# Towards a General Solution for Layout of Visual Goal Models with Actors

Yilin Lucy Wang, Alicia M. Grubb
Department of Computer Science
Smith College, Northampton, MA, USA
{lwang, amgrubb}@smith.edu

***Abstract*—Goal models help stakeholders make trade-off decisions in the early stages of project development. While these approaches have significant analysis capabilities, they have yet to see broad industrial adoption, with the construction of scalable, realistic goal models acting as a significant barrier. Over the last decade, researchers have used *force-directed algorithms*, specifically GraphViz, to layout goal models and have called for improved layout algorithms to better accommodate the unique challenges presented by actor-based models. We extend a force-directed algorithm to include goal model heuristics, and independently arrived at a domain specific version of a generic layout algorithm for *undirected compound graphs*. As initial validation of the effectiveness and scalability of our algorithm, we implement our approach in BloomingLeaf, a goal model analysis tool. Initial results are promising; yet, further collaboration and validation across the various goal modeling approaches (e.g., GRL, iStar, Tropos) is required before we can recommend our approach to be adopted in tooling. This paper presents early results and lays a foundation for discussion within our GORE community.**

## I. Introduction

Goal-Oriented Requirements Engineering (GORE) helps stakeholders elicit and specify requirements in the early-phases of a project. For example, stakeholders may model system requirements and the intentions of their users. GORE frameworks also provide powerful analysis capabilities to solve problems facing stakeholders, and connect their models with downstream development activities [1], [2], [3]. Most notably, stakeholders can ask trade-off questions of their models and make project decisions; however, GORE analysis techniques are not used in industry because of the scalability issues with the modeling (i.e., the actual creation and understanding of models) [4], [5]. We aim to support stakeholders' ability to create large realistic goal models.

In this paper, we explore and describe the problem of automatic layout of actor-based goal models (AGMs). We define actor-based GORE languages as those that have the construct of an actor that subsumes intentionality where both the actor and their intentions are presented in the same visualization. Tropos, GRL, and i* (iStar) are actor-based. The KAOS language does not group intentions within actor boundaries; therefore, standard layout algorithms designed for trees can be used [6]. Similarly, other RE notations that describe processes (e.g., BPMN [7] and UCM [8]) can layout models using a grid.

Some actor-based goal modeling tools implement *force-directed algorithms* (see Sect. II), while others continue to call for automatic layout algorithms in their future work [9], [10]. The presence of actors, as *containers* for intentions, limits the effectiveness of standard force-directed algorithms [11]. Creating appropriate algorithms for layout is essential to the application of merging goal models, generating models from natural language requirements, and creating tooling and visualizations to allow for industrial adoption and model management [9], [12].

The implications of layout decisions have already been studied in the literature. Laue and Storch noted problems associated with misleading layouts when validating goal model syntax [13]. Santos et al. investigated the effects of bad layouts on iStar models, and found no significant difference in understanding and reviewing tasks, for small models (i.e., 2 actors and 20 intentions) [14]. They argued that the negative effects of a bad layout escalate as the model size increases, reinforcing the scalability issues described in prior work.

In this paper, we present FLAAG, an automated **F**orce-directed **L**ayout **A**lgorithm for **A**ctor-based **G**oal models. We define a set of heuristics to evaluate FLAAG and compare it to related work. We discuss three research questions:

- RQ1 How does FLAAG compare with prior automatic-layout algorithms?
- RQ2 To what extent does FLAAG accommodate goal model syntax and semantics?
- RQ3 To what extent does FLAAG scale to large realistic models?

As this paper documents our ongoing research, we present preliminary results for these research questions and add additional questions that arose during validation (see Sect. V).

The remainder of this paper is organized as follows. Sect. II provides background of the current state of the art in automatic layout of actor-based goal models. Sect. III describes our evaluation criteria and introduces our approach, while Sect. IV presents FLAAG in detail. Sect. V discusses our initial evaluation and future roadmap for collaboration.

## II. Background

In this section, we introduce force-directed layout algorithms and their application within the GORE community.

