Flying Sidekick Traveling Salesman Person - Report

Antonio Miguel Mosquera Prudencio
Miguel Estévez Díaz

Optimization algorithms

Index:

1. Introduction:

Last-mile distribution has become a key bottleneck in many logistics systems. The growth of e-commerce and the pressure for fast deliveries motivate the use of hybrid delivery systems in which traditional ground vehicles cooperate with unmanned aerial vehicles (drones). The **Flying Sidekick Traveling Salesperson Problem (FSTSP)** models such a system, where a truck and a drone jointly serve a set of customers and must return to a depot.

In the version considered in this project there is **one truck and one drone**. Every customer must be visited exactly once, either by the truck or by the drone, and the objective is to **minimize the total completion time**. Some customers are **truck-only** because of weight limitations. Truck travel times between nodes are derived from a Manhattan distance, while drone travel times are based on Euclidean distances. The drone operates through **sorties**: it takes off from the truck, serves exactly one customer, and lands again on the truck at another node. While the drone is airborne, the truck may keep moving and visiting customers, but both vehicles must satisfy synchronization and battery constraints.

The drone has a maximum **endurance E** that limits the total flight time between launch and recovery. Two additional service times are considered: **launch time SL** and **recovery time SR**. These times contribute to the total completion time but do not consume battery. For a successful landing the truck and the drone must meet at the same node at the same time; the drone is allowed to wait in the air for the truck as long as the total airborne time does not exceed E. If the first sortie starts directly from the depot, its launch time is not charged in the objective.

The FSTSP can be formulated as a mixed-integer linear program (MILP) with decision variables describing the truck route, the drone flights and their synchronization. However, solving the MILP is computationally demanding even for medium-sized instances. The project statement provides **Gurobi optimal solutions** for a family of benchmark instances (with a 1-hour time limit) and asks us to design a heuristic that runs in at most **10 minutes per instance**.

The goal of this work is to develop and implement a **Tabu Search heuristic** for the FSTSP. Solutions are represented as permutations of customers; given a permutation, a greedy procedure decides which customers are assigned to drone sorties and a simulation model evaluates the resulting truck–drone schedule. The performance of the heuristic is finally assessed on the benchmark instances and compared to the Gurobi solutions provided in the project statement.

2. Heuristic algorithm:

2.1 Solution representation:

We represent a solution by two objects:

- A **permutation of all customers:**
  Let $\pi = (\pi 1, \pi 2, \ldots, \pi_{|C|})$ be a permutation of the customer set C. From it we build a basic truck tour:

  $$\text{route\_truck\_base}=[s,\pi 1,\ldots,\pi|C|,t]$$

  Where s and t are the start and end depots. This is the route followed by the truck if it alone visits all customers.

- A **set of drone sorties:**
  A sortie is described by a triple (L_index,h,R_index). It means that:

  - the truck is at node L when the drone is launched,
  - the drone visits customer h,
  - and it lands back on the truck at node R, with R_index>L_index in the truck route.

When a sortie (L_index,h,R_index) is used, customer hhh is removed from the truck route and is served by the drone instead, while the truck directly travels from L to R. The final solution therefore consists of:

- a permutation π,
- the **reduced truck route** obtained by removing all drone-served customers,
- and the list of associated sorties.

This representation is simple, compact, and easy to modify with local search moves on the permutation.

2.2 Construction of the initial solution:

The initial solution is built in two steps: a nearest-neighbor truck tour and a greedy assignment of drone sorties.

2.2.1 Nearest-neighbor heuristic for the truck:

The procedure `build_initial_perm` constructs a permutation using a nearest-neighbor rule based on **truck travel times**:

1. Start at the depot sss.
2. While there are unvisited customers, choose as next node the customer with minimum truck travel time from the current node.

3. Append this customer to the permutation and update the current node.

The resulting permutation defines a reasonably short truck tour that is later improved by Tabu Search.

2.2.2 Greedy assignment of drone sorties:

Given a truck route [s,…,t], the procedure `assign_sorties_greedy` scans consecutive triplets (L,h,R) of nodes in the route and tests whether customer h can be removed from the truck route and served by the drone through a sortie:

- **Feasibility checks**:
  - h is not truck-only.
  - Drone arcs L→h and h→R exist and have positive finite time.
  - The drone flight time $d_{Lh}+d_{hR}$ does not exceed the endurance E.
  - The direct truck arc L→RL is valid.
- **Approximate benefit check**:
  Let $t_{old} = t_{Lh}+t_{hR}$ be the truck time when h is served by the truck, and $t_{new}=t_{LR}+SL_{eff}+SR$ the time with a drone sortie (including launch and recovery). The sortie is accepted only if:

  $$t_{old}>t_{new}$$

  Where the first sortie launched from the depot may have $SL_{eff}=0$.

