MOBILITY CASE STUDY / ROUTING AND SIMULATION

Size the Fleet for the Queue, Not the Peak

Genetic-algorithm routing and discrete-event simulation for an on-demand mobility service

EXECUTIVE CONCLUSION

Sizing for the peak buys reliability at a price the business model cannot carry; sizing for the average fails exactly when riders need the service. Only a simulation of routing, dispatch and variable demand together finds the fleet in between.

Fleet size was the largest capital decision in an on-demand and autonomous mobility business. This note describes the two pieces of work that made it a calculation: genetic algorithms that solved the vehicle routing problem for dispatch and scheduling, and a discrete-event simulator that pushed realistic demand through that full dispatch and routing logic so fleet size, service targets and dispatch policy could be tested together.

27%
smaller fleet at the same service

300K+
trips scheduled by the routing algorithms

Genetic
algorithms solving the routing problem

Discrete-event
simulator testing fleet and policy together

Work led by Adam DeJans Jr. in an earlier industry role at a global automaker's mobility services division, not a Bit Bros client engagement. The 27% and 300K+ are project-reported and not independently audited. Exhibits 1 to 3 are simplified synthetic models built for this note.

The planning problem

Riders request trips from an origin to a destination, expecting a vehicle within a few minutes. Each request must be accepted or declined, assigned to a vehicle and inserted into that vehicle's route in near real time, without making the riders already on board late. Demand varies sharply by hour, day and zone, and the fleet is fixed in the short run.

Two decisions sit on different clocks. Dispatch and routing happen in seconds, trip by trip. Fleet size is decided months ahead and determines capital cost, utilization and, ultimately, whether the service model makes money. The second decision cannot be made well without modeling the first, because the fleet a service needs depends on how cleverly it is dispatched.

THE DECISION

What is the smallest fleet N such that the service standard is met with stated confidence, given realistic demand and the dispatch and routing policy that will actually run?

The standard must be defined before anything is sized. Here it is the share of requests, during a defined peak window, whose pickup occurs within a target wait w of the request, with declined requests counted as failures. A daily average is the wrong metric: it is dominated by off-peak hours in which almost any fleet performs well.

1. Routing every trip with genetic algorithms

The static subproblem is a pickup-and-delivery problem with time windows

At each dispatch epoch, the open requests and the vehicles' committed stops define a static pickup-and-delivery problem with time windows (PDPTW), of the dial-a-ride family. Let $\mathcal{R}$ be the open requests, $P = \{r^+\}$ the pickup nodes and $D = \{r^-\}$ the drop-off nodes, with pairing map $\delta(r^+) = r^-$. Each node i has a window $[e_i, l_i]$, a dwell time s_i and a load change q_i ($+1$ at a pickup, -1 at a drop-off); t_{ij} is travel time, C vehicle capacity and L_r the maximum ride time of request r. With $x_{ijk} = 1$ if vehicle k travels from i to j and B_i the service start at i, the core model is

$$\min \sum_k \sum_{i,j} t_{ij}\, x_{ijk} + \alpha \sum_r \left(B_{r^+} - e_{r^+}\right)$$

subject to each request served once by a single vehicle, flow conservation, and

$$\begin{aligned} B_{r^-} &\geq B_{r^+} + s_{r^+} + t_{r^+ r^-} && \text{(precedence)} \\ e_i &\leq B_i \leq l_i && \text{(time windows)} \\ B_{r^-} - \left(B_{r^+} + s_{r^+}\right) &\leq L_r && \text{(ride time)} \\ 0 &\leq Q_i \leq C && \text{(capacity)} \end{aligned}$$

Exact branch-and-cut solves static instances of moderate size, but not reliably within a dispatch cycle of seconds, which is why metaheuristics, and genetic algorithms among them, dominate operational dispatch.

Representation decides what the operators can break

A genetic algorithm searches over *genotypes* that a decoder turns into route plans. Three encodings are standard, and they differ in what they keep feasible by construction.

