Santa's Workshop Tour Challenge

Oscar Aguilar
Management Science & Engineering
Stanford University
Stanford, CA 94304
osthoag@stanford.edu

Louis Gautier
Management Science & Engineering
Stanford University
Stanford, CA 94304
lgautier@stanford.edu

Abstract

This report presents our approach to solving the Santa's Workshop Tour Kaggle challenge, a constrained optimization problem involving matching families to visit days at Santa's workshop. We initially applied a mixed integer linear programming (MILP) model using Gurobi but encountered intractability due to the problem's size. To improve the solution, we employed simulated annealing and a genetic algorithm as local search meta-heuristics. While these methods provided modest objective value improvements, we aimed to find the global optimum and explored ways to reduce the feasible region using an ILP formulation. By restricting the ILP and adjusting parameters, we achieved significant enhancements. This report demonstrates the effectiveness of combining MILP, local search, and problem-specific modifications to achieve optimized solutions for large-scale optimization challenges.

1 Problem description

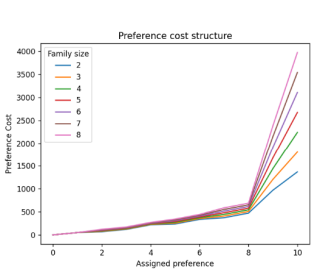
This report describes our approach to tackling the 2019 edition of the annual Kaggle optimization challenge, the Santa's Workshop Tour Kaggle challenge[3]. This constrained optimization problem consists in matching families to days when they can visit Santa's workshop, given a list of preferred days for each family, capacity constraints, and operational costs linked to the number of customers visiting the workshop each day. This problem is associated with one large instance available on Kaggle containing the preference lists for 5,000 different families.

Decision variable The goal is to assign each family i for $i = 1, \ldots, 5000$ with $n_i \in [|2, 8|]$ family members to one of the $d = 100, \ldots, 1$ days before Christmas (Christmas being day 1) to visit Santa's workshop. If encoding the assignment of family i to day d by the binary variable $x_{i,d}$, there are 500,000 binary decision variables in this problem.

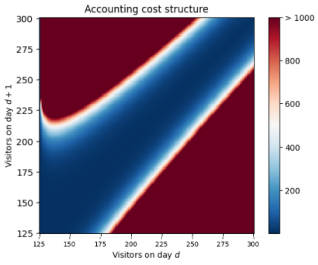
Objective function The objective function consists of the sum of two terms. The preference cost represents the cost associated with meeting the preferences of the families. It encourages the assignment of families to their preferred choices and penalizes deviations from these preferences. It is made of a fixed part and a marginal cost proportional to the number of family members. The non-linear accounting cost represents operational costs linked to opening and maintaining Santa's workshop. It is computed as a non-linear function of the number of people visiting the workshop on day d N_d and on the following day $d + 1$ N_{d+1}: $A = \sum_{d=100}^{1} \frac{(N_d - 125)}{400} N_d^{\left(\frac{1}{2} + \frac{|N_d - N_{d+1}|}{50}\right)}$. This function encourages smoothing the number of people scheduled on consecutive days. Both of these functions can be visualized in figure 1a and 1b.

As it can be seen in figure 1c, the demand for each day is very heterogeneous and most of the families tend to have similar preference patterns, prioritizing weekends and days closer to Christmas.

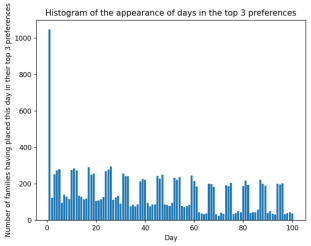
37th Conference on Neural Information Processing Systems (NeurIPS 2023).



(a) Preference cost structure



(b) Accounting cost structure



(c) Histogram of the three first preferences of each family

Figure 1: Structure of Santa's Workshop Problem

Constraints On each day, capacity constraints are enforced: the number of people visiting Santa's workshop should be between 125 and 300.

This setting is very interesting from an applicative point of view, as a lot of assignment problems have the same structure: customers with similar preferences are competing to get access to some resources at some high-demand times whereas capacity constraints and operational costs make the amount of resources quite inelastic. The instance we have to solve is very large and intractable with modern MILP solvers like Gurobi, as many application problems.

