A Comparison of Several Speed Computation Methods for the Safe Shortest Path Problem

Aurélien Mombelli, Alain Quilliot, Mourad Baiou

A Comparison of Several Speed Computation Methods for the Safe Shortest Path Problem

Aurélien Mombelli[1], Alain Quilliot[1] and Mourad Baiou[1]
LIMOS, UCA, 1 rue de la chebarde, 63170 Aubière, France
{*f_author, s_author*}*@uca.fr,*

Keywords: Dynamic Programming, Risk Aware, Time-dependant, Reinforcement Learning

Abstract: This paper is based on (Mombelli et al., 2022). In their article, they dealt with a fleet of autonomous vehicles which is required to perform internal logistics tasks in some protected areas. This fleet is supposed to be ruled by a hierarchical supervision architecture which, at the top level, distributes and schedules Pick up and Delivery tasks, and, at the lowest level, ensures safety at the crossroads and controls the trajectories. They presented the problem of finding a shortest path under risk constraints and proposed a way to compute speed functions along the path. In this paper, we present some theoretical results and focus on the fixed path problem. We propose several new ways of computing speed functions including a couple with reinforcement learning.

1 INTRODUCTION

While human drivers are capable of local risk avoidance, autonomous vehicles (AV) do not have such intrinsic capabilities and risk aware algorithms are absolutely necessary.

Monitoring a fleet involving AVs usually relies on hierarchical supervision. The trend is to use three levels. At the low level, or *embedded level*, robotic-related problems are tackled for specific AVs like controlling trajectories in real-time and adapting them to the possible presence of obstacles, see (Martínez-Barberá and Herrero-Pérez, 2010). At the middle level, or *local level*, local supervisors manage priorities among AVs and resolve conflicts in a restricted area (Chen and Englund, 2016; Philippe et al., 2019) who worked on crossroad strategies. Then, at the top level, or *global level*, global supervisors assign tasks to the fleet and schedule paths. This level must take lower levels into account to compute its solution. See (Koes et al., 2005; Vis, 2006; Le-Anh and De Koster, 2006) for example or (Wurman et al., 2008) who compute the shortest path thanks to the A* algorithm but assign each task to the fleet of AVs using a multi-agent artificial intelligence to avoid most conflicts in arcs.

This study puts the focus on the global level: routing and giving instructions to an AV in a fleet. An AV, idle until now, is chosen to carry out a new task. Since a vehicle is moving from some origin to some destination, performing some loading or unloading transaction and keeping on. But some specific features impose new challenges:

- The time horizon for AVs is usually short and decisions have to be taken online, which means that decision processes must take into account the communication infrastructure, see (Vivaldini et al., 2013), and the way the global supervisor can be provided, at any time, with a representation of the current state of the system and its short term evolution;
- As soon as AVs are involved, safety is at stake (see (Pimenta et al., 2017)). The global supervisor must compute and schedule routes in such a way that not only tasks are going to be efficiently performed, but also that local and embedded supervisors will perform their job safely.

To compute a safe shortest path, (Ryan et al., 2020) used, a weighted sum of time and risks in Munster's roads in Ireland using an A* algorithm. In their case, the risk is a measure of dangerous steering or braking events on roads. But these techniques mostly cannot be applied here because the risk, in our case, is time-dependent. Here, the risk will not be a measure of dangerous events on the roads of a city but the expected repair costs in a destructive accident scenario, should it happens. To compute such expectation, planning of risks is to be computed from the already working fleet because the path they will follow and their speed functions are known. A risk planning procedure is then used to transform previous information into risk functions over time that is necessary to compute the expected repair costs for the AV.

In an article presented at the ICORES conference in 2022, (Mombelli et al., 2022) presented the problem of finding a shortest path under risk constraints and proposed a way to compute speed functions along the path. In this contribution, we will present new advances related to:

- the comprehension of the way risk functions may be defined and computed
- the complexity of the global problem
- the different ways decisions may be designed and induce transitions
- the way statistical learning may be involved in order to filter both decisions and states

First, a precise description of the global problem is presented with structural results. Then, the problem with fixed path where only the speed functions are unknown is tackled and it's complexity is discussed. In the fourth section, the fixed path problem is approached with dynamic programming with several speed generation methods followed by, in the fifth section, filtering and learning processes. Lastly, numerical experiments are presented with a conclusion.

2 RISK INDUCED BY THE ACTIVITY OF AN AV FLEET

In this section, the problem we already talked about in the previous paper is described together with the specific problem on a fixed path. Then, some structural results of the solution will be presented leading to a reformulation of our problem. Lastly, the complexity of the problem is tackled.

2.1 Measuring the Risk

A warehouse is represented as a planar connected graph $G = (N, A)$ where the set of nodes N represents crossroads and the set of arcs A represents aisles. For any arc $a \in A$, L_a represents the minimal travel time for an AV to go through aisle a. Moreover, two aisles may be the same length but one may stock fragile objects so that vehicles have to slow down.

Also, risk functions $\Pi_a : t \mapsto \Pi_a(t)$, generated from activities of aisle a, are computed using the risk planning procedure on experimentation in a real warehouse that we are provided with. It is important to note that the risk is not continuous. Indeed, there is, in an aisle, a finite number of possible configurations: empty, two vehicles in opposite directions, etc. (see Figure 1). Each configuration is, then, associated with an expected cost of repairs in the event of accidents.

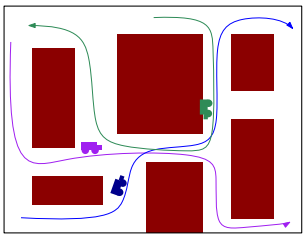

Figure 1: At time t, 3 aisles have 1 vehicle each. At the next time , Blue and Purple join in the same aisle. One time after, all 3 vehicles join, generating high risks in this aisle (figure from (Mombelli et al., 2022)).

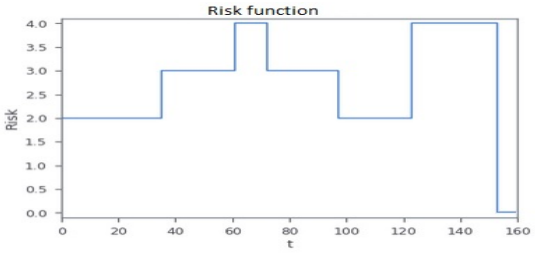

Figure 2: Risk function of an aisle (figure from (Mombelli et al., 2022)).

Therefore, they are staircase functions evaluated in a currency (euro, dollars, etc.). Figure 2 shows an example of a risk function of an aisle.

From a risk function, we can have an estimation of the risk an AV takes in an aisle a between two times t_1 and t_2 with $v : t \mapsto v(t)$ as its speed function with Equation 1.

$$risk(t_1, t_2, v) = H(v) \int_{t_1}^{t_2} \Pi_a(t)\, dt \tag{1}$$

We impose function H to be such that $H(v) \ll \frac{v}{v_{max}}$ in order to express the fact that a decrease of the speed implies a decrease of the risk. In further sections, H is set to $H : v \mapsto \left(\frac{v}{v_{max}}\right)^2$.

