Smart Routing: A Novel Application of Collaborative Path-finding to Smart Parking Systems

Callum Rhodes, William Blewitt, Craig Sharp, Gary Ushaw and Graham Morgan
School of Computing Science, Newcastle University
Newcastle upon Tyne, United Kingdom
Email: (c.g.rhodes, w.f.blewitt, craig.sharp, gary.ushaw, graham.morgan)@newcastle.ac.uk

***Abstract*—We utilise collaborative path-finding to improve efficiency of smart parking systems and therefore reduce traffic congestion in metropolitan environments, while increasing efficiency and profitability of parking garages. A significant portion of traffic in urban areas is accounted for by drivers searching for an available parking space. Many cities have adopted a parking guidance and information system to try to alleviate this traffic congestion. Typically these systems entail informing the driver of the whereabouts of an available space, reserving that space for the specific driver, and providing directions to reach the destination. Little or no account is taken of how much congestion will be caused by multiple drivers being directed to the same car-park concurrently. We introduce the concept of collaborative path-finding to the problem. We simulate a smart parking system for an urban environment, and show that a novel approach to collaboratively planning paths for multiple agents can lead to reduced traffic congestion on routes toward busy parking areas, while reducing the amount of time when parking spaces are vacant, thereby increasing the revenue earned.**

I. Introduction

Drivers searching for a vacant car-parking space can account for more than 30% of traffic in a metropolitan area at any particular time [1]. Clearly a smart parking system which can efficiently guide motorists to available parking spaces could alleviate this problem. Traffic authorities in many cities have instigated parking guidance and information (PGI) systems, providing drivers with up-to-date information on the availability and location of parking spaces [2]. The information may be presented to drivers via dynamic street signage, or over the internet.

Many smart parking schemes exist based on resource allocation and reservation [3], [4], [5], whereby the PGI system knows how many spaces are currently available at each site and drivers are directed accordingly. The systems are typically based on locating the car-park or street with available spaces which is nearest to either the driver's entry point into the controlled area, or the driver's intended destination within that area. Many systems also identify the most suitable space by including a pricing factor, sometimes based on auction or electricity trading (in the case of electric or hybrid vehicles) [6]. When the target parking space has been identified and reserved, a Global Positioning System (GPS) can be used to plot the driver's route to the parking destination. This can result in multiple vehicles being directed toward the same parking garage at the same time, or along routes which cross over one another, which can lead to further traffic congestion along those routes.

In this paper we introduce the concept of collaborative path-finding to the field of smart parking. We adapt a standard A-star path-finding algorithm to incorporate multiple agents plotting paths concurrently, while taking into account one another's progress along their assigned routes. In our simulation, the agents represent drivers being assigned a parking space, the destinations are the locations of the parking spaces themselves, and the nodes of the path-finding grid are the streets and junctions of the metropolitan area. Our approach considers multiple scenarios wherein agents have taken different decisions in order to avoid over-occupying the same node on the path-finding grid. A selection technique is employed to identify the scenario which provides the most efficient solution for all agents at any particular time.

We show that employing this "smart routing"scheme within a PGI system can be beneficial in a number of ways. Congestion is reduced, as drivers are sent along routes which do not interfere with one another. A dynamic approach ensures that, as new drivers enter the controlled area, they are not only assigned an available space, but are assigned a route which causes minimal further congestion. Journey times for drivers are therefore reduced. The approach also leads to greater efficiency for the parking garages themselves, as spaces are vacant for smaller amounts of time, so revenue is earned over a greater proportion of the day.

II. Background and Related Work

In this paper we build on our previous work in applying complex system protocols to domains of commercial interest [7]. We present a brief background to the fields of smart parking and multi-agent path-finding, in order to arrive at our contribution of smart routing.

A. Smart Parking

Parking guidance and information systems play an increasingly vital role in most major metropolitan areas worldwide [3]. Car parking is a revenue generator, rather than a cost centre, in most cities. Utilising a smart parking system can have a positive effect on that revenue due to improved occupancy rates, market-sensitive pricing and more efficient revenue collection. Further to this, the benefits to both commerce and the environment of reducing traffic congestion make smart parking an attractive proposition.

Earlier implementations of PGI schemes involved informing drivers on the availability of spaces and guiding them toward parking garages or streets identified as having free

spaces [2]. This could often result in many drivers being directed toward the same place while car-parks with only a few spaces were being ignored, even if they presented a better solution. More recently, schemes have introduced the concept of reservations, whereby a a driver is allocated a specific space which is then marked as unavailable until the specific driver arrives [4]. The reserved space may be password protected until the assigned driver arrives (passwords are communicated through SMS) [8]. The reservation approach has been augmented by the introduction of, for example, auctioning, price factoring, and trading of electricity, in the case of hybrid and electrical car schemes [6].

Technology for detecting whether a parking space is occupied (including inductive loops, weight sensors, pneumatic road tubes, etc.) is beyond the scope of this paper; for our purposes it is assumed that information on the occupancy and location of spaces is available and correct.