TABLE I: Actor-based Goal Model Layout Heuristics

Each row lists a heuristic and which algorithm satisfies (Y-Yes / N-No / P-Partial) the heuristic, where, FDA is a generic force-directed algorithm; UCG is the undirected compound graph algorithm; and FLAAG is the actor-based goal model algorithm described in this paper. Heuristics denoted with a † were adapted from [14].

| Heuristics | FDA | UCG | FLAAG |
|---|---|---|---|
| $H_1$ Evenly distribute elements within the model. | Y | Y | Y |
| $H_2$ Have minimal space between elements. | Y | Y | Y |
| $H_3$ Avoid intersecting/overlapping links with other links† | N | N | P |
| $H_4$ Avoid or minimize overlapping boundaries of Actors where possible† | N | Y | Y |
| $H_5$ Limit the number of links that cross over actors. | N | N | P |
| $H_6$ Language specific hierarchical elements (e.g., tasks lower than goals). | N | N | Y |
| $H_7$ Use a consistent direction for the goal decomposition hierarchy† | N | N | Y |
| $H_8$ Decomposition relationships are closer together. | N | N | Y |
| $H_9$ Decomposition children are below their parents. | N | N | Y |
| $H_{10}$ Avoid intersecting/overlapping links with elements' text† | Y | Y | P |
| $H_{11}$ All intentions that belong to an actor, must be placed inside the actor boundary† | N | Y | Y |
| $H_{12}$ Keep elements horizontal. Do not tilt or twist them† | Y | Y | Y |
| $H_{13}$ Keep links identifiers outside the boundaries of actors to improve readability† | N | N | Y |
| $H_{14}$ Use circles for actors' boundaries unless other shapes improve readability† | N | Y | Y |
| $H_{15}$ Don't extend the text of the name of the element beyond the element's border† | N | N | Y |

### A. Force-directed Algorithms (FDAs)

*Force-directed algorithms* (FDAs) layout graphs of nodes and edges, by distributing nodes in models evenly in the entire space [15]. While they vary in complexity for graphs of different shapes [16], these algorithms generally work by exerting repulsions between all nodes in combination with exerting attractions between nodes with links, without bending any of the links in the model [11].

GraphViz (http://graphviz.org) is an online open source library that implements customizable FDAs with layout options (i.e., *dot, neato, fdp, sfdp, twopi* and *circo*). For example, *fdp* and *sfdp* implements the FDA on different sized models. The *dot* option works best in drawing hierarchical directed graphs, whereas *neato* uses a spring model and distributes the nodes by minimizing the overall energy of the system.

### B. Automatic Layout within Actor-based Goal Models

FDAs, specifically GraphViz, have been exclusively used in developing tooling for validation of goal modeling approaches. Here, we describe goal modeling tools and their approach to automatic layout based on documented details. OpenOME is an early tool for modeling iStar [17]. OpenOME can layout basic models using *dot* in GraphViz, meaning the models appear to be hierarchical based on the direction of the links. REDEPEND-REACT, an architecture analysis tool that includes iStar elements, creates models using a tree-form hierarchy [18]. jUCMNav is a well developed tool for GRL models and Use Case diagrams. It uses GraphViz to complete auto-layout features, but to the best of our knowledge the feature is limited and only supports simple models [8], [19]. MUSER is a modeling tool by Li et al. for security requirements analysis for socio-technical systems using iStar and Techne [20]. MUSER inherits similar layout features from OmniGraffle, the proprietary tool it extends. To improve the layout within MUSER, Li and collaborators completed initial work to customize an FDA for laying out iStar SD models [21], [22]. Finally, several text-based languages for goal models exist. For example, iStarJSON for iStar has a custom tool that uses GraphViz [23], and TGRL for GRL uses jUCMNav (discussed above) for visual layout [24]. FDAs have only provided limited support for laying out goal models; thus, we propose a new algorithm to improve model layouts.

## III. Problem & Approach

In this section, we define our evaluation criteria and demonstrate the need for FLAAG.

### A. Evaluation Criteria: Layout Heuristics

In this paper, we evaluate actor-based goal model layouts against the list of heuristics in Table I. Using the guidelines provided by Santos et al. [14], we selected all heuristics related to visual layout (denoted by † in Table I) and ignored all rules associated with wellformedness and completeness. We propose six additional heuristics based on our experience with tool development [25]. While this list may be incomplete, it as a first attempt to evaluate the effectiveness of three algorithms.

### B. Limitations of FDAs for Actor-based Goal Models

As discussed in Sect. II, FDAs (i.e., GraphViz) are insufficient for the task of laying out goal models due to the presence of actors. As you can see from the FDA column in Table I, the force-directed algorithm described in Sect. II does not satisfy our needs. It allows for the even distribution of intentions and relationships (i.e., $H_1$, $H_2$) and separations of links and text (e.g., $H_{10}$, $H_{12}$), but does not allow for the strict grouping of elements by actors. Some GORE tools have modified a generic FDA (e.g., jUCMNav and MUSER), to allow specialized positioning by node type, which would satisfy $H_6$ and $H_9$ (see Table I), as well as have additional post algorithm procedures to improve the visual appeal, satisfying $H_{11}$ and $H_{13}$. FDAs do not satisfy all heuristics in Table I

and are insufficient for layout of both actors and intentions because they only handle a single level of abstraction.

