Abstract—Carpooling has been long deemed a promising approach to better utilizing existing transportation infrastructure. However, there are several reasons carpooling is still not the preferred mode of commute in the United States: first, complex human factors, including time constraints and not having right incentive structures, discourage the sharing of rides; second, algorithmic and technical barriers inhibit the development of online services for matching riders. In this work, we study algorithms for 3+ high-occupancy vehicle (HOV) lanes, which permit vehicles that hold three or more people. We focus on the technical barriers but also address the aforementioned human factors. We formulate the HOV3 Carpool problem, and show that it is NP-Complete. We thus pose the relaxed problem HOV3- Carpool problem, allowing groups of up to size three, and propose several methods for solving the problem of finding globally optimal carpool groups that may utilize these 3- HOV lanes. Our methods include local search, integer programming, and dynamic programming. Our local search methods include sampling-based (hill-climbing and simulated annealing), classical neighborhood search, and a hybrid random neighborhood search. We assess the methods numerically in terms of objective value and scalability. Our findings show that our sampling-based local search methods scale up to 100K agents, thereby improving upon related previous work (which studies up to 1000 agents). The hill climbing local search method converges significantly closer and faster towards a naive lower bound on cumulative carpooling cost.

I. INTRODUCTION

Transportation-related costs of air pollution, greenhouse gas emissions, noise, delay from traffic congestion, and losses and injury from collisions are estimated to be approaching $1.1 trillion annually in the US [1], [2], [3]. We investigate ridesharing [4], [5] as a promising path forward for reducing the multiple costs associated with transportation.

The main goal of our article is studying the carpooling incentive of high-occupancy-vehicle (HOV) lanes, and how such incentives can be algorithmically incorporated into ridesharing systems. Studying HOV lanes is compelling because 1) they have the potential to counter social barriers against carpooling, such as trust and personal commute preferences, by permitting the use of lower latency lanes, and 2) they provide a relatively easy to implement scheme, through the repurposing of existing infrastructure.

Commonly, HOV lanes may have restrictions which specify how many people must be in a vehicle in order to use the lane. In most locations, HOV lanes require at least two occupants. Increasingly, in bottleneck areas such as metropolitan areas, including the San Francisco Bay Area and Seattle, WA, HOV lanes are increasingly requiring three passengers in order to maintain high throughput. Optimizing our use of existing road network is a highly complex problem, but it starts with optimizing our use of the HOV lanes, which is the topic of this article. For a fixed set of drivers, the problem of having exactly two occupants in each vehicle reduces to bipartite graph matching, for which efficient algorithms exist. Remarkably, the problem of having exactly three occupants in each vehicle is NP-Complete, so it is very unlikely that an efficient algorithm can solve the problem exactly. In this article, we prove this complexity result and present algorithms to solve the problem approximately.

Contributions The main contributions of this article are:

• We formulate and present the HOV3 Carpool and HOV3- Carpool problems, which specifically integrate the carpooling incentive of 3+ high-occupancy vehicle (HOV) lanes.
• We prove that the HOV3 Carpool problem is NP-Complete by reduction from Exact Cover by 3-Set. In contrast, we demonstrate that the HOV3- Carpool problem is amenable to iterative methods such as local search.
• We devise four local search methods for the relaxed HOV3- Carpool problem: hill climbing, simulated annealing, classical local search, and local search with random neighborhood. The first two are sampling-based...
local search methods; the third is a classical local search approach; the fourth combines sampling with the classical local search approach.

- To demonstrate the limitations of classical and general methods for the carpool problem, we present for comparison a integer program (IP) formulation and a dynamic programming formulation for the HOV3- Carpool problem. Our IP formulation utilizes the problem structure to yield a concise representation.
- We empirically compare the above six methods on Euclidean and Bay Area synthetic workloads (simulated participants) of up to 100K agents. We show that the hill climbing method by far outperforms the rest in terms of scale, computational runtime, and convergence to a naive lower bound. For the largest problem size of 100K, the hill climbing method converges to a ratio close to 1 to the lower bound.

