Monitoring a Fleet of Autonomous Vehicles through A* like Algorithms and Reinforcement Learning

Mourad Baiou, Aurélien Mombelli, Alain Quilliot

Monitoring a Fleet of Autonomous Vehicles through A* like Algorithms and Reinforcement Learning

Mourad BAIOU, Aurélien MOMBELLI, Alain QUILLIOT

LIMOS CNRS 6158, Labex IMOBS3, Clermont-Ferrand, France
mourad.baiou@isima.fr, aurelien.mombelli@uca.fr, alain.quilliot@isima.fr

1 Introduction

Intelligent vehicles, provided with an ability to move with some level of autonomy, recently became a hot spot in the mobility field. Still, determining what can be exactly done with new generations of autonomous or semi-autonomous vehicles able to follow their own way without being physically tied to any kind of track (cable, rail,...) remains an issue. Most people are doubtful about the prospect of seeing such vehicles moving without any external control inside crowded urban areas. Instead they foresee that the use of those vehicles is likely to be restricted to protected areas for specific purposes: relocation of free access vehicles inside large parking areas, rural or urban logistics inside closed areas, pick up and delivery transactions inside warehouses (see [1, 17]), rescue or repair interventions in a context of natural disaster.

This point of view raises the general challenge of monitoring a fleet of such vehicles, required to perform internal logistics tasks while safely interacting with other players: workers, machines and standard vehicles. Related decision problems are at the intersection of Robotics and are at the intersection of Robotics and Operations Research.

Fig. 1. An Autonomous Vehicle

When it comes to the management autonomous vehicle fleets, current trend is to the implementation of a 3-levels supervision architecture:

- The first level, or embedded level, is defined by the monitoring and sensing devices which are embedded inside the vehicles, with the purpose of controlling trajectories in real time and adapting them to the possible presence of obstacles: currently, most effort from the robotics community remains devoted to this embedded level (see [3]), which mostly involves optimal control and artificial perception techniques.

- The second one, or middle one, is in charge of small tricky areas, like for instance crossroads (see Figure 2). It sends signals and instructions to the vehicles in order to regulate their transit and avoid them to collide when they get through those areas (see [3, 10]).
- The third one, or global one, refers to the dynamic planning and routing of the fleet, in order to make this fleet achieve some internal logistics requests (see [6, 14, 16]).

A true challenge is about the synchronization of those monitoring levels and about the control of communication processes which will allow them to interact.

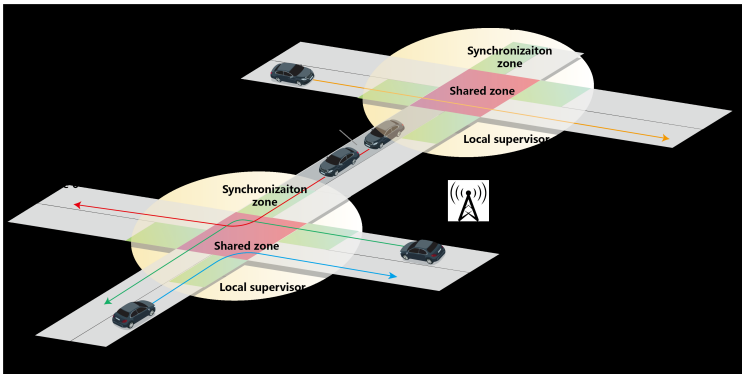

Fig. 2. Hierarchical Supervision Architecture

We deal here with the global control level, while assuming that this level is in charge of vehicle routing and scheduling decisions. At a first glance, one may think into related problem as a kind of PDP: Pick up and Delivery problem (see [2]), since an elementary task will consists for a vehicle in 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 autonomous or semi-autonomous vehicles is usually short and decisions have to be taken on line, which means that decisional processes must take into account the communication infrastructure (see [13]) 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 autonomous vehicles are involved, safety is at stake (see [12, 11]). 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 more easily.

