CSC4120 Project Report
Party Together Problem (PTP)

Ziqi Chen
123090059@link.cuhk.edu.cn

Xinyu Wu
123090635@link.cuhk.edu.cn

Ming Wen
123090613@link.cuhk.edu.cn

December 18, 2025

Group ID: 25

1 Introduction

The *Party Together Problem* (PTP) models a driver starts and ends at home and needs to pick up a group of friends distributed over a road network, where allows each friend to either be picked up at their own home or walk a short distance to a neighboring node. This flexibility introduces a trade-off between the driving cost of the car and the walking cost incurred by the friends.This problem also generalizes the *Pick-up at Home Problem* (PHP), which requires the car to visit all friends' homes and is closely related to the Traveling Salesman Problem (TSP). As a result, PTP inherits significant computational challenges.

In this report, we first study the theoretical properties of PTP, including its NP-hardness and approximation bounds. We then propose and implement several solution approaches, ranging from greedy heuristics and local search to simulated annealing and exact integer linear programming, and discuss their practical trade-offs.

2 environment Setup

2.1 Prerequisites

- Python $\geq$ 3.13

2.2 Installation

2.2.1 Step 1: Create a Virtual Environment (Recommended)

```
python -m venv venv
source venv/bin/activate    # On macOS/Linux
# or
venv\Scripts\activate       # On Windows
```

2.2.2 Step 2: Install Required Dependencies

```
pip install networkx>=3.6 numpy>=2.3.5 matplotlib>=3.10.7 pulp
   >=2.7.0
```

2.2.3 Step 3: (Optional) For Jupyter Notebook Support

```
pip install ipykernel>=7.1.0
```

2.3 Quick Install

Install all dependencies in one command:

```
pip install networkx numpy matplotlib pulp ipykernel
```

2.4 Dependencies Summary

Package	Version	Purpose
networkx	≥ 3.6	Graph operations and algorithms
numpy	$\geq 2.3.5$	Numerical computations
matplotlib	$\geq 3.10.7$	Visualization and plotting
pulp	$\geq 2.7.0$	Integer Linear Programming solver
ipykernel	$\geq 7.1.0$	Jupyter notebook support (optional)

Table 1: Required Python packages and their purposes

3 Theoretical Questions

Setup

Let $G = (V, E)$ be a weighted undirected graph with shortest-path metric $d(\cdot, \cdot)$, start/end vertex 0, and homes $H \subseteq V$. In **PTP**, each friend at $h \in H$ chooses a pickup $p(h) \in \{h\} \cup N(h)$ and the car tour $\mathcal{T}$ must visit all $p(h)$. Objective:

$$C_{\mathrm{PTP}}(\mathcal{T}, p) = \alpha \, \mathrm{len}(\mathcal{T}) + \sum_{h \in H} d\big(h, p(h)\big), \qquad C^*_{\mathrm{PTP}} = \min C_{\mathrm{PTP}}(\mathcal{T}, p).$$

In **PHP**, all pickups are at home ($p(h) = h$), so the objective is the minimum tour length visiting all H: $C_{\mathrm{PHP}} = \min \mathrm{len}(\mathcal{T})$.

5.1 PTP is NP-hard

Lemma 1 *If $\alpha \leq \frac{1}{2}$, then for any feasible PTP solution $(\mathcal{T}, p)$ there exists a feasible solution $(\mathcal{T}', p')$ with $p'(h) = h$ for all $h \in H$ and $C_{\mathrm{PTP}}(\mathcal{T}', p') \leq C_{\mathrm{PTP}}(\mathcal{T}, p)$.*

Fix h with $p(h) \neq h$. Since $p(h) \in N(h)$, detour $p(h) \to h \to p(h)$ (whenever at $p(h)$) makes pickup at h. Driving increases by $2d(h, p(h))$, walking decreases by $d(h, p(h))$, so

$$\Delta C = 2\alpha \, d(h, p(h)) - d(h, p(h)) = (2\alpha - 1) d(h, p(h)) \leq 0 \quad (\alpha \leq \tfrac{1}{2}).$$

Apply to all such h.