Related work Many variants of the carpooling problem have been studied, for instance focusing on maximizing the number of participants [8] or minimizing the total distance or time [9]; for more variants, we refer the reader to excellent surveys on ridesharing by [10] and [5]. Although there are numerous works studying the HOV or high-occupancy toll (HOT) lanes from the perspective of pricing [11], [12], we are not aware of any work that explicitly optimizes for its utilization.

Our work temporarily side-steps the pricing aspect of the carpooling problem to focus on the optimization aspect; however, since we optimize for the global cost, our work is compatible with the incentive-compatible pricing mechanism introduced by [13], based on the Vickery auction [14]. A closely related work [15] studies MILP and CP methods for a variant of the globally optimum problem where passengers travel to the pickup locations that are along the path of the driver. This tremendously simplifies the complexity of the problem but may provide a less seamless experience for passengers. We provide a formulation which permits driver deviations for passenger pickups.

Several related works have also shown most variants of the ridesharing problem to be NP-Hard and have pointed out some special cases to be solvable in polynomial time, for instance focusing on maximizing the number of passengers. We provide a formulation which permits driver deviations for passenger pickups.

A natural way to optimize for HOV-lane use is to optimize for the best configuration of carpools of size three. However, we prove in this section by reduction that this problem is NP-Hard. Subsequently, we study a relaxed version where we optimize for carpools of size up to (and including) three. The relaxed problem lends itself more easily to approximation methods such as local search.

A. Optimizing usage of the HOV3+ lane

In this article, we study the carpool incentive of permitting usage of specific lower latency lanes (often on the freeway) which require at least three occupants in a vehicle. Under such a policy, there is little incentive for vehicles to have more than three occupants, since each additional passenger adds additional divergence costs to the carpool group. Consequently, we reason that the best reasonably achievable solution would optimize for three occupants per vehicle.

Now we specialize to the HOV3 carpool problem, in which the goal is to find size-three ridesharing groups that minimize the overall cost of the system. Additionally, we consider time constraints.

Definition 1 (HOV3 carpool problem): Let $\mathcal{U}$ be the set of agents. Each agent is endowed with a home and a work location $(x^u_h, x^u_w), \forall u \in \mathcal{U}$ as well as common time windows $T^u = (t^u_1, t^u_2, \ldots), \forall u \in \mathcal{U}$ with $t^u_i = [t^u_i, t^u_i]$. The collection of feasible subsets of the universe (rideshare groups) $\mathcal{S}$ consists of groups of size 3 such that the agents have a non-empty common time window. That is,

$$\mathcal{S} := \left\{ S : S \subseteq 2^\mathcal{U}, |S| = 3, \bigcap_{u \in S} t^u \neq \emptyset \right\}.$$  

The overall objective of the HOV3 carpool problem is to find a subset $\mathcal{R} \subseteq \mathcal{S}$ such that

- $\mathcal{R}$ minimizes some given cost function $C : 2^\mathcal{S} \rightarrow \mathbb{R}$
- $\mathcal{R}$ forms a partition of $\mathcal{U}$, as no agent may participate simultaneously in multiple ridesharing groups.

The specific cost function we study is given in Equation 4. The search version (as opposed to the optimization version) of this problem, which excludes costs, is already NP-complete, as we will prove next in Section II-B.

B. Reduction from Exact Cover by 3-Set

The Exact Cover by 3-Set problem is a known NP-Complete problem (by generalization of Tripartite Matching, which is NP-Complete [18]). The problem is as follows: $F = \{S_1, \ldots, S_m\}$ of subsets of a finite set $U$ with $|S_i| = 3$ and $|U| = 3m$ for some $m$. Find $m$ sets in $F$ that are disjoint and have $U$ as their union.

We observe that Exact Cover by 3-Set problem is closely related to the search version of the HOV3 carpool problem, allowing us to show the following complexity result:

Theorem 1: HOV3 carpool search is NP-complete.

Proof: First, the problem is trivially in NP. Given a set of rideshare groups, it can be checked in polynomial time that groups indeed form a partition. Naively this can be done in $O(n^2)$. Furthermore, even if given a solution that may contain arbitrary subsets of $\mathcal{U}$ (not necessarily in $\mathcal{S}$), it can be checked in linear time if the groups satisfy time and size constraints (and thus are feasible subsets in $\mathcal{S}$).

