Optimizing Placement in Mixed-Size Circuits

1st Arunachalam Ramesh
B.Tech, Engineering Physics
Indian Institute of Technology, Madras
ep22b004@smail.iitm.ac.in

2nd Dr S Ganga Prasath
Assistant professor, Dept. of AMBE
Indian Institute of Technology, Madras
sgangaprasath@smail.iitm.ac.in

3rd Dr Ramprasath S
Assistant professor, Dept. of EE
Indian Institute of Technology, Madras
ramprasath@ee.iitm.ac.in

***Abstract*—The performance and convergence of downstream operations are greatly impacted by mixed-size placement, making it a crucial step in the physical design flow of integrated circuits. Purely analytical approaches had trouble with speed and scalability, whereas traditional approaches, such as stochastic methods, frequently led to poor convergence and increased design complexity. Current approaches turn the placement procedure into an optimization problem by utilizing pseudo-energy or potential functions between components to minimize wire length and handle density restrictions. With optimization guided by a natural gradient framework, this study investigates ways to reorganize components so that their density distribution meets ideal goals while minimizing wire length.**

I. Introduction

In the physical design cycle of integrated circuits, placement comes after floor-planning and has a big impact on downstream process convergence and efficiency. Millions of cells and thousands of macros of various sizes are found within a rectilinear boundary in contemporary integrated circuits; these cells are usually restricted to rows with fixed or integer-multiple heights. Every cell will line up with these rows without overlapping if the arrangement is correct. This issue was addressed by earlier stochastic techniques like simulated annealing [1], but they were limited by their inability to scale and vulnerability to less-than-ideal local minima. As a result, a two-step strategy was adopted: (a) global placement to reduce wire length in a coarse manner, and (b) detailed placement and legalization to guarantee viability. In the global placement stage, the approximate solution to the wire-length reduction problem as a non-linear constrained optimization is used to determine the coarse placements of the cells. In order to reach a satisfactory solution (often a local minimum) in the vicinity of the global placement, a thorough placement/legalization process is then carried out, assigning legal coordinates to each cell. It has been demonstrated that several analytical formulations that capture the wire-length and distribution constraints—based on a pseudo-energy/potential between various cells—identify better solutions than the stochastic ones. The successful ones approximate the cells as: (a) a system of point masses and linear springs (from simple harmonic potential) with engineered spring constants [2], and (b) the distribution of point charges (from Coulomb potential) in an electric field [3]. A smooth, differentiable function is utilized to approximate the wire length in all analytical formulations, and this function is then employed as the minimization objective. A restriction on density stops the naive solution of all point masses/charges collapsing into a single homogeneous cluster. Density is defined as the ratio of the occupied cell area per unit area of the placement region. The charges and masses are redistributed throughout the placement zone by the repulsive force between the cells, which is represented by an electrostatic potential (other limitations like routing congestion can be treated similarly). Though the solution from energy-based techniques as they exist today have improved considerably from the previous generation, three important questions remain unanswered: (a) why do specific interaction potentials between cells work better than others, (b) is there a potential which generalizes the different potentials (such as harmonic potential, electrostatic Coulomb potential), (c) are there general ways of finding solutions with minimal influence of the legalization procedure. In this project we propose to address some of these questions by (1) effect of different force fields on the mass-spring system , (2) developing a generalized force field with tunable interaction to control the strength and length-scale of interaction to reduce the influence of the legalization procedure on the ultimate solution, (3) use the Mahalanobis [4] Metric to accommodate natural convergence in simple mass-spring system, or graph node system and eventually making it serve our purpose. Also we tell our potential to be a very generic one so that it can also be extended to force-field related problems, since there is repulsive force field being set across the space, with attractive force fields between nodes in system.

Indian Institute of Technology, Madras.

II. Theoretical background

In this section, we go through the mathematical framework and concepts used in this project. It will contain the primitive methods we used in course of our project as well as the the ones which we apply in our final framework. The methods added here include standard minimization methods, from say the Gradient Descent, and then moving into Natural gradients and eventually getting into the core of the idea, **differential simulations** by starting with Auto Differentiaion.

