ECAI 2016
G.A. Kaminka et al. (Eds.)
© 2016 The Authors and IOS Press.
This article is published online with Open Access by IOS Press and distributed under the terms of the Creative Commons Attribution Non-Commercial License 4.0 (CC BY-NC 4.0).
doi:10.3233/978-1-61499-672-9-295

Combining Deterministic and Nondeterministic Search for Optimal Journey Planning Under Uncertainty

Akihiro Kishimoto, Adi Botea and **Elizabeth Daly**[1]

Abstract. Optimal multi-modal journey planning under uncertainty is a challenging problem, due in part to an increased branching factor generated by nondeterministic actions. Deterministic search, which ignores all uncertainty, can be much faster, but deterministic plans lack correctness and optimality guarantees in the uncertainty-aware domain.

We present a novel approach that combines the strengths of both deterministic and nondeterministic search in order to achieve superior performance. Initially, an A* search is used checking whether the resulting deterministic plan remains correct and optimal under uncertainty. When the plan is invalid, a backpropagation step through the A*'s search graph improves the initial heuristic while preserving its admissibility. After the backpropagation, an AO* search is run with the new improved heuristic. A theoretical analysis proves that our approach is sound and optimal. Our backpropagation correctly handles a subtle issue arising in the presence of state-dominance pruning. This supports the use of these two powerful speedup techniques in combination, for a better overall performance.

We empirically evaluate our solution in multi-modal journey planning under uncertainty, with realistic data from three European cities. Our results show that our approach brings a significant performance improvement over a state-of-the-art, highly optimized journey planning engine based on AO* search.

1 Introduction

Both domain-independent and dependent-specific planning have been extensively studied for decades in the AI research community. Despite recent improvements in domain-independent planning, domain-specific planning is still necessary, due to a strong demand for efficiency. Additionally, ideas developed in domain-specific planning can often be generalized and transferred to domain-independent planning. Conversely, domain-independent planning algorithms can be specialized into high-performance domain-specific methods.

Multi-modal journey planning has garnered increased attention in recent years, for several reasons. First off, this is driven by an increasing practical need, as multi-modal travel can potentially contribute to reducing pollution, congestion and carbon emissions. Secondly, an increasing availability of input data, such as public transport schedules, and smart phone technology facilitate the development and the use of journey planning systems.

Traditional multi-modal journey planning systems are deterministic, implicitly assuming that the input data is accurate which makes journey planning relatively straightforward. However, in real life, a transport network can have a great deal of uncertainty, such as significant differences between published schedules and the actual arrival and departure times of buses, trams or trains due to unforeseen delays. This can result in missed connections, which can in turn cause other segments in the journey to become unrealistic. Recent work has investigated multi-modal journey planning under uncertainty [8, 29], as an approach to provide more reliable journey plans [7].

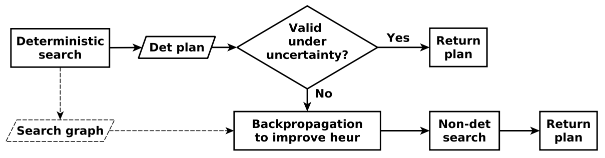

Figure 1. Architecture of our approach.

Given the high volume and time dependent nature of such an application, the speed performance for optimal multi-modal journey planning is extremely important. As a result, the current rational choice of practitioners is the use of domain-specific planning specialized to journey planning. The need for speeding up optimal multi-modal journey planning is particularly important when uncertainty is modeled as part of the problem, for two reasons. First off, deterministic multi-modal journey planning has benefited from more research efforts than the uncertainty-aware problem, being thus a more mature field (see e.g., [3]). Secondly, planning under uncertainty is inherently more computationally difficult.

We present an optimal approach to journey planning under uncertainty, that combines the strengths of deterministic and nondeterministic search, as illustrated in Figure 1. A deterministic solver, based on A* [13] search with pruning enhancements, provides a deterministic plan. This is followed by a so-called validity test, to decide whether the deterministic plan remains correct and optimal in the nondeterministic domain. If the test is positive, the nondeterministic problem has been solved with a standard deterministic search, which is typically faster than nondeterministic search. On the other hand, when the validity test fails, a backpropagation step through the A*'s search graph computes new (improved) heuristic estimations for the states visited in the search. Then, a nondeterministic search, based on AO* [27, 28] search with pruning enhancements, is run using the more accurate heuristic. The new heuristic improves the performance due to the knowledge gathered during the original deterministic search.

The high-level idea summarized in the previous paragraph and illustrated in Figure 1 is domain-independent. However, we develop it and study its use in an application-specific context, as our motivation was advancing the state-of-the-art in multi-modal journey

[1] IBM Research, Ireland

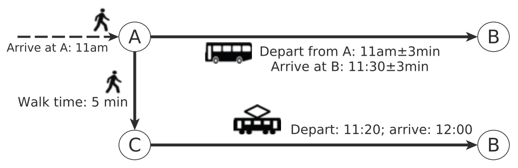

Figure 2. Toy example.

planning under uncertainty. Creating components such as an effective backpropagation technique is not a trivial task. In addition, combining our contributions with a state-of-the-art AO*-based solver introduces a number of challenges to address. Part of the reason is that the highly-optimized A* and AO* search engines use pruning rules, including state-dominance pruning [8]. As discussed in the paper, there are important differences between deterministic and nondeterministic search regarding the conditions under which state-dominance pruning can safely applied. Transferring informed knowledge from a deterministic domain relaxation to the nondeterministic domain, as done with our backpropagation algorithm, needs a special attention, to handle a subtle issue that may otherwise affect the optimality.