Now, we show by reduction from Exact Cover by 3-Set that the HOV3 carpool problem is NP-Hard.
other time window used so far, and let $t$ be the time window the vehicle has no overlap with any other time window used so far, and let $t \in T^u, \forall u \in S_i$. This assignment ensures that $S_i \in F$ and the uniqueness of the time window ensures that $S_i \in S \implies S_i \in F$. Note that the size-3 constraint is satisfied by definition of the Exact Cover by 3-Set problem.

Corollary 2: The HOV3 carpool problem (optimization version) (see Definition 1), is NP-Hard.

In addition to the complexity result, the HOV3 Carpool problem has several additional pain points. First, simply finding a feasible solution to the HOV3 problem is difficult.

Lemma 3: Finding a feasible solution to the HOV3 problem (optimization version) is NP-Complete.

Proof: Finding a feasible solution is the same as the search version of the problem, which disregards the cost. Then, the result follows from Theorem 1.

Second, due to the partition constraint, it is difficult to use standard approximation techniques such as linear programming (LP) relaxation. Once the partition constraint is violated, such as through a LP rounding scheme, it is difficult to recover a partition; recovering it may be seen as another combinatorial optimization problem.

C. Relaxation of HOV3 carpool problem

Finally, we present the HOV3 carpool problem, which we study for the remainder of the article, referred to simply as the carpool problem. The only difference from the previous problem is that feasible rideshare groups may now have size less than or equal to three (\(|S| \leq 3\)), so due to space limitations, we only provide the new definition for the collection of feasible subsets $S$:

$$S := \left\{ S : S \in 2^U, |S| \leq 3, \bigcap_{u \in S} T^u \neq \emptyset \right\}. \quad (2)$$

D. Problem setting

Global optimum In this article, we study computing the globally optimal physical distance. In particular, any agent may be a driver or a passenger. The driver picks up all the passengers. The cost of a rideshare group is the distance traveled by the driver. The overall cost of a carpool solution is the sum of the cost of each rideshare group in the solution. That is, $c_S = \sum_{(u,v) \in \text{TSP}(S)} d_{uv}$

$$C = \sum_{S \in \mathcal{S}} c_S \quad (4)$$

where TSP($S$) denotes the solution to the Traveling Salesman Problem. Given a set of agents $S$, TSP($S$) computes the minimum cost tour (given as pairs of locations, i.e. $(x_1, x_2), (x_2, x_3), \ldots$). We study the global optimum in the sense that this objective minimizes the total vehicle distance traveled, and thus has implications for fuel consumption and greenhouse gas (GHG) emissions.

Capacity We assume that all agents have vehicles, that the capacity is three, that the vehicle capacity is uniform.

Single destination, single time window We specialize to the setting where all agents have the same destination and all agents are allowed to specify one time window.

Static We solve the static version of the problem, where travel is instantaneous; there is no time overhead for pickups.

III. Problem formulation

A. Carpool as a set partition problem

We give a natural 3-set partition formulation of the carpool problem.

$$\min_x \sum_{S \in S} c_S x_S \quad (5)$$

subject to:

$$\sum_{S \in \mathcal{S}} x_S = 1, \quad \forall u \in U \quad (6)$$

$$x_S \in \{0, 1\}, \quad \forall S \in S \quad (7)$$

where $S$ is the feasible subsets defined in Equation 2 and $c_S$ is the group cost defined in Equation 3.

This formulation, unlike the integer and dynamic programming formulations, not only decouples the cost computation $c_S$ from the rest of the problem, it allows representing all the constraints concisely as $S$. Additionally, by representing the solution vector as a binary vector of size $|S|$, this formulation enables the easy design and expression of neighborhoods with respect to the solution vector.

Neighborhoods We now describe the neighborhoods we studied. Due to space limitations, we have excluded their explicit representation. We consider two neighborhoods in our subsequent methods: the swap neighborhood and the move neighborhood. The swap neighborhood consists of a single pairwise swap between two groups. The move neighborhood consists of a single move of an agent from one group to another.