A. Gradient Descent Method

First, we briefly review gradient descent. Suppose we're interested in minimizing a function $f(x) : \mathbb{R}^p \to \mathbb{R}$ with respect to $x \in \mathbb{R}^p$. Recall that the gradient $\nabla f(x)$ is the vector of partial derivatives with respect to each of its arguments:

$$\nabla f(x) = \left[\frac{\partial}{\partial x^{(1)}} f(x), \frac{\partial}{\partial x^{(2)}} f(x), \ldots, \frac{\partial}{\partial x^{(p)}} f(x)\right]^\top$$

where $x^{(i)}$ is the i-th element of x [4].

Gradient descent is an algorithm that iteratively takes a step in the direction of the negative gradient. Let x_t be the value of x on the t-th iteration. Then the gradient descent update is given by:

$$x_{t+1} = x_t - \gamma \nabla f(x_t)$$

where γ is a learning rate parameter.

This is called a Static Gradient, where in the gradient mentions, represents, and accounts for only local information about the function at that specific point for calculation. It does not account for how the function behaves beyond that point until the parameters are adjusted and a new gradient is computed.

B. Natural Gradient

The natural gradient is a generalization of a typical gradient that accounts for the curvature of the underlying manifold on which optimisation is performed. This is generally applied to improve the performance of the gradient as living on the manifold guarantees that we do not miss minima and further the step-size between multiple gradient steps is dynamically adapted based on the local curvature of the underlying manifold [4].

Let us consider θ_t be the vector of parameter, and $\mathbf{f}(\theta_t)$ be the function to be minimized. The equation to be solved as per the steepest descent looks like

$$\theta_{t+1} = \min\left[f(\theta_t) + \nabla f(\theta_t) \cdot (\theta - \theta_t) + D(\theta, \theta_t)\right] \tag{1}$$

where $D(\theta, \theta_t)$ depends on the method used.

Here we implement mahalanobis Metric[4], where we change our metric to a non-Euclidean form.

$$D(\theta, \theta_t) = (\theta - \theta_t)\mathbb{A}(\theta - \theta_t)^\mathsf{T} \tag{2}$$

This can be written in terms of $\mathbf{x}$ for $\mathbf{x}, \mathbf{x}^{'} \in \mathbb{R}^n$ as,

$$D(\mathbf{x}, \mathbf{x}^{'}) = (\mathbf{x} - \mathbf{x}^{'})\mathbb{A}(\mathbf{x} - \mathbf{x}^{'})^\mathsf{T} \tag{3}$$

where $\mathbb{A}$ is a Positive Semidefinite matrix, which implies it is symmetric and has a non-negative eigenvalues matrix of the dimensions $p\ \times p$ matrix, where p is the dimension of $\mathbf{x}$. The update equation equation becomes like,

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \mathbb{A}^{-1}\nabla f(\mathbf{x}_t). \tag{4}$$

C. Manhattan Distances and Log-Sum Exponent Methods

1) Manhattan Distances: Since our entire treatment of the area where components are going to be placed is characterized as a place of grids, the way we need to characterize our potential is not using Euclidean distances, but using something known as Manhattan Distance. This assumes that the distance as the units measured in only write angular paths.

$$Distance = |x_i - x_j| + |y_i - y_j|$$

Since our requirement is for a grid-based environment it is better to focus on aligning with it and trying to implement a solution based on treating a normal environment as Manhattan-based.

2) Log Sum Exponent Method: Our final arrangement should not only be uniformly dense, but also should have the minimum wire length running throughout our circuit. And hence, we try to minimize it by creating a loss function, that is known as **HPWL** (Half Perimeter Wire Length).

$$\mathit{HPWL} = \max_{i,j}(|\mathbf{x}_i - \mathbf{x}_j| + |\mathbf{y}_i - \mathbf{y}_j|)$$

However, this is not a minimizable function using the other gradients. To make it smooth and differentiable, we use softer maximums, and one such method is **LSE**(Log-Sum Exponential), where we

$$\mathit{HPWL} = \gamma(\log(\Sigma e^{|\mathbf{x}_i - \mathbf{x}_j|/\gamma}) + \log(\Sigma e^{|\mathbf{y}_i - \mathbf{y}_j|/\gamma})) \tag{5}$$

Here the γ is used to make the maximum term dominate more so that it will nullify the other terms that have a lesser value, and when it goes into a logarithm as an exponent, only the maximum term comes back.