Little or no work has been carried out on planning how drivers reach their allocated parking space. Existing systems typically rely on GPS navigation for individual drivers, or dynamic roadside signage for directing many vehicles along a shared route [5]. These approaches take no account of the congestion, and therefore time delays, introduced by sending multiple vehicles along shared routes toward reserved smart parking spaces. Further congestion can be introduced at streets or traffic junctions where directed routes intersect. In this paper we address for the first time the issue of planning route information for multiple vehicles approaching allocated parking spaces, by the application of a novel collaborative path-finding approach.

B. Collaborative Path-finding

Finding a path through an environment for a single agent is a well understood problem [9]. However extending this work to consider multiple agents working together to find the most mutually beneficial paths is an ongoing research topic. The multi-agent problem is PSPACE-hard, wherein the number of states and branching factor grow exponentially with the number of simultaneously moving agents [10].

David Silver [11] explores an algorithmic solution to the issue which sacrifices some optimality, and introduces behavioural irregularities, in order to minimise the branching complexity of the issue. While Silver's system is highly intuitive, and appropriate for many real-time systems where pathing conflicts can be offset using secondary algorithms, it is vulnerable to unresolvable conflicts in more complex systems.

It has been noted that, in the non-collaborative case, multi-agent path planning is suitable for general purpose GPU computing due to the independently parallel nature of the calculations [16]. Significant efforts have been made to reduce the complexity of A-star for multiple agents through subdivision of the search space [12], [13], demonstrating performance gains over the traditional case. Many algorithmic approaches balance computability against absolute optimality [14], [15].

Our approach in this context is similar, in that absolute optimality is not necessarily beneficial in the domain of traffic routing, given how swiftly situations on the road network can change. As such, we opt to relax our optimality conditions in favour of prompt, computationally inexpensive update throughput.

Also relevant to our own work, Sharon et al [17] proposed Conflict-Based Search, which minimises the search-space through consideration solely of collision points. In this fashion, Sharon's work has analogues with our own; the approach we shall proceed to outline in section III has a similar basis, but considers the problem from a more agent-centric rationale.

The original contribution of our work, then, comes in two parts. First, a novel, conflict-based approach to multi-agent path planning and, second, the application of that approach to smart parking techniques. We call this combined, collaborative approach *smart routing*.

III. Protocol

In this section, we present our smart routing protocol for multi-agent path finding. Our algorithm is a variant on multi-agent path finding using A-star. Utilising a square grid we represent a portion of a city, each square containing data concerning the capacity of the corresponding area of the road network. Using the idea of reversing the direction of A-star we determine the exact distance to a goal. Employing reverse A-star as our heuristic for distance, we reason about the possibilities that a square on our grid can take, each possibility being a specification of that particular square. This specification allows us to reference a look up table to help make decisions.

We apply an ordering to each cardinal direction, enabling us to remove arbitrary decisions from the system. This implies an ordering on paths from their respective starts to their goals. Using specific paths relating to this ordering we can construct any possible route on the grid. We call routes constructed in this manner vehicle routes or *agentverses*. We select groups of vehicle routes together to represent all the vehicles in the system. We call these groups collective routes or *multiverses*.

The goal of our algorithm is to select a collective route which minimizes the total congestion within the system. We achieve this by applying A-star to the collective routes. We run a set number of iterations of A-star over the collective routes, storing results with low congestion while the algorithm maintains lower total path length. When congestion is detected within a collective route, the algorithm redirects random portions of the traffic to create a new collective route which is reintroduced back into the algorithm.

In our experiments the path finding algorithm is applied once every time step of the system. This simulates an evolving system, allowing us to introduce new vehicles into the system to test the adaptability of the protocol. Key elements of the algorithm are discussed below.

A. Path finding

Our approach to optimised car-park routing requires the definition of several algorithmic terms that are used throughout this work. In this section we outline the concepts underpinning the algorithm.

Grid Square: In this context, grid squares of the map (or graph) can be considered to represent likely points of intersection between flows of traffic rather than a truly representative route-map of a city. The grid square structure contains several important variables in our algorithm. Specifically, the structure permits us to determine the position of a grid square, its implicit neighbours, and a capacity value for each direction (including 'pausing', which is an increment through time rather than space). These capacities represent sustainable throughflow of traffic.

Node: One approach to solving multiagent path planning is to represent the state of every agent at every timestep as a node. In our simulation, the term node represents a collection of potential, complete routes - one for each vehicle within the system.

Reverse A-star: Reverse A-star is a useful technique to calculate a perfect heuristic [11]. The principles of generic A-star [9] are applied to every square on the graph, from the agent's goal. The heuristic cost is therefore not a 'best guess', but an accurate cost from a given square *n* to the goal assuming no changes to the environment occur during the journey. Given that an A-star system is always aware of the absolute cost taken to reach the square under consideration, Reverse A-star provides an exact and absolute cost for any given route. For our purposes, this should be considered the exact cost (in terms of time to travel the route) between a given carpark and any point at which a given car might be located in the map, assuming the car does not need to avoid traffic choke points.