B. Carpool as a dynamic programming problem

We present a formulation which lends itself to a dynamic programming (DP) scheme for exhaustively searching the solution space for the optimal solution. This approach provides a naive but exact baseline for comparing our methods, as this is a highly expensive method but one that is guaranteed to return the optimal answer (in exponential time).
Let $I = \{1, 2, \ldots\}$ denote the index set numbering the rideshare groups in the solution. For convenience, let index $j = 0$ represent an unassigned agent, and let $j = 1, 2, \ldots$ be the rideshare group number. Let $M \in (I \cup \{0\})^n$ denote a (partial) assignment vector, where agent $i$ is assigned to rideshare group number $M[i]$ (which may be 0, i.e. unassigned). This vector keeps track of subproblems in our DP formulation.

Now, we define the following subproblem: $G(M, j)$ gives the best configuration for the unassigned agents indicated by $M$ and assigns the first unassigned agent (defined as $i$) to rideshare group number $j$:

$$
G(M, j) = \min \left\{ c_{(i)} + G(M + j1_i, j + 1), \quad (8) \\
c_{(i,g)} + \min_{g \in M_0 \setminus \{i\}} G(M + j1_g + j1_i, j + 1), \quad (9) \\
c_{(i,g)} + \min_{g \in \{M_0 \setminus \{i\}\}} G(M + j1_g + j1_i, j + 1) \right\} \quad (10)
$$

where $M_0$ denotes the indices of the zero entries of $M$, and we let $i := \min M_0$, the index of the first unassigned agent. We denote $1_i$ as the indicator vector, with a one in the $i$th position and zero everywhere else. The three terms within the minimization refer to placing $i$ in a carpool group of size one, two, and three, respectively, and then labeling that as the $j$th rideshare group. We use the binomial coefficient in Equation (10) as shorthand for selecting all groups of size two:

$$
g \in \left( \frac{M_0 \setminus \{i\}}{2} \right) \iff \quad (12)
$$

$$
g = \{a_1, a_2\} \in M_0 \setminus \{i\} \times M_0 \setminus \{i\} : a_1 \neq a_2 \quad (13)
$$

The subsequent subproblem assigns the $j + 1$th rideshare group and excludes the agents assigned to group number $j$. Notice that there is no explicit collection of feasible subsets, so let $c_S = \infty$ when $S$ is infeasible (incompatible time windows). The base case (Equation (11)) is invoked when all agents have been assigned, i.e. $M_0 = \emptyset$.

To solve the overall problem, we invoke $G(0, 1)$, where $0$ denotes the all zeros vector. That is, at first, all agents are unassigned, and without loss of generality we can place the first agent in rideshare group number 1.

A rough estimate gives that a problem of 10 agents requires checking $6 \times 10^7$ possibilities. A problem with 20 agents requires checking $1.6 \times 10^{21}$ possibilities. With 20 agents, if each possibility can be checked in 1 nsec, then the computation would take 50,735 years. Needless to say, we were not able to solve the problem for 20 agents. The size 10 problem, on the other hand, computes in 0.80 seconds.

### C. Carpool as an integer program

#### Variables

- $od_u =$ distance between the origin and destination of agent $u$
- $t^d_i = (t_i^d, t_i^d)$ = the time window of valid departure times for agent $i$
- Output $y_{uvw} \in \{0, 1\} = 1$ if driver $u$ is carpooling with passengers $v$ and $w$, 0 otherwise. In addition, if a driver is picking up only one passenger, $y_{uvw} = 1$ and if no passengers, $y_{uvw} = 1$.

#### IP Formulation

The key insight of our integer programming (IP) formulation, which permits a clean and concise representation, is to encode fixed-size groups of size three, and pad smaller groups as needed. Thus, a lone driver $u \in \mathcal{U}$ would be represented by a group $(u, u, u)$; observe that the distance traveled is $a_{uu} + a_{uu} + od_u = 0 + 0 + od_u = od_u$.

