Drawing graphs with mathematical programming and variable neighborhood search

Article *in* Electronic Notes in Discrete Mathematics · April 2017

DOI: 10.1016/j.endm.2017.03.027

2 authors, including:

Haroldo Gambini Santos

Universidade Federal de Ouro Preto

41 PUBLICATIONS **268** CITATIONS

Drawing Graphs with Mathematical Programming and Variable Neighborhood Search

Cézar Augusto N. e Silva [1] Haroldo Gambini Santos [2]

Abstract

In the Graph Drawing problem a set of symbols must be placed in a plane and their connections routed. To produce clear, easy to read diagrams, this problem is usually solved trying to minimize edges crossing and the area occupied, resulting in a NP-Hard problem. Our research focuses in drawing Entity Relationship (ER) diagrams, a challenging version of the problem where nodes have different sizes. Mathematical Programming models for the two solution phases, node placement and connection routing, are discussed and their exact resolution by an Integer Programming (IP) solver is evaluated. As the first phase proved to be specially hard to be solved exactly, a hybrid Variable Neighborhood Search (VNS) heuristic is proposed. Using IP techniques we obtained provably optimal (or close to optimal) solutions for the two different phases, at the expense of a large computational effort. We also show that our VNS based heuristic approach can produce close to optimal solutions in very short times for the hardest part of the solution process. Using either methods we have produced clearly better drawings than existing solutions.

Keywords: Grid Graph Drawing. Mixed Integer Programming. Variable Neighborhood Search.

1 Introduction

In the Graph Drawing problem a set of symbols must be placed in a plane and their connections routed. The objective is to produce aesthetically pleasant, easy to read diagrams. As a primary concern one usually tries to minimize edges crossing, edges' length, waste of space and number of bents in the connections. When formulated with these constraints the problem becomes NP-Hard[4]. In practice many additional complicating requirements can be included, such as non-uniform sizes for symbols[2]. Thus, some heuristics such as the generalized force-direct method and Simulated Annealing [3] have been proposed to tackle this problem. [1] uses a grid structure to approach the Entity-Relationship (ER) drawing problem, emphasizing the differences between ER drawing and the more classical circuit drawing problems. [6] presented different ways of producing graph layouts (e.g.: tree, orthogonal, visibility representations, hierarchic, among others) for general graphs with applications on different subjects.

The ability to automatically produce high quality layouts is very important in many applications, one of these is Software Engineering: the availability of easy to understand ER diagrams, for instance, can improve the time needed for developers to master database models and increase their productivity. In this work we present Integer Programming and Local Search approaches to solve the Entity-Relationship graph drawing problem. To the best of our knowledge this is the first time that Integer Programming is used in a comprehensive graph drawing problem. Our solution approach involves two phases: (i) firstly the optimal placement of entities is solved, i.e.: entities are positioned so as to minimize the distances between connected entities; and (ii) secondly, edges are routed minimizing bends and avoiding the inclusion of connectors too close. We observed that the model of the first phase was significantly harder to solve exactly. Thus, a heuristic based in Variable Neighborhood Search (VNS)[5] is proposed. Our approach allowed us to determine lower and upper bounds for instances generated from real world applications, leading to provably optimal (or very close to optimal) solutions for phases (i) and (ii). This paper is

[1] Departamento de Computação - Instituto de Ciências Exatas e Biológicas - Universidade Federal de Ouro Preto - Campus Universitário Morro do Cruzeiro- 35400-000 Ouro Preto, MG - Brasil
Email: cans5812@gmail.com
[2] Departamento de Computação - Instituto de Ciências Exatas e Biológicas - Universidade Federal de Ouro Preto - Campus Universitário Morro do Cruzeiro- 35400-000 Ouro Preto, MG - Brasil
Email: haroldo@iceb.ufop.br

organized as follows: Section 2 briefly presents the problem, Sections 3 and 4 describe the Integer Programming formulations used in this work for the first and second phases respectively, Section 5 presents the proposed VNS heuristic and finally, Section 6 presents some computational results and conclusions.