Taking care of safety requires quantifying the risk induced by the introduction into the system of any additional vehicle. Addressing this issue means turning

real time collected traffic data into risk estimators (see [12, 15]). We do not do it here. Instead, we focus on the way resulting estimators may be used in order to take safe routing and scheduling decisions. So we assume that, at the time when we are trying to route and schedule a given vehicle V, we are provided with a procedure which, for any arc $e = (x, y)$ or the transit network and any time value t, can compute a rough estimation of the risk related to make V be running on e at time t. Then our goal becomes to schedule the route Γ that V is going to follow, in such a way that its arrival time is minimal and that induced risk estimation remains bounded by some threshold. For the sake of simplicity, we limit ourselves to one vehicle V and one origin/destination move (o, d). Our problem may then be view as the search for a constrained shortest path (see [7]). But two features make it significantly more difficult:

- We must deal with a *time dependent* network (see [4, 5, 9]);
- The *on line* context keeps us from relying on a heavy machinery like those related to mathematical programming.

According to this purpose, we proceed in 3 steps:

- The first one is devoted to the setting of our **SPR**: Shortest Path under Risk problem and to a discussion about its structural properties.
- The second step is devoted to the design of algorithms designed for a static context: almost exact algorithms which adapt well-know A* algorithm for path searching in a large state space (see [8]); local search heuristic algorithms, which estimate the quality of a given route Γ through application of a filtered dynamic programming procedure. In both case, we try several notions of arc traversal decisions, relying on respectively risk versus time, risk versus distance and distance versus time estimations.
- The last step deals with the online issue. We turn above mentioned algorithms designed according to a static paradigm into reactive algorithms for on line contexts. According to this prospect, we apply statistical learning and auto-adaptative reinforcement learning techniques, in order to associate, with any current traffic patterns, ad hoc arc traversal decisions.

So the paper is organized as follows: in Section II we formally describe our model and state some structural results. In Section III we describe the global structure of an exact A* algorithm and a local search heuristic, and present the ways the notion of arc traversal decision may be implemented. In Section IV, we address the on line issue and explain how statistical learning techniques may be used in order to turn static tree search or dynamic programming algorithms into fast decision rule based algorithms. Section V is about numerical experiments.

2 The SPR: Shortest Path under Risk Model

We refer here to a fleet of autonomous vehicles, which evolves throughout the time inside some kind of industrial infrastructure, for instance a warehouse, with the purpose of achieving internal logistic tasks (item storing and retrieving,

maintenance, inventory,...). Those tasks have to be performed in a safe way under small standard costs (time, energy,...). It comes that, in order to set **SPRC** model, we first need to formalize here the *safety notion*.

2.1 Transit Network and Risk Function

We suppose that our fleet of vehicles moves inside a simple almost planar transit network G = (N, A), N denoting the node set and A the arc set. This network G is likely to represent for instance a warehouse (see Figure 3), or any kind of similar industrial or rural restricted area.

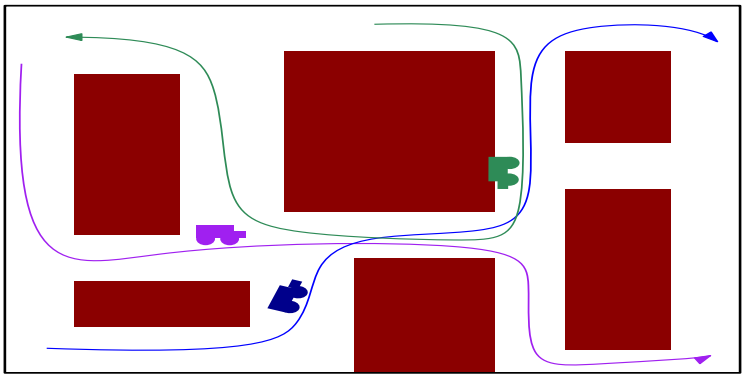

Fig. 3. A Warehouse like Transit Network

To any arc e = (x, y) corresponds a length Le and a maximal speed v_{max_e}: an autonomous vehicles traversing e is not allowed to go faster than v_{max_e} while moving along e. We denote by L^* the shortest path distance induced by values $Le, e \in A$. We suppose that at time $t = 0$, when the global supervisor of the fleet must take a decision about a target vehicle V, he is provided with some knowledge about the routes which are followed by the other vehicles, their schedule, and the tasks that they are going to perform. This knowledge allows him to derive a risk estimation $\Pi^e(t)$ function whose meaning comes as follows: For any small value dt, $\Pi^e(t).dt$ is an estimation of the Expected Damage in case V moves at maximal speed v_{max_e} along e between time t and time $t + dt$.

Obtaining functions Π^e is not part of this study: it requires experimental data analysis. But, since the global supervisor must maintain those risk estimation functions all along the process, those functions must be expressed according to a simple format. So we make the assumption that any function Π^e is piecewise linear (see Figure 4).

