OBHIVE - A BEE-INSPIRED VEHICLE ROUTING ALGORITHM FOR REAL-TIME PASSENGER PICKUP & DROP-OFF

Taylor G. Smith
TC Labs
Toyota Connected
taylor.smith@toyotaconnected.com

September 26, 2019

ABSTRACT

This paper introduces OBHIVE, a novel variant of the classical Bees algorithm tailored specifically for the capacitated pickup and delivery problem with time windows and multiple trips (CPDPTWMT). OBHIVE is adapted from the Enhanced Bees algorithm [1], itself a VRP-adapted variation of the Bees algorithm [2], and is amended to address quickly optimizing vehicle assignments and itineraries in the context of an on-demand, real-time ridesharing application.

***K*eywords** Bees Algorithm · Swarm Intelligence · Vehicle Routing · Ridesharing · CPDPTWMT

1 Introduction

The vehicle routing problem (VRP) is a specific application of the traveling salesman problem concerned with routing one or more vehicles from a set of "sources," or depots, through a collection of "sinks," or destinations. It was first introduced by Dantzig and Ramser in their paper, "The Truck Dispatching Problem" [3], and has since become an area of intense research. In a commercial setting, VRP is a common problem faced by logistics and shipping companies who are looking to efficiently route freight to warehouses or stores. Effective routing systems have been shown to save companies hundreds of millions of dollars annually [4].

In the vehicle routing problem, vehicles may begin at one or more depots and may visit zero or more of the destination nodes, as long as all of the freight makes it to the destinations, and all other arbitrary constraints are satisfied. The pickup and delivery problem (PDP) is a further subset of the VRP wherein destinations may have complex interdependencies, each of whose visitation may be ordinally dependent on others'. In other words, a subset of destinations may serve as depots for some other arbitrary subset of destinations, and a vehicle must find an optimal tour through the nodes such that each sink is fed by its respective source. The growth of ridesharing and on-demand delivery services such as Uber and Lyft has given further rise to an interest in optimization approaches for driver scheduling, route optimization, and many more PDP applications. Even more specific families of PDP introduce yet more complex constraints: capacity-limited vehicles, pickup or delivery with time windows, and potentially multiple trips. In the setting of ridesharing, all of these constraints must be satisfied, and we ultimately become faced with the very difficult-to-solve capacitated pickup and delivery problem with time windows and multiple trips (CPDPTWMT).

The core challenge with this family of problem is the rapid pace at which it scales to a point of computational infeasibility. Even in its simplest form, the VRP problem is $\mathcal{NP}$-hard, scales combinatorially, and an exact solution cannot be solved-for within any reasonable amount of time. Add to that the fact that ridesharing applications permit users to make on-demand requests in real-time and that they expect a response within a reasonable amount of seconds, and the feasible region of the problem continues to shrink.

Despite the growth in such ridesharing applications, the literature around on-demand PDP with flexible time windows is very scant. In this paper we present our solution to the problem, OBHIVE (**O**n-demand **B**ee **H**ive-**I**nspired **VE**hicle routing), an improved bee-inspired algorithm to solve the CPDPTWMT in the context of a real-time ridesharing application.

1.1 Content Outline

Section 2 gives an overview of contemporary meta-heuristic optimization techniques and bee-inspired algorithms. Section 3 contains a detailed walkthrough of the OBHIVE algorithm and how it builds on the enhanced bees optimizer, adapting it to address CPDPTWMT. Section 4 reports the OBHIVE performance and results. Section 5 is our conclusion.

2 Meta-heuristics and Bee Algorithms

Meta-heuristic optimization techniques are high-level, iterative frameworks by which candidate solutions are initially seeded (often randomly) and then procedurally improved-upon and evaluated for feasibility and "fitness." Unlike their exact solution counterparts, which hinge around exploring the feasible regions within hard constraint boundaries (i.e., the Simplex algorithm [5, 6, 7]), meta-heuristics are designed to eventually arrive at a solution that approximates the known best solution within a small percentage and a much shorter timespan. Moreover, they are easily adapted to handle any number of arbitrary constraints, and are well-equipped to handle large-scale operational problems, making them an ideal framework for our CPDPTWMT problem. One such notable meta-heuristic method, the bees algorithm [2], mimics the foraging pattern of honey bees in order to quickly arrive at an initial solution and then incrementally search increasingly more optimal subspaces. The algorithm behaves similar to the "waggle dance" that bees perform for their peers after returning from a food source they've found. The more frantic the dance, the more the bee communicates to its peers that the food source (i.e., "solution") is highly fit for exploration, and should be searched by more workers.

