Introduction of fast APSP algorithm and optimization algorithms for grid graphs

Masahiro Nakao\textsuperscript{†}, Maaki Sakai\textsuperscript{†}, Yoshiko Hanada\textsuperscript{‡}

(\textsuperscript{†}RIKEN R-CCS, \textsuperscript{‡}Kansai University)
Obtain ASPL and diameter

- Metaheuristic such as SA and GA are often used
- To evaluate a graph, its diameter and ASPL are needed which are calculated by APSP algorithm **many times**
- It is very important to calculate APSP at high-speed

*e.g. For a problem \((n, d) = (1M, 32)\), the time required for one APSP is about 37 hours by the method based on Breadth-First Search (BFS) on Intel Gold 6126*
Our APSP algorithms

- Our previous research provides a parallel APSP algorithm based on BFS (BFS-APSP) [1-3]
- This presentation introduces a new parallel APSP algorithm based on adjacency matrix (ADJ-APSP) [4-5]. The original ADJ-APSP was developed by Ryuhei Mori [6]

You can download our program from

https://github.com/mnakao/APSP/

ADJ-APSP(1/3)

- Let A be an adjacency matrix of a graph
- If the value of an element \(a_{i,j}\) in \(A^k\) is 1, it means that the vertex i can reach the vertex j within k hops

\[(n, d) = (10, 3)\]

```java
for(int i=0;i<n;i++)
```
### ADJ-APSP (2/3)

<table>
<thead>
<tr>
<th>A</th>
<th>adjlist</th>
<th>$A^1$</th>
<th>$A^2$</th>
<th>$A^3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0 0 0 0 0 1</td>
<td>2 3 5</td>
<td>0 0 0 0 1 0 1 1 0 1</td>
<td>1 0 1 0 1 1 1 1 1</td>
<td>1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0 0 0 1 0 0</td>
<td>5 6 8</td>
<td>0 1 0 1 1 0 0 0 1 0</td>
<td>1 1 1 1 1 0 0 0 1 1</td>
<td>1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0</td>
<td>0 3 4</td>
<td>0 0 0 0 0 1 1 1 0 1</td>
<td>1 0 1 0 1 1 1 1 0 1</td>
<td>1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0</td>
<td>0 2 9</td>
<td>1 0 0 0 0 0 1 1 0 1</td>
<td>1 0 0 1 1 1 1 1 0 1</td>
<td>1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0</td>
<td>2 7 9</td>
<td>1 0 1 0 0 1 0 1 0 0</td>
<td>1 1 1 1 1 1 1 1 0 1</td>
<td>1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0</td>
<td>0 1 7</td>
<td>0 0 1 0 1 0 0 0 1 1</td>
<td>0 1 1 1 1 1 1 1 1 1</td>
<td>1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td>1 8 9</td>
<td>1 1 0 1 0 0 0 0 1 0</td>
<td>1 1 1 1 1 1 0 1 0 0</td>
<td>1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td>4 5 8</td>
<td>0 1 1 0 1 1 0 0 0 0</td>
<td>1 1 1 1 1 0 1 1 1</td>
<td>1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td>1 6 7</td>
<td>0 1 1 1 0 0 0 0 1 0</td>
<td>1 1 1 1 1 0 0 1 0 0</td>
<td>1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td>3 4 6</td>
<td>1 0 0 1 0 1 1 0 0 0</td>
<td>1 1 1 0 1 1 1 1 1</td>
<td>1 1 1 1 1 1 1 1 1</td>
</tr>
</tbody>
</table>