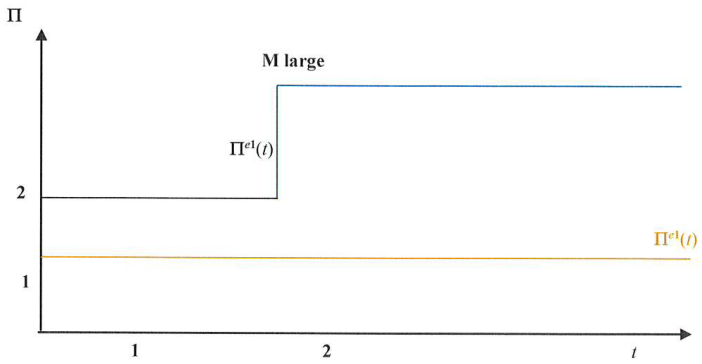

Fig. 4. A piecewise function Π^e

As a matter of fact, if additional vehicle V moves across arc e at a speed v less than maximal speed v_{max_e}, induced risk will decrease. We assume that, if V traverses arc e during some interval $[t, t + dt]$ at speed $v \leq v_{max_e}$, then related Expected Damage is given by Equation 1:

$$Risk(v, t) = \Phi\left(\frac{v}{v_{max_e}}\right).\Pi^e(t).dt \tag{1}$$

where Φ is an increasing convex function with values in $[0, 1]$, such that for any u, $\Phi(u) \ll u$, which mean that $\Phi(u)$ is significantly smaller than u. The meaning of condition $\Phi(u) \ll u$ is that, since going slower implies for vehicle V a larger traversal time, $\Phi(u) \ll u$ will also implies that the risk induced by the traversal of e decreases while the speed decreases, even if the traversal time increases. In the next sections, we shall set $\Phi(u) = u^2$.

It comes that if vehicle V moves across arc e between time T and time $T + \delta$, according to speed function $t \mapsto v(t)$, then related *Expected Damage* is given by Equation 2:

$$\int_T^{T+\delta} \Phi\left(\frac{v}{v_{max_e}}\right).\Pi^e(t). \tag{2}$$

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 here that, for any arc e, $v_{max_e} = 1$. According to this we deal with reduced speed values $u \in [0, 1]$ and L_e means the minimal traversal time for arc e.

2.2 Routing Strategies and the SPR Problem

Let us suppose now that origin o and destination d are given, which are both nodes of the transit network $G = (N, A)$. A routing strategy from o to d for additional vehicle V, is going to be defined by a pair (Γ, u), where:

- Γ is a path from o to d in the network G.
- u is a speed function, which, to any time value $t \geq 0$, makes correspond the reduced speed $u(t) \leq 1$ of the vehicle V. Notice that if we refer to the previously described speed normalization process the true speed $v(t)$ of V is going to depend on the arc e where V is located.

Such a routing strategy (Γ, u) being given, path Γ may be viewed in a standard way as a sequence $e_1, \ldots, e_n$ of arcs of G. If we set $t_0 = 0$ and denote by t_i when V arrives to the end-node of e_i, then values t_i are completely determined by speed function $t \mapsto u(t)$. Then we set:

- $Time(\Gamma, u) = t_n = \textit{global duration}$ induced by the routing strategy (Γ, u)
- $Risk(\Gamma, u) = \sum_i \int_{t_{i-1}}^{t_i} \Phi(u(t))\Pi^e(t)dt = \textit{global risk}$ induced by (Γ, u).

Then the SPR: Shortest Path Under Risk comes in a natural way as follows:

{**SPR: Shortest Path Under Risk**: Given origin o and destination d, together with some threshold R_{max}, compute a routing strategy (Γ, u)such that $Risk(\Gamma, u) \leq R_{max}$ and $Time(\Gamma, u)$ is the smallest possible}.

2.3 Some Structural Results

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

Proposition 1. *Optimal solution (Γ, u) of **SPR** may be chosen in such a way that u 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 $\Pi_i^e, i = 1, \ldots, n$.*

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 $u(t)$ is not constant between δ_1 and δ_2 then we may replace $u(t)$ by the mean value u^* of function $t \mapsto u(t)$ between δ_1 and δ_2. Time value $Time(\Gamma, u)$ remains unchanged, while risk value $Risk(\Gamma, u)$ decreases because of the convexity of function Φ. So we conclude.