### C. Treating Actors as Containers

Our initial approach was to consider force-directed algorithms at both the actor-level and the intention level of the model. For example, consider the iStar toy-example illustrated in Fig. 1(i), which contains actors Mike, Student, and Travel Agent. Student depends on Travel Agent to satisfy the goal Trip Bundle Booked. This way we would apply a force-directed algorithm on the intentions in Student and Travel Agent, then again to the actors, but this does not enforce that Student and Travel Agent be adjacent, or that Bundle Booked be adjacent to Trip Bundle Booked.

In the next section, we present FLAAG. We independently arrived at this algorithm; yet, detailed comparisons with related work revealed that our algorithm can be considered a domain specific instantiation of a generic algorithm for *undirected compound graphs* (UCG) [27]. We begin by comparing the premises and assumptions of the UCG algorithm, and its ability to satisfy our heuristics, which motivate our domain specific version.

Dogrusoz et al. undirected compound graph (UCG) algorithm, which builds on the traditional force-directed algorithm, is able to handle arbitrary levels of nesting and non-uniform node sizes, as well as graph edges that span multiple levels of nesting and links that connect to non-leaf nodes [27]. The UCG algorithm works by creating a gravity force at the center of each level of the nested graph to pull elements from the same level of the graph together. Nodes only pull and repel other nodes on the same nesting level of the graph. The bounding boxes on some levels of graph are elastic, allowing them to change size along with changes in the shape of the graph. Intergraph edges are treated specially, where part of the edge inside the bounding box for a given nesting is assigned a constant force so that nodes connected to the box will be pulled to the boundary. The remaining links are represented as regular springs.

The layout of actor-based goal models is improved with each of the capabilities of the UCG algorithm. For our purpose, the UCG algorithm can be simplified because all goal models can be represented with only one level of nesting. For example, Fig. 1(ii) illustrates an abstracted undirected version of our toy example. a, g, and i were actors, but in this view a, g, h and i are at the same level. Each intention not part of an actor (e.g., dependums in iStar) is place inside an *invisible actor* container for our algorithm.

Furthermore, we can write a transformation function between the syntax of a goal model and the syntax of a compound graph. An *actor-based goal model* is a tuple $\langle A, G, R\rangle$, where $A$ is a set of actors, $G$ is a set of intention nodes and $R$ is a set of goal relations over $G$ [28]. A *compound graph* is a tuple $\langle V, E, F\rangle$, where $V$ is a set of nodes, $E$ is a set of adjacency edges, and $F$ is a set of inclusion edges denoting which nodes belong to another node (i.e., are members of a sub-graph) [27]. In this transformation, the set $V$ becomes the union of $A$ and $G$, plus the invisible actors created for independent intentions. The set $E$ is mapped to $R$, and $F$ is the mapping between actors $A$ and intentions $G$. Returning to Fig. 1, the compound graph for our toy example is: $V = \{\mathsf{a,b,c,d,e,f,g,h,i}\}$, $E = \{\{\mathsf{a,g}\}, \{\mathsf{b,c}\}, \{\mathsf{b,d}\}, \{\mathsf{d,e}\}, \{\mathsf{e,f}\}\}$, $F = \{\mathsf{gb}, \mathsf{gc}, \mathsf{gd}, \mathsf{he}, \mathsf{if}\}$.

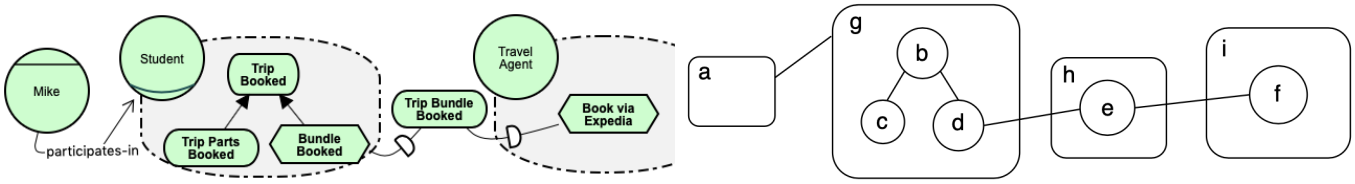
Although we can represent goal models as compound graphs, the algorithm presented by Dogrusoz et al. on its own does not sufficiently satisfy the heuristics listed in Table I (see UCG column), where only about half of the heuristics are assigned a Y. Thus, further customization is required. In the next section, we give the details of FLAAG, which does satisfy our heuristics (see FLAAG column in Table I).