Pham, et al., focused their bees algorithm on function optimization, applying their method to the classic Shekel's Foxholes problem and showing that it outperformed many such other meta-heuristics (ant colony optimization [8], genetic algorithms [9, 10]). Fenton built upon the bees algorithm, introducing the enhanced bees algorithm [1], which adjusts the strategy by which its large neighborhood search produces new solution candidates, and effectively adapted it for use in a commercial VRP setting. Fenton's results were extremely promising, demonstrating the ability to approximate a known-best solution on a collection of public domain VRP benchmarks within 5% under 60 seconds. Given the enhanced bees algorithm's highly similar set of objectives (laid out in detail in Section 3), and demonstrably high performance within a short window of time, it was selected as a baseline for our algorithm.

3 OBHIVE

The OBHIVE algorithm was developed to control assignments of real-time ride-share requests in an on-demand setting. It is based on the enhanced bees algorithm [1] and, as such, prioritizes the same set of requirements Fenton laid out in his 2016 paper (slightly modified to align with our demands):

1. Ensure that all constraints are met. Specifically that no vehicles exceed their capacity.
2. Have a good runtime performance. We more highly prioritize the discovery of any feasible solution in a very short amount of time (5 seconds or fewer) than the most optimal solution.
3. Produce good quality results. All other objectives considered, the solution must be close to the global optimum.
4. Designed such that it can be embarrassingly parallelizable.

Similar to the enhanced bees algorithm, OBHIVE is designed for a commercial setting where computation time is a crucial factor. However, its real-time nature and the addition of flexible time window constraints make the problem different enough as to warrant several more performance requirements:

5. Accommodate pickup time for users who scheduled their pickup ahead of time, as well as satisfying ride-share requests of passengers already onboard and of those submitting requests in an on-demand fashion.
6. Accomodate specifed delivery time of customers onboard within some slack amount while permitting the vehicle the flexibility to deviate course and add on-demand passengers in real time.
7. Penalize routes which would cause a customer to remain in the vehicle for a significant amount of time longer than the estimate of their direct route.

Specifically, it is more important for the ridesharing nature of the problem that vehicle capacity never be violated, but that pick-up/drop-off times be honored as much as possible. Therefore, OBHIVE handles all constraints within some band of slack, imposing vehicle capacity as the only hard constraint. Additionally, OBHIVE is used in a near-real-time ridesharing application, so timeliness is paramount. It is generally more desirable that the algorithm return a semi-optimal, fully feasible result in a few seconds than it is to wait a long period of time for a more optimal solution. Furthermore, since passengers may already be onboard a moving vehicle when a new ride-share request is submitted, identifying a feasible route insertion point as quickly as possible while minimizing the interruption of existing customers' rides is highly desirable.

3.1 Formulation

The OBHIVE algorithm's core procedure is largely based on the enhanced bees algorithm, with most of the changes necessary to adapt it to the PDP domain and permit it to operate in a real-time manner occurring in the seed, search and evaluation phases. Its core program is specified as follows:

Algorithm 1: OBHIVE

```
S = seedSites(existingSolution)
b = sortByFeasibilityAndFitness(S)
while termination condition not met do
    for s_i ∈ S do
        S = explore(s_i, ϵ)
        b = sortByFeasibilityAndFitness(S)
        if removal criteria met then
            removeKWorstSites
        end
    end
end
```

Like the enhanced bees algorithm, OBHIVE maintains a collection of sites, S, each of which contains a number of bees, $\mathfrak{S}_i$, which represent candidate solutions. Initially, the sites are seeded with the existing solution (that is, the vehicles' current locations and any ride-share requests that are currently en route), and any new ride-share requests are randomly inserted into routes while disregarding feasibility. At each step, S is explored via a large neighborhood search (LNS; see Section 3.3) and each site's bees are sorted on the basis of their feasibility and fitness (Section 3.4). The site with the globally "most fit" bee becomes the new best site. OBHIVE is iterative, meaning that a maximum number of iterations, I, is specified as a tunable parameter and program execution will halt after I iterations have been performed. Additionally, OBHIVE constrains runtime and will exit after a maximum runtime is reached, even if the number of iterations performed is fewer than I. The algorithm returns the best bee, $\mathfrak{S}_{\texttt{BEST}}$, at the end of its execution.