Remark 1. ***Speed Normalization****: We only care here about traversal times of arcs $e \in A$, and not about their true length, in the geometric sense. So we suppose, without loss of generality that, for any arc e, $v_{max_e} = 1$. Therefore we deal with reduced speed values $v \in [0, 1]$ and L_e means the minimal traversal time for arc e.*

2.2 Discussion about Risk Functions

We are going to discuss here the way functions Π come from and how they can be derived from observations in a real example. As a matter of fact, the key issue is about the way the risk related to the activity of a fleet of AVs inside a transit network $G = (N, A)$ may be defined.

In any case, the risk should be defined here as related to the expected damage $\int_{[t,t+dt]} \Pi^e(s).ds$ which is induced, on a given arc e and during a time interval $[t,t+dt]$, by the activity of the fleet along arc e. It comes that the issue for us is to determine risk functions $t \mapsto \Pi^e(t), e \in A$. We notice that this activity may not only involve vehicles which are moving along $e=(i,j)$ but also those who are performing some storage or retrieval activity, and also that we may have to consider vehicles which move along the arc $e^{-1}=(j,je)$.

2.2.1 The Simplest Case

Here, the risk is induced by identical moving vehicles, which follow the arcs of G. Then, risk function Π^e should take the form $\Pi^e(t)=\Pi^e_n(v_1(t),\ldots,v_n(t))$, where n is the number of vehicles which are moving along arc e at time t, and $v_1,\ldots,v_n$ their respective speed. Since we suppose all vehicles to be identical, function Π^e_n should be symmetrical. In order to express the impact of the introduction of an additional vehicle at a given speed v, we should be able to write:

$$\begin{aligned}&\Pi^e_{n+1}(v_1,\ldots,v_n,v)\\ &=H(v).Q^e_n(v_1,\ldots,v_n)+\Pi^e_n(v_1,\ldots,v_n)\end{aligned}$$

with $H(v)<<v\leq 1$ and $Q^e_n(v_1,\ldots,v_n)$ symmetrical. This expression means that we distinguish the damages which explicitly involve additional vehicle $n+1$ from the damages which only involve other vehicles.

Above decomposition identifies function $\Pi^e=\Pi^e(t)$ with function $Q^e_n(v_1,\ldots,v_n)$ and quantity $H(v).\Pi^e(t)$ involved in Section 3 with the marginal expected damage induced by the introduction into arc e and at speed v, of an additional vehicle. Since this marginal expected damage is likely to involve, not only the additional vehicle, but also some vehicles which are currently moving inside e, we should also have:

$$\begin{aligned}\sum_{i=1,\ldots,n+1} &H(v_i).Q^e_n(v_1,\ldots,v_i,\ldots,v_n)\\ &\geq \Pi^e_{n+1}(v_1,\ldots,v_n,v_n+1)\end{aligned}$$

Above formulas may be implemented into many ways. A simple one comes as follows:

$$\Pi^e_n(v_1,\ldots,v_n)=(\Pi_{i=1,\ldots,n}H(v_i)).H^e(n)$$

where H^e is an increasing function of n such that $H^e(n)-H^e(n-1)$ is also increasing. According to this, we may set:

$$\begin{aligned}\Pi^e(t)=&\Pi^e_{n+1}(v_1(t),\ldots,v_n(t),1)\\ &-\Pi^e_n(v_1(t),\ldots,v_n(t))\end{aligned}\qquad(2)$$

2.2.2 The Case When non Moving Vehicles Must Be Taken Into Account

We proceed the same way to arrive to the formula:

$$\Pi^e_{n,p}(v_1,\ldots,v_n)=(\Pi_{i=1,\ldots,n}H(v_i)).K^e(n,p)$$

where n denotes the number of vehicles which are moving along e and p the number of vehicles involved into some storage/retrieval activity. Function K^e should be increasing in n and p and such that:

- $K^e(n+1,p)-K^e(n,p)$ is increasing in n
- $K^e(0,p)=0$ and $K^e(n,0)=H^e(n)$

2.2.3 The Case When Arc e Support Bi-directional Vehicles

We proceed the same way to arrive to the formula:

$$\begin{aligned}&\Pi^e_{n,q}(v_1,\ldots,v_n,u_1,\ldots,u_q)\\ &=(\Pi_{i=1,\ldots,n}H(v_i)).(\Pi_{j=1,\ldots,q}H(u_i)).L^e(n,q)\end{aligned}$$

where n denotes the number of vehicles which are moving along e and q the number of vehicles which are moving along e^{-1}. Function L^e should be increasing in n and q and such that:

- $L^e(n+1,q)-L^e(n,q)$ is increasing in n
- $L^e(n,q+1)-L^e(n,q)$ is increasing in q
- $L^e(0,p)=H^{e^{-1}}(n)$ and $L^e(n,0)=H^e(n)$

3 OUR PROBLEM: SEARCHING FOR SPEED FUNCTIONS UNDER RISK CONSTRAINTS

An idle AV must now carry out a new task inside the warehouse. Its task is to go through a path Γ. A way of finding such path has been tackled in the previous article and we will focus on determining the speed functions in each aisle a of this path **while being provided with**:

- The minimum travel time L_a of every arc a;
- The risk function $R_a : t \mapsto R_a(t)$ of every arc a;

Then, **we want to compute**:

- the leaving time t_a of every arc a of Γ.
- the speed functions v to apply when the vehicle is located inside every arc a of Γ.

As it is, we could have worked with a multi-objective problem. However, we want to compute speed functions such that the AV is "safe". That means a maximum risk value constraint is added. The warehouse manager will impose a maximum value of risk

R_{max} (quantified in currency, it can correspond to the cost of replacing a vehicle in the event of an accident) that an AV can take for a task. Then, the objective is to determine quickly:

SSPP: Safe Shortest Path Problem

Compute leaving times t_a and speeds functions v_a on every arc a of path Γ

such that:

- the arrival time at the end of the path is minimal;
- the global risk $\sum_{a\in\Gamma} risk\,(t_{a-1},t_a,v_a) <= R_{max}$.

where t_{a-1} is the leaving time of the previous arc (i.e. the entry time of arc a).

3.1 Some Structural Results

As it is stated, **SSPP** looks more like an optimal control problem than a combinatorial one. But, as we are going to show now, we may impose restrictions on speed function v, which are going to make the model get closer to a discrete decision model.

Proposition 1. *Optimal solution (Γ,v) of **SSPP** may be chosen in such a way that v is piecewise constant, with breakpoints related to the times t_i when vehicle V arrives at the end-nodes of arcs $t_i, i=1,\ldots,n$, and to the breakpoints of function $\Pi^e_i, i=1,\ldots,n$.*

Proof. Let us suppose that V is moving along some arc $e=e_i$, and that δ_1, δ_2 are 2 consecutive breakpoints in above sense. If $v(t)$ is not constant between δ_1 and δ_2 then we may replace $v(t)$ by the mean value v^* of function $t\mapsto v(t)$ between δ_1 and δ_2. Time value $Time(\Gamma,v)$ remains unchanged, while risk value $Risk(\Gamma,v)$ decreases because of the convexity of function H. So we conclude. □

Proposition 2. *If optimal **SSPP** trajectory (Γ,v) is such that $v(t)\neq 1$ at some t, then $Risk(\Gamma,v)=R_{max}$.*

Sketch of the proof. Let us suppose that path Γ is a sequence $e_1,\ldots,e_n$ of arcs of G. We proceed by induction on n.

First case: $n=1$.

We suppose above assertion to be false and set:

- $q_0 = largest\ q$ such that $v<1$ between t_q and t_{q+1};
- $v_0 = related\ speed$; $l_0 = distance\ covered$ by V at time t_{q_0}.

