Mitigating environmental uncertainty of autonomous vehicles via shortest-paths in digraph models

November 2, 2025

Abstract

Uncertain environments exert additional challenges to the safety of autonomous vehicles. An example of such an unpredictable element is object detection error. Mitigation techniques include multi-sensor data integration, motion planning, and risk assessment. Upon accommodating multiple scenarios and environments, challenges arise due to scalability concerns and the additional work incurred by building a module for each distribution. We propose a graph-theoretic framework, modeling uncertainty as a vector of random variables for a subset of vertices. The probabilistic expected harm is reduced to finding a shortest path. Neutral digraph zones are queried by a faster subroutine, to reduce the computational time on uncertain zones. Besides correctness proofs, tested emulations on Carla show a reduction in harm caused.

1 Introduction

broad intro to AV. Autonomous vehicles are a central pillar of many workflows in various industries. Yet, safety remains a critical bottleneck of its widespread adoption in public roads. The issue has social and regulatory implications as well. It is crucial to foster public trust, and address communication and transparency among stakeholders. A safe navigation relies on predictable environment alongside accurate sensors' data. Any unexpected environmental events or inaccuracies in sensors endanger people's lives. Understanding their sources, how they affect safety, and mitigating them is crucial for system's robustness and reliability.

uncertainty sources and types. Those unpredictable elements include: weather conditions like rain, snow, fog, and strong winds; Dull lightning conditions; Terrains variations like slopes; Sensors noise as for example in LiDARs by signals distortion in fog; Dynamic movements of objects like vehicles and pedestrians whereby vehicles may take sudden lane changes or stops, and pedestrians may move in an unusual or unpredictable way [**alharbi2020global**]. Technically, uncertainty could be in misclassifying objects, inaccurate positioning, unpredictable trajectories, or in sudden appearance of objects. The focus of this paper is in the first two categories of uncertainties.

implications of uncertainty while neglecting or avoiding it. If a decision-making mechanism doesn't consider uncertainty, then this could clearly lead to unacceptable paths. A child may be hit because it is misclassified as a plastic bag [**tahir2024object**]. A planner may hit an object due to distance measurements inaccuracies [**itkina2022uncertainty**] whereby an object is reported to be more faraway than its actual position. If the mechanism aims to avoid uncertainty by being more conservative, then it is likely to complicate the path to avoid any potential threat. In some cases, a path avoiding all uncertainty may not be existing. Even if it exists, it needs to resolve more constraints incurring more computational time.

literature approaches of reasoning under uncertainty. These uncertainties necessitate algorithms and modeling techniques capable of reasoning under uncertainty. Ongoing efforts in the literature include: Multi-sensor data integration [**vargas2021overview**] where it addresses the limitations of individual sensors, such as Global Navigation Satellite Systems (GNSS) inaccuracies in urban canyons or LiDAR in adverse weather conditions, by merging data from various sensors; Motion planning [**tang2022driving**] where new developments are in path density adjustments and the integration of hierarchical motion planning methods; Risk assessment [**wang2021risk**] where a risk index is estimated based on various indicators like visibility and the presence of obstructions, allowing for dynamic updates in decision-making strategies; Probabilistic optimization frameworks [**han2022non**] where the uncertainty of environmental conditions and obstacles are modeled. Those enable algorithms to generate risk-bounded decisions minimizing the likelihood of undesirable actions, as in trajectory planning.

challenges of literature approaches. In an environment, some factors are completely known with total confidence, while others are uncertain. Their unpredictability degrees could range from low to high in the same environment. In practice, optimizing for these multiple degrees of uncertainty often requires considering each distribution or scenario separately, solving each with a distinct solution. For example, in [**OnboardMitigation˙2**] the planner is designed to be conservative to handle perception errors. The bounding boxes of all objects, regardless of perception accuracy, are enlarged. Similarly in [**OnboardMitigation˙3**], an estimation or assumption of the perception uncertainty is required. In [**OnboardMitigation˙4**] an uncertainty adaptive planner is designed to estimate the degree of uncertainty. Therefore, in order to accommodate different distributions, and environmental factors, in a single module, a hybrid approach is frequently employed, where specialized models or methods are combined to handle various smaller sub-problems. However, the hybrid approach introduces several drawbacks: Scalability issues arise [**scalabilityML**] because the complexity of the solution increases as multiple specialized solutions are integrated, to account for more factors or constraints; Additional time and cost are incurred due to designing, implementing, testing, and maintaining multiple models. More efforts are required for the integration and maintainability of the system as a whole; Reduced interpretability [**UncertainSurvey**], as it is more difficult to analyze or explain decision-making processes, based on multiple specialized solutions working together. Tracing would require analyzing each specialized module and how their outcomes are combined to produce the end result; Moreover, The effectiveness of data-driven models heavily depends on the quality of available datasets, as poor or biased data won't result in a robust agent. So accommodating multiple distributions and scenarios shall raise the concern of finding quality datasets for the whole case-study in hand; The trade-off among multiple objectives and scenarios in hybrid solutions requires a careful tuning on behalf of the designer, as favoring one objective could tamper others.

contribution. In this work, we tackle object misclassification and inaccurate positioning uncertainties. We adopt graph theoretic modeling of the environment, where the vehicle's decision is a path over the digraph, computed algorithmically. It is independent of any particular scenario, dataset, environment knowledge. We choose probabilistic expectation as a quantitative measure for the path planning in uncertain environments. It captures both the likelihood of objects classification alongside associated penalties of their priorities to the planner. For instance, it is natural to assign hitting pedestrians a greater penalty than hitting inanimate objects like a banner. The linearity of expectation allows us to compute the total expected risk of a path by summing uncertain zones' expectations the path navigates through. It aligns well with traditional shortest path algorithms, which evaluate path weights by summing the costs of the vertices they traverse. Our solution reduces computational time by treating differently uncertain zones and neutral zones devoid of uncertainty. A faster subroutine is used for the latter, so that expensive shortest path queries are exclusive on uncertain regions.

evaluation. Algorithmic correctness is proved using graph theory and logic techniques. Empirical emulation is based on dropout sampling, whereby a decision is taken according to a probability distribution over the digraph, then we emulate the decision on a sampled digraph, corresponding to the environment. Emulations are on Carla open-source simulator for autonomous driving research.

Carla's implementation. Carla's waypoints correspond to vertices, and probabilities correspond to vertices' weights. Implementation is by Carla's built-in basic agent module [**py-carla**]. Carla's built-in path planning modules were not used as more waypoints are needed within two-adjacent road segments. A simple Python script is written to construct a digraph from Carla's waypoints.

2 Outline

We start with basic graph-theory and probability notation. We recall some useful properties of graphs' adjacency matrices. In Section 3, we illustrate how graph-theoretic notions and vertices weights do capture the external uncertain environment of autonomous vehicles. We prove the equivalence of some scenarios with respect to probabilistic expectation. In Section 4 we illustrate our framework from a higher-level view, showing how uncertain subgraphs are connected to neutral subgraphs. Then we give more details of subroutines. Namely, reducing vertex weights to edge weights, and querying a neutral path. The section ends with dynamic updates whereby a path is adjusted without recomputing everything from scratch. In Section 5, the dropout sampling methodology is explained, and how is it applied on digraph traversal. Finally, Section 5.2 shows Carla emulation testing results on selected scenarios.