We give a theoretical analysis which proves that our approach is sound and optimal. The proposed solution makes it feasible to harness two powerful speedup techniques (i.e., our approach and dominance pruning) in tandem, for a better overall performance.

We use Botea et al.'s domain modeling [8], a state-of-the-art approach to multi-modal journey planning under uncertainty. As discussed later, the domain is represented as a state space that allows nondeterministic transitions. A plan is a probabilistic contingent plan (or policy in MDP terms). Figure 2 shows a toy problem. A user arrives (e.g., with another connection) at a point A by 11am. There are two options to continue to the destination B: take a fast bus directly; or walk to a nearby tram stop C, and take a tram from there. The first option is faster but, due to uncertainty in the bus departure time, it may or it may not be possible to catch the bus route. As such, the take-bus action has two possible nondeterministic outcomes (branches): "success" and "missed bus" (failure). In general, the probability of each possible outcome is computed from the probabilistic times of the user arrival and the bus departure [8]. In our example, the optimal contingent plan is as follows: At location A, attempt to take the bus. If the bus is missed, walk to the tram stop C and take the tram.

We experimentally evaluate our proposed solution demonstrating the contribution with realistic multi-modal transportation data from three European cities, a testbed reused from the literature [7]. Our approach achieves significantly improved performance, compared to a state-of-the-art approach based on AO* search and on search enhancements such as those introduced by Botea et al. [8].

2 Related Work

We start with an overview of research in nondeterministic planning, followed by surveying previous research in deterministic search.

2.1 Nondeterministic Planning

Variations of planning under uncertainty include conformant planning [11, 32], contingent planning [30], and probabilistic contingent planning, with differences stemming from the assumptions on the properties of the actions and the observability (sensing capabilities). See e.g., [4] for a 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 remark [4], 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 [2]. 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. [8]. Hansen and Zilberstein present LAO* [12], 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, research has often focused on suboptimal approaches, for a better speed and scalability. Hoffmann and Brafman [18] use Weighted AO*. Bonet and Geffner take an approach based on a greedy, real-time action selection [4, 5, 6]. 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 [22, 23, 35, 24, 21, 34, 25, 26]. For example, Kuter et al. [24] 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 [35] invokes a deterministic planner for the initial state and for the states in the plan where unexpected outcomes are observed in the nondeterministic scenario.

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 backpropagation and run full-scale AO*. To our knowledge, our approach is the first to combine deterministic and nondeterministic search in this way with evidence that this approach solves realistic, difficult journey planning problems fast and optimally.

2.2 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*) [19], pattern databases [10], and merge-and-shrink abstractions [14, 15]. 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 P-g heuristic [19]. 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. There are multiple notable differences between our approach and HA*. First, we consider domains with multiple types of "consumable resources", such as the a maximum walking time and a 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 [33, 20]. 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 [16, 17]. Caching techniques used for this purpose are similar to those featured in HA*, reviewed earlier.

3 Preliminaries

We start with a description of the multi-modal journey planning problem. Then we discuss how the problem is converted into a nondeterministic planning problem. We give definitions, optimization criteria and assumptions needed in the formal analysis. **ND** and **D** refer to the nondeterministic and deterministic domains, respectively.

3.1 Input Data in Multi-modal Journey Planning

A multi-modal journey planner takes as input a user request ("the instance") and a network transport snapshot ("the domain").

The request includes the origin, the destination, the departure or the arrival time,[2] and the transport modes acceptable in the journey at hand. It also includes so-called *quotas*, which are max acceptable values for the walking time, the cycling time, and the number of legs (segments) in the trip.

[2] In this work we focus on scenarios where the user request specifies a departure time rather than an arrival time.

The network snapshot contains information available about a multi-modal transportation network and includes the following data:

- *Relevant locations* include public-transport stops, bike stations in a city's shared-bike network along with the location of the origin and destination of the request. All relevant locations have their lat and lon coordinates specified, besides their type (e.g., stop or bike station), name and id.
- A *public-transport route* can be represented as an ordered sequence of stops. Each route is served by several *trips* during the day. A trip has an arrival and departure time associated with every stop along the route. Traditionally, these times are deterministic. However, we allow a more general representation, where a departure or arrival time is a probability distribution.
- *Road map data* is represented as a graph with nodes and segments. Each segment is labeled to reflect its direction (e.g., one-way or bi-directional), access to cyclists, access to pedestrians, access to cars and driving speed.

Next we discuss how such input data is converted into a nondeterministic state space.

3.2 Nondeterministic Domain Modeling

We adopt the formalization introduced by Botea et al. [8] for multi-modal journey planning under uncertainty. For a self-contained presentation, we summarize how the problem is formalized here as a nondeterministic state space, defining states and transitions.

A state s is a tuple $(p_s, t_s, q_s, \alpha_s)$ where:

- p_s is the position of the traveler, which is either a relevant location on the map, or the id of a trip, for those states where the user is aboard a trip (e.g., a bus);
- t_s is a probability distribution of the time when the user has reached position p_s;
- q_s is a vector which lists quotas left in this state (e.g., max walking time and max number of legs in the rest of the trip);
- α_s is a subset of additional variables which are skipped for brevity.

