Optimizing the Diamond Lane:
A More Tractable Carpool Problem and Algorithms

Cathy Wu[1], K. Shankari[1], Ece Kamar[4], Randy Katz[1], David Culler[1],
Christos Papadimitriou[1], Eric Horvitz[4], Alexandre Bayen[1,2,3]

***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. This crossing from a tractable (HOV2) to intractable (HOV3) problem, if not addressed, would greatly impede the scalability of any real-world rideshare matchup system.

We cast the ridesharing problem with HOV lanes into a general combinatorial optimization framework [6], [7], which enables us to devise and invoke several types of solution methods and assess their accuracy and scalability. In particular, we formulate our problem as a set partitioning problem, an integer program, and as a dynamic programming recurrence. We study local search, integer programming, and dynamic programming methods to the problem.

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

*This research is supported in part by the NSF Graduate Research Fellowship, NSF CISE Expeditions Award CCF-1139158, DOE Award SN10040 DE-SC0012463, and DARPA XData Award FA8750-12-2-0331, and gifts from Amazon Web Services, Google, IBM, SAP, The Thomas and Stacey Siebel Foundation, Apple Inc., Arimo, Blue Goji, Bosch, Cisco, Cray, Cloudera, Ericsson, Facebook, Fujitsu, HP, Huawei, Intel, Microsoft, Pivotal, Samsung, Schlumberger, Splunk, State Farm and VMware.

[1] Department of Electrical Engineering and Computer Science at UC Berkeley
[2] Department of Civil and Environmental Engineering at UC Berkeley
[3] Institute of Transportation Studies at UC Berkeley
[4] Microsoft Research

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 the case of pre-assigning driver and rider roles reduces to bipartite graph matching [16], [17]. Most studies thus largely focus on small problem sizes (up to about 1000 agents) [8], which is a strong technical limitation for providing carpooling at scale as a suitable transportation alternative. One of the primary goals of our work is to devise highly scalable methods for our specific HOV carpool problem.

II. The carpool problem

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 $(x_h^u, x_w^u), \forall u \in \mathcal{U}$ as well as time windows $T^u = (t_0^u, t_1^u, \ldots), \forall u \in \mathcal{U}$ with $t_i^u = [t_s^u, t_e^u]$. 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,

$$\mathcal{S} := \left\{ S : S \in 2^{\mathcal{U}}, |S| = 3, \bigcap_{u \in S} t^u \neq \emptyset \right\}. \tag{1}$$

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 $C : 2^{\mathcal{S}} \to \mathbb{R}$.
- $\mathcal{R}$ 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 $\mathrm{O}(n^2)$. 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.

($\Rightarrow$) : An instance of Exact Cover by 3-Set can be mapped in polynomial time to an instance of the HOV3 carpool problem as follows: Let $\mathcal{U} := U$. For each subset $S_i \in F$, select a new time window t which 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 $S_i \in F \implies S_i \in \mathcal{S}$, and the uniqueness of the time window ensures that $S_i \in \mathcal{S} \implies S_i \in F$. Note that the size-3 constraint is satisfied by definition of the Exact Cover by 3-Set problem.

($\Leftarrow$) : A solution, i.e. a partition, $\mathcal{R}$ to the HOV3 carpool instance can then be mapped directly to a solution to the Exact Cover by 3-Set instance, as all sets $S \in \mathcal{R}$ are size three, are subsets of $\mathcal{U} = U$, and are by construction in $\mathcal{S} = F$.

Then, Exact Cover by 3-Set reduces to HOV3 carpool problem, and thus the latter is NP-hard. We conclude that the HOV3 carpool problem is NP-Complete. ■

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 $\mathcal{S}$:

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

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,

$$c_S = \sum_{(u,v) \in \mathrm{TSP}(S)} d_{uv} \tag{3}$$

$$C = \sum_{S \in \mathcal{R}} c_S \tag{4}$$

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.

