 10.0 [35]. Experiments were truncated after the target number of optimizations was performed, or 5 minutes elapsed. E.g. for a target of 10 optimizations per minute, computation was halted once 50 optimizations were performed. In practice, most of the networks completed the target number of optimizations within the 5-minute bound.

5 Results

Our experimental results, broadly speaking, provide strong evidence for the usefulness of optimal Large Neighborhood

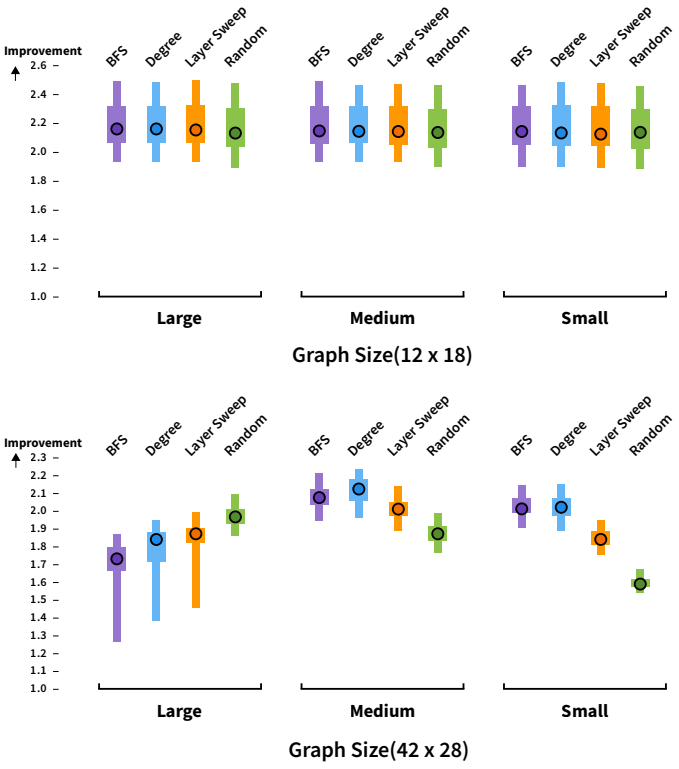


Fig. 6. Crossing improvement across neighborhood techniques for 50 networks of various sizes. The circles outlined in black show the median for each of the five candidate selection methods. The wider boxes contain the middle 50%, and the longer boxes contain the middle 95% of the data. The smallest network size (18×12) is shown in the upper figure, with all techniques displaying near-equal results. The lower figure is the largest network (42×28). As network size increases, the medium and small neighborhood sizes outperform the large size for most methods except for the random selection. This happens because the large neighborhoods don't have the opportunity to explore as much of the network. The degree centrality search technique with the medium neighborhood size performs best for the larger network sizes.

Search in the layered network visualization space. Results are reported in terms of *crossing improvement*, which is calculated as the number of crossings in the initial layout divided by the number of crossings in the final step.

Improvement over iterated barycenter. Our studied techniques consistently yielded a 2–2.5× improvement within the allotted 5 minutes over the standard barycenter heuristic. This is extremely good—reducing crossings by over half provides a huge boost to network readability. This greatly outperforms other reported improvements over the barycenter by previous heuristics, e.g., [27], [34], [44], [50]. Furthermore, this improvement is achieved very quickly on the smaller networks evaluated, needing only a small portion of the 5 minute allotment. This high degree of improvement was generally true across all the techniques and variables studied, highlighting the strength of the optimal-neighborhood search technique for layered crossing minimization.

5.1 Candidate + Neighborhood Evaluation

Neighborhood size is the most important factor for solution convergence. For smaller networks, all neighborhood sizes generally converged to the same number of crossings, as seen in fig. 6 and based on the large number of ties in fig. 5. As network size grows, this factor has a greater effect. We see that large neighborhoods start to perform worse, due to not having time to converge within 5 minutes. Given enough time to converge, however, larger neighborhoods outperform the small neighborhoods, since they are able to find ways of detangling larger structures in the network (fig. 7). Hence, for the larger networks we examine, we find that the medium-sized neighborhoods (∼50 optimizations per minute) perform best.

BFS and Degree centrality neighborhood searches perform best. We see that in fig. 5 that these two techniques outperform the other two for nearly all graph sizes and neighborhood sizes tested, with the degree centrality technique edging out BFS on average. The only subset of the data which did not exhibit these results were very large networks when optimized with large neighborhoods, where random search consistently outperformed the others. This is likely due to properties of the technique wherein it selects larger neighborhoods compared to the other techniques. In effect, it optimizes multiple disconnected neighborhoods simultaneously, allowing it to improve faster but converge without much improvement, as seen in fig. 7.

Diversity in candidate selection is the highest priority. We find that generally speaking, candidate selection on its own does not greatly affect the solution quality, as evidenced by the random candidate method slightly outperforming the others on average. Results are summarized in fig. 5, with full results broken down by candidate in section . The strong performance of the random method implies that the most important factor in candidate selection is diversity, which is a finding in line with the recommendations of Blum et al. for implementations of metaheuristics [4]. We suspect that our penalty function (section 3.3.1) is greatly helping to ensure diversity of the selected candidates, making their performance more similar.

5.2 Limited Mobility Evaluation

The primary takeaway from our evaluation of limited mobility is that even with neighborhoods limited to only half of the available positions to move to, performance is not affected. Hence, this mobility limiting can safely be used without much impact to performance when input networks have layers with many nodes—reducing the considered variables in the model by a polynomial quantity and potentially resulting in considerable performance gains in terms of solver optimization speed.

Even greater improvement can be achieved on certain types of networks. We find that, particularly for the graphs with a small number of extra-large layers, that performance can be improved even further with respect to the barycenter layout. On these networks, the performance of the BFS and degree centrality techniques exceeded 2.8× the performance of the iterated barycenter.

