Log-Space Harmonic Function Path Planning

Kyle Hollins Wray, Dirk Ruiken, Roderic A. Grupen, and Shlomo Zilberstein

College of Information and Computer Sciences
University of Massachusetts
Amherst, MA 01003, USA
{wray,ruiken,grupen,shlomo}@cs.umass.edu

October 7, 2016
High-Level View

- Robotic path planning
- *Anytime* solution required
- Avoid hitting obstacles
- Plan for *all* states
- *Potential field* solution
- Leverage special log-space
- Parallelize on GPUs
Solutions to Laplace’s Equation: Harmonic Functions

• Laplace’s equation is the partial differential equation (PDE):

\[ \nabla^2 \phi = \sum_{i=1}^{n} \frac{\partial^2 \phi}{\partial \omega_i^2} = 0 \]

for \( n \)-dimensional region \( \Omega \), function \( \phi : \Omega \rightarrow \mathbb{R} \), and boundary \( \partial \Omega \).

• Assign Dirichlet boundary conditions at \( \omega \in \partial \Omega \):
  - \( \phi(\omega) = 1 \) if \( \omega \) is an obstacle
  - \( \phi(\omega) = 0 \) if \( \omega \) is a goal
Algorithm and Solution: Gauss-Seidel and Streamlines

- **Key Idea:** Solutions perform neighbor averaging in a grid.
- Grid $\Omega$ ($m$ discrete states) and iterate until convergence.

  - *Gauss-Seidel* applied to state $\mathbf{x} \in \mathbb{N}^n$ with $x_i \in \{1, \ldots, m\}$ is:

    $$u^{t+1}(\mathbf{x}) = \frac{1}{2n} \sum_{i=1}^{n} u^t(x_i \pm 1, \mathbf{x}_{-i})$$

    with $\mathbf{x}_{-i}$ denoting all other states fixed, and function $u : \mathbb{N}^m$.

- Solutions are *streamlines* computed by gradient descent and interpolation of $u$ in $\Omega$. 

Kyle Hollins Wray, Dirk Ruiken, Roderic A. Grupen, and Shlomo Zilberstein
Advantages of Harmonic Functions

- **Complete** (up to the grid resolution) with a path for any location; there are no local minima.
- Support *multiple* goals and complex obstacles.
- Streamlines are *smooth*; easy to follow via path follower.
- Gauss-Seidel is an *anytime algorithm.*
- Solution *minimizes obstacle hitting probability.*
Disadvantages of Harmonic Functions

Three major problems prevent harmonic functions from use:

[1] Solution is only as good as the grid resolution.

[2] Computationally expensive, compared to RRT or PRM.

[3] Numerical precision (IEEE floating point) issues prevent anything but the smallest of problems from ever being solved.

Multigrid methods and increasing resolution can solve [1].

We provide a solution to [2] via parallelization on GPUs, and a solution to [3], the long-standing numerical precision issue, via a special log-space mapping.
Why do iterative methods fail for large (real) problems?

- **Key Idea**: Goals are rare in the grid, most values are near-one.

- Arguably, real-world problems have:
  - few goals as grid states (e.g., 1 or a handful)
  - many obstacles as grid states (e.g., complex maps or C-space)

- Iterative methods (Gauss-Seidel) do neighbor averaging, e.g.:
  \[
  \frac{1}{4}(0.9999999982+0.9999999975+0.9999999981+0.9999999993)
  \]

- Addition is unstable for these near-one numbers (IEEE flt pt).
Gauss-Seidel returns a valid path (goal reachable) at blue locations.

The *log-space* Gauss-Seidel version produces a valid path everywhere. It can solve the maze from *any* starting location, while optimally avoiding the walls.
A Useful Trick: The Log-Sum-Exp Technique

- **Key Idea**: Add thousands near-zero values together without issues in practical implementations.

- For any values $\mathbf{y} = [y_1, \ldots, y_k]^T$ as:

  $$
  \ell_{se}(\mathbf{y}) = y^* + \log \sum_{i=1}^{k} e^{y_i - y^*} \quad (1)
  $$

  with $y^* = \max_i y_i$ in most cases.
Log-Space Gauss-Seidel Mapping: Key Idea

For neighbor averaging in $n = 2$ dimensions:

$$u'(x, y) = \frac{1}{4} \left( u(x+1, y) + u(x-1, y) + u(x, y+1) + u(x, y-1) \right).$$

Take the log:

$$\log u'(x, y) = \log \left[ \frac{1}{4} \left( u(x+1, y) + u(x-1, y) + u(x, y+1) + u(x, y-1) \right) \right].$$

Apply $u = e^{\log u}$ and let $v = \log u$:

$$v'(x, y) = \log \left[ \frac{1}{4} \left( e^{v(x+1,y)} + e^{v(x-1,y)} + e^{v(x,y+1)} + e^{v(x,y-1)} \right) \right].$$

Rewrite with $2n = 4$, and $\ell se$:

$$v'(x, y) = \ell se(x, y) - \log 2n$$
Log-Space Gauss-Seidel Mapping: Actual Equation

First, map the $u(x) \in [0, 1]$ values to $v \in \mathbb{R}$

$$v(x) = \log \left( (1 - u(x))(1 - \delta) + \delta \right)$$ (2)

for some $\delta \in (0, 1)$ which approximates $\log 0$.

