General-Purpose Layout Algorithm for GroIMP : Mathematical Background

The general-purpose layout algorithm of GroIMP includes an energy layout which uses mainly two optimization methods : a modified-Newton algorithm and a simulated annealing algorithm. This is a short summary of the mathematical points involved.

This energy layout is mainly inspired from [1], [2] and [3]. More details concerning the use of numerical optimization for graph drawing can be found in [4], and precisions concerning numerical optimization methods in general in [5].

Energy Layout : overview

By analogy to atoms interactions and crystallography, the main idea of an energy layout is to define an energy function taking the position of the nodes as parameter, and describing the quality of the layout (the higher the energy, the worse the layout is). Thus the problem of finding a good layout configuration is transformed into an optimization problem. We can without loss of generality assume that the graph to layout is connected (if it is not the case, it can be divided into several connected components, which can be in turn layouted). No further assumptions are needed.

The energy is defined as the sum of terms which can be classified in two types :

- "edge energy" : any pair of linked nodes adds a spring-like term $E^e = \frac{k}{2} \cdot (d - l_0)^2$ (d being the length of the edge, k the spring constant and l_0 the rest length of the spring) ;
- "node energy" : any pair of two different nodes adds a repulsive term $E^n = \frac{\lambda}{d}$ (λ being a constant and d the distance between the two nodes).

Thus energy function is defined as follow, with n the number of nodes, (x_i, y_i) the coordinate of any node i, d_{ij} the distance between any pair (i, j) of nodes, and d_e the length of the edge e :

$$\begin{array}{rccl} E^{tot} : & D & \longrightarrow & \mathbb{R} \\ & (x_1, y_1, ..., x_n, y_n) & \longrightarrow & \displaystyle\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{\lambda}{d_{ij}} + \sum_{e:edges} \frac{k}{2} \cdot (d_e - l_0)^2 \end{array}$$

With :

$$D = \mathbb{R}^{2 \cdot n} \backslash \{(x_1, y_1, ..., x_n, y_n) \in \mathbb{R}^{2 \cdot n} / \exists (i, j) \in [\![1, n]\!]^2, i \neq j, st. (x_i, y_i) = (x_j, y_j)\}$$

Remark that, given any node i_0, the energy can be rewritten as $E^{tot} = E^{tot}_{i_0} + E^{tot}_{n-\{i_0\}}$ with $E^{tot}_{i_0}$ a term taking into account all dependencies of E^{tot} into the node i_0 and $E^{tot}_{n-\{i_0\}}$ a term independent of the node i_0 :

$$E_{i_0}^{tot} = \sum_{j \neq i_0} \frac{\lambda}{d_{i_0 j}} + \sum_{\{e : edges / i_0 \in e\}} \frac{k}{2} \cdot (d_e - l_0)^2$$

$$E_{n-\{i_0\}}^{tot} = \sum_{i=1\,;\, i \neq i_0}^{n-1} \sum_{j=i+1\,;\, j \neq i_0}^{n} \frac{\lambda}{d_{i_0 j}} + \sum_{\{e : edges / i_0 \notin e\}} \frac{k}{2} \cdot (d_e - l_0)^2$$

The $E_{i_0}^{tot}$ term will therefore be referred to as the "energy of the node i_0", however one should note that it is in fact an interaction term and does not strictly "belongs" to the node i_0. In particular, the total energy of the layout is **not** the sum of the energy of each node. Whenever the position of only one node i_0 is modified, this decomposition enables one to recompute the total energy of the layout in linear time – as only the $E_{i_0}^{tot}$ term has to be recomputed – rather than in quadratic time.

A priori finding a minimum for the energy function is a high-dimension optimization problem, as it is defined on a $2 \cdot n$-dimensions space. However, a heuristic is used to reduce the complexity of the problem : the energy is considered to be a function of the position of one node only – the other nodes being "frozen" – and the position of this node is optimized accordingly (then the node which is moving is changed, and this process iterated until convergence). Therefore the optimization space is now of dimension 2.

In practice, two methods are used to optimize the energy function : a modified-Newton algorithm, and a simulated annealing algorithm.

Modified-Newton algorithm : principle

When optimizing the layout with respect to the position of the node i_0 and considering the other nodes as "frozen", the $E_{n-\{i_0\}}^{tot}$ term in the energy function is constant, and, as so, it is useless to take it into account. Thus, the function to optimize is $E_{i_0}^{tot}$, which is $C^\infty(D)$.

Starting from an initial point x, the general principle of the modified-Newton algorithm is the following :

1. choose a descent direction (that is a direction in which the energy function is decreasing) : the direction chosen here satisfies $B \cdot p = -\overrightarrow{\nabla} E_{i_0}^{tot}$, with B a positive definite matrix based on the Hessian of $E_{i_0}^{tot}$;
2. choose a step-length α for this direction : an Armijo backtracking line-search is implemented ;
3. do $x \equiv x + \alpha \cdot p$, and iterate this scheme until reaching a minimum (in practice : until the gradient is small enough)

See the detailed section at the end of this document for more precisions.