Node Possibilities and Costs: Assuming a normalised graph, the application of Reverse A-star facilitates a valuable algorithmic optimisation. Specifically, it permits us to reason that if we consider a given square *n*, that the neighbouring squares to *n* can only have one of three heuristic pathing costs associated with them: h, $h+1$ or $h-1$, where h is the cost of the considered square *n*.

In the special case of a square grid the possibilities are reduced to $h+1$ and $h-1$. In our applied protocol, when combined with the perfect heuristic outlined above, this permits us to employ our path-planning in a step-by-step fashion. In terms of engineering optimisation, this enables us to assign a specification to each square and reference a look-up table for swift decision-making when rerouting.

Path Ordering: A consequence of the normalised graph in conjunction with Reverse A-star is that the system will often be required to decide between squares of equal heuristic cost (if re-pathing, $h+1$). In order to assist that decision-making, we introduce a new property to the pathing algorithm which favours (prioritises) one direction over another of equal cost.

For our purposes when plotting routes for multiple agents through time, pausing is considered a direction in its own right, and indicates not only consequences of traffic flow but, additionally, opportunities which careful traffic flow management can encourage. The inclusion of a direction priority implicitly defines an ordering on the paths from a square to a goal; from each square on the graph we can select a preferred, unique path to the goal, which we call the α path. A path with a higher f value would be higher on the list, but within the same f value set ordering is decided lexically with respect to the ordering of directions. We call this process 'preferential ordering'.

B. Vehicle Route or Agentverse

We employ the term vehicle route, or agentverse, as a means of differentiating between the path proposed by a single vehicle to reach its destination, and the overall collective route or multiverse. A vehicle route is the path proposed for a given vehicle to reach its destination carpark, with no consideration of the other cars moving through the map. A key assumption in our approach is that vehicle routes will be largely similar to their most optimal paths, meaning they rank lower (better) in terms of preferential ordering. The collective route, functionally, is a collection of suitable vehicle routes which reduce congestion.

We represent vehicle routes in our algorithm as segments of optimal paths. As such, we can use segments of α paths to represent all paths. Vehicle routes are represented using a start time, an α path, and a past. The past is a reference to another vehicle route from which the currently considered route branched. In this fashion, the solution's memory footprint and computational complexity is lowered. The beginning of a particular vehicle route segment connects to a point along another route segment for the same vehicle.

These connections are the only way a vehicle route can transition in a manner out of the preferred order. They are triggered generally in a case where an α path is found to be potentially non-viable due to square occupancy (too many vehicles being advised to pass through a given intersection, potentially leading to unmanageable congestion).

Let us consider the following example, at time step t. The directions South and West are toward the goal. South is favoured by preferential ordering, but we can construct a vehicle route with start time $t+1$ starting directly to the West of the current square. This vehicle route, with the original as its past, would have the same f value as the current vehicle route but would represent a path which took West as its next direction at time t.

The same can be done with the North and East directions but this will yield a vehicle route with an increased cost of $f+2$. A pause can also be represented. This is done by repeating the current square as the start of the α path, with the same start time of $t+1$ and past, as the other examples.

The only vehicle route segments without a past are the initial vehicle routes computed at the beginning of the planning algorithm. This initial vehicle route stores the complete α path from the vehicle's start square to the goal, with a start time of 0. All vehicle route segments will ultimately form a chain that leads back to this initial vehicle route, forming a traceable tree from which the final vehicle route is assembled.

One potential issue with regards process efficiency and memory management is the recreation of vehicle route segments which already exist. We circumvent this issue by storing forward pointers alongside the α path step data. Each step along the path can store a pointer to a single vehicle route. If the time of that step is t then the vehicle route would have start time $t+1$, meaning that that step would be the corresponding vehicle route's immediate past. There can be up to at most 4 vehicle routes that would share that timestep as its immediate past, (other directions and pause at time step t + 1). The other

vehicle routes are stored in a linked list; each vehicle route stores a *next* pointer to facilitate the linked list.

C. Path Progression "Next"

As highlighted in the previous section, multi-agent path-planning is a PSPACE-hard problem, with computation time exponentially connected to the number of agents being considered. In order to mitigate this issue, some form of pruning of the state space is required, based upon domain-specific assumptions. Our main assumption is that vehicles will pursue routes which mostly progress towards the goal. Hence we use combinations of fully formed vehicle routes as our nodes, which we term a collective route, which shall be outlined later in this section. The branches of these collective routes are made by replacing individual vehicle routes with like routes ranked higher (worse) in terms of preferential ordering. The algorithm we use to do this we call 'next'.

When a vehicle route is constructed the decision of which step to take is made immediately, comparing this to the traditional A-star approach (in which nodes represent a single timestep of the system), this results in branching decisions being made out of order. In situations where the traditional A-star approach would have selected from equivalent nodes on the priority queue to resolve a conflict, our algorithm has made the default decision of moving forwards, creating congestion which we need to resolve.

