Journal of Artificial Intelligence Research 65 (2019) 633-674 Submitted 03/2019; published 08/2019

Computing Multi-Modal Journey Plans under Uncertainty

Adi Botea ADIBOTEA@IE.IBM.COM
Akihiro Kishimoto AKIHIROK@IE.IBM.COM
IBM Research
Dublin, Ireland

Evdokia Nikolova NIKOLOVA@AUSTIN.UTEXAS.EDU
The University of Texas at Austin
Austin, TX, USA

Stefano Braghin STEFANOB@IE.IBM.COM
Michele Berlingerio MICHELE.BERLINGERIO@GMAIL.COM
Elizabeth Daly ELIZABETH.DALY@IE.IBM.COM
IBM Research
Dublin, Ireland

Abstract

Multi-modal journey planning, which allows multiple types of transport within a single trip, is becoming increasingly popular, due to a strong practical interest and an increasing availability of data. In real life, transport networks feature uncertainty. Yet, most approaches assume a deterministic environment, making plans more prone to failures such as missed connections and major delays in the arrival.

This paper presents an approach to computing optimal contingent plans in multi-modal journey planning. The problem is modeled as a search in an And/Or state space. We describe search enhancements used on top of the AO* algorithm. Enhancements include admissible heuristics, multiple types of pruning that preserve the completeness and the optimality, and a hybrid search approach with a deterministic and a nondeterministic search. We demonstrate an NP-hardness result, with the hardness stemming from the dynamically changing distributions of the travel time random variables. We perform a detailed empirical analysis on realistic transport networks from cities such as Montpellier, Rome and Dublin. The results demonstrate the effectiveness of our algorithmic contributions, and the benefits of contingent plans as compared to standard sequential plans, when the arrival and departure times of buses are characterized by uncertainty.

1. Introduction

Journey planning systems have become increasingly popular. Factors that favor the progress include the practical need among users for reliable journey plans, the ubiquity of mobile devices, and the increasing availability of relevant sensor data, such as bus GPS records. In its simplest form, journey planning is a shortest path problem: given a graph representation of a transportation network, find a shortest route between a start location and a destination. A more realistic modelling, such as considering uncertainty, can lead to problem variants that are computationally hard (Nikolova, Brand, & Karger, 2006; Nonner, 2012).

Multi-modal journey planning allows combining multiple transportation means, such as buses, trams and walking, into one single trip. Most existing approaches to multi-modal journey planning operate under deterministic assumptions (Zografos & Androutsopoulos, 2008; Liu & Meng, 2009). However, in real life, key information needed in journey planning, such as bus arrival times,

is characterised by uncertainty. Even small variations of bus arrival times can result in a missed connection, with a great negative impact on the actual arrival time at the destination. Journey plans should be able to handle such unexpected situations without compromising the plan quality (e.g., arrival time) beyond some acceptable level.

This paper investigates computing optimal contingent plans for multi-modal journey planning under uncertainty. Our contributions include the modeling of the problem, hardness results, a scalable optimal solver with novel algorithmic enhancements, and a detailed empirical evaluation.

The presented approach is probabilistic planning in an And/Or state space with continuous time, stochastic action durations, and probabilistic action effects. Uncertainty is modelled in the estimated times of arrival (ETAs) of scheduled-transport vehicles (e.g., buses) at stops, and the duration of actions such as walking and cycling. This leads to nondeterminism in the outcomes of actions such as boarding a scheduled-transport vehicle, which can either succeed or fail. The structure of the discrete And/Or search space is directly impacted by the time-related probability distributions defined in the model. For example, the likelihood that a user can catch a bus depends on the user's ETA, and the bus ETA at a given bus stop.

Contingent plans can be computed with the AO* (Nilsson, 1968, 1980) algorithm. To make it scalable in this domain, the AO* algorithm is combined with multiple search enhancements. Our heuristic to guide AO* uses two lookup tables with admissible estimates (i.e., not over-estimating the true value), one for the travel time and one for the number of legs in the journey. Furthermore, users can specify maximum acceptable amounts for the walking time, the cycling time, and the number of legs in a journey. States are pruned when the amount spent so far, plus a precomputed admissible estimate of the amount needed from here on, exceed the maximum acceptable amount. We perform state-dominance pruning. In nondeterministic planning, the correct application of our dominance pruning depends on the types of the nondeterministic branches on the path to each of the two states considered.

Given a branch from the initial state to a goal state in a contingent plan, we define the cost of the branch as a linear combination between the travel time and the number of legs (segments). The weight in the weighted sum is an input parameter. In this work we focus on a quality metric for contingent plans that minimizes the worst-case scenario (i.e., the worst cost across all branches), and breaks ties on the expected cost.

Besides the AO*-based search, we introduce a hybrid approach that combines A* and AO*. The initial A* deterministic search serves for two purposes. If the deterministic solution can be shown to be an optimal solution in the nondeterministic domain, the solving process completes before invoking a potentially more costly nondeterministic search. Otherwise, more accurate heuristic values are obtained as a back-propagation through the search tree of A*. In effect, the nondeterministic AO* search can use a more informed heuristic.

We present an NP-hardness result, with the hardness stemming from the *dynamically changing distributions* of the travel time random variables. This complements previous results for related problems with *static* distributions (Nikolova et al., 2006).

Experiments are performed in realistic multi-modal transportation networks from Montpellier, Rome and Dublin. Our empirical analysis focuses in a few directions: evaluate the speed and scalability of the planner, evaluate the impact of various features to the overall performance, and compare contingent plans to standard deterministic plans. The results show that our approach is effective, computing optimal plans fast. They also show that contingent plans can be more beneficial

than standard sequential plans, when the arrival and departure times of scheduled public transport vehicles are characterized by uncertainty.

We contrast deterministic plans and contingent plans in multi-modal journey planning in Section 2. Section 3 reviews related work. The input data used in our multi-modal journey planner is presented in Section 4. Searching for a plan is highlighted in Section 5, followed by a description of the And/Or state space in Section 6, and a definition of the cost of a contingent plan in Section 7. Section 8 presents an admissible heuristic for the travel time. The following section presents admissible heuristics for other metrics (e.g., the total number of legs in a trip), and their use in the system. Pruning with state dominance is addressed in Section 10. A hybrid approach that combines deterministic and nondeterministic search is the topic of Section 11. Simulating the execution of deterministic and contingent plans in the presence of uncertainty, is covered in the following section. A theoretical analysis comes next. An empirical evaluation, followed by conclusions and future work ideas, complete the paper.

Part of the material has been presented in short conference papers. Botea, Nikolova, and Berlingerio (2013) introduce an initial version of the system, with nondeterministic search and an initial set of search enhancements, such as heuristics and dominance pruning. Botea and Braghin (2015) present an empirical comparison of deterministic journey plans and contingent journey plans. Kishimoto, Botea, and Daly (2016) focus on the hybrid deterministic—nondeterministic approach. The current paper is extended in several directions. We introduce additional pruning rules, such as quota-based pruning. We provide improved pruning rules based on state dominance. We extend the way to compute the cost of a plan from an approach based solely on the travel time to any linear combination between the travel time and the number of legs in the trip. We provide additional experiments as compared to the three original papers, and also report significantly better results.

2. Contingent vs Sequential Journey Plans

In this section we illustrate potential differences between contingent and sequential journey plans in the presence of uncertainty. It is important to articulate potential differences upfront, to better emphasize the motivation to work on contingent journey planning. For simplicity, we consider just the arrival time as the plan quality metric, and discuss differences in terms of worst-case and best-case arrival times.

Consider the example illustrated in Figure 1, where a user wants to travel from A to B. As shown in Table 1, there are three ways to complete the journey. Each of these can be seen as a separate sequential plan. In plan 1, it is uncertain whether the connection at stop C will succeed, due to the variations in the arrival and departure times of routes 38 and 40, illustrated in Figure 1. Indeed, bus 38 arrives at C at 11:20$\pm$2 min, and the first bus 40 departs at 11:21$\pm$3 min.

The best policy is a combination of the sequential plans 1 and 3, as follows. Take bus 38 from A to C. If it's possible to catch bus 40 that departs at 11:21, take it from C to E, and then walk to B. Otherwise, walk to stop D, take bus 90 at 11:30, get off at stop F, and walk to B. The arrival time is $\sim$12:10pm (best case) or $\sim$12:20pm (worst case). This is illustrated in the last row of Table 2.

This policy is an example of a contingent plan (Peot & Smith, 1992). A contingent plan can be viewed as a tree of pathways from the origin to the destination. In this work, a contingent plan includes both *priorities* and *probabilities* associated with the options available in a state. In the example, the contingent plan contains a branching point at location C. The option of taking bus 40 that departs at around 11:21am has the highest priority. Walking to stop D is the backup,

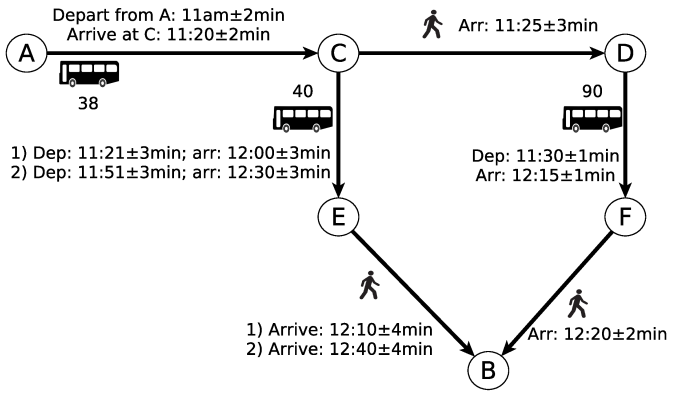

Figure 1: A toy example with bus links and walking links.

No	Sequential trajectory	Steps	Remarks
1	$A \to C \to E \to B$	11:00 – Take bus 38 to C 11:21 – Take bus 40 to E 12pm – Walk to B	 Uncertain connection in C If successful, arrive at around 12:10
2	$A \to C \to E \to B$	11:00 – Take bus 38 to C 11:51 – Take bus 40 to E 12:30 – Walk to B	 Arrive at around 12:40
3	$A \to C \to D \to F \to B$	11:00 – Take bus 38 to C 11:20 – Walk to D 11:30 – Take bus 90 to F 12:15 – Walk to B	 Arrive at around 12:20

Table 1: Sequential plans in the example.

lower-priority option. The probability p of following the first option can be computed from the distributions of two random variables, as shown later in this paper, in Proposition 1. One variable models the arrival time of the traveller at stop C, which in our example coincides with the arrival time of bus 38. The other variable is the departure time of bus 40. The probability of taking the backup option is $1 - p$.

Table 2 compares the performance of the sequential plans and the contingent plan. As sequential plans are restricted to one unique trajectory each, no sequential plan is capable of matching the performance of the best policy in the running example. In cases such as the one presented in the example, sequential plans lack the expressiveness needed to encode the best policy.

For instance, while plan 3 matches the worst-case arrival of the contingent plan, it is weaker than the contingent plan in terms of the best case arrival time. The sequential plan 2 is weaker in terms of both worst-case and best-case arrival times.

Plan 1 is weaker in terms of the worst-case arrival. But what is the worst-case arrival time of the sequential plan 1 anyways? To answer this question, we note that, in the case of a missed

Plan	Trajectory	Best-case arrival	Worst-case arrival
Sequential plan 1	$A \to C \to E \to B$	$\sim$**12:10**	$\sim$12:40
Sequential plan 2	$A \to C \to E \to B$	$\sim$12:40	$\sim$12:40
Sequential plan 3	$A \to C \to D \to F \to B$	$\sim$12:20	$\sim$**12:20**
Contingent plan	$A \to C \to E \to B$ $\downarrow$ $D \to F \to B$	$\sim$**12:10**	$\sim$**12:20**

Table 2: Comparing plans in terms of best-case and worst-case arrival times. Arrival times in bold fonts show the best performers on each column.

connection, sequential plans have the implicit backup option of waiting for the next trip on the same route. Hence, in the sequential plan 1, in the worst case, the user will wait for the next trip on route 40, which arrives at around 11:51. As this implicit backup option of plan 1 is essentially equivalent to plan 2, we conclude that the worst-case arrival time for plan 1 is about 12:40pm.

More generally, a sequential plan can be optimal if it is both *safe* (i.e., all actions in the sequence will be applicable with probability 1.0) and *fast* (i.e., the arrival time is optimal). On the other hand, as shown in the example, there can exist states where one option is good (i.e., providing a good arrival time) but uncertain, whereas another option is safe but slower. These are cases where sequential plans lose their ability to implement an optimal policy, and where contingent plans can make a difference, as illustrated in our example.

3. Related Work

In a problem related to uni-modal transport, such as driving in a road network, Nikolova et al. (2006) consider stochastic costs for edges, modelled with random variables. More such related models that integrate risk-aversion have been considered by (Loui, 1983; Nikolova, Kelner, Brand, & Mitzenmacher, 2006; Nikolova, 2010; Fan, Kalaba, & J.E. Moore, 2005; Nikolova & Karger, 2008). Our notion of robust plans is somewhat related to robust optimization (Bertsimas, Brown, & Caramanis, 2011) and percentile optimization in Markov Decision Processes (Delage & Mannor, 2010). Wellman, Larson, Ford, and Wurman (1995) investigate route planning under time-dependent edge traversal costs. They take a dynamic programming approach, using a notion of stochastic dominance to prune away dominated partial paths.

In multi-modal journey planning, the robustness to delays has been one topic of interest. Bast et al. (2016) provide a good survey. Several approaches rely on receiving dynamic updates and using accurate information at the time when computing a plan.

For example, Bast, Sternisko, and Storandt (2013) have studied the robustness of precomputed data used to speed up the journey plan computation, when the status of the network changes between the preprocessing time and the time when a plan should be computed. Their approach is deterministic. Changes in the network can occur after the precomputation, but nevertheless, at the planning time, the knowledge about the network is assumed to be accurate. In contrast, in our non-deterministic approach, we assume imperfect knowledge about the network at the planning time.

Dibbelt, Pajor, Strasser, and Wagner (2013) present an approach to multi-modal journey planning based on an indexing of so-called *connections*. A connection is an atomic segment of a trip,

connecting two consecutive stops along a route. A connection is characterized by its starting stop, starting time, arrival stop and arrival time. In deterministic planning, journey plans are computed with a technique that performs a linear scan of a datastructure where connections are ordered by the departure time. In addition, this work presents an extension to planning under uncertainty, where uncertainty models the fact that connections can have delays in their arrival. Trips are optimized to the minimum expected arrival time. We optimize across multiple criteria, such as the expected arrival time, and the worst-case arrival time, with ties broken on the expected arrival time. Another difference is that we allow to define the cost as a linear combination of two factors, such as the travel time and the number of interchanges.

Contingent planning (Peot & Smith, 1992) is planning under uncertainty with some sensing capabilities. Uncertainty stems from a partially known initial state And/Or nondeterministic action effects. Approaches to contingent planning include searching in a space of belief states (Bonet & Geffner, 2000a; Hoffmann & Brafman, 2005), and compilation from a belief state space to fully observable states (Albore, Palacios, & Geffner, 2009). Our planner searches in an And/Or space of explicit states.

Pruning based on state dominance has been applied to problems such as robot navigation (Mills-Tettey, Stentz, & Dias, 2006), pallet loading (Bhattacharya, Roy, & Bhattacharya, 1998) and test-pattern generation for combinational circuits (Fujino & Fujiwara, 1993). For example, in robot navigation (Mills-Tettey et al., 2006), both a robot's position and its battery level are part of a state. This bears some similarity with our definition of dominance. The difference is in the "resources" considered, such as the battery level versus the (stochastic) time, and the quotas for walking, cycling and vehicle inter-changes.

Next we overview research in nondeterministic planning. Then, we summarize related ideas from previous research in deterministic search.

3.1 Nondeterministic Planning

Variations of planning under uncertainty include conformant planning (Goldman & Boddy, 1996; Smith & Weld, 1998; Cimatti & Roveri, 2000; Palacios & Geffner, 2009), contingent planning (Peot & Smith, 1992), and probabilistic contingent planning, with differences stemming from the assumptions on the properties of the actions and the observability (sensing capabilities). See e.g., Bonet and Geffner's (2000b) discussion.

In conformant planning, actions can be nondeterministic, the initial state is not fully known, and no sensing is available when executing the plan. Thus, a conformant plan should guarantee reaching the goal under such conditions. In contingent planning, sensing allows to detect the resulting state when a nondeterministic action is applied. Plans can have a tree structure, to ensure that the goal will be reached from any potential nondeterministic successor. Probabilistic contingent planning allows probabilistic actions and sensing. As Bonet and Geffner (2000b) remark, probabilistic contingent planning can generally be modeled as Partially Observable Markov Decision Processes (POMDPs). However, when full observability (full sensing) is assumed, we fall into the framework of MDPs.