When a sortie is accepted, hhh is removed from the route and the corresponding `DroneSortie` is stored. At the end we obtain a reduced truck route plus a set of sorties that are locally beneficial and satisfy the endurance constraint.

2.3 Evaluation function: truck–drone simulation:

To evaluate a solution we must compute the **total completion time** under all synchronization and endurance constraints. This is done by the function `simulate_route`, which performs an explicit simulation along the truck route and the list of sorties.

Given:

- a truck route `truck_route = [s, ..., t]`
- a set of sorties

The simulation proceeds edge by edge:

1. **Service consistency:**
   It first checks that each customer is served exactly once: either appears in the internal nodes of the truck route or as the customer hhh of

exactly one sortie. No truck-only customer may be assigned to the drone.

2. **Mapping of sorties:**
   For each sortie (L_index,h,R_index) we record:
   - the edge index where the drone is launched (between `truck_route[L_index]` and `truck_route[L_index+1]`),
   - the edge index where it must be recovered.
     If two sorties use the same launch or recovery index, the solution is declared infeasible.

3. **Time evolution:**
   The global clock is initialized at zero. For each edge i→j in the truck route:
   - If a sortie starts here, we add the corresponding launch service time (zero for the first sortie from the depot, SL otherwise), and compute the drone arrival time at the recovery node, checking the endurance on pure flight time.
   - The truck then travels from i to j; its travel time is added to the clock.
   - If a sortie is supposed to land at node j, we compare truck and drone arrival times. If the drone arrives later, the truck waits. The total airborne time (from launch to meeting) must not exceed E. The recovery service time SR is then added.

4. **End of the route:**
   At the final depot, all sorties must have been completed; otherwise the solution is infeasible.

If any feasibility check fails, the function returns `math.inf`. Otherwise, it returns the final value of the global clock, which is the **objective value** of the solution. The evaluation of a permutation `perm` is performed by `evaluate_perm`, which builds the basic truck route, applies the greedy assignment of sorties, and calls `simulate_route`.

2.4 Tabu Search framework:

The core search procedure is a **Tabu Search** operating on permutations of customers.

- **State**: current permutation `current_perm`.
- **Neighborhood**: permutations obtained by swapping the customers in two positions i and j.
- **Evaluation**: `evaluate_perm(instance, perm)`.
  Tabu list: recently used swaps, stored as unordered pairs of customer IDs (a,b).
- **Aspiration**: tabu moves are allowed if they produce a solution better than the current global best.

Each iteration samples up to `MAX_NEIGHBORS` swaps:

1. Randomly select positions $i \neq j$ and build the neighbor by swapping `perm[i]` and `perm[j]`.
2. Encode the move as a pair $(\min(a,b), \max(a,b))$, where a and b are the swapped customers.
3. Evaluate the neighbor. Infeasible neighbors (infinite cost) are discarded.
4. If the move is tabu and does not improve the global best cost, it is skipped.
5. Among all admissible neighbors, we keep the one with minimum cost as the **best candidate**.

If at least one feasible candidate is found, the algorithm moves to the best candidate and inserts the corresponding move into the tabu list with an expiry iteration `iteration + TABU_TENURE`. Periodically, expired entries are removed. If no feasible neighbor is available in the current iteration, the algorithm performs a **restart** from a new nearest-neighbor permutation.

The search stops when one of the following conditions is satisfied:

- the number of iterations reaches `MAX_ITERS`, or
- the CPU time of the run exceeds `TIME_LIMIT_TS_RUN`.

At the end of each run, the final best permutation is reevaluated to extract the corresponding truck route, sorties and objective value, together with the total number of evaluated neighbors (`expansions`).

3. Experimental evaluation:

3.1 Experimental setup:

The heuristic is implemented in **Python**, using only standard libraries plus `matplotlib` for plotting. Instances are read from the `instances` directory in the format provided by the project repository, and all results are written to a `results` directory, with one subfolder per instance. For each instance we perform **10 independent runs** of Tabu Search, each started with a different random seed.

The main Tabu Search parameters are:

- `MAX_ITERS = 1000` iterations,
- `MAX_NEIGHBORS = 200` neighbors evaluated per iteration,
- `TABU_TENURE = 20` iterations,
- `TIME_LIMIT_TS_RUN = 600` seconds per run.