3 Modeling

real-life scenario. We illustrate our framework for autonomous vehicle path planning in environments with uncertain elements. An uncertain object's diameter and probability is modeled as a subgraph with weighted edges. Navigating through uncertain conditions, minimizing total probabilistic expectation, is accounted by finding a shortest-weight path. See *Figure 1* for a vehicle's environment's object detection.

math notation within a real-life context.

graph. A vehicle's possible paths may be thought of by a set of discrete waypoints, whereby each waypoint allows the transition to some other waypoints. In terms of graph theory, it is a digraph G where waypoints correspond to vertices $V(G)$, and valid transitions are given by directed edges $E(G)$. An edge (a, b) indicates a transition from waypoint a to waypoint b is possible.

classification. An object's uncertain classification is given by a probability distribution over classes. For example we may classify an object as either a dog with probability 0.7 or a cat with probability 0.3. Likelihoods and penalties could be combined by probabilistic expectation $Ex[X]$. For example, the expected harm or penalty is $Ex[X] = (2 \cdot 0.3) + (1 \cdot 0.7)$ given cats are twice as important or penalizing as dogs. A vertex weight $w(v)$ is set with such expectation denoting classification uncertainty.

positioning. An object's uncertain position corresponds to potential subset of vertices, which we call an uncertain subgraph U. Since the object is known to exist, vertices of U constitute a probability distribution. For example, if an object's position may be uniformly uncertain among nearby vertices v_1, v_2, v_3, v_4, v_5 where $(v_1, v_2), (v_1, v_3), (v_1, v_4), (v_5, v_1) \in E(G)$. Similarly, a constant penalty, since it is the same object, may be set for all vertices of U.

path. We will prove later it is equivalent to set a weight for edge (a, b) in place of vertex b weight. By linearity of expectation, the total probabilistic expectation of a path is the summation of edges' weights the path navigates through. It follows, computing a shortest path suffices for minimizing the total expectation.

images/sensor.png

Figure 1: Vehicle's detected environment [1].

precise formal definition

We are given a simple directed graph G. That is, a set V together with an irreflexive binary relation E on V. We call them vertices and directed edges. Our digraphs are anti-symmetric between any two vertices. The sample space Ω is the set of all possible objects at some waypoint. By convention, no two objects are present in the same waypoint, so $Pr[E] = 0$ for all $E \subseteq \Omega$ where $|E| \geq 2$. A random variable X associates a penalty for each outcome, and the corresponding random variable distribution μ_X associates probabilities for outcomes.

Definition. For a subset of vertices $S = \{v_1, \ldots, v_k\} \subseteq V(G)$ and a set of random variable distributions $\{\mu_{X_1}, \ldots, \mu_{X_k}\}$, an *uncertain zone* is a function U

$$U : v_i \mapsto \mu_{X_i}.$$

Examples. See *Figure 2* for a visualizing aid.

- *Case 1.* $U = \{(v_8, \{(1, 0.7)\})\}$ denotes an object is present in some vertex v_8 with probability 0.7.
- *Case 2.* $U = \{(v_8, \{(1, 0.3)\}), (v_{12}, \{(1, 0.3)\}), (v_{14}, \{(1, 0.3)\})\}$ denotes an object is present in one of vertices v_8, v_{12}, and v_{14}, each with probability 0.3.
- *Case 3.* $U = \{(v_8, \{(100, 0.5), (5, 0.5)\})\}$ denotes v_8 either has a box with probability 0.5 and penalty 5, or a pedestrian with probability 0.5 and penalty 100.

Definition. "given an uncertain zone" "number definitions". Since each X induces $Ex[X]$ on a vertex, an *uncertain environment* is declared to be a tuple $(G, h : V(G) \to \mathbb{R})$, of a digraph G and a partial function h labeling vertices.

[1] taken from [**sensor**]