Simulated Annealing

Simulated Annealing is a general meta-heuristic optimization method. It is useful in this problem in order to find a "good" minimum (*i.e.* a "good" layout) as classic optimization methods such as the modified-Newton algorithm implemented converge towards local minima, and the energy function to optimize indeed presents many local minima far from any "good" configuration. The implemented version is largely based on the algorithm described in [3] . The principle of the simulated annealing optimization is the following :

1. define an initial "high temperature" T for the system, a cooling schedule for this "temperature" and a maximum number of steps ;
2. starting from some configuration C_0 of energy E_0, randomly modify it, and assess the new energy ;
3. if the energy E_1 of the new configuration C_1 is less than the initial energy E_0 , then do $C_0 \equiv C_1$;
4. else do $C_0 \equiv C_1$ with a probability $e^{\frac{E_0 - E_1}{T}}$;
5. apply the cooling schedule, and increment the number of steps. If the maximum number is not reached, go back to step 2 ;

Modified-Newton algorithm : details and calculations

We have :

$$E_{i_0}^{tot} = \sum_{j \neq i_0} \frac{\lambda}{d_{i_0 j}} + \chi_{i_0 j} \cdot \frac{k}{2} \cdot (d_{i_0 j} - l_0)^2$$

with : $\chi_{i_0 j}$ the number of edges between the nodes i_0 and j, and :

$$d_{i_0 j} = \sqrt{{\Delta_{i_0 j}^x}^2 + {\Delta_{i_0 j}^y}^2} \text{ and } \begin{cases} \Delta_{i_0 j}^x = & x_{i_0} - x_j \\ \Delta_{i_0 j}^y = & y_{i_0} - y_j \end{cases}$$

Gradient :

$$\overrightarrow{\nabla} E_{i_0}^{tot}(x_{i_0}, y_{i_0}) = \begin{pmatrix} \frac{\partial E_{i_0}^{tot}}{\partial x_{i_0}}(x_{i_0}, y_{i_0}) \\ \frac{\partial E_{i_0}^{tot}}{\partial y_{i_0}}(x_{i_0}, y_{i_0}) \end{pmatrix} = \begin{pmatrix} \sum_{j \neq i_0} \left(\frac{-\lambda}{{d_{i_0 j}}^2} + \chi_{i_0 j} \cdot k \cdot (d_{i_0 j} - l_0) \right) \cdot \frac{\Delta_{i_0 j}^x}{d_{i_0 j}} \\ \sum_{j \neq i_0} \left(\frac{-\lambda}{{d_{i_0 j}}^2} + \chi_{i_0 j} \cdot k \cdot (d_{i_0 j} - l_0) \right) \cdot \frac{\Delta_{i_0 j}^y}{d_{i_0 j}} \end{pmatrix}$$

Hessian matrix :

$$H_{E_{i_0}^{tot}} = \begin{pmatrix} \dfrac{\partial^2 E_{i_0}^{tot}}{\partial^2 x_{i_0}}(x_{i_0}, y_{i_0}) & \dfrac{\partial^2 E_{i_0}^{tot}}{\partial x_{i_0} \partial y_{i_0}}(x_{i_0}, y_{i_0}) \\ \dfrac{\partial^2 E_{i_0}^{tot}}{\partial y_{i_0} \partial x_{i_0}}(x_{i_0}, y_{i_0}) & \dfrac{\partial^2 E_{i_0}^{tot}}{\partial^2 y_{i_0}}(x_{i_0}, y_{i_0}) \end{pmatrix}$$

with :

$$\begin{cases} \dfrac{\partial^2 E_{i_0}^{tot}}{\partial^2 x_{i_0}}(x_{i_0}, y_{i_0}) = & \displaystyle\sum_{j \neq i_0} \frac{1}{{d_{i_0 j}}^3} \cdot \left({\Delta_{i_0 j}^x}^2 \cdot \left(\frac{3 \cdot \lambda}{{d_{i_0 j}}^2} + \chi_{i_0 j} \cdot k \cdot l_0 \right) - \lambda \right) + \chi_{i_0 j} \cdot k \cdot \left(1 - \frac{l_0}{d_{i_0 j}} \right) \\ \dfrac{\partial^2 E_{i_0}^{tot}}{\partial^2 y_{i_0}}(x_{i_0}, y_{i_0}) = & \displaystyle\sum_{j \neq i_0} \frac{1}{{d_{i_0 j}}^3} \cdot \left({\Delta_{i_0 j}^y}^2 \cdot \left(\frac{3 \cdot \lambda}{{d_{i_0 j}}^2} + \chi_{i_0 j} \cdot k \cdot l_0 \right) - \lambda \right) + \chi_{i_0 j} \cdot k \cdot \left(1 - \frac{l_0}{d_{i_0 j}} \right) \\ \dfrac{\partial^2 E_{i_0}^{tot}}{\partial x_{i_0} \partial y_{i_0}}(x_{i_0}, y_{i_0}) = & \dfrac{\partial^2 E_{i_0}^{tot}}{\partial y_{i_0} \partial x_{i_0}}(x_{i_0}, y_{i_0}) = \displaystyle\sum_{j \neq i_0} \left(\frac{3 \cdot \lambda}{{d_{i_0 j}}^2} + \chi_{i_0 j} \cdot k \cdot l_0 \right) \cdot \frac{\Delta_{i_0 j}^x \cdot \Delta_{i_0 j}^y}{{d_{i_0 j}}^3} \end{cases}$$