As in the enhanced bees algorithm, OBHIVE also removes a percentage, k, of the worst sites every λ iterations. This allows the algorithm to focus its time on the sites showing the most promise, similar to simulated annealing's cooling schedule [11]. Since one use case of the OBHIVE algorithm addresses scheduled rides in batch, the algorithm is permitted to run for a significantly longer period of time, and reducing the search terrain periodically is helpful in minimizing runtime. Sites are reduced under the following conditions:

$$S = S - s_w \qquad \text{if } i \bmod \lambda \equiv 0$$

Where s_w represents the worst sites that should be dropped from the search procedure. While Fenton recommended dropping the single worst site during the cooling stage, OBHIVE parameterizes this step, removing the worst $k\%$. For a more thorough explanation, see Fenton, 2016 [1].

3.2 Seeding Sites

In the enhanced bees algorithm, sites are seeded in a pseudo-random fashion, assigning random stops to vehicles to initially populate them, then adding any unassigned stops incrementally by selecting a random vehicle and finding its nearest insertion point. The OBHIVE algorithm's added constraints dictate a slightly more involved seeding process, and moreover, the PDP nature of the problem means that waypoints cannot be arbitrarily added irrespective of one another; each ride-share request, r_i, is implicitly a pair:

$$r_i = [p_i, d_i]$$

Each pair of points is subject to the following constraints (not imposed or penalized by the algorithm itself, but guaranteed as an implementation detail):

$$\begin{aligned} &p_i \in v_k, \\ &d_i \in v_k, \\ &p_i \ \ before \ \ d_i \end{aligned}$$

Where V is a matrix of eligible vehicles in the fleet, and v_k is a vector representing the kth vehicle, meaning both p_i and d_i belong to the same vehicle, and p_i occurs prior to d_i, but not necessarily continguously. To further complicate matters, each vehicle, $v_k \in V$, may already contain an itinerary of confirmed riders or on-board riders when a new ride-share request, r_n, enters the system, so each vehicle must be initialized while honoring its existing itinerary, if any, prior to randomly assigning new requests. The seeding step is roughly defined in the following procedure:

Algorithm 2: seedSites

```
V = vehicles eligible for assignment
r = a collection of ride-share requests
for v_k ∈ V do
    if v_k has assignments then
        Initialize vehicle with its existing itinerary
    else
        Initialize v_k as empty vehicle
        if any r_i ∈ r unassigned then
            insertRandomUnassigned(r, v_k)
        end
    end
end
for r_i ∈ r do
    if r_i unassigned then
        v_r = selectRandomVehicle()
        insertIntoNearest(r_i, v_r)
    end
end
```

It's important to note that even though a vehicle may already have an existing itinerary or onboard passengers, we permit the insertion of new ride-share requests into any point in the vehicle's route, allowing the possibility that a candidate route would cause a vehicle to deviate for a new pickup. Whether such a deviation is feasible or desirable is left up to the fitness function to evaluate.

3.3 Large Neighborhood Search

The majority of the OBHIVE algorithm takes place after seeding the sites. At each step, each s_i is explored via LNS procedure similar to that proposed by Shaw [12] and leveraged in the enhanced bees algorithm. To adapt the LNS phase to fit the ridesharing use case, the following notable changes were introduced:

1. During the **destroy** stage, prohibit onboard passengers from being removed from a vehicle's route
2. During the **reconstruct** stage, leverage a pre-computed nearest neighbors k-d tree to insert removed $[p_i, d_i]$ pairs into a separate route containing the nearest neighboring pair [1]. p_i is inserted randomly before or after its neighboring pickup, and r_i is inserted randomly into a slot later in the route.

As a site matures (i.e., survives the site pruning operation), the search space ϵ grows, and the algorithm explores its subspace more widely. In a discrete function optimization process like our case, ϵ does not represent a true search space radius, but rather a coefficient used to compute the number of `2-opt` permutations a site will evaluate, e.g.:

$$randInt(0, max(1, \epsilon * n))$$

Where n represents the number of ride-share requests, r. Note that even though we do not remove them, ride-share requests corresponding to assigned or onboard passengers are included in r, meaning the number of passengers onboard will still impact the number of searches a site will entertain.