These values were chosen to obtain a reasonable exploration of the search space while respecting the global time limit imposed by the assignment.

3.2 Benchmark instances:

The benchmark set is the one specified in the project statement, consisting of 10 FSTSP instances with different sizes and drone speed configurations: 40, 60 and 80 nodes and two speed settings. For each instance an **optimal objective value** was obtained by solving the MILP model with Gurobi under a 1-hour time limit, and these values are provided in the assignment. In the code, they are stored in the dictionary `OPTIMAL_OBJECTIVES` and are used to compute the gaps of the heuristic solutions with respect to the optimum.

3.3 Performance metrics:

For each instance and each run we record:

- **Objective value**: total completion time of the final solution.
- **Runtime**: CPU time of the run in seconds.
- **Expansions**: number of neighbor evaluations (calls to `evaluate_perm`).
- **Truck and drone workload**: number of customers served by the truck and by the drone.

Using only runs with finite cost (feasible solutions), we compute:

- the **best cost**,
- the **average cost** and its standard deviation,
- the **average runtime** and its standard deviation,
- the **average expansions** and its standard deviation.

When the optimal value $z^{\backslash *}$ is available, we also compute:

- **GAP(best)** = 100 · (best_cost – $z^{\backslash *}$) / $z^{\backslash *}$
- **GAP(avg)** = 100 · (avg_cost – $z^{\backslash *}$) / $z^{\backslash *}$
- **best/optimal ratio** = best_cost / $z^{\backslash *}$

These metrics allow us to analyze both solution quality and computational effort.

Table 1 reports, for each benchmark instance, the Gurobi optimal objective value, the best and average objective values obtained over the 10 runs of the Tabu Search heuristic, the corresponding percentage gaps, and the average runtime and number of neighbor evaluations. This table will be used as the main numerical summary of the experimental results.

instance	opt_value	best_cost	avg_cost	gap_best	gap_avg	avg_time	avg_exp
FSTSP-40-1-1	241.0	300.0	328.7	24.480	36.39	9.04	194884.9
FSTSP-40-1-2	210.0	231.0	254.2	10.0	21.05	9.13	194880.1
FSTSP-60-1-1	321.0	433.0	464.8	35.29	44.8	11.48	196601.1
FSTSP-60-1-2	282.0	331.0	363.3	17.38	29.23	13.31	196601.3
FSTSP-60-2-1	305.0	387.0	393.4	27.29	29.38	12.34	196602.1
FSTSP-60-2-2	286.0	323.0	338.9	13.34	18.5	12.25	196602.1
FSTSP-80-1-1	357.0	531.0	590.7	49.14	65.46	16.37	197472.8
FSTSP-80-1-2	345.0	441.0	490.2	28.23	42.09	19.18	197483.8
FSTSP-80-2-1	385.0	542.0	577.4	41.18	50.37	20.18	197472.8
FSTSP-80-2-2	375.0	457.0	500.3	22.27	33.41	17.17	197478.2

Table 1: Summary of the heuristic performance on all benchmark instances. For each instance we report the Gurobi optimal value, the best and average objective values over 10 runs, the percentage gaps with respect to the optimum, and the average runtime and number of neighbor evaluations.

3.4 Generated plots:

For each instance, several figures are produced:

- **Costs per run**: bar chart showing the objective value of each feasible run.
- **Runtimes per run**: bar chart of CPU time per run.
- **Expansions per run**: bar chart of the number of evaluated neighbors.
- **Truck vs drone customers per run**: grouped bar chart showing, for each run, how many customers are served by the truck and how many by the drone.
- **Routes of all runs**: a multi-subplot figure where each subplot displays the final truck route in red and drone sorties in dashed lines; the best solution is highlighted.

Across instances with known optimal values we also generate:

- a **GAP per instance** plot, with bars for GAP(best) and GAP(avg),
- and a **best/optimal ratio per instance** plot.

To obtain a compact view of the solution quality across all benchmark instances, Figure 1 displays the GAP(best) and GAP(avg) with respect to the Gurobi optimum. In addition, Figure 2 shows the ratio between the best heuristic cost and the optimal cost; values close to 1.0 indicate that the heuristic is able to reproduce almost optimal solutions.