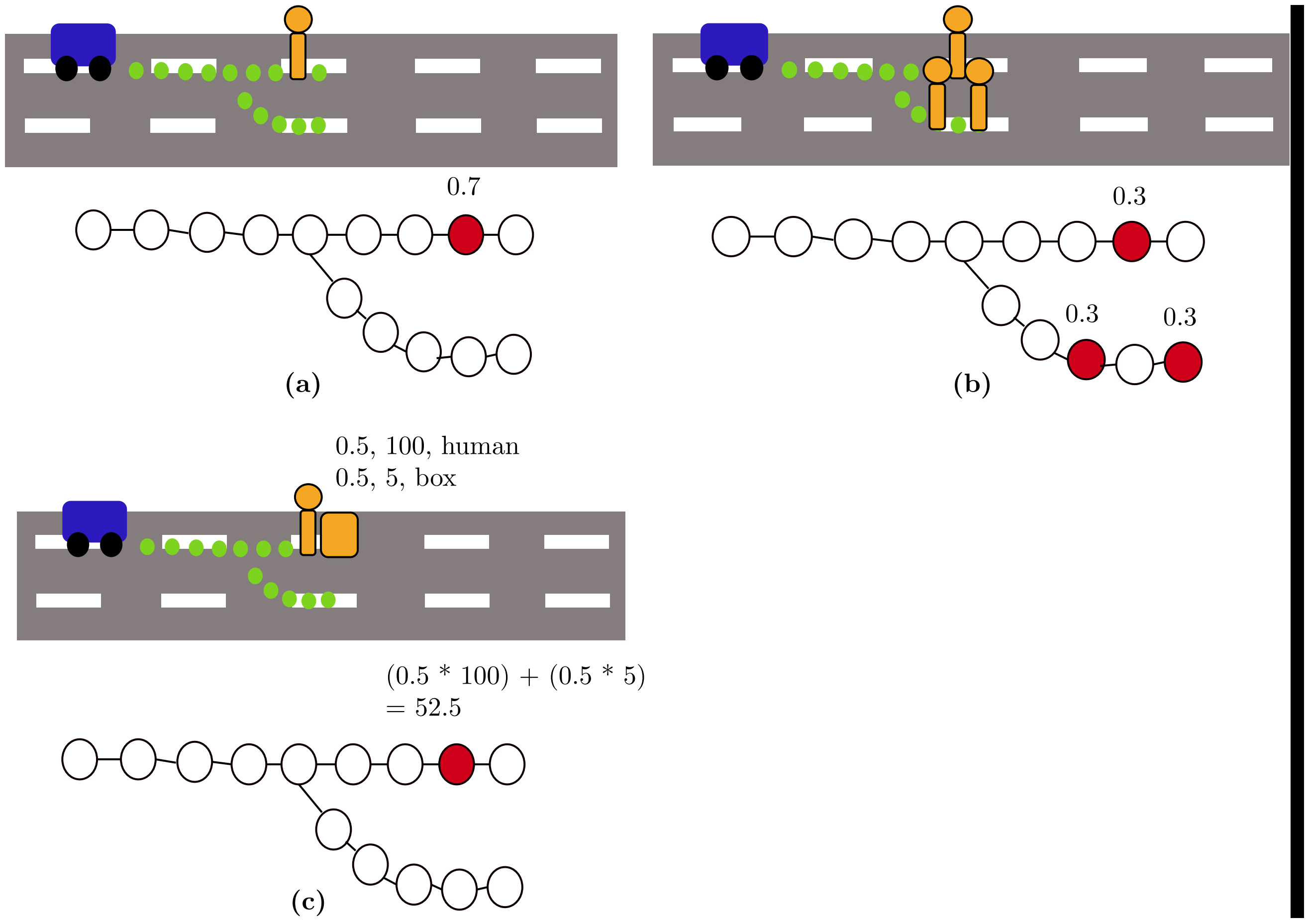


Figure 2: The environment modeled as a digraph

Observe two cases. In the former, a single object's existence is known but its position is probabilistically uniform. In the latter, it is possible to have multiple existing objects or none at all. In both cases, the expectation of a vertex $Ex[\mu_{X_i}]$ is the same. As we aim in our work to reduce the expected harm, we don't need to distinguish the former case from the latter. So a single approach enables us to solve both cases.

Lemma. For an uncertain zone U, the following two scenarios result in exactly the same expectation for vertices: (1) An object is uniformly placed into a vertex of the subgraph; (2) Each vertex independently has an object with probability 1/k.

Proof. Let X_i be an indicator random variable, taking value 1 if an object is placed in the ith vertex and 0 otherwise. In the first case, since the object is placed uniformly, each vertex has exactly $1/k$ probability of receiving it. In the second case, by definition, each vertex independently has probability $1/k$ of containing an object. Therefore, in both scenarios, $Pr[X_i = 1] = 1/k$ for any vertex i. The expected value of X_i is $Ex[X_i] = 1Pr[X_i = 1] + 0Pr[X_i = 0] = 1(1/k) + 0((k-1)/k) = 1/k$. Thereby, despite the different underlying distribution, both scenarios yield identical expected labeling across all vertices in the subgraph.

Proposition. In both cases, traversing the whole subgraph, in expectation hits a single object. This means that regardless of whether we use the first method (uniformly placing exactly one object across the k vertices) or the second method (independently assigning an object to each vertex with probability

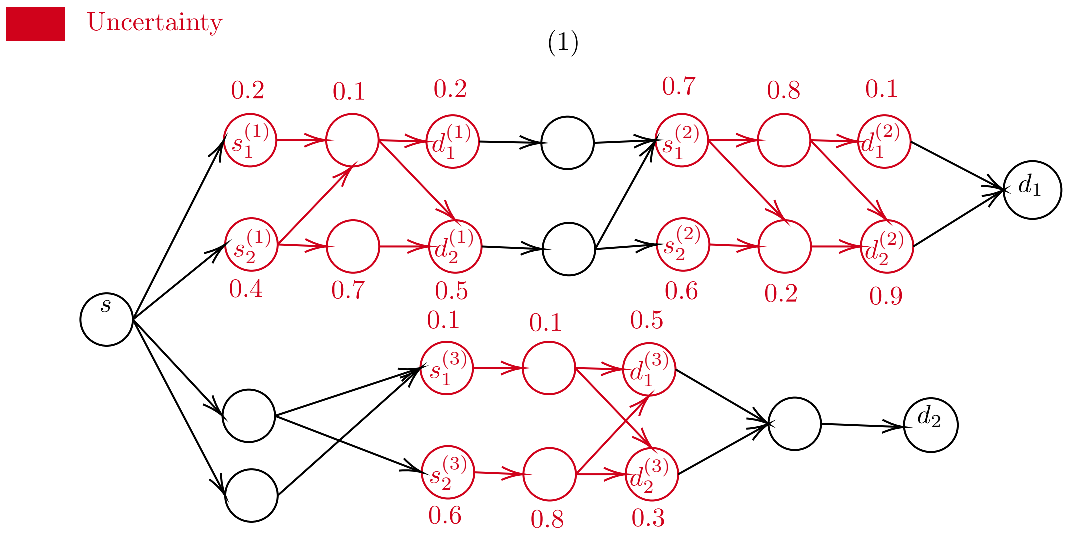
1/k), the expected number of objects encountered when examining all vertices remains the same.

Proof. By applying the linearity of expectation property, the expected total number of objects in the subgraph is $Ex[X_1 + X_2 + \cdots + X_k] = Ex[X_1] + Ex[X_2] + \cdots + Ex[X_k]$. We already know $Ex[X_i] = 1/k$ for each vertex i, and there are k vertices in total, then the expectation is $k(1/k) = 1$. While the proof is simple but the observation is counterintuitive, as the second scenario could potentially place multiple objects or no objects at all.

4 Algorithm

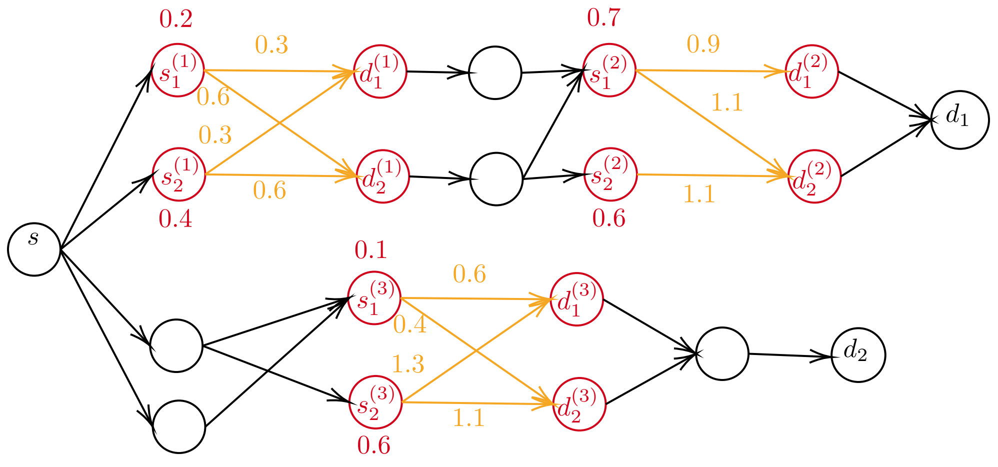
solution idea. We distinguish between uncertain subgraphs and neutral subgraphs devoid of any uncertainty. Shortest-path algorithms are queried on uncertain subgraphs. A fast subroutine searches for paths among the neutral zone, connecting those uncertain subgraphs with each other. Combining computed paths of uncertain zones and neutral zones is realized by constructing a smaller higher-level graph as in *Figure (3-4)*. Finding a shortest-path of it, accounts for both uncertain subgraphs and neutral subgraphs, minimizing total expectation. Any black-box shortest-path subroutine could be used, and the choice of it is out of this paper's scope. Dynamic updates follow the basic idea of avoiding recomputing a path from s to d if all intermediary vertices are unchanged. If an object's subgraph probabilities change, then the higher-level graph will correspondingly change, requiring only re-querying shortest-path subroutine on the smaller higher-level graph, utilizing all what was previously computed. If an object's diameter changes, a new object appears or disappears, then a single vertex of the higher-level graph will correspondingly change. The faster neutral path gets re-queried, then again we look for a better path on the higher-level graph. In all cases, expensive computations of uncertain subgraphs shortest-paths are re-utilized. Tackling vertex weights is not common in the literature. Notable rare examples are [**vertexWeighted˙1**], [**vertexWeighted˙2**], and [**vertexWeighted˙3**].