We can resolve these conflicts by adding in some of these decisions afterwards. In order to resolve these conflicts the full A-star algorithm would have picked a node representing an earlier timestep which could potentially avoid the conflict. Representing this in our algorithm we need to select a branch from a current conflicting vehicle route or one of its past vehicle routes. This branch must happen before the timestep of the conflict. We also prefer to branch later in time, as this avoids adding conflicts at timesteps we have already solved. The resulting vehicle route is of the same f value or higher, or equivalently higher on the preferential ordering list.

Given these facts our *next* algorithm needs to select the highest timestep before the conflict with a branch at a preferred f value. The algorithm can be split into two cases, 1) the current timestep we branch from is part of an α path of the vehicle route, 2) the timestep is a branch of the current vehicle route chain (i.e. the current timestep is t and the vehicle routes have start time $t + 1$).

In case 1) there is no need to consider directions past the second in our preferential ordering of directions (this includes Pause if no other direction moves forwards). If there is a need to consider these cases, they will eventually be reached as a subset of case 2), a branch of an vehicle route. This means the 'second priority' of a timestep becomes important in our algorithm. The second priority can be looked up using the specification of the current square, calculated from our augmented reverse A-star algorithm.

In order to speed up the process of finding the needed timestep each vehicle route stores an array of three indices (pq), and each timestep stores a single index (pq_next). The pq array stores the maximum time that a timestep has a second priority branch with a corresponding f value offset. This means that $pq(0)$ would store the maximum time step at which a branch can be made that would not effect the f value. The index of $pq(1)$, and $pq(2)$ correspond to vehicle routes with f value + 1, and + 2 respectively. The pq_next index creates a linked list of timesteps with the same second priority.

Using this extra data we can skip ahead to a known priority offset, then we can follow the linked list until we are below the conflict time. This can be interleaved with another technique; by starting at the timestep before the conflict and incrementally testing for the correct second priority value. Together one or the other process will eventually terminate, either with a result, or by proving there is no branch at that priority.

D. Branch Compression "Stem"

To reduce the number of branches we compress a set of nodes (each being a collective route) with a common property into a single meta node called the stem. This node can be inserted into the priority queue with priority equal to the minimum of the group of nodes it represents. When the stem is popped off the queue it can be expanded into a subset of the nodes it represents together with another stem of the remaining nodes.

This technique has the advantage of postponing the processing of a large number of nodes until they are needed. Stems work most efficiently if the stem can be constructed to have a higher f value than the remaining nodes, which will be processed before the stem.

In the case of the *next* incrementing algorithm, the stem is used to correctly split the higher f value branches from the lower ones. Given a vehicle route, repeatedly calling *next* with its result will give vehicle routes with successively lower branches, as these branches are always available to the algorithm. In this manner all branches at a particular f value can be represented by one node. However if a higher f value is needed those solutions may be cut off before they can be considered as possibilities. If a clash happens at timestep t and the first branch available is at timestep $s < t - 1$, then if the solution lies at a higher f value, branches made between s and t will not be considered (since the new branch replaces this time period).

Using a *stem* we can use *next* with the same vehicle route but using successively higher f values. This solves the issue of missing higher f value solutions.

E. Collective Route or Multiverse

The collective route, or multiverse, is a collection of vehicle routes, and represents a potential solution. These data structures make up the nodes of the main A-star algorithm. Each node has a cost associated with the sum length of the individual vehicle routes, and each vehicle route within the collective route has a corresponding integer value representing its current stem (i.e. the current f value *next* should look for). When a collective route is popped off the priority queue it is checked for congestion (starting from time 0 until all agents have reached their goal). Points of contention are placed onto a priority queue, with priority proportional to the amount of congestion.

After a congestion is detected and the point of highest congestion is popped off the priority queue, a number of random combinations of vehicle routes are incremented via *next*. These combinations form individual collective routes, and a corresponding collective route has its vehicle route's stems incremented. We maintain these stem nodes to preserve the structure of branches within our algorithm. As discussed earlier, if we did not maintain the stem, certain possibilities would be unreachable.

For each vehicle in a collective route we store the corresponding goal node. In this manner we can mix collective routes which use different goal nodes allowing a vehicle to choose between viable goals. However we restrict the number of possibilities, since if all combinations of vehicles and goals were considered the system would become unmanageable (this could be rectified with another stem system).

F. Traffic Planner

We define a data structure called the traffic planner. Its job is to simulate a use case of the algorithm. Every timestep a full path plan is completed for all vehicles currently in the system. Depending on the result we move each vehicle one step along the planned path. During each round of the full path planning we cap the maximum number of iterations, i.e. maximum number of collective routes considered. This ensures the algorithm takes a reasonable time to complete.