- As $k$ is increased in increments of 1, the value of $k$ is the **diameter** when all elements are 1.
- Every time $k$ is increased from 1 to the diameter, the **total distance** is obtained by summing all the elements whose value is 0.
  - **ASPL** is calculated by dividing the **total distance** by the number of elements.
We have developed Serial, Multi-threads GPU, Multi-GPUs versions

```c
function SERIAL_ADJ_APSP(vertices, nodes)
  diameter ← 1
  distance ← nodes*(nodes-1)
  elements ← [nodes/E]
  A, B ← INITIALIZE(nodes, elements)
  for k=1 ... nodes-1
    for i=1 ... nodes
      for n ∈ neighbors(i, vertices)
        for j=1 ... elements
    num ← 0
    for i=1 ... nodes
      for j=1 ... elements
        num ← num+POPCNT(B[i][j])
    if(num = nodes*nodes) break
    SWAP(A, B)
    diameter++
    distance ← distance+(nodes*nodes−num)
  average_distance ← distance/((nodes−1)*nodes)
  return diameter, average_distance
```
Experiment environment

### Cygnus system in Univ. of Tsukuba

<table>
<thead>
<tr>
<th>Component</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Intel Xeon Gold 6126 (12 Cores, 2.6GHz) × 2</td>
</tr>
<tr>
<td>Memory</td>
<td>DDR4 (128GB/s × 2, 192GB)</td>
</tr>
<tr>
<td>GPU</td>
<td>NVIDIA Tesla V100 (900GB/s, 32GB) × 4</td>
</tr>
<tr>
<td>Network</td>
<td>InfiniBand HDR100 (12.5GB/s) × 4</td>
</tr>
<tr>
<td>Software</td>
<td>intel/19.0.3, mvapich/2.3.1, cuda/10.1</td>
</tr>
</tbody>
</table>

- The Cygnus system is provided by Interdisciplinary Computational Science Program in the Center for Computational Sciences, University of Tsukuba
  - Computing resources such as Cygnus and Oakforest-PACS can be used **for free**
- This entry is held around mid-December
Result (Change from BFS to ADJ)

Speed to calculate APSP for graph with (1M, 32)
Intel Xeon Gold 6126 2.6GHz

134,300 sec (37 hours)  
BFS  x 35.5  
3,780 sec.  
ADJ
Result (Multi-threads)

Speed to calculate APSP for graph with (1M, 32)
Intel Xeon Gold 6126 2.6GHz (12 cores)

TIME (sec.)

<table>
<thead>
<tr>
<th></th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADJ</td>
<td>3,780</td>
</tr>
<tr>
<td>Multi-threads</td>
<td>474</td>
</tr>
</tbody>
</table>

3,780 sec. x 8.0 = 474 sec.
Result (GPU)

Speed to calculate APSP for graph with (1M, 32)
Intel Xeon Gold 6126 2.6GHz (12 cores) -> NVIDIA V100

- Multi-threads: 474 sec.
- GPU: 28.7 sec.

GPU speed is approximately 16.5 times faster than Multi-threads.
Result (Multi-GPUs)

Speed to calculate APSP for graph with (1M, 32)
Intel Xeon Gold 6126 2.6GHz (12 cores) -> NVIDIA V100 x 128

- **GPU**: 28.7 sec.
- **Multi-GPUs**: 0.28 sec.

The speed of calculation is increased by a factor of 101.1 when using Multi-GPUs compared to a single GPU.
Summary

Of course, when using graph symmetry, the calculation time can be reduced more greatly.

Speed to calculate APSP for graph with (1M, 32)
Intel Xeon Gold 6126 2.6GHz (12 cores) -> NVIDIA V100 x 128

134,300秒 -> 0.28秒 (x 500,000)
Graph symmetry

- The algorithm is inherited from mine for general graph last year [1-3]
- Examples of the graph symmetry with \((W, H, D, R) = (6, 6, 4, 2)\)

\[
\begin{align*}
\text{g=1 (normal graph)} & \quad \text{g=2} & \quad \text{g=4}
\end{align*}
\]

- The variable \(g\) is the number of groups \((g\) must be 1 or 2 or 4)\)
- If a graph is rotated by \(360/g\) degrees, the connection relationship between the vertex and edge becomes the original one
Edge exchange based on 2-opt

Perform 2-opt method while maintaining symmetry

In case of $g=4$

(1) Randomly select two edges from all the edges
(2) Select edges symmetrically related to (1)
(3) Apply the 2-opt method to above edges each other
The vertical axis is the ASPL Gap Ratio when the result of $g=1$ is 1.0