intuitive example on figure 3. Let's reflect that on a concrete example of *Figure 3*. (1) is the given input graph. For an uncertain subgraph, (2) reduces the shortest path from a starting vertex to a destination vertex, to a single weighted edge. (3) transforms every source-destination pair to a single vertex. (4) queries the faster neutral path mechanism, to find arbitrary paths between the yellow highlighted vertices. Resulting edges are not weighted since selecting which neutral path to take has nothing to do with minimizing expected harm. Now we are given a vertex-weighted graph again, so we can query shortest-path on it in (5). As seen in (6), the path computed in (5), alongside neutral paths, and shortest-paths computed earlier, are all combined to produce the final optimal path of (1).



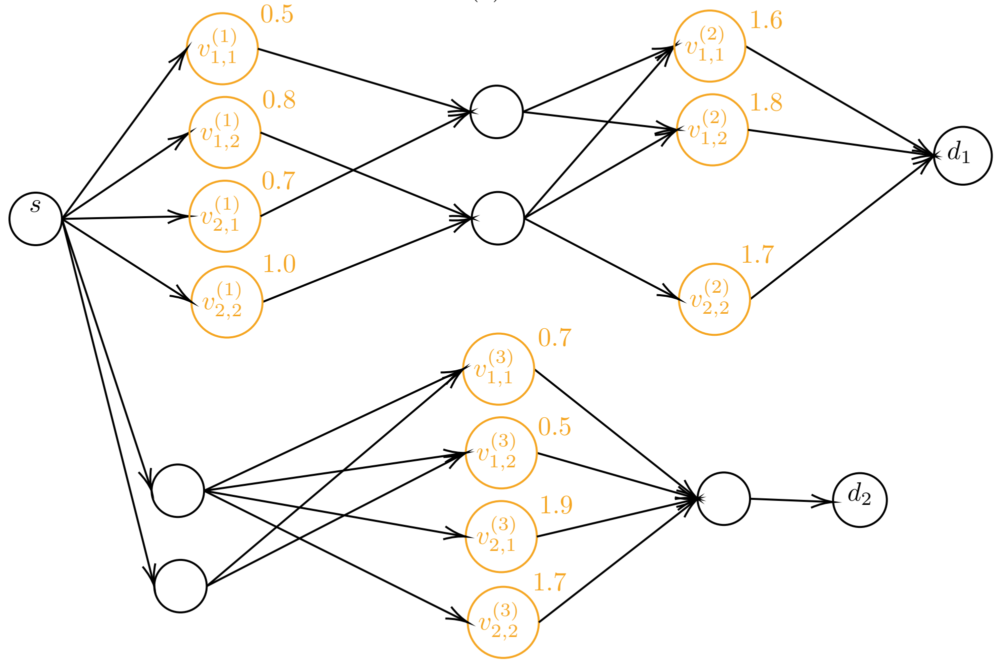
🟧 Shortest path edge

(2)



🟧 Shortest path vertex

(3)