Theorem 1 *For $\alpha \leq \frac{1}{2}$, an optimal PTP solution can be chosen with $p(h) = h$ for all h; hence $C^*_{\mathrm{PTP}} = \alpha \, C_{\mathrm{PHP}}$. Therefore PHP $\leq_p$ PTP and PTP is NP-hard.*

5.2 For $\alpha = 1$, $\beta \leq 2$ and is tight

Let $\alpha = 1$. For an optimal PTP solution, write

$$C^*_{\text{PTP}} = D^* + W^*, \quad D^* = \text{len}(\mathcal{T}^*), \quad W^* = \sum_{h \in H} d\big(h, p^*(h)\big).$$

Define $\beta = \dfrac{C_{\text{PHP}}}{C^*_{\text{PTP}}}$.

Theorem 2 (Upper bound) *For $\alpha = 1$, $C_{\text{PHP}} \leq 2C^*_{\text{PTP}}$, hence $\beta \leq 2$.*

From $(\mathcal{T}^*, p^*)$, build a PHP-feasible tour by traversing $\mathcal{T}^*$ and, for each friend h picked at $p^*(h)$, adding a detour $p^*(h) \to h \to p^*(h)$. Added driving is $2d(h, p^*(h))$ per friend, totaling $2W^*$, so a PHP tour of length $\leq D^* + 2W^*$ exists. Thus

$$C_{\text{PHP}} \leq D^* + 2W^* \leq 2(D^* + W^*) = 2C^*_{\text{PTP}}.$$

Theorem 3 (Tightness) *There are instances with $\beta \to 2$.*

Take a star-like tree with vertices $\{0, c, h_1, \ldots, h_k\}$ and unit edges $(0, c)$ and (c, h_i). PTP optimum: pick all at c and drive $0 \to c \to 0$, so $C^*_{\text{PTP}} = 2 + k$. PHP must visit all h_i, so $C_{\text{PHP}} = 2 + 2k$. Hence

$$\beta = \frac{2 + 2k}{2 + k} = 2 \cdot \frac{k + 1}{k + 2} \xrightarrow[k \to \infty]{} 2.$$

4 PTP Solver Approaches

This Project implement multiple method to solve Party Together Problem (PTP), where then main solver `ptp_solver` will automatically select method based on problem size and return the optimal solution.

4.1 Approach 1: Greedy algorithm

4.1.1 Describption

We can take the insert/delete heuristic [1] to build the solution iteratively, each time either to improve the feasibility or reduce the cost. We start with a initial tour **T** "0 to 0" and do a local search by either delete a node T or adding a new node with **lowest cost**. The algorithm stop when there is no further improvement can be made.

4.1.2 Implementation Details

The algorithm employs the Floyd-Warshall algorithm to precompute all-pair shortest path distances, which are stored in a distance matrix D where $D[i][j]$ represents the shortest path distance between nodes i and j.

During the insertion phase, new nodes are inserted only between consecutive key nodes in the current sequence. Specifically, when inserting a node v into a key node sequence $\tau = [v_0, v_1, \ldots, v_k]$, we evaluate all possible insertion positions between consecutive pairs (v_i, v_{i+1}) for $i = 0, \ldots, k-1$, and select the position that minimizes the cost.

Cost computation is performed based on the shortest path distances between consecutive key nodes, leveraging the triangle inequality property of the graph. The algorithm maintains a compact representation using only key nodes during the search process, and the final tour is constructed by expanding the key node sequence into a complete path using precomputed shortest paths between consecutive key nodes.

4.1.3 Complexity analysis

Time Complexity. Let $|V|$ denote the number of nodes, $|H|$ the number of friends, $|\tau|$ the number of key nodes in the current solution (typically $|\tau| = O(|H|)$), and Δ the maximum degree.

- **Preprocessing**: Floyd-Warshall all-pairs shortest paths: $O(|V|^3)$.
- **Single iteration**: For each of $O(|V|)$ candidate nodes, `least_cost_insert` evaluates $O(|\tau|)$ insertion positions, each requiring $O(|H| \cdot |\tau|)$ to compute the cost function. This gives $O(|V| \cdot |H| \cdot |\tau|^2)$ per iteration.
- **Number of iterations**: $O(|V|)$ due to the triangle inequality (each modification strictly improves the solution).

Assuming $|\tau| = O(|H|)$, the total time complexity is:

$$T(n) = O(|V|^3) + O(|V|^2 \cdot |H|^3)$$

Space Complexity. The algorithm maintains:

- Distance matrix: $O(|V|^2)$
- Shortest paths dictionary: $O(|V|^2)$
- Key nodes and candidate solutions: $O(|V|)$

The total space complexity is $O(|V|^2)$.