Boundaries improve performance of random and layer sweep techniques. We see in fig. 8 that while the vertical boundaries have little detectable performance improvement for the BFS and degree centrality search techniques, the random and layer sweep techniques perform better with tighter boundaries. This brings their performance more in

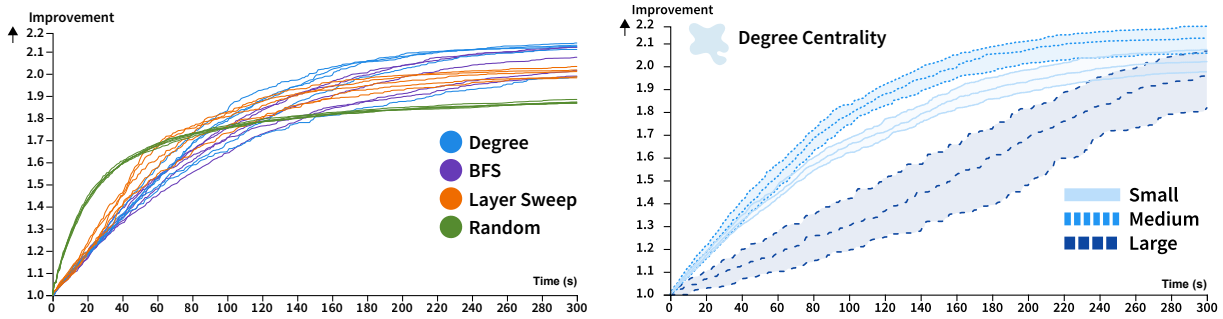


Fig. 7. The left figure displays median crossing improvement over time for all combinations of candidate and neighborhood methods on large 42×28 networks for the medium neighborhood size. There is generally little difference for lines of the same color, meaning candidate selection was not much of a factor. The right figure shows the same data but for only the degree centrality neighborhood technique, with varying neighborhood size and a band representing the middle 50% of the data. The small neighborhood size results in fast convergence, quickly reaching a plateau that doesn't improve further. The large neighborhood size does not have time to converge for these large networks, allowing the medium size to win out for the 5-minute timeframe.

line with BFS and degree centrality, though not enough to eclipse them.

6 Discussion

In this section, we draw out some key insights from our previously described results, and use this to inform recommendations for algorithms implementing LNS for layered network optimization.

A key takeaway from our results was the fast convergence of small neighborhoods but higher ceiling achievable by large neighborhoods. To take advantage of this, it is advisable to **start with small neighborhoods, finding "easy" improvements first, before moving to large neighborhoods that can make more complex rearrangements**.

As for the techniques used to aggregate neighborhood nodes, we consistently find that BFS and degree centrality search provide the best results. The performance of these two techniques align with our hypothesis that connected subgraphs have the greatest potential for crossing reduction—meanwhile, the layer-by-layer technique selects all nodes in the layer regardless of their connectedness, and the random technique ends up selecting many disconnected components. However, boundaries on node mobility have the effect of condensing the selected neighborhoods into more connected groups for these two techniques, improving their performance (fig. 8). Hence, **subgraphs should be selected using a proximity-based connectivity metric**.

For candidate node selection, we found little evidence of a winning technique but still suspect this choice has some impact, particularly for networks with high variance in edge density, which we did not study here. Our candidate techniques also may not be sophisticated enough—many of the optimizations performed result in no change. There may be better ways of choosing candidates that yield improvement.

Additionally, we find evidence that diversity is highly important in neighborhood selection (fig. 5). This means that **there may be benefits to rotating between different candidate and neighborhood selection techniques**.

As described previously, the size of the model scales polynomially with the number of nodes in a layer, due to needing ILP variables for every pair of nodes. Our results show that we can limit this vertical mobility by 50% without any major impact to performance, hence **limiting vertical mobility should be used for networks with very large or dense layers**.

Finally, we see in fig. 7 and will see again in section 7 that the quality of the LNS solution is subject to the quality of the initial solution. Therefore, **practitioners should utilize a high-quality but fast heuristic for the initial layout**.

7 Case Study: Control Flow Graph (CFG) Optimization

CFGs are often represented as layered networks, with nodes depicting code blocks and edges indicating transitions, to aid reverse engineering binary code. Popular decompiler tools like Ghidra and Radare2 can generate CFGs from binary files, revealing code structure and connections between sections of the code. Research highlights the importance of CFG readability in reverse engineering domain tasks [49], [68]. Malware analysts in particular commonly utilize these visualizations as part of their workflow [73], [74]. Tasks include identifying and tracking variables between code blocks and identifying structures that could lead to malicious behavior. Layout computation often employs a Sugiyama-style approach—Radare2's documentation cites Buchheim et al. [6], for instance. Recent advancements from Devkota et al. [13] include more sophisticated approaches to enhance CFG readability and usability even for large graphs. Hence, task performance could be further improved by LNS optimization techniques, especially since these networks can often grow to be very large.

We examine the use of large neighborhood search on the decompiled control flow graph extracted from the code for 13 different Linux commands. To layer these directed networks, we first break cycles using the heuristic of Eades et al. [18], then apply the minimum-width heuristic of Tarassov et al. [66] before re-adding the removed edges. Dummy nodes are then inserted on long edges to create a proper layering, per the technique of Eades and Wormald [21]. These CFGs contained 468–2572 nodes and 540–2828 edges after performing these steps.

For this study, we evaluate both the crossing reduction model described in section 3.1 and a more complex model following the technique of Di Bartolomeo et al. [14]. This

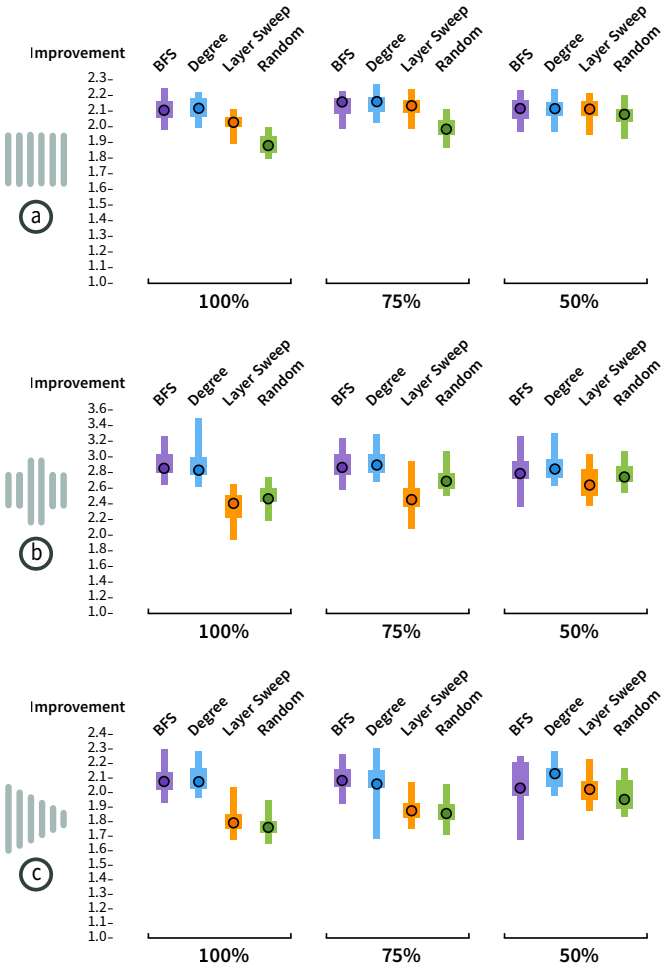


Fig. 8. Results for the *neighborhoods with boundaries* experiment on medium-large networks, broken down by dataset (a: rectangular, b: extra-large layers, c: triangular) and level of vertical neighborhood restriction (100%, 75%, 50%). Results were similar on all three datasets, with the performance in (b) increasing to over 2.8× on average for BFS. BFS and degree centrality searches were not affected much by restricting node mobility, but greater restrictions improved the performance of the layer-by-layer and random searches.