The initial state s_0 is constructed with the components $p_{s_0}, t_{s_0}, q_{s_0}$ taken from the user request, representing the origin, the departure time and the quotas acceptable for the entire trip respectively. A valid state s is not allowed to have negative values on any component of the quota vector q_s. A state s_G is a goal state if it is valid and p_{s_G} corresponds to the destination specified in the user request. Depending on the user preferences, other conditions may be added to the goal state, such as not having a hired bike in possession in a goal state (i.e., if the user hires a bike, the bike should be eventually returned to a bike station). For clarity, we assume that this condition is part of the goal definition as well.

It is common in nondeterministic planning research to group regular states together into a *belief state*, using for instance decision diagrams to represent belief states. This choice seen in related work is motivated by two reasons: to cover domains with partial observability, where a belief state could contain all possible current states; and to mitigate state explosion, obtaining a more compact state space definition. Botea et al.'s modeling [8], which we adopt in this research, works with regular, not belief states, one of the reasons being that states are assumed to be fully observable during the plan execution.

Transitions include `Walk`, `TakeTrip` (i.e., boarding a public transport vehicle), `GetOffTrip`, `HireBike`, `Cycle` and

ParkBike. Except for TakeTrip actions, all transitions are deterministic, in the sense that they always succeed and thus each of them has exactly one successor. As illustrated earlier in Figure 2, a TakeTrip action can succeed or fail, depending on the possibly uncertain arrival times of the vehicle and the traveler. Botea et al. show how to compute the probability of each outcome [8]. Obviously, when the probability of success is 1, the TakeTrip action at hand becomes deterministic, with only the successful branch defined. When the probability is 0, no TakeTrip action is defined for that particular trip in that particular state.

On the successful branch, the successor state s of a TakeTrip action has the position p_s set to the id of the trip just boarded. The time t_s is the (stochastic) departure time of the trip from the stop at hand. On the failed branch, the position and the time remain as in the parent state. However, in the successor state, a new flag is set to true, indicating that the trip at hand has just been missed. As such, the parent and the successor state are different, and therefore the failed branch is not a self-loop transition. This property, together with the fact that the time is a state component, ensures that there are no cycles in this state space. In the search, the space is modeled as a tree, rather than a directed acyclic graph.

For brevity, we skip discussing how other transitions generate their successor states. Such details are not difficult to infer based on the discussion provided, and the interested reader can refer to the original work [8].

A state s dominates a state s', i.e., $s \prec s'$, iff $p_s = p_{s'}$, $q_s \geq q_{s'}$ component-wise, and $P(T_s \leq T_{s'}) = 1$. T_s is the random variable with the density function t_s. In other words, the position is the same, and one state is no worse than the other in terms of available quotas and time. The relation is not symmetric, apart from the obvious exception that every state dominates itself.

$A_{nd}(s)$ is the set of all actions applicable to a state s in **ND**. For a state s and an action $a \in A_{nd}(s)$, $B(s,a)$ is the set of all possible branches. I.e., $|B(s,a)| = 1$ for deterministic actions, and $|B(s,a)| = 2$ for nondeterministic actions with just two outcomes (i.e., succeed and fail). Each branch b uniquely determines a transition to a successor state. It has a probability p_b as illustrated earlier, a cost $c_b(s,a)$ associated with the transition,[3] and a successor state $\gamma_b(s,a)$. As the deterministic action has a unique branch, in their case the notations get simplified into $c(s,a)$ and $\gamma(s,a)$. Thus, deterministic actions are a particular type of successful branches.

3.3 Deterministic Domain

We obtain the 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. $A_{det}(s)$ is the set of actions applicable to a state s in **D**. 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**. Such an optimistic approach is one of the conditions that we need to ensure the optimality of plans in our approach, as discussed later in the paper. In particular, we will make use of the following straightforward observation.

Observation 1. *All successful branches in* **ND** *are available as deterministic actions in* **D**.

[3] The cost can depend on both action a and state s, 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".

3.4 Optimality Criteria

The cost function $c(n,m)$ returns a non-negative value as a state transition cost from state n to state m. Additionally, $f(n)$, $g(n)$ and $h(n)$ represent an *f-value*, a *g-value*, and a *h-value* (or heuristic value) at state n, respectively. The f-value is defined as $f(n) = g(n) + h(n)$, where $g(n)$ is the sum of the transition costs from the initial state to reach n, and $h(n)$ is a value estimating a cost to reach the destination from n. In this work, the cost is set to the travel time.

The optimal cost $v_{det}(n)$ of a state n in **D**, the optimal expected cost $v_{exp}(n)$ in **ND**, and the optimal worst-case cost $v_{wst}(n)$ in **ND** are defined as:

1. If n is a goal state, $v_{det}(n) = v_{exp}(n) = v_{wst}(n) = 0$.
2. If $|A_{det}(n)| = 0$, $v_{det}(n) = v_{exp}(n) = v_{wst}(n) = \infty$.
3. Otherwise,
   - $v_{det}(n) = \min_{a \in A_{det}(n)} (c(n,a) + v_{det}(\gamma(n,a)))$
   - $v_{exp}(n) = \min_{a \in A_{nd}(n)} \sum_{b \in B(n,a)} p_b(c_b(n,a) + v_{exp}(\gamma_b(n,a)))$
   - $v_{wst}(n) = \min_{a \in A_{nd}(n)} \max_{b \in B(n,a)} (c_b(n,a) + v_{wst}(\gamma_b(n,a)))$.