While approaches such as AO* and dynamic programming can return optimal plans (policies), computing optimal plans is challenging due to the potentially many states that need to be processed during planning. For instance, decision-theoretic algorithms based on dynamic programming can suffer from large memory requirements, that can be exponential in the domain feature size (Barto, Bradtke, & Singh, 1995). Adding heuristic enhancements to AO*, such as an informed admissible

heuristic or effective pruning rules can reduce the size of the state space explored in a search for an optimal plan, which is the approach taken by Botea et al. (2013). Hansen and Zilberstein (2001) present LAO*, an optimal algorithm based on AO* that is capable of finding solutions with loops, and apply LAO* to solve MDPs. We note that the search space of our multi-modal journey planning has no loops, one main reason being that a state includes the time among its components.

Due to the computational difficulty of optimal nondeterministic planning, researchers have often preferred suboptimal approaches, for a better speed and scalability. Hoffmann and Brafman (2005) use Weighted AO*, a faster and suboptimal variant of AO*. Bonet and Geffner (2000b, 2003, 2005) take an approach based on a greedy, real-time action selection. One of their heuristics, called the min-min state relaxation, is related to our approach of using the results of a deterministic relaxation to obtain an admissible heuristic in the nondeterministic domain.

Some approaches rely on using deterministic planning as part of solving a nondeterministic planning problem (Kurien, Nayak, & Smith, 2002; Kuter & Nau, 2004; Yoon, Fern, & Givan, 2007; Kuter, Nau, Reisner, & Goldman, 2008; Kolobov, Mausam, & Weld, 2009; Teichteil-Königsbuch, Cedex, & Kuter, 2010; Muise, McIlraith, & Beck, 2012; Nguyen, Tran, Son, & Pontelli, 2012). For example, Kuter et al. (2008) run multiple deterministic planning rounds, with a deterministic relaxation of the original domain, to gradually add branches to a contingent plan under construction. FF-Replan (Yoon et al., 2007) invokes a deterministic planner for the initial state and for the states in the plan where unexpected outcomes are observed in the nondeterministic scenario. Another example where relaxing uncertainty by optimistically assuming that the best outcome will be available is work in the Canadian Traveler's Problem, where optimistic roadmaps assume that all roads will always be accessible (Eyerich, Keller, & Helmert, 2010).

Thus, the high-level idea of using deterministic planning as part of the process to solve a nondeterministic planning problem appears in both previous work and our work. There are important differences, however, stemming in part from the fact that we ensure solution optimality, whereas such previous work does not. We run exactly one deterministic search. If the result is a correct and optimal nondeterministic plan, we are done. Otherwise, we improve the available heuristic function with the back-propagation and run full-scale AO*. To our knowledge, our approach is the first system to combine deterministic and nondeterministic search in this way with evidence that this approach solves realistic, difficult journey planning problems fast and optimally.

3.2 Deterministic Search

Using a deterministic relaxation of a nondeterministic problem to compute an admissible heuristic can be used as a form of domain abstraction. Using state-space abstractions to build an admissible heuristic is broadly used in planning and heuristic search. See e.g., Hierarchical A* (HA*) (Holte, Perez, Zimmer, & MacDonald, 1995), pattern databases (Culberson & Schaeffer, 1998), and merge-and-shrink abstractions (Helmert, Haslum, & Hoffmann, 2007; Helmert, Haslum, Hoffmann, & Nissim, 2014). Among these, HA* is more closely related to our method. HA* builds a hierarchy of abstractions, with the lowest level being the original state space. A* search in one abstract space can be used to build an admissible heuristic for the lower abstraction level. HA* features caching techniques to reduce duplicate search effort across multiple levels in the hierarchy.

We incorporate Holte et al.'s (1995) P-g heuristic. In HA*, P-g is the heuristic $h(n) = P - g(n)$, where P is an optimal solution cost in the next abstraction level, and $g(n)$ is the g-cost in the next abstraction level of the abstract node corresponding to n. This has been shown to be an admissible

heuristic (Holte et al., 1995). Indeed, given a node n, we have $g(n) + h^*(n) \geq P$, where $h^*(n)$ is the perfect heuristic. It follows that $P - g(n) \leq h^*(n)$ which means that the heuristic is admissible.

There are multiple notable differences between our approach and HA*. First, we consider domains with multiple types of "consumable resources", such as the quotas for the maximum walking time and the maximum number of legs in a trip. This makes state dominance an important pruning mechanism. Ensuring the correctness and optimality of a hybrid A*–AO* approach, especially in the presence of state dominance, is a non-trivial contribution.

Secondly, unlike HA*, our approach backpropagates information through the graph search of A*, for a stronger improvement of the heuristic. Furthermore, our approach runs at most two searches, one deterministic and one nondeterministic. That is, our approach defines exactly two "hierarchical levels", and employs only one search in the "abstracted" (i.e., deterministic) state space. The number of searches can be larger in HA*, due to a potentially larger number of levels, as well as potentially multiple searches at a given abstraction level. In HA* all search spaces are deterministic, whereas we report a contingent planning system.

Incremental heuristic search aims at efficiently reusing previous search results to solve "similar" new problems. It has mainly been applied to real-time path-planning with dynamic domains. There are two common principles for reusing information in incremental search. One is to start a new A* search with the open and closed lists adapted from the previous A* search (Stentz, 1995; Koenig, Likhachev, & Furcy, 2004). The idea is effective when small changes are observed in the problem from one search to another. On the other hand, in our setting, differences between deterministic and nondeterministic search are more fundamental. In particular, the two types of problems require very different search algorithms, such as A* and AO*. The second common principle in incremental search is to make a heuristic more informed based on previous search results (Hernández, Meseguer, Sun, & Koenig, 2009; Hernández, Sun, Koenig, & Meseguer, 2011). Caching techniques used for this purpose are similar to those featured in HA*, reviewed earlier.

4. Defining the Input

A multi-modal journey planing task is determined by a user query (i.e., a request to compute a journey plan), and the status of a multi-modal transport network. This section contains formal definitions of these. In this work we focus on transportation modes (means) such as scheduled-transport modes (e.g., buses, trains and trams), bicycles available in a city's bike sharing network, and walking. Our system can also handle private cars, shared rides in a car and taxi rides, but these are beyond our focus in this paper.

A *user query* is a tuple $\langle o, d, t_0, m, q \rangle$, where: o and d are the origin and the destination locations; t_0 is the start time; m specifies a subset of transportation means that should be considered; and q states maximal values (quotas) to parameters such as the *expected walking time*, the *expected cycling time*, and the *number of legs* in a trip.

A *multi-modal network snapshot* is a structure $\langle \mathcal{L}, \mathcal{R}, \mathcal{T}, \mathcal{W}, \mathcal{C} \rangle$ that encodes (possibly uncertain) knowledge about the current status and the predicted status in time of a transport network. $\mathcal{L}$ is a set of *relevant locations* on the map, such as the stops for scheduled transport (e.g., bus stops) and the bike stations. In addition, given a user query, the origin and the destination are added to $\mathcal{L}$ as relevant locations, unless already present. $\mathcal{R}$ is a set of *routes*. A route r is an ordered sequence of $n(r)$ locations, corresponding to the stops along the route. For a given route label in real life, such as "Route 39", there can be multiple route objects in set $\mathcal{R}$. For example, there is one distinct route

object for each direction of travel (e.g., eastbound and westbound). Also, variations across different times of the day, or different days of the week, if any, result in different route objects.

$\mathcal{T}$ is a set of *trips*. Informally, each trip is one individual scheduled-transport vehicle (e.g., a bus) going along a route (e.g., the bus that starts on route 39 at 4:30pm, going westbound). Formally, a trip i is a structure $\langle r, f_{i,1}, \ldots f_{i,n(r)} \rangle$, where r is the id of its route, and each $f_{i,k}$ is a probability distribution representing the estimated arrival time at the k-th stop along the route.

$\mathcal{W}$ and $\mathcal{C}$ provide walking times and cycling times, for pairs of locations, as probability distributions.

The part of the snapshot structure encoding the public transport data is similar to the GTFS format[1]. GTFS, however, handles no stochastic data. Furthermore, our snapshots cover additional transport modes, such as shared-bike data.

The way the network snapshot is defined is partly justified by practical reasons. It encodes available knowledge about the current status and about the predicted evolution over a given time horizon (e.g., estimated bus arrival times). At one extreme, the snapshot can use static knowledge, such as static bus schedules, or static knowledge based on historical data. At the other extreme, increasingly available sensor data can allow to adjust the predicted bus arrival times frequently, in real time (Bouillet, Gasparini, & Verscheure, 2011). Our network snapshot definition is suitable for both types of scenarios, allowing to adjust the approach to the level of accuracy available in the input data.

5. Searching for a Plan

In this paper we present two solving approaches. The first one is based on AO* (Nilsson, 1968, 1980) search. The second is a hybrid approach combining deterministic A* search with nondeterministic AO* search.

AO* is a best-first algorithm for And/Or search spaces. The algorithm maintains a partial best solution (e.g., a tree), initialized to the root node. As long as the current partial best solution has unexpanded tip nodes (i.e., leaf nodes in the current partial best solution that are not goal states), the current partial best solution is expanded, by picking an unexpanded tip node and expanding it. This can trigger changes in the current cost estimations of the expanded node, and its ancestors. As a further effect, the current partial best solution can also change (e.g., if a cost increase of a node makes it look less promising than some sibling node). The process continues until a complete solution is found. With an admissible heuristic[2] in use (to estimate the costs of tip nodes), AO* finds an optimal solution.

The state space explored in AO* search is presented in Section 6. The search algorithms use multiple enhancements to improve their speed and scalability. Admissible heuristics are discussed in Sections 8 and 9. Pruning rules are discussed in Sections 9 and 10. The hybrid method is presented in Section 11.

Both types of search explore their corresponding state spaces as a tree (as opposed to a graph with multiple pathways to a given state). Part of the reason is that in some search enhancements discussed in this paper, such as pruning based on state dominance, the path from the initial state to a given state is relevant.

1. `https://developers.google.com/transit/gtfs/reference`
2. A heuristic is admissible if the estimation it provides for a node does not exceed the real cost of that node.

6. And/Or State Space

Our journey planning performs AO* search. In this section we define the And/Or state space explored in the search. We call this the nondeterministic state space, or **ND**.

The search space is an And/Or state space with continuous time, stochastic action durations, and probabilistic action effects. States and transitions between states are discrete, just as in a typical And/Or space. The probabilities associated with the nondeterministic effects of an action depend on stochastic time parameters, such as the estimated time of arrival (ETA) of a trip (e.g., a bus) and a user's ETA at a (bus) stop. In this section we describe the And/Or state space, with a focus on how the continuous, stochastic modelling of time-related parameters affects the structure of the state space.

The core components of a state s include a position p_s, a density function t_s,[3] and a vector of numeric variables q_s. Some auxiliary state components are used for correctness (e.g., to define what types of actions apply in which states) and for pruning. In this section, we focus on the core components, and particularly the time distribution t_s.

The position $p_s \in \mathcal{L} \cup \mathcal{T}$, representing the position of the user in the state at hand, can be either a relevant location on the map or a trip id, in which case the user is aboard that trip. The time t_s is a probability distribution representing the uncertain time when the user has reached position p_s. Quotas left in this state (e.g., for walking time, cycling time, number of legs) are available in q_s. In the initial state, the quotas are taken from the user query. They get updated correspondingly as the search progresses along an exploration path.

Actions considered are `TakeTrip`, `GetOffTrip`, `Cycle`, and `Walk`. `TakeTrip` actions can have up to two nondeterministic outcomes (success or failure), as discussed below. Other actions always succeed and therefore they always are deterministic actions (i.e., applying such an action results in exactly one successor state). The probabilities associated with nondeterministic branches (effects of an action) are determined dynamically, when the transition is generated, based on one or two time-related probability distributions, as shown later in this section. Waiting for a trip to arrive is implicitly considered, being the time between the user's arrival at a stop and the time of boarding a trip.

We now describe the transitions, their probabilities and the successors that are generated with a `TakeTrip` action applied in a state s with the location p_s being a bus stop. Let i be a trip that stops at p_s, the k-th stop along the route. The ability to catch trip i depends on the arrival time of the user at p_s, and the arrival time of the trip. Recall that t_s, available from s, models the user's arrival time at location p_s, whereas $f_{i,k}$, available as input data from the network model, represents the arrival time of the trip. Let U be a random variable corresponding to t_s, and B a random variable corresponding to $f_{i,k}$.

Proposition 1. *Under the assumption that U and B are independent, the probability of being able to catch the trip is*

$$P(U < B) = \int_{-\infty}^{\infty} f_{i,k}(y) \int_{-\infty}^{y} t_s(x)dxdy.$$

Proof. Fix a value y for the trip arrival time. The probability that the user makes it before y is $P(U < y) = \int_{-\infty}^{y} t_s(x)dx$. Generalizing to all values y across the range, we obtain the desired formula. □

3. For clarity, we stick with the case of continuous random variables. Discrete distributions are handled very similarly.

In state s, $P(U < B) > 0$ is a precondition for the action of boarding trip i. When $0 < P(U < B) < 1$, there are two nondeterministic effects, corresponding to success and failure, each outcome having probability $P(U < B)$ and $1 - P(U < B)$ respectively. When $P(U < B) = 1$, only the successful branch is defined. In other words, when $P(U < B) = 1$, the `TakeTrip` action at hand is deterministic.

In the successor state s' along the successful branch, the time of boarding the trip is a random variable corresponding to the estimated arrival time of the trip, conditioned by the fact that the user makes it to the stop before the trip. Using a conditional probability in this case, as opposed to simply the trip's estimated arrival time, is preferable to avoid negative waiting times.

Proposition 2. *The density function $t_{s'}$, corresponding to the time of boarding the trip, is*

$$t_{s'}(y) = \frac{f_{i,k}(y) \int_{-\infty}^{y} t_s(x)dx}{\int_{-\infty}^{\infty} f_{i,k}(y) \int_{-\infty}^{y} t_s(x)dxdy}.$$

Proof. The boarding time is a random variable Z defined by the trip's arrival time B, conditioned by $U < B$. Applying the Bayes rule to the continuous variable B and the discrete binary variable $U < B$, we obtain: $t_{s'}(y) = \frac{P(U<B|B=y)f_{i,k}(y)}{P(U<B)} = \frac{f_{i,k}(y) \int_{-\infty}^{y} t_s(x)dx}{\int_{-\infty}^{\infty} f_{i,k}(y) \int_{-\infty}^{y} t_s(x)dxdy}$ □

Corollary 1. *If $P(U < B) = 1$, then $t_{s'} = f_{i,k}$.*

That is, when the user certainly makes it to the stop before the trip (bus), then the distribution of the boarding time is the same as the distribution of the trip arrival time.

Denote by s' and s'' the two successor states corresponding to the two possible outcomes of a `TakeTrip` action: the user boards the trip and the user misses the trip respectively.

The locations in the two successor states are $p_{s'} = i$ (i.e., the user is aboard the trip) and $p_{s''} = p_s$ (the user is still at the stop). Auxiliary state variables are used to forbid multiple (failed) attempts to board the same trip[4] at location p_s, either in a sequence or interleaved with failed attempts to board other trips. This is implemented in a straightforward way, in the style of a STRIPS encoding, and the exact details are beyond the scope of this paper.

On the failed branch, the time of reaching the successor state s'' is a random variable with the same distribution as t_s, corresponding to the user's time of arrival at p_s.

When aboard a trip i, `GetOffTrip` actions are defined for all stops along the route subsequent to the boarding stop. When applying a `GetOffTrip` action, to get off at the k-th stop along the route of trip i, the time associated with the successor state s' is the same as the estimated arrival time of the trip: $t_{s'} = f_{i,k}$.

In the case of `Cycle` and `Walk` actions applied in a state s, the time of reaching the destination depends on both the starting time, represented by the density function t_s, and the duration of the action (with a given density function g). Cycling (or walking) time is assumed to be independent of the starting time. Thus, the density function of reaching the destination (i.e., the time distribution of the successor state s') is the convolution of the two densities t_s and g: $t_{s'}(x) = (t_s * g)(x) = \int_{-\infty}^{\infty} t_s(y)g(x - y)dy$.