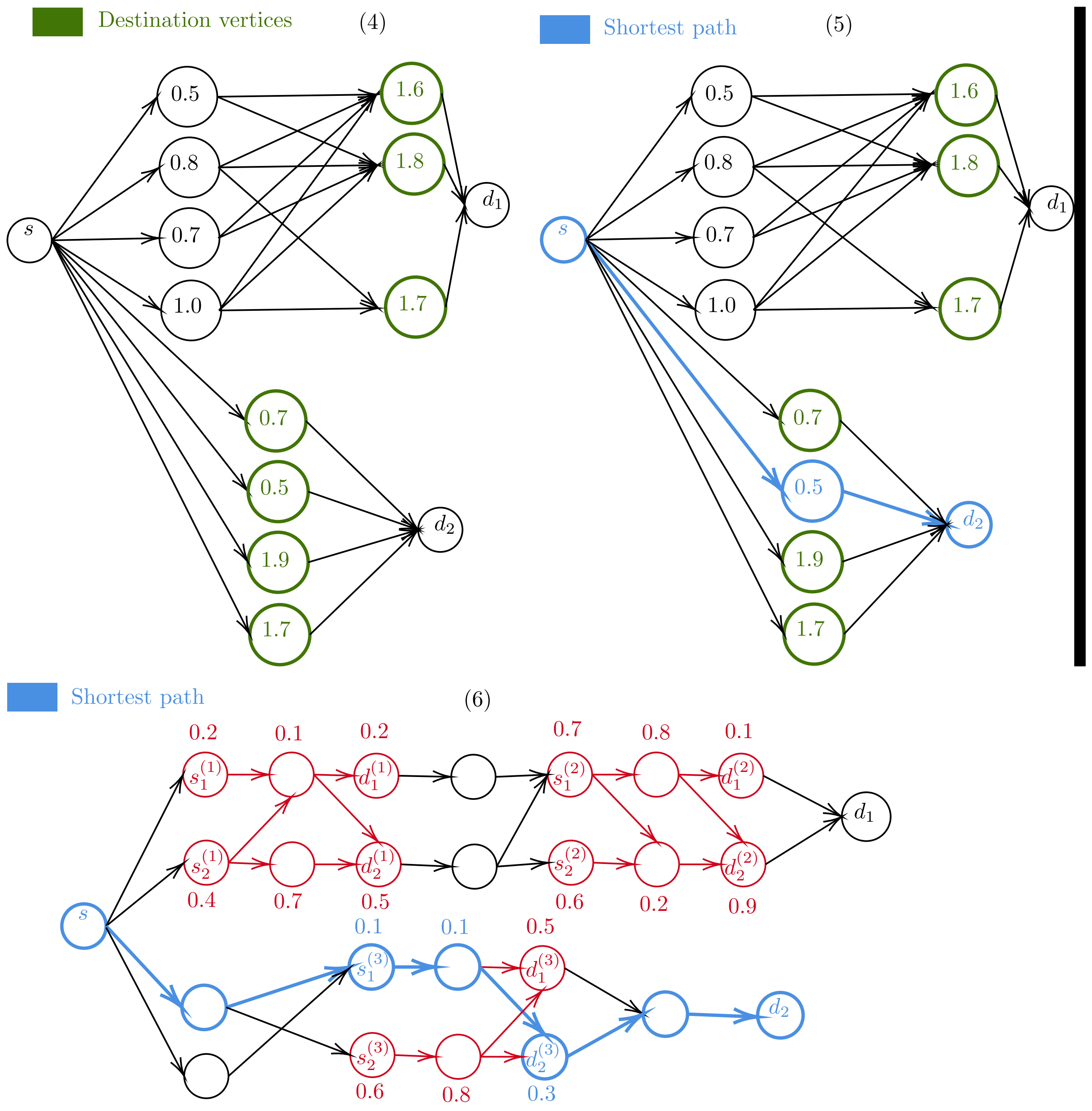


Figure 3: High-level Architecture

solution within math notation. The input is a graph containing uncertain subgraphs $\{G^{(z)}\}_z$, each represents object uncertainty in the environment. For each subgraph $G^{(z)}$ the shortest-path distance of all pairs $(s_i^{(z)}, d_j^{(z)})$ of local starting and destination vertices gets computed. Those pairs get converted to weighted edges with weights $w_{ij}^{(z)}$ corresponding to shortest-path weights. Each such pair $(s_i^{(z)}, d_j^{(z)})$ gets transformed to a vertex $v_{ij}^{(z)}$ weighted by $w_{ij}^{(z)}$, whereby it has inner edges $(v, v_{ij}^{(z)})$ if there were $(v, s_i^{(z)})$ for all v, and has outer edges $(v_{ij}^{(z)}, v)$ if there were $(d_j^{(z)}, v)$. For every pair of vertices $v_{ij}^{(z)}$ and $v_{qp}^{(z')}$, a random path is sampled, and a corresponding edge $(v_{ij}^{(z)}, v_{qp}^{(z')})$ is added. Now we can compute the shortest path to any destination. Through our transformations and computations, a computed path could be reduced back to the original read graph to have a shortest-path.

Procedure.

- (1) Read a graph with uncertain subgraphs $\{G^{(z)}\}_z$.
- (2) Reduce every uncertain subgraph $G^{(z)}$ to edges $(s_i^{(z)}, d_j^{(z)})$ weighted with the shortest distance $w_{ij}^{(z)}$ between $s_i^{(z)}$ and $d_j^{(z)}$.
- (3) For every z, reduce every pair $(s_i^{(z)}, d_j^{(z)})$ to vertex $v_{ij}^{(z)}$ weighted by $w_{ij}^{(z)}$. Set inner edges $(v, v_{ij}^{(z)}) \longleftrightarrow (v, s_i)\ \forall v$ and set outer edge $(v_{ij}^{(z)}, v) \longleftrightarrow (d_j, v)\ \forall v$.
- (4) For every $v_{ij}^{(z)}$ and $v_{qp}^{(z')}$, find a neutral path from $v_{ij}^{(z)}$ to $v_{qp}^{(z')}$, then set edge $(v_{ij}^{(z)}, v_{qp}^{(z')})$.
- (5) Compute a shortest path to any destination.
- (6) Output the path but on the original graph, which shows the neutral paths and shortest paths of the uncertain subgraphs altogether.

In our implementation, we construct graph (4) immediately from graph (1). Detailed steps are given for a clearer illustration.

4.1 Vertex-weighted Shortest Path

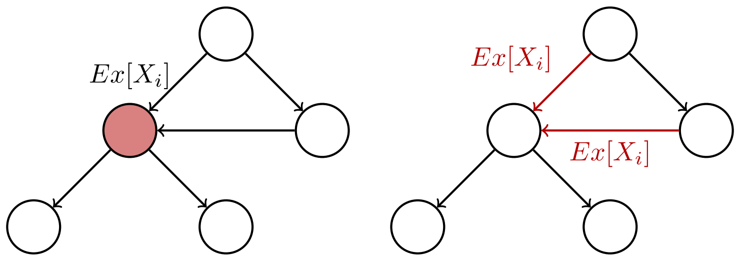


Figure 4: Vertex uncertainty to edge weight

Definition. *weight-backward* transformation. Suppose we are given a graph G where the ith vertex is labeled with expectation $Ex[X_i]$. Construct a digraph G' by labeling the edges of G as follows: for every $v_i \in V(G)$, we label an edge (v_k, v_i) by $Ex[v_i]$.

Theorem. If G is transformed to G' by *weight-backward*, then a path p is of expectation w in G iff it is of expectation w in G'.

Proof. Recall by definition $Ex[X_{root}] = 0$. For the base case where $|p| = 1$, total expectation $Ex[X] = 0$ in G. In G', p won't have any edges, so $Ex[X] = 0$ in G' as well.

Assume the statement holds for $k \geq 1$, and consider an arbitrary path p where $|p| = k + 1$. Call $p = (v_1, \ldots, v_{k+1})$. By hypothesis $\sum_{i=1}^{k} Ex[X_i] = \sum_{i=2}^{k} Ex[X_{(v_{i-1}, v_i)}]$. By definition we have $Ex[X_{k+1}] = Ex[X_{(v_k, v_{k+1})}]$. Hence the inductive step follows as

$$\left(\sum_{i=1}^{k} Ex[X_i]\right) + Ex[X_{k+1}] = \left(\sum_{i=2}^{k} Ex[X_{(v_{i-1}, v_i)}]\right) + Ex[X_{(v_k, v_{k+1})}]$$
$$\sum_{i=1}^{k+1} Ex[X_i] = \sum_{i=2}^{k+1} Ex[X_{(v_{i-1}, v_i)}]$$

Corollary. For a *directed graph* G and a path p, p is minimal or maximal in G if and only if p is minimal or maximal in transformed G'.

Follows immediately since ordering is preserved by the previous theorem.

4.2 Neutral Path

The matrix A here is the adjacency matrix of a given digraph G whereby $A[i,j] = 1$ if and only if $(v_i, v_j) \in E(G)$. A useful property of adjacency matrices is that $A^k[i,j]$ is exactly the number of distinct walks of length k from vertex v_i to vertex v_j.

Consider a relation composition G^k where aG^kb iff there is a length-k walk in G from a to b. Observe $G^0 \cup G^1 \cup \cdots \cup G^{|V(G)|-1} = (G \cup G^0)^{|V(G)|-1}$. Accordingly, if we consider the graph $G \cup G^0$ in place of G, with matrix $A + I_n$ in place of A, then $A^k[i,j]$ is the number of walks of length at most k from v_i to v_j. See [**discreteHandbook**] for a background.

Definition. Boolean Vector. A vector whose entries are zero or one.

Definition. Boolean Dot Product. Given boolean vectors v and w, the boolean dot product is $v \odot w = \bigvee_k \big(v[k] \wedge w[k]\big)$, whose co-domain is 1 and 0.

Definition. Matrix Boolean Product. Given two matrices A and B, their boolean product is $A \odot B = C$ whereby $C[i,j] = A[i,] \odot B[,j]$.

Notation. We denote $A^{[k]} = \underbrace{A \odot \cdots \odot A}_{k \text{ times}}$.

Proposition. $A^{[k]}[a,b] = 1$ iff aG^kb. It decides whether a walk exists, instead of counting walks in usual matrix multiplication.

Proposition. Given a graph of order n, if $k < n-1$, and $A^k[i,j] = 1$, then there is a path from i to j, but if $A^k[i,j] = 0$ then the existence of a path from i to j in the graph is undecided.