Let us increase v_0 by $\varepsilon>0$, such that $v_0+\varepsilon\leq 1$ and that induced additional risk taken between t_{q_0} and t_{q_0+1} does not exceed $R_{max}-Risk(\Gamma,v)$. Then, at time t_{q_0+1}, vehicle V covered a distance $l>l_0$. If $l<L_e$, then it keeps on at speed $v=1$, and so arrives at the end of e before time t_{q_0}, without having exceeded the risk threshold R_{max}. We conclude.

Second case: $n>1$.

Let us suppose above assertion to be false and apply the following strategy:

On the first arc e_1 where there exists t such that $v(t)<1$, increase the speed as described in the case $n=1$. The additional risk taken does not exceed $R_{max}-\frac{Risk(\Gamma,v)}{2}$. Therefore, vehicle V reaches the end of e_1 at some time $t_1-\beta,\beta>0$, with an additional risk no larger than $(R_{max}-\frac{Risk(\Gamma,v))}{2}$. So, for any $i=2,\ldots,n$ we compute speed value v_i such that moving along e_i at speed v_i between $t_{i-1}-\beta$ and t_{i-1} does not induce an additional risk more than $(R_{max}-\frac{Risk(\Gamma,v))}{2n}$ and finish the path strictly before time t_n. We conclude.

Proposition 3. *Given an optimal **SSPP** trajectory (Γ,v), with $\Gamma=\{e_1,\ldots,e_n\}$ and v satisfying Proposition 1. Let us denote by t_i the arrival time at the end of arc e_i. Then, for any $i=1,\ldots,n$, and any t in $[t_{i-1},t_i]$ such that $v=v(t)<1$, the quantity $H'(v(t)).\Pi^{e_q}(t)$ is independent on t, where $H'(v)$ denotes the derivative of H in v.*

Proof. Once again, let us denote by t_i time when vehicle V arrives at the end of arc e_i. For a given i, we denote by $\delta_1,\ldots,\delta_{H(i)}$, the breakpoints of function Π^{e_i} which are inside interval $]t_{i-1},t_i[$, by $\Pi^{i_q}_q$ related value of Π^{e_q} on the interval $]\delta_j,\delta_{j+1}[$, by $v_0,\ldots,v_q,\ldots,v_{H(q)}$, the speed values of V when it leaves those breakpoints, and by R_q the risk globally taken by V when it moves all along e_q. Because of proposition 2, vector $(v_0,\ldots,v_{H(q)})$ is an optimal solution of the following convex optimization problem:

$$\text{Compute } (v_0,\ldots,v_{H(q)}) \text{ such that } \sum_q v_q.(\delta_{q+1}-\delta_q)$$
$$\text{and which minimizes } \sum_q H(v_q)\Pi^{e_i}_q(\delta_{q+1}-\delta_q).$$

Then, Kuhn-Tucker conditions for the optimality of differentiable convex optimization program tell us that there must exists $\lambda\geq 0$ such that: for any q such that $v_q<1, H'(v_q).\Pi^{e_i}_q=\lambda$. As a matter of fact, we see that λ cannot be equal to 0. We conclude. □

Remark 2. *In case $H(v)=v^2$, above equality $H'(v_q)\Pi^{e_i}_q=\lambda$ becomes $v_qR^{e_i}_q=\frac{\lambda}{2}$ where $v_qR^{e_i}_q$ means the instantaneous risk per distance $\frac{dR}{dL}$ at the time when V moves along e_i between times δ_q and δ_{q+1}.*

3.2 Adaptation of the Notion of Wardrop Equilibrium

Let us suppose now that we are dealing with a risk measure which derives from the above mono-directional simple case, and that we are imposed some time horizon $[0,T^{max}]$. Then, we consider a vehicle fleet, with n vehicles $k=1,\ldots,n$, whose activity is scheduled

between 0 and T^{max}, according to routing strategies $(\Gamma_k, v_k), k = 1, \ldots, n$, where Γ_k is a path which starts from some origin o_k and arrives at some target destination d_k. b $(\Gamma_k, v_k), k = 1, \ldots, n$, allow us to derive, for any arc e a risk function $t \mapsto \Pi^e(t)$ defined with:

- A collection of auxiliary risk functions $t \mapsto \Pi^e_k(t)$ whose meaning is: $\Pi^e_k(t)$ is the risk function, which we obtain while removing vehicle k and related routing strategy (Γ_k, v_k)
- A collection of marginal risk functions $t \mapsto \Pi^e_k(t)$, obtained from equation 2 and whose meaning is: if we consider all vehicles but k, and if we make vehicle k run along e between t and $t + dt$ at speed, $v \leq 1$, then additional expected damage explicitly involving for k is equal to $H(v).\Pi^e(t).dt$

Then we may extend the well-known notion of Wardrop Equilibrium by defining a Risk Wardrop Equilibrium as follows: A collection of strategies $\{(\Gamma_k, v_k), k = 1, \ldots, n\}$ define a Risk Wardrop Equilibrium if, for any $k = 1, \ldots, n : (\Gamma_k, v_k)$ is an optimal strategy for vehicle k in the sense of the SSPP instance involving marginal risk functions $t \mapsto \Pi^e_k(t)$ and time horizon $[0, T^{max}]$. We state:

Proposition 4. *If $T^{max} \geq sup_k\{L^*(o_k, d_k)\}$, where L^* means the shortest path distance in the sense of the length L, then there exists a Risk Wardrop Equilibrium.*

Proof. It directly derives from the way we defined $t \mapsto \Pi^e_k(t)$ as related to the marginal expected damage induced by the introduction of additional vehicle k into a transit network where other vehicles $1, \ldots, k-1, k+1, \ldots, n$, are supposed to be already evolving. More precisely, function $t \mapsto \Pi^e_k(t)$ is defined for any k by equations 2 which links it to risk function $t \mapsto \Pi^e(t)$ and risk functions $t \mapsto \Pi^e_k(t)$. Let us consider an optimal solution $\{(\Gamma^{opt}_k, u^{opt}_k), k = 1, \ldots, n\}$ of the following GRM: Global Risk Minimization problem:

Compute a strategy collection $\{(\Gamma_k, v_k), k = 1, \ldots, n\}$ such that resulting global risk value $\sum_e \int_{[0,T^{max}]} \Pi^e(t)dt$ be the smallest possible.