$$\min_x \sum_{S \in \mathcal{S}} c_S x_S \tag{5}$$

$$\text{s.t.} \sum_{S : u \in S} x_S = 1, \quad \forall u \in \mathcal{U} \tag{6}$$

$$x_S \in \{0, 1\}, \quad \forall S \in \mathcal{S} \tag{7}$$

where $\mathcal{S}$ 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 $\mathcal{S}$. Additionally, by representing the solution vector as a binary vector of size $|\mathcal{S}|$, 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 a 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:

$$G(M, j) = \min \Big\{ c_{\{i\}} + G(M + j\mathbf{1}_i, j+1), \tag{8}$$

$$c_{\{i,g\}} + \min_{g \in M_0 \setminus \{i\}} G(M + j\mathbf{1}_g + j\mathbf{1}_i, j+1), \tag{9}$$

$$c_{\{i\} \cup g} + \min_{g \in \binom{M_0 \setminus \{i\}}{2}} G(M + j\mathbf{1}_g + j\mathbf{1}_i, j+1) \Big\} \tag{10}$$

$$G(M, j) = 0 \quad \forall j \quad \text{if } M_0 = \emptyset \quad \text{(base case)} \tag{11}$$

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 $\mathbf{1}_i$ as the indicator vector, with a one in the ith position and zero everywhere else. The three terms within the minimization refers to placing i in a carpool group of size one, two, and three, respectively, and then labeling that as the jth rideshare group. We use the binomial coefficient in Equation (10) as shorthand for selecting all groups of size two:

$$g \in \binom{M_0 \setminus \{i\}}{2} \iff \tag{12}$$

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

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 $G(\mathbf{0}, 1)$, where $\mathbf{0}$ 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 6×10^7 possibilities. A problem with 20 agents requires checking 1.6×10^{21} possibilities. With 20 agents, if each possibility can be checked in 1nsec, 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: We first define some additional needed notation.

Inputs
- oo_{uv} = distance between the origin of agent u and origin of agent v. Note that $oo_{uu} = 0$.
- od_u = distance between the origin and destination of agent u
- $t_i^u = (t_s^u, t_e^u)$ = 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, $y_{uvv} = 1$ and if no passengers, $y_{uuu} = 1$.

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 $oo_{uu} + oo_{uu} + od_u = 0 + 0 + od_u = od_u$. The IP formulation is as follows:

$$\min_y \sum_{i \in \mathcal{U}} y_{uvw} \cdot [oo_{uv} + oo_{vw} + od_w]$$

subject to

$$\sum_{v,w \in \mathcal{U}} y_{uvw} + \sum_{v,w \in \mathcal{U}} y_{vuw} + \sum_{v,w \in \mathcal{U}} y_{vwu} \tag{14}$$

$$- \sum_{v \in \mathcal{U}} y_{uuv} - \sum_{v \in \mathcal{U}} y_{uvu} - \sum_{v \in \mathcal{U}} y_{vuu}$$

$$+ y_{uuu} = 1, \forall u$$

$$y_{uuv} = 0, \forall u, v, u \neq v \tag{15}$$

$$y_{uvw} \implies [t_s^u, t_e^u] \cap [t_s^v, t_e^v] \cap [t_s^w t_e^w] \neq \emptyset, \forall u, v, w \tag{16}$$

$$(1 - y_{uvw}) \vee (\max_{x \in \{u,v,w\}} (t_s^x) \leq \min_{x \in \{u,v,w\}} (t_e^x)), \forall u, v, w$$

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 y_{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_{uuv} or y_{uvv}, but not both. We have chosen arbitrarily y_{uvv} for this case, so we need to ensure that y_{uuv} 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

$$\text{obj}_{LB} = \frac{1}{3} \sum_u od_u \tag{17}$$

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 is updated).

Sampling a neighbor takes $\mathrm{O}(n)$ and computing the cost difference takes $\mathrm{O}(1)$, so each iteration takes $\mathrm{O}(n)$ 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 β.

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

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 memoized 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 c_S; 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 precompute the $\mathrm{O}(n^2)$ required distances. For 100K agents, the space requirement for pre-computation is on the order of 20GB; this is easily supportable even with a general purpose `m4.2xlarge` AWS instance, which has **32GB** of RAM.

Unfortunately, due to the $\mathrm{O}(n^2)$ 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.

agents	gen-gauss	calc-dist	gen-real	roadsindb
10	0.135	0.009	80.754	0.043
100	2.130	0.721	85.204	2.010
1000	79.979	71.760	129.450	209.356
10K	70.622	412.742	169.458	12.3×10^3

TABLE I

TIME SCALING WHEN DISTANCE IS CALCULATED USING QUERIES TO THE CACHED `ROADSINDB` DATA. ALL TIMES ARE IN SECS.