The OBHIVE LNS procedure is as follows:

Algorithm 3: Large Neighborhood Search

$\mathfrak{S}$ = current solution
$numSearches = randInt(0, max(1, \epsilon * n))$
for $i \in numSearches$ **do**
  $\mathfrak{S}' = bee$
  $destroy(\mathfrak{S}')$; `// remove a random [p, d] pair from a vehicle`
  $repair(\mathfrak{S}')$; `// insert into a separate route via precomputed k-d tree`
  if $fitness(\mathfrak{S}') < fitness(\mathfrak{S})$ **then**
    $\mathfrak{S} = \mathfrak{S}'$
  end
end
Result: $\mathfrak{S}$

Following the LNS phase, the new collection of sites, S', is sorted according to each site's bees' feasibilities and fitnesses, and the top site's most fit bee, $\mathfrak{S}_{\texttt{BEST}}$, is stored as the new best solution (if better than the current best).

3.4 Fitness Function

The advent of the enhanced bees algorithm is its consideration of any candidate solution, irrespective of its feasibility. This permits the LNS heuristic to iteratively search the subspace around a highly fit infeasible candidate, potentially finding a feasible relative with a similar or better fitness. Fenton proposed the following fitness function to measure the efficacy of a proposed solution, $\mathfrak{S}_i$:

[1] While OBHIVE uses time-to-travel estimates to measure distance between waypoints, the nearest neighbors operation uses Manhattan distance.

$$c(R) = \sum_{i \in R} c_{i,i+1} \tag{1}$$

$$d(R) = max\left(\sum_{i \in R} d_i - q, 0\right) \tag{2}$$

$$t(R) = max\left(\sum_{i \in R} t_i + c(R) - t, 0\right) \tag{3}$$

$$f(\mathfrak{S}) = \sum_{R \in \mathfrak{S}} (\alpha c(R) + \beta d(R) + \gamma c(R)) \tag{4}$$

Where $c(R)$ computes the cost or distance of a route, R [2]. The function $d(R)$ measures the sum of capacity violations for a route. In our application, each stop is associated with an amount of "demand," or the number of passengers boarding or disembarking. $t(R)$ calculates the time overage, or amount of distance or time driven beyond the permissible amount. This is useful in commercial settings where there may be an additional cost of a driver's overtime.

$f(\mathfrak{S})$ is the weighted sum of a solution's ($\mathfrak{S}$) respective fitness, where a higher value corresponds to a higher "cost" of a route, and α, β, and γ are all tunable hyper-parameters. Where a fitness function composing of these three terms is perfectly reasonable in a freight scenario, some problems arise if it's applied directly to a passenger pickup/delivery scenario:

1. There is no assurance that a passenger's total trip time will be minimized. Without such a term, the algorithm will find the path that minimizes the existing fitness function and satisfies all constraints such that the passenger will be picked up, but it may not prioritize his/her drop off time until well beyond the duration of what would be considered a reasonable trip length.
2. The existing fitness function does not enforce that a customer be picked up at the time requested. This is a key part of our application, which supports scheduled pickups in addition to on-demand rides.

To fill these gaps and adapt the algorithm to a ridesharing context, OBHIVE adds two new terms to its fitness function:

$$p(P) = max(\sum_{i \in P} abs(p_i - r_i), 0) \tag{5}$$

$$s(R) = \sum_{c \in R} \begin{cases} \phi^{c_a/c_e}, & \text{if } c_a/c_e > 1 \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

Where $p(P)$ calculates the "promptness" penalty for $P \in R$, the subset of all stops in the route that are pickups. In $p(P)$, a vehicle is penalized for arriving too early or too late to a pickup point outside a specified amount of "slack" by computing the absolute difference between the true pickup time, p_i, and the requested time, r_i. The function $s(R)$ is an exponential term that heavily penalizes routes that take a passenger on a much longer trip than the direct route would take. The terms c_a and c_e correspond to the estimated total duration of the passenger's actual trip, and the estimated duration of the direct trip requested, respectively. This term prevents a vehicle from making a significant number of detours to pick up new passengers in favor of dropping the onboard passengers off as soon as possible. These additional terms make the overall OBHIVE objective function:

[2] The enhanced bees algorithm uses the Manhattan distance in the application for which it was designed, but OBHIVE was built around delivery time optimization and uses a matrix of travel time estimates instead.