Encoding	Strength and weakness for PDPTW
Giant tour + Split	One permutation of stops, cut into routes by an exact DP (Prins, 2004). Elegant for capacitated routing; a cut can separate r^+ from r^-
Assignment + decoder	A vehicle per request; a constructive decoder sequences each route. Pairing safe; the decoder does the hard work and hides it from evolution
Route-based, signed requests	One sequence per vehicle of $+r$ (pickup) and $-r$ (drop-off). Pairing and precedence enforced by every operator; time windows and capacity left to penalties and repair

For PDPTW the route-based encoding is the usual choice. A solution is $\sigma = (\rho_1, \ldots, \rho_K)$ with $\rho_k = (v_{k,1}, \ldots, v_{k,m_k})$, every request appearing in exactly one route as $+r$ before $-r$. Its schedule follows from forward recursions along each route, starting at the depot with $D_{k,0} = 0$, $Q_{k,0} = 0$:

$$\begin{aligned} A_{k,i} &= D_{k,i-1} + t_{v_{k,i-1} v_{k,i}} && \text{(arrival)} \\ B_{k,i} &= \max\{A_{k,i},\, e_{v_{k,i}}\} && \text{(service start)} \\ D_{k,i} &= B_{k,i} + s_{v_{k,i}} && \text{(departure)} \\ Q_{k,i} &= Q_{k,i-1} + q_{v_{k,i}} && \text{(load)} \end{aligned}$$

so a route is evaluated in time linear in its length, and the plan is feasible when every $B \leq l$, every $Q \leq C$ and every ride time is within L_r.

Constant-time feasibility checks make insertion cheap

Almost every operator inserts or removes requests, so the cost of testing an insertion dominates run time. Savelsbergh's forward time slack makes the time-window test constant-time. With waiting $W_{k,p} = B_{k,p} - A_{k,p}$, the slack at position i is

$$F_{k,i} = \min_{i \leq j \leq m_k} \left(l_{v_{k,j}} - B_{k,j} + \sum_{i < p \leq j} W_{k,p} \right),$$

the largest delay that can be pushed into position i without violating any later window. An insertion that delays position i by Δ respects all windows if and only if $\Delta \leq F_{k,i}$. Precomputing F and prefix loads per route turns each candidate pickup-and-drop-off position pair into an $O(1)$ test, with ride times checked for the requests whose span the insertion crosses.

Penalize soft violations, adapt the penalty, repair the rest

Pairing and precedence are hard and are enforced by the encoding. Time windows, capacity and ride time are handled in the fitness, letting the search cross infeasible regions:

$$\begin{aligned} f(\sigma) = c(\sigma) &+ \omega_{\mathrm{TW}} \sum_{k,i} \left(B_{k,i} - l_{v_{k,i}}\right)^+ \\ &+ \omega_Q \sum_{k,i} \left(Q_{k,i} - C\right)^+ + \omega_L \sum_r \left(\mathrm{ride}_r - L_r\right)^+, \end{aligned}$$

where $c(\sigma)$ is travel time and $(x)^+ = \max\{x, 0\}$. The weights adapt so that a target share ξ^* of new children is feasible: after each generation with observed share ξ,

$$\omega \leftarrow \begin{cases} \omega(1+\gamma) & \xi < \xi^* - \varepsilon \\ \omega(1-\gamma') & \xi > \xi^* + \varepsilon \\ \omega & \text{otherwise,} \end{cases}$$

with ξ^* typically 0.2 to 0.4 (Vidal et al., 2012). A penalty that is too high makes the search timid; too low, and it settles in infeasible space.