D. Auto-Differentiation

Auto-differentiation is a method used to calculate derivatives in forward problems or differential simulations. It involves a systematic approach to compute gradients effectively.

1) Initialization: A loss function $s(\mathbf{x}(t_1))$ is defined, which evolves based on the system dynamics:

$$s(\mathbf{x}(t_1)) = s(\mathbf{x}(t_0)) + f(t), \tag{6}$$

where t_0 is the final time point of interest. The loss function depends on the final position $\mathbf{x}(t_0)$, which defines the trajectory.

2) State-Space Construction: The state-space representation and the forward Ordinary Differential Equation (ODE) governing the system dynamics are constructed as:

$$\dot{\mathbf{x}} = f(\mathbf{x}(t)), \tag{7}$$

where f is the driving parameter of the system. The state-space vector is given by:

$$\mathbf{x}(t) = \begin{bmatrix} x_1 \\ \dot{x}_1 \\ \theta \end{bmatrix}. \tag{8}$$

3) Gradient Calculation: The objective is to compute the gradient of the loss function with respect to the state-space vector $\mathbf{x}$ at the final time point t_0, based on its value at t_1.

4) Gradient Propagation Methods: Two primary methods can be used to propagate the gradients. In the forward ODE method, the driving force equation $f(\mathbf{x}(t))$ is used to predict future state-space conditions. Alternatively, the reverse ODE method involves computing the time derivative of the gradient and propagating it backward in time to predict the gradient itself.

5) Implementation: In practice, these methods are implemented using discrete forward propagation of the loss function and reverse propagation of its gradient. Forward propagation incrementally computes the loss function, while reverse propagation iteratively computes the gradient updates. It uses **Discrete Adjoint Method** which can be briefed as follows. **Forward Time Stepping**

$$s(\mathbf{x}^{t+n}) = s(s(s......^{ntimes}(s(\mathbf{x}^{t})).....) \tag{9}$$

Reverse Time Stepping, where $\mathbf{f}$ is the quantity responsible for propagation of $\mathbf{x}$.

$$\frac{\partial s}{\partial \mathbf{x}}^{T}(t-1) = \frac{\partial \mathbf{f}}{\partial \mathbf{x}}^{T} \frac{\partial s}{\partial \mathbf{x}}^{T}(t) \tag{10}$$

III. Dynamics of a Spring Mass system

A. Getting a basic formal solution

A spring-mass system with one degree of freedom is described the following force balance equation:

$$m\ddot{x} + \eta\dot{x} + kx = f, \tag{11}$$

where m is the mass, η denotes the damping constant, k is the spring constant, and $f_o\cos(\omega t)$ represents the external periodic forcing with oscillating frequency ω.

Excluding the fact that mass goes negligible and $x(t)$ goes to some x^* and if we make a general solution using the Green's Function (which gives solution to an impulsive force), we get,

$$g(t) = e^{-\eta t/2m}(e^{dt/2m} - e^{-dt/2m})/d \tag{12}$$

where

$$d = \sqrt{\eta^2 - 4km} \tag{13}$$

and $g(t)$ is the Green's Function. The formal solution goes on to be

$$\mathbf{x}(t) = \int_0^t g(t-t')q(t')\,dt' \tag{14}$$

where $q(t)$ is meant to be the force in this case.

B. Solutions for different driving forces

Formal Solutions for various outputs are shown below in the Figures 3.1,3.2,3.3,3.4 .

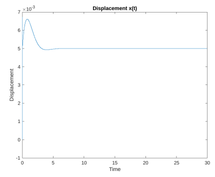


Fig. 1. $q(t) = e^{-t}sin(t) + 1$

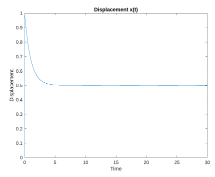


Fig. 2. $q(t)$ = $100e^{-100t} + sin(t)/20 + 100$

C. Finding Control Input - PD Control

The approach to finding the control input follows the path of Non-Linear Programming (NLP). This technique is used to solve the optimization problem defined by:

$$\lim_{t\to\infty}\left(x^*(T) - x_0\right) = 0, \tag{15}$$

where $x^*(t)$ is the predicted solution for displacement, T is the maximum time targeted for convergence, and x_0 is the desired target displacement.