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

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: $n = 1$.
  Let us suppose above assertion to be false. Breakpoints of $e = e_1$, may be written $t_0 = 0, t_1, \ldots, t_Q = Time(\Gamma, u)$, and we may set:

- $q_0 = largest\ q$ such that $u < 1$ between t_q and t_{q+1};
- $u_0 = related\ speed$; $l_0 = distance\ covered$ by V at time t_{q_0}.

Let us increase u_0 by $\epsilon > 0$, such that $u_0 + \epsilon \leq 1$ and that induced additional risk taken between t_{q_0} and t_{q_0+1} does not exceed $R_{max} - Risk(\Gamma, u)$. 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 $u = 1$, and so arrives at the end of e before time t_Q, without having exceeded the risk threshold R_{max}. We conclude.

- Second case: $n > 1$.
  Let us suppose above assertion to be false and denote by R_1 the risk taken at the end of arc e, and by t_1 related time value. Induction applied to arcs $e_2, \ldots, e_n$, and risk threshold $R_{max} - R_1$ implies that the speed of V is equal to 1 all along the arcs $e_2, \ldots, e_n$. Let us denote by $\tau_0 = 0, \tau_1, \ldots, \tau_Q$ the breakpoints of e_1 which are between 0 and t_1 and let us set $\tau_{Q+1} = t_1$ and:
  - $q_0 = largest\ q$ such that $u < 1$ between τ_q and τ_{q+1};
  - $u_0 = related\ speed$; $l_0 = distance\ covered$ by V at time t_{q_0+1}.

  Then we increase u_0 by $\epsilon > 0$, such that $u_0 + \epsilon \leq 1$ and that induced additional risk taken between τ_{q_0} and τ_{q_0+1} does not exceed $(R_{max} - \frac{Risk(\Gamma,u))}{2}$. While moving at speed $u_0 + \epsilon$ along e_1, vehicle V faces 2 possibilities: either it arrives at the end of e_1 before time τ_{q_0+1} or it may keep on moving from time τ_{q_0+1} on along e_1 at speed $u = 1$. In any case, it reaches the end of e_1 at some time $t_1 - \beta, \beta < 0$, with an additional risk no larger than $(R_{max} - \frac{Risk(\Gamma,u))}{2}$. So, for any $i = 2, \ldots, n$ we compute speed value u_i such that moving along e_i at speed u_i between $t_{i-1} - \beta$ and t_{i-1} does not induce an additional risk more than $(R_{max} - \frac{Risk(\Gamma,u))}{2n}$. So we apply to V the following strategy: move as described above on arc e_1 and next, for any $i = 2, \ldots, n$, move along e_i at speed u_i between $t_{i-1} - \beta$ and t_{i-1} and next at speed 1 until the end of e_i. The additional risk induced by this strategy cannot exceed $(R_{max} - Risk(\Gamma, u))$. On another side, this strategy makes vehicle V achieve its trip strictly before time t_n. We conclude.

Proposition 3. *Given an optimal **SPR** trajectory (Γ, u), with $\Gamma = \{e_1, \ldots, e_n\}$ and u 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 $u = u(t) < 1$, the quantity $\Phi'(u(t)).\Pi^{e_q}(t)$ is independent on t, where $\Phi'(u)$ denotes the derivative of Φ in u.*

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 $\Pi_q^{i_q}$ related value of Π^{e_q} on the interval $]\delta_j, \delta_{j+1}[$, by $u_0, \ldots, u_q, \ldots, u_{H(q)}$, 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 $(u_0, \ldots, u_{H(q)})$ is an optimal solution of the following convex optimization problem:

- Compute $(u_0, \ldots, u_{H(q)})$ such that $\sum_q u_q.(\delta_{q+1} - \delta_q)$ and which minimizes $\sum_q \Phi(u_q)\Pi_q^{e_i}(\delta_{q+1} - \delta_q)$.

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 $u_q < 1, \Phi'(u_q).\Pi_q^{e_i} = \lambda$. As a matter of fact, we see that λ cannot be equal to 0. We conclude.

Remark 1. In case $\Phi(u) = u_2$, above equality $\Phi'(u_q)\Pi_q^{e_i} = \lambda$ becomes $u_q\Pi_q^{e_i} = \frac{\lambda}{2}$ where $u_q\Pi_q^{e_i}$ means the instantaneous risk per distance $\frac{dR}{dL}$ value at the time when V moves along e_i between times δ_q and δ_{q+1}.