Practical Performance. In practice, the algorithm performs better than the worst-case bound suggests, as the number of key nodes $|\tau|$ is typically close to $|H|$ (much smaller than $|V|$), and the algorithm converges to a local optimum in far fewer iterations than $O(|V|)$.

4.1.4 Pros and Cons

+ Time efficient to calculate a good solution.

− May stuck in local optimal

4.2 Approach 2: Multi-Start Local Search

4.2.1 Description

Multi-start local search addresses the limitation of greedy algorithms getting stuck in local optima by running the same local search procedure from multiple diverse initial solutions. The algorithm generates a set of initial solutions using various heuristics, applies local search to each, and returns the best solution found across all runs.

4.2.2 Implementation Details

The algorithm generates initial solutions using four distinct strategies:

1. **Empty tour**: Start with the trivial tour $[0, 0]$.
2. **Random sampling**: Randomly select a subset of friends' home nodes.
3. **Nearest neighbor**: Greedily construct a tour by repeatedly visiting the nearest unvisited friend's home from the current position.

4. **Farthest insertion**: Iteratively insert the node farthest from the current tour at its best position.

For each initial solution, the algorithm applies the same insert/delete local search procedure as in the greedy approach. The local search continues until no improvement is possible. After all starting points are explored, the solution with the minimum cost is returned.

4.2.3 Complexity Analysis

Time Complexity. Let K denote the number of starting points (default $K = 10$), and let $|\mathcal{K}|$ denote the average size of key nodes in the tour (typically $|\mathcal{K}| = O(|H|)$).

- **Preprocessing**: Floyd-Warshall all-pairs shortest paths: $O(|V|^3)$.
- **Initial solution generation**: Dominated by farthest insertion, which takes $O(|H|^2 \cdot |\mathcal{K}|) = O(|H|^3)$.
- **Single local search iteration**: For each of $O(|V|)$ candidate nodes, `least_cost_insert` evaluates $O(|\mathcal{K}|)$ insertion positions, each requiring $O(|H| \cdot |\mathcal{K}|)$ to compute the cost function. This gives $O(|V| \cdot |H| \cdot |\mathcal{K}|^2)$ per iteration.
- **Number of iterations**: $O(|V|)$ due to the triangle inequality.
- **Single local search**: $O(|V|) \times O(|V| \cdot |H| \cdot |\mathcal{K}|^2) = O(|V|^2 \cdot |H| \cdot |\mathcal{K}|^2)$.

Assuming $|\mathcal{K}| = O(|H|)$, the total time complexity is:

$$T(n) = O(|V|^3) + O(K \cdot |V|^2 \cdot |H|^3)$$

Space Complexity. Same as the greedy approach: $O(|V|^2)$ for the distance matrix and shortest paths.

4.2.4 Pros and Cons

+ Explores diverse regions of the solution space, reducing the chance of missing good solutions.
+ Easily parallelizable since each starting point is independent.
− Increased computation time proportional to the number of starting points.
− Still deterministic local search; may miss global optimum if no initial solution is close to it.

4.3 Approach 3: Simulated Annealing

4.3.1 Description

Simulated Annealing (SA) is a probabilistic metaheuristic inspired by the annealing process in metallurgy. Unlike greedy local search, SA can accept worse solutions with a probability that decreases over time (controlled by a "temperature" parameter), allowing the algorithm to escape local optima and explore the solution space more broadly.

4.3.2 Implementation Details

The algorithm maintains a current solution and iteratively generates neighbor solutions through random perturbations. Four neighborhood operations are employed with the following probabilities:

- **Insert** (40%): Add a random node not in the current tour at a random position.
- **Delete** (30%): Remove a random node from the tour (excluding node 0).
- **Swap** (20%): Exchange the positions of two random nodes in the tour.
- **2-opt** (10%): Reverse a random segment of the tour.

The acceptance criterion follows the Metropolis rule: if the new solution has lower cost, accept it; otherwise, accept with probability $\exp(-\Delta C/T)$, where ΔC is the cost increase and T is the current temperature.

Parameter	**Value**
Initial temperature	$T_0 = 1000$
Final temperature	$T_f = 0.1$
Cooling rate	$\alpha = 0.95$ (geometric)
Iterations per temperature	100
Maximum iterations	10,000

Table 2: Simulated Annealing parameter settings

4.3.3 Complexity Analysis

Time Complexity. Let I denote the total number of iterations, and $|\tau|$ denote the number of key nodes (typically $|\tau| = O(|H|)$).