To solve this problem, the same differential equation as Equation (3.1) is utilized. However, instead of $f_0\cos(wt)$, a control term $q(t)$ is introduced. Considering that the right-hand side must align with the left-hand side, a control input is constructed as:

$$q(t) = -K_p x - K_d\dot{x}. \tag{16}$$

This reformulates the problem into a standard Ordinary Differential Equation (ODE) that can be solved.

This method is analogous to a closed-loop control system. CasADi's NLP solver is employed to optimize the parameters K_p and K_d. From this setup, the simulation derivative, which

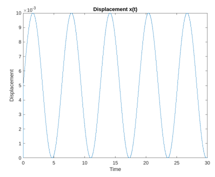


Fig. 3. $q(t) = sin(t) + 1$

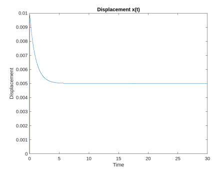


Fig. 4. $q(t) = e^{-t} + 1$

represents the derivative of x with respect to q, is computed as:

$$\frac{\partial x}{\partial q} = \frac{\frac{\partial x}{\partial t}}{\frac{\partial q}{\partial t}}. \tag{17}$$

Both $x(t)$ and $q(t)$ take the form:

$$x(t) = A\exp(-Dt)\sin(Bt + C) + E. \tag{18}$$

The NLP solver provides discrete values of $x(t)$ and $q(t)$. By fitting these values to the above model, the equations of $x(t)$ and $q(t)$ can be obtained, simplifying the computation of the simulation derivative.

The final set of coefficients and targets used in this analysis are:

$$m = 1000, \quad \eta = 0.5, \quad k = 1, \quad x_0 = 10. \tag{19}$$

For this configuration, the optimized values of K_p and K_d are found to be $K_p = 999.9997$ and $K_d = 999.995$, respectively. The control input parameters for $q(t)$ are $(A, B, C, D, E) = (11561.4, 0.870, -63.78, 0.497, -4990)$, while for $x(t)$, they are $(A, B, C, D, E) = (11.55, 0.87, 1.047, 0.497, 9.99)$.

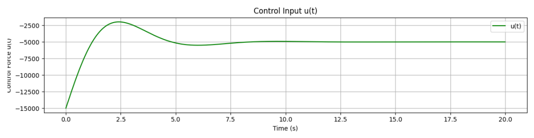


Fig. 5. $q(t)$ for the given set of parameters (m, η, k).

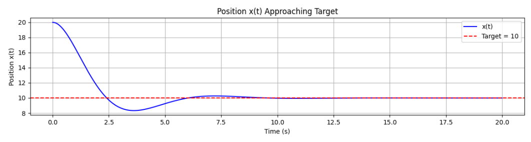


Fig. 6. $x(t)$ for the given set of parameters (m, η, k).

D. Simulation Derivative from PD Control Solution

The simulation derivative, $\frac{\partial x}{\partial q}$, is computed by inducing small perturbations in the dataset of q and observing the corresponding changes in $x(t)$. The ratio of the changes in $x(t)$ to the perturbations in q provides the dataset for the simulation derivative.

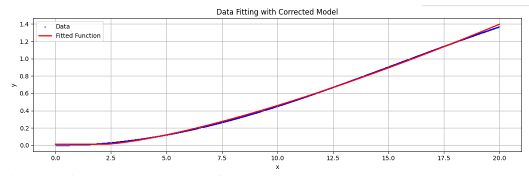
The resulting solution for the simulation derivative is represented in the form:

$$\frac{\mathrm{d}x}{\mathrm{d}q} = a(t-d)^b + c, \tag{20}$$

where the optimized parameters are determined to be $a = 0.027$, $b = 1.362$, $c = 0.012$, and $d = 2.27$. These parameters exhibit minor deviations from the dataset points.