2.4 A consequence: Risk Versus Distance Reformulation of the SPR Model

Remark 1 leads us to define the Risk versus Time coefficient for arc e_i as the value $2\Phi'(u_q)\Pi_q^{e_i}$ involved in Proposition 3. This proposition, combined with Proposition 1, allows us to significantly simplify **SPR**: 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_e^{RD} associates, with any arc e in Γ, *Risk versus Distance coefficient* $\lambda_e^{RD} = 2\Phi'(u)\Pi_e$. In case $\Phi(u) = u_2$, we notice that this coefficient means the amount of risk per distance unit induced on arc e at any time t such that $u(t) < 1$, by any trajectory (Γ, u) which satisfies Proposition 3.

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

$$u(t) : \begin{cases} \Phi'^{-1}(\frac{\lambda_e^{RD}}{2\Pi_e}), & \text{if } \Phi'^{-1}(\frac{\lambda_e^{RD}}{2\Pi_e}) < 1 \\ 1, & \text{otherwise} \end{cases} \tag{3}$$

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

Risk versus Distance SPR 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.

2.5 Discussion about the Complexity

The time dependence of the transit network together with the proximity of the **SPR** model with Shortest Path Constraint models suggests that **SPR** is a complex problem. Practical difficulty of **SPR** may be captured through the following example, which makes appear that if (Γ, u) defines an optimal **SPR** trajectory, the risk per distance value $\lambda_e^{RD} = 2\Phi'(u(t))\Pi_e$ 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 $t > 1$. $R_{max} = \frac{3}{4}$; Function $\Phi is : u \mapsto \Phi(u) = u^2$.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 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 $Risk(\Gamma, u) = \frac{3}{4}$ and $Time(\Gamma, u) = 2$.

Sill, identifying the complexity of **SPR** is not that simple, since we are dealing with continuous variables. As a matter of fact, complexity also depends on function Φ. We conjecture that:

Conjecture 1. If $\Phi(u) = u^2$ then **SPR** is in NP time and is NP-Hard.

3 Algorithms

Our algorithms all rely on notions of state and decision. A state is a 3-uple (i, T, R), where:

- i is a node of G where vehicle V is currently located;
- T is the times spent in order to reach i, and R is the amount of risk induced by this process of moving from origin o to node i.

Then a decision will consist in:

- Choosing the arc $e = (i, i^o)$ along which the vehicle is going to move;
- Choosing some parameter λ which is going to determine the speed function u along the arc e.

Previous section 2 suggests the use of risk versus distance coefficient λ_e^{RD} as decision parameter λ. But other choices are possible. We restrict ourselves to the case when $\Phi(u) = u^2$.

3.1 Decision Scheme

As told above, a natural approach is to refer to Proposition 3 and consider $\lambda = \lambda^{RD}$ as expressing the mean Risk versus Distance coefficient $\Phi'(u)\Pi^e$. But another intuitive approach is to consider $\lambda = \lambda^{SP}$ as expressing the mean speed of V along e, and deduce this way the arrival time on i^o in a straightforward way. Finally, we may also consider that $\lambda = \lambda^{RS}$ expresses the mean Risk Speed of V along e, which means the amount of risk vehicle V takes per time unit as it advances along e. We are going to describe here those 3 possibilities, together with the way resulting state (i^o, R^o, T^o) may be deduce from λ and (i, R, T).

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

Since $Phi(u) = u^2$, $\Phi'(u(t))\Pi^e(t) = 2u(t)\Pi^e(t)$ for any t during the traversal of e. It comes that if we fix λ^{RD} the speed value $u(t)$ is given by: $u(t) = Inf(1, \lambda^{RD}/\Pi^e(t))$. Resulting state (i^o, R^o, T^o)will be obtained from λ^{RD} and (i, R, T) through the following iterative process:

Risk_Distance Transition procedure:
Let us set $t_0 = T$, and let us denote by $t_1, \ldots, t_Q$ the breakpoints of Π^e which are larger than T and by $\Pi^e 0, \ldots, \Pi^e_Q$ related Π^e values.
Initialization: $t \leftarrow t_0$; $r \leftarrow R$; $L \leftarrow 0$; $q \leftarrow 0$; $Not\ Stop$;
While $Not\ Stop$ do
$\pi \leftarrow \Pi^e_q$; $q \leftarrow q + 1$; $\delta \leftarrow t_q - t$; $u = Inf(1, \lambda^{RD}/\pi)$;
If $L_e > L + u\delta$ then
$L \leftarrow L + u\delta$; $r \leftarrow r + \Phi(u)\pi\delta$; $t \leftarrow t_q$;
Else
$\delta \leftarrow \delta \frac{L_e - L}{u\delta}$; $t \leftarrow t + \delta$; $r \leftarrow r + \Phi(u)\pi\delta$; $L \leftarrow L_e$; $Stop$;
$R^o \leftarrow r$; $T^o \leftarrow t$;
If $R^o > R_{max}$ then Fail else Success;