2 Problem Definition

This work is specialized in the drawing of Entity Relationship Diagrams (ERs). Entities are database tables with names and a list of attributes and data types. Thus, entities have different space requirements (width and height) to be drawn. Foreign keys are references in one table to another table and define edges between entities. We denote this problem as `ERDraw`. The following input data defines an `ERDraw` instance:

V set of entities, with dimensions in each axis $a \in \mathcal{A} = \{r, c\}$: d_{cv}(width) and d_{rv}(height);

E set of edges indicating entities' relationships;

p minimum padding between entities;

$(\overline{r}, \overline{c})$ the rectangular(non-orthogonal) drawing grid dimensions(respectively rows and columns).

3 MIP Model for Optimal Placement

The first phase of our model aims to optimally place the entities in the grid. Thus, entities positions are used as parameters for the second phase, where edges will be drawn. In this phase one has to determine entities' positions avoiding overlapping and minimizing the distance between connected entities. The main decision variables of this phase are $x_{av} \in \mathbb{Z}^+$, that indicate the absolute position of each entity v considering each axis a (rows and columns).

Let $\Pi = \{(v, w) \in V \times V : v < w\}$, the relative position of entities pairs is determined by auxiliary binary variables $z^{ap}_{vw} : (v, w) \in \Pi$ such that:

$$z^{ap}_{vw} = \begin{cases} 1 \text{ if in axis } a, \text{ entities } v \text{ and } w \text{ do not overlap and } v \text{ is at a relative} \\ \qquad \text{position } p \in \Gamma_a \\ 0 \text{ otherwise} \end{cases}$$

Valid relative positions Γ_a for an axis a are:

$$\Gamma_a = \begin{cases} \Gamma_a = \{l, r\} \text{ if a=}\mathbf{x} \\ \Gamma_a = \{b, t\} \text{ if a=}\mathbf{y} \end{cases}$$

A solution without overlapping must satisfy constraints 1,2 and 5. The objective function for the first phase minimizes firstly, for all $(v, w) \in E$ the Manhattan distance (continuous variables $\delta_{avw}, a \in \mathcal{A}$) between them and secondly, the Manhattan distance to the center(Δ_{av} variables). Finally let $\overline{a}$ be the respective axis size(rows or columns):

$$\textit{Minimize} \sum_{a \in \mathcal{A}} [(\overline{r} + \overline{c}) \times \sum_{(v,w) \in E} \delta_{avw} + \sum_{v \in V} \Delta_{av}]$$

Subject to

$$x_{av} - x_{aw} \leq -p - d_{av} + (1 - z^{al}_{vw}) \times (\overline{a} - 1 + p + d_{av}) \ \forall (v, w) \in \Pi, \forall a \in \mathcal{A} \quad (1)$$

$$x_{av} - x_{aw} \geq p + d_{aw} - (1 - z^{ar}_{vw}) \times (\overline{a} - 1 + p + d_{aw}) \ \forall (v, w) \in \Pi, \forall a \in \mathcal{A} \quad (2)$$

$$\Delta_{av} \geq x_{av} + \frac{d_{av} - \overline{a}}{2} \ \forall v \in V, a \in \mathcal{A} \quad (3)$$

$$\Delta_{av} \geq \frac{\overline{a} - d_{av}}{2} - x_{av} \ \forall v \in V, a \in \mathcal{A} \quad (4)$$

$$\sum_{a \in \mathcal{A}} \sum_{p \in \Gamma} z^{ap}_{vw} = 1 \ \forall v, w \in V : v < w \quad (5)$$

$$\delta_{avw} \geq x_{av} - x_{aw} + \frac{d_{av} - d_{aw}}{2} \ \forall \ (v, w) \in E, \forall a \in \mathcal{A} \quad (6)$$

$$\delta_{avw} \geq x_{aw} - x_{av} + \frac{d_{aw} - d_{av}}{2} \ \forall \ (v, w) \in E, \forall a \in \mathcal{A} \quad (7)$$

4 MIP Model for Optimal Routing

Once a solution for the first phase is available (absolute positions for entities in the grid), the routing of edges between entities is drawn in the grid in the second phase. Only grid nodes not covered by entities are considered. Thus, paths must be found to connect all $(v, w) \in E$ entities.