E. Optimal Control Using Discrete-Time Dynamics

The optimal control of the spring-mass-damper system is formulated using the following algorithm:



```
Fitted Parameters: [0.02754032 1.36245054 0.01236006 2.27537185]
```

Fig. 7. Plot for simulation derivative $\frac{\mathrm{d}x}{\mathrm{d}q}$.

Algorithm 1 Optimal Control for Spring-Mass-Damper System

```
0: System Dynamics: $\dot{x}_1 = x_2, \quad \dot{x}_2 = \frac{u - nx_2 - kx_1}{m}$
0: Discretization:
     $x_{k+1} = x_k + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4)$
   where:
     $k_1 = f(x_k, u_k), \quad k_2 = f(x_k + \frac{\Delta t}{2}k_1, u_k), \quad k_3 = f(x_k + \frac{\Delta t}{2}k_2, u_k),$
   with $f(x, u) = \begin{bmatrix} x_2 \\ \frac{u - nx_2 - kx_1}{m} \end{bmatrix}$.
0: Cost Function:
     $J = \sum_{k=0}^{N-1} 10u_k^2 + (x_1[N] - x_{\text{target}})^2 + x_2[N]^2$
0: Constraints:
     $x_1[0] = x_{\text{start}}, \quad x_2[0] = 0, \quad x_{k+1} = x_k + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4), \quad \forall$
0: Optimization: Use IPOPT solver in CasADi to minimize $J$.
0: Output: Optimized state trajectory and control input. =0
```

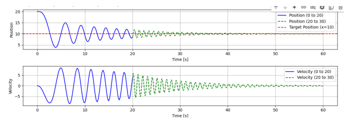


Fig. 8. Position and velocity evolution.

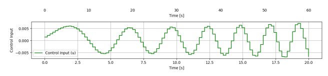


Fig. 9. Optimized control input trajectory.

IV. Graph Networks as Spring-Mass Systems

A. Networking Between Different Nodes

The first step of the project involves connecting N nodes by imposing two primary constraints: length constraints and degree constraints. These constraints are designed to limit the minimum and maximum number of connections per node within a network.

1) Calculations for Length Constraint: The following steps ensure adherence to distance and net constraints in the graph:

a) Pairwise Distance Calculation:: Pairwise Euclidean distances are computed by transforming the $n \times 2$ coordinate matrix into a 3D structure ($n, 1, 2$ and $1, n, 2$). Subtracting these forms yields a $n \times n$ distance matrix after summing along the last axis.

b) Initial Node Assignment:: Each node is connected to m nodes selected from the m^2 closest based on Euclidean distance. An optional triangular connection step may enhance network robustness.

c) Enforcing Net Constraints::

- **Underpopulated Nets:** Additional connections are added to meet the minimum element requirement.
- **Overpopulated Nets:** Farthest neighbors are iteratively removed until the maximum threshold is met.

d) Reinforcement Step:: To ensure consistency:

- All isolated nodes are connected to the nearest available node.
- Self-loops are removed from the graph.

Algorithm 2 Graph Constraint Enforcement

```
0: Input: Coordinate matrix points, graph $G$, min/max components min_components, max_components
0: Compute pairwise distances for points
0: Assign $m$ edges to each node based on proximity
0: Add triangular connections (optional)
0: for each net in $G$ do
0:    if size $<$ min_components then
0:       Add edges to random nodes until size is met
0:    else if size $>$ max_components then
0:       Remove farthest neighbors until size is met
0:    end if
0: end for
0: Connect isolated nodes to the nearest available node
0: Remove self-loops
0: Output: Updated graph $G$ =0
```

B. Using Static Gradients

Each edge in the network is modeled as an over-damped spring connecting two point masses. The system evolves dynamically as follows:

a) Difference Equation for Motion:: The motion is defined by a difference equation for each edge based on the positional differences between connected nodes.

b) Time-Stepping and Force Computation:: Using a time step of $\mathrm{d}t \in [0, 0.1]$, forces are computed as functions of the positional differences, $\mathbf{x}_{ij} = \mathbf{x}_j - \mathbf{x}_i$, for every edge.