As vehicles get added to the system we randomly pick one goal in the simulation and free one space. This ensures the simulation can never reach deadlock, where a vehicle does not have a goal to reach. This does not change the semantics of the problem since if there wasn't a goal for a vehicle to use it would not have been entered into the system. As vehicles enter the system we do a full path plan with all goals available. The resulting goal from this solution becomes that vehicle's permanent goal.

A priority queue is maintained using a fitness criterion to decide priority. All the collective routes we iterate get pushed onto this results priority queue, such that if we reach the maximum number of iterations we do not select a collective route which introduces more congestion.

IV. Simulation

We model a portion of a city with a 9x9 grid map, each grid square representing a collection of intersecting roads. Each square corresponds to the same journey time segment (i.e. each square takes the same time to traverse), as opposed to a geometric correspondence. In practice these may correspond to geometric locations, but for our purposes a normalized cost applies.

Each cardinal direction of each square was assigned a random capacity between 1 and 3. These capacities simulate the different road throughputs. A capacity of 1 might represent a slow road, a multi-lane road subject to roadworks, or a road with arbitrarily low traffic throughput; similarly, a capacity of 3 might correspond to a high speed limit or a multi-lane road operating at full capacity. All non-goal pauses were left at capacity 1. This defines the maximum capacity of our simulated segment of city to be roughly 730 (the number of capacity slots a vehicle can occupy across the entire map). We assigned 9 goals to the map, representing car parks, and randomly distributed 125 available parking spaces among them.

We designed two scenarios for simulation; each scenario was run several times, generating two sets of results. Each set varied the number of agents in increments of 20 vehicles, starting from 20 and continuing up to 500 vehicles. Our proportional occupancy of the road network across its length equated to between approximately 3% and over 68%, giving us a broad spectrum of occupancy proportions for analysis.

The first scenario deals with a constant flow of traffic. Half the vehicles are initialised at the beginning of the simulation; these vehicles are positioned on the map using a pseudo-random algorithm which avoids goal squares (parking spaces). The remainder are introduced over the next 9 time steps. We call this scenario the 'standard traffic scenario'. This represents a situation where a large initial spread of agents are navigating towards parking spaces, and a lower, but still proportionally significant, number of vehicles enters our portion of the city over subsequent time-steps, during which all are cooperatively and reactively navigating. This ensures an approximate constant flow since the maximum distance a vehicle will travel is 18 on a 9x9 grid, on average a vehicle will travel less than half this distance. Extending the start time any higher than 9 means that we are no longer testing this vehicle in conjunction with the majority of the original vehicles (equivalent to the same results with less vehicles but higher throughput).

The second scenario starts all vehicles at time step 0. This scenario simulates initially high throughput, which tapers off as vehicles find their destination; this will give a higher peak in activity but will not introduce unexpected factors such as vehicles arriving to our portion of the city later in the simulation. The scenario can be considered representative of a traffic stress point, where the road network begins with proportionally high occupancy, relative to the number of vehicles on it.

Whenever vehicles are positioned in the system, they are done so using this pseudo-random method. An additional constraint, aside from the limitation regarding goal squares, is that the maximum number of vehicles which can begin in a square is equal to the sum of all its capacities (four cardinal directions, and a single pause). Agents are always added to the simulation in the same ordering. As agents are added a pseudo random goal is picked and its capacity is incremented. The sequence of goals which are picked is always the same ensuring comparisons between methods are meaningful.

Each scenario places strain on different aspects of our algorithm. The first shows how our protocol handles unexpected input into the system. Vehicles can be added to the area leading to congestion where there would not have been before. Our system tends to spread vehicles out among the grid squares, which may introduce congestion when we add new vehicles, but the system should be better adapted to handle this change.

The traffic stress point scenario has the advantage that vehicles are not introduced after the first time step, meaning that the original simulation results can be built upon. Successive time steps will pass the point in time at which a particular congested square was a problem and redistribute iterations onto the later time steps. However the beginning of the simulation

will start with a high number of vehicles in arbitrary clumps, which will be spread out through the network in later time steps; in this fashion, occupancy is disproportionately higher than in the former scenario. The fitness of later time steps should almost never exceed that of earlier time steps because there are fewer points of contention to consider (locations at points in time), however since branching is randomized this is not guaranteed to be the case.

We use two existing (non-collaborative) traffic routing protocols as our comparisons. The first simulates a road network without smart parking and therefore no parking reservation. Each vehicle uses an in-car route planner to find their route to the nearest car park (regardless of whether that car park has spaces available). In our simulation once these vehicles have reached their destination they are removed whether or not spaces are available. This is a best case for this simulation: vehicles which reach their goal are allotted spaces unknown to the original system. These vehicles would usually re-enter the system in search of another car park with an available space.

Our second test run simulates a smart parking system where parking spaces can be booked beforehand. This simulation is run as a single iteration of our algorithm, in which case each agent selects the nearest goal and deducts a space from it. This guarantees each vehicle a space at the end of its route. This test is representative of current smart parking solutions, which to date have not taken account of collaborative path finding. This smart parking approach would reduce congestion of traffic searching for a space among several car parks, which is not factored into the previous comparison.