We modeled the problem of this phase as a multi-commodity binary flow model augmented with some side constraints. The first procedure is to determine which grid nodes can be used to start a path on each one of entities' borders (entity connectors), these sets will represent sources and sinks in the flow constraints. The model minimizes edge crossings by computing penalties related to the over usage of a grid node. Penalizations are also computed when adjacent entity connectors are used or if the same connector is used more than once. Even though this is not a classical multi-commodity flow problem, we observed that this model was usually solved very quickly using our Integer Programming solver.

5 A Heuristic Approach for the first phase

We developed a VNS heuristic using three types of neighborhoods to generate a solution for the first phase of the algorithm. The general structure of the algorithm

Algorithm 1 Hybrid VNS

```
1: k ← 1; j ← 0; Γ ← (1, 2, 3); π* ← ∅
2: costOf(decode(π*)) ← ∞
3: for i = 1 to ρ do
4:     π ← SD(RBFS())                    ▷ Generate and choose ρ permutations
5:     if costOf(decode(π)) < costOf(decode(π*)) then
6:         π* ← π
7: S* ← S ← decode(π*)                   ▷ Initial best known solution
8: for i = 1 to maxIterations do
9:     S ← N_{Γ_k}(S)
10:    if S < S* then
11:        k ← 1; j ← 0; S* ← S
12:    else
13:        k ← k + 1; S ← S*
14:        if k = 4 then
15:            k ← 1; j ← j + 1; S ← shake(S, i, τ, γ)
16:            if j = 2 then
17:                j ← 0; Γ ← shuffle(Γ)        ▷ Shuffle Neighborhoods
18: return S*
```

can be seen in Algorithm 1 and will be discussed in details in this section. Initially, to generate a feasible solution w.r.t. entities overlapping, a multi-start constructive algorithm, which operates over an indirect representation with random keys is employed. These random keys indicate allocation priorities for each entity and generate a permutation of entities $\pi = (\pi_1, \ldots, \pi_{|V|})$ which is then decoded by our constructive algorithm. This algorithm sequentially compacts each entity close to the previously placed entity according to the permutation π.

Since the objective function minimize distances between connected entities, interesting permutations π usually maintain these entities in close positions. To generate different permutations preserving this property we developed a multi-start randomized Breadth First Search (**RBFS** acronym) algorithm which navigates in the entity relationship graph: starting from a random entity, a layered graph is built, where the first layer includes all neighbors of this entity and each additional layer k includes all entities at distance k of the initial entity. While decoding a random keys sequence, nodes in the initial layers are always allocated first. Different random keys sequences are generated by shuffling the priorities on each layer. If the entity graph is not fully connected, this process is repeated for every connected component. Once an entity is selected for placement, different relative positions considering the previously allocated entity (if any) are evaluated. Different directions, where free spaces are searched are considered: $\uparrow, \nwarrow, \searrow, \nearrow, \swarrow, \downarrow$. A greedy strategy is used here, selecting the relative direction where the closest free space found has the best evaluation considering the partial solution.

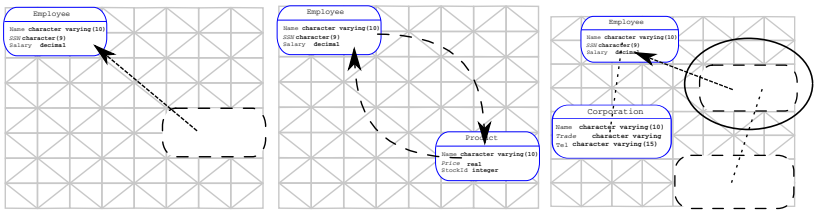

Figure 1. The moves used in our heuristic, are described as they are presented, respectively:

$\mathcal{M}_1$ move an entity to a different grid position;

$\mathcal{M}_2$ swap the position of two entities;

$\mathcal{M}_3$ move an entity to a different grid position, relocating all its direct neighbors to the same relative position.

For each one of the ρ (=20 in our experiments) sequences π built, a steepest descent (**SD** acronym) local search is performed. The neighborhood explored is composed by all alternative sequences π' which are generated by inverting subsequences of π of size $\theta \in \{2, \ldots, |V|\}$ starting in positions $\delta \in \{1, \ldots, |V| - 1\}$, such that $\theta + \delta \leq |V|$.