Repair complements penalties when a violation is cheap to fix locally. Requests on violated routes are removed and reinserted by cheapest feasible insertion. For request r, route k and positions $i \le j$ for its pickup and drop-off, the insertion cost is

$$\Delta_k(r,i,j) = c\big(\rho_k \oplus_{i,j} r\big) - c(\rho_k),$$

and positions failing the slack test are skipped. Reinserting several requests greedily can strand the hard ones, so *regret-h* insertion (Ropke and Pisinger, 2006) inserts next the request maximizing

$$\sum_{g=2}^{h} \big(\Delta_r^{(g)} - \Delta_r^{(1)}\big),$$

where $\Delta_r^{(g)}$ is its g-th best route option: a request with only one good vehicle is placed before a flexible one takes that slot. Repair can be *Lamarckian*, writing the repaired plan back into the population, or *Baldwinian*, scoring the repaired plan while keeping the original genotype. Lamarckian repair converges faster; Baldwinian preserves more diversity.

Variation, local search and diversity

Crossover must recombine without breaking pairs. Route-based crossover copies parent 1, injects a route from parent 2, deletes the requests that now appear twice and reinserts any that went missing by regret insertion, which is also where repair happens. *Mutation* relocates or exchanges whole requests between vehicles and merges or splits routes. *Local search* on each child, with the same pair-preserving moves applied until no move improves, supplies most of the solution quality in modern hybrid genetic algorithms. Parents are chosen by binary tournament.

Survivors are chosen to balance cost and diversity. A useful distance between plans is the *broken-pairs distance*: the share of consecutive stop pairs in one plan that are not consecutive in the other. Each individual receives a biased fitness

$$\mathrm{BF}(\sigma) = \mathrm{rank}_c(\sigma) + \Big(1 - \frac{n_{\mathrm{elite}}}{|\mathcal{P}|}\Big)\mathrm{rank}_d(\sigma),$$

combining its cost rank with the rank of its mean distance to its closest neighbours (Vidal et al., 2012); the worst by BF are culled, the best n_{elite} by cost are kept, and the population is partly restarted after long stagnation.

A HYBRID GENETIC ALGORITHM FOR ONE DISPATCH EPOCH

Require: frozen stops, open requests, time budget τ, previous plan σ^{prev}
$\mathcal{P} \leftarrow \{\sigma^{\mathrm{prev}}\} \cup$ mutants of $\sigma^{\mathrm{prev}} \cup$ regret-insertion builds
while elapsed $< \tau$ **do**
  $p_1, p_2 \leftarrow$ binary tournaments on f
  $c \leftarrow \textsc{RouteCrossover}(p_1, p_2)$; mutate with prob. π_m
  $c \leftarrow \textsc{Repair}(c)$ ▷ regret-h reinsertion
  $c \leftarrow \textsc{LocalSearch}(c)$ ▷ pair relocate, exchange
  $\mathcal{P} \leftarrow \mathcal{P} \cup \{c\}$; cull by BF if $|\mathcal{P}| > \mu + \lambda$
  update ω toward feasible share ξ^*
end while
return best feasible σ by $c(\sigma)$

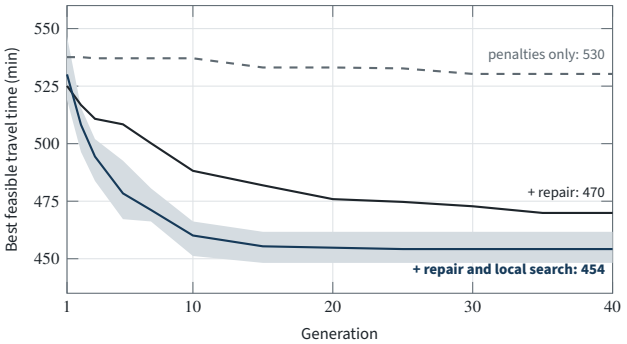

Exhibit 1. Best feasible plan cost by generation for three variants of the same genetic algorithm, mean of five seeds; band shows the range across seeds for the full hybrid. Synthetic static PDPTW: 36 requests, 6 vehicles of capacity 3, 15-minute pickup windows; the script uses a simplified diversity measure and biased fitness.