model simultaneously considers both edge crossings and edge bends, another highly important readability metric [58]. We use $\gamma_1 = 3$ and $\gamma_2 = 1$, meaning the model weighs crossings three times more heavily than edge bends—in this case, bend is quantified as the difference in vertical position of the source and target node of an edge, when nodes can only be placed at integer-valued vertical positions. The optimization function, where y_u is a variable representing the vertical position of node u, is then changed to:

$$OPT = 3 \sum_{e_1,e_2 \in E} c_{e_1,e_2} + \sum_{(u,v) \in E} |y_v - y_u| \qquad (4)$$

The computational results provided by Di Bartolomeo et al. show that this model takes nearly 100× longer to optimize than the crossings-only model, making the benefit of a metaheuristic technique even more apparent.

For the LNS implementation, we use degree centrality neighborhoods and random candidate selection with a small-to-medium neighborhood size of $|C| = 1000$ (eq. (2)), with the barycenter layout [64] as a starting solution. We compare this to an implementation of Tabu search (TS) [44] which has been set to run until cut off by the time limit, and the ILP model with symmetry breaking recommended by Wilson et al. [71].

To our knowledge, no incremental improvement-based metaheuristic has previously been evaluated on the hybrid crossings and edge bends model. For this model, we substitute Tabu Search for the weighted median heuristic with optimal placement (WM) of Gansner et al. [25], part of the popular *dot* method [26], though note that it is not a metaheuristic and typically terminates within a few seconds. The optimal node placement algorithm performed after the weighted median step computes final positions for the nodes minimizing total edge length, and can be computed quickly. For our LNS method, we use the same placement algorithm for initial node positions after applying the barycenter and prior to entering the LNS phase.

For each of the 13 control flow graphs, we ran all three methods on both the crossings and crossings+bends model ($13 \cdot 3 \cdot 2 = 78$ runs) for 15 minutes each.

Study Results and Observations. When run on a desktop computer with an Intel Core i7-8700K CPU with 6 cores, 32 GB RAM, Windows 10, & Gurobi 10, we achieve a **median improvement of 5.7× over the optimal model** and **8.6× over Tabu Search** for the 15-minute timeframe. We see in fig. 9 that although the optimal ILP method outperforms LNS on the smaller inputs, this performance is erratic, making highly sporadic improvements to the layout that mean the larger networks are far from optimal after 15 minutes. We note that the parameters of Tabu Search are not tuned for these large, sparse CFGs—the authors examine much smaller networks in their study with much shorter execution times than 15 minutes, and note that their technique, like many crossing minimization heuristics, struggles with sparse networks [44]. Nevertheless, these results are very impressive, indicating that even an unrefined implementation of LNS can fill our aforementioned theoretical niche between optimal methods and heuristics. We provide all layouts generated on OSF with post-processing done for final node placement [25].

The gap is closer for the crossings+edge length model, where LNS improves on the optimal ILP method by 2.6× and the weighted median by 1.5×, on average. Due to the increased complexity of the model, ILP is unable to even find a valid solution for some of the larger graphs. The weighted median is a strong technique for this task, producing high-quality solutions in seconds that can take the LNS method minutes to reach. This suggests that our adaptation of LNS may require some careful parameter tuning to achieve peak performance—the technique would likely improve by using a model-specific heuristic for starting placement such as weighted median instead of barycenter, and learning a neighborhood size calculation for the model.

All layouts generated are available in our supplemental materials at `https://osf.io/w3fev/`. The edge crossing minimization layouts were post-processed with the optimal model of Gansner et al. [25] to reduce edge length subject to the calculated node ordering. For an example, the different layouts for the CFG of `ptx` are linked here: `[ILP]` `[Tabu]` `[LNS]` (crossing minimization only), `[ILP]` `[WM]` `[LNS]` (crossing and edge length minimization)

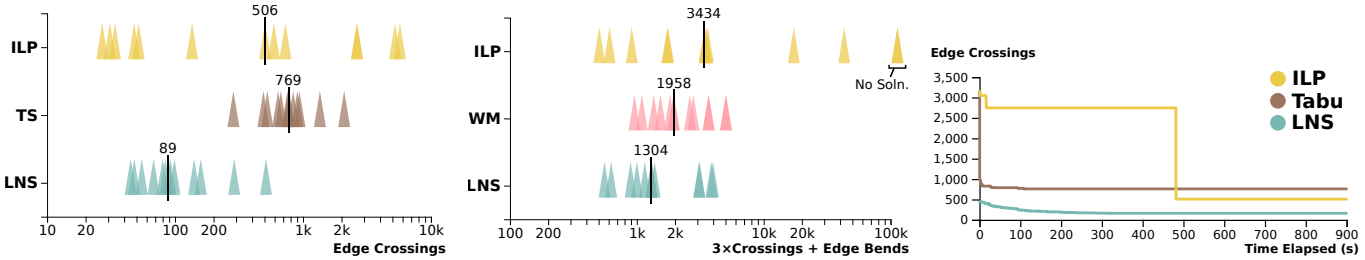


Fig. 9. The left two figures display the final objective function values after 15 minutes for the the ILP technique [14], [71], Tabu Search (TS) [44], the weighted median heuristic (WM) [25], and our LNS adaptation, for 13 large control flow graphs. The left chart plots the results for the edge crossing minimization study, and the middle plots the more complicated model that optimizes both crossings and edge length. The median for each method is highlighted with a black line. In both cases, our LNS-based method produces the best results on average. The behavior of the ILP solver is well-illustrated by the figure on the left, which shows the objective value over time for the edge crossing minimization of the `csplit` graph: the ILP model makes little progress until it gets lucky and finds an improving solution, whereas the LNS approach makes much more consistent progress. The full layouts of `csplit` after the 15 minutes of optimization can be viewed here: `ILP`, `Tabu`, `LNS`.

8 Limitations and Future Work

Limitations. While we analyze many techniques and their performance, it is outside the scope of this work to present a generalized technique. This is due to the lack of existing work analyzing large neighborhood search in this domain and the non-specific nature of the LNS metaheuristic—LNS requires considerable work to justify a process for neighborhood selection, which we have attempted here. We wish to continue towards the presentation of a generalized algorithm in the future.