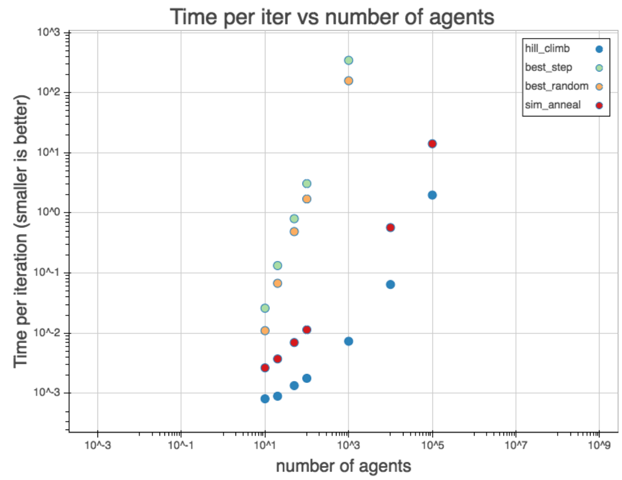

Fig. 1. Time per iteration for varying numbers of agents. Note that the scale is log-log. The `best_step` and `best_random` methods were not able to compute problem sizes 10k and 100K within the time alloted for the respective problem sizes (see Section V-A).

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.

Exact solvers We were unable to compute exact solutions for problem sizes greater than 50 for IP and greater than 10 for dynamic programming (see Table II).

size	IP obj	IP time	HC obj	HC time
10 agents	0.898	2.137	0.898	1.11
20 agents	1.578	44.360	1.577	2.77
50 agents	3.413	12225.513	3.513	8.35

TABLE II

COMPARISON BETWEEN THE OBJECTIVE AND THE RUN-TIME (SEC) FOR THE IP AND HILL CLIMBING (HC) METHODS. THIS DATA IS FROM A SINGLE RANDOM INSTANCE FROM THE EUCLIDEAN SETTING.

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

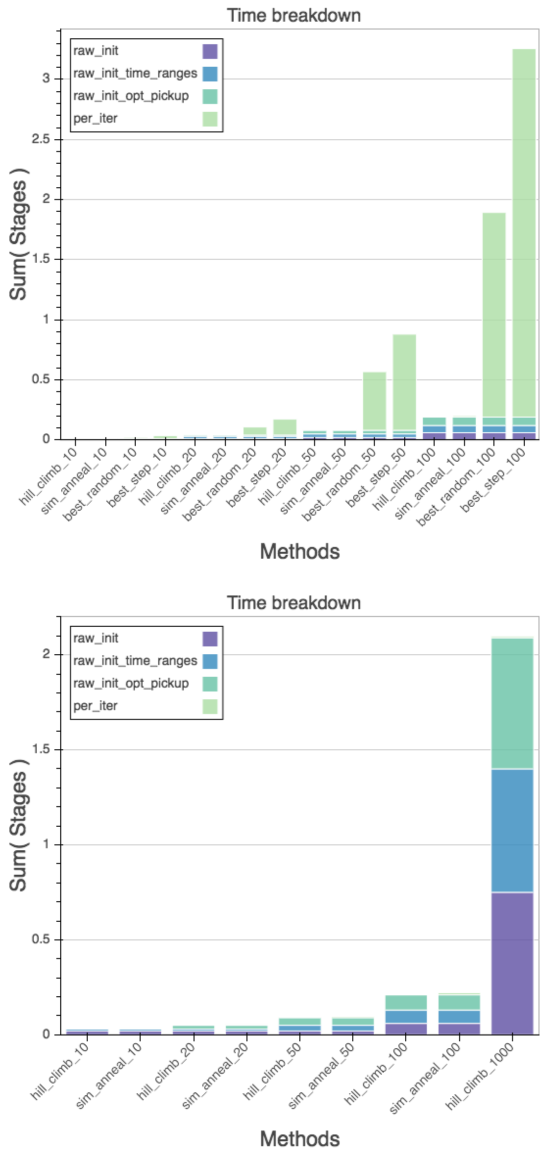

Fig. 2. This figure demonstrates the scalability of the methods. **Top:** With 20 agents or more, the per iteration computation dominates the local search methods which perform a full neighborhood search. Beyond 100 agents, the relative timing becomes indistinguishable because the per iteration time dominates, so we have excluded the larger values. **Bottom:** We focus on the sampling-based local search methods, which scale much better in per iteration computation time (the topmost stacked bar, barely visible). At 1000 agents, the per iteration computation is only a small fraction of the total initialization time, but for larger sizes, the initialization far dominates the per iteration computation because the initialization time is $O(n^2)$, whereas the per iteration computation is $O(n)$.