The larger the value of $g$ is, the smaller the ASPL Gap is

- Larger problems tend to have larger performance differences

The results show that graph symmetry is also useful in grid graphs
Genetic Algorithm

- GA is a direct search method that can be applied for various complex problems.
- GA shows good performance in large-scale TSP.
- Crossover design is the most important to improve search performance of GA.
  - Sophisticated crossover that deals with problem-specific features are required.
Crossover Method

1. Overlap two graphs.

2. Continue removing the edge until the graph satisfy constraint.
   - Select nodes with the highest degree and do not satisfy the degree constraint.
   - Find the edge with the smallest effect on ASPL.

\[ \text{over} (W, H, D, R) = (3, 2, 3, 3) \]
Crossover Method

3. Continue adding the edge while edge can be added.
   • Select nodes with the lowest degree that can accept edges.
   • Find the best edge that improves ASPL.

\[(W, H, D, R) = (3, 2, 3, 3)\]
## Numerical Experiments

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance</td>
<td>(W, H, D, R) = (10, 5, 4, 2), (10, 10, 4, 2)</td>
</tr>
<tr>
<td>Group size</td>
<td>1, 2, 4</td>
</tr>
<tr>
<td>Generation alternation model</td>
<td>MGG[^1]</td>
</tr>
<tr>
<td>Max evaluation count</td>
<td>100M</td>
</tr>
<tr>
<td>Population size</td>
<td>100</td>
</tr>
<tr>
<td>Offspring size</td>
<td>200</td>
</tr>
<tr>
<td>Mutation rate (Shuffle)</td>
<td>0.01</td>
</tr>
</tbody>
</table>

\[^1\]: 佐藤 浩, 小野 功, 小林 重信, 遺伝的アルゴリズムにおける世代交代モデルの提案と評価, 人工知能学会誌, Vol. 12, No. 5, pp734 – 744, 1997

![Graphs](g=1.png, g=2.png, g=4.png)
Result using GA

<table>
<thead>
<tr>
<th>Rectangular Graph</th>
<th>Square Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>10, 5, 4, 2</td>
<td>10, 10, 4, 2</td>
</tr>
<tr>
<td>$g = 1$</td>
<td>$g = 1$</td>
</tr>
<tr>
<td>3.10694</td>
<td>4.01434</td>
</tr>
<tr>
<td>3.110529</td>
<td>4.020768</td>
</tr>
<tr>
<td>3.11265</td>
<td>4.02687</td>
</tr>
<tr>
<td>$g = 2$</td>
<td>$g = 2$</td>
</tr>
<tr>
<td>3.1102</td>
<td>3.99293</td>
</tr>
<tr>
<td>3.110445</td>
<td>3.996365</td>
</tr>
<tr>
<td>3.11265</td>
<td>4.00162</td>
</tr>
<tr>
<td>$g = 4$</td>
<td>$g = 4$</td>
</tr>
<tr>
<td>3.98586</td>
<td>3.98586</td>
</tr>
<tr>
<td>3.993577</td>
<td>3.993577</td>
</tr>
<tr>
<td>3.99596</td>
<td>3.99596</td>
</tr>
</tbody>
</table>

- For rectangular instance, the symmetric grouping technique makes the performance worse.
- For square instance, grouping technique enhances the performance, and the larger number of grouping is better.
Remarkable traits that have not been found a conventional approach based on 2-opt were observed.

Best solutions obtained by GA include non-regular nodes of which degree is less than 4, while best solutions obtained by SA based on 2-opt consist of completely regular nodes.

Result using GA

- GA: \((10, 5, 4, 2)\)
- SA: \((10, 10, 4, 2)\)
Summary

- We applied GA to solve Graph Golf instances.
- Symmetric grouping technique works well on square instance.
- Remarkable traits that cannot be found in conventional approach were obtained by GA.
- GA requires much computation cost, so that we should improve the efficiency of crossover.