In summary, a continuous, stochastic time modelling has a direct impact on the discrete structure of the And/Or space, influencing the applicability of actions, the number of (nondeterministic) branches of an action, and their probabilities. In addition, plan quality metrics are impacted, such as the arrival time. Plan quality is discussed in the next section.

4. Of course, taking a later trip on the same route is allowed.

7. Contingent Journey Plans and Plan Cost Function

Definition 1. *A* contingent plan *in multi-modal journey planning is a collection of states and actions (as defined in Section 6). One node corresponds to the initial state of the problem instance. One or more nodes, called the goal nodes, correspond to the destination location. Each non-goal node (state) has exactly one outgoing action (deterministic or nondeterministic) that is applicable in that state.*

Given a transition from a state s_1 to a state s_2 in **ND**, we assume the transition has a non-negative cost $c(s_1, s_2)$. In this work, possible types of costs considered for an individual transition include the travel time, the number of legs, the cycling time and the walking time. For the purpose of computing the cost of a plan (i.e., the topic of this section), we consider the following types of transition costs: the travel time, the number of journey legs, and any linear combination of these. For other purposes, such as observing the quotas and pruning the state space with quota estimates, we use the walking time, the cycling time and the number of legs, as discussed in Section 9.

Based on the costs of individual transitions, the optimal expected cost $v_{exp}(s_1)$ in **ND**, and the optimal worst-case cost $v_{wst}(s_1)$ in **ND** of a state s_1 are defined as:

1. If s_1 is a goal state, $v_{exp}(s_1) = v_{wst}(s_1) = 0$.
2. If s_1 is a dead end state (i.e., a non-goal state with zero successors), $v_{exp}(s_1) = v_{wst}(s_1) = \infty$.
3. Otherwise,
   - $v_{exp}(s_1) = \min\limits_{a \in A_{nd}(s_1)} \sum\limits_{b \in B(s_1,a)} p_b(c_b(s_1, a) + v_{exp}(\gamma_b(s_1, a)))$
   - $v_{wst}(s_1) = \min\limits_{a \in A_{nd}(s_1)} \max\limits_{b \in B(s_1,a)} (c_b(s_1, a) + v_{wst}(\gamma_b(s_1, a)))$.

Given a state s_1, $A_{nd}(s_1)$ is the set of outgoing actions in **ND**. Given such an action a, $B(s_1, a)$ is the number of branches of that action. For a deterministic action, $|B(s_1, a)| = 1$. For a nondeterministic action, $|B(s_1, a)| = 2$. Given a branch b, p_b is its corresponding probability and $c_b(s_1, a)$ is its cost.[5] The transition function $\gamma_b(s_1, a)$ provides the successor state of the transition at hand.

As our objective function for the quality of a plan, we consider a 2-element tuple (v_{wst}, v_{exp}) in the lexicographic order, i.e., minimize the worst case, and break ties in favor of better expected costs. We write $(p_1, p_2) \leq_{lex} (q_1, q_2)$ iff $(p_1 < q_1) \vee (p_1 = q_1 \wedge p_2 \leq q_2)$. Unless otherwise mentioned, this is the objective function used in this work. However, in the experiments we include the evaluation of an additional metric where we swap the order of the two elements in the lexicographic order (i.e., minimize the expected cost, and break ties on the worst case).

8. Admissible Heuristics for the Travel Time

Given a state s, a heuristic for the travel time is an estimation of the actual travel time from s to the destination. As previously mentioned, a heuristic is *admissible* if, in every state, the heuristic value

5. The cost can depend on both action a and state s_1, not just on action a. This allows for instance to integrate waiting into the cost of a transition from "arrived at a stop" to "boarded a bus".

does not exceed the actual cost of that state. When the cost is the shortest travel time, an admissible heuristic should not overestimate the shortest possible travel time from that state to the destination.

Given a destination location, an admissible heuristic is pre-computed in low-polynomial time and stored as a look-up table. For every location (stops, bike stations, origin), the table stores an admissible estimation of the travel time from that location to the destination. The relaxation considered in the computation of the heuristic ignores waiting times, slow-downs along a route segment, any actual quotas left in a state, and the possibility that some connections may depart before the user arrives at the connection point.

The heuristic look-up table is built with a regression run of the Dijkstra algorithm, in a *relaxed graph* obtained as follows. Each location is a node. There are multiple kinds of edges, each corresponding to one transport mode, such as taking a trip, cycling, and walking. Two consecutive stops along a route are connected with an edge whose cost is a lower bound for the travel time of a trip along that segment. If the shortest walking time between two given locations is smaller than a threshold T_W given as a parameter, an edge is added between the two locations. The cost is a lower bound on the walking time between the two locations. Edges corresponding to cycling are created similarly, with a different threshold parameter T_C.

Observation 1. *Let q_w and q_c be the maximal walking time, and the maximal cycling time set in a given user query. If $T_W \geq q_w$ and $T_C \geq q_c$, then the heuristic computed as above is admissible for the instance at hand.*

Consider the following example where a user needs to travel from A to C, departing at 11am. There is a train from A to B, typically departing at 11am, and arriving at 12am. A train from B to C typically departs at 11:50, and arrives at 12:50. According to our uncertainty model, the best possible duration from A to B is 50 minutes, and the best possible duration from B to C is 55 minutes. When computing the heuristic with Dijkstra search, the link $A \rightarrow B$ contributes with 50 minutess, and the link $B \rightarrow C$ with 55 minutess. The heuristic travel time from A to C is $50 + 55 = 105$ minutes. In a Dijkstra search for heuristic computation, nodes store location information, but not a specific time of the day. Since Dijkstra nodes do not store a time of the day, the heuristic computation ignores the fact that the second link may depart before the first link arrives.

The accuracy of the heuristic described earlier can be improved by limiting the set of edges considered in the relaxed graph to a relevant subset. In the previous example, further consider that an express route goes from A to C in a very short time, say 80 minutes. Assume further that the express route runs only once per day, at 6:00pm. If this direct connection is included as an edge in the relaxed graph used for the computation of the heuristic in our instance (with the starting time set to 11:00am), the heuristic will be too optimistic and its accuracy will suffer. This is why we allow to specify a temporal window and add to the relaxed graph only edges that can occur within the temporal window at hand. The temporal window starts at the departure time of the trip, and its duration is an upper bound on the maximal acceptable duration of a trip in a given city. This maintains the admissibility of the heuristic, and helps improve its accuracy.

In the hybrid solving approach that combines deterministic A* with nondeterministic AO*, discussed in Section 11, the heuristic used in the AO* search can further be improved based on the initial deterministic A* search. See Section 11 for details.

9. Admissible Quota Heuristics and Their Use in Pruning

Recall that, when posing a query to the planning system, the user can specify quotas such as a maximum walking time q_W, a maximum cycling time q_C, and a maximum number of legs q_L.

Pruning with admissible quota estimates is a powerful pruning technique that preserves the completeness and the optimality of solutions. As the name suggests, the method requires the availability of admissible heuristic estimations of the number of legs, the walking time and the cycling time needed to reach the destination from a given location. Given a relevant location l, $h_L(l)$ is a lower bound on the minimum number of legs required to reach the destination from l, regardless of the pathway from l to the destination. The heuristics h_W and h_C are defined similarly, for the walking time and for the cycling time respectively.

Pruning with admissible quota estimates works as follows. Consider a state s in search and let p_s be the position of that search. Reaching s from the initial state (root node of the search) requires $g_L(s) = 3$ legs (e.g., walk, bus ride, walk). According to the admissible heuristic for the number of legs, reaching the destination from p_s requires at least $h_L[p_s] = 3$ more legs, regardless of the route being followed from n to the destination. Overall, the total number of legs in a plan containing node s would be at least $g_L(s) + h_L[p_s] = 3 + 3 = 6$. On the other hand, assume that the user imposed a constraint of at most 5 legs in the journey. Clearly, state s can be pruned from the search, since it is impossible to reach the destination without violating this constraint. Similar pruning rules are implemented for the walking time and the cycling.

The three heuristics are implemented as look-up tables, similarly to the travel-time heuristic look-up table presented in Section 8. Recall that the travel-time heuristic is built with a Dijkstra search in a relaxed graph. The three new heuristic tables are computed similarly, with one Dijkstra search each in the same relaxed graph. The difference is that each Dijkstra search uses as the edge cost the metric at hand (i.e., the number of legs, the walking time and the cycling time respectively).

10. Pruning with State Dominance

State dominance (Horowitz & Sahni, 1978) can help pruning a search space while maintaining the completeness and the solution optimality. The idea is to identify pairs of states (nodes) with the property that expanding one of them (the "dominating" state) is sufficient.

In journey planning, assume that the same location l can be reached in two different ways, resulting in two distinct states (nodes) in the search tree. Recall that a state s is a tuple cotaining among its components a position p_s, a stochastic time t_s when the traveller arrives at p_s, and a tuple of quotas q_s (walking time, cycling time, number of legs), being the quotas that can be used to reach the destination from state s.

Definition 2. *We say that a state s dominates a state s', and write $s \prec s'$, iff $p_s = p_{s'}$, $q_s \geq q_{s'}$ component-wise, and $P(T_s \leq T_{s'}) = 1$.*[6]

In other words, reaching s from the origin does not require a longer time than reaching s', and it does not require larger quotas either.

Detecting pairs of states in a dominance relation is generally slower than duplicate detection (i.e., detecting pairs of identical states), as the latter can be implemented efficiently with hash codes, such as Zobrist hashing (Zobrist, 1970), and transposition tables. Thus, an efficient (i.e., fast)

6. T_s is the random variable with the density function t_s.

implementation of a full-scale dominance check, comparing every new state against previously visited states, is not trivial. Here we present an effective approach to partial dominance checking.

Given a state s, let its *subtree cost* $\text{sc}(s) = (v_{wst}(s), v_{exp}(s))$ be the optimal cost of a plan rooted at s (i.e., plan to an instance where the starting time, starting location and quotas are taken from s).

Recall that, in our modeling, part of the actions are deterministic and part of them are nondeterministic. A nondeterministic action has two possible branches (outcomes), one being successful and the other one being the failure outcome. Let $\xrightarrow{d}$ denote a deterministic transition. Similarly, $\xrightarrow{s}$ denotes a transition on a successful nondeterministic branch, and $\xrightarrow{f}$ a transition on a failed nondeterministic branch. In particular, successful branches include deterministic branches. Symbols $\overset{d}{\rightsquigarrow}$, $\overset{s}{\rightsquigarrow}$, and $\overset{f}{\rightsquigarrow}$ denote sequences of one or more transitions of a given type.

Observation 2. *Let a be an action with two nondeterministic outcomes in a valid[7] plan π. Removing action a from π, together with the entire subtree along the successful branch, results in a valid plan.*

We illustrate this observation in Figure 2.

Theorem 1. *Let $s \xrightarrow{s} s'$ and $s \xrightarrow{f} s''$ be two branches of an action a in an optimal plan. Then $sc(s') \leq sc(s'')$.*

Proof. The intuition is that, if $\text{sc}(s') > \text{sc}(s'')$, then removing action a and its subtree under the successful branch will result in a strictly better plan.

Let us denote $\text{sc}(s')$ as $c' = (c'_{wst}, c'_{exp})$ and $\text{sc}(s'')$ as $c'' = (c''_{wst}, c''_{exp})$. Assume $\text{sc}(s') > \text{sc}(s'')$ (in other words, $c' > c''$).

Case $c'_{wst} > c''_{wst}$: Eliminating action a from π reduces the first (main) cost component from c'_{wst} to c''_{wst}, thus obtaining a valid plan with a better cost. This contradicts the optimality of π.

Case $c'_{wst} = c''_{wst} \wedge c'_{exp} > c''_{exp}$: Removing action a has no impact on the main cost component, since $c'_{wst} = c''_{wst}$. Focusing on the tie-breaking component, let p' and p'' be the probabilities associated with the two outcomes of action a. Let π_2 be the plan after removing a. From the optimality of π, we have $c_{exp}(\pi) \leq c_{exp}(\pi_2) \Rightarrow p'c'_{exp} + p''c''_{exp} \leq c''_{exp} \Leftrightarrow p'c'_{exp} \leq (1-p'')c''_{exp}$. As $c'_{exp} > c''_{exp}$, we obtain $p'c''_{exp} < (1-p'')c''_{exp}$, which leads to $p' + p'' < 1$. This is a contradiction, as $p' + p'' = 1$. □

In previous work (Botea et al., 2013) we have proven the following result:

Theorem 2. *Let s_1, s_2 and s_3 be three states such that $s_1 \xrightarrow{d} s_2$ and $s_1 \overset{s}{\rightsquigarrow} s_3$. If $s_2 \prec s_3$ (i.e., s_2 dominates s_3), then assigning an ∞ score to s_3 does not impact the correctness, the completeness and the optimality of a tree search.*

The previous result is limited to pairs of states (s_2, s_3) with the property that the parent of s_2 is also an ancestor of s_3. The following new result relaxes that constraint, extending the applicability of pruning with state dominance.

7. We say that a plan is valid if it complies with Definition 1.

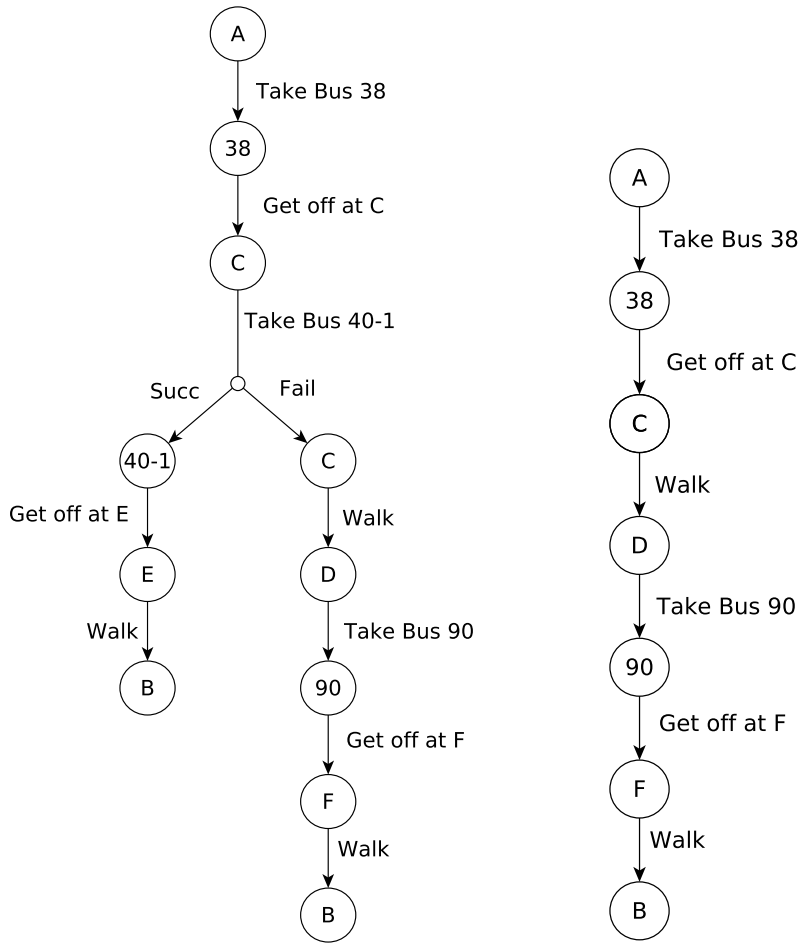

Figure 2: Part (a) shows the contingent plan illustrated more informally in Table 2, bottom row. For an easier readability, we adapt slightly action names (e.g., "Take Bus" instead of `TakeTrip`, "Get off at" instead of `GetOffTrip`), and show just the location information in the states. Part (b) shows the resulting plan after removing the nondeterministic action "Take Bus 40-1", together with the subtree under the successful branch. This is equivalent to the sequential plan 3 shown in Table 2.

Theorem 3. *Let s_1 be a state obtained on a path of deterministic transitions, and let s_2 be a state obtained on a path of successful transitions. Assume that the cost is a linear combination of one or several elements comprising the travel time and the quotas. If $s_1 \prec s_2$, then marking s_2 as a dead end impacts neither the completeness nor the optimality of a search.*

Proof. Let c_1 be the cost of an optimal plan π_1 containing $s_0 \overset{d}{\rightsquigarrow} s_1$, and c_2 the cost of an optimal plan π_2 containing $s_0 \overset{s}{\rightsquigarrow} s_2$. We show that $c_1 \leq c_2$.

According to Theorem 1, $c_2' \leq c_2$, where c_2' is the combined cost of $s_0 \overset{s}{\rightsquigarrow} s_2$ together with the subtree of s_2.