- **Preprocessing**: Floyd-Warshall all-pairs shortest paths: $O(|V|^3)$.
- **Single iteration**: For each of $O(|V|)$ candidate nodes, `least_cost_insert` evaluates $O(|\tau|)$ insertion positions, each requiring $O(|H| \cdot |\tau|)$ to compute the cost function. This gives $O(|V| \cdot |H| \cdot |\tau|^2)$ per iteration.
- **Number of iterations**: With geometric cooling from T_0 to T_f, the number of temperature levels is $L = O(\log(T_0/T_f)/\log(1/\alpha))$. Total iterations $I = L \times \text{iterations\_per\_temp}$.

The total time complexity is:

$$T(n) = O(|V|^3) + O(I \cdot |V| \cdot |H|^3)$$

With the default parameters ($T_0 = 1000$, $T_f = 0.1$, $\alpha = 0.95$, 100 iterations per temperature), $I \approx 17800$ iterations.

Space Complexity. $O(|V|^2)$ for the distance matrix and shortest paths.

4.3.4 Pros and Cons

+ Ability to escape local optima through probabilistic acceptance of worse solutions.
+ Often finds high-quality solutions for combinatorial optimization problems.
− Stochastic nature means results vary between runs; multiple runs may needed to find suitable result.

4.4 Approach 4: Integer Linear Programming (ILP)

4.4.1 Description

Integer Linear Programming formulates PTP as a mathematical optimization problem with linear objective and constraints over integer decision variables. We adopt the Miller-Tucker-Zemlin (MTZ) formulation [2] to eliminate subtours. When the solver finds an optimal solution, it is guaranteed to be globally optimal.

4.4.2 Implementation Details

Decision Variables.

- $x_{ij} \in \{0, 1\}$: equals 1 if the car travels directly from node i to node j.
- $y_{hp} \in \{0, 1\}$: equals 1 if friend h is picked up at location $p \in \{h\} \cup \mathcal{N}(h)$.
- $u_i \in \{1, \ldots, |V| - 1\}$: auxiliary variable for MTZ subtour elimination, representing the visit order of node i.

Objective Function. Minimize total cost:

$$\min \quad \alpha \sum_{i \neq j} d_{ij} \cdot x_{ij} + \sum_{h \in H} \sum_{p \in \{h\} \cup \mathcal{N}(h)} d_{hp} \cdot y_{hp}$$

Constraints.

1. **Flow conservation**: The tour starts and ends at node 0; each visited node has equal in-degree and out-degree.
2. **MTZ subtour elimination**: $u_i - u_j + |V| \cdot x_{ij} \leq |V| - 1$ for all $i, j \neq 0$.
3. **Pickup assignment**: Each friend must be picked up exactly once: $\sum_p y_{hp} = 1$ for all $h \in H$.
4. **Pickup location in tour**: If $y_{hp} = 1$, then node p must be visited by the tour.

The problem is solved using the PuLP library with the CBC (Coin-or Branch and Cut) solver.

4.4.3 Complexity Analysis

Time Complexity. ILP is NP-hard in general. The number of binary variables is $O(|V|^2 + |H| \cdot \Delta)$, and the number of constraints is $O(|V|^2 + |H| \cdot \Delta)$. The worst-case time complexity is exponential in the number of variables.

Space Complexity. $O(|V|^2)$ for storing the problem formulation and distance matrix.

Practical Applicability. Due to the exponential worst-case complexity, we only apply ILP when $|V| \leq 20$. For larger instances, heuristic methods are used instead.

4.4.4 Pros and Cons

+ Guarantees global optimality when the solver completes.

− Exponential worst-case time complexity; impractical for large instances.

5 Final Solution

The main solver is a hybird method, when the grpah is at small size($|V|$ less than 20) it will consider ILP, and for each method it will repeat several times and pick the best solution to return.