Our adaptation of LNS, while more scalable than optimal techniques, still has limitations on input size. Since the size of the model scales polynomially in the number of edges/number of nodes per layer, the additional runtime each optimization from iterating each constraint adds up and creates inefficiency for larger networks. For the largest networks, even simply loading the model can incur a non-negligible runtime penalty. With diminishing returns for node-link visualizations on larger networks due to reduced emphasis on individual nodes and edges, however, we believe this technique may scale close to the limits of practicality for node-link visualization, but more work is needed to understand the extent of the scalability of this method.

Future Work. While the benefits of a neighborhood search-based method for layered network optimization are obvious, we leave the construction of a general-purpose algorithm as future work. Following the design guidelines we specify (section 6), this requires an evaluation of existing fast layout heuristics, and an investigation of modifying the neighborhood size once convergence is reached. There may be additional benefits worth evaluating from rotating between different neighborhood techniques, candidate techniques, and vertical movement restrictions, thereby increasing the diversity of explored neighborhoods.

Other tasks left to future work include performing a computational evaluation of the state-of-the-art methods for layered network visualization, such as Tabu Search [8], [44]. We hope to explore extensions of this technique in the context of the broader graph drawing and multilayer network visualization [51] fields.

9 Conclusion

Layout techniques for layered networks often fall into two categorizations: fast heuristics that produce reasonable layouts and optimal methods that create perfect layouts but do not scale well. We have shown that utilizing large neighborhood search (LNS) to repeatedly optimize small subgraphs within a larger network creates high-quality layouts within a bounded execution time. In our evaluations, we show that LNS can be used to reduce the edge crossings in large networks from by over 50% within a few minutes, starting from the standard barycenter layout. By exploring different subgraph aggregation techniques, candidate selection techniques, and constraints on neighborhood size, we have empirically validated a set of recommendations for implementing LNS for network layout optimization. Our case study provides one possible framework for this implementation, showing that high-quality layouts of huge control flow networks can be created with LNS, even using a more complex objective function combining both edge crossings and bends.

Acknowledgments

This work was partly supported by the U.S. National Science Foundation (NSF) under award number IIS-2145382.

References

[1] R. K. Ahuja, Ö. Ergun, J. B. Orlin, and A. P. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(1-3):75–102, Nov. 2002. doi: 10.1016/S0166-218X(01)00338-9

[2] S. D. Bartolomeo, Y. Zhang, F. Sheng, and C. Dunne. Sequence Braiding: Visual Overviews of Temporal Event Sequences and Attributes. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):1353–1363, Feb. 2021. doi: 10.1109/TVCG.2020.3030442

[3] F. Beck, M. Burch, S. Diehl, and D. Weiskopf. A Taxonomy and Survey of Dynamic Graph Visualization. *Computer Graphics Forum*, 36(1):133–159, 2017. doi: 10.1111/cgf.12791

[4] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, Sept. 2003. doi: 10.1145/937503.937505

[5] U. Brandes and D. Wagner. *Analysis and Visualization of Social Networks*, pp. 321–340. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. doi: 10.1007/978-3-642-18638-7_15

[6] C. Buchheim, M. Jünger, and S. Leipert. A fast layout algorithm for k-level graphs. In J. Marks, ed., *Graph Drawing*, pp. 229–240. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[7] M. Burch, C. Vehlow, F. Beck, S. Diehl, and D. Weiskopf. Parallel edge splatting for scalable dynamic graph visualization. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2344–2353, 2011. doi: 10.1109/TVCG.2011.226

[8] S. Cavero, E. G. Pardo, F. Glover, and R. Martí. Strategic oscillation tabu search for improved hierarchical graph drawing. *Expert Systems with Applications*, 243:122668, June 2024. doi: 10.1016/j.eswa.2023.122668

[9] E. Chikofsky and J. Cross. Reverse engineering and design recovery: a taxonomy. *IEEE Software*, 7(1):13–17, 1990. doi: 10.1109/52.43044

[10] T. N. Dang, N. Pendar, and A. G. Forbes. TimeArcs: Visualizing fluctuations in dynamic networks. *Computer Graphics Forum*, 35(3):61–69, 2016. doi: 10.1111/cgf.12882

[11] S. Devkota, P. Aschwanden, A. Kunen, M. Legendre, and K. E. Isaacs. CcNav: Understanding compiler optimizations in binary code. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):667–677, Feb. 2021. doi: 10.1109/TVCG.2020.3030357

[12] S. Devkota and K. E. Isaacs. CFGExplorer: Designing a visual control flow analytics system around basic program analysis operations. *Computer Graphics Forum*, 37(3):453–464, 2018. doi: 10.1111/cgf.13433

[13] S. Devkota, M. P. LeGendre, A. Kunen, P. Aschwanden, and K. E. Isaacs. Domain-centered support for layout, tasks, and specification for control flow graph visualization. In *2022 Working Conference on Software Visualization (VISSOFT)*, pp. 40–50, Oct. 2022. doi: 10.1109/VISSOFT55257.2022.00013

[14] S. Di Bartolomeo, M. Riedewald, W. Gatterbauer, and C. Dunne. Stratisfimal layout: A modular optimization model for laying out layered node-link network visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):324–334, 2022. doi: 10.1109/TVCG.2021.3114756

[15] G. Di Battista, A. Garg, G. Liotta, R. Tamassia, E. Tassinari, and F. Vargiu. An experimental comparison of four graph drawing algorithms. *Computational Geometry*, 7(5-6):303–325, Apr. 1997. doi: 10.1016/S0925-7721(96)00005-3

[16] C. Dunne, S. I. Ross, B. Shneiderman, and M. Martino. Readability metric feedback for aiding node-link visualization designers. *IBM Journal of Research and Development*, 59(2/3):14:1–14:16, Mar. 2015. doi: 10.1147/JRD.2015.2411412

[17] P. Eades and D. Kelly. Heuristics for reducing crossings in 2-layered networks. *Ars Combinatoria*, 21(A):89–98, 1986.

[18] P. Eades, X. Lin, and W. Smyth. A fast and effective heuristic for the feedback arc set problem. *Information Processing Letters*, 47(6):319–323, Oct. 1993. doi: 10.1016/0020-0190(93)90079-O

[19] P. Eades, X. Lin, and R. Tamassia. An algorithm for drawing a hierarchical graph. *International Journal of Computational Geometry & Applications*, 1996. doi: 10.1142/S0218195996000101

[20] P. Eades and N. Wormald. *The Median Heuristic for Drawing 2-Layered Networks*. Technical Report. University of Queensland, Department of Computer Science, 1986.

[21] P. Eades and N. C. Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 11(4):379–403, Apr. 1994. doi: 10.1007/BF01187020

[22] L. C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40(1):35–41, 1977.