Let us denote by $t \mapsto R^{opt^e}(t)$, $t \mapsto R^{opt^e}_k(t)$ and $t \mapsto \Pi^{opt^e}_k(t), k = 1, \ldots, n$, related risk functions and marginal risk functions. Then we see that, for any k, $(\Gamma^{opt}_k, u^{opt}_k)$, which meets time horizon $[0, T^{max}]$, minimizes under this constraint $\sum_e \int_{[0,T^{max}]} \Pi^e(t)dt - \sum_e \int_{[0,T^{max}]} R^{opt^e}_k(t)dt = \sum_e \int_{[0,T^{max}]} H(v(t)).\Pi^{opt^e}_k(t)dt$, and so is an optimal solution of the SSPP instance induced by marginal risk functions $t \mapsto \Pi^{opt^e}_k(t)$. If $T^{max} \geq sup_k\{L^*(o_k, d_k)\}$ then GRM clearly admits a feasible solution. What remains to be proved is that it admits an optimal solution. We notice that no more than n vehicles can be located at a given time t on a same arc e, and that, once a vehicle leaves e, it does not come back later. It comes that the number of breakpoints of a function $\Pi^e_k(t)$ cannot exceed n. Then a simple topological argument about compactness allows us to check that from any sequence $\{(\Gamma^p_k, u^p_k), k = 1, \ldots, n, p = 1, \cdots + \infty\}$, we may extract a convergent subsequence in the simple sense, and that Lebesgue Theorem may be applied to this convergence. We deduce that there must exist a collection $\{(\Gamma^{opt}_k, u^{opt}_k), k = 1, \ldots, n\}$ which achieves $Inf_{(\Gamma_k, v_k), k=1,\ldots,n}(\sum_e \int_{[0,T^{max}]} \Pi^e(t)dt)$. □

3.3 Risk Versus Distance SSPP Reformulation

Remark 2 leads us to define the Risk versus Distance coefficient for arc e_i as the value $2H'(v_q)\Pi^{e_i}_q$ involved in Proposition 3. This proposition, combined with Proposition 1, allows us to significantly simplify **SSPP**: We define a *risk versus distance strategy* as a pair (Γ, λ^{RD}) where:

- Γ is a path, that means a sequence $\{e_1, \ldots, e_n\}$ of arcs, which connects origin node o do destination node d;
- λ^{RD}_e associates, with any arc e in Γ, *Risk versus Distance coefficient* $\lambda^{RD}_e = 2H'(v)R_e$. In case $H(v) = v^2$, we notice that this coefficient means the amount of risk per distance unit induced on arc e at any time t such that $v(t) < 1$, by any trajectory (Γ, v) which satisfies Proposition 3.

Let us suppose that we follow a trajectory (Γ, v) which meets Proposition 3, and that we know value λ^{RD}_e for any arc e of Γ.Since H is supposed to be convex and such that $H(v) \ll v$, we may state that H' admits a reciprocal function H'^{-1}. Then, at any time t when vehicle V is inside arc e, we are able to reconstruct value

$$v(t) : \begin{cases} H'^{-1}(\frac{\lambda^{RD}_e}{2R_e}), & \text{if } H'^{-1}(\frac{\lambda^{RD}_e}{2R_e}) < 1 \\ 1, & \text{otherwise} \end{cases} \quad (3)$$

According to this and Proposition 3, **SSPP** may be rewritten as follows with the notations $Risk(\Gamma, v)$ and $Time(\Gamma, v)$ as $Risk(\Gamma, \lambda^{RD})$ and $Time(\Gamma, \lambda^{RD})$:

Risk versus Distance Reformulation: Compute *risk versus distance strategy* (Γ, λ^{RD}) such that $Risk(\Gamma, \lambda^{RD}) \leq R_{max}$ and $Time(\Gamma, \lambda^{RD})$ is the smallest possible.

3.4 Discussion About the Complexity

The time dependence of the transit network together with the proximity of the SSPP model with Shortest Path Constraint models suggests that SSPP is a complex problem. Sill, identifying the complexity of SSPP is not that simple, since we are dealing with continuous variables. Complexity also depends on function H, and so we suppose here that $H(v) = v^2$. We first may check that:

Proposition 5. *SSPP is in NP.*

Sketch of proof. It is clearly enough to deal with the case when $\Gamma = \{e_1, \ldots, e_n\}$ is fixed. Let us denote by t_i the time when vehicle V will arrive at the end of e_i (we set $t_0 = 0$). If we know values $\{t_1, \ldots, t_n\}$, then we may retrieve values $\lambda_e^{RD}, e \in \Gamma$, through binary search. It comes that the core of our problem is about the computation of values $\{t_1, \ldots, t_n\}$. For any i, value t_i may be either equal to some breakpoint of the risk function or located two of them. In order to make the distinction between those 2 configurations, we introduce the function σ that is equal to 0 on every breakpoints and 1 between them. The number of possible functions σ is bounded by C_{H^n}. Now we notice that, once function σ is fixed, the problem becomes about computing values for those among variables $\{t_1, \ldots, t_n\}$ which are not non-instantiated, together with speed values for all consecutive intervals defined by elements of Δ and by those time values. This problem can be formulated as a cubic optimization problem, and one may check that, in case this problem has a feasible solution, then first order Kuhn-Tucker equations for local optimality determine exactly one local optimum. We conclude.

Conjecture 1. *SSPP is in NP-Hard.*

This conjecture is motivated by the fact that SSPP seems to be close to the constrained shortest path problems, which are most often NP-Hard (Lozano and Medaglia, 2013). Practical difficulty of SSPP may be captured through the following example (Figure 3), which makes appear that if (Γ, v) defines an optimal SSPP trajectory, the risk per distance value $\lambda_e^{RD} = 2.H'(v(t)).\Pi^e$ may be independent on t on arc e as told in Proposition 3, but cannot be considered as independent on arc e.

Path Γ contains 2 arcs, e_1 and e_2, both with length 1 and maximal speed 2. Function Π^{e_2} is constant and equal to 1. Function Π^{e_1} takes value 2 for $0 \leq t \leq 1$, and a very large value M (for instance 100) for $t > 1$ (see Figure 3). $R_{max} = \frac{3}{4}$; Function H is: $v \mapsto H(v) = v^2$.Then we see that vehicle V must go fast all along the arc e_1, in order to get out of e_1 before this arc becomes very risky. That means that its speed is equal to 1 on e_1, and that its risk per distance value is equal to $\frac{1}{2}$. Next

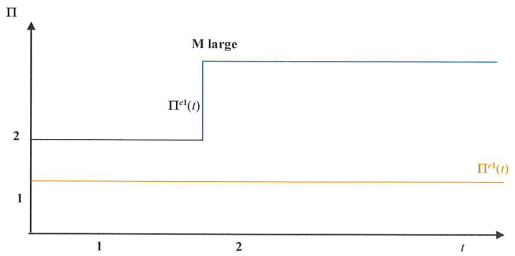

Figure 3: Functions Π^{e_1} and Π^{e2}.

it puts the brake, in the sense that its speed remains equal to 1 but its risk per distance value decreases to $\frac{1}{4}$. It is easy to check that this routing strategy is the best one, with $Risk(\Gamma, v) = \frac{3}{4}$ and $Time(\Gamma, v) = 2$.

3.5 Deriving Speed Functions from Compact Decisions

As we saw in Section 3.3, there is a Risk versus Distance method to generate decisions in the form of a single value λ^{RD} and this value will be used to create the speed functions. Therefore, in this section, we will talk about the transition procedure used to create a speed function from a decision. Moreover, we propose two other methods to generate decisions: Risk versus Time and Distance versus Time.

- **First approach:** *The Risk versus Distance approach.*

Since $H(v) = v^2$, $H'(v(t))\Pi^e(t) = 2u(t)\Pi^e(t)$ for any t during the traversal of e. It comes that if we fix λ^{RD} the speed value $v(t)$ is given by: $v(t) = Inf(1, \lambda^{RD}/\Pi^e(t))$. Resulting state (j, r_2, t_2)will be obtained from λ^{RD} and (i, r_1, t_1) through the iterative process described in Algorithm 1.

A special attention will be put for the first value of dt as it needs to be less than $T_{q+1} - t_1$ and not $T_{q+1} - T_q$.