c) Potential Energy Function:: The potential energy (or loss) function is:

$$\mathcal{V}(\mathbf{x}_i) = \sum_{i,j} \frac{1}{2} \mathbf{x}_{ij}^{\mathsf{T}} \mathbb{K} \mathbf{x}_{ij}, \tag{21}$$

where $\mathbb{K}$ is the stiffness matrix.

d) Node Motion and Damping:: Node velocities and positions are updated iteratively:

$$\dot{\mathbf{x}}_i = \dot{\mathbf{x}}_i(1 - \mathrm{d}t) - \nabla_{\mathbf{x}_i} \mathcal{V}(\mathbf{x}_i)\mathrm{d}t/\eta,$$
$$\mathbf{x}_i = \mathbf{x}_i + \dot{\mathbf{x}}_i \mathrm{d}t$$

with damping factor η ensuring stability.

e) Simulation and Results:: The system is simulated for 10,000 iterations, with outputs recorded as a video using the FFmpeg writer.

f) Key Observations::

- High damping ($\eta \ll 0.01$) causes erratic node motion.
- Large time steps ($\mathrm{d}t$) result in overshooting the desired range.
- The stiffness matrix $\mathbb{K}$ (default: identity matrix) affects stability if non-diagonal elements dominate.

C. Natural Gradient with the Mahalanobis Metric

The Natural Gradient is constructed using the Mahalanobis metric[4] to transform the 2D Euclidean gradient into a Non-Euclidean gradient space. The steps involved are outlined below.

1. **Constructing the A Matrix** To ensure that the matrix A is positive semi-definite (PSD), it is defined as:

$$A = B^T B, \tag{22}$$

where B is sampled from a normal distribution. 2.**Defining the Loss Function** The potential energy is replaced by a loss function, computed for every edge as:

$$V_k = \sum_{ij} x_{ij}^T A x_{ij}, \tag{23}$$

where $x_{ij} = x_j - x_i$. The total loss is the sum across the entire system.
3.**Mahalanobis Gradient Matrix** The Mahalanobis [4] Gradient Matrix is given by:

$$C(x_i) = \mathrm{Cov}(g_i), \tag{24}$$

where $\mathrm{Cov}(g_i)$ captures the distribution of the gradients of the loss function, and $g_i = \nabla_{x_i} V$. 4.**Regularization of the Covariance Matrix** The regularized covariance matrix is expressed as:

$$C_{\text{mahab}}(x_i) = C(x_i) + \lambda I, \tag{25}$$

where λ prevents singularity and reduces oscillations near convergence.

5. **Computing the Natural Gradient**
The natural gradient is computed as:

$$M^{-1}(x_i) = (C_{\text{mahab}}^{-1}(x_i))^T \nabla_{x_i} V(x_i). \tag{26}$$

Detailed Algorithm is given as in Algorithm 1. Node updates are performed iteratively:

$$\dot{x}_i = \dot{x}_i(1 - \mathrm{d}t) - M^{-1}(x_i)dt/\eta, \tag{27}$$
$$x_i = x_i + \dot{x}_i dt. \tag{28}$$

a) Key Parameters and Their Effects:

- λ: Prevents singularity; optimal values (10^{-2}) dampen oscillations and enhance stability.
- η: Introduces damping; values around 0.99 ensure quicker convergence.
- dt: Time step; $dt = 0.0345$ balances step size and stability.

b) Convergence Results: For $dt = 0.01$ and $\lambda = 10^{-2}$:

- Convergence achieved in 170-180 iterations.
- Higher node counts accelerate convergence with fixed λ and η.
- Animated results are stored as a video for visualization.

Algorithm 3 Natural Gradient Descent with Mahalanobis Metric

```
0: **Initialize:** Positions $x_i$, velocities $v_i$, and matrix $A = B^T B$
0: Set parameters: $\lambda = 10^{-2}$, $\eta = 0.99$, $dt = 0.01$
0: **for** $t = 0$ to max_iterations **do**
0:     Compute pairwise differences: $x_{ij} = x_j - x_i$
0:     Compute loss gradient: $\text{grad}_V = \sum x_{ij}^T A x_{ij}$
0:     Regularize covariance: $C_{\text{mahab}} = C(x_i) + \lambda I$
0:     Invert covariance: $M_{\text{inv}} = (C_{\text{mahab}}{}^{-1})^T \nabla \mathcal{V}$
0:     $v_i = v_i(1 - \text{d}t) - M_{\text{inv}} \cdot \text{d}t/\eta$
0:     $x_i = x_i + v_i \cdot \text{d}t$
0:     Save current state
0: **end for**=0
```