Also, $c_1 \leq c'_2$, since (1) path $s_0 \overset{d}{\rightsquigarrow} s_1$ has a smaller cost than path $s_0 \overset{s}{\rightsquigarrow} s_2$, according to the domination relation and the definition of the cost; and (2) more resources are left available in state s_1, allowing to obtain a solution subtree under s_1 at least as good as the one under s_2.

It follows that $c_1 \leq c_2$, which completes the proof.

□

Applying state dominance to a nondeterministic domain is more involved than in the case of deterministic domains. As shown in the conditions of the theorems presented, in a nondeterministic domain, we need to be careful about the types of transitions that lead to the two states at hand, s and s'. For example, if the pathway from the initial state s_0 to the state s' contains failed transitions, it is not necessaily safe to mark s' for pruning.

We illustrate this with an example, showing that it can make sense to re-visit a given location l, if the path from the initial state s_0 to the state s' at the re-visiting time contains at least one failed branch. Consider that a location l is a stop and there is ample time to wait for the next trip b. An optimal policy could be to walk to a nearby stop and attempt to take a faster, express connection. If unable to catch the express line (*failed branch*), return to l and catch trip b.

In this example, state s is the state when we are at location l for the first time, i.e., before walking to the express-line stop. State s' is the state after coming back to l. Obviously, state s dominates s': they involve the same location l, s' is reached at a later time, and reaching s' involves an additional quota spending (more walking and more legs). Despite this dominance relation, it is not safe to prune away s', as s' is part of the optimal policy.

We note that the relevance of the results presented in this section goes beyond the journey planning domain. They apply to any domain where nondeterministic actions have two possible outcomes, namely one successful outcome and one failure outcome. This is a fairly generic modelling, applying to many other domains besides journey planning under uncertainty. For example, in robotics, the actions attempted by a robot can either succeed or fail, matching the generic uncertainty framework under consideration.

11. Combining Deterministic and Nondeterministic Search

In this section we present a hybrid solving approach that combines deterministic and nondeterministic search. This is introduced to speed up the solving process in difficult instances. The idea is illustrated in Figure 3. The hybrid approach starts with a deterministic search, and the resulting plan is checked for validity in the presence of uncertainty. In other words, we check if the plan remains correct and optimal in the nondeterministic domain. If that is the case, the problem is solved with just one deterministic search, which is typically significantly faster than a nondeterministic search. Otherwise, the search tree built in the deterministic search is used to improve the accuracy of the heuristic function available. A nondeterministic search is launched with the improved heuristic in use.

The next section presents the state space **D** used in the deterministic search. Then we point out some relations between **D** and **ND** that will be relevant in the analysis of our hybrid method. Finally, we present and analyze the hybrid method in detail.

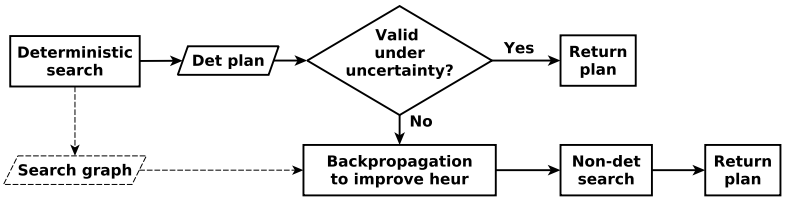

Figure 3: Architecture of the hybrid approach.

11.1 Deterministic Domain Relaxation

We obtain a deterministic domain as a relaxation from **ND**. Specifically, we convert nondeterministic actions into deterministic actions, by keeping only the successful branch. Actions that were deterministic in **ND** are preserved unchanged. Let $A_{det}(s)$ be the set of actions applicable to a state s in **D**. Recall that $A_{nd}(s)$ is the set of actions applicable to s in **ND**. Clearly, $|A_{det}(s)| = |A_{nd}(s)|$.

This relaxation is optimistic in the sense that, even when the probability of the successful branch is small (but strictly positive), we still consider that as the only outcome of the action in **D**.

Observation 3. *All successful branches in* **ND** *are available as deterministic actions in* **D**.

Recall the cost function $c(s_1, s_2)$ that returns a non-negative value as a state transition cost from state s_1 to state s_2. Additionally, $f(s_1)$, $g(s_1)$ and $h(s_1)$ represent an *f-value*, a *g-value*, and a *h-value* (or heuristic value) at state s_1, respectively. The f-value is defined as $f(s_1) = g(s_1) + h(s_1)$, where $g(s_1)$ is the sum of the transition costs from the initial state to reach s_1, and $h(s_1)$ is a value estimating a cost to reach the destination from s_1.

We define the optimal cost $v_{det}(s_1)$ of a state s_1 in **D** in a similar fashion to the definitions of the cost components $v_{exp}(s_1)$ and $v_{wst}(s_1)$ in **ND** provided in Section 7:

1. If s_1 is a goal state, $v_{det}(s_1) = 0$.
2. If $|A_{det}(s_1)| = 0$, $v_{det}(s_1) = \infty$.
3. Otherwise, $v_{det}(s_1) = \min_{a \in A_{det}(s_1)} \Big(c(s_1, a) + v_{det}(\gamma(s_1, a)) \Big)$.

As shown later, in Corollary 2, v_{det} is useful as an admissible heuristic in **ND**.

11.2 Cost and Heuristic Relations in D and ND

We present a few results that draw a connection between deterministic and nondeterministic search. Bonet and Geffner (2000b, 2005) have observed similar properties in their work on MDPs and probabilistic planning. Thus, the results presented in this section are not particularly new, but it is important to show they hold into our context as well, as they ensure the optimality of our hybrid approach.

Theorem 4. *For any state* s_1, $v_{det}(s_1) \leq v_{exp}(s_1) \leq v_{wst}(s_1)$.

Proof. We only prove that $v_{det}(s_1) \leq v_{exp}(s_1)$, as the other inequality is straightforward. Let π be a contingent plan in **ND** that is optimal on v_{exp}, and p be the pathway in the plan with the smallest

cost. Clearly, p may contain a mixture of successful branches (including deterministic actions in **ND**) and failed branches. By removing from π all actions (if any) containing a failed branch along p, we obtain a correct contingent plan in **ND**, according to Observation 2. The pathway p after the removal of failed branches forms a correct plan in **D**, according to Observation 3. Clearly, its cost v_{det} is no greater than $v_{exp}(\pi)$ as well as at least as large as $v_{det}(s_1)$, since p was the best-cost pathway in π and since $v_{det}(s_1)$ is the optimal cost in **D**. □

Theorem 5. *Any $h(s_1)$ admissible in* **D** *is admissible in* **ND**.

Proof. Let $h_{nd}(s_1) = (h(s_1), h(s_1))$ and $v_{opt}(s_1) = (v_{wst}(s_1), v_{exp}(s_1))$. It follows from Theorem 4 that $h_{nd}(s_1) = (h(s_1), h(s_1)) \leq_{lex} (v_{det}(s_1), v_{det}(s_1)) \leq_{lex} (v_{wst}(s_1), v_{exp}(s_1)) = v_{opt}(s_1)$. □

A heuristic is said to be *perfect* if it provides exact values, rather than estimations.

Corollary 2. *The perfect heuristic in* **D** $h^*(s_1) = v_{det}(s_1)$ *is admissible in* **ND**.

These results show that we can use an optimal cost in **D** or a lowerbound to admissibly guide the search in **ND**. We emphasize that the cost components v_{det}, v_{exp} and v_{wst} are specific to a given goal condition (i.e., they are initialized to 0 at goal states), but the initial state is irrelevant. As such, the results derived in this section are specific to a goal condition, but make no assumption about the initial state.

11.3 Hybrid Algorithm

Algorithm 1 Hybrid Deterministic Nondeterministic Search

Require: Initial state $root$
1: $O = C = H = \phi$
2: (v_{det}, p_1) = A*($root$, O, C, h)
3: **if** (all actions in p_1 are deterministic in **ND**) **then**
4: **return** p_1
5: **else**
6: Update($root$, O, C, H, v_{det})
7: **return** AO*($root$, $\max(H, h)$)

We now present our hybrid algorithm, whose main steps are illustrated in Algorithm 1 and Figure 3. Our approach first runs A* with an open list O and a closed list C, returning an optimal plan p_1 in **D**, as well as its cost v_{det}. Unlike standard A*, our A* takes into account the state dominance. Assume that A* generates a successor s and detects that there is a state u in $O \cup C$ such that $u \prec s$ (i.e., u dominates s). According to the definition of the domination relation, u and s have the same position $p_u = p_s$, but u is at least as good as s in terms of the travel time from the origin, the walking time from the origin, the cycling time from the origin and the number of legs from the origin. It follows easily that $v_{det}(u) \leq v_{det}(s)$ and $g(u) \leq g(s)$ in **D**, which implies that A* can safely discard s. That is, A* does not explore further the search space rooted at s. This plays a crucial role in significantly reducing A*'s search space.

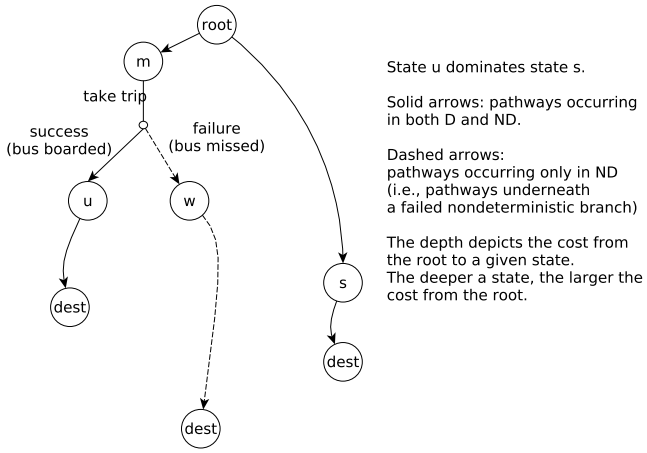

Figure 4: A simple example where a dominated state in **D** belongs to an optimal plan in **ND**.

Theorem 6. *Consider a deterministic plan π_1 computed in the deterministic search. If all π_1's actions b correspond to a deterministic action in* **ND** *(as opposed to b being the relaxation of a nondeterministic action in* **ND***), then π_1 is correct and optimal in* **ND***.*

Proof. The correctness follows from the fact that actions, being deterministic in **ND**, will behave as in the deterministic domain, being applicable in the corresponding state and succeeding with probability 1. For optimality, assume there is a nondeterministic plan π_2 with a better score $v_{opt}(\pi_2) <_{lex} v_{opt}(\pi_1)$. Let r_i be the root of $\pi_i, i = 1, 2$. Applying Theorem 4 to π_2, we obtain $v_{det}(\pi_2) \leq v_{exp}(\pi_2) \leq v_{wst}(\pi_2)$. At the same time, $v_{det}(\pi_1) = v_{exp}(\pi_1) = v_{wst}(\pi_1)$, since π_1 is linear and thus has no multiple branches. As both π_1 and π_2 are valid plans in **D**, and π_1 is optimal in **D**, it follows that $v_{det}(\pi_1) \leq v_{det}(\pi_2)$. Putting all these together, it follows that $v_{opt}(\pi_1) = (v_{wst}(\pi_1), v_{exp}(\pi_1)) = (v_{det}(\pi_1), v_{det}(\pi_1)) \leq_{lex} (v_{det}(\pi_2), v_{det}(r_2)) \leq_{lex} (v_{wst}(\pi_2), v_{exp}(\pi_2)) = v_{opt}(\pi_2)$, which is a contradiction. □

Theorem 6 provides a way to perform the validity test for a deterministic plan, on line 3 of Algorithm 1. When the test returns a negative result, the hybrid approach updates the heuristic, after which it runs AO*.

The Update method shown in Algorithm 2 back-propagates information through the A* search tree, computing more informed h-values and storing them in a table H. As in related previous work (Reinefeld & Marsland, 1994; Akagi, Kishimoto, & Fukunaga, 2010), Update traverses the search graph backwards and sets $H(s_1) = \min_{s \in C(s_1)}(f(s) - g(s_1))$, where $C(s_1)$ is the set of non-expanded frontier states reachable from s_1. However, unlike previous approaches, Update does not perform iterative deepening, and its depth-first search is limited to the threshold initialized to v_{det} at the initial state (see Algorithm 1), and to the set of states examined by A*.

At lines 1–4 of Algorithm 2, Update initializes the admissible heuristic r of a given state n to the maximum value between the heuristic $h(n)$ and the value stored in the table H (if any). At lines 6–11, the value r is recursively improved based on the successors' values. This is the backpropagation part. Line 13 is discussed later in this section. Finally, the r value of the node at hand is stored into the table H (line 15), and also provided as the return value of the method Update.

A* applies dominance pruning as mentioned earlier in this section. AO* applies dominance pruning as presented in Section 10. To ensure the plan optimality, our Update procedure correctly handles a subtle but important detail stemming from the use of dominance pruning. Assume a new state s is dominated by an older state $u \in O \cup C$. As in A*, Update skips examining s due to the pruning scheme with state dominance (see line 5 in Algorithm 2). While A* correctly marks s as a dead end in **D**, setting $H(s) = \infty$ in Update would be an error. Instead, in such cases we set $H(s) = h(s)$, where h is the initial heuristic (line 1 in Algorithm 2). The reason is that an aggressive update such as $H(s) = \infty$ could exceed the scope of the formal results presented in Section 11.2. Recall that, in those formal results, the goal is set but nothing is assumed about the initial state. On the other hand, dominance pruning proves s as a dead end only in a more specific context, considering the paths from the *current* initial state to u and s. Unlike the issue pointed out by Akagi et al. (2010) in IDA* with transposition tables (Reinefeld & Marsland, 1994), our scenario occurs even if the search space has no loops.

Example 1. *Figure 4 illustrates a scenario where a state s dominated in* **D** *belongs to an optimal plan in* **ND***, and therefore it would be a mistake to set $H(s) = \infty$. Assume that u dominates s. Recall that in* **D** *pathways starting with a failed branch are not generated. Thus, in* **D** *(solid arrows only), there are two plans: the sequential pathway through u and the sequential pathway through s. The one containing u is an optimal plan, and s can safely be pruned in* **D** *as a state dominated by u.*

In **ND** *(solid plus dashed arrows), there are two plans: the contingent plan with a branching point under m, and the sequential pathway through s. Here, the latter is an optimal plan, as we assume that the nondeterministic branch in the former plan has a very large cost.*

Therefore, setting $H(s) = \infty$ in Update would result in ignoring the plan through s in **ND** *and returning a suboptimal solution. To bypass this, our Update method propagates back a conservative admissible heuristic value $h(s)$ for s.*

Line 13 in Algorithm 2 is related to Holte et al.'s (1995), P-g heuristic, discussed in the related work section. Back-propagation (without line 13) provides better estimates than P-g (line 13 alone) in many cases. However, we found line 13 to be particularly useful in states pruned away with state dominance. As such pruned states are not expanded further, their exploration subtree is empty. On the other hand, back-propagation (without line 13) works by increasing the heuristic of a state based on the heuristic of its children. Clearly, when a state has no children (as it happens for states pruned away), back-propagation cannot increase the heuristic of that state. In such cases the P-g rule (line 13) is the only mechanism that can improve the heuristic, which of course can trigger further improvements up in the tree.

We argue that the P-g heuristic preserves the admissibility of H in **D** as follows: 1) When a state s_1 is processed in Update, we have $\theta = v_{det} - g(s_1)$. This follows easily from the initialization of θ to v_{det} (line 6 in Algorithm 1), and from the way it is updated in each recursive call (line 9 in Algorithm 2); 2) It is known that, at the end of an A* search that returns an optimal cost v_{det}, $v_{det} - g(s_1)$ is an admissible heuristic for every state s_1 with an optimal g-cost.