- **Second approach:** *The Mean Speed approach.*

Fixing λ^{SP} means fixing the time T^o as: $T^o = T + \frac{Le}{\lambda^{SP}}$. In order to determine the function $t \mapsto u(t)$ and the value R^o, we solve the following quadratic program:

Mean_Speed Program:
Let us denote by $t_0 = T$, $t_1, \ldots, t_q = T^o$ the breakpoints of Π^e which belong to $[T, T^o]$ and by $\Pi^e_1, \ldots, \Pi^e_Q$ related $\Pi^e values$.
Then we must compute speed values $u_1, \ldots, u_Q \in [0, 1]$ such that:
$\sum_q u_q(t_q - t_{q-1}) = T^o - T$
$\sum_q u_q^2 \Pi^e_q(t_q - t_{q-1}) < R_{max}$.

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

- **Third approach:** *The Risk Speed approach.*

Since $\Phi(u) = u^2$ we have that at any time t during the traversal of e, related risk speed $\frac{dR}{dT(t)}$ is equal to $u(t)^2\Pi^e(t)$. It comes that if we fix λ^{RS} we get: $u(t) = Inf(1, \left(\frac{\lambda^{RS}}{\Pi^e(t)}\right)^{1/2})$.
Resulting state (i^o, R^o, T^o) will be obtained from λ^{RD} and (i, R, T) through the same following iterative process a for the Risk versus Distance approach.

3.2 A Local Search Algorithm Involving Dynamic Programming.

This local search heuristic **LS_SPR** works in 2 steps:

LS_SPR Algorithm:
Initialize Γ as the shortest path according to L from o to d; *Not Stop*;
While *Not Stop* do
First step: *Evaluate* Γ, and get the arrival time T_i of vehicle V in any node i of Γ;
Second step: *Update* Γ;
Keep the best path solution Γ ever obtained.

Several kinds of controls may be applied to above process: one may do a random walk descent. In any case, we need to discuss both Update and Evaluate steps.

- **Update step**: It relies on a pre-process which is applied to the transit network G and involves some proximity threshold S_Prox. For any two nodes i, j of G such that $L^*_{i,j} \leq S\_Prox$, we pre-compute a collection $Path_{i,j}$ of elementary path from i to j. This provides us with an operator $Detour$, which acts on any path Γ though parameters i, j, γ as follows:
  - i, j are nodes of Γ such that i precedes j in Γ; γ is some path in $Path_{i,j}$;
  - $Detour(\Gamma, i, j, \gamma)$ replaces the restriction $\Gamma_{i,j}$ of Γ from i to j by path γ.

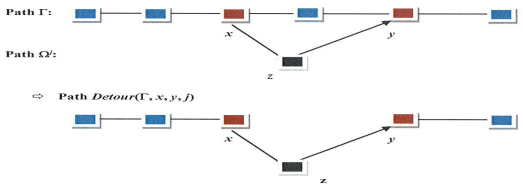

Fig. 5. Detour Operator

Since *Detour* may admit a rather large number of parameters values (i, j, γ), we first identify pairs of nodes (i, j) in Γ, such that the slowdown coefficient $\left(\frac{T_j - T_i}{T^*_{i,j}}\right)$ is large, and pick up such a pair (i, j). Next we choose path γ in $Path_{i,j}$ under the condition that is not very crowded between time T_i and time T_j, that means which is such that the sum, for the arcs e of γ of mean $\Pi^e(t)$ values between time T_i and time T_j is small.