Descent Direction (dependence in (x_{i_0}, y_{i_0}) left out thereafter):

$$\text{with } \; H_E = \begin{pmatrix} a & b \\ b & c \end{pmatrix},$$

H_E definite positive *iif* $(a \cdot c - b^2) > 0$, $a > 0$ and $c > 0$. If it's the case, the descent direction p is given by $H_E \cdot p = -\overrightarrow{\nabla} E$ (Newton Method).

In practice, to ensure that H is "sufficiently" positive definite, we use the matrix B ($\rightarrow$ **Modified-Newton Method**), defined as (λ_1 and λ_2 being the eigenvalues of H) :

$$B \cdot p = -\overrightarrow{\nabla} E \text{ with } \begin{cases} H = Q \cdot D \cdot Q^T \text{ diagonalization of H by orthogonal matrices} \\ B = Q \cdot M \cdot Q^T \\ M = \begin{pmatrix} \max(|\lambda_1|, \epsilon) & 0 \\ 0 & \max(|\lambda_2|, \epsilon) \end{pmatrix} \text{ with } \epsilon \text{ "small"} \end{cases}$$

ie :

$$\boxed{p = -(Q^T \cdot M^{-1} \cdot Q) \cdot \overrightarrow{\nabla} E}$$

NB : this possible because H_E being a real symmetric matrix, it is diagonalizable by orthogonal matrices.

Diagonalization :

$$D = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} \text{ with } \lambda_{1,2} = \frac{a + c \pm \sqrt{\Delta}}{2} \text{ eigenvalues, and } \Delta = (a - c)^2 + 4 \cdot b^2$$

and :

$$Q = \begin{pmatrix} \dfrac{b}{n_1} & \dfrac{b}{n_2} \\ \dfrac{\lambda_1 - a}{n_1} & \dfrac{\lambda_2 - a}{n_2} \end{pmatrix} \text{ with } n_{1,2} = \sqrt{b^2 + (\lambda_{1,2} - a)^2}$$

$$Q^T \cdot M^{-1} \cdot Q = \begin{pmatrix} \dfrac{b^2}{{n_1}^2 \cdot v_1} + \dfrac{(\lambda_1 - a)^2}{{n_1}^2 \cdot v_2} & \dfrac{b^2}{n_1 \cdot n_2 \cdot v_1} + \dfrac{(\lambda_1 - a) \cdot (\lambda_2 - a)}{n_1 \cdot n_2 \cdot v_2} \\ \dfrac{b^2}{n_1 \cdot n_2 \cdot v_1} + \dfrac{(\lambda_1 - a) \cdot (\lambda_2 - a)}{n_1 \cdot n_2 \cdot v_2} & \dfrac{b^2}{{n_2}^2 \cdot v_1} + \dfrac{(\lambda_2 - a)^2}{{n_2}^2 \cdot v_2} \end{pmatrix}$$

with $v_1 = \max(|\lambda_1|, \epsilon)$, and $v_2 = \max(|\lambda_2|, \epsilon)$

Backtracking-Armijo linesearch :

Starting from a point x_k, with a descent direction p_k :

Let : $\alpha_{init} \in \mathbb{R}_+^*$, $(\beta, \tau) \in]0;1[^2$, $\alpha = \alpha_{init}$

Do :

1. Let : $\tilde{x} = x_k + \alpha \cdot p_k$:
2. If $E(\tilde{x}) \leq E(x) + \beta \cdot \alpha \cdot < p_k \mid \overrightarrow{\nabla} E >$, then $x_{k+1} = \tilde{x}$
3. Otherwise, let $\alpha = \tau \cdot \alpha$ and iterate this scheme until the above condition is satisfied.

Bibliography

[1] T. M. J. Fruchterman and E. M. Reingold, "Graph drawing by force-directed placement," *Softw. Pract. Exper.*, vol. 21, pp. 1129–1164, November 1991.

[2] T. Kamada and S. Kawai, "An algorithm for drawing general undirected graphs," *Inf. Process. Lett.*, vol. 31, pp. 7–15, April 1989.

[3] R. Davidson and D. Harel, "Drawing graphs nicely using simulated annealing," *ACM Trans. Graph.*, vol. 15, pp. 301–331, October 1996.

[4] N. I. M. Gould and S. Leyffer, *An introduction to algorithms for nonlinear optimization*, 2003.

[5] D. Tunkelang, "A numerical optimization approach to general graph drawing," tech. rep., 1999.