The third test run represents our own system, combining smart-parking with centralised smart-routing. We selected between optimal computation time and result quality to decide the maximum number of iterations and chose a branching factor slightly higher than 1 to determine the number of vehicles redirected per congested square. We compare the overall congestion as our measure of success. However we also compare total path length to ensure we haven't mitigated congestion at the cost of path length. Our first test run will always have the minimal total path length. A good result would indicate that the additional cost of path length would be comparable to that of smart parking with no smart routing.

V. Results and Evaluation

The experimental results from our scenarios are presented as data sets comparing the congestion of the system with the summation of the total vehicle path lengths. Before informed conclusions can be drawn regarding these factors, they should be explicitly defined.

Total Path Length: Total path length represents the summation of each vehicle's route time from their entry into the simulation until they reach their respective goal. We use total path length as an indicator of the extra distance vehicles will have to travel as a consequence of avoiding congestion. We require the total path length to be comparable to that of our second comparison, smart parking.

Extension: We define extension of capacity as the difference between the activity of a capacity slot and the maximum capacity of that slot. The extension represents how much a particular slot is over capacity. In the case of a slot which is under capacity this value is 0.

Congestion: Total congestion is the summation of extension over the entire map during the entire simulation. This gives us a measure of fitness for our algorithm: the lower the resulting congestion, the better the algorithm has performed. This must be offset against any cost increase in total path length.

Point of Maximum Congestion: The point of maximum congestion is equivalent to the maximum extension, assessed throughout the entirety of the simulation, highlighting the location most vulnerable to congestion. The higher this value, the more likely a given route plan is to exacerbate congestion on a wider scale throughout the road network. This could be considered analogous to choke points causing traffic jams, or grid lock, in particularly busy areas of the road network.

A. Standard Traffic Scenario

The standard traffic scenario models a situation where a large amount of traffic begins within the system, and the traffic linearly increases over the course of the simulation.

Figures 1.1 and 1.2 illustrate the results obtained from the three test runs of this scenario. It should be noted that the number of vehicles along the x-axis indicates the total number of vehicles present within that iteration of the scenario, and does not represent an incremental increase on vehicle count for a single test. To illustrate, a vehicle count of 20 indicates an test where 10 vehicles were present in the simulation at time step 0, and 10 more were added over the course of the next 9 time steps; a vehicle count of 500 indicates a test where 250 vehicles were present in the simulation at time step 0, and 250 more were added over the course of the next 9 time steps.

In Figure 1.1 we note that the total pathing cost of our algorithm is marginally higher than the total pathing cost of both the non-collaborative smart parking and generic car routing solutions for vehicle counts above 100. We recall that the generic car routing solution shall always provide the lowest possible total path cost, as each vehicle plots a route to its destination without consideration to other vehicles, and our percentage comparisons are based upon this.

Taken on average across all vehicle counts, when compared to the car routing approach, the non-collaborative smart parking simulation is 5.55% more expensive. In comparison, our smart routing algorithm is 12.49% more expensive. This means that the minimum possible time for a vehicle to reach its goal is some 12.5% higher under our system than under a system with no smart parking or smart routing, *but* it assumes that congestion plays no role in the time taken to reach a destination.

We know that busy road networks are particularly vulnerable to congestion; the corollary to this is that the more congested a road, the higher the *actual* time taken to reach a destination, whatever the ideal shortest time might be. The reduction of congestion is a key goal in our ongoing research, and Figure 1.2 illustrates the performance of our smart routing algorithm in that context.

Statistically, we compare congestion against smart parking, rather than generic car routing. This is a more meaningful and

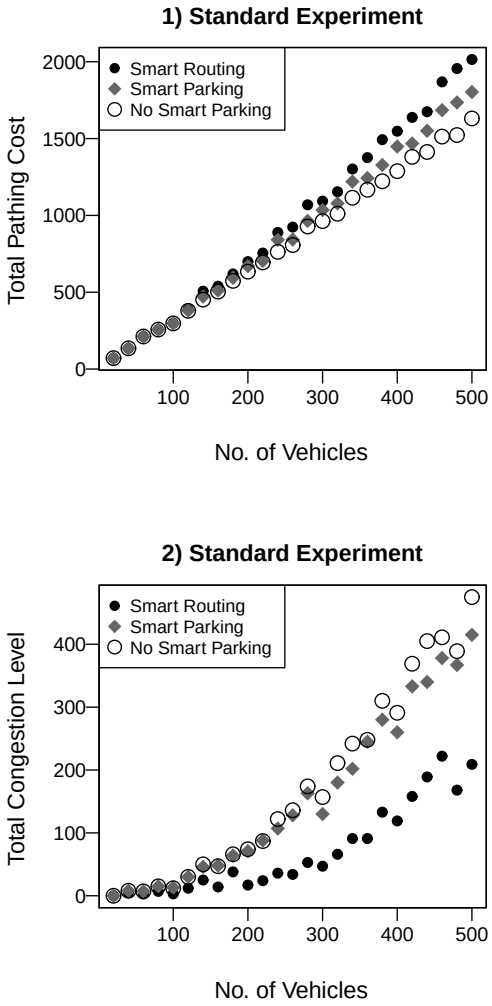

Fig. 1. Comparison of journey times and congestion level for standard traffic scenario (half of traffic introduced at start of simulation, half introduced linearly as simulation progresses).

challenging comparison for our algorithm, as smart parking has uniformly lower congestion values than generic car routing. Taking the average across all vehicle counts, the congestion level obtained through the application of smart routing is 42.00% that of the congestion observed when simulating smart parking without collaborative path finding.