- **Second approach:** *The Risk versus Time approach.*

Since $H(v) = v^2$ we have that at any time t during the traversal of e, related risk speed $\frac{dR}{dT(t)}$ is equal to $v(t)^2\Pi^e(t)$. It comes that if we fix λ^{RT} we get: $v(t) = Inf(1, \left(\frac{\lambda^{RT}}{\Pi^e(t)}\right)^{1/2})$.
Resulting state (j, r_2, t_2) will be obtained from λ^{RT} and (i, r_1, t_1) through the same following iterative process of Algorithm 1.

- **Third approach:** *The Distance versus Time approach (mean speed approach).*

This method is a little different from the two other as a λ^{DT} will not give the speed function easily but only

Algorithm 1 Risk_Distance Transition procedure

Require: L_e the length of arc e
Require: $T_1,\dots,T_Q$ the breakpoints of Π^e which are larger than t_1 and $\Pi^e_0,\dots,\Pi^e_Q$ related Π^e values.
Initialisation:
$t_2 = t_1$, $r_2 = r_1$, $l = 0$ and $q = 0$.
while $L_e < l$ **do**
  $v = \frac{\lambda^{RD}}{\Pi^e_q}$ if < 1 else 1
  $dt = \frac{Le-l}{v}$ if $< T_{q+1} - T_q$ else $T_{q+1} - T_q$
  $t_2 = t_2 + dt$
  $l = l + v.dt$
  $r_2 = r_2 + v^2 \Pi^e_q dt$
  $q = q + 1$
end while
if $r_2 < R_{max}$ **then**
  return Success
else
  return FAIL
end if

the value t_2: $t_2 = t_1 + \frac{L_e}{\lambda^{DT}}$. In order to determine the speed function $t \mapsto v(t)$ and the value r_2, the quadratic program described in Algorithm 2 must be solved:

Algorithm 2 Distance_Time Transition procedure

Require: L_e the length of arc e
Require: $T_1,\dots,T_Q$ the breakpoints of Π^e which are larger than t_1 and $\Pi^e_0,\dots,\Pi^e_Q$ related Π^e values.
Then we must compute speed values $u_1,\dots,u_Q \in [0,1]$ such that:
$\sum_q u_q^2 \Pi^e_q (T_q - T_{q-1})$ is minimal
$\sum_q u_q (T_q - T_{q-1}) = t_2 - t_1$

This quadratic convex program may be solved through direct application of Kuhn-Tucker 1st order formulas for local optimality. Then we get r_2 by setting: $r_2 = r_1 + \sum_q u_q^2 \Pi^e_q.(T_q - T_{q-1})$. If $r_2 > R_{max}$ then the Mean Speed transition related to λ^{DT} yields a Fail result.

3.6 Generic Algorithmic Scheme

To solve the **SSPP**, the following generic algorithm scheme is used:

- **State space**: $s = (i,t,r)$
  t (resp. r) is a feasible date (resp. sum of the risks) at node i.
- **Decision space**: λ
  A value from one of the three approaches presented in Section 3.5.
- **Transition space**: $s \xrightarrow{\lambda} s'$
  Transition from (i,t,r) to $(j, t_i,\ r + risk(t,t_i,v_i))$ by deriving the speed function v_i from the compact decision λ.

In the **SSPP** case, the decision space is a little different as the next arc is unknown. Then a decision would be $((i,j),\lambda)$ and this scheme becomes an A* algorithm.

In the case where the path Γ is fixed, if all states from the same node are used before moving on the next node, this scheme becomes a forward dynamic programming scheme.

Lastly, a greedy algorithm can be derived from the Risk versus Distance reformulation on the shortest path as Algorithm 3.

Algorithm 3 RD Greedy Algorithm

$l = 0$
$(t,r) = (0,0)$
for $e \in \Gamma$ **do**
  $\lambda^{RD} = \frac{R_{max} - r}{L^* - l}$
  Compute the speed function v thanks to Algorithm 1 and λ^{RD}.
  $t' = t + \frac{L_e}{v}$
  $l = l + L_e$
  $(t,r) = (t', r + risk(t,t',v))$
end for

4 GENERATING DECISIONS AND FILTERING STATES WITH STATISTICAL LEARNING

First, we will talk about the decisions and how to generate them. Then, in a second part, we consider a way to speed up our algorithms in order to adapt to a dynamic context: limiting the number of states (t,r) linked to any node i. We do so by focusing on the $\lambda = \lambda^{RD}$ case.

4.1 Generating Decisions

Starting from state (i,t,r), a lot of new states can be generated. But most of them are useless or not promising enough to be considered (too slow, too risky, slower and riskier than another state, etc.).

Even for the Risk versus Distance reformulation, the optimal value λ_{opt} of every individual arc is unknown, more so for the other two methods. We propose to generate decisions by searching between a low and high estimation of λ_{opt}: λ_{inf} and λ_{sup}. Those

generated values will be distributed between λ_{inf} and λ_{sup} and led by λ_{midst} (half of them uniformly distributed between λ_{inf} and λ_{midst} and half of them between λ_{midst} and λ_{sup} for example).

R_{max}, L^* and the time value t_{greedy} of the greedy algorithm can be used to initialise λ_{midst} as:

- $\lambda_{midst}^{RD} = \frac{R_{max}}{L^*}$
- $\lambda_{midst}^{RT} = \frac{R_{max}}{t_{greedy}}$
- $\lambda_{midst}^{DT} = \frac{L^*}{t_{greedy}}$

Then, $\lambda_{inf} = \frac{\lambda_{midst}}{\rho}$ and $\lambda_{sup} = \rho\lambda_{midst}$ with ρ a given value.

4.2 The Filtering Issue

There is a logical filtering that can be applied first: if the lower bound of the arrival time from a state (the time it took to finish the path at v_{max}) is greater than an existing solution (from the RD Greedy algorithm for example). However, the logical filtering alone will not be very efficient. We propose two more filtering processes: one to limit the number of generated decisions and one to limit the number of states.

4.2.1 State Limitation

In order to filter the set of states $State[i]$ linked to a given node i and to impose a prefixed threshold S_{max} on the cardinality of $State[i]$, several techniques can be applied. For example, 2 states (t,r) and (t',r') can be considered as equivalent if $|t-t'|+|r-r'|$ does not exceed a rounded threshold. We will not follow this approach which does not guarantee that we will maintain the cardinality of $State[i]$ below the imposed threshold S_{max}. Instead, we're going to pretend there's a natural conversion rate ω that turns risk into time. Based on this, we will order the (t,r) pairs according to increasing $\omega t + r$ values and continue with only the best S_{max} states according to this order.

The key issue here is the value of ω. Intuitively, ω should be equal to $\frac{R_{max}}{t^o}$, where t^o is the optimal value of **SSPP**, and we should be able to learn its value depending on the main characteristics of the **SSPP** instances: the most relevant characteristics seem to be the risk threshold R_{max}, the length L^* of the path Γ, the average Δ of the functions $\Pi^e, e \in A$, and the frequency of the breakpoints of these functions. We can notice that if all the functions Π^e are constant and equal to a certain value Δ, then the speed v is constant and equal to $\frac{R_{max}}{L^*\Delta}$, and the time value t^o is equal to $\frac{L^*}{v} = \frac{\Delta L^{*2}}{R_{max}}$. This will lead to initializing ω as $\omega = \frac{R_{max}^3}{\Delta L^{*2}}$.