Our heuristic update strategy implements a few additional ideas. As shown in Algorithm 1, we take the $\max(h(s_1), H(s_1))$ as the heuristic to use in AO*. The reason is that, for those states s_1 not visited in A*, $H(s_1)$ returns 0. Secondly, when $u = (l, t_u, \dots)$ dominates $s = (l, t_s, \dots)$ in **D**, $t_s + H(s) = f(s) \geq f(u) = t_u + H(u)$, which further allows us to increase $H(s)$ to $H(u) + t_u - t_s$ within method GetValue in Algorithm 2. Thirdly, we added to AO* (both as a module of our method

Algorithm 2 Update

Require: State n, open list O, closed list C, hash table H, and threshold θ
1: $r = h(n)$
2: **if** $n \in H$ **then**
3: t = GetValue(n, H)
4: $r = \max(r, t)$
5: **if** $\theta \geq r \wedge n$ is not a goal $\wedge\ n \notin O \wedge$ no state $m \in O \cup C$ dominates n **then**
6: /* Improve heuristic values of successors */
7: $v = \infty$
8: **for** each of n's successor s **do**
9: $v = \min(v, c(n, s) + \text{Update}(s, O, C, H, \theta - c(n, s)))$
10: /* Increase r based on an improved heur. value v */
11: $r = \max(r, v)$
12: /* Increase r based on the optimal solution cost θ */
13: $r = \max(r, \theta)$
14: /* Save an improved heuristic value */
15: Save(n, r, H)
16: **return** r

and as a standalone benchmark) an extra heuristic update rule. Let s_s and s_f be the successful and the failed successors of a nondeterministic action. The heuristic of s_f can admissibly be increased to the heuristic of s_s, a property that follows from Theorem 1.

Assumption 1. *We assume that the initial heuristic available h is admissible in* **D**.

In particular, this holds for the heuristics discussed in Sections 8 and 9. When the cost is the travel time, the admissible heuristic for the travel time described in Section 8 plays the role of our initial heuristic. When the cost is the number of legs, the initial heuristic is the admissible heuristic for the number of legs, described in Section 9. When the cost is a linear combination between the two, the initial heuristic is the linear combination of the corresponding heuristics.

Theorem 7. *The hybrid method returns optimal solutions in* **ND**.

Proof. It is sufficient to prove $H(s_1) \leq v_{det}(s_1), \forall s_1$. Then H is admissible in **ND** according to Theorem 5. With no generality loss, assume that $H(s_1) = h(s_1)$ if s_1 is not found in H. Here we only show that $H(s_1) = \min_{a \in A_{det}(s_1)}(c(s_1, a) + H(\gamma(s_1, a))) \leq v_{det}(s_1)$. Other rules (e.g., increasing $H(s_1)$ to θ) were discussed earlier. The proof is related to Akagi et al.'s (2010) work. Let H_k be the table H after k updates. If $k = 0$, the theorem holds, as nothing new is saved in H, and $h(s_1) \leq v_{det}(s_1)$ according to Assumption 1. Assume that the result holds for k (i.e., $H_k(s) \leq v_{det}(s)$), and we are about to save a new result for node s_1 at step $k + 1$. We have: $H_{k+1}(s_1) = \min_{a \in A_{det}(s_1)}(c(s_1, a) + H_k(\gamma(s_1, a))) \leq \min_{a \in A_{det}(s_1)}(c(s_1, a) + v_{det}(\gamma(s_1, a))) = v_{det}(s_1)$. □

12. Simulating Plans

In a multi-modal journey advisor, simulating a plan is useful to detect when new knowledge about the network snapshot or about the trip at hand can invalidate the journey plan under considera-

tion (Botea et al., 2014). For instance, an incident on a tram route may cancel out all tram trips on that route for the rest of the day. If a plan currently under execution (i.e., a trip in progress) includes a leg on that tram route at some point in the future, plan simulation can detect that the plan becomes invalid. The traveler can be notified and re-planning can be performed. With plan simulation in use, re-planning can be performed just for a subset of plans impacted by recent changes, as opposed to blindly re-planning for all trips in progress.

In this work, we use plan simulation to evaluate differences between plans computed under deterministic assumptions (i.e., no uncertainty in the network snapshot), and plans computed by taking uncertainty into account. The results are discussed in the experiments section. In this section we overview our plan simulation procedure.

The simulator takes as input a network snapshot n, a plan p, a state s in the plan, and a probability distribution t representing a stochastic time. It simulates the rest of the plan, starting from state s and from time t, given the network snapshot n. It is important to note that the snapshot used for plan simulation is not necessarily the same one used for computing the plan. The simulation is a recursive procedure that returns both the worst-case and the expected arrival time. The recursions simulate forward the travel along each pathway in the contingent plan, and propagate the arrival times from the leaves all the way up, as shown later in this section.

We assume that each branch in the contingent plan p is annotated with a time interval within which the traveler is allowed to attempt to follow that action. We call this a branch *activation interval*. In practice, the activation interval is useful in those cases when the traveler has no information regarding whether a given trip has already passed through a given stop. Instead of indefinitely waiting for a trip that is already gone, the traveler knows that they should follow the next branch available in the contingent plan (if any) at the end of the activation interval of the branch at hand.

Consider the contingent plan shown in Table 2, for the toy network shown in Figure 1. At location C, the traveler should first attempt to take trip 40 departing at around 11:21. The activation interval of this action is (LT, 11:24]. LT is a constant. It represents a time moment that is earlier than the expected time of boarding the trip by a sufficient margin. (For instance, set LT to 00:01am). The right end value 11:24 is set according to the uncertainty level assumed in that example (i.e., that bus departs at 11:21+3 min = 11:24 at the latest). The semantics are that, if the user arrives at the stop before 11:24, they should give priority to boarding this trip, as long as the time does not exceed 11:24. The activation interval of the walking branch from C to D is (11:24, UT). UT is a constant. It represents a time moment sufficiently bigger than the expected execution time of this leg. According to this, if, by 11:24, the traveler did not manage to board a trip along route 40, the traveler should give up on the bus trip leg and consider the walking leg instead.

For a deterministic branch in the plan, such as "take route 90 from D to F", the activation interval is (LT, UT). Every node in a contingent plan has exactly one outgoing branch with the property that its activation interval ends at UT. We call this the last option branch at that node.

When simulating a branch in a plan, it is possible that 0, 1 or more actions can be simulated for that particular leg. The first and the third cases require a special attention when simulating a plan to extract the worst-case arrival time and the expected arrival time. We discuss these, starting with the case of 2 or more actions being possible.

Consider a branch in the plan involving a public transport leg. When simulating the plan, it is possible in principle to have more than one trip satisfying the following properties: i) the trip belongs to the route at hand; ii) the probability to catching the trip at the stop at hand is greater than 0; iii) the (uncertain) departure time of the trip overlaps with the activation interval at hand; iv)

no earlier trip on the same route is safe (a trip is safe if the user can catch it with probability 1 at the current stop). Our simulation procedure handles the case of multiple trips satisfying the given properties. The procedure generates a new branch in the simulation tree for each trip satisfying conditions i) to iv).

Cases where no action can be simulated for a leg in a plan include, for instance, the case when a public transport route is significantly delayed due to an incident on the road. Assume that route 90 is interrupted for the rest of the day due to an incident. Then, when attempting to simulate that leg in the plan, no action will be generated, since in the new network snapshot (i.e., after the incident) no action satisfies properties i) to iv) mentioned in the preceding paragraph.

When no action is generated for a branch that is not the last option of its parent node, then the simulation continues by exploring other branches in the simulation tree. On the other hand, if no action is generated for a last-option branch, then the simulation of the plan fails. The reason is that the plan has a state from where no valid way exists to continue the simulation of the trip. We call this a plan interruption failure.

If no plan interruption failure is triggered, the simulation goes down all the way to leaf nodes, and worst-case arrival times and expected arrival time are back-propagated through the tree. The worst-case arrival time of a parent node is set to the max value among its children. The expected arrival time of a parent node is set to the weighted sum of the values of the children.

After back-propagating these arrival times, the worst-case and the expected arrival time can be compared with the original worst-case and expected arrival time. If the differences exceed predefined thresholds, the simulation of the plan is also considered to be failed.

13. Hardness Results

In this section, we develop a theoretical foundation for understanding the complexity of the multimodal journey planning problem. To pin down causes of difficulty, we identify problems that turn out to be computationally difficult. We contrast our worst-case travel time objective with the more standard expected travel time objective in terms of problem hardness.

The main conclusion from this section is that if the distributions of the travel times along edges vary in time, then calculating the worst-case travel time of a given plan or route is NP-hard.

Consider a graph $G = (V, E)$ with $|V| = n$ nodes, $|E|$ edges and a given source-destination pair of nodes S, T. As in Section 8, each node in the graph corresponds to a location, and there may be multiple kinds of edges, each corresponding to one transport mode, such as taking a trip, cycling, and walking. The edge travel times are assumed to be stochastic, represented by independent random variables coming from given distributions. A routing policy specifies, for each node in the graph and for each set of random variable observations, the best successor to continue the route on. We consider two variations of the problem under the following assumptions on random variable observations:

Assumption 2. *The edge travel time is observed once an edge is traversed.*

Assumption 3. *The edge travel time is observed once an endpoint (the beginning) of the edge is reached.*

The second assumption is associated with the Canadian Traveller problem (Papadimitriou & Yannakakis, 1991; Nikolova & Karger, 2008), in which the goal is to find an optimal policy for reaching from the source to the destination in minimum *expected* travel time.

In multimodal journey planning both assumptions can arise. For example, whether a bridge is traversable or not depends on its status (open or closed), which can be observed when a user arrives at the bridge. On the other hand, riding a bus has an associated stochastic travel time whose actual value can only be observed once the end-point of the trip is reached.

We consider multiple combinations of Assumptions 2 and 3. In the *standard* setting, Assumption 2 holds, whereas Assumption 3 holds in the *look-ahead* setting. The *mixed* setting considers both assumptions. We now examine the computational complexity for the associated problems with the two objective functions, expected and maximum travel time, in the different settings.

13.1 Expected Travel Time Objective

13.1.1 STANDARD SETTING

When our goal is to reach the destination with minimum expected travel time, by linearity of expectation, it suffices to replace each edge random variable travel time with its expectation and run a deterministic shortest path algorithm for finding the optimal route. The expected value of the edges that have not yet been traversed does not depend on the edges traversed so far. Therefore, the offline optimal solution (optimal plan) is also an optimal policy and, in particular, contingencies need not be considered.

Proposition 3. *The standard setting under the expected travel time objective can be solved in polynomial time via a deterministic shortest path algorithm.*

13.1.2 LOOK-AHEAD AND MIXED SETTING

The look-ahead setting with expected travel time objective is also known as the Canadian traveller problem, which is #P-hard (Papadimitriou & Yannakakis, 1991). Since the mixed setting includes the look-ahead setting as a special case, it is at least as hard.

13.2 Worst-Case Travel Time Objective

The worst-case travel time objective is well-defined only if there exists a path from the source to the destination along which all edge travel times have finite upper bounds. We have this assumption when talking about the worst-case travel time objective. On the other hand, the expected travel time can be finite even if a distribution includes infinite values, as in the case of the Normal, Exponential and Gamma distributions, for example.

13.2.1 STANDARD SETTING

In the standard setting, edges traversed so far have no effect on the values of edge travel times that have not yet been traversed. Therefore, we can simply replace all edge travel times by their maximal values and run a deterministic shortest path algorithm with respect to these values.

Proposition 4. *The standard setting under the worst-case travel time objective can be solved in polynomial time via a deterministic shortest path algorithm.*

Note, in particular, that just as in the standard setting under the expected travel time objective, the optimal plan and the optimal policy here coincide, namely the shortest path that is computed should always be followed regardless of traversed edges and their realized travel times. Claims such

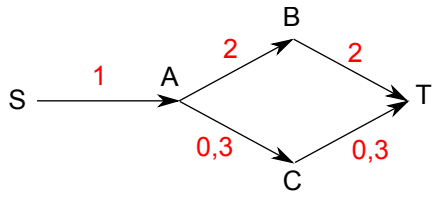

Figure 5: Worst-case travel time routing changes depending on observed edge travel times in the look-ahead setting, while it does not change in the standard setting.

as Propositions 3 and 4 are not necessarily new. They are variants of the shortest-path problem in a graph, where the edge costs are set to the expected traversal time (Proposition 3) or the maximum traversal time (Proposition 4).

13.2.2 LOOK-AHEAD AND MIXED SETTING

In the worst case, we will observe the worst case edge values and will go along the shortest path with respect to these worst-case edge values.

Proposition 5. *The look-ahead setting under the worst-case travel time objective can be solved in polynomial time via a deterministic shortest path algorithm with respect to edge weights equal to the maximal edge travel times.*

The difference from the standard setting above is that while routing, the shortest path from the current node to the destination may change depending on the (look-ahead) observations so far. Consider the example in Figure 5. At the source node S, the path minimizing the worst-case travel time is through B. However, upon reaching A and seeing that $AC = 0$, the optimal path becomes the one through C.

Finally, since a shortest path algorithm with respect to edge weights equal to the maximum edge travel times yields the optimal routing strategy in both the standard and look-ahead setting, it will do so in the mixed setting, as well.

Proposition 6. *The mixed setting under the worst-case travel time objective can be solved in polynomial time via a deterministic shortest path algorithm with respect to edge weights equal to the maximal edge travel times.*

13.3 Time-Varying Distributions

In the previous sections, we assumed that the edge travel time distributions do not change with time. We showed that in that case, worst-case travel time routing can be performed in polynomial time. In practice, the distributions change with time. For example, peak and off-peak travel time distributions often differ. This changes the problem complexity considerably. In this section, we show that even in the simplest case with only two time periods (where the travel time distribution of an edge may change from the first to the second period), the problem of just computing the worst case travel time along a *given* path is computationally hard, and consequently the problem will be hard under all settings discussed above.

Theorem 8. *Given a graph with stochastic edge travel times that come from static distributions before or at time t and from different static distributions after time t, it is NP-hard to compute the worst-case travel time along a given path.*

Proof. We reduce from the subset sum problem (for a given set of integers $a_1, ..., a_n$ and a target t, decide if there exists a subset which sums to t), which is NP-hard (Garey & Johnson, 1979). Without loss of generality, we assume that $a_i \neq 0$ for all $i = 1, ..., n$, since removing the zero elements has no effect on the problem. Consider a single path consisting of n edges $e_1, e_2, ..., e_n$ in that order. Suppose that before time t, the edge travel times come from two-valued distributions: edge e_i takes on values $\{0, a_i\}$. After time t, all edge travel times are 0. Denote the realized edge travel times by d_i, $i = 1, ..., n$.

Without loss of generality, suppose that we start our journey along the path at time 0. The question is, where are we going to be at time t so as to know which distribution to use for each edge? We use the convention that if time t occurs at the start or during traversal of edge e_i, its value comes from the second distribution (namely, its value is 0) and otherwise its value comes from the first distribution.

If e_k is the last edge that follows distribution 1, namely time t occurs at the beginning or inside edge e_{k+1}, then the travel time along the path can be expressed as $\sum_{i=1}^{k} d_i = \sum_{i=1}^{k} a_i x_i \leq t$, where $x_i \in \{0, 1\}$ is an indicator variable of whether $d_i = 0$ or $d_i = a_i$. Therefore, the worst-case path travel time is given by the mathematical program:

$$\max_{k,x}\{\sum_{i=1}^{k} a_i x_i \mid \sum_{i=1}^{k} a_i x_i \leq t\,;\ \sum_{i=1}^{k} a_i x_i + a_{k+1} > t\,;\ x_i \in \{0,1\} \quad \forall i = 1, ..., k\} \tag{1}$$

The two constraint inequalities represent the fact that time t occurs at the beginning or inside of edge e_{k+1}. The maximum is taken over $k = 1, 2, ..., n$. (To be precise, when $k = n$, the second inequality is eliminated.)

Next, we will show that the value of program (1) is t if and only if there is a subset of $\{a_1, ..., a_n\}$ that sums to t.