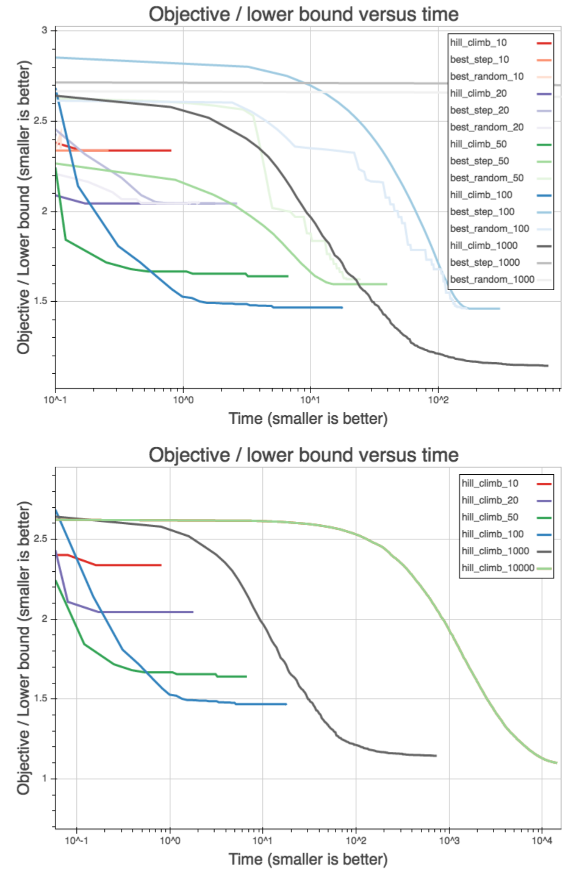

Fig. 3. This figure demonstrates the convergence of the methods. **Top:** For the smaller problem instances: $\{10, 20\}$, the three methods displayed converge to the same ratio to the lower bound, but hill climbing converges much faster and, for the larger problem instances of $\{50, 100, 1000\}$, to a lower ratio. The classical local search method gives a steady but slow convergence, whereas the random neighborhood search gives a more jagged convergence. In the case of size 50 and 100, the classical local search method converges to a slightly lower ratio than hill climbing, implying that hill climbing may have reached a different local optimum (or not reached one at all). In the case of problem size 1000, the `best_step` and `best_random` (the gray near-horizontal lines) are further from the lower bound and were cut short due to the timeout. **Bottom:** We display convergence only for the hill climbing method and problem instances $\leq$ 100K. In all instances the problem converges to between 1.1 to 2.4 times the lower bound. Notably, in the largest problem size, hill climbing achieves a ratio close to 1, implying that the solution is near optimal. Note that the x-scale is on a log scale.

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.

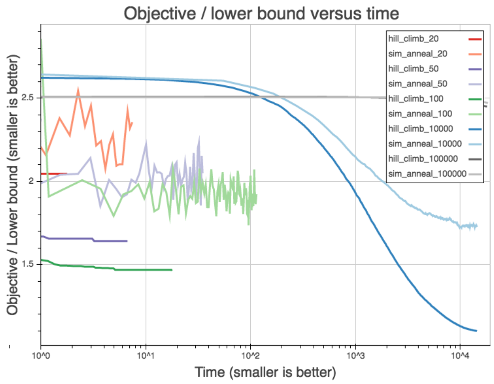

Fig. 4. Convergence of the simulated annealing method (lighter shades) is much more chaotic than hill climbing (darker shades), and simulated annealing does not converge to as low a ratio to the lower bound as the hill climbing method. Note that the x-scale is on a log scale.

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 or faster methods, introduce new technical problems, and improve the feasibility of carpooling in the real-world. We especially encourage the research community to embrace the complexity and take advantage of the structure within these complex human factors.

There are several extensions to pursue next. We would like to understand the algorithmic complexity of the HOV3-problem, and we are interested in convergence guarantees for the hill climbing method, perhaps by casting the method into a MCMC framework and studying Markov chain summaries. We are also interested in studying more complex agent constraints and costs; for instance, agents may have limited tolerance for deviating from a baseline travel plan. We are interested in conducting more extensive experiments after improving implementations, combining methods, or trying alternative warm-starting schemes.

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

[1] L. Blincoe, T. R. Miller, E. Zaloshnja, and B. A. Lawrence, "The economic and societal impact of motor vehicle crashes, 2010 (revised)," Tech. Rep., 2015.