Exhibit 1 isolates what each ingredient contributes on a small synthetic instance, with every variant starting from the same kind of population and running the same budget. Penalties alone barely improve the starting plans: random reinsertion rarely lands requests in good, feasible positions. Adding regret-insertion repair cuts the final cost by 11%; adding local search cuts it by 14% and, as importantly, shrinks the range across seeds to 13 minutes, against 53 with repair alone and 44 with penalties alone, so the plan a dispatcher gets depends far less on luck.

Making it work inside a live dispatch loop

In dispatch, the problem changes while it is being solved. At decision epochs $t_1 < t_2 < \dots$, typically every few seconds or on each new request, the stops already served, the leg each vehicle is driving and every pickup promised to a rider become a *frozen prefix* of its route; only the remainder is open to search. The population is seeded from the previous epoch's survivors with the new requests inserted by regret insertion, so search resumes near a good plan instead of starting over. Each epoch runs for a fixed budget τ inside the dispatch cycle and returns the best feasible plan found; anything unplaceable within the service standard is declined explicitly. The routing work also produced a patented dynamic programming method for route optimization; this note does not describe its claims.

Parameters, complexity and what a genetic algorithm cannot certify

Hybrid genetic algorithms are less sensitive to tuning than their reputation suggests, but a few parameters matter. The values below are typical starting points from the vehicle-routing literature and from the synthetic experiment, to be tuned on a representative set of the operation's own epochs.

Parameter	Typical setting and effect
Population μ, offspring λ	$\mu \approx 25$, $\lambda \approx 40$ in HGS; larger values explore more and converge more slowly
Elite n_{elite}, neighbours n_{close}	A handful each; they set how strongly diversity enters BF
Feasible share ξ^* and steps γ, γ'	$\xi^* \approx 0.2$ to 0.4; multiplicative steps such as 1.2 and 0.85
Regret depth h	2 or 3; deeper regret protects constrained requests at higher cost
Epoch budget τ	Set by the dispatch cycle; quality should be measured as a function of τ

The cost of a generation is dominated by insertion. Evaluating a route is $O(m)$ in its length m; with forward slack, testing all pickup and drop-off position pairs of one request on one route is $O(m^2)$, so placing one request across K vehicles is $O(Km^2)$ and one regret pass over R requests is $O(R^2Km^2)$ in the worst case. Restricting candidate vehicles to those whose current position can reach the pickup within its window, a simple geometric filter, usually removes most of that work.

A genetic algorithm returns good plans, not proofs. To know how good, solve a set-partitioning reformulation, $\min\{\sum_\rho c_\rho y_\rho : \sum_{\rho \ni r} y_\rho = 1 \ \forall r,\ \sum_\rho y_\rho \le K\}$ over feasible routes ρ, by column generation on a sample of recorded epochs offline. Its linear-programming bound z_{LB} gives the optimality gap $(z_{\mathrm{GA}} - z_{\mathrm{LB}})/z_{\mathrm{GA}}$ of the plans the dispatcher actually produced, which is the honest way to report heuristic quality.

Running the optimizer inside the simulator

Calling the genetic algorithm at every simulated epoch makes the simulator expensive: a service day with E epochs costs about $E\tau$ of optimization per replication. Two practices keep it tractable. Give the simulated dispatcher the *same* budget τ as production, so simulated service reflects the plan quality the live system will achieve rather than an idealized optimizer; and screen candidate fleet sizes with fewer replications first, reserving full replication counts for the sizes near the decision.

2. A simulation that runs the real policy

Why spreadsheet capacity models fail here

A capacity spreadsheet multiplies a demand rate by an average cycle time and divides by a target utilization. It cannot see that demand peaks last a couple of hours, that a short queue absorbs much of a peak, that vehicles end trips in the wrong places, or that routing and pooling change the effective cycle time. Each of those effects moves the answer by tens of percent, in different directions.