Note that there are many goal states for one goal, since the time and quotas are part of the state definition.

We use Botea et al.'s objective function [8], which minimizes the worst case, and break ties in favor of better expected costs (i.e., take (v_{wst}, v_{exp}) in the lexicographic order):

Definition 1. *The cost in* **ND** *of a plan π is the pair $v_{opt}(\pi) = v_{opt}(r) = (v_{wst}(r), v_{exp}(r))$, where r is the root node of the plan.*

We write $(p_1, p_2) \leq_{lex} (q_1, q_2)$ iff $(p_1 < q_1) \vee (p_1 = q_1 \wedge p_2 \leq q_2)$. The inequality is strict when the two pairs are not identical. We can use other objectives (e.g., swap the lexicographic order [7]) with minor changes, but this is beyond our focus.

We list a few additional facts relevant to our analysis:

Proposition 1 ([8]). *In an optimal plan, the cost when following the failed branch of an action cannot beat the cost along the successful branch.*

Observation 2 ([8]). *Let a be an action with two nondeterministic outcomes in a correct plan π. Cutting action a from π, plus the entire subtree along the successful branch, results in a correct plan.*

The example shown in Figure 2 helps see the intuition behind these two claims. The part of the plan under the failed branch of the take-bus action is "walk to C and then take the tram", as these are the actions taken after failing to catch the bus. Regarding Proposition 1, if this part of the plan under the failed branch had a better cost (i.e., better arrival time) then the bus trip, there would be no point to even attempt to take the bus. Regarding Observation 2, indeed, we can eliminate the attempt to take the bus, together with the part under the successful branch (i.e., ride the bus to B), and still obtain a perfectly valid (but not necessarily optimal) plan. This plan would be "...arrive at A, walk to C and take the tram to B." More formally, this stems from the fact that we compute *strong (acyclic) plans* [9], meaning every pathway in a contingent plan ends up at the destination. See the original work for more formal proofs of these two claims.

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

In particular, this holds for Botea et al.'s heuristic [8], which we reuse in experiments as the initial heuristic available.

4 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 have observed similar properties in their work on MDPs and probabilistic planning [4, 6]. The discussion presented in this section is important to ensure the optimality of our hybrid approach.

Theorem 1. *For any state n, $v_{det}(n) \leq v_{exp}(n) \leq v_{wst}(n)$.*

Proof Sketch (for $v_{det}(n) \leq v_{exp}(n)$). Let π be a plan in **ND**, rooted at n, that is optimal on v_{exp} (i.e., $v_{exp}(\pi) = v_{exp}(n)$). Let p be a smallest-cost pathway in π. Consider all nondeterministic actions having their failed branch within p. Remove these actions from π, together with the subtree under their successful branch, obtaining a plan π' that is correct in **ND**, cf. Observation 2. In π', p has lost zero or more failed branches (let p' be the new pathway). As p' has only successful branches, it is a correct plan in **D**, cf. Observation 1. Clearly, $v_{det}(p') \leq v_{exp}(\pi) = v_{exp}(n)$, since p was a best-cost pathway in π. Furthermore, $v_{det}(n) \leq v_{det}(p')$, since the former is the optimal value in **D**, and the latter is the cost of one correct plan in **D** (which may be optimal or not).

□

Theorem 2. *Any $h(n)$ admissible in* **D** *is admissible in* **ND**.

Proof Sketch Let $h_{nd}(n) = (h(n), h(n))$ and $v_{opt}(n) = (v_{wst}(n), v_{exp}(n))$. It follows from Theorem 1 that $h_{nd}(n) = (h(n), h(n)) \leq_{lex} (v_{det}(n), v_{det}(n)) \leq_{lex} (v_{wst}(n), v_{exp}(n)) = v_{opt}(n)$. □

Corollary 1. *The perfect heuristic in* **D** *$h^*(n) = v_{det}(n)$ is admissible in* **ND**.

These results show that we can use an optimal cost in **D** or a lower-bound to admissibly guide the search in **ND**. We emphasize that the previous cost definitions 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.

5 Hybrid Approach

Algorithm 1 Hybrid Deterministic Nondeterministic Search

```
Require: Initial state $root$
1: $O = C = H = \phi$
2: $(v_{det}, \pi_1)$ = A*($root$, $O$, $C$, $h$)
3: if (all actions in $\pi_1$ are deterministic in ND) then
4:     return $\pi_1$
5: else
6:     Update($root$, $O$, $C$, $H$, $v_{det}$)
7:     return AO*($root$, $\max(H, h)$)
```

We now introduce our new algorithm, whose main steps are illustrated in Algorithm 1 and Figure 1. Our approach first runs A* with an open list O and a closed list C, returning an optimal plan π_1 in **D**, as well as its cost v_{det}. Unlike standard A*, our A* takes into account the state dominance (see Algorithm 2). Assume that A* generates a successor s and detects that there is a state u in $O \cup C$ such that $u \prec s$. Then, A* discards s, since both $v_{det}(u) \leq v_{det}(s)$ and $g(u) < g(s)$ always hold in the deterministic scenario. That is, A* does not explore further the search space rooted at s. This is plays a crucial role in significantly reducing A*'s search space.