[23] Y. Frishman and A. Tal. Online Dynamic Graph Drawing. *IEEE Transactions on Visualization and Computer Graphics*, 14(4):727–740, July 2008. doi: 10.1109/TVCG.2008.11

[24] G. Gange, P. J. Stuckey, and K. Marriott. Optimal k-level planarization and crossing minimization. In U. Brandes and S. Cornelsen, eds., *Graph Drawing*, vol. 6502, pp. 238–249. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. doi: 10.1007/978-3-642-18469-7_22

[25] E. Gansner, E. Koutsofios, S. North, and K.-P. Vo. A technique for drawing directed graphs. *IEEE Transactions on Software Engineering*, 19(3):214–230, Mar. 1993. doi: 10.1109/32.221135

[26] E. R. Gansner, E. Koutsofios, and S. North. *Drawing Graphs with DOT*. Graphviz, 2015.

[27] E. R. Gansner, S. C. North, and K. P. Vo. DAG—a program that draws directed graphs. *Software: Practice and Experience*, 18(11):1047–1062, 1988. doi: 10.1002/spe.4380181104

[28] M. R. Garey and D. S. Johnson. Crossing Number is NP-Complete. *SIAM Journal on Algebraic Discrete Methods*, 4(3):312–316, 1983. doi: 10.1137/0604033

[29] K. Genova and V. Guliashki. Linear Integer Programming Methods and Approaches–A Survey. *Cybernetics and Information Technologies*, 11, 2011.

[30] F. Glover, V. Campos, and R. Martí. Tabu search tutorial. A Graph Drawing Application. *TOP*, 29(2):319–350, July 2021. doi: 10.1007/s11750-021-00605-1

[31] F. Glover and M. Laguna. Tabu Search. In D.-Z. Du and P. M. Pardalos, eds., *Handbook of Combinatorial Optimization: Volume1–3*, pp. 2093–2229. Springer US, 1998. doi: 10.1007/978-1-4613-0303-9_33

[32] T. E. Gorochowski, M. Di Bernardo, and C. S. Grierson. Using Aging to Visually Uncover Evolutionary Processes on Networks. *IEEE Transactions on Visualization and Computer Graphics*, 18(8):1343–1352, Aug. 2012. doi: 10.1109/TVCG.2011.142

[33] M. Gronemann, M. Jünger, F. Liers, and F. Mambelli. Crossing Minimization in Storyline Visualization. In Y. Hu and M. Nöllenburg, eds., *Graph Drawing and Network Visualization*, Lecture Notes in Computer Science, pp. 367–381. Springer International Publishing, Cham, 2016. doi: 10.1007/978-3-319-50106-2_29

[34] W. Günther, R. Schönfeld, B. Becker, and P. Molitor. K-Layer Straightline Crossing Minimization by Speeding Up Sifting. In J. Marks, ed., *Proceedings of the 8th International Symposium on Graph Drawing*, Lecture Notes in Computer Science, pp. 253–258. Springer, Berlin, Heidelberg, 2000. doi: 10.1007/3-540-44541-2_24

[35] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual. https://www.gurobi.com/documentation/10.0/refman/index.html.

[36] P. Healy and N. S. Nikolov. *Hierarchical Drawing Algorithms*, p. 46. Chapman and Hall/CR, 2013.

[37] I. Herman, G. Melancon, and M. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, Jan. 2000. doi: 10.1109/2945.841119

[38] B. Hurley and B. O'Sullivan. Statistical regimes and runtime prediction. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, 7 pages, p. 318–324. AAAI Press, 2015.

[39] F. Hutter, L. Xu, H. H. Hoos, and K. Leyton-Brown. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206:79–111, Jan. 2014. doi: 10.1016/j.artint.2013.10.003

[40] IBM ILOG CPLEX. V12. 1: User's manual for CPLEX. *International Business Machines Corporation*, 46(53):157, 2009.

[41] M. Jünger, E. K. Lee, P. Mutzel, and T. Odenthal. A polyhedral approach to the multi-layer crossing minimization problem. In G. Goos, J. Hartmanis, J. van Leeuwen, and G. DiBattista, eds., *Graph Drawing*, vol. 1353, pp. 13–24. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997. doi: 10.1007/3-540-63938-1_46

[42] M. Jünger and P. Mutzel. 2-Layer Straightline Crossing Minimization: Performance of Exact and Heuristic Algorithms. *Journal of Graph Algorithms and Applications*, 1, Jan. 1997. doi: 10.1142/9789812777638_0001

[43] M. Laguna and R. Marti. GRASP and Path Relinking for 2-Layer Straight Line Crossing Minimization. *INFORMS Journal on Computing*, 11(1):44–52, Feb. 1999. doi: 10.1287/ijoc.11.1.44

[44] M. Laguna, R. Martí, and V. Valls. Arc crossing minimization in hierarchical digraphs with tabu search. *Computers & Operations Research*, 24(12):1175–1186, Dec. 1997. doi: 10.1016/S0305-0548(96)00083-4

[45] B. Lee, C. Plaisant, C. Parr, J.-D. Fekete, and N. Henry Riche. Task taxonomy for graph visualization. In *Proceedings of AVI Workshop on BEyond time and Errors*, pp. 1–5, 05 2006. doi: 10.1145/1168149.1168168

[46] C. D. G. Linhares, J. R. Ponciano, F. S. F. Pereira, L. E. C. Rocha, J. G. S. Paiva, and B. A. N. Travençolo. A scalable node ordering strategy based on community structure for enhanced temporal network visualization. *Computers & Graphics*, 84:185–198, 2019. doi: 10.1016/j.cag.2019.08.006

[47] M. Liu, J. Shi, Z. Li, C. Li, J. Zhu, and S. Liu. Towards better analysis of deep convolutional neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):91–100, 2017. doi:10.1109/TVCG.2016.2598831. doi: 10.1109/TVCG.2016.2598831

[48] B. Malone, K. Kangas, M. Järvisalo, M. Koivisto, and P. Myllymäki. Empirical hardness of finding optimal Bayesian network structures: Algorithm selection and runtime prediction. *Machine Learning*, 107(1):247–283, 2018. doi: 10.1007/s10994-017-5680-2

[49] A. Mantovani, S. Aonzo, Y. Fratantonio, and D. Balzarotti. RE-Mind: A First Look Inside the Mind of a Reverse Engineer. In *31st USENIX Security Symposium (USENIX Security 22)*, 2022.