4.2.2 Filtering by Learning a Good ω Value

However, a lack of flexibility in the pruning procedure associated with a not perfectly adjusted ω value can give, for a given node i, an unbalanced $State[i]$ collection. More precisely, we can qualify a pair (t,r) as risky if $\frac{r}{\sum_{j \geq i+1} L_j}$ is large compared to $\frac{R_{max}}{L^*}$, or cautious if the opposite is true. Our pruning technique may then produce (t,r) pairs that, taken together, are either too risky or too cautious. In order to control this kind of side effect, we make the ω value self-adaptive as:

- if, in average, generated states are risky, ω should be decreased to emphasise lower risk states
- on the contrary, if, in average, generated states are safe, ω should be increased to emphasise faster and riskier states

Then, if $\#State[i] > S_{max}$, which states must be removed? If the ω value is very close to $\frac{R_{max}}{t^o}$, the S_{max} lowest value states are kept and all others are removed from $State[i]$. However, if it is not, a high state can be better than the lowest state. Therefore, a method to determine whether ω is a good approximation must be used. We propose to compute the deviation of the state's risks of $State[i]$ from the travelled percentage of the path as in Equation 4.

$$\Omega = \frac{\sum_{(t,r) \in State[i]} \left(\frac{r}{R_{max}} - \frac{\sum_{j \geq i+1} L_j}{L^*}\right)}{\#State[i]} \quad (4)$$

If Ω's absolute value is high, generated states take, on average, too much risk or too little (meaning they can go faster).
States are, then, removed depending on Ω's value:

- If $|\Omega|$ is "high" (close to 1), ω is supposed a bad approximation: $\left\lfloor \frac{\#State[i]-S_{max}}{3} \right\rfloor$ states are removed from each third of $State[i]$ independently.
- If $|\Omega|$ is "medium": $\left\lfloor \frac{\#State[i]-S_{max}}{2} \right\rfloor$ states are removed from the union of the 1st and 2nd third of $State[i]$ and the 3rd third of $State[i]$ independently.
- If $|\Omega|$ is "small" (close to 0), ω is supposed a good approximation: the first S_{max} states are kept and all others are removed.

5 Two Reinforcement Learning Algorithms

In this section, two approaches will be presented. The first one use the modifications of the decisions se-

quence as a direction and move in the feasible set depending on whether the last direction improved the solution or not. The second approach is based on the fact that a global solution is also a local solution, thus modifying the current decisions toward local solutions.

5.1 Learning a Good Direction of Modifications

The algorithm **Direction_SSPP** that we are now going to present, introduces the principle of reinforcement learning, in the sense that the vehicle V advances along the path Γ by making a decision at each step i (greedy algorithm), but learns to make good decisions through training, i.e. by traversing Γ several times, in order to learn to anticipate the consequences of changes in the function Π^e when it passes from an arc to its successor in the path Γ.

The **Direction_SSPP** algorithm will also work by having vehicle V train multiple times along path Γ. The first time V will follow Γ of size Q by applying the standard decision $\lambda^{RD} = \frac{r^{max}}{L^*}$. Then, at each step of i along Γ, V will make decisions that will aim to improve its current trajectory. We denote by $(r_q, t_q, \lambda_q, t_q)$ respectively the risk of travel, the time of travel, the decision and the time at the end of the arc a_q. Suppose also that t and r denote a current lead, respectively in time and in risk, with respect to the situation which prevailed in q at the end of the previous path along Γ and that $\delta > 0$ is a current target move.

The idea here is to modify all the decisions slightly (a direction of modification) and evaluate their impact. At the next execution, if the direction had a positive gain (i.e. arrived sooner at the end of Γ), the same direction will be took again along with new random modifications. We propose to use a decreasing amplitude $step$ for the random modifications and the movement learned during the previous execution. From now on, a movement defines a direction and its amplitude.

The learning greedy algorithm is therefore presented as Algorithm 4.

The calculation of $(r_q, t_q, \lambda_q, t_q)$ values follows the pseudo-algorithm Algorithm 5.

5.2 Learning While Locally Minimising the Risk

As the **Direction_SSPP** algorithm, the **LocMin_SSPP** we are going to present now, will go through Γ several times. However, this time, the decisions will be modified by trying to minimising the risk locally. Then, the decisions can be increased a little to speed up and use the risk we retrieved previously.

Algorithm 4 Direction_SSPP Algorithm

Require: A first run of the greedy algorithm presented in Section 3.3
This first run provides us with a feasible solution Sol_{best}, as well as a time value t_{best}.
$step = 0.5$
for 10 iterations **do**
  $t_{prec} = t_Q$
  Calculate the values $(r_q, t_q, \lambda_q, t_q)$ along Γ
  if $t_Q < t_{best}$ **then**
    Update Sol_{best} and t_{best}
    $r = r^{max} - \sum_q r_q$
  end if
  $step = 0.8 \times step$
end for

Algorithm 5 Computation of $(r_q, t_q, \lambda_q, t_q)$ values

Require: $step$ the current amplitude
δ the movement applied at the previous execution
δ^{gain} the gain associated with the movement δ
$q = 0$
while $q \leq Q - 1$ **do**
  Draw a random move δ^{alea}
  for all λ_i **do**
    $\lambda_i = \lambda_i - step.(\delta^{gain}\delta_i + \delta_i^{alea})$
  end for
  Apply transition and move to next arc
end while
Apply the transition on the last arc according to the RD reformulation.

To do this, the arcs will be taken in pairs. The entry time of the first arc and the exit time of the second arc are considered fixed and the local problem of minimizing the risk between those two dates will lead the modifications of the decisions.

Here again, we will not search for the exact solution as we want a very fast algorithm. Instead, we will try to make the derivative of the optimization function tend to 0. Therefore, during the transition of an arc for a specific decision λ, the derivatives G_t^λ of t with respect to λ and G_r^λ the derivative of r with respect to λ will be computed. Like for the speed function, $(G_t^\lambda, G_r^\lambda)$ can be computed on every risk plateau:

- for any risk plateau before the end of the arc, $G_t^\lambda = 0$ and $G_r^\lambda = G_r^\lambda + 2.v.\Pi.dT.G_v^\lambda$
- for the last risk plateau, $G_t^\lambda = G_v^\lambda.\frac{dL}{v^2}$ and $G_r^\lambda = G_r^\lambda + 2.v.\Pi.dT.G_v^\lambda + v^2.\Pi.G_v^\lambda$

where Π (resp. dT) is the value (resp. duration) of the risk plateau, G_v^λ is the derivative of v with respect to λ that is 0 if $v = 1$ and $\frac{1}{\Pi}$ otherwise and dL is the

remaining distance before the end of the arc.

At the end the arc and the beginning of the next one, the derivatives (G_t^T, G_r^T) of t and r with respect to T, the leaving time of the first arc (and entry time of the next one), can be computed as Equation 5.

$$\begin{aligned} G_t^T &= \left(\frac{v_1}{v_2} - 1\right) \\ G_r^T &= v_1.(v_2.\Pi_2 - v_1.\Pi_1) \end{aligned} \tag{5}$$

where Π_1 (resp. Π_2) the risk value just before (resp. just after) T and v_1 (resp. v_2) is the speed just before (resp. just after) T according to decision λ, Π_1 and Π_2.