2 Modeling as an ILP

We translated the above problem into the following complete formulation as an Integer Linear Program. The key step in converting the problem to an ILP is to observe that the domain of the non-linear accounting cost function is restricted to $\{125, \ldots, 300\}^2$. Therefore, it can be discretized by creating indicator variables for each possible value in the domain and storing the associated cost coefficients. We realize this discretization of the accounting cost thanks to the helper variable $y_{d,i,j}$, that encodes the information of the amount of people scheduled on consecutive days. Our ILP formulation of the problem for the instance posted on Kaggle is the following:

Variables

- Decision variable $x_{f,d} \in \{0, 1\}$: Indicator that family f is assigned to day d
- Helper variable $y_{d,i,j} \in \{0, 1\}$: Indicator that day d has i people and day $d + 1$ has j people scheduled

Parameters given in the instance

- $n = 5000$: Total number of families
- $m = 100$: Total number of days
- $l = 125$: Minimum number of people assigned to a day
- $u = 300$: Maximum number of people assigned to a day
- a_f: Number of people in family f
- $b_{f,d}$: The preference cost of assigning family f to day d
- $c_{i,j}$: The accounting cost of having i people in a day followed by j people the next day

Objective

$$\text{minimize} \quad \sum_{f=1}^{n} \sum_{d=1}^{m} x_{f,d} b_{f,d} + \sum_{d=1}^{m} \sum_{i=l}^{u} \sum_{j=l}^{u} y_{d,i,j} c_{i,j}$$

Constraints

- Each family is only assigned one day: $\sum_{d=1}^{m} x_{f,d} = 1$, $f = 1, \ldots, n$

- The capacity constraints are verified: $l \leq \sum_{f=1}^{n} a_f x_{f,d} \leq u$, $d = 1, \ldots, m$
- The indicator of crowd size is consistent: $\sum_{i=l}^{u} \sum_{j=l}^{u} i \cdot y_{d,i,j} = \sum_{f=1}^{n} a_f x_{f,d}$, $d = 1, \ldots, m$
- The indicator of crowd size is unique each day: $\sum_{i=l}^{u} \sum_{j=l}^{u} y_{d,i,j}$, $d = 1, \ldots, m$
- Contiguous days have consistent crowd size: $\sum_{i=l}^{u} y_{d,i,k} = \sum_{j=l}^{u} y_{d,k,j}$, $d = 1, \ldots, m-1$, $k = l, \ldots, u$
- An initial condition is imposed to compute accounting costs on day 100: $\sum_{i=l}^{u} y_{d,i,i} = 1$

3 Methodology

3.1 Attempting to solve the whole problem with a MILP solver

After modeling the previously described ILP with the library Gurobipy, we were able to launch the resolution of the problem using the Mixed Integer Linear Programming solver Gurobi. As expected, we found that this large-size problem is intractable and can't be solved to optimality in a reasonable amount of time. We gave the solver a 12 hours time budget and it was only able to go to a depth of two in the search tree, meaning it was only able to branch on two variables in the branch-and-bound algorithm. It indicates that the structure of the objective function in the search tree is very complex and that it is not easy to prune branches of the tree. Nevertheless, it found five feasible solutions (the first one after 8 hours of execution), including a seemingly very good one with an estimated optimality gap of 1.4%, an objective value of 69,341 and a lower bound of 68,382. To try to improve this solution, we will have to resort to other methods, and especially local search methods relying on some meta-heuristics, due to the size of our problem exceeding the capacities of Gurobi.

3.2 Metaheuristics: local search strategies

We tried to improve the good solution that we obtained with Gurobi by using local search methods relying on different meta-heuristics.

3.2.1 Simulated annealing

The first method that we tried was simulated annealing [7], a probabilistic meta-heuristic that allows to explore the space of feasible solutions by iteratively perturbing solutions and accepting the perturbations either if they decrease the objective value, or with a decaying probability if they increase it. At each iteration n, worse solutions are accepted with probability $p_n = exp(-\frac{\Delta E}{T(n)})$, where ΔE is the difference in objective value induced by the perturbation and $T(n)$ is a parameter called temperature, which is decayed throughout the execution of the algorithm. Accepting worse solutions enables the algorithm to escape local optima and search for potentially better solutions in different regions of the search space. Decaying $T(n)$ allows to gradually reduce the exploration intensity during the search. It is worth noting that simulated annealing was primarily designed for continuous optimization problems, and will work best if we define the neighborhood of a solution (the possible perturbations that can be applied to a solution) in a way that close solutions lead to a close objective value.