We first show that the maximum in program (1) is attained at $k = n$, namely, it is equivalent to the following program:

$$\max_{x}\{\sum_{i=1}^{n} a_i x_i \mid \sum_{i=1}^{n} a_i x_i \leq t;\ x_i \in \{0,1\}\ \forall i = 1, ..., n\} \tag{2}$$

Claim 1. *Programs (1) and (2) have the same optimal value.*

Proof. First, we note that the maximum value of program (1) can only increase if we remove the second constraint. On the other hand, with the second constraint removed, the maximum $(x_1, ..., x_k)$ of

$$\max_{x}\{\sum_{i=1}^{k} a_i x_i \mid \sum_{i=1}^{k} a_i x_i \leq t;\ x_i \in \{0,1\}\ \forall i = 1, ..., k\} \tag{3}$$

extended by $x_j = 0$ for $j > k$ is a feasible solution to program (2), and thus its value is no more than the value of program (2), for all k. So, the value of program (1) is less than or equal to the

value of program (2). On the other hand, the value of program (2) is less than or equal to that of program (1), since the latter is a maximum over program (2) (for $k = n$) and other programs (for $k < n$). Thus, the values of the two programs are equal. □

Claim 2. *The value of program (2) is t if and only if there is a subset of $\{a_1, ..., a_n\}$ that sums to t.*

Proof. Suppose the value of program (2) is t. Then, the objective of program (2) specifies a subset of $\{a_1, ..., a_n\}$ that sums to t. Conversely, suppose there is a subset that sums to t. Taking $x_i = 1$ if a_i is in the subset and $x_i = 0$ otherwise gives a feasible solution to the program of value t. Therefore, the maximum of the program is at least t. On the other hand, the maximum is at most t by the inequality constraint. Therefore, it is exactly t, as desired. □

To conclude, for any given set of integers $a_1, ..., a_n$, we have provided a polynomial-time transformation to the worst-case travel time problem, whereby a subset of the integers sums to t if and only if the worst-case travel time is t. Therefore, the problem of computing the worst-case travel time on a given path is NP-hard. □

Corollary 3. *In the conditions of Theorem 8, it is NP-hard to answer whether a given journey plan is optimal with respect to the worst-case travel time.*

Proof. This can be shown with a reduction from the problem constructed in the proof to Theorem 8. Consider an arbitrary instance of that problem. In other words, consider a chain of n edges $e_1, e_2, \ldots, e_n$ in this order, with the travel time distributions for each edge e_i as explained in the proof to the theorem.

Construct in polynomial time a journey planning instance with two routes to travel from A to B: One route is precisely the chain $e_1, e_2, \ldots, e_n$. The other route has a deterministic travel time of $t - 0.5$, by construction.

Observe that the travel time on the chain $e_1, e_2, \ldots, e_n$ is always an integer, since all numbers $a_1, a_2, \ldots, a_n$ are integers by construction. Based on this, it is easy to check that the deterministic route is optimal (i.e., the journey planning instance has an optimal worst-case travel time of $t - 0.5$) if and only if the chain $e_1, e_2, \ldots, e_n$ has a worst-case arrival time of t.

□

14. Experimental Evaluation

We present a detailed empirical evaluation of our ideas.

Our system, DIJA, is implemented in C++. It integrates ideas presented in this paper, including AO* search, the hybrid deterministic–nondeterministic search approach, admissible heuristics for travel time and quotas, pruning techniques, and the cost metrics discussed earlier. It allows computing (contingent) journey plans and simulating their execution.

We have used transportation data from three European cities, Montpellier, Dublin, and Rome. The Dublin data contain 4,739 stops, 120 routes, and 7,308 trips per day. The road network has 301,638 nodes and 319,846 segments. In Montpellier, bus and tram data amount to 1,297 stops, 36 routes and 3,988 trips per day. The road network has 152,949 nodes and 161,768 links. In Rome, buses, trams, subways and light trains sum up to 391 routes, with 8,896 stops and 39,422 trips per day. The road map contains 522,529 nodes and 566,400 segments.

The original data is deterministic. This was extended with a stochastic noise assigned to the original deterministic arrival and departure times. The noise follows a Normal distribution, truncated to a confidence interval of 99.7%. We report results with two distinct levels of noise. The smaller level has $\sigma^2 = 1600$ seconds, equal roughly to ± 2 minutes around the original deterministic arrival or departure times. In the larger noise level we set $\sigma^2 = 6400$, which roughly corresponds to a ± 4-minute uncertainty interval.

We have set the mean arrival (departure) values to the scheduled arrival (departure) times available in the GTFS data for simplicity. The side effect is that vehicles could depart earlier than scheduled. However, if desired, this can be addressed by increasing the mean values by a small constant, and increasing the departure times in all queries by the same value. This would lead to identical experimental results, but the early departures would be reduced.

We used 1,000 journey plan requests (instances) for each city. The origins and the destinations are picked at random, and the departure time is 11AM. Quotas are set to at most 20 minutes of walking, and at most 5 segments per trip, unless explicitly mentioned otherwise.

Tests are run on a RedHat machine with an Intel Xeon CPU @ 3.47GHz.

14.1 Evaluating Search Enhancements

In this section we evaluate search enhancements such as admissible heuristics, and pruning techniques described earlier in the paper.

14.1.1 ADMISSIBLE HEURISTIC

Recall that the cost of a branch in a plan can be set to any linear combination between the number of legs L and the travel time T (measured in seconds):

$$C = L \times (1 - cw) + T \times cw,$$

with $cw \in [0, 1]$.

The heuristic evaluation of a state is computed accordingly, as a weighted sum between an admissible estimation of the number of legs, and an admissible estimation of the travel time. In our work, the admissible estimation of the number of legs turns out to be very accurate. Part of the reason is that the number of legs typically takes just a few discrete values, and trajectories computed heuristically often have the same number of legs as an optimal trajectory. On the other hand, the travel time heuristic presented in Section 8 is less accurate. Part of the reason is that the heuristic ignores any waiting that might be needed in a trip.

As a result, a smaller cw value would result in a more accurate heuristic. At the same time, a smaller cw value could potentially result in plans better optimized for the number of legs, and less optimized for the travel time.

We have performed an empirical evaluation, using a range of values for cw, and observing the changes in the performance, in terms of planner speed and total travel time.

The results are illustrated in Figure 6, for Rome, with the level of uncertainty $\sigma^2 = 6400$. Results for other combinations of a city (i.e., Dublin, Montpellier and Rome) and a level of uncertainty ($\sigma^2 \in \{1600, 6400\}$) are similar.

Figure 6 shows the performance of a few values for cw, such as 0.1 (at the left), 0.005 (the middle column) and 0.0001 (at the right). In each case, the performance is compared to the system running with the reference value $cw = 1$ (i.e., the system optimizing purely the travel time). In the

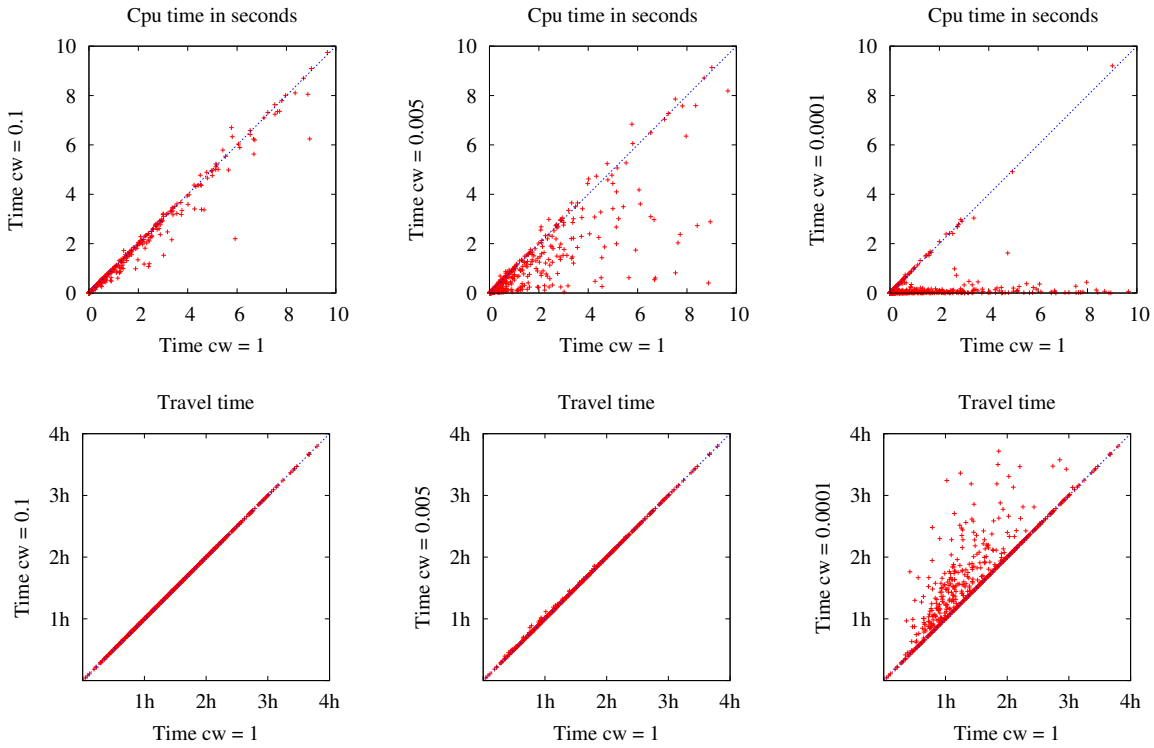

Figure 6: Impact of varying weight cw. Results shown for Rome, $\sigma^2 = 6400$.

figures, the charts at the top show the CPU time necessary to complete the search for a plan. The charts at the bottom show the travel time.

At the left ($cw = 0.1$ vs $cw = 1$), the travel time is virtually the same, and an improvement in the CPU time can often be observed. In the CPU time chart, all points below the main diagonal show a speed up achieved with $cw = 0.005$ in use. In the middle column ($cw = 0.005$ vs $cw = 1$) the CPU time speed further increases. The travel time is optimal in most cases. Sometimes the travel time increases (dots slightly above the diagonal in the chart), but the increase is small and it is compensated by a reduction in the number of journey legs. At the right, we evaluate a much smaller value of $cw = 0.0001$ versus the reference value $cw = 1$. As expected, the speed performance increases even further. On the other hand, we observe that giving too much weight to the number-of-legs cost component loses control over the travel time, as seen in the chart in the bottom-right corner. All points above the main diagonal indicate cases where the travel time is sub-optimal. The figure indicates that, with $cw = 0.0001$, in many cases, the total travel time can be quite far from optimal.

Based on the experiment presented here, $cw = 0.005$ arguably is a good trade-off between the travel time, the number of legs and the speed performance, which is why we have set this as the default value for the cw parameter.

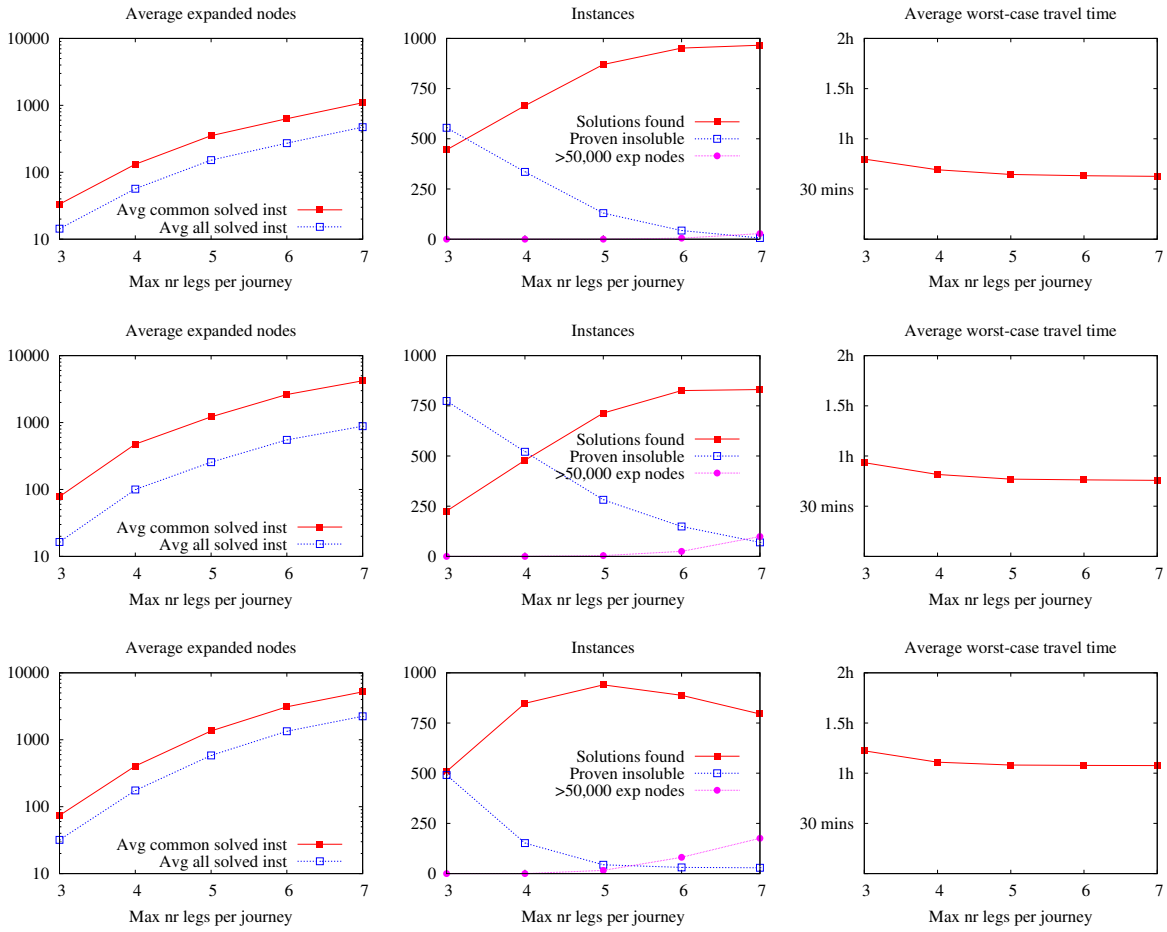

Figure 7: Impact of pruning with admissible quota estimates. Top: Montpellier; middle: Rome; bottom: Dublin. $\sigma^2 = 1600$.

14.1.2 PRUNING WITH ADMISSIBLE QUOTA ESTIMATES

In experiments, it turns out that pruning with admissible quota estimates is a powerful speed-up technique. Figure 7 summarizes the results for the number-of-legs quota.

The charts in the left column of Figure 7 show the number of nodes expanded in search, as the max number of legs q_L is increased. We show two curves in each case. One curve shows the number of expanded nodes per query on average. To ensure that each average value is computed over the same subset of queries, we consider in this computation the subset of queries solved for all q_L values considered. A query is "solved" if a solution is found or the query is proven insoluble. In other words, solved queries are those that do not exceed the 50,000 node-expansion threshold. The second curve shows the expanded number of nodes averaged over all queries solved for the current value q_L.

Tighter constraints on the quotas (e.g., fewer legs allowed) make this pruning technique more agressive, resulting in a faster plan computation. This is not surprising.

City	CPU time		Speedup	$> 50K$ EN		Avg #	Max # of
	Dp on	Dp off		Dp on	Dp off	legs	branches
$\sigma^2 = 1600$							
Montpellier	**0.03**	0.56	15.82	**0%**	7.1%	3.87	4
Rome	**0.25**	0.81	3.20	**0.4%**	3.0%	4.35	6
Dublin	**0.29**	1.53	5.11	**1.6%**	14.3%	3.54	4
$\sigma^2 = 6400$							
Montpellier	**0.03**	0.73	19.49	**0.3%**	19.3%	3.80	6
Rome	**0.55**	1.25	2.25	**1.7%**	7.7%	3.84	8
Dublin	**0.58**	1.87	3.19	**4.9%**	23.7%	4.34	6

Table 3: Summary statistics for dominance pruning (Dp). We show the average CPU time in seconds, the average speedup, the percentage of the instances that require more than 50,000 expanded nodes ($> 50K$ EN), and node branching information.

To illustrate why, let us revisit the example presented in Section 9. In that example, a state s is reached from the root in three legs (i.e., $g_L(s) = 3$). An admissible estimation of the legs needed to reach the destination from s is $h_L(s) = 3$. With max 5 legs allowed, we could prune away node s, as we need at least 6 legs on any solution containing s. However, if the user allows at most 6 legs, node s cannot be pruned with the information available. The available information $g_L(s) + h_L[s] = 6$ states that it is possible in principle to reach the destination within 6 or more legs, using a plan that contains node s.

Relaxing a constraint such as q_L can increase the set of solutions (valid plans) of a query (instance). Two direct effects of this are that the number of instances for which a solution exists can increase; and the optimal travel time for a given query can decrease.

The middle column in Figure 7 shows the impact of q_L on the number of instances (queries) with a solution. As such, for a given q_L, the set of queries is partitioned into three subsets: instances that have a solution, instances for which no solution exists, and instances that could not be solved after expanding 50,000 nodes. Relaxing the constraints increases the number of instances with a solution. The number of instances with a solution gets stable around a value of $q_L = 6$ or even $q_L = 5$ in Dublin. The rightmost column shows the impact on the travel time. We observe that the behavior is fairly stable, except perhaps for small q_L values, such as 3. Based on the tendencies observed in the middle and the rightmost columns, we conclude that 5 or 6 are reasonable default values for the q_L parameter. In experiments reported in this paper, the default value in use is 5.