From $(G_t^{\lambda_1}, G_r^{\lambda_1})$ of the first arc, $(G_t^{\lambda_2}, G_r^{\lambda_2})$ of the second arc and (G_t^T, G_r^T), the true derivatives of $(G_t^{\lambda_1}, G_r^{\lambda_1})$ can be computed as in Equation 6.

$$\begin{aligned} G_t^{\lambda_1} &= G_t^{\lambda_2} + (G_t^{\lambda_1} + G_t^T) \\ G_r^{\lambda_1} &= G_r^{\lambda_2} + (G_r^{\lambda_1} + G_r^T) \end{aligned} \tag{6}$$

Finally, the descent direction $(\Delta^{\lambda_1}, \Delta^{\lambda_2})$ is computed thanks to $G_t^{\lambda_1}$, $G_r^{\lambda_1}$, $G_t^{\lambda_2}$ and $G_r^{\lambda_2}$ by projecting $(G_r^{\lambda_1}, G_r^{\lambda_2})$ on the plane orthogonal to $(G_t^{\lambda_1}, G_t^{\lambda_2})$ and normalising the resulting vector as Equations 7:

$$\begin{aligned} (\Delta^{\lambda_1}, \Delta^{\lambda_2})^* &= (G_r^{\lambda_1}, G_r^{\lambda_2}) - \\ &\quad \frac{< (G_r^{\lambda_1}, G_r^{\lambda_2}), (G_t^{\lambda_1}, G_t^{\lambda_2}) >}{||(G_t^{\lambda_1}, G_t^{\lambda_2})||^2} . (G_t^{\lambda_1}, G_t^{\lambda_2}) \\ (\Delta^{\lambda_1}, \Delta^{\lambda_2}) &= \frac{(\Delta^{\lambda_1}, \Delta^{\lambda_2})^*}{||(\Delta^{\lambda_1}, \Delta^{\lambda_2})^*||} \end{aligned} \tag{7}$$

But now that the risk has decreased, the speed can be increased and use back the risk we retrieved previously. We propose to multiply the decision λ_q by the quotient of the previous risk and the new risk. Then, if we actually retrieve some risk, the quotient will be superior to 1 and equal to 1 if we did not.

Here again, the amplitude of the modifications will be decreasing as in Algorithm 4. Therefore, to minimise the risk locally, several iterations of the minimisation process are applied (3 for example) and the calculation of $(r_q, t_q, \lambda_q, t_q)$ values will follows the pseudo-algorithm Algorithm 6.

Algorithm 6 Computation of $(r_q, t_q, \lambda_q, t_q)$ values

Require: *step* the current amplitude
  $q = 0$
  while $q \leq Q - 1$ **do**
    r^{prev} the risk on arc q before any modification
    for 3 iterations (arbitrary choice) **do**
      Compute $(G_t^{\lambda_1}, G_r^{\lambda_1})$ and $(G_t^{\lambda_2}, G_r^{\lambda_2})$ on arcs q and $q+1$
      Compute the descent direction $(\Delta^{\lambda_1}, \Delta^{\lambda_2})$
      $\lambda_q = \lambda_q - step.\Delta^{\lambda_1}$
      $\lambda_{q+1} = \lambda_{q+1} - step.\Delta^{\lambda_2}$
      Compute new risk r on arc q
    end for
    $\lambda_q = \frac{r^{prev}}{r}\lambda_q$
    Apply transition with the new λ_q value
  end while
  Apply the transition on the last arc according to the RD reformulation.

6 NUMERICAL EXPERIMENTS

Goal: We perform numerical experiments with the purpose of studying the behavior of **DP_Evaluate**, **RechLoc_SSPP** and **Direction_SSPP** algorithms.

Table 1: Instance parameters table

id	$\|A\|$	Freq	$\overline{R}$	α	L^*	Greedy
1	4	0.2	2	0.4	59	142.32
2	5	0.25	1.9	1	55	82.47
3	6	0.19	2	1.5	63	72.72
4	4	0.43	2	0.4	67	141.15
5	6	0.6	1.9	1	61	118.89
6	7	0.42	2	1.5	68	93.34
7	6	0.16	1.9	0.4	104	317.61
8	7	0.18	1.9	1	96	142.3
9	6	0.18	2	1.5	93	102.79
10	7	0.41	2	0.4	102	194.87
11	6	0.45	2	1	104	185.33
12	8	0.32	2	1.5	101	129.06

Technical Context: Algorithms were implemented in C++17 on an Intel i5-9500 CPU at 4.1GHz. CPU times are in milliseconds.

Instances: We generate random paths that can be summarized by their number $|A|$ of arcs. Length values $L_e, e \in A$, are uniformly distributed between 3 and 10. Function H is taken as function $u \mapsto H(u) = u^2$. Function Π^e are generated by fixing a time horizon T_{max}, a mean frequency $Freq$ of break points t_i^e, and an average value $\overline{R}$ for value $\Pi^e(t)$: More precisely, values Π^e are generated within a finite set $\{2\overline{R}, \frac{3\overline{R}}{2}, \overline{R}, \frac{\overline{R}}{2}, 0\}$. As for threshold R_{max}, we notice that if functions Π^e are constant with value $\overline{R}$ and if we follow a path Γ with length L_{diam}, the diameter of network G, at speed $\frac{1}{2} = \frac{v_{max}}{2}$, then the expected risk is $\frac{L_{diam}\overline{R}}{2}$. It comes that we generate R_{max} as a quantity $\alpha\frac{L_{diam}\overline{R}}{2}$, where α is a number between 0.2 and 2. Finally, since an instance is also determined by origin/pair (o, p), we

Table 2: DynProg - Impact of λ^{mode}, with $Smax = +\infty$, $Gmax = 21$ and $\rho = 8$

id	T^{RD}	R^{RD}	T^{RT}	R^{RT}	T^{DT}	R^{DT}
1	102.3	98.3	118.5	99.9	126.2	78.9
2	66.1	96.1	68.1	99.1	62	87.8
3	63	84.1	63	84.1	72.3	49.5
4	129.4	79.8	136.6	89.7	131.7	96.3
5	72	97.2	70.2	98.2	75.3	92.7
6	78.7	99.8	77.6	98.1	76.2	99.6
7	279.1	97.3	278.1	99.5	284.6	99.2
8	116	96.3	116.2	94.3	118.5	99.9
9	94.7	98.9	93.2	97.9	93.2	86.2
10	176.7	92.6	167.4	99.3	174.7	99.8
11	123.5	92.8	123.4	93.5	123.2	92.6
12	110.3	99.8	109.6	99.7	111	99.9

denote by L^* the value $L^*_{o,p}$. **Table 1** presents a package of 18 instances with their characteristics and the time value obtained by the greedy algorithm presented in Section 3.3.

☐ Outputs related to the behavior of **DP_SSPP**.

We apply **DP_SSPP** while testing the role of parameters $\lambda = \lambda^{RD}, \lambda^{RT}, \lambda^{DT}$, as well as S_{max} and ρ. So, for every instance, we compute:

- in **Table 2**: The time value T_DP^{mode}, the percentage of risk R_DP^{mode} from R_{max}, and the CPU times (in milliseconds.) CPU^{mode}, induced by application of **DP_SSPP** on the shortest path between o and p with $\lambda^{mode} = \lambda^{RD}, \lambda^{RT}, \lambda^{DT}$, $S_{max} = +\infty$, $G_{max} = 21$ and $\rho = 8$;
- in **Table 4**: The time value T_DP^{mode}, the percentage of risk R_DP^{mode} from R_{max}, and the CPU times (in milliseconds.) CPU^{mode}, induced by application of **DP_SSPP** on the shortest path between o and p with $\lambda^{mode} = \lambda^{RD}, \lambda^{RT}, \lambda^{DT}$, $S_{max} = 21$, $G_{max} = 5$ and $\rho = 4$;
- in **Table 3**: For the specific mode λ^{RD}, mean number $\#S$ of states per node i, together with time value T^{RD}, when $S_{max} = +\infty$, $G_{max} = 21$ and $\rho = 1.5, 4$.