Similarly to A*, AO* performs pruning with state dominance. However, Botea et al. [8] have pointed out that correctly applying state dominance pruning in a nondeterministic search requires additional conditions to satisfy, and they presented sufficient conditions for this purpose. Our AO* implementation observes these.

Algorithm 2 A* with state dominance pruning

```
Require: State $n$, open list $O$, closed list $C$ and consistent heuristic
    function $h$
1: Enqueue($O$, $n$, $h(n)$)
2: while $O \neq \emptyset$ do
3:     $t$ = Dequeue($O$)
4:     if $t$ is a goal then
5:         return $(f(t), path(n, t))$
6:     Save($t$, $C$)
7:     for each of $t$'s successors $s$ do
8:         if $s \notin C \land$ no state $u \in C \cup O$ dominates $s$ then
9:             Enqueue($O$, $s$, $g(t) + c(t, s) + h(s)$)
10: return $(\infty, \emptyset)$
```

Theorem 3. *Consider a deterministic plan π_1 computed in the deterministic search. If all π_1's actions a correspond to a deterministic action in* **ND** *(as opposed to a being the relaxation of a nondeterministic action in* **ND***), then π_1 is correct and optimal in* **ND**.

Proof Sketch 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 1 to r_2, we obtain $v_{det}(r_2) \leq v_{exp}(r_2) \leq v_{wst}(r_2)$. At the same time, $v_{det}(r_1) = v_{exp}(r_1) = v_{wst}(r_1)$, 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 $v_{det}(r_1) \leq v_{det}(r_2)$. Putting all these together, it follows that $v_{opt}(r_1) = (v_{wst}(r_1), v_{exp}(r_1)) = (v_{det}(r_1), v_{det}(r_1)) \leq_{lex} (v_{det}(r_2), v_{det}(r_2)) \leq_{lex} (v_{wst}(r_2), v_{exp}(r_2)) = v_{opt}(r_2)$, which is a contradiction.

The previous result is important because it formalizes the validation step in our approach (line 3 of Algorithm 1). When the validity test fails, our approach updates the heuristic, after which it runs AO*.

The Update method shown in Algorithm 3 implements the backpropagation idea, computing more informed h-values that are stored in a table H. As in related previous work [31, 1], Update traverses the search graph backwards and sets $H(n) = \min_{s_i \in C(n)}(f(s_i) - g(n))$, where $C(n)$ is the set of not-expanded frontier states reachable from n. 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*.

As mentioned, both A* and AO* apply dominance pruning. 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 $u \in O \cup C$. Both A* and Update skip examining s due to the pruning scheme with state dominance (see line 8 in Algorithm 2 and line 5 in Algorithm 3). However, while regarding s as a deadend in **D**

Algorithm 3 Update

```
Require: State $n$, open list $O$, closed list $C$, hash table $H$, and threshold $\theta$
 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 $\theta$ */
13: $r = \max(r, \theta)$
14: /* Save an improved heuristic value */
15: Save($n$, $r$, $H$)
16: return $r$
```

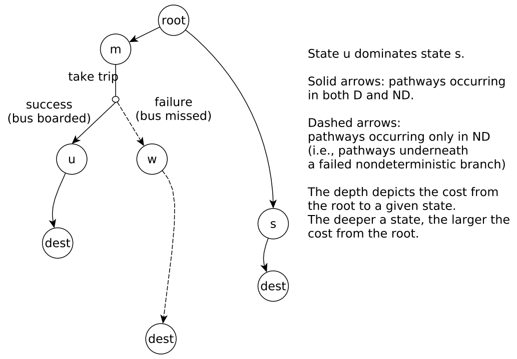
root
m
take trip
success (bus boarded)
failure (bus missed)
u
w
s
dest
dest
dest
State u dominates state s.
Solid arrows: pathways occurring in both D and ND.
Dashed arrows: pathways occurring only in ND (i.e., pathways underneath a failed nondeterministic branch)
The depth depicts the cost from the root to a given state. The deeper a state, the larger the cost from the root.

Figure 3. A simple example where a dominated state in **D** belongs to an optimal plan in **ND**

is correct in A*, 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 3). We illustrate why aggressively setting $H(s) = \infty$ in Update would be a mistake with the following example.

Example 1. *Figure 3 illustrates a scenario where a state s dominated in* **D** *belongs to an optimal plan in* **ND**. *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 3 is related to Holte et al.'s P-g heuristic [19], discussed in the related work section. Backpropagation (without line 13) provides better estimates than P-g (line 13 alone) in many cases. Interestingly, while experiments presented later show that P-g alone is not effective in this domain, we found that P-g is helpful in combination with our backpropagation. Specifically, 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, backpropagation (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), backpropagation 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 n is processed in Update, we have $\theta = v_{det} - g(n)$. 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 3); 2) It is known that, at the end of an A* search that returns an optimal cost v_{det}, $v_{det} - g(n)$ is an admissible heuristic for every state n with an optimal g-cost.