## IV. FLAAG: Algorithm Description

FLAAG is presented in Algo. 1 on Lines 1–13. For space considerations, full details are not provided for each helper function[1]. Algo. 1 takes as input a goal model M, three constants, as well as two optional inputs containing initial layout information and the maximum number of iterations (i.e., timeout). Algo. 1 consists of initialization (on Lines 1–2), a main loop (Lines 3–8), and four visualization functions that convert the resulting layout back to a format appropriate for rendering (Lines 9–13). The INITIALIZATION function assigns initial coordinates to each element either sequentially or using *initLay*. The main loop (see Lines 3–8 in Algo. 1) creates the relative positioning of each element in the model by iteratively calculating the forces between each element and updating the displacement accordingly (see ADJUST function on Lines 15–30). The main loop completes when CHECKCOND determines whether the intermediate product satisfies $H_2$–$H_5$ and $H_8$–$H_9$ (see Table I), or if the maximum number of iterations *maxItr* has been reached. At this point in Algo. 1, the graph is rendered. SETCOORDINATEPOSITIVE (see Line 9) uses an off-set to shift the coordinates of each element to become relative to $(0, 0)$. Next, GETSIZEOFACTOR calculates the width and length of each actor boundary, and CALCULATEACTORPOSWITHREC ensures that actor boundaries do not overlap given the updates to the boundary sizes in GETSIZEOFACTOR. Finally, MOVENODESTOABSPOS calculates the final position of each node within each actor.

**ADJUST Function.** ADJUST changes the position of *curNode* relative to the elements in the model by calculating the attraction and repulsion forces. The function takes as input, the model (represented by the node and actor set), the current node (*curNode*), and whether *curNode* is an actor (*curIsActor*), as well as the user specified constants $C_A, C_N$, and $C_M$. $C_A$ and $C_N$ are used and described in the ATTRACTION function (see below). $C_M$ is a factor in calculating the distance an element is moved on each call to ADJUST (see Lines 29–30). First, ADJUST constructs the correct node set for later calculation (see Lines 15–21 in Algo. 1). If *curNode* is an actor, then *elemSet* contains all the actors (on Line 15), otherwise, it contains the remainder of the nodes within the same actor

[1]https://doi.org/10.35482/csc.001.2020




(i) iStar Goal Model of Toy Example (Adapted from [26]) (ii) Component Spring Model of Toy Example

Fig. 1: Goal Model and Component Spring Model of Toy Example

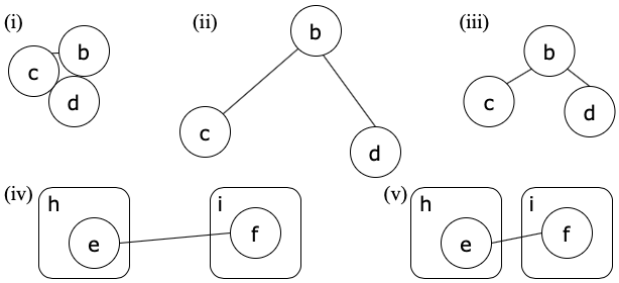



Fig. 2: REPULSION and ATTRACTION on Toy Example

as *curNode*. Next, using the *for* loop (Lines 22–27), ADJUST calculates the forces on each node on *curNode* by calling the REPULSION and ATTRACTION functions on Line 24 and Line 26, respectively. Finally, ADJUST updates the position of *curNode* on Lines 28–30 based on the forces from each element in the model.

One thing that makes FLAAG unique from the generic UCG algorithm, is that ADJUST (see Line 28) allows node types (e.g., goals, tasks) to have individualize gravity values to indicate the hierarchical level of the node in the model.

**REPULSION Function.** The REPULSION function (see Lines 31–38 in Algo. 1) calculates the repulsion between any two given nodes (i.e., *node1* and *node2*). All elements in the graph have repulsion forces with each other. REPULSION uses *Pythagoras' theorem* to calculate the distance between two nodes ($D$, see Line 32). The closer two elements are in space, the greater the repulsion between the elements. We take the repulsion formula from [11], where the repulsion is $-{C_N}^2/D$, where $C_N$ is the same constant used in the ATTRACTION function (below). The $x$ and $y$ components of the force are calculated and returned according to the relative position of the nodes (see Lines 34–38).

Consider nodes b, c, and d in the toy example in Fig. 1(ii). Initially, the nodes are distributed as shown in Fig. 2 (i). REPULSION forces each pair of nodes away from each other, and node b, c, and d are redistributed as shown in Fig. 2 (ii).

**ATTRACTION Function.** Similarly, the ATTRACTION function calculates the attraction between any two given nodes (i.e., *node1* and *node2*). The function takes *n1IsActor* indicating if *node1* is an actor, and two positive reals as constants: $C_N$ and $C_A$ representing the force between connected nodes and actors, respectively. An attraction force only exists between two nodes if there is a link between them. If *node1* is an actor (i.e., *n1Actor* = *true*) then the calculated attraction is the number of links between the actor and *node2* times $D^2/C_A$; otherwise, the attraction between two nodes is simply $D^2/C_N$. $D$ is the distance between nodes (see Line 40). Again, the direction of the force is calculated and returned according to the relative position of the nodes (see Lines 47–51).