The following neighborhoods $\mathcal{N}_1, \mathcal{N}_2$ and $\mathcal{N}_3$ are explored, operating over a direct solution representation. These neighborhoods are composed, respectively, by all solutions which can be generated using the following types of moves, all depicted in Figure 1.

These neighborhoods are explored in a sequential VND algorithm. The exploration order of the neighborhoods is periodically changed (**shuffle**, line 17).

After a local optimum with respect to all these neighborhoods is reached, the **shake** procedure is applied (line 15). As in the Skewed VNS method, our shake procedure uses a modified objective function to improve diversification without deteriorating too much the solution quality. The shake procedure starts by selecting a random entity v. Then, a sample of $\tau = 10$ solutions generated by random permutations of three moves m_1, m_2, m_3, each one including v, is evaluated and the best one according to the modified objective function is selected. This objective function includes a transition based memory metric, to increase the probability of selecting neighbors generated by modifying solution components which remained fixed for more iterations. Thus, if a candidate solution s' was generated by moving an entity v which was moved m_v times in previous iterations of the search, its evaluation is then $f'(s') = f(s') \times (1 - \gamma \times (1 - \frac{m_v}{k}))$, where $f(s')$ is the original objective function, k is the current iteration and γ (=20 in our experiments), controls the maximum allowable modification in the original objective function. To prevent a quick return to the same previously visited local optimum, the next VND iteration is executed removing from all neighborhoods solutions where v is moved.

		MIP Approach				Heuristic Approach				τ
Id	$\|V\|,\|E\|$	Obj	dGap	LB	Time	Obj	SD	hGap	Time	
1	6.5	2653.5	0.0	2653.5	4s	2653.5	0.0	0.0	2s	2s
2	9.12	8094.5	0.0	8094.5	4.6m	8364.0	1.8	3.3	5s	50s
3	8.11	7118.5	0.0	7118.5	2.2m	7241.3	92.9	1.7	4s	50s
4	13.13	7602.0	20.6	6303.0	2h	7602.0	51.5	0.0	9s	-
5	12.12	6243.0	27.4	4901.5	2h	6344.8	53.2	1.6	7s	7m
6	13.15	21530.0	35.9	15842.0	2h	21638.8	550.6	0.5	18s	40m
7	19.22	24540.5	57.2	15609.0	2h	23573.5	455.9	0.0	43s	-
8	18.12	17404.0	42.2	12235.0	2h	18662.9	229.6	7.2	62s	4.3m
9	17.26	30012.0	60.1	18744.0	2h	30278.4	775.9	0.9	23s	1.4h
10	14,16	18469.0	16.2	15897.0	2h	19274.6	632.1	0.0	15s	42m
Gap Summaries		**dGap**	**Max**	**Min**	**Avg**	**hGap**	**Max**	**Min**	**Avg**	
			60.1	0.0	25.9		7.2	0.0	1.5	

Figure 2. A Comparison between the exact and heuristic methods. The first two columns in the Table are, respectively, the instance id and its dimension. Solution costs are informed in the OBJ columns, the best solution for the MIP approach is included and the average solution costs obtained with the VNS heuristic (along with the standard deviation SD) also appear. LB indicates the lower bound computed by the MIP solver. The duality gap $dGap = (\frac{OBJ}{LB} - 1) \times 100$ is also included. For the heuristic approach the gap is computed considering the best known solution OBJ^*: $hGap = (\frac{OBJ}{OBJ^*} - 1) \times 100$. The last column($\tau$) shows the time consumed by the exact solver to generate a solution with cost better or equal to the average cost generated by the VNS heuristic.

6 Computational Experiments and Conclusions

To evaluate the quality of the proposed methods, a set of benchmark instances was compiled from real ER diagrams available on the world-wide-web.

The proposed heuristic was implemented in `C` and experiments ran in a computer equipped with 7.7 GiB RAM, Intel®Core™i5-3210M CPU @ 2.50GHz × 4 running openSUSE 13.2 64-bit. As MIP solver we used the COIN-OR Branch-and-Cut solver. Both heuristic and MIP solver were executed using all available cores.