Minimize:

$$f(\mathfrak{S}) = \sum_{R \in \mathfrak{S}} (\alpha c(R) + \beta d(R) + \gamma c(R) + \tau p(P) + s(R)) \tag{7}$$

In our fitness function, α, β, γ, τ and ϕ are all tunable parameters that implicitly control the priorities of vehicles for a candidate solution. While meta-heuristic-driven solutions typically shy away from imposing hard constraints where possible, the fitness function may penalize a technically impossible solution so heavily that it is artificially deemed infeasible. While there is no one configuration that will fit every use case, we first honed in on a set of base values by using benchmark data and then we found live experimentation and in-car user testing to be a suitable method for tuning these hyper-parameters.

4 Results

To evaluate the efficacy of our solution, the OBHIVE algorithm was benchmarked against a collection of classic public domain datasets, including the collection of VRP datasets from Augerat, et al [13] [3]. Figure 1 shows the algorithm's performance over a collection of VRP datasets, marking their fitness scores against the known best for each dataset.

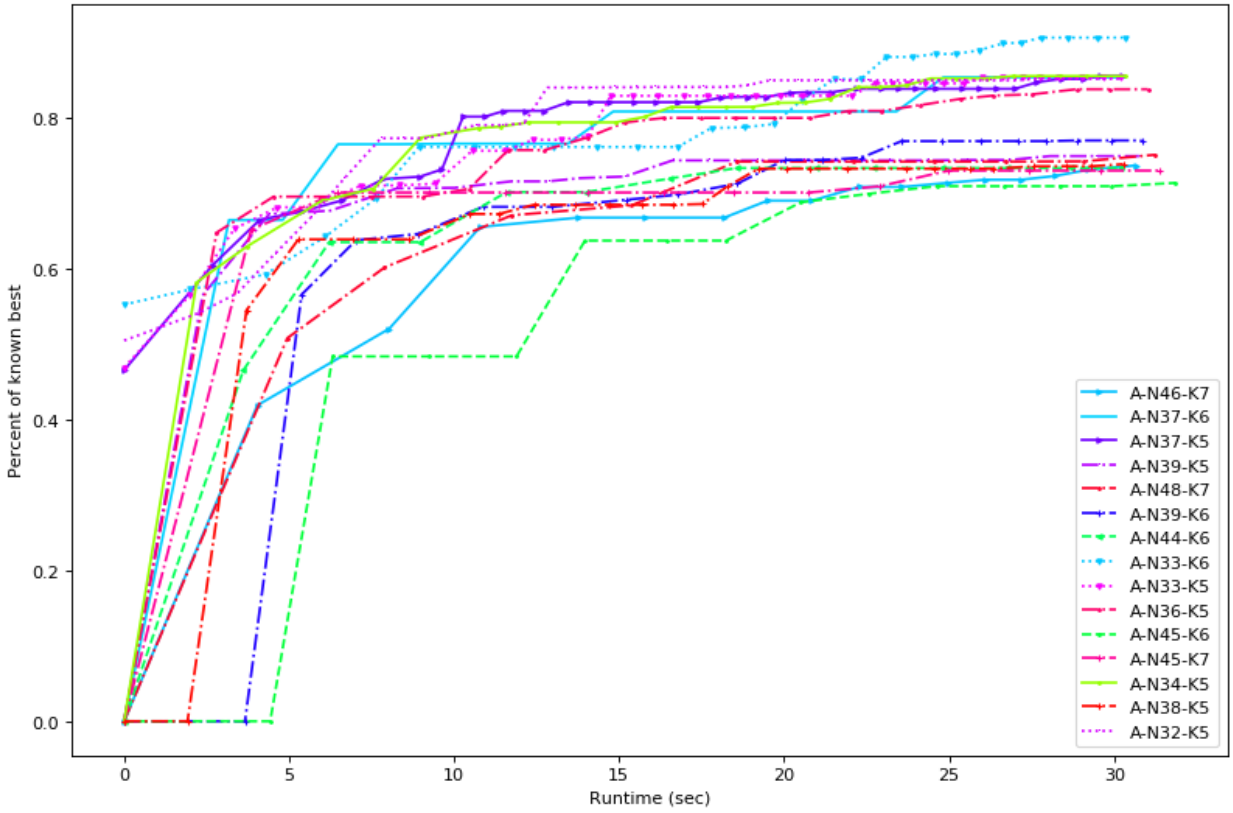

Figure 1: *Benchmarking OBHIVE against various Augerat, et al, datasets. Infeasible solutions are denoted by a fitness of zero.*