Recall nodes b, c, and d from the toy example in Fig. 1(ii). Prior to calling ATTRACTION, the nodes have been separated by the REPULSION function as shown in Fig. 2 (ii). For each pair of connected nodes {b,c} and {b,d}, ATTRACTION exerts a positive force along the link. In this way, the link between c and b will be shortened, similarly for the link between b and d, resulting in the nodes being pulled together as in Fig. 2 (iii).

Similarly, prior to calling ATTRACTION actors h and i are distributed as in Fig. 2 (iv). ATTRACTION counts the number of links that two actors have between the nodes that reside in them. In this case, there is one link between node e and node f, resulting in an attractive force along the line between h and i, which will bring actor h and actor i closer as shown in Fig. 2 (v).

**CHECKCOND Function.** CHECKCOND (see Lines 52–57) takes as input the model, current iteration (*curCtr*) and maximum number of iteration (*expectedIterations*). As mentioned above, CHECKCOND acts as a stop condition for the main loop in Algo. 1. If the maximum number of iterations has not been reached (Line 53), then using CHECKCROSSOVER, CHECKDECOMPODISTANCE and MINIMALSPACE, CHECKCOND determines whether the model has sufficiently converged and satisfies $H_2$–$H_5$ and $H_8$–$H_9$ (see Table I).

## V. DISCUSSION

In this section, we return to our research questions RQ1–RQ3, introduced in Sect. I. To further explore our research questions, we implemented FLAAG, described as Algo. 1 in Sect. IV, in BloomingLeaf, a GORE modeling and analysis tool. The tool supports the Tropos model syntax [29].

### A. Initial Evaluation

First, we address RQ1: How does our algorithm compare with prior automatic-layout algorithms? We rated FLAAG against our heuristics in the rightmost column of Table I. We found that our implementation of some heuristics can be improved and we have not satisfied all our heuristics; nonetheless, we believe FLAAG is more suitable to the task of laying out actor-based goal models than the undirected compound graph or generic force-directed algorithms.

**Algorithm 1** FLAAG: Actor-based Goal Model Layout

**Require:**
Goal Model $M = \langle A, G, R \rangle$ ▷ See Sect. III.
Constants $C_A, C_N, C_M$ ▷ Constants for Actors, Nodes, and Moves.
Maximum Layout Iterations *maxItr* ▷ Optional Timeout
Initial Layout Information *initLay* ▷ Optional coord. for elements in $M$.
**Ensure:**
Final Graph Layout Information

(*actorSet*, *nodeSet*) ← INITIALIZATION($M$, *initLay*)
*curCtr* = 0 ▷ Initializes iteration counter.
**while** CHECKCOND(*curCtr*, *actorSet*, *nodeSet*, *maxItr*) **do**
  **for** *node* ∈ *nodeSet* **do**
    ADJUST(*node*, *actorSet*, *nodeSet*, *False*, $C_A, C_N, C_M$)
  **for** *actor* ∈ *actorSet* **do**
    ADJUST(*actor*, *actorSet*, *nodeSet*, *True*, $C_A, C_N, C_M$)
  *curCtr*++
SETCOORDINATEPOSITIVE(*nodeSet*)
GETSIZEOFACTOR(*actorSet*, *nodeSet*)
CALCULATEACTORPOSWITHREC(*actorSet*)
MOVENODESTOABSPOS(*actorSet*, *nodeSet*)
**return** (*actorSet*, *nodeSet*)

**function** ADJUST(*curNode*, *nodeSet*, *actorSet*, *curIsActor*, $C_A, C_N, C_M$)
  *elemSet* ← *actorSet*
  **if** ¬*curIsActor* **then** ▷ *curNode* is an actor.
    *nodeInA* = {}
    **for** *node* ∈ *nodeSet* **do**
      **if** *node.parent* = *curNode.parent* ∧ *node.id* ≠ *curNode.id* **then**
        *nodeInA.add*(*node*)
    *elemSet* ← *nodeInA*
  **for** *node* ∈ *elemSet* **do**
    **if** *curNode.name* ≠ *node.name* **then**
      *rForces* ← REPULSION(*curNode*, *node*, $C_N$)
      UPDATEFORCES(*curNode*, *rForces*)
      *aForces* ← ATTRACTION(*curNode*, *node*, *isActor*, $C_N, C_A$)
      UPDATEFORCES(*curNode*, *aForces*)
    *curNode.forceY* = *curNode.forceY* + *curNode.gravity*
    *curNode.x* = *curNode.x* + *curNode.forceX* ∗ $C_M$
    *curNode.y* = *curNode.y* + *curNode.forceY* ∗ $C_M$