- **Evaluation step**: This evaluation step relies on a dynamic programming procedure **DP_Evaluate** whose main features come as follows:
  - Let us denote by $e_1, \ldots, e_n$ the arcs of Γ, and by $i_0, \ldots, i_n$ related nodes; So the time space of **DP_Evaluate** comes in a natural as the set $\{0, 1, \ldots, n\}$ and a state at time $q = 0, 1, \ldots, n$, is a pair (T, R), where T means the time when vehicle V arrives in i_q, and R the cumulative risk at this time. Clearly, initial state is $(0, 0)$ and final state should be any pair (T, R) such that $R \leq R_{max}$.
  - Then a decision at time q becomes a value λ, $(\lambda^{RD}, \lambda^{SP}, \lambda^{RS})$ in the sense of Section 3.1 and such a decision induces a transition $(q, R, T) \rightarrow (q+1, R^o, T^o)$ as described in Section 3.1, with cost $R^o - R$. This decision is feasible if it does induce a Fail result.

According to this, Bellman principle may be applied:
the algorithm **DP_Evaluate** scans the time space $\{0, 1, \ldots, n\}$, and, for any $q = 0, 1, \ldots, n$, computes related state set $State[q]$, according to the following instructions:

- Initialize $State[0]$ as $\{(0, 0)\}$ and $State[q]$ as Nil for any $q > 0$;
- For $q = 1, \ldots, n-1$ do
  Generate decision set Λ;
  For any λ in Λ and any state (T, R) in $State[q]$ do (I1)
  Compute (in case λ is feasible) resulting state (T^o, R^o);
  If there does not exist (t_1, R_1) in $State[q+1]$ such that $t_1 \leq T^o$ and $R_1 \leq R^o$ then insert (T^o, R^o) into $State[q+1]$ and remove from $State[q+1]$ any (t_1, R_1) such that $t_1 \geq T^o$ and $R_1 \geq R^o$.

In case we are already provided with some feasible **SPR** solution (Γ^*, u^*) with value T^*, then we may apply the following filtering rule:

- Lower Bound Based Filtering Rule: Let (T^o, R^o) be the state involved in instruction (I1). If $T^o + L^*_{i(q+1),d} \geq T^*$ then state (T^o, R^o) may be killed: we do not insert it into $State[q+1]$, since we cannot expect it to be extended into a better solution than current solution (Γ^*, u^*).

Remark 2. We turn **LS_SPR** algorithm into a greedy algorithm by removing the update step and by generating Λ in such a way its cardinality is 1.

3.3 A A* Algorithm

A* algorithm [8] was designed in order to deal with path search for robots evolving in very large (possibly infinite) state spaces. It can be adapted to our problem, since solving **SRP** means searching for a shortest path in a Risk Expanded network, whose nodes are pairs $(i, T, R), i \in N, 0 \leq R \leq RMax, T \geq 0$, and arcs corresponds to transition $((i, T, R) \rightarrow decision\lambda \rightarrow (i^o, T^o, R^o))$ as described in Section 3.1. In the present case, it will rely on the following data structures:

- An expansion list LE, which contains states (i, T, R), ordered according to increasing optimistic estimation value W. The optimistic estimation value of state (i, T, R) is equal to $T + L^*_{i,d}$ and provides us with a lower bound to the best possible value of a **SPR** solution (Γ, u) which would extend the path which allowed us reaching state (i, T, R).
- A pivot list $LPivot$, which contains states (i, T, R), together with optimistic estimation value W, which already appeared as the first element ($Head$) of LE. There should not exist in LE an element (i, T, R) which is dominated by another element (i, T_1, R_1) in $LPivot \bigcup LE$, that means which is such that $T_1 \leq T$ and $R_1 \leq R$.

Then A*_SPR algorithm may be described as follows:
A*_SPR Algorithm:
Initialize $LPivot$ as Nil and LE as $\{(o, 0, 0)\}$; $Not\ Stop$;
While $(NotStop) \wedge (LE \neq Nil)$ do

$(i, T, R) \leftarrow Head(LE)$;
If $i = d$ then
  Stop; Retrieve the **SPR** solution Γ related to (i, T, R);
Else
  Remove (i, T, R) from LE and Insert it into $LPivot$;
  Generate $\lambda_{decision} set\Lambda$; (I2)
  For any arc $e = (i, i^o)$ and any λ in Λ do
    Compute resulting state (i^o, T^o, R^o) together with value $W^o = T^o + L*_{i^o,d}$;
    If $R^o \leq RMax$ and if there does not exist (i^o, T_1, R_1) in $LPivot \bigcup LE$ such that such that $T_1 \leq T^o$ and $R_1 \leq R^o$ then
      Insert (i^o, T^o, R^o) into LE and remove from LE any (i^o, T_1, R_1) such that $T_1 \geq T^o$ and $R_1 \geq R^o$. Do it in such a way that LE remains ordered according to optimistic estimation values W;

Remark 3. If we are able to generate all decisions likely to appear inside a given optimal decision sequence, then above algorithm A*_SPR is optimal.