The IP formulation is as follows:

$$
\min \sum_{i \in \mathcal{U}} y_{uu} \cdot [a_{uu} + a_{uv} + od_u] \quad (14)
$$

subject to

$$
\sum_{v,w \in \mathcal{U}} y_{vw} + \sum_{v \in \mathcal{U}} y_{uv} + \sum_{v \in \mathcal{U}} y_{uw} \quad (15)
$$

$$
y_{uv} = 0, \forall u, v, u \neq v \quad (15)
$$

$$
y_{uvw} \rightarrow [t^u_s, t^v_e] \cap [t^v_s, t^w_e] \cap [t^w_s, t^u_e] \neq \emptyset, \forall u, v, w \quad (16)
$$

$$
(1 - y_{uvw}) \vee \left( \max_{x \in \{u,v,w\}} (t^x_e) - \min_{x \in \{u,v,w\}} (t^x_e) \right), \forall u, v, w \quad (17)
$$

The overall objective is the total distance traveled by the drivers. Equation (14) ensures that each agent be in exactly one group. Since each agent may be a driver $u_{u**}$, first passenger $y_{u**}$, or the second passenger $y_{u***}$, we need to delete double- and triple-counted duplicates.

Equation (15) addresses the fact that we can represent a two-person group with $(u, v)$ as either $y_{uv}$ or $y_{vu}$, but not both. We have chosen arbitrarily $y_{uv}$ for this case, so we need to ensure that $y_{uw}$ is never set. Finally, Equation (16) enforces that for each group of agents, there is some overlap in their time windows.

#### D. Lower bound

Let’s denote the perfect carpooling lower bound on the carpool problem by

$$
\text{obj}_{LB} = \frac{1}{3} \sum_u od_u \quad (17)
$$

which assumes that every agent is in a three-person carpool and incurs no additional pickup costs.

### IV. METHODS FOR SOLVING THE CARPOOL PROBLEM

In this section, we describe the algorithms for solving the carpool problem. We describe methods for each of the problem formulations given in Section III. For the set partitioning problem, we pose four local search methods; for dynamic programming, we use a memoized recursive
approach; for integer programming, we invoked an open-source MILP solver.

**Local search: hill climbing** Hill climbing is the simplest and most efficient of our local search methods. In each iteration, a random coin flip determines which neighborhood to sample from. Then, a swap or move is sampled accordingly. Then, if the action improves the overall objective value, then the action is accepted \((X \text{ is updated})\).

Sampling a neighbor takes \(O(n)\) and computing the cost difference takes \(O(1)\), so each iteration takes \(O(n)\) time.

**Local search: simulated annealing** A generalization of hill climbing is simulated annealing, which has a few key aspects: 1) it allows multiple actions to occur in a single iteration, 2) it allows actions which worsen the objective to be accepted, and 3) it allows tuning the number of actions and probability of accepting a worse solution, even as the algorithm progresses.

At each iteration, this method consists of sampling a number of actions (moves or swaps) according to the current temperature \(T\). All actions are performed and then evaluated relative to the current iterate \(X\). If the objective is better, it is accepted as before. If the objective is worse, the actions can be accepted with probability according to the magnitude of change and a temperature parameter. At the end of each iteration, the temperature parameter is decayed at rate \(\beta\).

The iteration complexity of this method is again \(O(n)\). However, the constant factor is larger and determined by the temperature parameters \((T, \beta)\).

**Local search: swap and move neighborhood** In the classical local search method, at each iteration, a full search of the swap and move neighborhoods is performed and the single best action is selected.

**Local search: random neighborhood** Our last local search method combines some of the deterministic aspects of the classical local search method with the stochastic nature of hill climbing. At each iteration, a coin flip chooses a random neighborhood (move or swap). Then, the best action within that neighborhood is selected.

**Exhaustive search: dynamic programming** We use a memorized recursive DP scheme to solve the subproblems given by Equations (8)-(11) in Section III-B.

**Integer program** We implemented our IP formulation in Python-based Numberjack [19], an open-source framework that interfaces with MILP and constraint programming (CP) solvers. Specifically, we used the SCIP solver [20].

V. NUMERICAL IMPLEMENTATION

For our experiments, we generated problem instances with agent sizes that are exponentially increasing (from 10 to 100K) and attempted to solve them using multiple methods, both exact and approximate.

**A. Workloads**

We experimented with problem instances drawn from two different settings.

**Euclidean setting** In the Euclidean setting, we generated agent home locations based on a Gaussian distribution over the \(\mathbb{R}^2\) space.