14.1.3 PRUNING WITH STATE DOMINANCE

Figure 8 shows the impact of dominance pruning on the search time. Table 3 shows summary statistics with the impact of dominance pruning. Significant improvements are observed for all combinations of a city and an uncertainty level. Having the dominance pruning turned on is a default setting of the planning system. Thus, results presented in other sections of this paper are obtained with dominance pruning on. We further remark that the positive impact of dominance pruning is stronger on smaller problems. As a problem space gets larger, the likelihood of finding dominated nodes might decrease.

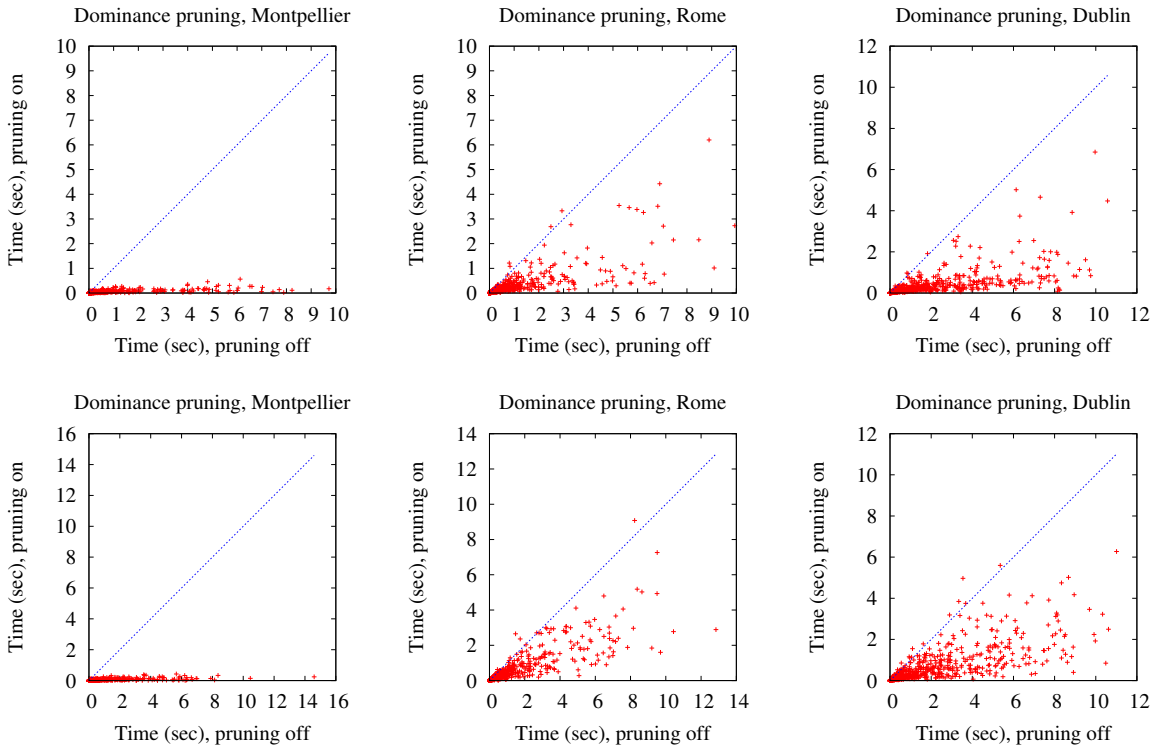

Figure 8: Impact of dominance pruning. Top: $\sigma^2 = 1600$; bottom: $\sigma^2 = 6400$. Instances requiring more than 50,000 node expansions with dominance pruning turned off are not included.

City	DP	ASM	ASP	DP	ASM	ASP
	$\sigma^2 = 1600$			$\sigma^2 = 6400$		
Dub	30.18%	28.48	24.48%	41.49%	27.73	23.45%
Rom	24.89%	17.35	18.36%	31.54%	17.89	18.18%
Mon	15.05%	19.52	24.94%	24.12%	19.94	24.73%

Table 4: Contingent vs deterministic plans: key statistics with worst-case travel time. DP = percentage of cases when differences occur. ASM = average savings per trip, in minutes, when differences occur. ASP = average savings per trip, as a percentage of the trip time, when differences occur.

From the usability point of view, note that the number of branches is relatively small, making the system usable and understandable from the point of view of the user. Namely, we observed that in Dublin scenario 98.03% of the plans had less than 3 branches. This number is confirmed in both Montpellier and Rome use cases too, with respectively 84.73% and 89.16% plans with at most 3 branches. Note that the percentage of trips with at most 4 branches is 99.57% for Dublin, 86.22% for Montpellier and 97.92% for Rome scenarios.

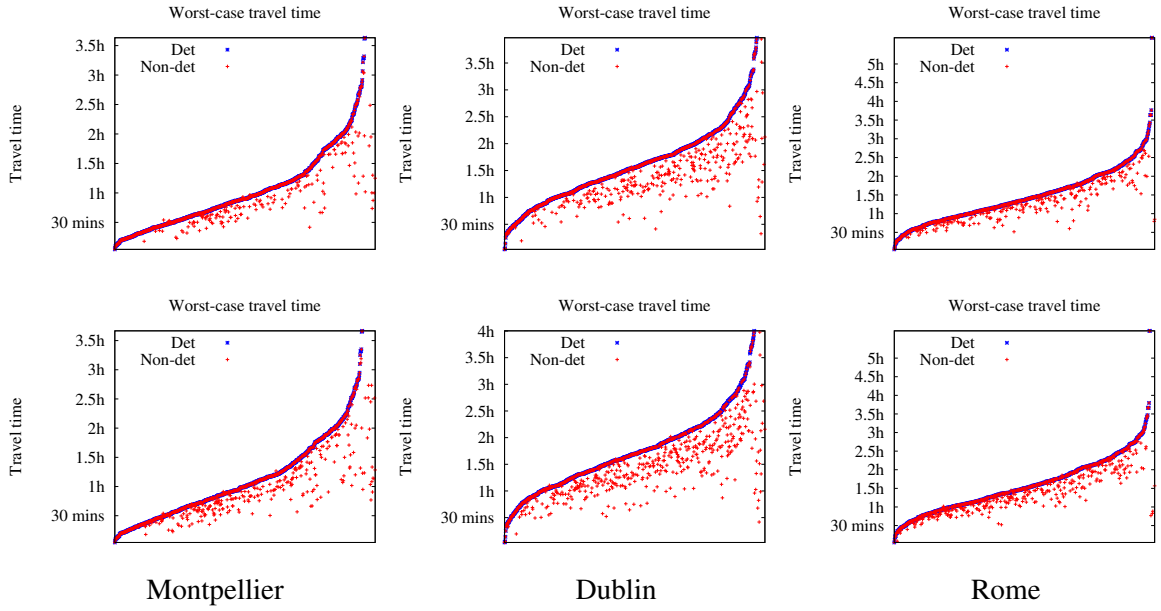

Figure 9: Top: $\sigma^2 = 1600$; bottom: $\sigma^2 = 6400$. Sequential plans ("Det") vs contingent plans ("Non-det"). Instances ordered increasingly on the worst-case arrival time of sequential plans.

14.2 Non-Deterministic vs Deterministic Plans

In this experiment we use three distinct snapshots for each city: the original deterministic snapshot ($\sigma^2 = 0$), a non-deterministic snapshot with $\sigma^2 = 1600$, and a non-deterministic snapshot with $\sigma^2 = 6400$.

To obtain a deterministic, sequential plan, we have run DIJAwith a deterministic network snapshot. Uncertainty-aware plans are computed with DIJAusing a non-deterministic snapshot. Both kinds of plans are simulated in a snapshot with uncertainty, and both the worst-case arrival time and the expected arrival time are measured.

14.2.1 WORST-CASE ARRIVAL TIME

Figure 9 compares deterministic and uncertainty-aware plans, in terms of simulated worst-case arrival time, for Montpellier, Dublin and Rome.

Each query has two points: one for deterministic planning (blue) and one for nondeterministic planning (red). Along the horizontal axis, queries are ordered in such a way that the travel time computed with deterministic planning is monotonically increasing.

The "main curve" shows the deterministic-plan data. Depending on how "det" and "non-det" values compare, we distinguish three behaviours in this figure. First off, in a majority of cases, ranging from 58.51% in Dublin, $\sigma^2 = 6400$ to 84.95% in Montpellier, $\sigma^2 = 1600$, "non-det" points fall on the "main curve", indicating instances where both simulated times coincide. Secondly, in most remaining cases, "non-det" points fall underneath the main curve, corresponding to cases where contingent plans have a better simulated time. Table 4, discussed later, shows exact percentages and other key statistics corresponding to this behaviour. Thirdly, in just a few cases, deterministic plans have a better arrival time. Part of the explanation is that the DIJAplanner optimizes plans on a

City	Strategy	DP	ASM	ASP	DP	ASM	ASP
		$\sigma^2 = 1600$			$\sigma^2 = 6400$		
Dublin	ND better	**12.18%**	13.33	14.12%	**19.65%**	12.13	12.57%
	D better	5.26%	10.34	12.79%	8.39%	10.33	12.28%
Rome	ND better	**7.96%**	9.49	10.87%	**14.53%**	9.12	9.71%
	D better	6.68%	8.40	11.41%	6.68%	10.28	12.59%
Montpellier	ND better	**4.30%**	11.31	15.33%	**10.66%**	10.27	15.36%
	D better	2.27%	12.21	18.31%	2.33%	10.91	16.47%

Table 5: Contingent vs deterministic plans: key statistics with expected travel time. DP = percentage of cases when differences occur. ASM = average savings per trip, in minutes, when differences occur. ASP = average savings per trip, as a percentage of the trip time, when differences occur. "ND better" is for cases when contingent plans have a better expected arrival time, and "D better" is for cases when deterministic plans feature a better expected arrival time.

linear combination of the number of journey legs (segments) and the arrival time. In a few cases, the optimal contingent plan has a later arrival time, and fewer legs than the corresponding deterministic plan.

We conclude from Figure 9 is that uncertainty-aware plans can often help arrive at the destination earlier.

Table 4 shows key statistics of the comparison. Header DP shows the percentage of "Non-det" dots not placed on the main curve in Figure 9. As expected, increasing the level of uncertainty increases the DP value. Remarkably, ASM and ASP remain stable when σ^2 varies.

Furthermore, it appears that DP, ASM and ASP depend on the "density" of the trips in a city (number of trips relative to the size of the network). For instance, in the data we used, Dublin has a smaller number of trips than Rome, which increases the DP, the ASM and the ASP values in the former city. A similar tendency is observed when varying the number of trips in a given city. In a different experiment, we have used a larger number of trips, obtained by combining trips from different days and, in the case of Dublin, including a few tram and train lines in addition to the existing bus routes. More trips in use reduce DP, ASM and ASP. For instance, increasing the number of trips per day in Dublin from 7,308 to 21,056 reduces DP to 32.45%, ASM to 18.18 minutes and ASP to 15.71% ($\sigma^2 = 6400$). In Montpellier, an increase from 3,988 to 5,985 trips per day results in DP $= 16\%$, ASM $= 13.02$, ASP $= 16.36\%$ ($\sigma^2 = 6400$).

14.2.2 EXPECTED ARRIVAL TIME

Table 5 shows key statistics in terms of the expected travel time. For every combination of a city and a level of uncertainty, cases when non-deterministic plans are more beneficial are significantly more frequent than cases where deterministic plans are more beneficial.

For example, in the case of Dublin, $\sigma^2 = 1600$, 12.18% cases favor contingent plans. When this happens, the average savings in terms of expected arrival time are 13.33 minutes, or 14.12% of the expected trip duration. 5.26% cases are in favor of deterministic plans, with average time savings per trip equal to 10.34 minutes (12.79%). For $\sigma^2 = 6400$, 19.65% cases are in favor of contingent plans. In such cases, average savings per trip amount to 12.13 minutes, or 12.57% of the

City	Strategy	DP	ASM	ASP	DP	ASM	ASP
		$\sigma^2 = 1600$			$\sigma^2 = 6400$		
Dublin	ND better	**15.82%**	19.25	17.48%	**21.09%**	20.21	17.26%
	DDR better	0.51%	13.45	18.75%	1.82%	12.81	16.51%
Rome	ND better	**10.14%**	14.24	15.65%	**13.25%**	14.65	15.28%
	DDR better	0.50%	7.72	11.43%	1.13%	13.17	12.51%
Montpellier	ND better	**5.43%**	27.99	27.48%	**10.99%**	23.47	25.23%
	DDR better	0.30%	7.57	15.29%	0.41%	19.09	24.09%

Table 6: Contingent planning vs DDR: key statistics for the worst-case travel time.

trip duration. 8.39% cases favor deterministic plans. Average savings per trip boil down to 10.33 minutes (12.28%). See Table 5 for similar statistics corresponding to Rome and Montpellier.

In terms of the worst-case arrival time, differences between contingent and deterministic plans have been evaluated in Section 14.2.1. This section has focused on the expected arrival time. Significant advantages are observed on both metrics. However, in terms of expected arrival times, differences between contingent and deterministic plans are smaller, and they go in both directions. This is consistent with the planner's optimization strategy: the worst-case cost is the main criterion, and the expected cost is for tie-breaking.

14.2.3 DYNAMIC DETERMINISTIC REPLANNING (DDR)

For evaluation purposes, we have implemented DDR, a strategy that performs, in every state, a deterministic replanning, but simulates the first leg of each plan under a snapshot with uncertainty. In other words, DDR computes deterministic plans, and it replans frequently, to alleviate the effect of suboptimal actions caused by ignoring the uncertainty during the planning time.

We have measured the simulated worst-case arrival time. DDR is better than deterministic planning, but not as good as nondeterministic contingent planning (ND). See Table 6 for summary statistics. Differences between ND and DDR in terms of worst-case arrival times can go in both directions, but cases when DDR is better have a much lower frequency. Specifically, when comparing DDR to contingent plans, the percentage of cases favorable to contingent plans varies from 5.43% (Montpellier, $\sigma^2 = 1600$) to 21.09% (Dublin, $\sigma^2 = 6400$). The percentage of cases favorable to DDR is significantly smaller, varying between 0.30% and 1.82%.

14.3 Evaluating the Hybrid Search Approach

As optimizing purely on the travel time is the most computationally challenging scenario (as compared to other linear combinations between the number of legs and the travel time), we focus on improving the performance in this scenario. The optimality criterion is minimizing the worst-case travel time, with ties broken in favor of a better expected travel time.

Figure 10 shows the CPU time in the hybrid method, denoted by HYBRID, compared to the standard DIJA, which uses AO*, but not A* search. Dots under the main diagonal are cases where HYBRID is faster, and dots above the diagonal show the opposite. The three parallel lines correspond to the main diagonal $y = x$, $y = 3x$ (above the main diagonal) and $y = \frac{1}{3}x$ (below the main diagonal). These will help better understand the results.

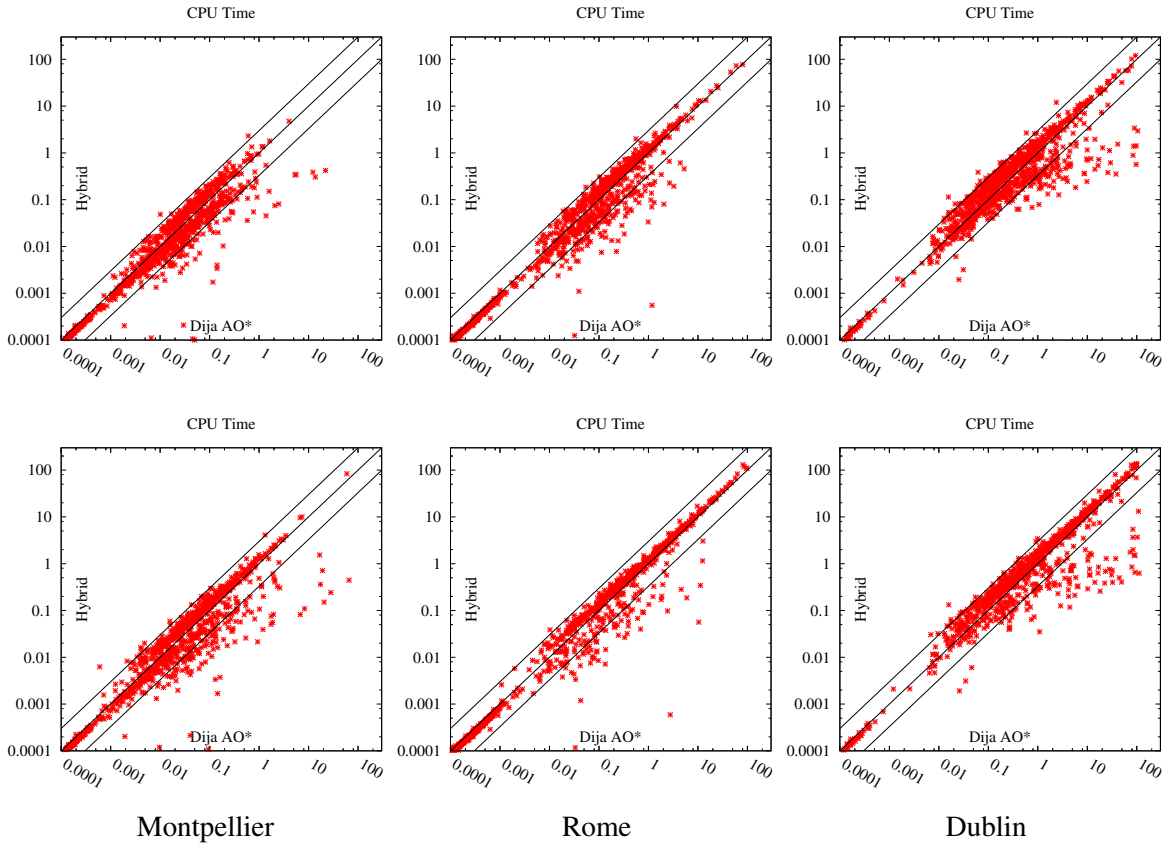

Figure 10: CPU time in HYBRID and DIJA AO* on a logarithmic scale. Top row: $\sigma^2 = 1600$; bottom row: $\sigma^2 = 6400$.