Definition. Intermediary matrices M corresponding to $A^{[1]}, A^{[2]}, A^{[4]}, \ldots, A^{[2^m]}$, where $A^{[1]} = G \cup G^0$ and $A^{[2k]} = A^{[k]} \odot A^{[k]}$.

$$M^1[i,j] = \begin{cases} \phi & A^1[i,j] = 1,\ i \neq j \\ Nil & \text{otherwise} \end{cases}$$

$$M^{2k}[i,j] = \begin{cases} q & A^k[i,q] = 1 = A^k[q,j],\ i,j,q \text{ distinct} \\ Nil & \text{otherwise} \end{cases}$$

Algorithm. Given:

- an adjacency matrix A of $G \cup G^0$ whose order is n.
- a power of two $k \geq n$.
- vertices i and j.

a constructed path from vertex i to j is returned.

```
# input:  k, power of 2; i & j, M^k[i,j] != Nil
# output: intermediary path between i and j
intermediaryPath(i, j, k)

    # base case. directly connected, then no intermediary path
    if G[i,j] = phi return ()
    # not directly connected, then we pick an intermediary vertex
    q = M^k[i,j]

    # recursively construct intermediary paths from i to q, and from q to j.
    return intermediaryPath(i, q, k/2) + (q) + intermediaryPath(q, j, k/2)

# input: k power of 2; i & j
# output: path from i to j
main(i, j, k)

    # trivial path
    if i == j return (i)

    # directly connected
    if G[i,j] == 1:
        return (i, j)

    # if not directly connected, and no intermediary vertex exists,
    # then no path exists
    if M^k[i,j] == Nil return Nil

    # if an intermediary vertex exists,
    # then so does some intermediate path
    return (i) + intermediaryPath(i, j, k) + (j)
```

Correctness Proof. By strong induction on the length of paths.

For the base case where $|p| = 1$, clearly $i = j$ and we get the right answer.

For the base case where $|p| = 2$, clearly $p = (i, j)$, and either i is directly connected to j, or $M^1[i, j] = Nil$.

Consider a path p such that $|p| = m > 2$ and denote $p = (v_1, v_2, \ldots, v_m)$. Assume the algorithm is correct for any path p where $|p| < m$.

For the case of $v_1 \neq v_m$ and $M^k[i, j] \neq Nil$, we know there is an intermediary vertex q, with a path p_0 from v_1 to q, and a path p_1 from q to v_m. Clearly $|p_0| < k$ and $|p_1| < k$, and by induction hypothesis, paths p_0 and p_1 produced are correct. It follows the path $(v_1) + p_0 + (q) + p_1 + (v_m)$ is correct from i to j as intended.

4.3 Dynamic Update

Weights

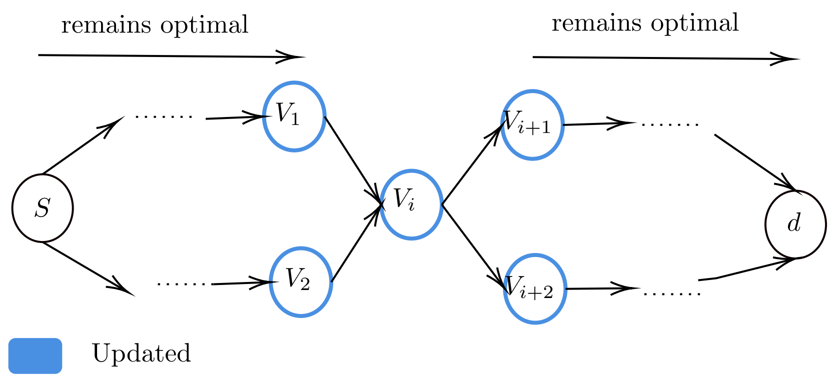


Figure 5:
Dynamic update by vertices weights

Proposition. If no weight updates occur to the intersection of the set of fathers / predecessors of d_j and the set of children of s_i, then the optimality of $s_i - d_j$ paths are preserved.

Proof. Let $\Delta = ((w'(s_i) - w(s_i) + (w'(d_i) - w(d_i))$ be the net weight change of s_i and d_j. Then for any path p_i, the new weight is $w'(p_i) = w(p_i) + \Delta$. It follows a minimal path weight p_k is preserved. Indeed, if we have real numbers $a_1, a_2, \ldots, a_k$ with minimum a_k, then $(a_k + \Delta)$ would be a minimum of $(a_1 + \Delta), \ldots, (a_k + \Delta)$.

Procedure.

- Memoize all pairs in the left subgraph and right subgraph through the previously computed distances and predecessors matrices.
- Call any all pairs shortest path algorithm.

```
# input: matrix updatedGraph, with the newly updated weights
#        list updatedVertices
#        2d list preDistances, the previously computed distances matrix
#        2d list prePredecessors, the previously computed predecessor matrix
def dynamicAllPairsShortestPath(updatedGraph, updatedVertices, preDistances, prePredecessors):

    # length
    n = len(updatedGraph)

    # initialize empty memoization
    distMemo = {}
    predMemo = {}

    # for all pairs
    for i in range(n):
        for j in range(n):
            # left subgraph with no updated vertices
            if i < min(updatedVertices) and j < min(updatedVertices):

                # memoize the previously computed distance
                dist_memo[(i,j,n)] = preDistances[i][j]

                # memoize the previously computed predecessor
                pred_memo[(i,j,n)] = prePredecessors[i][j]

            # right subgraph with no updated vertices
            if i > max(updatedVertices) and j > max(updatedVertices):

                # memoize the previously computed distance
                dist_memo[(i,j,n)] = preDistances[i][j]

                # memoize the previously computed predecessor
                pred_memo[(i,j,n)] = prePredecessors[i][j]

    # execute all pairs shortest path algorithm but without recomputing memoized pairs
    return AllPairsShortestPath(updatedGraph, distMemo, predMemo)
```

Corollary. Procedure correctness.

Proof. We already know the correctness of memoization from the previous *lemma*. Given a correct memoization, the correctness of the all pairs shortest path is given, since it is used as a black-box subroutine. ■