**function** REPULSION(*node1*, *node2*, $C_N$)
  D ← $\sqrt{(node1.x - node2.x)^2 + (node1.y - node2.y)^2}$
  *calcRep* ← $-{C_N}^2/D$
  *forceX* ← $cos * calcRep$
  *forceY* ← $sin * calcRep$
  **if** *node2.nodeX* < *node1.nodeX* **then** *forceX* = −*forceX*
  **if** *node2.nodeY* < *node1.nodeY* **then** *forceY* = −*forceY*
  **return** [*forceX*,*forceY*]

**function** ATTRACTION(*node1*, *node2*, *n1Actor*, $C_N$, $C_A$)
  D ← $\sqrt{(node1.x - node2.x)^2 + (node1.y - node2.y)^2}$
  **if** *n1Actor* **then** ▷ *node1* is an actor.
    *calcAtt* ← (*node1.linkCount*(*node2*)) ∗ $D^2/C_A$
  **else if** *node1.isConnectedTo*(*node2*) **then**
    *calcAtt* ← $D^2/C_N$
  **else**
    *calcAtt* ← 0
  *forceX* ← $cos * attraction$
  *forceY* ← $sin * attraction$
  **if** *node2.nodeX* < *node1.nodeX* **then** *forceX* = −*forceX*
  **if** *node2.nodeY* < *node1.nodeY* **then** *forceY* = −*forceY*
  **return** [*forceX*,*forceY*]

**function** CHECKCOND(*curCtr*, *actorSet*, *nodeSet*, *expectedIterations*)
  **if** *expectedIterations* ≤ *curCtr* **then return** false
  **else if** ¬CHECKCROSSOVER(*actorSet*, *nodeSet*) **then return** true
  **else if** ¬CHECKDECOMPODIST(*actorSet*, *nodeSet*) **then return** true
  **else if** ¬MINIMALSPACE(*actorSet*, *nodeSet*) **then return** true
  **return** false

TABLE II: Model Validation Data

| Name & Source | Actors | Nodes | Links | Time (ms) | Rating |
|---|---|---|---|---|---|
| Tolls System [14] | 2 | 21 | 24 | 126 | A |
| Goods Aquisition [14] | 2 | 17 | 19 | 113 | A |
| GRAD [30] | 2 | 14 | 18 | 145 | A |
| Scheduler [31] | 3 | 18 | 20 | 104 | B |
| Spadina Plan [32] | 5 | 43 | 55 | 98 | C |
| Spadina Opp [32] | 6 | 38 | 35 | 208 | B |
| Bike Lanes Full [32] | 0 | 30 | 37 | 92 | A |
| Vendor/Service [33] | 3 | 16 | 19 | 180 | B |
| Media Shop [29] | 1 | 23 | 27 | 106 | B |
| Toyota [28] | 0 | 30 | 39 | 120 | C |

Next, we consider RQ2: To what extent does our layout algorithm accommodate goal model syntax and semantics? We took models from a variety of GORE papers in the literature, and recreated them in Tropos using BloomingLeaf. Table II lists the name and source for each model we used in our initial evaluation. When creating models, we did not give elements an initial position to avoid biasing Algo. 1. We asked two students (i.e., *testers*) trained in GORE to assign a letter grade to the layout for each model. To avoid biasing our testers, we told them to first place the layouts into *best-to-worst* buckets and give rationale for their bucketing. We then combined their grades (see *Rating* column in Table II). Models rated as B or C were reported to not support stakeholder understanding. On the whole, testers reported that actors and intentions were placed too far apart, and that some links overlapped decreasing readability. This was a greater issue for larger models.

Finally, we evaluate RQ3: To what extent does our layout algorithm scale to large realistic models? For each model listed in Table II, we recorded run-time data using a 1.1 GHz Intel Core M (Dual-Core) with 8G of RAM. BloomingLeaf produced layouts for each model within 210 ms (see *Time* column in Table II). This is consistent with other analysis operations in BloomingLeaf. We conclude that our implementation of FLAAG is scalable for medium-sized models.

Our ongoing work is looking at improving the spacing between intentions and actors. Overall, these results appear to be promising, but do not provide conclusive evidence and have obvious *threats to validity* (e.g., experimenter expectancies). Instead, they inform our plans for further validation and experimentation. Future work is required to test our approach on large models with more than fifty elements.

### *B. Validation with an Empirical Study*

These are early results requiring discussion of our chosen heuristics among the GORE community. To validate this work, we will complete a full empirical evaluation with GORE experts and stakeholders and then triangulate results among the actor-based languages (i.e., Tropos, GRL, iStar) and tools (e.g., jUCMNav, OpenOME, MUSER, and CreativeLeaf).