[2] J. Grant, W. Schroeer, B. Petersen, and M. O'Neill, "Our built and natural environments: A technical review of the interactions between land use, transportation, and environmental quality," EPA 231-R-00-005): US Environmental Protection Agency, Tech. Rep., 2000.

[3] D. Schrank, B. Eisele, T. Lomax, and J. Bak, "Urban mobility scorecard," Technical Report August, Texas A&M Transportation Institute and INRIX, Inc, Tech. Rep., 2015.

[4] N. D. Chan and S. A. Shaheen, "Ridesharing in north america: Past, present, and future," *Transport Reviews*, vol. 32, no. 1, pp. 93–112, 2012.

[5] M. Furuhata, M. Dessouky, F. Ordóñez, M.-E. Brunet, X. Wang, and S. Koenig, "Ridesharing: The state-of-the-art and future directions," *Transportation Research Part B: Methodological*, vol. 57, pp. 28–46, 2013.

[6] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[7] L. A. Wolsey and G. L. Nemhauser, *Integer and combinatorial optimization*. John Wiley & Sons, 2014.

[8] V. Armant, N. Mahbub, and K. N. Brown, "Maximising the number of participants in a ride-sharing scheme: MIP versus CP formulations," in *Tools with Artificial Intelligence (ICTAI), 2015 IEEE 27th International Conference on*. IEEE, 2015, pp. 836–843.

[9] W. M. Herbawi and M. Weber, "A genetic and insertion heuristic algorithm for solving the dynamic ridematching problem with time windows," in *Proceedings of the 14th annual conference on Genetic and evolutionary computation*. ACM, 2012, pp. 385–392.

[10] N. Agatz, A. Erera, M. Savelsbergh, and X. Wang, "Optimization for dynamic ride-sharing: A review," *European Journal of Operational Research*, vol. 223, no. 2, pp. 295–303, 2012.

[11] H. Yang and H.-J. Huang, "Carpooling and congestion pricing in a multilane highway with high-occupancy-vehicle lanes," *Transportation Research Part A: Policy and Practice*, vol. 33, no. 2, pp. 139–155, 1999.

[12] H. Konishi and S.-i. Mun, "Carpooling and congestion pricing: Hov and hot lanes," *Regional Science and Urban Economics*, vol. 40, no. 4, pp. 173–186, 2010.

[13] E. Kamar and E. Horvitz, "Collaboration and shared plans in the open world: Studies of ridesharing." in *IJCAI*, vol. 9, 2009, p. 187.

[14] W. Vickrey, "Counterspeculation, auctions, and competitive sealed tenders," *The Journal of finance*, vol. 16, no. 1, pp. 8–37, 1961.

[15] V. Armant and K. N. Brown, "Minimizing the Driving Distance in Ride Sharing Systems." IEEE, Nov. 2014, pp. 568–575.

[16] G. Simonin and B. O'Sullivan, "Optimisation for the ride-sharing problem: a complexity-based approach." in *ECAI*, 2014, pp. 831–836.

[17] L. Knapen, I. B.-A. Hartman, D. Keren, S. Cho, T. Bellemans, D. Janssens, G. Wets *et al.*, "Scalability issues in optimal assignment for carpooling," *Journal of Computer and System Sciences*, vol. 81, no. 3, pp. 568–584, 2015.

[18] S. Dasgupta, C. H. Papadimitriou, and U. Vazirani, *Algorithms*. McGraw-Hill, Inc., 2006.

[19] E. Hebrard, E. O'Mahony, and B. O'Sullivan, "Constraint programming and combinatorial optimisation in numberjack," in *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. Springer, 2010, pp. 181–185.

[20] T. Achterberg, "Scip: Solving constraint integer programs," *Mathematical Programming Computation*, vol. 1, no. 1, pp. 1–41, 2009, http://mpc.zib.de/index.php/MPC/article/view/4.

[21] A. Pozdnoukhov, A. Campbell, S. Feygin, M. Yin, and S. Mohanty, "The smartbay project: Connected mobility in san francisco bay area," in *The Multi-Agent Transport Simulation MATSim*, N. K. Horni A. and K. Axhausen, Eds. Ubiquity, London, 2016, ch. 83, pp. 541 – 546.

[22] Cynthia Kroll, Shija Lu, Aksel Olsen, and Hing Wong, "Regional Forecast for Plan Bay Area 2040," Association of Bay Area Governments, Tech. Rep., Feb. 2016.

[23] H. Samet and J. Sankaranarayanan, "Path oracles for spatial networks," Jun. 3 2014, US Patent 8,744,770.