Diameter

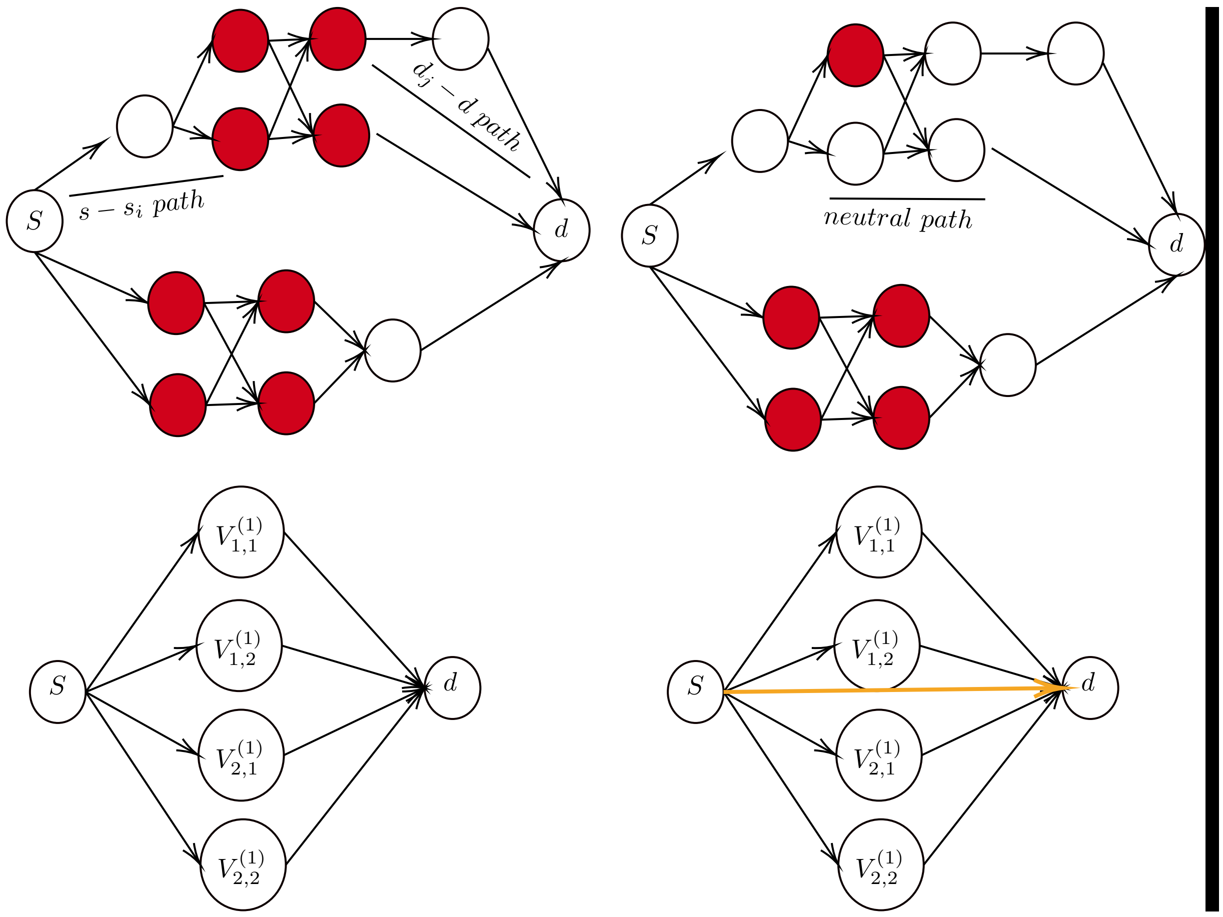


Figure 6: Dynamic update by diameter reduction

Procedure.

- If an $s_i^{(k)} - d_j^{(k)}$ path is found in $G^{(z)}$, add an edge trespassing vertex $v^{(z)}$ in the uncertain subgraph.

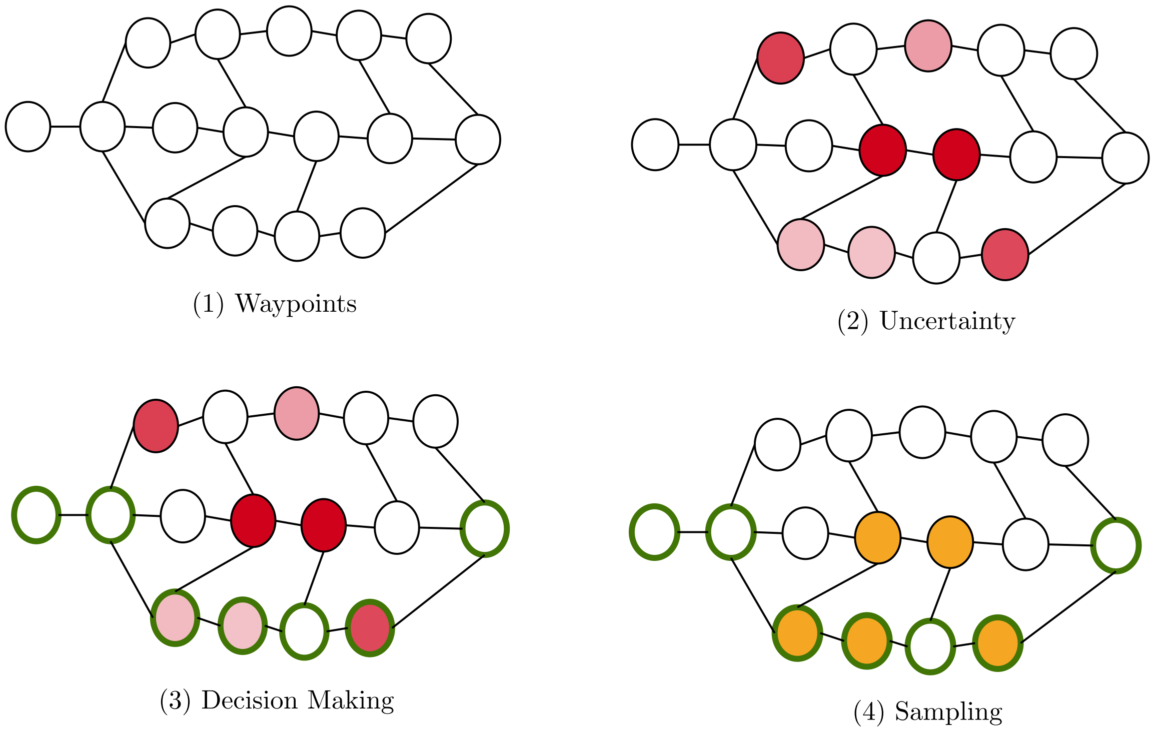
- Recompute $G^{(z)}$ shortest path.
- Recompute shortest path in the uncertain vertex.

Given a diameter collapse of an uncertain subgraph $G^{(z)}$, we look for neutral paths of source-destination pairs $(s_i^{(k)}, d_j^{(k)})$ within it. If found, a new edge is added bypassing the vertex $v^{(z)}$, enabling the avoidance of traversing through $G^{(z)}$. Moreover, the shortest path within $G^{(z)}$ is recomputed as some vertices might be reachable only through the collapsed $G^{(z)}$. Finally, the shortest path of uncertain vertices $v^{(z)}$ is recomputed, hopefully finding more optimal paths through the new bypasses.

5 Evaluation

5.1 Methodology

The probabilistic graph is converted into a deterministic instance through sampling. Each uncertain parameter, like obstacle positioning, is constructing by drawing from its probability distribution. Uncertainty gets reduced into a single concrete scenario, enabling deterministic emulation and evaluation. Multiple such samples are generated to well-approximate the probability distribution, and accordingly well-quantify our decision making mechanism on the decided path outcomes. Metrics include collisions, lives lost, traffic laws violations. Statistical measures include expected total harm or reward, and variance.



(1) Waypoints

(2) Uncertainty

(3) Decision Making

(4) Sampling

Figure 7: Evaluation methodology

Procedure.