Comments:
There are two important points in those four tables:

- the length of the shortest path of the instance 3 is 63 time units. Therefore solutions that go at full speed do not reach R_{max};
- the *RD* method always performs best or is near the best solution as it can be expected because of the *RD* reformulation, see Section 3.3;
- the *DT* method is designed to minimise its risks, then it may end without reaching R_{max} more often than the two other methods;

Table 3: DynProg - Impact of ρ, with λ^{RD}, $Smax = +\infty$ and $Gmax = 21$

ρ	1.5		4	
id	T^{RD}	$\#S$	T^{RD}	$\#S$
1	113.4	236.75	103.2	75.5
2	79.7	370.3	62	221.15
3	66.2	118.75	63	254.75
4	130.6	70.7	127.9	62.9
5	81.9	151.8	71.2	174.7
6	80.4	723.8	76.1	623.45
7	282.8	233.06	275.7	203.4
8	118.5	550.13	115.8	331.63
9	95.8	61.73	93.2	170.86
10	183.2	222.36	176.5	293.76
11	127.3	91.2	123.2	210.36
12	112.9	542	110.3	511.86

Table 5: ProgDyn with learning

id	T^{LocMin}	T^{Dir}	T^{Gr}
1	139.4	139.4	142.3
2	68.0	62.4	82.4
3	71.6	64.0	72.7
4	141.0	141.0	141.1
5	83.5	83.9	118.8
6	87.8	81.4	93.3
7	285.2	317.6	317.6
8	121.0	142.3	142.3
9	95.5	96.0	102.7
10	194.8	194.8	194.8
11	131.2	125.2	185.3
12	115.5	120.5	129.0

- on instance 11, the tree methods end without reaching R_{max}, so we can deduce that the last arc is not risky around the arrival date.

Then, we can see that the solutions in **Table 4** are fairly close to the solutions in **Table 2** but with just a few states kept at every node (hence the very short computing time).

☐ Outputs related to the behavior the improved greedy algorithms.

Finally, we will compare the different approaches based on the greedy algorithm: the local search and the reinforcement learning. For each instance, we calculate in the **Table 5**: the time value T^{alg} induced by applying **LocMin_SSPP** (*LocMin*) and **Direction_SSPP**(*Dir*) in comparison with the greedy algorithm (*Gr*).

Table 4: DynProg - Impact of λ^{mode}, with $Smax = 21$, $Gmax = 5$ and $\rho = 4$

id	T^{RD}	R^{RD}	cpu^{RD}	T^{RT}	R^{RT}	cpu^{RT}	T^{DT}	R^{DT}	cpu^{DT}
1	140	93.9	0.28	132.8	99.6	0.27	142.3	87.8	0.53
2	72.3	99.5	1.57	78.7	96.1	0.46	79.6	88.5	1.19
3	63	84.1	1.22	63	84.1	1.14	72.6	55.1	1.04
4	140.3	96.6	0.34	137.1	91.7	0.19	229.1	91.3	0.18
5	90.2	90.4	0.98	76.2	98.2	1.1	85.6	96.6	1.34
6	81.4	99.9	1.36	78	99.6	1.72	78.2	97.8	1.99
7	280.5	98	1.21	283.3	98.4	1.01	287.5	82.4	1.12
8	120.2	99.2	1.5	121.2	96.5	1.44	125.8	99.6	1.03
9	95.7	97.2	0.99	93.2	97.4	0.88	94.6	83.5	1.06
10	187.2	94.7	1.08	188.9	97.6	1.07	194.8	95.8	1.11
11	124.9	96.2	1.08	124.5	93.8	1.14	125.1	95.9	1.74
12	126	95.9	1.5	111.2	99.2	2.84	115.6	98.4	3.43

7 CONCLUSION

We dealt here with the problem of finding good speed function with risk constraints for autonomous vehicles, which we handled under the prospect of fast, reactive and interactive computational requirements. We also show how, from observation of real situation, one can compute the risk functions and presented some properties of such functions. Those properties allowed us to reformulate the problem which helped us in the creation of several heuristics that we compared. It comes that a challenge from industrial players becomes to use our models in order to estimate the best-fitted size of an AGV fleet, and the number of AVs inside this fleet. We plan addressing those issues in the next months.

REFERENCES

Chen, L. and Englund, C. (2016). Cooperative Intersection Management: A Survey. *IEEE Transactions on Intelligent Transportation Systems*, 17(2):570–586.

Koes, M., Nourbakhsh, I., and Sycara, K. (2005). Heterogeneous multirobot coordination with spatial and temporal constraints. *AAAI Workshop - Technical Report*, WS-05-06:9–16.

Le-Anh, T. and De Koster, M. B. (2006). A review of design and control of automated guided vehicle systems. *European Journal of Operational Research*, 171(1):1–23.

Lozano, L. and Medaglia, A. L. (2013). On an exact method for the constrained shortest path problem. *Computers and Operations Research*, 40(1):378–384.

Martínez-Barberá, H. and Herrero-Pérez, D. (2010). Autonomous navigation of an automated guided vehicle in industrial environments. *Robotics and Computer-Integrated Manufacturing*, 26(4):296–311.

Mombelli, A., Quilliot, A., and Baiou, M. (2022). Searching for a Safe Shortest Path in a Warehouse. In *Proceedings of the 11th International Conference on Operations Research and Enterprise Systems*, pages 115–122. SCITEPRESS - Science and Technology Publications.

Philippe, C., Adouane, L., Tsourdos, A., Shin, H. S., and Thuilot, B. (2019). Probability collectives algorithm applied to decentralized intersection coordination for connected autonomous vehicles. In *IEEE Intelligent Vehicles Symposium, Proceedings*, volume 2019-June, pages 1928–1934. IEEE.

Pimenta, V., Quilliot, A., Toussaint, H., and Vigo, D. (2017). Models and algorithms for reliability-oriented Dial-a-Ride with autonomous electric vehicles. *European Journal of Operational Research*, 257(2):601–613.

Ryan, C., Murphy, F., and Mullins, M. (2020). Spatial risk modelling of behavioural hotspots: Risk-aware path planning for autonomous vehicles. *Transportation Research Part A: Policy and Practice*, 134:152–163.

Vis, I. F. (2006). Survey of research in the design and control of automated guided vehicle systems. *European Journal of Operational Research*, 170(3):677–709.

Vivaldini, K. C. T., Tamashiro, G., Junior, J. M., and Becker, M. (2013). Communication infrastructure in the centralized management system for intelligent warehouses. In *Communications in Computer and Information Science*, volume 371, pages 127–136.

Wurman, P. R., D'Andrea, R., and Mountz, M. (2008). Coordinating hundreds of cooperative, autonomous vehicles in warehouses. In *AI Magazine*, volume 29, pages 9–19.