[50] C. Matuszewski, R. Schönfeld, and P. Molitor. Using Sifting for k-Layer Straightline Crossing Minimization. In J. Kratochvíyl, ed., *Graph Drawing*, Lecture Notes in Computer Science, pp. 217–224. Springer, Berlin, Heidelberg, 1999. doi: 10.1007/3-540-46648-7_22

[51] F. McGee, M. Ghoniem, G. Melançon, B. Otjacques, and B. Pinaud. The State of the Art in Multilayer Network Visualization. *Computer Graphics Forum*, 38(6):125–149, 2019. doi: 10.1111/cgf.13610

[52] D. R. Morrison, S. H. Jacobson, J. J. Sauppe, and E. C. Sewell. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19:79–102, 2016. doi: 10.1016/j.disopt.2016.01.005

[53] A. Napoletano, A. Martínez-Gavara, P. Festa, T. Pastore, and R. Martí. Heuristics for the Constrained Incremental Graph Drawing Problem. *European Journal of Operational Research*, 274(2):710–729, Apr. 2019. doi: 10.1016/j.ejor.2018.10.017

[54] N. S. Nikolov and A. Tarassov. Graph layering by promotion of nodes. *Discrete Applied Mathematics*, 154(5):848–860, 2006. doi: 10.1016/j.dam.2005.05.023

[55] B. Peng, S. Wang, D. Liu, Z. Su, Z. Lü, and F. Glover. Solving the incremental graph drawing problem by multiple neighborhood solution-based tabu search algorithm. *Expert Systems with Applications*, 237:121477, Mar. 2024. doi: 10.1016/j.eswa.2023.121477

[56] A. Perer and D. Gotz. Data-driven exploration of care plans for patients. In *CHI '13 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '13, 6 pages, p. 439–444. Association for Computing Machinery, New York, NY, USA, 2013. doi: 10.1145/2468356.2468434

[57] H. Purchase. Which aesthetic has the greatest effect on human understanding? In G. DiBattista, ed., *Graph Drawing*, Lecture Notes in Computer Science, pp. 248–261. Springer, Berlin, Heidelberg, 1997. doi: 10.1007/3-540-63938-1_67

[58] H. C. Purchase. Metrics for Graph Drawing Aesthetics. *Journal of Visual Languages & Computing*, 13(5):501–516, Oct. 2002. doi: 10.1006/jvlc.2002.0232

[59] G. R. Raidl and J. Puchinger. Combining (Integer) Linear Programming Techniques and Metaheuristics for Combinatorial Optimization. In C. Blum, M. J. B. Aguilera, A. Roli, and M. Sampels, eds., *Hybrid Metaheuristics: An Emerging Approach to Optimization*, pp. 31–62. Springer, 2008. doi: 10.1007/978-3-540-78295-7_2

[60] S. Ropke and D. Pisinger. An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows. *Transportation Science*, 40(4):455–472, Nov. 2006. doi: 10.1287/trsc.1050.0135

[61] P. Shaw. Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In M. Maher and J.-F. Puget, eds., *Principles and Practice of Constraint Programming — CP98*, Lecture Notes in Computer Science, pp. 417–431. Springer, Berlin, Heidelberg, 1998. doi: 10.1007/3-540-49481-2_30

[62] M. A. Smith, B. Shneiderman, N. Milic-Frayling, E. Mendes Rodrigues, V. Barash, C. Dunne, T. Capone, A. Perer, and E. Gleave. Analyzing (social media) networks with nodexl. In *Proceedings of the Fourth International Conference on Communities and Technologies*, C&T '09, 10 pages, p. 255–264. Association for Computing Machinery, New York, NY, USA, 2009. doi: 10.1145/1556460.1556497

[63] J. Song, r. lanka, Y. Yue, and B. Dilkina. A General Large Neighborhood Search Framework for Solving Integer Linear Programs. In *Advances in Neural Information Processing Systems*, vol. 33, pp. 20012–20023. Curran Associates, Inc., 2020.

[64] K. Sugiyama, S. Tagawa, and M. Toda. Methods for Visual Understanding of Hierarchical System Structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(2):109–125, Feb. 1981. doi: 10.1109/TSMC.1981.4308636

[65] R. Tamassia. *Handbook of Graph Drawing and Visualization*. Chapman & Hall/CRC, 1st ed., 2016.

[66] A. Tarassov, N. S. Nikolov, and J. Branke. A Heuristic for Minimum-Width Graph Layering with Consideration of Dummy Nodes. In C. C. Ribeiro and S. L. Martins, eds., *Experimental and Efficient Algorithms*, Lecture Notes in Computer Science, pp. 570–583. Springer, Berlin, Heidelberg, 2004. doi: 10.1007/978-3-540-24838-5_42

[67] S. van den Elzen, D. Holten, J. Blaas, and J. J. van Wijk. Reordering massive sequence views: Enabling temporal and structural analysis of dynamic networks. In *2013 IEEE Pacific Visualization Symposium (PacificVis)*, pp. 33–40, 2013. doi: 10.1109/PacificVis.2013.6596125

[68] D. Votipka, S. Rabin, K. Micinski, J. S. Foster, and M. L. Mazurek. An Observational Investigation of Reverse Engineers' Processes. In *29th USENIX Security Symposium (USENIX Security 20)*, pp. 1875–1892. USENIX Association, 2020.

[69] S. Voulgaropoulou, N. Samaras, and N. Ploskas. Predicting the Execution Time of the Primal and Dual Simplex Algorithms Using Artificial Neural Networks. *Mathematics*, 10(7):1038, 2022. doi: 10.3390/math10071038

[70] C. Ware, H. Purchase, L. Colpoys, and M. McGill. Cognitive Measurements of Graph Aesthetics. *Information Visualization*, 1(2):103–110, June 2002. doi: 10.1057/palgrave.ivs.9500013

[71] C. Wilson, E. Puerta, T. Crnovrsanin, S. Di Bartolomeo, and C. Dunne. Evaluating and extending speedup techniques for optimal crossing minimization in layered graph drawings, Aug 2024. doi: 10.31219/osf.io/abz8e

[72] K. Wongsuphasawat, D. Smilkov, J. Wexler, J. Wilson, D. Mané, D. Fritz, D. Krishnan, F. B. Viégas, and M. Wattenberg. Visualizing dataflow graphs of deep learning models in tensorflow. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):1–12, 2018. doi:10.1109/TVCG.2017.2744878. doi: 10.1109/TVCG.2017.2744878

[73] K. Yakdan, S. Dechand, E. Gerhards-Padilla, and M. Smith. Helping Johnny to Analyze Malware: A Usability-Optimized Decompiler and Malware Analysis User Study. In *2016 IEEE Symposium on Security and Privacy (SP)*, pp. 158–177. IEEE, San Jose, CA, USA, May 2016. doi: 10.1109/SP.2016.18

[74] M. Yong Wong, M. Landen, M. Antonakakis, D. M. Blough, E. M. Redmiles, and M. Ahamad. An Inside Look into the Practice of Malware Analysis. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pp. 3053–3069. ACM, Virtual Event Republic of Korea, Nov. 2021. doi: 10.1145/3460120.3484759