Our heuristic update strategy implements a few additional ideas. As shown in Algorithm 1, we take the $\max(h(n), H(n))$ as the heuristic to use in AO*. The reason is that, for those states n not visited in A*, $H(n)$ 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 3. Thirdly, we added to AO* (both as a module of our method 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 Proposition 1.

Theorem 4. *The hybrid method returns optimal solutions in* **ND**.

Proof Sketch It is sufficient to prove $H(n) \leq v_{det}(n), \forall n$. Then H is admissible in **ND** cf. Theorem 2. With no generality loss, assume that $H(n) = h(n)$ if n is not found in H. Here we only show that $H(n) = \min_{a \in A_{det}(n)}(c(n,a) + H(\gamma(n,a))) \leq v_{det}(n)$. Other rules (e.g., increasing $H(n)$ to θ) were discussed earlier. The proof is related to Akagi et al.'s work [1]. Let H_k be the table H after k updates. If $k = 0$, the theorem holds, as nothing new is saved in H, and $h(n) \leq v_{det}(n)$ cf. 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 n at step $k+1$. We have: $H_{k+1}(n) = \min_{a \in A_{det}(n)}(c(n,a) + H_k(\gamma(n,a))) \leq \min_{a \in A_{det}(n)}(c(n,a) + v_{det}(\gamma(n,a))) = v_{det}(n)$. □

6 Experimental Evaluation

We implemented our ideas on top of DIJA [8], a state-of-the-art engine for multi-modal journey planning under uncertainty. It is a highly-optimized, AO*-based planning engine, implementing techniques described by Botea et al. [8]. For a fair comparison, the AO* algorithm is the same in both DIJA and our approach, except for the heuristic used. As described earlier in the paper, our approach first attempts to solve the problem with deterministic A* search. If the resulting plan is not valid in the nondeterministic domain, a backpropagation step provides an improved heuristic to the AO* engine.

The optimality criterion is minimizing the worst-case travel time, with ties broken in favor of a better expected travel time. Botea et al. [7] handle a linear combination between the travel time and the number of legs. However, optimizing purely on the travel time is the most computationally challenging scenario, which is why we focus on improving the performance in this case.

We have used testbed data from the literature [7]. This is 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 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. Combined with two levels of noise and two solvers, this sums up to $3,000 \times 2 \times 2 = 12,000$ runs in total.

Figure 4 shows the CPU time in our method, denoted by "Hybrid", compared to the benchmark state-of-the-art approach DIJA. Dots under the main diagonal are cases where our method 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.

A main conclusion from Figure 4 is that, when our method 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 our method 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 our method, each run invokes at most three time-consuming operations: the A* search ("Hybrid A*"), the backpropagation, 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 5 (left) shows differences between A* and AO* search, both with the initial heuristic in use. The backpropagation traverses A*'s search space, being thus comparable with A* as CPU time. The AO* of our method is typically faster than the DIJA AO*, thanks to the better heuristic obtained through the backpropagation phase. Figure 5 (middle) shows the impact of the improved heuristic on AO* search. The impact of backpropagation will be discussed in more detail later in this section.

Table 1 complements the observations drawn from Figure 4 with summary statistics. For each σ^2 level, Part A reports the total CPU time ratio between DIJA and our method. Our method is consistently better on this metric, as all values reported are greater than 1. This is an important result, showing that the new approach is faster in average on all 6 combinations of a city and a noise level. The average speed is important, for instance, in a server implementation, where a journey planning engine has to answer several queries in a short amount of time. Part B shows the maximum speedup of each system, showing a consistent advantage in favor of the new hybrid approach. Part C reports the percentage of instances where the deterministic plan is valid under uncertainty, and Part D presents the percentage of instances where our method is faster (i.e., dots below the main diagonal in Figure 4). Due to the asymmetric speedup behavior, our method is faster in terms of average solving time (Part A) even in those cases where Part D shows a value lower than 50%.

Table 1. Summary statistics. Easy instances, requiring less than 1 second with the baseline approach, are skipped.

	City	Mon	Rom	Dub
	Uncertainty level: $\sigma^2 = 1600$			
A	CPU time ratio DIJA/Hybrid	5.97	1.07	1.71
B	Max speedup factor Hybrid	166.01	48.77	101.81
	Max speedup factor DIJA	1.51	1.44	1.74
C	Det plan valid %	58.10	27.70	35.80
D	Instances Hybrid faster %	66.66	40.32	49.26
	Uncertainty level: $\sigma^2 = 6400$			
A	CPU time ratio DIJA/Hybrid	2.14	1.26	1.18
B	Max speedup factor Hybrid	269.56	82.18	158.07
	Max speedup factor DIJA	5.12	3.28	1.54
C	Det plan valid %	39.70	10.40	15.40
D	Instances Hybrid faster %	57.6	65.85	25.26

Savings in speed come from both the ability to avoid backpropagation and AO* search, when the deterministic search is sufficient, and from the better informed AO* heuristic when the deterministic search is not sufficient. Note that, given a σ^2 level, Part D shows substantially higher percentages than Part C, confirming that our method is faster substantially more often than just the cases when our method stops after A*.

We have also evaluated a partial version of Hybrid, with backpropagation turned off. In other words, when the deterministic plan is invalid in the nondeterministic domain, the heuristic is updated using only the P-g rule, without any backpropagation. Comparing the middle and the rightmost charts in Figure 5 convincingly illustrates the benefits of backpropagation.