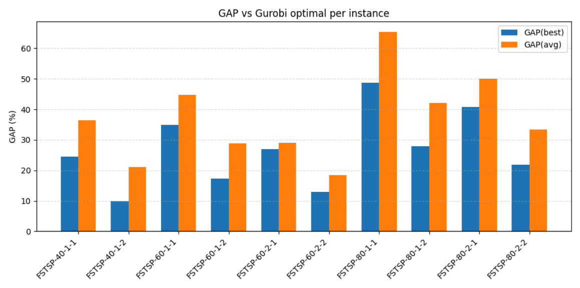

Figure 1: GAP(best) and GAP(avg) of the Tabu Search heuristic with respect to the Gurobi optimal value for each benchmark instance.

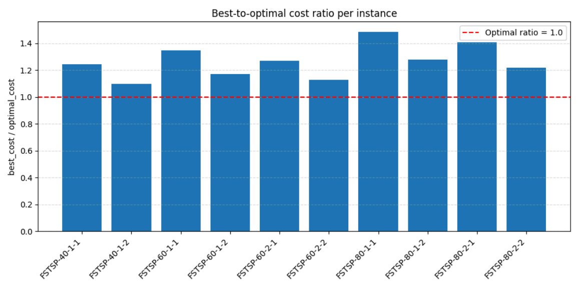

Figure 2: Ratio between the best heuristic objective value and the Gurobi optimal value for each benchmark instance. A ratio close to 1.0 indicates that the heuristic solution is nearly optimal.

In this report, these global plots are complemented by Table 1, which provides the detailed numerical results for each instance.

3.5 Qualitative discussion of results:

The experimental analysis allows us to study several aspects of the behavior of the Tabu Search heuristic:

- **Solution quality vs. Gurobi:**
  Using the optimal values provided in the project statement, we can measure how close the heuristic gets to the optimum. Table 1 and Figures 1–2 show, for each instance, how the best and average solutions compare to Gurobi. This highlights which instances are more challenging and where the heuristic has room for improvement.

- **Variability across runs:**
  The bar chart of costs per run, together with the standard deviation of the objective values, indicates how sensitive the algorithm is to its random choices. A small variability suggests that the heuristic is robust, while a large variability reveals that some runs fall into poor local minima and that multiple restarts are useful.

- **Computational effort:**
  The runtime and expansions plots show the trade-off between search intensity and time. They help verify that the algorithm respects the time

limits of the assignment and give an indication of how many neighbor evaluations are typically needed to reach a good solution.

- **Role of the drone:**
  The truck-vs-drone customers plot shows how often the greedy rule decides to serve a customer by the drone instead of the truck. This helps understand whether the heuristic is conservative (few sorties, strong focus on feasibility) or aggressive (many sorties) and how this choice affects the final cost.

Overall, the combination of metrics, plots and tables provides a clear picture of the strengths and limitations of the proposed Tabu Search on the given benchmark set.

4. Conclusions:

This project has presented a **Tabu Search heuristic** for the Flying Sidekick Traveling Salesperson Problem, in which a truck and a drone cooperate to deliver goods to a set of customers. Solutions are encoded as permutations of customers, from which a greedy procedure extracts drone sorties of the form (L,h,R). A detailed simulation model evaluates each candidate solution, enforcing all synchronization and endurance constraints and computing the total completion time.

The proposed method has several attractive features. The solution representation is simple, the neighborhood (swapping two customers) is easy to implement, and the greedy assignment of sorties avoids an explicit combinatorial explosion in the truck–drone decisions. The simulation-based evaluation ensures that all feasibility conditions are checked consistently. On the other hand, the heuristic also has limitations: the neighborhood is relatively small, the greedy rule only looks at local patterns, and the Tabu Search parameters are fixed rather than tuned or adapted dynamically.

The computational study on the benchmark instances shows how the algorithm behaves in practice in terms of solution quality, gaps with respect to Gurobi, runtimes, and number of evaluated neighbors. The analysis of truck and drone workload also reveals how often the heuristic exploits the drone to speed up the tour.

Several extensions are possible. Future work could incorporate richer neighborhoods (e.g., insertion or 2-opt moves), more advanced diversification and intensification strategies, or a more global assignment of drone sorties. Hybrid approaches that combine the metaheuristic with MILP components on restricted subproblems could also be explored. Despite its simplicity, the algorithm developed in this project provides a useful baseline for solving the FSTSP under strict time limits and illustrates how metaheuristics can be effectively applied to complex vehicle routing problems.

5. Annex – Source Code Repository

The source code for this project is publicly available in a GitHub repository. The repository includes:

- The full implementation of the Tabu Search heuristic.
- The benchmark instances used in the experiments.
- The scripts for generating all figures and tables included in this report.
- Execution instructions and project documentation.

The repository can be accessed at the following link:

https://github.com/m-estevezdiaz/FSTSP-Tabu-Search.git