The discrete-event structure

A discrete-event simulator advances a clock from one event to the next: a request arriving, a vehicle being assigned, a pickup, a drop-off and, where used, a repositioning move. Demand is generated from the observed pattern by hour and zone, either as a non-homogeneous Poisson process fitted to request logs or by resampling historical days, which preserves correlations a parametric model may miss. The decisive design choice is that the simulator calls the same genetic-algorithm dispatch and routing logic that runs in production, so the fleet is sized for the policy the business will actually operate.

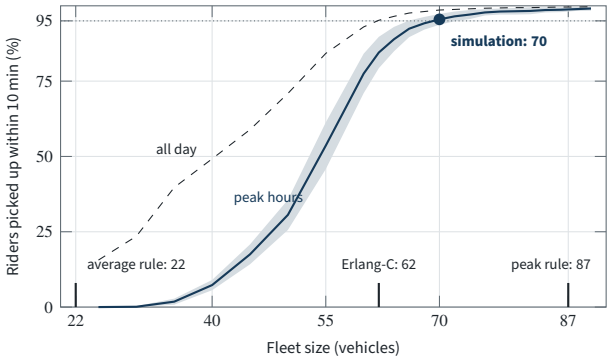

Exhibit 2. Share of riders picked up within 10 minutes against fleet size, in the evening peak (solid, with 95% confidence band) and over the whole day (dashed), against a 95% peak-hour standard (dotted). Synthetic model, 30 replications per point with common random numbers.

A worked example

Exhibit 2 comes from a deliberately simple model built for this note: a 10 km by 10 km grid with Manhattan distances, vehicles at 30 km/h carrying one party at a time, assignment of each request to the vehicle that can reach it first, and about 1,850 trips a day with morning and evening peaks. The average trip takes 13.3 minutes; a planner's rule of thumb adds a 4-minute pickup for a cycle of about 17 minutes.

An *average rule*, average demand times cycle time, asks for 22 vehicles and serves fewer than one rider in six on time over the day, and none at peak. A *peak rule*, peak demand times cycle time at a planning utilization of 70%, asks for 87. The simulation's mean crosses the 95% peak-hour standard at 70 vehicles, about 20% fewer than the peak rule; requiring the lower confidence bound to clear 95% raises the answer to 74, still 15% fewer. The real system's routing, pooling and demand were far richer, but the shape of the result is general.

An analytic floor from queueing theory

Queueing theory gives a quick check on any simulation. Treat the fleet at each moment as an $M/M/N$ queue with the current arrival rate $\lambda(t)$ and mean busy time per trip $1/\mu$, here 16.0 minutes including 2.8 minutes of empty driving to the pickup. With offered load $a = \lambda/\mu$ and $N > a$, the Erlang-C formula gives the probability that a request must wait,

$$C(N,a) = \frac{\dfrac{a^N}{N!}\dfrac{N}{N-a}}{\displaystyle\sum_{k=0}^{N-1}\frac{a^k}{k!} + \frac{a^N}{N!}\frac{N}{N-a}},$$

and the wait is exponential beyond that, $\Pr\{W_q > t\} = C(N,a)\, e^{-(N\mu-\lambda)t}$, so the standard "picked up within w is met when $1 - C(N,a)\, e^{-(N\mu-\lambda)(w-\bar{d})} \geq 0.95$, with $\bar{d}$ the mean empty-driving time to the pickup. Applying it hour by hour, the pointwise stationary approximation of Green and Kolesar, the worst hour needs 62 vehicles. The figure is optimistic, and should be: it pools every vehicle into one queue, ignores that empty driving grows when few vehicles are free, and assumes the queue reaches steady state within each hour. An analytic floor below the simulation, by a plausible margin, is evidence the simulation is not wildly wrong; a simulation below the floor would signal a bug.