Figure 2 includes a table with results obtained using the Integer Programming formulation and the proposed heuristic. The stopping criterion for the heuristic was $500 \times log(|V|)$ iterations, while the MIP solver had a time limit of 2 hours. The last column indicates the time needed by the MIP solver to produce a solution with cost smaller or equal than the solution cost produced by the heuristic. As it can be seen, for a larger instances the heuristic can provide in a few seconds a solution with quality that is only reached after over four minutes of processing by the MIP

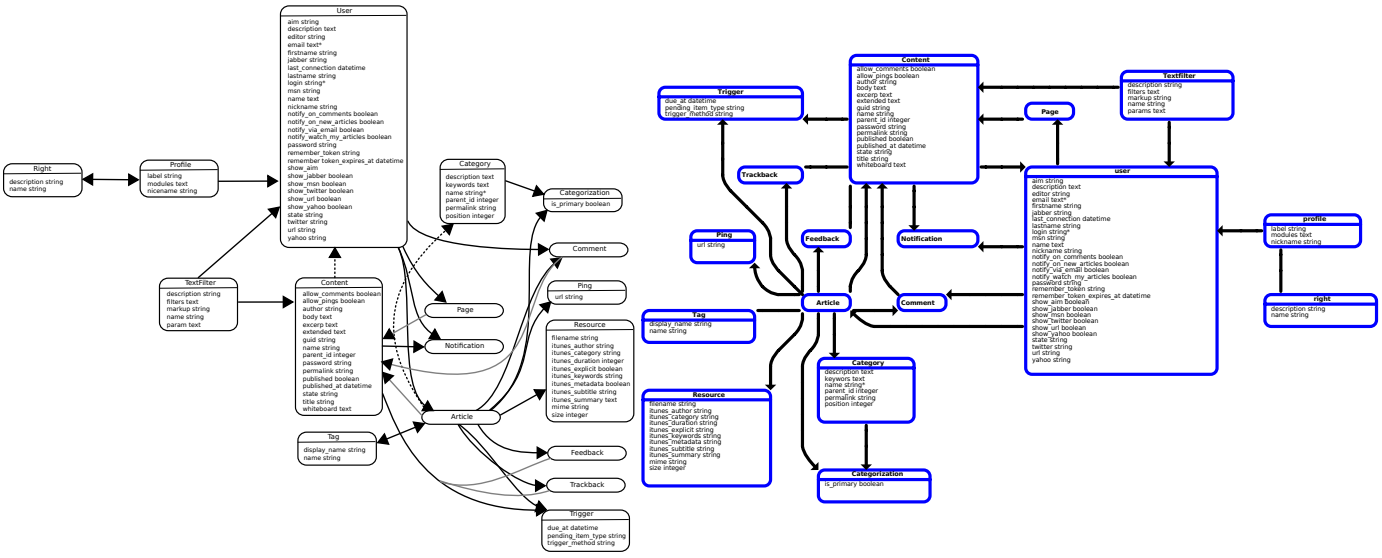

Figure 3. **On the left:** An Entity-Relationship diagram found on the web (http://goo.gl/uBlYG, May 15th, 2016) redrawn as a vectorial image for better resolution, preserving conflicts. **On the right:** The solution produced by our heuristic, using the VNS for the first phase and the Integer Programming model to route the edges.

solver.

References

[1] C.Batini, R., M.Talamo, *Computer aided layout of entity relationship diagrams*, The Journal of Systems and Software **4** (1984), pp. 163–173.

[2] David Harel, Y. K., *Drawing graphs with non-uniform vertices*, in: *Proceedings of the Working Conference on Advanced Visual Interfaces*, ACM, 2002, pp. 157–166.

[3] Davidson, R. and D. Harel, *Drawing graphs nicely using simulated annealing*, ACM Transactions on Graphics **15** (1996), pp. 301–331.

[4] Johnson, D., *The NP-completeness column: An ongoing guide*, J. Algorithms **3** (1982), pp. 88–89.

[5] Nenad Mladenović, P. H., *Variable neighborhood search*, Computers & Operations Research **24** (1997), pp. 1097–1100.

[6] Roberto Tamassia, C. B., Giuseppe Di Battista, *Automatic graph drawing and readability of diagrams*, IEEE Transactions On Systems, Man and Cybernetics **18** (1988), pp. 61–79.