[75] D. C. Zarate, P. L. Bodic, T. Dwyer, G. Gange, and P. Stuckey. Optimal Sankey Diagrams Via Integer Programming. In *2018 IEEE Pacific Visualization Symposium (PacificVis)*, pp. 135–139, Apr. 2018. doi: 10.1109/PacificVis.2018.00025

[76] J. Zink, J. Walter, J. Baumeister, and A. Wolff. Layered drawing of undirected graphs with generalized port constraints. *Computational Geometry*, 105–106:101886, 2022. doi: 10.1016/j.comgeo.2022.101886

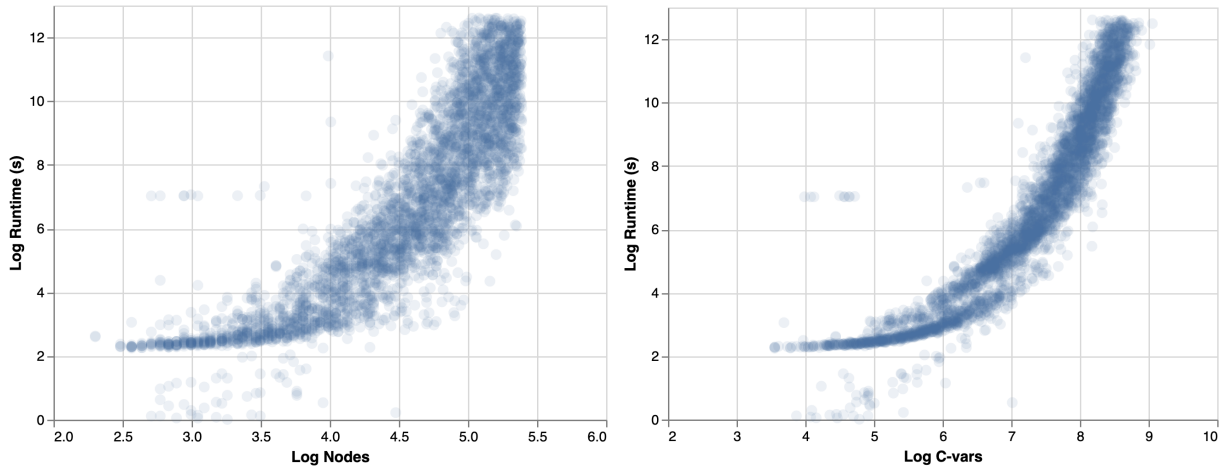


Fig. 10. On the left, the runtime of optimal model in log-milliseconds are graphed as a function of the natural log of the number of nodes, and on the right as a function of the natural log of the number of crossing variables, for the same set of 3,880 networks. While these results still display a high degree of variance, the crossing variable count is clearly a better runtime predictor.

Runtime Prediction

A previous study examining the performance of the optimal crossing minimization model of Jünger et al. [41] on a collection of 3,880 networks from the Rome-Lib graphs [15] found results supporting the conclusion that the number of crossing variables in the model is a better predictor of runtime than the number of nodes in the graph—see fig. 10. While not a perfect predictor, we find that this calculation is practice is more reliable and consistent than number of nodes, and can be calculated easily from information readily available about the input network, described in section 3.4.1. To find a better predictor, it may be necessary to train a machine learning model, a procedure described in [69].

Supplemental Figures

We provide a number of additional figures describing our experimental results from section 4.2.2. fig. 11 and fig. 12 contain improvement results not included in section 5, fig. 13 shows winning method across all networks by candidate function instead of neighborhood function.

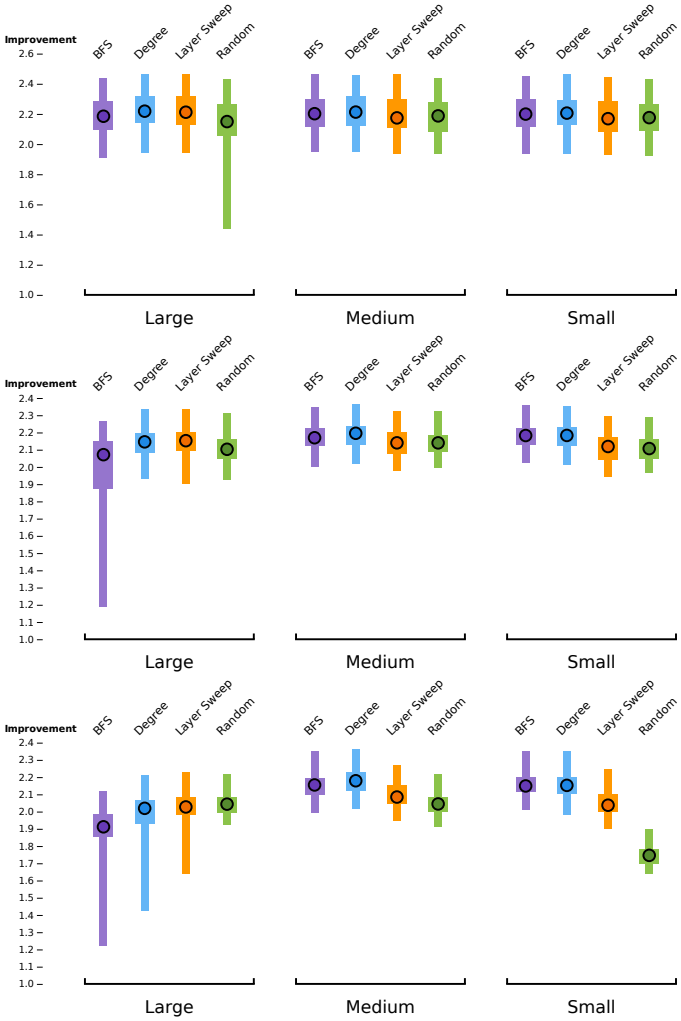


Fig. 11. Improvement results for the middle network sizes not pictured in fig. 6. From top to bottom, we have 24×16, 30×20, and 36×24. As network size increases, the relative performance of the degree candidate and BFS search techniques begins to eclipse the other two for the medium and small neighborhood sizes.