Output analysis: a service day is a terminating simulation

A service day that starts and ends nearly empty is a *terminating* simulation. The correct analysis is independent replications, one day each, with a confidence interval across days, not batch means within a single long run. With n replications of a daily service share $X_1, \ldots, X_n$, sample mean $\bar{X}$ and standard deviation s, the interval is $\bar{X} \pm t_{n-1,1-\alpha/2}\, s/\sqrt{n}$, and the replications needed for a target half-width h^* follow from a pilot run as $n^* \approx n\,(h/h^*)^2$. Warm-up deletion, for example by Welch's method, belongs to steady-state questions such as "how does the fleet perform during a sustained peak"; for a whole day, the initial state is part of the system and should be realistic, not deleted.

Common random numbers, feeding every fleet size the same demand days, make comparisons far sharper. For the service shares X and Y of two fleet sizes,

$$\mathrm{Var}(X - Y) = \mathrm{Var}(X) + \mathrm{Var}(Y) - 2\,\mathrm{Cov}(X,Y),$$

and identical demand makes the covariance large and positive. In the worked model, the difference in peak service between 66 and 68 vehicles is 1.8 ± 0.9 points with common random numbers; with independent demand draws the same comparison has a half-width of 3.5 points, four times wider, and would need about sixteen times as many replications for the same precision.

Choosing the fleet is optimization via simulation

Because service rises with fleet size, the smallest adequate fleet can be found by bisection on N, replicating at each candidate until the confidence interval clears or misses the target decisively. Comparing several dispatch or repositioning policies, rather than fleet sizes, is a selection problem, for which ranking-and-selection procedures allocate replications to the competitive alternatives and stop with a guaranteed probability of choosing the best.

Validating the simulator before believing it

A simulator earns trust by reproducing the operation it models. Replay a set of historical days through it, using the actual requests, and compare simulated and observed distributions of wait, ride time and vehicle utilization by hour, not just their averages. Check the demand model separately: the variance and peak shape matter as much as the mean. Only then does the curve in Exhibit 2 become a planning instrument rather than a hypothesis.

3. Decide fleet, service and dispatch together

Fleet size, the service standard and the dispatch policy are not independent. A smarter dispatch policy shortens effective cycle times and lowers the fleet needed; a stricter wait target raises it; pooling riders reduces vehicle-hours but lengthens ride times; repositioning idle vehicles toward expected demand cuts empty driving at the cost of extra miles. Testing them jointly turns a single capital number into a set of trade-off curves leadership can read, one per policy, each re-run at plausible demand growth so the fleet decision is robust to the forecast as well as to the day-to-day variability.

The fleet grows more slowly than demand

Re-running the curve at other demand levels shows a property capacity spreadsheets miss entirely. Exhibit 3 finds, for each demand multiple, the smallest fleet whose mean peak-hour service reaches 95%, by bisection on fleet size with common random numbers.

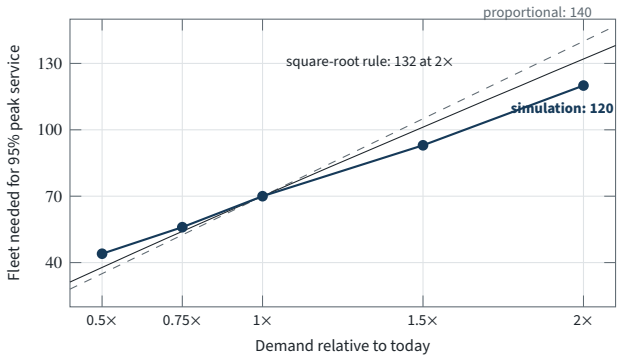

Exhibit 3. Smallest fleet meeting the 95% peak-hour standard at each demand level (points), against proportional scaling from today's 70 vehicles (dashed) and the square-root staffing rule calibrated at today's demand (solid). Same synthetic model, 30 replications per candidate.