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 backpropagation 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 backpropagation switched on, performs significantly better than the original heuristic, as shown in the middle chart in Figure 5. Even when the system has to perform A*, backpropagation and AO*, the speedups are obtained in AO* due to a better-informed heuristic offsets and even exceed the overhead of the A* and backpropagation steps on average. In particular, this explains why values in Part D are higher than values in Part C in Table 1. Thus, backpropagation plays a significant role in the performance of our system.

7 Conclusions

We presented an approach to nondeterministic planning combining A* and AO* search. We focused on multi-modal journey planning under uncertainty, an application domain where speeding up optimal solving approaches is an important task. Our theoretical analysis shows that our approach creates optimal plans even in the presence of

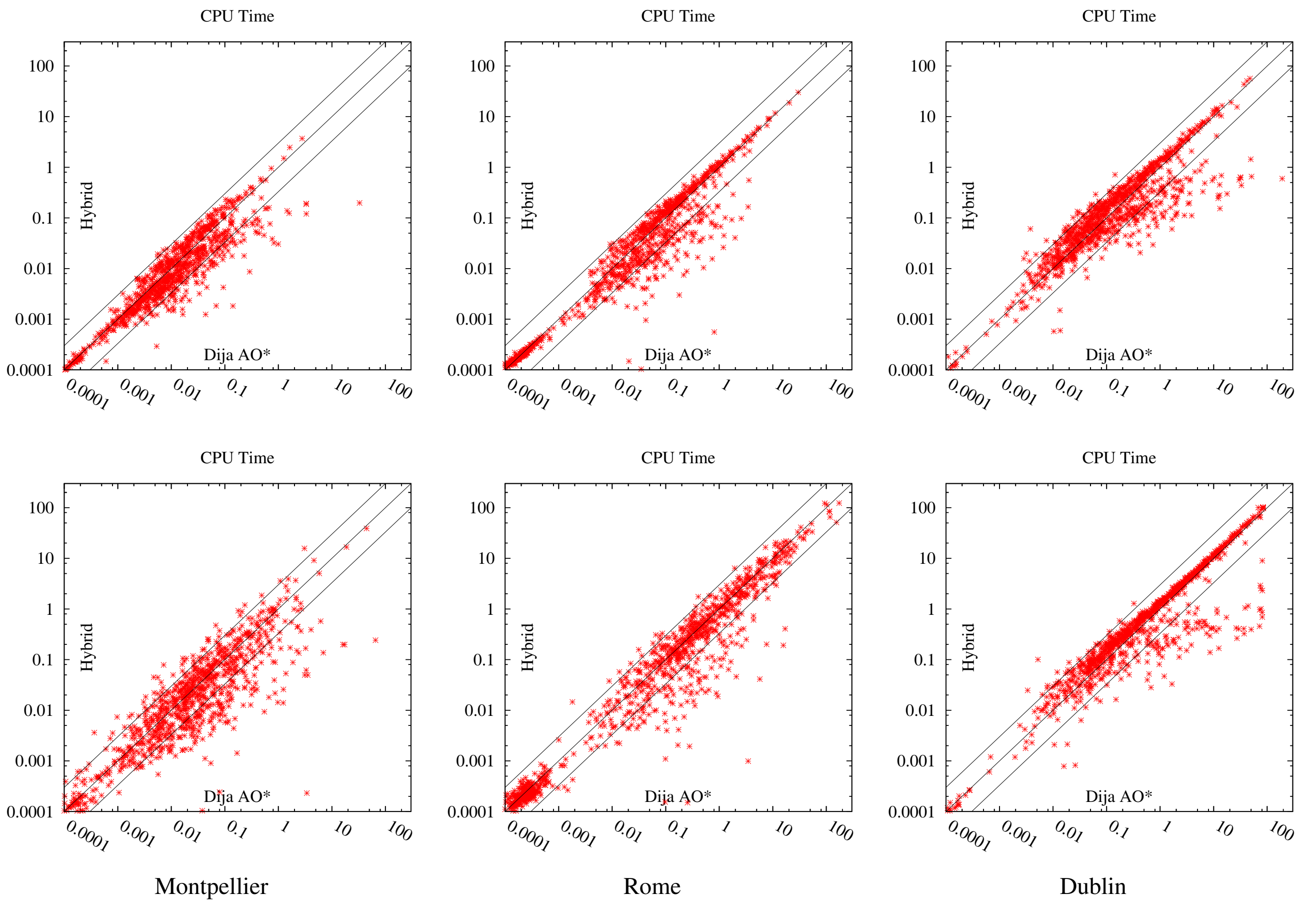

Figure 4. CPU time in our method and Dija AO* on a logarithmic scale. Top row: $\sigma^2 = 1600$; bottom row: $\sigma^2 = 6400$.