Perturbation scheme Therefore, we defined the possible perturbations to a solution so that they transform a feasible solution into another feasible solution, and they minimally modify the objective value. The perturbation procedure we chose is thus based on preference levels swaps:

1. Uniformly sample two families $i \neq j \in [|1, n|]$
2. If these two families were assigned different preference levels, swap this assignment. If it leads to an infeasible solution, return to step 1.
3. If these two families were assigned the same preference level, pick one of them uniformly and choose to increment its preference by 1 with probability 1/2, or to decrement it otherwise. If this perturbation is infeasible, return to step 1.

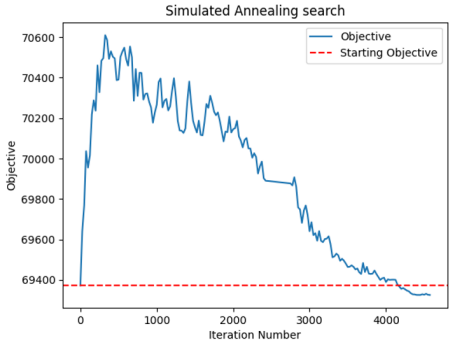


Figure 2: Evolution of the objective value with Simulated Annealing over 5,000 iterations with a linear temperature decay

As a result, the preference cost remains constant when $i \neq j$, and the accounting cost does not vary a lot when $i = j$, which leads the objective function to be relatively smooth in the neighborhood of a solution. This perturbation procedure should also theoretically allow us to explore the whole feasible solution space, as we can have an arbitrary number of families assigned to each preference level thanks to the case where $i = j$.

The last step to be able to implement this algorithm was to efficiently compute the change in objective value induced by a perturbation, without having to read the entire solution again. This was made by enumerating the numerous cases that can occur when swapping the preferences of two families.

Analysis of results We experimented with various ways of decaying the temperature, with different initial and final temperature values, but the in the end, the most efficient decaying procedure was a linear decay $T(n) = T_0 - (T_0 - T_f)\frac{n}{N}$, with $T_0 = 5$, $T_f = 0.01$, and $N = 5000$. The evolution of the iterates of objective values is presented in figure 2. As can be seen, the acceptance of worse solutions at the beginning of the search allowed us to leave the local minimum that had been given by the solution found by Gurobi, and the exploration of the search space of feasible solutions eventually led to a solution with a better objective value than the initial one. However, the improvement of the objective was modest, as it only allowed to decrease the objective value from 69,341 to 69,325 (with a best lower bound at 68,382), and this local search procedure took quite a lot of time, as 10 hours were needed to find 5,000 accepted perturbations.

3.2.2 Genetic algorithm

This led us to use another local search meta-heuristic, namely a genetic algorithm. Instead of updating a single solution like in simulated annealing, it keeps track of a population of solutions and modifies it according to procedures inspired from genetics and evolution.

Perturbations definitions We took inspiration from [6] to design our genetic algorithm. Several procedures are required to make a population of solutions converge to a local minimum:

- A **crossover** operator, which, given two uniformly picked solutions in the population (called *parents*), produces a child solution (*offspring*) by combining the content of the two parents solution. We used the uniform crossover operator illustrated in figure 3: for each family, we choose its assignment in the first parent solution with probability 1/2 and in the second parent solution otherwise. We keep sampling binomial random variables that give us this mapping for all families until we found a feasible child solution. Finally, this operator computes the objective value for the newly created child solution.
- A **mutate** operator, which slightly perturbs one solution in the population. Similarly to one type of perturbations in our simulated annealing procedure, it picks a family uniformly at random in the solution and increments or decrements its assigned level of preference with probability 1/2. Once again, we keep uniformly sampling families until we find a feasible perturbation.