The Gauss-Seidel update is transformed into:

$$v^{t+1}(x) = \begin{cases} 
0, & \text{if } x \text{ is a goal} \\
\log \delta, & \text{if } x \text{ is an obstacle} \\
\ell_{se}(v^t) - \log 2n, & \text{otherwise}
\end{cases}$$ (3)

with $v^t = \langle v^t(x_1 \pm 1, x_{-1}), \ldots, v^t(x_n \pm 1, x_{-n}) \rangle$ and obstacle penalty $\log \delta \ll 0$ (e.g., $\log \delta = -1e^{15}$, which implies $\delta = e^{-1e^{15}}$).
Proposition 1: Mapping is Correct

**Proposition**

Regular Gauss-Seidel mapped to the log-space is equivalent to the log-space Gauss-Seidel.

This simply ensures the mapping is correct.
Proposition 2: Resultant Streamlines Are Correct

Proposition

The streamline solution produced by log-space Gauss-Seidel is equivalent to regular Gauss-Seidel

Importantly, this guarantees:

[1] equivalent solutions using *log-space* Gauss-Seidel

[2] streamlines do *not* require converting back to non-log-space

Note: Converting back to non-log-space every time would cause floating point issues again.
Proposition 3: Error Bound and Convergence

Proposition

For convergence criterion \( \epsilon > 0 \), the error between updates is bounded by:

\[
\| u^t - u^{t-1} \|_\infty < \frac{e^\epsilon - 1}{e^\epsilon (1 - \delta)}
\]  
(4)

The convergence rate and properties are essentially the same; it also converges to 0 as \( \epsilon \to 0 \).
GPU Parallelization

- Gauss-Seidel is known to be parallelizable by ordering updates
- Construct an $n$-dimensional “checkerboard-like” pattern (e.g., red and black cells)
- At each iteration, execute the equation for all red in parallel, then all black again in parallel.
- This work uses an Nvidia GPU (CUDA) for parallelization.
Experimentation on Real-World Map Dataset

<table>
<thead>
<tr>
<th>Domain</th>
<th>( N )</th>
<th>( % )</th>
<th>( T_u )</th>
<th>( T_c )</th>
<th>( T_u )</th>
<th>( T_c )</th>
<th>( T_u )</th>
<th>( T_c )</th>
</tr>
</thead>
<tbody>
<tr>
<td>UMass</td>
<td>152600</td>
<td>90.50</td>
<td>0.74</td>
<td>0.51</td>
<td>6.27</td>
<td>160.51</td>
<td>0.19</td>
<td>4.88</td>
</tr>
<tr>
<td>Willow</td>
<td>1694561</td>
<td>00.67</td>
<td>7.79</td>
<td>2.64</td>
<td>63.81</td>
<td>7095.86</td>
<td>0.39</td>
<td>42.55</td>
</tr>
<tr>
<td>Mine S</td>
<td>1221120</td>
<td>20.25</td>
<td>2.63</td>
<td>1.31</td>
<td>16.48</td>
<td>786.32</td>
<td>0.26</td>
<td>12.37</td>
</tr>
<tr>
<td>Mine L</td>
<td>2461965</td>
<td>10.24</td>
<td>3.71</td>
<td>0.99</td>
<td>18.57</td>
<td>887.45</td>
<td>0.36</td>
<td>16.96</td>
</tr>
<tr>
<td>C-Space</td>
<td>95352</td>
<td>35.67</td>
<td>0.37</td>
<td>0.09</td>
<td>2.94</td>
<td>67.90</td>
<td>0.08</td>
<td>1.86</td>
</tr>
<tr>
<td>Maze S</td>
<td>194084</td>
<td>01.71</td>
<td>1.03</td>
<td>0.09</td>
<td>9.21</td>
<td>185.12</td>
<td>0.08</td>
<td>1.62</td>
</tr>
<tr>
<td>Maze L</td>
<td>925444</td>
<td>00.14</td>
<td>5.00</td>
<td>0.37</td>
<td>44.19</td>
<td>8321.81</td>
<td>0.28</td>
<td>51.83</td>
</tr>
</tbody>
</table>

Results for 7 domains with differing numbers of cells \( N \).

Algorithms: CPU SOR (\( C \)), CPU log-GS (\( \ell-C \)), GPU log-GS (\( \ell-G \)).

Metrics: percentage of cells with a valid streamline (\( \% \)), time per update (\( T_u \) in milliseconds), time for convergence (\( T_c \) in seconds).

Kyle Hollins Wray, Dirk Ruiken, Roderic A. Grupen, and Shlomo Zilberstein
Experimentation on a Robot Platform: uBot-6
Summary of Contributions

- Log-space algorithm for robot path planning
- Three propositions proving correctness and convergence
- Experiments demonstrating it works on real world maps
- Demonstration on a real robot: uBot-6
- GPU implementation which vastly improves scalability
- ROS library “epic” for general-purpose use
Thank You!

Any questions?