In one or more studies, we could do a direct comparison of force-directed algorithms (e.g., GraphViz) with our algorithm for reviewing and updating tasks, tracking how often and in what ways subjects rearrange the model elements. By

developing off of the tracking work in [34], we can determine to what extent modelers need to rearrange elements after layout happens automatically. This leads to our interest in five additional research questions (RQ6-8 were adapted from [12]):

RQ4 Under what circumstances is automatic layout better than manual layout?

RQ5 Which algorithm (e.g., FDA, UCG, FLAAG) produces the best layout for a given goal model?

RQ6 To what extent do modelers gain value in manually laying out goal models?

RQ7 Can participants gain similar benefits to creating models from reviewing and correcting models?

RQ8 To what extent is a visual representation beneficial?

Additional empirical studies will in-turn validate the completeness of the heuristics in Table I.

### C. Summary and Future Work

In this paper, we presented the problem of automatic layout for actor-based goal models. We introduced a set of heuristics to benchmark automatic layout efforts and showed that prior implementations were insufficient for laying out models with actors. We proposed FLAAG, our domain-specific variant of a generic algorithm for undirected compound graphs, and demonstrated the effectiveness of FLAAG in satisfying the heuristics. We also presented evidence for the scalability, in terms of run-times, of our implementation.

Our aim in this early contribution is to gain feedback from the RE community and foster collaborations with other GORE researchers to complete extensive validation of our algorithm across the multiple actor-based languages and tools (see Sect. V-B for details). In addition to implementation improvements and further validation, we will extend our algorithm to allow for expansion and contraction of actor boundaries, as well as model slicing (i.e., split a large model into fragments to facilitate rendering and presentation [14]).

## REFERENCES


[1] J. Horkoff, T. Li, F.-L. Li, M. Salnitri, E. Cardoso, P. Giorgini, J. Mylopoulos, and J. Pimentel, "Taking Goal Models Downstream: A Systematic Roadmap," in *Proc. of RCIS'14*, May 2014, pp. 1–12.

[2] J. Horkoff, F. B. Aydemir, E. Cardoso, T. Li, A. Maté, E. Paja, M. Salnitri, J. Mylopoulos, and P. Giorgini, "Goal-Oriented Requirements Engineering: A Systematic Literature Map," in *Proc. of RE'16*, 2016, pp. 106–115.

[3] J. Horkoff, F. B. Aydemir, E. Cardoso, T. Li, A. Maté, E. Paja, M. Salnitri, L. Piras, J. Mylopoulos, and P. Giorgini, "Goal-Oriented Requirements Engineering: An Extended Systematic Mapping Study," *Requirements Engineering*, vol. 24, no. 2, pp. 133–160, Jun 2019.

[4] H. Estrada, A. M. Rebollar, O. Pastor, and J. Mylopoulos, "An Empirical Evaluation of the *i** Framework in a Model-Based Software Generation Environment," in *Proc. of CAiSE'06*, 2006, pp. 513–527.

[5] A. Mavin, P. Wilkinson, S. Teufl, H. Femmer, J. Eckhardt, and J. Mund, "Does Goal-Oriented Requirements Engineering Achieve Its Goal?" in *Proc. of RE'17*, 2017, pp. 174–183.

[6] R. Matulevičius and P. Heymans, "Visually Effective Goal Models Using KAOS," in *Proc. of ER'07*, 2007, pp. 265–275.

[7] I. Kitzmann and C. Konig, "A Simple Algorithm for Automatic Layout of BPMN Processes," in *Proc. of CEC'09*, 2009.

[8] D. Amyot, A. Shamsaei, J. Kealey, E. Tremblay, A. Miga, G. Mussbacher, M. Alhaj, R. Tawhid, E. Braun, and N. Cartwright, "Towards Advanced Goal Model Analysis with jUCMNav," in *Proc. of ER'12*, 2012, pp. 201–210.

[9] J. Gillain, C. Burnay, I. Jureta, and S. Faulkner, "AnalyticGraph.com: Toward Next Generation Requirements Modeling and Reasoning Tools," in *Proc. of RE'16*, 2016, pp. 341–346.

[10] A. M. Grubb and M. Chechik, "BloomingLeaf: A Formal Tool for Requirements Evolution over Time," in *Proc. RE'18: Posters & Tool Demos*, 2018, pp. 490–491.

[11] S. G. Kobourov, "Force-directed Drawing Algorithms," in *Handbook of Graph Drawing and Visualization*, R. Tamassia, Ed. CRC Press, 2013.

[12] A. M. Grubb, "Reflection on Evolutionary Decision Making with Goal Modeling via Empirical Studies," in *Proc. of RE'18*, 2018, pp. 376–381.