At best, observed with 200 vehicles (road network occupancy of approximately 25%), the congestion value obtained through smart routing is 23.08% of that observed through smart parking alone. Considering our points of maximum congestion, smart routing reduces the congestion of these 'gridlock' areas by 25%, on average, and occasionally by as much as 58% (with 420 vehicles in the network).

Taking Figures 1.1 and 1.2 together, we can also compare the trends in relative performance. As vehicle count increases, there is a very visible performance benefit in terms of congestion level, while a far shallower performance hit in terms of total pathing cost. This invites significant financial benefits to the city as a whole: the road network is able to ferry more vehicles, consistently; the reduction in congestion means that time spent idling in heavy traffic is reduced, beneficial to both the local environment and consumer; commercial districts within a city can encourage a greater throughflow of high street consumers.

B. Traffic Stress Point Scenario

The traffic stress point scenario is designed to present a worst-case environment for our smart routing algorithm, where all vehicles accessing our portion of the road network arrive simultaneously and require collaborate routing en masse. Figures 2.1 and 2.2 illustrate this scenario's experimental results.

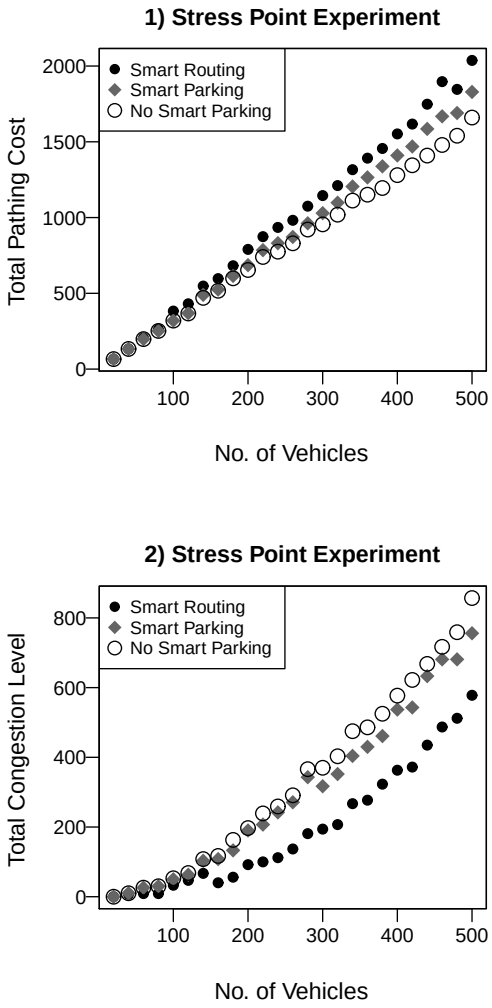

Fig. 2. Comparison of journey times and congestion level for traffic stress point scenario (high volume of traffic all introduced at start of simulation).

Comparing average total pathing cost increase once again with the best-case, minimum-cost paths provided by generic car routing, non-collaborative smart parking increases total path cost by 5.88%, while smart routing increases total path cost by 16.85%.

In the case of congestion, however, which we again compare to the smart parking case in order to illustrate

how collaborative route planning and smart parking can be employed in tandem to greater effect, there are significant performance gains. Taken across all vehicle counts, smart routing reduces congestion caused by traffic using the system to 58.92%, relative to smart parking alone. At the best case, which occurs with road network occupancy of approximately 10%, congestion is reduced by 70%.

The points of maximum congestion across all vehicle counts are lowered by an average of 13.32%, with the best case occurring at network occupancy of approximately 28%, where the worst 'gridlock' point's congestion was reduced by over 46%.

While these figures are not as impressive as those observed in the standard traffic scenario, this is to be expected as the stress point reflects a worst-case situation for our algorithm. The standard traffic scenario, where traffic is added and removed from the system over the course of time, is a better representation of true traffic flow.

Even in this worst case, congestion caused by traffic utilising smart routing is reduced by over 40% relative to smart parking alone. This, combined with the comparatively lower increase in total pathing cost, invites statistically significant financial benefits, both to the commercial districts of cities which might employ smart routing, and to enterprises whose performance hinges upon flowing road networks.