5.1 Performance Summary

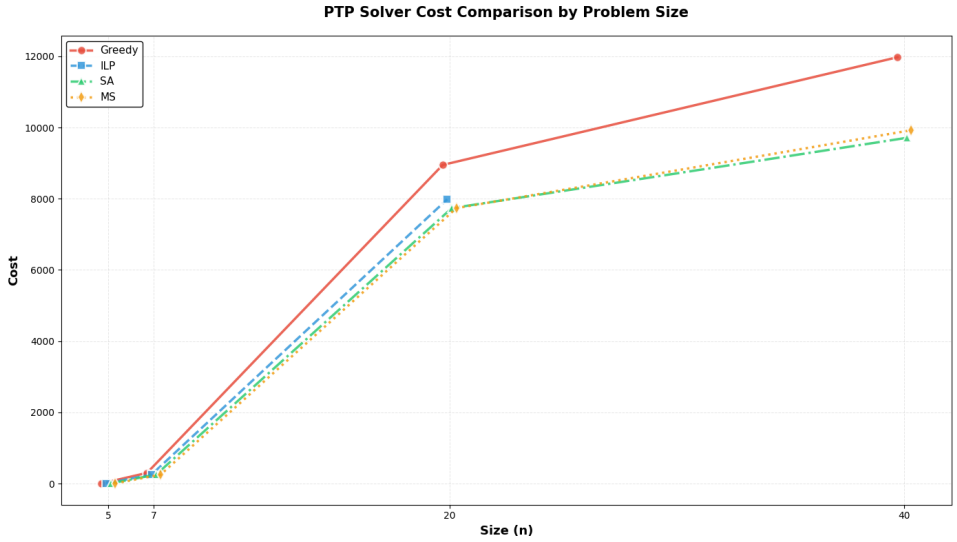

Figure 1: Cost between different approaches

Figure 1 compares solution quality (total cost) across four methods—Greedy, ILP, Simulated Annealing (SA), and MS—under increasing problem sizes. As the number of nodes grows, all methods exhibit increasing cost, but Greedy consistently performs worst. ILP achieves better solutions for small and medium instances but scales poorly. SA and MS deliver lower costs for larger instances, indicating superior scalability and robustness for larger PTP problems.

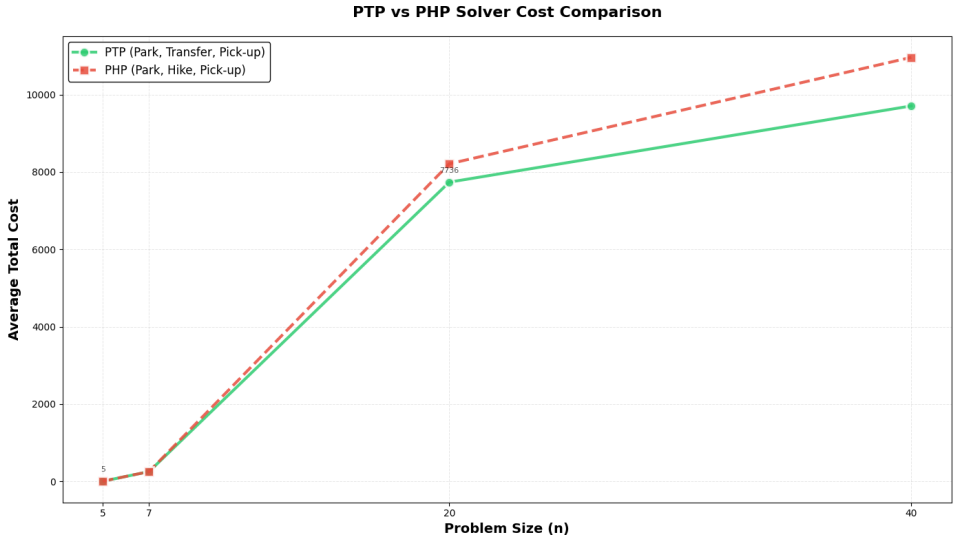

Figure 2: Cost between PTP and PHP

Figure 2 contrasts the average total cost of PTP and PHP formulations. Across all problem

sizes, PTP consistently yields lower total cost than PHP. The performance gap widens as problem size increases, highlighting that allowing transfer-based pick-ups (PTP) is more cost-efficient than enforcing hiking constraints (PHP), especially for larger instances.