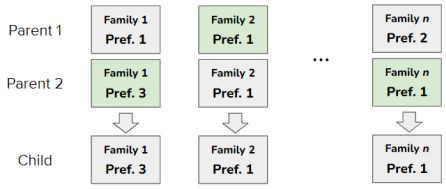


Figure 3: Illustration of the uniform crossover procedure

- A **select** operator, which keeps only the fittest individuals in the population, in our context which keeps only the solutions with the lowest objective value in our population.

- An **initialization** procedure to initialize the population. In our case, we started five simulated annealing searches around the five solutions given by Gurobi, and we used some regularly spaced iterates of these searches as an initialization population.

The algorithm then combines these operators in the way described in algorithm 1.

Algorithm 1: Pseudocode of our genetic algorithm

```
population = initialize_population();
while population has not converged do
    SELECT(population);
    CROSSOVER(population);
    MUTATE(population);
end
```

Results Figure 4 presents the evolution of the minimum and the average of the objective values of the solutions in the population with a population size of 50. In the first iterations, the best solution found by Gurobi is mutated and modified as part of the algorithm's exploration mechanism. That's why, similarly to the simulated annealing iterates profile, the best objective in the population first increases before sharply decreasing. The mean objective value and best objective value in the population first decrease fast, before decreasing much more slowly as the diversity in the population gets smaller, as it takes more time to find mutations that improve the objective value. In the end, the improvement of the objective was also modest, as it allowed to decrease the objective value from 69,341 (initial best solution found by Gurobi) to 69,319.

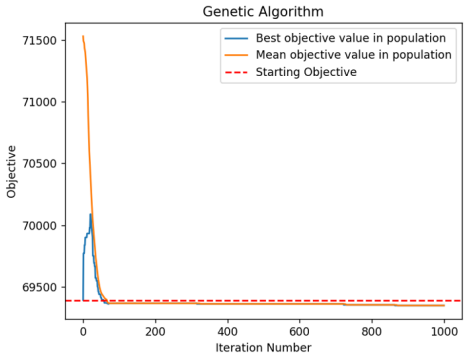


Figure 4: Evolution of the mean and best objective values in solution population over 1,000 iterations

3.3 Tractability: reducing the ILP size

3.3.1 Reduction of variables

With the above heuristics, we were able to reliably find local minima. However, we still had our sights set on finding the global optimum. Thus, we turned our eyes back to the original ILP an looked for ways to reduce the size of the feasible region, inspired by discussion in the Kaggle community.[1] First, we noted that instead of tracking the exact day to which each family is assigned, we can track which day preference each family received and bucket any days outside of the top ten into day "11" with the same worst-case cost. Accordingly, we introduced the following changes to the ILP formulation:

New Variables

- Decision variable $x_{f,p} \in \{0, 1\}$: Indicator that family f is assigned preference p
- Helper variable $z_d \in \mathbb{R}^n_+$: Number of people assigned to day d

New Parameters

- $o = 11$: Range of preferences for each family
- $P_{f,p}$: The p^{th} day preference of family f
- $b_{f,p}$: The cost of assigning family f their p^{th} day preference

Modified Objective

$$\text{minimize} \quad \sum_{f=1}^{n}\sum_{p=1}^{o} x_{f,p} b_{f,p} + \sum_{d=1}^{m}\sum_{i=l}^{u}\sum_{j=l}^{u} y_{d,i,j} c_{i,j}$$

New or Modified Constraints

- Preference assignments for matched families lower bound the total assignment for each day: $\sum_{f=1}^{n} \sum_{\{p|d=P_{f,p}\}}^{o-1} a_f x_{f,p} \leq z_d \quad d = 1, \ldots, m$
- The total day assignments equals the total number of people: $\sum_{d=1}^{m} z_d = \sum_{f=1}^{n} a_f$

While this formulation reduces the number of binary decision variables x from $n \times p$ to $n \times o$ (almost tenfold), it is not a complete formulation since we are not directly generating a policy of assigning families to days. For example, if an odd number of "unmatched" people are assigned to a day, but all unmatched families have an even number of people the solution found would be infeasible. In practice, we did not have any unmatched families; in fact, all families got assigned preferences 1 - 5.