A main conclusion from Figure 10 is that, when HYBRID is faster, it can be faster by a large margin, especially in solving difficult instances. The speedup can significantly exceed one order of magnitude and, in a few cases, two orders of magnitude. On the other hand, when HYBRID is slower, its slowdown is bounded by a factor of 3 in most but not all cases (see the $y = 3x$ line above the main diagonal line). We call this the *asymmetric speedup behavior*.

This (not strict) bounding factor of 3 is present because, in HYBRID, each run invokes at most three time-consuming operations: the A* search ("HYBRID A*"), the back-propagation, and the AO* search ("HYBRID AO*"). Each of these is typically smaller than the standalone AO* search ("DIJA AO*"), as follows. The A* search has a smaller space to explore, being a deterministic search. Figure 11 (left) shows differences between A* and AO* search, both with the initial heuristic in use. The back-propagation traverses A*'s search space, being thus comparable with A* as CPU time. The AO* in HYBRID is typically faster than the DIJA AO*, thanks to the better heuristic obtained through the back-propagation phase. Figure 11 (middle) shows the impact of the improved heuristic on AO* search.

We have also evaluated a partial version of HYBRID, with back-propagation turned off. In other words, when the deterministic plan is invalid in the non-deterministic domain, the heuristic

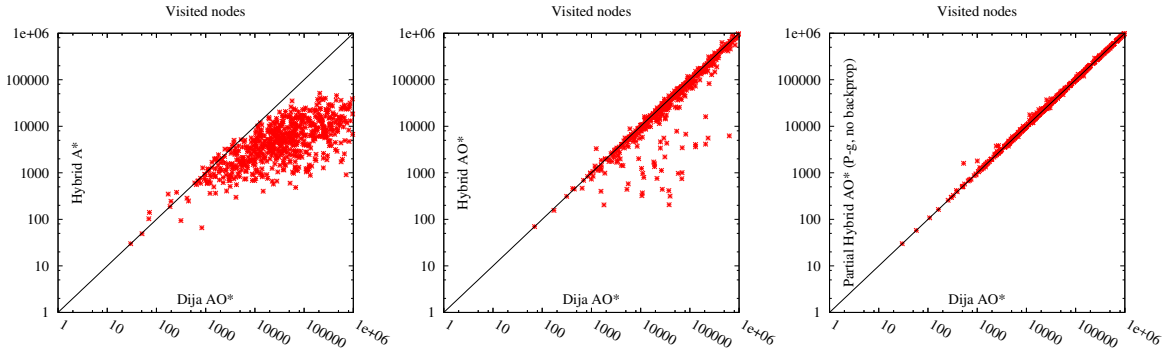

Figure 11: Left: A* in HYBRID vs DIJA AO*; middle: AO* in HYBRID vs DIJA AO*; right: AO* with a partial heuristic update strategy vs AO* with the original heuristic. Data shown for Rome with $\sigma^2 = 6400$. Other combinations (city, noise level) lead to a similar conclusion.

is updated using only the P-g rule, without any back-propagation. Comparing the middle and the rightmost charts in Figure 11 convincingly illustrates the benefits of back-propagation.

In the rightmost chart, we observe that, in our problems, the P-g rule alone does not change the AO* performance significantly. In particular, this implies that, for the subset of instances where A* alone is not sufficient, a hybrid approach without back-propagation would consistently be slower than a pure AO* solver. Indeed, in such cases, the system has the additional overhead of running A*. On the other hand, the full heuristic update method, with both P-g and back-propagation switched on, performs significantly better than the original heuristic, as shown in the middle chart in Figure 11.

15. Summary and Future Work

Deterministic planning is a de facto standard in multi-modal journey planning, both in the literature and across deployed systems. We have presented an optimal approach to computing uncertainty-aware, contingent plans for multi-modal journey planning. We have shown how to model the problem as a search in an And/Or state space. We have described algorithmic contributions, implemeted on top of AO*, such as admissible heuristics, multiple types of pruning that preserve the completeness and the optimality of the solver, and a hybrid approach combining A* and AO* search. We have proved an NP-hardness result where the hardness is due to the presence of dynamically changing distributions or random variables representing travel times. A detailed empirical analysis demonstrates that our system is fast and scalable, and that contingent plans can be more beneficial than standard sequential plans.

In future work, we plan to further improve the scalability, and to extend our theoretical analysis. We plan to extend the scale of the problems addressed from the city-level to larger areas.

References

Akagi, Y., Kishimoto, A., & Fukunaga, A. (2010). On transposition tables for single-agent search and planning: Summary of results. In *Proceedings of the 3rd Symposium on Combinatorial*

Search (SOCS), pp. 1–8.

Albore, A., Palacios, H., & Geffner, H. (2009). A translation-based approach to contingent planning. In *Proceedings of the 21st International Jont Conference on Artifical Intelligence*, IJCAI'09, pp. 1623–1628, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Barto, A. G., Bradtke, S. J., & Singh, S. P. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence*, *72*(1-2), 81–138.

Bast, H., Delling, D., Goldberg, A. V., Müller-Hannemann, M., Pajor, T., Sanders, P., Wagner, D., & Werneck, R. F. (2016). Route planning in transportation networks. In *Algorithm Engineering*.

Bast, H., Sternisko, J., & Storandt, S. (2013). Delay-Robustness of Transfer Patterns in Public Transportation Route Planning. In Frigioni, D., & Stiller, S. (Eds.), *ATMOS - 13th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*, Vol. 33 of *OpenAccess Series in Informatics (OASIcs)*, pp. 42–54.

Bertsimas, D., Brown, D. B., & Caramanis, C. (2011). Theory and applications of robust optimization. *SIAM Rev.*, *53*(3), 464–501.

Bhattacharya, S., Roy, R., & Bhattacharya, S. (1998). An exact depth-first algorithm for the pallet loading problem. *European Journal of Operational Research*, *110*(3), 610–625.

Bonet, B., & Geffner, H. (2000a). Planning with incomplete information as heuristic search in belief space. In *Artificial Intelligence Planning Systems (AIPS)*, pp. 52–61. AAAI Press.

Bonet, B., & Geffner, H. (2000b). Planning with incomplete information as heuristic search in belief space. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems*, pp. 52–61.

Bonet, B., & Geffner, H. (2003). Faster heuristic search algorithms for planning with uncertainty and full feedback. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1233–1238.

Bonet, B., & Geffner, H. (2005). mGPT: A Probabilistic Planner Based on Heuristic Search. *Journal of Artificial Intelligence Research*, *24*, 933–944.

Botea, A., Berlingerio, M., Bouillet, E., Braghin, S., Calabrese, F., Chen, B., Gkoufas, Y., Laummans, M., Nair, R., & Nonner, T. (2014). Docit: An integrated system for risk-averse multimodal journey advising. Tech. rep., IBM Research, Dublin, Ireland.

Botea, A., & Braghin, S. (2015). Contingent versus deterministic plans in multi-modal journey planning. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 268–272.

Botea, A., Nikolova, E., & Berlingerio, M. (2013). Multi-modal journey planning in the presence of uncertainty. In *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 20–28.

Bouillet, E., Gasparini, L., & Verscheure, O. (2011). Towards a real time public transport awareness system: case study in dublin. In Candan, K. S., Panchanathan, S., Prabhakaran, B., Sundaram, H., chi Feng, W., & Sebe, N. (Eds.), *ACM Multimedia*, pp. 797–798. ACM.

Cimatti, A., & Roveri, M. (2000). Conformant planning via symbolic model checking. *Journal of Artificial Intelligence (JAIR)*, *13*, 305–338.

Culberson, J. C., & Schaeffer, J. (1998). Pattern databases. *Computational Intelligence*, *14*(3), 318–334.

Delage, E., & Mannor, S. (2010). Percentile Optimization for Markov Decision Processes with Parameter Uncertainty. *Operations Research*, *58*(1), 203–213.

Dibbelt, J., Pajor, T., Strasser, B., & Wagner, D. (2013). Intriguingly simple and fast transit routing. In Bonifaci, V., Demetrescu, C., & Marchetti-Spaccamela, A. (Eds.), *Experimental Algorithms*, pp. 43–54, Berlin, Heidelberg. Springer Berlin Heidelberg.

Eyerich, P., Keller, T., & Helmert, M. (2010). High-quality policies for the canadian traveler's problem. In *Proceedings of the 24th National Conference on Artificial Intelligence (AAAI)*.

Fan, Y., Kalaba, R., & J.E. Moore, I. (2005). Arriving on time. *Journal of Optimization Theory and Applications*, *127*(3), 497–513.

Fujino, T., & Fujiwara, H. (1993). A search space pruning method for test pattern generation using search state dominance. *Journal of Circuits, Systems and Computers*, *03*(04), 859–875.

Garey, M., & Johnson, D. (1979). *Computers and Intractability*. Freeman Press, New York.

Goldman, R. P., & Boddy, M. S. (1996). Expressive planning and explicit knowledge. In *Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems (AIPS)*, pp. 110–117.

Hansen, E. A., & Zilberstein, S. (2001). LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, *129*, 35–62.

Helmert, M., Haslum, P., Hoffmann, J., & Nissim, R. (2014). Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the ACM*, *61*(3). Article 16.

Helmert, M., Haslum, P., & Hoffmann, J. (2007). Flexible abstraction heuristics for optimal sequential planning. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 176–183.

Hernández, C., Meseguer, P., Sun, X., & Koenig, S. (2009). Path-adaptive A* for incremental heuristic search in unknown terrain. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 358–361.

Hernández, C., Sun, X., Koenig, S., & Meseguer, P. (2011). Tree adaptive A*. In *Proceedings of the 10th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 123–130.

Hoffmann, J., & Brafman, R. (2005). Contingent planning via heuristic forward search with implicit belief states. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS-05)*, pp. 71–80.

Holte, R., Perez, M., Zimmer, R., & MacDonald, A. J. (1995). Hierarchical A*: Searching abstraction hierarchies efficiently. Tech. rep., Department of Computer Science, University of Ottawa.

Horowitz, E., & Sahni, S. (1978). *Fundamentals of Computer Algorithms*. Computer Science Press.

Kishimoto, A., Botea, A., & Daly, E. (2016). Combining deterministic and nondeterministic search for optimal journey planning under uncertainty. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pp. 295–303.

Koenig, S., Likhachev, M., & Furcy, D. (2004). Lifelong planning A*. *Artificial Intelligence*, *155*(1-2), 93–146.

Kolobov, A., Mausam, & Weld, D. S. (2009). Retrase: Integrating paradigms for approximate probabilistic planning. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1746–1753.

Kurien, J., Nayak, P., & Smith, D. (2002). Fragment-based conformant planning. In *Proceedings of the 6th International Conference on Artificial Intelligence Planning Systems (AIPS)*, pp. 153–162.

Kuter, U., & Nau, D. S. (2004). Forward-chaining planning in nondeterministic domains. In *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI)*, pp. 513–518.

Kuter, U., Nau, D. S., Reisner, E., & Goldman, R. P. (2008). Using classical planners to solve nondeterministic planning problems.. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 190–197.

Liu, L., & Meng, L. (2009). Algorithms of multi-modal route planning based on the concept of switch point. *PFG Photogrammetrie, Fernerkundung, Geoinformation*, *2009*(5), 431–444.

Loui, R. P. (1983). Optimal paths in graphs with stochastic or multidimensional weights. *Commun. ACM*, *26*(9), 670–676.

Mills-Tettey, G. A., Stentz, A., & Dias, M. B. (2006). Dd* lite: Efficient incremental search with state dominance.. In *Proceedings of the 26th National Conference on Artificial Intelligence (AAAI)*, pp. 1032–1038. AAAI Press.

Muise, C., McIlraith, S. A., & Beck, C. (2012). Improved non-deterministic planning by exploiting state relevance. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 172–180.

Nguyen, H.-K., Tran, D.-V., Son, T. C., & Pontelli, E. (2012). On computing conformant plans using classical planners: A generate-and-complete approach. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 190–198.

Nikolova, E. (2010). Approximation algorithms for reliable stochastic combinatorial optimization. In *Proceedings of Approx/Random'10*, pp. 338–351, Berlin, Heidelberg. Springer-Verlag.

Nikolova, E., Brand, M., & Karger, D. R. (2006). Optimal route planning under uncertainty. In Long, D., Smith, S. F., Borrajo, D., & McCluskey, L. (Eds.), *Proceedings of the 16nd International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 131–141. AAAI.

Nikolova, E., & Karger, D. (2008). Route planning under uncertainty: The Canadian Traveller problem. In *Proceedings of the 23rd Conference on Artificial Intelligence (AAAI)*.

Nikolova, E., Kelner, J. A., Brand, M., & Mitzenmacher, M. (2006). Stochastic shortest paths via quasi-convex maximization. In *Lecture Notes in Computer Science 4168 (ESA 2006)*, pp. 552–563, Springer-Verlag.

Nilsson, N. J. (1980). *Principles of Artificial Intelligence*. Tioga Publishing Co, Palo Alto, CA.

Nilsson, N. J. (1968). Searching problem-solving and game-playing trees for minimal cost solutions.. In *IFIP Congress (2)*, pp. 1556–1562.

Nonner, T. (2012). Polynomial-time approximation schemes for shortest path with alternatives. In *Proceedings of the European Symposium on Algorithms, ESA*, pp. 755–765.

Palacios, H., & Geffner, H. (2009). Compiling uncertainty away in conformant planning problems with bounded width. *Journal of Artificial Intelligence*, *35*, 623–675.

Papadimitriou, C., & Yannakakis, M. (1991). Shortest paths without a map. *Theoretical Computer Science*, *84*, 127–150.

Peot, M. A., & Smith, D. E. (1992). Conditional nonlinear planning. In *Proceedings of the 1st International Conference on Artificial Intelligence Planning Systems (AIPS)*, pp. 189–197.

Reinefeld, A., & Marsland, T. (1994). Enhanced iterative-deepening search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *16*(7), 701–710.

Smith, D. E., & Weld, D. S. (1998). Conformant Graphplan. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI)*, pp. 889–896.

Stentz, A. (1995). The focussed D* algorithm for real-time replanning. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1652–1659.

Teichteil-Königsbuch, F., Cedex, T., & Kuter, F. U. (2010). Incremental plan aggregation for generating policies in mdps. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 1231–1238.

Wellman, M. P., Larson, K., Ford, M., & Wurman, P. R. (1995). Path planning under time-dependent uncertainty. In *In Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pp. 532–539. Morgan Kaufmann.

Yoon, S., Fern, A., & Givan, R. (2007). FF-Replan: A baseline for probablistic planning. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 352–359.

Zobrist, A. L. (1970). A new hashing method with applications for game playing. Tech. rep., Department of Computer Science, University of Wisconsin, Madison. Reprinted in *International Computer Chess Association Journal*, 13(2):169-173, 1990.

Zografos, K. G., & Androutsopoulos, K. N. (2008). Algorithms for itinerary planning in multimodal transportation networks. *IEEE Transactions on Intelligent Transportation Systems*, *9*(1), 175–184.