Doubling demand calls for 120 vehicles, not 140. Two economies of scale are at work. The first is statistical and classical: in an $M/M/N$ system the staffing that holds a service level grows as $N \approx a + \beta\sqrt{a}$ for offered load a, the square-root rule of Halfin and Whitt, where β is set by the target; doubling a multiplies the safety margin $\beta\sqrt{a}$ by only $\sqrt{2}$. The second is spatial: a denser fleet is closer to each request, so empty driving per trip falls as the fleet grows. The simulation captures both, which is why it beats even the square-root rule at high demand and needs more than the rule predicts at low demand, where vehicles are sparse. For a growing service, sizing proportionally to a demand forecast over-buys; for a launch in a thin market, a queueing formula alone under-buys.

Simplification in the worked model	Likely effect if relaxed
One party per vehicle	Pooling lowers the fleet needed; ride times rise
No repositioning	Repositioning lowers empty driving and the fleet needed
Uniform origins and destinations	Spatial imbalance raises the fleet needed
Nearest-arrival dispatch	Optimized dispatch, as with a genetic algorithm, lowers it
No cancellations	Riders who abandon long waits change the service metric

REFERENCE DECISION FLOW

Request patterns → genetic-algorithm routing → validated discrete-event simulation → service versus fleet curve with confidence → fleet commitment

4. What the project reported

The simulation identified a 27% reduction in required fleet size while maintaining service, and the routing algorithms scheduled more than 300,000 trips. These are project-reported figures and have not been independently audited by Bit Bros. The worked example above is illustrative and is not a reconstruction of the project's model or data.

5. Lessons that transfer

Size capacity for the policy that will run

A capacity model that ignores dispatch logic sizes a different system from the one the business will operate. Calling the production policy from the simulation removes that gap.

Peaks are short; queues are cheap to model

Most of the excess in peak-based sizing comes from treating a two-hour peak as if it lasted all day. A simulation prices the brief queue that absorbs it, and a queueing formula bounds it from below.

Report a curve with confidence, not a number

Leaders choose between service and capital. Give them the curve with its confidence band, and say whether the fleet is sized to the mean or to the bound. The difference, 70 against 74 vehicles in the worked model, is itself a risk decision.

A USEFUL EXECUTIVE DIAGNOSTIC

Was your capacity plan computed by running your actual dispatch policy against realistic demand, with a confidence interval, or by dividing a peak rate by a utilization target?

Further reading

Prins, C. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research* 31(12), 2004.

Vidal, T. et al. A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research* 60(3), 2012.

Cordeau, J.-F. and Laporte, G. The dial-a-ride problem: models and algorithms. *Annals of Operations Research* 153, 2007.

Savelsbergh, M. W. P. The vehicle routing problem with time windows: minimizing route duration. *ORSA Journal on Computing* 4(2), 1992.

Ropke, S. and Pisinger, D. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science* 40(4), 2006.

Nanry, W. P. and Barnes, J. W. Solving the pickup and delivery problem with time windows using reactive tabu search. *Transportation Research Part B* 34(2), 2000.

Halfin, S. and Whitt, W. Heavy-traffic limits for queues with many exponential servers. *Operations Research* 29(3), 1981.

Green, L. and Kolesar, P. The pointwise stationary approximation for queues with nonstationary arrivals. *Management Science* 37(1), 1991.

Kim, S.-H. and Nelson, B. L. Selecting the best system. In *Handbooks in Operations Research and Management Science*, vol. 13, Elsevier, 2006.

Law, A. M. *Simulation Modeling and Analysis*, 5th ed. McGraw-Hill, 2015.

About the case and Bit Bros

This paper draws on mobility routing and fleet-sizing work led by **Adam DeJans Jr.** in an earlier industry role. It was not a Bit Bros client engagement. The company's proprietary algorithms and parameters, the patented method's claims, and its data are not disclosed, and the mathematics is simplified to explain the approach. Bit Bros is a boutique applied economics and optimization firm that helps operating teams price their decisions and build the systems that make them repeatable.

Adam DeJans Jr.
Technical Managing Director, Bit Bros LLC
Decision scientist specializing in routing, simulation, optimization and decisions under uncertainty.

Simulate the policy you will run. Then buy the fleet.