[13] R. Laue and A. Storch, "A Flexible Approach for Validating i* Models," in *Proc. of i* Workshop (iStar'11)*, 2011.

[14] M. Santos, C. Gralha, M. Goulao, J. Araújo, A. Moreira, and J. Cambeiro, "What is the Impact of Bad Layout in the Understandability of Social Goal Models?" in *Proc of RE'16*, 2016, pp. 206–215.

[15] T. M. J. Fruchterman and E. M. Reingold, "Graph Drawing by Force-Directed Placement," *Softw. Pract. Exper.*, vol. 21, no. 11, pp. 1129–1164, 1991.

[16] J. Díaz, J. Petit, and M. Serna, "A Survey of Graph Layout Problems," *ACM Comput. Surv.*, vol. 34, no. 3, pp. 313–356, 2002.

[17] J. Horkoff and E. Yu, "Interactive Goal Model Analysis For Early Requirements Engineering," *Req. Eng.*, vol. 21, no. 1, pp. 29–61, 2016.

[18] G. Grau, X. Franch, and N. A. M. Maiden, "PRiM: An i*-based Process Reengineering Method for Information Systems Specification," *Inf. Softw. Technol.*, vol. 50, no. 1-2, pp. 76–100, 2008.

[19] i* wiki, "jucmnav," http://istar.rwth-aachen.de/tiki-index.php?page=jUCMNav, April 2013.

[20] T. Li, J. Horkoff, and J. Mylopoulos, "Holistic Security Requirements Analysis for Socio-technical Systems," *Softw Syst Model*, vol. 17, pp. 1253–1285, 2018.

[21] Y. Wang, T. Li, H. Zhang, J. Sun, Y. Ni, and C. Geng, "A Prototype for Generating Meaningful Layout of iStar Models," in *Proc. of ER'18: Workshops*, 2018, pp. 49–53.

[22] H. Zhang, T. Li, and Y. Wang, "Design of an Empirical Study for Evaluating an Automatic Layout Tool," in *Proc. of ER'18: Workshops*, 2018, pp. 206–211.

[23] O. Franco-Bedoya, D. Ameller, D. Costal, and L. López, "iStarJSON: A Lightweight Data-Format for *i** Models," in *Proc. of i* Workshop (iStar'16)*, 2016, pp. 37–42.

[24] V. Abdelzad, D. Amyot, S. A. Alwidian, and T. Lethbridge, "A Textual Syntax with Tool Support for the Goal-Oriented Requirement Language," in *Proc. of i* Workshop (iStar'15)*, 2015, pp. 61–66.

[25] T. Li, A. M. Grubb, and J. Horkoff, "Understanding challenges and tradeoffs in istar tool development," in *Proc. of i* Workshop (iStar'16)*, 2016, pp. 49–54.

[26] F. Dalpiaz, X. Franch, and J. Horkoff, "iStar 2.0 Language Guide," *arXiv:1605.07767*, 2016.

[27] U. Dogrusoz, E. Giral, A. Cetintas, A. Civril, and E. Demir, "A Layout Algorithm for Undirected Compound Graphs," *Information Sciences*, vol. 179, no. 7, pp. 980 – 994, 2009.

[28] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani, "Formal Reasoning Techniques for Goal Models," *Journal on Data Semantics*, vol. 1, pp. 1–20, 2003.

[29] P. Giorgini, J. Mylopoulos, and R. Sebastiani, "Goal-oriented Requirements Analysis and Reasoning in the Tropos Methodology," *Engineering Applications of Artificial Intelligence*, vol. 18, no. 2, pp. 159–171, 2005.

[30] B. C. Hu and A. M. Grubb, "Support for User Generated Evolutions of Goal Models," in *Proc. of MiSE'19*, 2019, pp. 1–7.

[31] R. Salay, M. Chechik, J. Horkoff, and A. D. Sandro, "Managing Requirements Uncertainty with Partial Models," *Requirements Engineering*, vol. 18, no. 2, pp. 107–128, Jun. 2013.

[32] A. M. Grubb, "Evolving Intentions: Support for Modeling and Reasoning about Requirements that Change over Time," Ph.D. dissertation, University of Toronto, 2019.

[33] D. Amyot, S. Ghanavati, J. Horkoff, G. Mussbacher, L. Peyton, and E. Yu, "Evaluating Goal Models Within the Goal-Oriented Requirement Language," *Int. J. of Intelligent Sys.*, vol. 25, no. 8, pp. 841–877, 2010.

[34] J. Horkoff, N. A. M. Maiden, and D. Asboth, "Creative Goal Modeling for Innovative Requirements," *Information and Software Technology*, vol. 106, pp. 85 – 100, 2019.