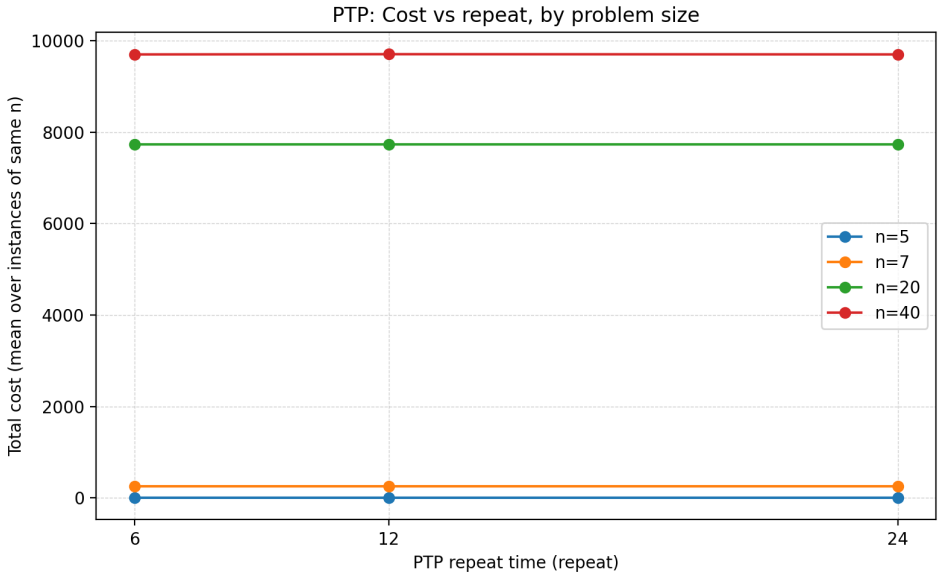

Figure 3: Cost between PTP and PHP

Figure 3 evaluates the sensitivity of the PTP solver to the number of repeated heuristic runs. For all tested problem sizes, increasing the repeat count yields minimal change in average cost. This indicates that the PTP heuristic converges quickly and produces stable solutions, suggesting diminishing returns from additional repetitions beyond a modest number.

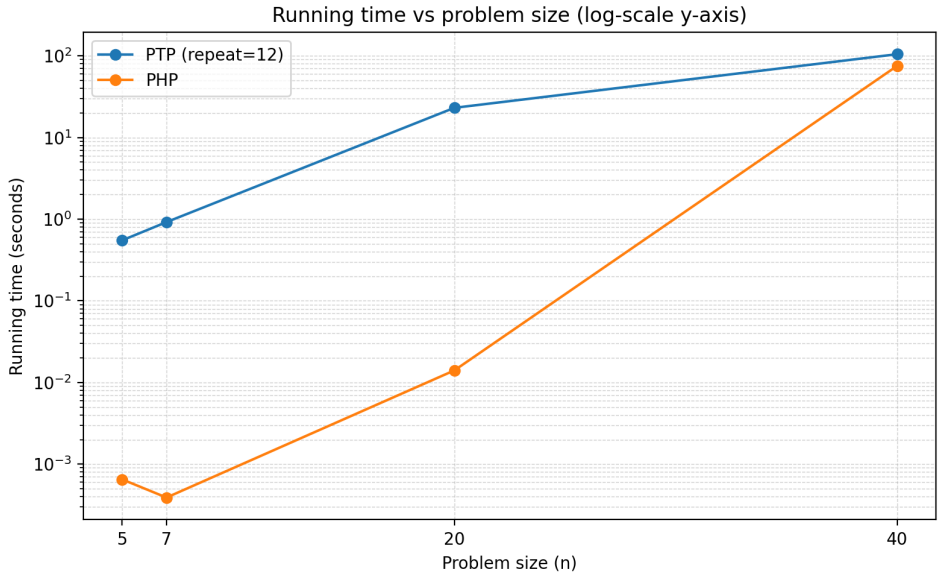

Figure 4: Cost between PTP and PHP

Figure 4 compares the runtime scalability of PTP and PHP solvers. PHP runs extremely fast for small instances but experiences a sharp increase in runtime as problem size grows. In contrast, PTP has higher runtime overall but scales more smoothly and predictably. This demonstrates

a clear trade-off: PTP achieves better solution quality at the cost of higher computation time, while PHP is faster but less scalable in large instances.

6 Conclusion

In this project, we studied the theoretical hardness of Party Together Problem (PTP), and implemented multiple solution strategies. Greedy local search provides fast, reasonably good solutions, while multi-start local search and simulated annealing improve solution quality by exploring a broader search space. For small instances, an integer linear programming formulation guarantees global optimality. Based on these methods, we designed a hybrid solver that automatically selects an appropriate approach according to problem size.

References

[1] J. Mittenthal and C. E. Noon, "An insert/delete heuristic for the travelling salesman subset-tour problem with one additional constraint," *Journal of the Operational Research Society*, vol. 43, no. 3, pp. 277–283, 1992.

[2] C. E. Miller, A. W. Tucker, and R. A. Zemlin, "Integer programming formulation of traveling salesman problems," *Journal of the ACM*, vol. 7, no. 4, pp. 326–329, 1960.