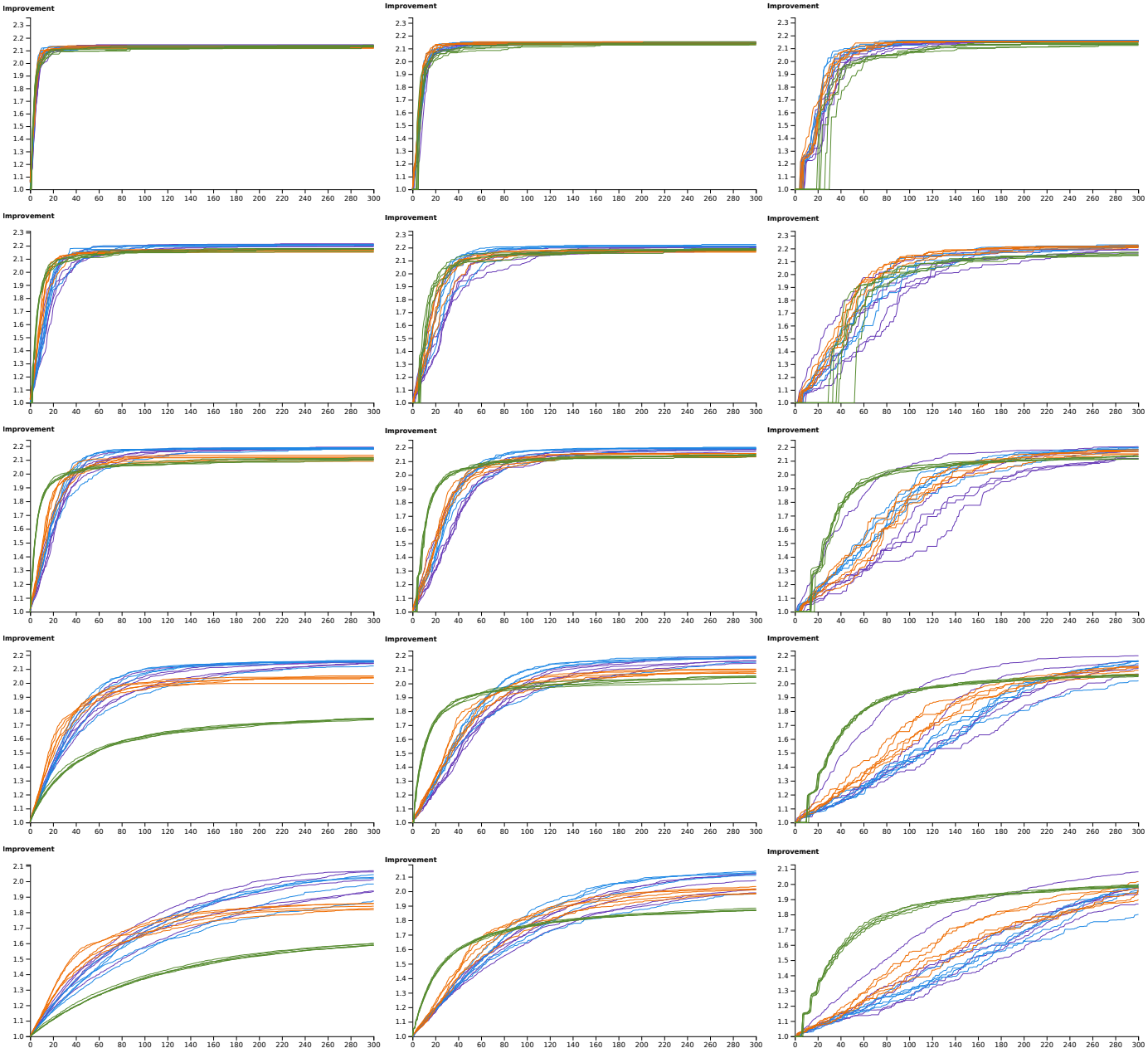


Fig. 12. Individual figures display all 15 candidate + neighborhood combinations graphed as median improvement over the 5-minute timespan. The color corresponds to neighborhood aggregation technique, where purple is BFS, blue is degree centrality search, orange is layer-by-layer search, and green is random search. From left to right, the columns are: small neighborhoods, medium neighborhoods, large neighborhoods. From top to bottom, the rows are: 18×12, 24×16, 30×20, 36×24, and 42×28 network sizes.

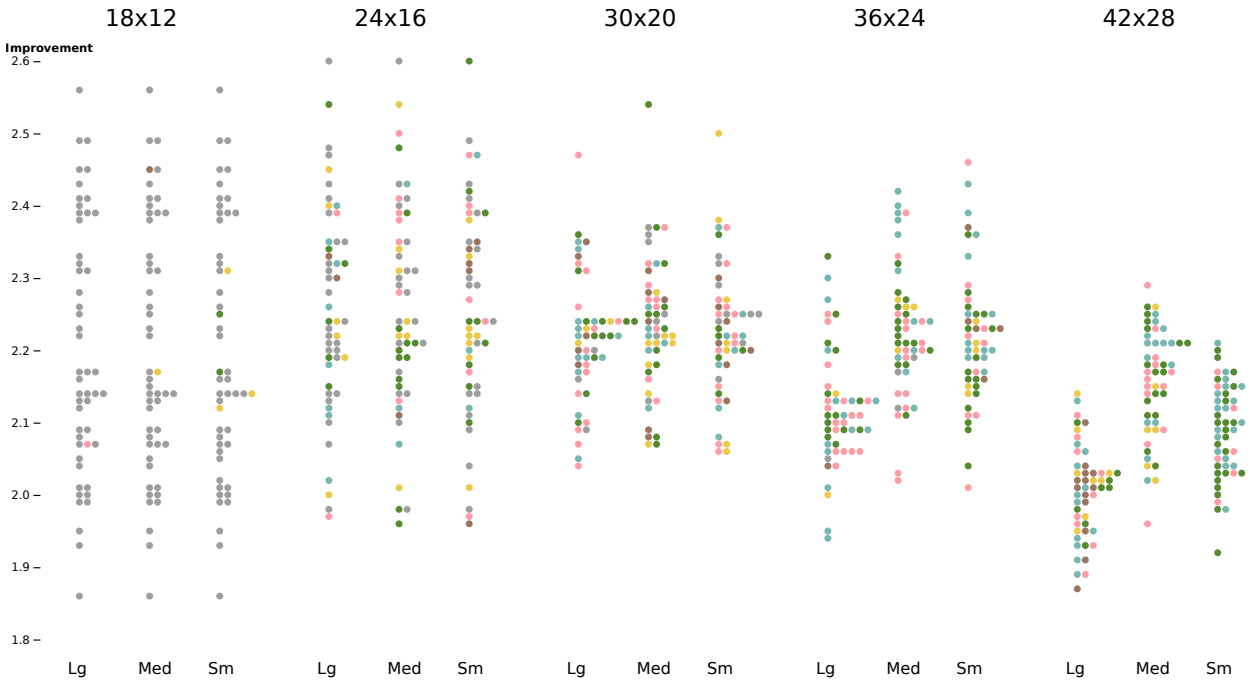


Fig. 13. Improvement results from fig. 5 but instead colored by winning candidate, where gray represents a tie. Results are broken up by increasing network size and then by neighborhood size. We see no clear winner for any particular graph or neighborhood size, though the average edge length candidate wins less then the other methods.