**San Francisco (SF) Bay Area setting** In this setting, we assigned agent home locations by sampling with replacement from a dataset consisting of 400k agent plans for the San Francisco Bay Area [21] based on the California Metropolitan Transportation Commission (MTC) travel model. We uniformly randomly selected a one-hour time window for each agent.

**Problem sizes** In the prior work, the largest instances have been 1000 and typically much smaller. However, if we want to support carpooling at a scale that can make a significant social impact, we need to work with much larger problem sizes. For example, the population of the nine-county Bay Area is more than seven million [22]. So supporting even 1-2% of the population will require supporting agent counts that are one-two orders of magnitude larger (70K - 140K). Therefore, we studied instances of \{10, 100, 1K, 10K, 100K\} agents.

**B. Initialization**

We now briefly describe our stages of computing an initial solution, which is important for the local search methods. We first greedily form agents into groups by repeatedly selecting an available agent and the closest two agents to her. We refer to this as the raw init stage because this initialization could violate the time constraints. Then to satisfy the time constraints, we arbitrarily break up groups which are not compatible in time. We refer to this as the raw init time ranges stage. Finally, we solve the small traveling salesman problem to optimize the ordering of the agents in each group and thereby compute cs; this is the raw init opt pickup stage.

**C. SF Bay Area distance pre-computation**

In the San Francisco Bay Area setting, the act of querying (i) distances from origin to destination of agents, and (ii) pairwise distances between agent origins, underlies the operation of every method. Although computing Euclidean distances takes roughly 0.5 msec, querying a routing service incurs I/O cost, thus increasing the time cost to the order of hundreds of milliseconds. Thus the non-Euclidean setting appears to immediately incur a penalty of 100x, regardless of how fast the algorithm is. To overcome this, we pre-compute the \(O(n^2)\) required distances. For 100K agents, the space requirement for pre-computation is on the order of 200G; this is easily supportable even with a general purpose m4.2xlarge AWS instance, which has 32GB of RAM.

Unfortunately, due to the \(O(n^2)\) nature of the pairwise distance computations, even the pre-computation time does not scale well beyond 10k agents. As we can see from Table I, even using the roadsindb distance oracle [23], which uses a pre-computed representation of driving distances between all locations in a particular region, scaling from problem size 1k to size 10k increases the time required from 209 secs to 3
hours. This quadratic increase implies that the pre-computing distances for 100k agents would require 300 hours = 12.5 days. Thus in the 100K agents instances, we instead queried distances on the fly as needed from roadsindb, and this was successfully used in our largest local search experiments.

<table>
<thead>
<tr>
<th>agents</th>
<th>gen-gauss</th>
<th>calc-dist</th>
<th>gen-real</th>
<th>roadsindb</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.135</td>
<td>0.009</td>
<td>80.754</td>
<td>0.045</td>
</tr>
<tr>
<td>100</td>
<td>2.130</td>
<td>0.721</td>
<td>85.204</td>
<td>2.010</td>
</tr>
<tr>
<td>1000</td>
<td>79.979</td>
<td>71.760</td>
<td>129.450</td>
<td>209.356</td>
</tr>
<tr>
<td>10K</td>
<td>70.622</td>
<td>412.742</td>
<td>169.458</td>
<td>12.3×10³</td>
</tr>
</tbody>
</table>

TABLE I
TIME SCALING WHEN DISTANCE IS CALCULATED USING QUERIES TO THE CACHED ROADSDNDB DATA. ALL TIMES ARE IN SECS.

D. Experimental Setup

Our experimental setup consists of a combination of a medium-sized AWS instance and a large shared server. The system configurations of the two servers: 1) an AWS instance with 8 CPUs and 30GB RAM, used to generate instances, pre-compute distance matrices, and execute the size 100k local search experiments, and 2) a shared lab server with 24 cores and 256GB RAM, used to run 10-10k instances using pre-computed distance matrices.

The experimental timeouts are set logarithmically; the timeouts are \{1, 2, 3, 4, 5\} hours for \{10, 100, 1000, 10k, 100k\} agents, respectively.

VI. NUMERICAL RESULTS

Scalability In line with results from Section V-C, algorithms with linear or greater running times did not scale. While the exponential exhaustive search algorithms was not expected to scale, we were surprised to find that it did not finish solving even the 20 agent problem in over a day.