In evaluating our algorithm, we chose a configuration that prioritized speed and feasibility over fitness, similar as in our true usecase. Given the near-real-time nature of our ride-sharing application, our aim was to discover a feasible solution as quickly as possible. The algorithm was permitted to run for 30 seconds with the following configuration:

[3] OBHIVE does not directly address VRP problems, but to our knowledge there are no publicly available CPDPTWMT datasets.

$$nWorkers = 3$$
$$nSites = 10$$
$$period = 5$$
$$kWorst = 1$$

While we certainly never intend to run for 30 seconds in a live scenario, we found it valuable to demonstrate the convergence with a known best score over time. The effect of our slim configuration was that the algorithm had a narrow solution subspace to search and that it was able to explore it relatively rapidly, converging to better than 70% of the known best solution within 10 seconds for most datasets.

Furthermore, a saving grace of the architecture of our application is the ability to horizontally scale our resources, and to geofence requests from vehicles. As a result, the amount of requests and vehicles our algorithm may have to process for any given request is dramatically smaller than the size of the datasets against which we benchmarked [4]. However, the algorithm still discovered a feasible solution nearly within the first 5 seconds for each dataset.

A final advantage of the design of the algorithm is that we're highly likely to already have passengers onboard when new ride-share requests come in, which makes the feasible solution space very fast to search.

5 Conclusion

In this paper, we have described the OBHIVE algorithm, a CPDPTWMT variant of the Enhanced Bees algorithm [1] for on-demand pick-up/drop-off requests that is shown to find feasible configurations for passenger assignment in a very short span of time. The results show the method to be competitive in discovering near known-best solutions for the VRP problem relatively quickly. Future work will explore additional destroy/reconstruct heuristics that may enhance the LNS stage, and the algorithm's ability to more efficiently traverse a complex search space.

References

[1] Aish Fenton. The bees algorithm for the vehicle routing problem. *arXiv preprint arXiv:1605.05448*, 2016.

[2] DT Pham, A Ghanbarzadeh, E Koc, S Otri, S Rahim, and M Zaidi. The bees algorithm. *Technical Note, Manufacturing Engineering Centre, Cardiff University, UK*, 2005.

[3] George B Dantzig and John H Ramser. The truck dispatching problem. *Management science*, 6(1):80–91, 1959.

[4] Chuck Holland, Jack Levis, Ranganath Nuggehalli, Bob Santilli, and Jeff Winters. Ups optimizes delivery routes. *Interfaces*, 47(1):8–23, 2017.

[5] George B Dantzig. Origins of the simplex method. Technical report, STANFORD UNIV CA SYSTEMS OPTIMIZATION LAB, 1987.

[6] Maria Gabriela S Furtado, Pedro Munari, and Reinaldo Morabito. Pickup and delivery problem with time windows: a new compact two-index formulation. *Operations Research Letters*, 45(4):334–341, 2017.

[7] SP Anbuudayasankar and K Mohandas. Mixed-integer linear programming for vehicle routing problem with simultaneous delivery and pick-up with maximum route-length. *International Journal of Applied Management and Technology*, 6(1):2, 2008.

[8] Alberto Colorni, Marco Dorigo, Vittorio Maniezzo, et al. Distributed optimization by ant colonies. In *Proceedings of the first European conference on artificial life*, volume 142, pages 134–142. Cambridge, MA, 1992.

[4] For instance, we would not include ride-share requests from Tokyo with a vehicle pool from Kyoto, and vice versa, and any given on-demand execution will very rarely contain any more than a few unassigned customers.

[9] Alex S Fraser. Simulation of genetic systems by automatic digital computers i. introduction. *Australian Journal of Biological Sciences*, 10(4):484–491, 1957.

[10] Hans J Bremermann. *The evolution of intelligence: The nervous system as a model of its environment*. University of Washington, Department of Mathematics, 1958.

[11] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[12] Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *International conference on principles and practice of constraint programming*, pages 417–431. Springer, 1998.

[13] Ph Augerat, Jose Manuel Belenguer, Enrique Benavent, A Corberán, D Naddef, and G Rinaldi. Computational results with a branch-and-cut code for the capacitated vehicle routing problem. 1998.