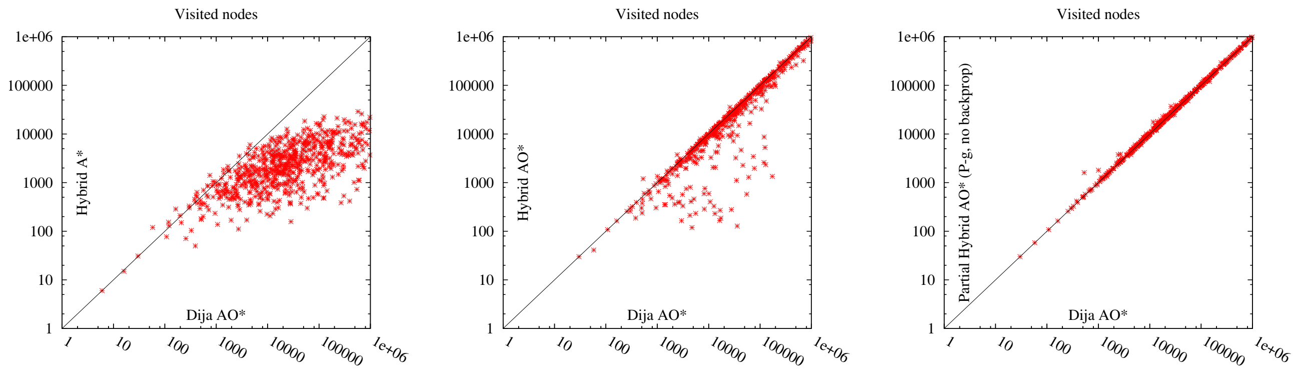

Figure 5. Left: A* in our method vs Dija AO*; middle: AO* in our method vs Dija AO*; right: AO* with a partial heuristic update strategy vs AO* with the original heuristic. Data shown for Rome with $\sigma^2 = 6400$. Other combinations (city, noise level) lead to a similar conclusion and are skipped to save room.

dominance relations between states. Empirical results in multi-modal journey planning under uncertainly demonstrate that our approach brings a significant performance improvement over a state-of-the-art journey planner based on AO*.

Assumptions such as modeling nondeterminism with successful and failed attempts are not limited to journey planning. We plan to develop our method in domain-independent planning, especially in domains with state dominance (e.g., planning under uncertainty with consumable resources). Another interesting direction is to further improve the heuristic function with backpropagation, extending the initial A* search with localized additional explorations.

REFERENCES

[1] Y. Akagi, A. Kishimoto, and A. Fukunaga, '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, (2010).

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

[3] Hannah Bast, Erik Carlsson, Arno Eigenwillig, Robert Geisberger, Chris Harrelson, Veselin Raychev, and Fabien Viger, 'Fast routing in very large public transportation networks using transfer patterns', in *Algorithms - ESA 2010, 18th Annual European Symposium. Proceedings, Part I*, pp. 290–301, (2010).

[4] B. Bonet and H. Geffner, '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, (2000).

[5] B. Bonet and H. Geffner, '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, (2003).

[6] B. Bonet and H. Geffner, 'mGPT: A probabilistic planner based on heuristic search', *Journal of Artificial Intelligence Research*, **24**, 933–944, (2005).

[7] A. Botea and S. Braghin, '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, (2015).

[8] A. Botea, E. Nikolova, and M. Berlingerio, '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, (2013).

[9] A. Cimatti, M. Pistore, M. Roveri, and P. Traverso, 'Weak, strong, and strong cyclic planning via symbolic model checking', *Artificial Intelligence*, **147**(12), 35 – 84, (2003). Planning with Uncertainty and Incomplete Information.

[10] J. Culberson and J. Schaeffer, 'Pattern databases', *Computational Intelligence*, **14**(3), 318–334, (1998).

[11] Robert P. Goldman and Mark S. Boddy, 'Expressive planning and explicit knowledge', in *Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems (AIPS)*, pp. 110–117, (1996).

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

[13] P. E. Hart, N. J. Nilsson, and B. Raphael, 'A formal basis for the heuristic determination of minimum cost paths', *IEEE Transactions on Systems Science and Cybernetics*, **4**(2), 100–107, (1968).

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

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

[16] C. Hernández, P. Meseguer, X. Sun, and S. Koenig, '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, (2009).

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

[18] J. Hoffmann and R. Brafman, 'Contingent planning via heuristic forward search with implicit belief states', in *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS-05)*, eds., Susanne Biundo, Karen Myers, and Kanna Rajan, pp. 71–80, Monterey, CA, USA, (2005).

[19] R.C. Holte, M.B. Perez, R.M. Zimmer, and A. J. MacDonald, 'Hierarchical A*: Searching abstraction hierarchies efficiently', Technical report, Department of Computer Science, University of Ottawa, (1995).

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

[21] A. Kolobov, Mausam, and D. S. Weld, 'ReTrASE: Integrating paradigms for approximate probabilistic planning', in *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1746–1753, (2009).

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

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

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

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

[26] H.-K. Nguyen, D.-V. Tran, T. C. Son, and E. Pontelli, '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, (2012).

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

[28] N. J. Nilsson, *Principles of Artificial Intelligence*, Tioga Publishing Co, Palo Alto, CA, 1980.

[29] T. Nonner, 'Polynomial-time approximation schemes for shortest path with alternatives', in *Proceedings of the European Symposium on Algorithms, ESA*, pp. 755–765, (2012).

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

[31] A. Reinefeld and T. A. Marsland, 'Enhanced iterative-deepening search', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **16**(7), 701–710, (1994).

[32] D. E. Smith and D. S. Weld, 'Conformant Graphplan', in *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI)*, pp. 889–896, (1998).

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

[34] F. Teichteil-Königsbuch, T. Cedex, and F. U. Kuter, '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, (2010).

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