The next thing that we noted is that with an upper bound on accounting cost UB_{AC}, we can remove many of the variables $y_{d,i,j}$. For this, we need an upper bound on total cost UB_C and a lower bound on

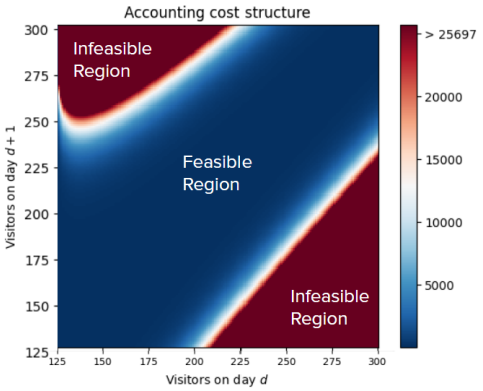


Figure 5: Feasible region for the variables $y_{d,i,j}$ with accounting threshold 25697

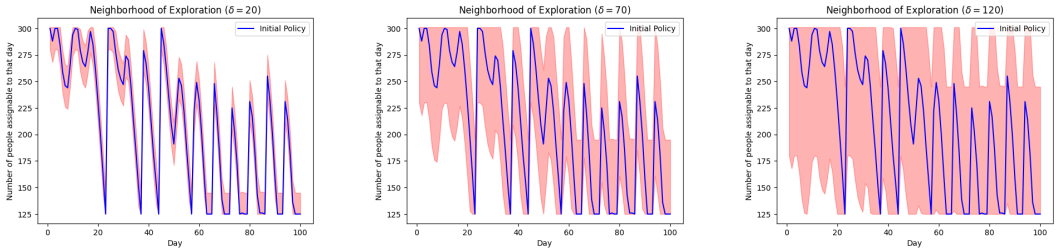
the preference cost LB_{PC}. The former can be taken as the objective value from a previously obtained feasible point. For this, we used the solution obtained by the genetic algorithm $UB_{TC} = 69319$. The latter amounted to removing the $y_{d,i,j}$ variables from the ILP along with the associated constraints and objective function. The result of solving this partial ILP is the lowest possible preference cost, with an optimal value of $LB_{PC} = PC_* = 43622$. Thus, we obtain the following upper bound on accounting cost: $UB_{\text{AC}} = UB_{\text{C}} - LB_{\text{PC}} = 69319 - 43622 = 25697$ Removing the indicator variables $y_{d,i,j}$ with accounting cost greater than UB_{AC} reduced the number from 3097600 to 2303914, over a 25% decrease. The new feasible region can be visualized in figure 5.

3.3.2 Neighborhood search

Finally, we further restricted the size of y further by limiting exploration to the neighborhood of an initial solution $\hat{x}, \hat{y}, \hat{z}$. In order to do this, we defined a distance $\delta \in \{0, 1, \ldots, l\}$, such that the number of people assigned to a day d (z_d) cannot differ from the assignments in the initial solution ($\hat{z}_d$) by δ. The neighborhood of the solution from the genetic algorithm is visualized in figure 6.