c) Inferences on Parameters: -Regularization parameter λ: - High λ (~ 1): Fails to prevent singularity at higher orders. - Low λ ($\sim 10^{-4}$): Destabilizes convergence by dominating covariance values. - Damping factor η: - $\eta = 0$: No natural damping, leading to slow convergence. - $\eta < 0.1$: Causes nodes to move erratically out of the system bounds.

Final Parameter Selection To stabilize and reduce dynamic behavior near convergence:

- $\lambda = 10^{-2}$,
- $\eta = 0.99$.

D. Trying to Infuse LSE

The system, originally based on the Euclidean framework, was modified to adopt a grid-based approach. This included replacing the loss function with a potential derived from the Log-Sum Exponential (LSE):

$$W = \gamma \left(\log \left(\sum e^{|x_i - x_j|/\gamma} \right) + \log \left(\sum e^{|y_i - y_j|/\gamma} \right) \right), \tag{29}$$

$$\mathcal{V}(x, y) = W^2 \tag{30}$$

a) Observations:

- With $\gamma = 0.1$ and $dt = 0.035$, convergence was achieved in approximately 600 iterations.
- Increasing and dt to 0.07 reduced convergence time to 370-470 iterations.
- For smaller node counts, higher values of γ remained effective. However, with 1000 nodes, $\gamma < 0.01$ was ineffective.
- For a Euclidean setup with a quadratic potential and $dt = 0.07$, the system failed to stabilize and oscillated around the minimum after 170-180 iterations.

b) Inferences:

- LSE provides a smoother potential and gradient, requiring higher step sizes to drive the data towards the minimum effectively.
- Lower values of γ resulted in dynamic motion for nodes with higher lengths but caused prolonged convergence for smaller node lengths.

c) Recommendations for Parameters:

- γ: Use $\gamma \in [0.1, 0.2]$ for faster convergence without instability.
- dt: Keep $dt \in [0.04, 0.07]$ to balance step size and precision.
- Node Count: For higher node counts (≥ 1000), avoid $\gamma < 0.01$ as it may lead to instability.

E. Adding Constraints to the Nodes

In this approach, in every net a point was fixed and then system dynamics was observed after using a Manhattan-based potential.

a) Observations and Adjustments:

- The fixed point in each net served as the convergence point, guiding nodes towards it due to the distance-based nature of the Manhattan potential.
- Initial parameter settings resulted in slow convergence, with the system requiring more than 800 iterations to reach stability.
- Increasing the time step ($dt = 0.16$) improved the convergence speed but led to instability.

b) Inferences:

- In contrast to quadratic-based potentials, the Manhattan-based potential provided a flatter and smoother landscape, which made it slower for nodes that were farther from the goal.
- Nodes had to adapt to the fixed goal, which caused a delay in convergence. Fixed points added to the potential but were unable to further reduce it.
- Because the potential had less of an impact on points further from the fixed point, those points showed slower convergence.

References

[1] C. Sechen and A. Sangiovanni-Vincentelli, "The TimberWolf placement and routing package," IEEE Journal of Solid-State Circuits, vol. 20, no. 2, pp. 510–522, 1985.

[2] P. Spindler, U. Schlichtmann, and F. M. Johannes, "Kraftwerk2—A Fast Force-Directed Quadratic Placement Approach Using an Accurate Net Model," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 27, no. 8, pp. 1398–1411, 2008.

[3] J. Lu, P. Chen, C.-C. Chang, L. Sha, D. J.-H. Huang, C.-C. Teng, and C.-K. Cheng, "ePlace: Electrostatics based placement using Nesterov's method," in Proceedings of the 51st Annual Design Automation Conference, pp. 1–6, 2014.

[4] Andy Jones "Natural Gradients". https://andrewcharlesjones.github.io/journal/natural-gradients.html (accessed Aug. 20, 2024)