When we analyzed the computation time of four local search methods (hill climbing, simulated annealing, classical local search (called best_step), and random neighborhood local search (called best_random) with increasing workloads, we found that the best_step and best_random methods did not finish even one 10k iteration before timing out after 4 hours. Figure 1 summarizes the time per iteration cost of each method for varying problem sizes.

We additionally break the computation time down into the various initialization stages (see Section V-B) and the per iteration computation. The results are summarized in Figure 2. Hill climbing and simulated annealing scale significantly better than the local search methods that perform a full neighborhood search. For these methods, the per iteration computation is significantly less than the initialization time.

This is an expected assessment because the local search methods that perform a full neighborhood search are much more thorough in their evaluation of which action to perform at each iteration (thus taking \(O(n^2)\) time to choose). The next natural question is whether it is a necessary expense.

Convergence We now turn to assessing the convergence of each of the methods to the naive lower bound. Relating the objective value to the naive lower bound conveniently allows us to assess different problem sizes in the same figure by normalizing the value. Figure 3 summarizes the convergence of the different methods (minus simulated annealing) for different problem sizes. We find that hill climbing by far converges the fastest. However, the classical local search method converges to a lower ratio in some instances. These findings suggest combining the two methods to achieve the best of both worlds: first using hill climbing to quickly converge close to a local optimum, then using classical local search to converge to an exact local optimum.

Simulated annealing is excluded from the previous figure due to its chaotic nature in convergence. To demonstrate this we have included a figure comparing the convergence of simulated annealing to hill climbing (see Figure 4). We note that our simulated annealing implementation is not highly tuned for our problem. With tuning such as specialized temperature schedules and restarts, simulated annealing has been demonstrated to perform very well in many practical combinatorial optimization settings. We do not claim that it
would not work here as well, but that we have not discovered optimal hyperparameters.

VII. CONCLUSION

In this article, we cast the HOV-3 Carpool problem into the framework of combinatorial optimization and shown that it is NP-Hard and difficult to solve iteratively. Next, we show that a relaxed version of the problem, the HOV3-Carpool problem, is amenable to many classes of solution methods for combinatorial optimization: local search, integer programming, and dynamic programming. In particular, the new relaxed formulation permits local search methods, and we experimentally show that a sampling-based local search method (hill climbing) scales up to 100K agents and converges to a ratio of within 1.1 of the lower bound in 5 hours. The other methods either do not complete in the time allotted or do not converge as well.

This work is a first step towards incorporating various, complex human considerations into carpool optimization. There are many considerations yet to be addressed for a real-world ride sharing system to be viable. For instance, our work does not consider the delay caused by waiting for participants, travel time, routing complications, or congestion.
Additionally, participants’ preferences with respect to when to travel or arrive may be non-uniform and may be correlated with spatial-temporal features. There are also many complex human factors that must be considered including fairness and social and cultural compatibility. One of the major benefits of local search methods such as we the ones we studied in this article is that they are extremely flexible with respect to different cost terms in the objective; however, further investigation is needed for an empirical confirmation.

On the other hand, local search methods are highly sensitive to the constraints of the optimization problem, and therefore new methods will be needed for new or modified constraints, such as allowing for more flexible carpool sizes or permitting participants to specify intermediate stops. However, problems with such constraints can potentially be decomposed into independent problems with simplified constraints.

In practice, we expect that some of these complex cost factors and constraints may actually contribute to cleaner participation in a ride-sharing scheme: MIP versus CP formulations, with Tools with Artificial Intelligence (ICTAI), 2015 IEEE 27th International Conference on, IEEE, 2015, pp. 836–843.

VIII. ACKNOWLEDGEMENTS

We would like to thank Alexei Pozdnukhov for giving us access to the dataset of 400K home locations in the Bay Area and for pointing us to the roadsindb project. We would also like to thank Thomas Raffill for giving us the elegant repetition idea for the integer programming formulation. We would like to thank Stephen Twigg for discussions on the combinatorics of our problem. Cathy Wu performed parts of this research during an internship at Microsoft Research.

REFERENCES