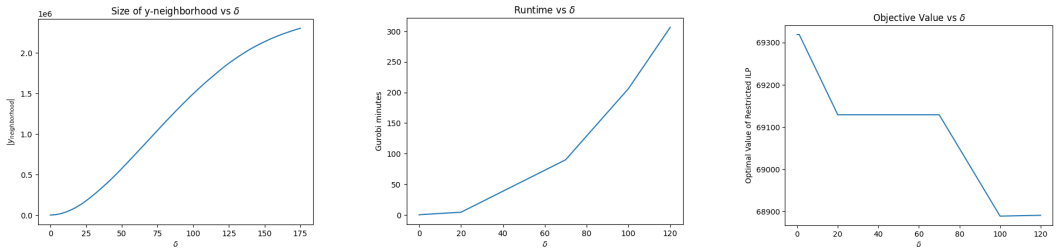
The result of compiling the restrictions on our original ILP yields the following formulation:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{f=1}^{n}\sum_{p=1}^{o} x_{f,p} b_{f,p} + \sum_{d=1}^{m}\sum_{i=l}^{u}\sum_{j=l}^{u} y_{d,i,j} c_{i,j} \\
\text{subject to} \quad & \sum_{p=1}^{o} x_{f,p} = 1 \quad f = 1, \ldots, n \\
& \sum_{f=1}^{n} \sum_{\{p \mid d = P_{f,p}\}}^{o-1} a_f \cdot x_{f,p} \leq z_d \quad d = 1, \ldots, m \\
& \sum_{d=1}^{m} z_d = \sum_{f=1}^{n} a_f \\
& \max\{l, \hat{z}_d - \delta\} \leq z_d \leq \min\{u, \hat{z}_d + \delta\} \quad d = 1, \ldots, m \\
& \sum_{i=l}^{u}\sum_{j=l}^{u} i \cdot y_{d,i,j} = z_d \quad d = 1, \ldots, m \\
& c_{i,j} \cdot y_{d,i,j} \leq UB_{\text{AC}} \quad d = 1, \ldots, m \quad i, j = l, \ldots, u \\
& \sum_{i=l}^{u}\sum_{j=l}^{u} y_{d,i,j} \quad d = 1, \ldots, m \\
& \sum_{i=l}^{u} y_{d,i,k} = \sum_{j=l}^{u} y_{d,k,j} \quad d = 1, \ldots, m-1 \quad k = l, \ldots, u \\
& \sum_{i=l}^{u} y_{m,i,i} = 1
\end{aligned}
$$



(a) Neighborhood with $\delta = 20$ (b) Neighborhood with $\delta = 70$ (c) Neighborhood with $\delta = 120$

Figure 6: The restricted neighborhood around the genetic algorithm solution for different δ



(a) Number of $y_{d,i,j}$ versus δ (b) Gurobi ILP runtime versus δ (c) Optimal Value versus δ

Figure 7: The scalability and solutions of the restricted ILP for multiple values of δ

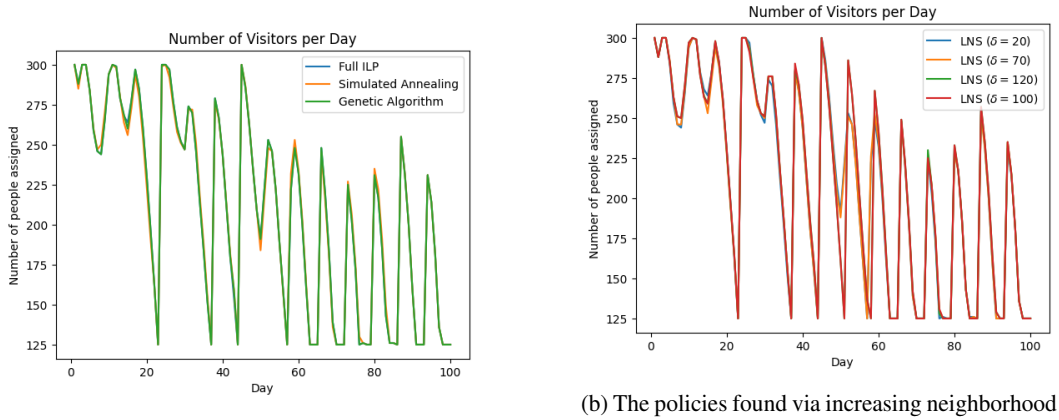
We fed this restricted ILP to Gurobi for multiple values of δ. Of course, the size (and tractability) scales with δ; this is visualized in figure 7a and 7b. However, the optimal solutions found improved drastically between certain intervals; the results are plotted in figure 7c. In the end, the best solution that we found for this challenge came from running the restricted ILP with $\delta = 100$, and took Gurobi 3 hours to produce with an objective value of 68,889 and duality gap of 0.01%.