VI. Conclusions and Future Work

In this work we have introduced the concept of smart routing to the increasingly vital field of smart car parking. We have presented a novel algorithm addressing multi-agent path planning, and applied it to the problem of congestion within major cities. The algorithm has been described in detail, with domain-specific terminology employed where it eases understanding of the underpinning mathematics. Two scenarios have been presented, and the algorithm has been applied to them. Results of those experiments have been provided and discussed in some depth, and shown to be very encouraging.

The results outlined in Section V make a strong case for the adoption of collaborative route planning, in conjunction with existing smart parking technologies. The computational complexity of the operation is reduced through the algorithmic approach outlined; the relatively small increases in total route length are not indicative of an overall increase in journey time, as reduced congestion on the road network would benefit traffic flow throughout. The reduced congestion caused by drivers searching for spaces has clear implications for city governance, both in terms of increased revenue from parking charges and increased commercial and environmental benefits from better traffic flow in metropolitan areas.

With the advent of GPS systems which communicate through mobile telecommunications networks as a means of relaying real-time traffic data, and the inclusion on many internet-capable smartphones of GPS-based navigation software, much of the required infrastructure to pursue this technology is already in place. Such systems already have the capability to provide post-hoc assessments of traffic choke points. Smart routing, if employed in conjunction with existing traffic-flow modelling techniques, can provide a deeper insight into road network intersections which are exceptionally vulnerable to congestion, while its commercial implementation would help offset that very vulnerability.

This research aims to encourage the commercial exploration of this potentially beneficial area of information technology. Future work in this area shall explore the application of the prototype engineering solution to large road network segments, drawn from cities noted for their traffic flow issues.

References

[1] R. Arnott, T. Rave, and R. Schöb, "Alleviating urban traffic congestion," *MIT Press Books*, vol. 1, 2005.

[2] D. Teodorović and P. Lučić, "Intelligent parking systems," *European Journal of Operational Research*, vol. 175, no. 3, pp. 1666–1681, 2006.

[3] G. Yan, W. Yang, D. B. Rawat, and S. Olariu, "Smartparking: a secure and intelligent parking system," *Intelligent Transportation Systems Magazine, IEEE*, vol. 3, no. 1, pp. 18–30, 2011.

[4] H. Wang and W. He, "A reservation-based smart parking system," in *Computer Communications Workshops (INFOCOM WKSHPS), 2011 IEEE Conference on*. IEEE, 2011, pp. 690–695.

[5] Y. Geng and C. G. Cassandras, "A new smart parking? system based on optimal resource allocation and reservations," in *Intelligent Transportation Systems (ITSC), 2011 14th International IEEE Conference on*. IEEE, 2011, pp. 979–984.

[6] S. Hashimoto, R. Kanamori, and T. Ito, "Auction-based parking reservation system with electricity trading," in *Business Informatics (CBI), 2013 IEEE 15th Conference on*. IEEE, 2013, pp. 33–40.

[7] M. Brook, C. Sharp, G. Ushaw, W. Blewitt, and G. Morgan, "Volatility management of high frequency trading environments," in *Business Informatics (CBI), 2013 IEEE 15th Conference on*. IEEE, 2013, pp. 101–108.

[8] N. Hanif, M. H. Badiozaman, and H. Daud, "Smart parking reservation system using short message services (sms)," in *Intelligent and Advanced Systems (ICIAS), 2010 International Conference on*. IEEE, 2010, pp. 1–5.

[9] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *Systems Science and Cybernetics, IEEE Transactions on*, vol. 4, no. 2, pp. 100–107, 1968.

[10] K.-H. C. Wang and A. Botea, "Fast and memory-efficient multi-agent pathfinding." in *ICAPS*, 2008, pp. 380–387.

[11] D. Silver, "Cooperative pathfinding," in *AI Game Programming Wisdom 3*, S. Rabin, Ed. Charles River Media, Inc., Massachusetts, 2006, ch. 2.1, pp. 99–112.

[12] M. E. Falou, M. Bouzid, and A.-I. Mouaddib, "Dec-a*: A decentralized multiagent pathfinding algorithm," in *IEEE 24th International Conference on Tools with Artificial Intelligence*, 2012.

[13] K.-H. C. Wang, "Tractable massively multi-agent pathfinding with solutions quality and completeness guarantees," in *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[14] G. Rger and M. Helmert, "Non-optimal multi-agent pathfinding is solved (since 1984)," in *SOCS*, 2012.

[15] R. C. H. Khorshid, Mokhtar M. and N. R. Sturtevant, "A polynomial-time algorithm for non-optimal multi-agent pathfinding," in *Fourth Annual Symposium on Combinatorial Search*, 2011.

[16] W. Blewitt, G. Ushaw, and G. Morgan, "Applicability of gpgpu computing to real-time ai solutions in games," 2013.

[17] G. Sharon, R. Stern, A. Felner, and N. Sturtevant, "Conflict-based search for optimal multi-agent path finding," in *Proceedings of the Fifth Annual Symposium on Combinatorial Search*, 2012.