- (1) Read a discrete graph of way-points, along probabilistic expectations along its vertices.
- (2) The path planner takes a decision under the uncertainty.
- (3) A deterministic graph is sampled by given distributions.

- (4) The path decided earlier is emulated on the deterministic graph, evaluating incurred harm or reward.
- (5) Repeat (3) and (4) multiple times, aggregating evaluations, to produce a quantification of uncertainty mitigation.

5.2 Experimentation

Scenarios here are inspired by Carla's driving challenge [**scenarios-carla**] but we introduce uncertainty aspects in the environment. Our numeric evaluation uses a penalty formula taken from Carla's challenge also [**eval-carla**].

$$P_i = \frac{1}{1 + \sum_j c_j \cdot \#\text{infractions}}$$

Event	**Weight** c_j
Collisions with pedestrians	1.0
Collisions with other vehicles	0.70
Collisions with static elements	0.60
Running a red light	0.40
Failure to yield to emergency vehicle	0.40
Running a stop sign	0.25

Table 1: Penalty Weights

The penalty is computed for each possible decision, so that we can evaluate whether the decision taken by our mechanism is better or not.

In below pictures of Carla's practical implementation, numbers *(i)*, *(ii)*, and *(iii)* do correspond to numbers *(2)*, *(3)*, and *(4)* in Section 5.1. Blue waypoints correspond to destination vertices; red waypoints corresponds to uncertain vertices; yellow waypoints correspond to the path decided by the vehicle.

For a probability distribution and the vehicle's decision induced by it, multiple trials are emulated, computing the penalty for each and for each possible decision, then we aggregate.

Obstacle Avoidance

images/lane-change.png

Figure 8: Carla scenario 17

It is inspired by *17 - Obstacle avoidance without prior action* where the vehicle aims to avoid pedestrians whose presence is probabilistic by lane change.

The right lane has an uncertain subgraph whose distribution is 0.2, 0.5, 0.2. The left lane has an uncertain subgraph whose distribution is 0.7, 0.8, 0.8. The vehicle decided to take the right lane. The results of 10 trials are as follows.

Trial	**Left Penalty**	**Right Penalty**	**Positive**
1	0.33	0.5	✓
2	0.5	0.5	✗
3	0.25	0.5	✓
4	0.25	1.0	✓
5	0.25	1.0	✓
6	0.25	0.5	✓
7	0.33	0.33	✗
8	0.25	1.0	✓
9	0.33	1.0	✓
10	0.33	0.5	✓

Table 2: Left and right lanes' penalties.

As the vehicle is ahead of deciding between two lanes, both lanes decisions are emulated on Carla, where penalty is computed for both. If our mechanism led to a better penalty, i.e higher score, we declare the trial to be positively justifying our decision. In this case, it is reducing collisions with pedestrians. Here 80% of tested trials are positive.

Break

images/brake.png

Figure 9: Carla scenario 16

It is inspired by *16 - Longitudinal control after leading vehicle's brake* where the vehicle brakes to avoid hitting pedestrians whose presence is probabilistic.

The uncertain subgraph distribution is $0.4, 0.1, 0.7$. The vehicle decided to brake before any uncertain vertex. The results of 10 trials are as follows.

Trial	**Brake Penalty**	**Advance Penalty**	**Positive**
1	0.5	0.5	✗
2	1.0	0.5	✓
3	1.0	1.0	✗
4	1.0	1.0	✗
5	1.0	0.5	✓
6	0.5	0.5	✗
7	0.5	0.33	✓
8	1.0	0.5	✓
9	1.0	0.5	✓
10	0.5	0.33	✓

Table 3: Brake versus advance penalties.

As the vehicle is ahead of deciding between braking and advancing, both lanes decisions are emulated on Carla, where penalty is computed for both. We declare a trial to be positive similarly as we did before. In this case, it is reducing collisions with pedestrians. Here 60% of tested trials are positive.

It is notable to highlight the percentage here is decreased as the vehicle takes some time until it completely stops. Even if it decides to break at a way-point which is safe to be positioned in, acceleration effects may cause it to encroach into the uncertain zone. The odds of that increase if the vehicle is moving faster and an object turns out to be positioned near the boundaries of the uncertain zone.

We analyze how many meters are encroached from the uncertain graph, given distance between the vehicle and the uncertain graph, alongside vehicle's speed, during the moment of braking. In *Image 3* the vehicle brakes only one waypoint, i.e 2 meters, from the uncertain graph colored in red.

`images/brake-distance.png`

Figure 10: Break analysis

The results of testing multiple values of distances and speeds are in *Table 5.* For example, when the vehicle was 2 meters ahead of the uncertain graph with speed 60 km/h, it encroached 6 meters from the uncertain subgraph.

Distance (m)	**Speed (km/h)**		
	30	**45**	**60**
2	2	4	6
4	2	4	4
8	0	0	0

Table 4: Meters encroached given distances and speeds.

Results suggest more than 4 meters are needed to avoid the uncertain subgraph upon braking.

In the *Lane Change* scenario, modifying the distance and speed has no notable difference; the vehicle performs lane changes under these modified variables.

Intersection

images/intersection.png

Figure 11: Carla scenario 3

It is inspired by *3 - Right turn at an intersection with crossing traffic.* where the vehicle breaks to avoid hitting crossing vehicles whose presence is probabilistic.

The uncertain subgraph distribution is $0.6, 0.5, 0.7$. The vehicle decided not to take the intersection. The results of 10 trials are as follows.

Trial	**Intersection Penalty**	**Brake Penalty**	**Positive**
1	0.59	1.0	✓
2	0.42	1.0	✓
3	0.59	1.0	✓
4	0.59	1.0	✓
5	0.59	1.0	✓
6	0.42	1.0	✓
7	1.0	1.0	✗
8	1.0	1.0	✗
9	0.42	1.0	✓
10	0.42	1.0	✓

Table 5: Brake versus intersecting penalties.

As the vehicle is ahead of deciding between taking the intersection and braking, both decisions are emulated on Carla, where penalty is computed for both. We declare a trial to be positive similarly as we did before. In this case, it is reducing collisions with vehicles. Here 80% of tested trials are positive.

Observe the percentage here is increased, as the vehicle moves slowly at conjunctions, so it can break confidently.

6 Conclusion

summary. We designed a mechanism that enables autonomous vehicles to take a decision under uncertainty. It is independent of any prior environmental knowledge and can be used with any module producing probabilistic information alongside penalties. Computational time is reduced by querying shortest-path only on uncertain zones. Our dynamic update mechanism utilizes computed uncertain subgraphs. The statistical dropout sampling technique was utilized for evaluation. In particular, minimizing probabilistic expectation conforms to the empirically tested trials on Carla emulator.

future work. In this work, we aimed for finding the optimal solution. It may happen that updates are frequent, up to the point where our mechanism has to re-compute everything from scratch. In such case, designing a time-bounded mechanism at the expense of maintaining optimality may be preferred. We suggest to the reading sampling random paths as quick as possible, then improving given the time the vehicle has. Checking *Iterative Methods* may be a good starting point.

7 References