4 Conclusion: comparison of results

Method	**Objective Value**
Gurobi with 12 hours budget	69,341
Simulated Annealing	69,325
Genetic Algorithm	69,319
Neighborhood Search with $\delta = 100$	68,889

Table 1: Comparison of our methods

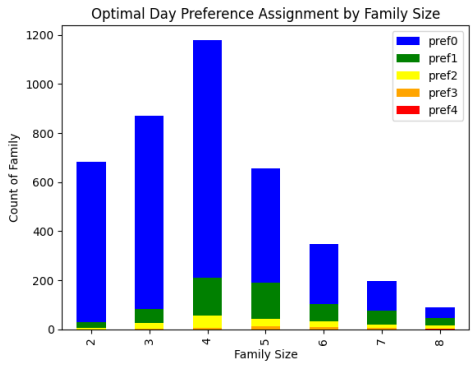
The results of the different methods we tried are summarized in the table 1. It is clear that looking for a global optimum through an expanding neighborhood search gave the best results. Additionally, the crowd assignments per day obtained through each of the methods tried are displayed in figure 8. The similar structure across all policies found emphasises the effectiveness of local search methods, as the best way to improve the objective was mainly to shuffle families around and gradually lower the preference cost. Finally, the optimal policy is visualized in figure 9. Unsurprisingly, all of the accounting costs fall within a narrow region with the highest being 237. Similarly, in the optimal preference assignment most families get their first choice of day. However, the proportion of families getting their top choices decreases with the size of the family, likely due to the ease of moving smaller



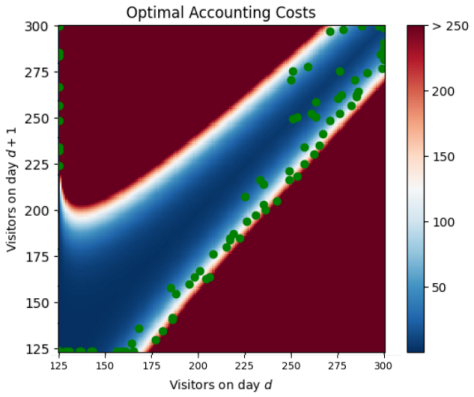
(a) The policies found via local search

(b) The policies found via increasing neighborhood search

Figure 8: The number of visitors assigned to each day by solutions from the different search methods



(a) The optimal preference assignments broken down by family size



(b) The optimal incurred accounting costs

Figure 9: The structure of the optimal solution

families around and the decreasing marginal cost (per person) with increased family size. To allow for easy replication of our results, all of our code is available on Github.[5]

In conclusion, this report outlines our journey in tackling the Santa's Workshop Tour Kaggle challenge, a complex optimization problem with constraints and operational costs. Initially, we employed an ILP model using Gurobi, which provided feasible solutions but fell short of finding the global optimum within a reasonable time frame. To enhance our results, we explored alternative methods such as simulated annealing and a genetic algorithm, which allowed us to find improved local minima. However, the gains achieved through these approaches were modest. With the aim of obtaining the global optimum, we revisited the ILP formulation and substantially reduced the number of decision variables through various techniques. By using the restricted ILP with a value of $\delta = 100$, we obtained the best solution, which Gurobi computed in 3 hours, yielding an objective value of 68,889 and a duality gap of 0.01%. This accomplishment highlights the significance of combining different optimization techniques and adapting the problem formulation to strike a balance between computational efficiency and solution quality. A potential future avenue of improvement for solving this problem more efficiently could be a clever formulation of column generation via Dantzig-Wolfe Decomposition[2] and/or implementing a branch and price algorithm[4]. However, this make for an entire project in it of itself. Thus, we were not able to explore such techniques too deeply. Overall, the insights gained from this project serve as valuable lessons in practical approaches to tackling Mixed Integer Linear Programming problems.

References

[1] Santa's workshop tour 2019 discussion.

[2] George B. Dantzig and Philip Wolfe. Decomposition principle for linear programs. *Operations Research*, 8(1):101–111, Feb 1960.

[3] inversion Elizabeth Park. Santa's workshop tour 2019, 2019.

[4] vorgelegt von Gerald Gamrath. *Generic Branch-Cut-and-Price*. Zib | Zuse Institute Berlin (ZIB), Mar 2010.

[5] Louis Gautier and Oscar Aguilar. Louis-gautier/ms-e311$_santa_project, Jun$2023.

[6] Sourabh Katoch, Sumit Singh Chauhan, and Vijay Kumar. A review on genetic algorithm: past, present, and future. *Multimedia Tools and Applications*, 80:8091–8126, 2021.

[7] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.