Remark 4. We turn A*_SPR algorithm into a shortest path algorithm by reducing Λ to 1 element.

3.4 Discussion: The Decision Set Λ

Both above algorithms rely on an instruction 'Generate $\lambda - decisionset\Lambda$'. But λ values are continuous ones. So, we must decide about the way we generate a finite $\lambda - decisionset\Lambda$.

The simplest case is the case when we deal with Risk versus Distance decisions λ^{RD}, since in such a case, propositions Proposition 2 and Proposition 3 suggests us that a mean value for λ^{RD} is going to be given by $\lambda^{RD}_{mean} = \frac{R_{max}}{L^*_{o,d}}$. Then a natural way to generate Λ is to fix an odd number $2.K + 1$ of λ^{RD} values, a geometric step value $\delta > 0$, and to set:

$$\Lambda = \{\lambda^{RD}_{mean}\} \wedge \{(1+\delta)^k.\lambda^{RD}_{mean}, k = 1, \ldots, K\} \wedge \{(1+\delta)^{-k}.\lambda^{RD}_{mean}, k = 1, \ldots, K\} \quad (4)$$

According to this, Λ is determined by K and δ. We may consider K as a flexible parameter. As for the choice of value δ, it becomes determined by K and by the minimal and maximal values $\lambda^{RD}_{max} = (1+\delta)^k \lambda^{RD}_{mean}$ and $\lambda^{RD}_{min} = (1+\delta)^{-k} \lambda^{RD}_{mean}$ which we want to assign to λ^{RD}. If we want to allow the vehicle to move with a speed twice as large as the speed suggested by λ^{RD}_{mean}, then we see that we must choose a value $\lambda^{RD}_{max} = 2\lambda^{RD}_{mean}$. That means that δ is determined by the acceleration coefficient ρ we may want to apply to the vehicle in order to make him possible to reach the end of an arc before some risky situation occurs on this arc. We shall test for instance $\rho \in [2, 3, 4]$.

In the case of Risk over Time and Mean Speed decisions λ^{RS} and λ^{SP}, we must arbitrarily fix mean values λ^{SD}_{mean} and λ^{SO}_{mean}, and try to learn them throughout the computational process. This opens the way to next section.

4 Speeding Algorithms through Statistical Learning Techniques

We consider here two ways of speeding our algorithms in order fit with a dynamic contexts. The first one impose a small number K of possible decisions and the second one is to bound the number of states (T, R) related to any node i. We do it while focusing on the case $\lambda = \lambda^{RD}$.

4.1 Bounding Decisions

Once acceleration parameter ρ has been tuned, controlling the size of decision set Λ means fixing value K. If we set $K = 1$ (greedy algorithm in the case of **DP_Evaluate** and shortest path algorithm in the case of **A*_SPR**), then the choice is about λ^{RD}_{mean}, which, in a first approach, should be equal to $\frac{R_{max}}{L^*_{o,d}}$. If $K \neq 1$, then we apply the following statistical learning process:

- We apply **DP_Evaluate** to instances which fit parameter ρ, while using some reference decision number K_{ref}. For any instance I, we retrieve the optimal decision sequence $\{\lambda_1, \ldots, \lambda_n\}$, Every decision λ_i is related to some number k_i in $-K_{ref}, \ldots, 0, \ldots, K_{ref}$.
- Then we compute, for every value k in $\{-K_{ref}, \ldots, 0, \ldots, K_{ref}\}$, the percentage $\tau(k)$ of occurrence of k in those decision sequences.
- Finally, K being the target decision number, we split the decision range $[\lambda^{RD}_{min}, \lambda^{RD}_{max}]$ for possible decisions λ^{RD} into $2(K+1)$ intervals corresponding to same percentages of decisions λ_i. For instance, if $K = 1$, we split interval $[\lambda_{min}, \lambda_{max}]$ into 4 intervals $[\lambda_{min}, \lambda^{\frac{1}{4}}], [\lambda^{\frac{1}{4}}, \lambda^{\frac{1}{2}}]$, $[\lambda^{\frac{1}{2}}, \lambda^{\frac{3}{4}}]$ and $[\lambda^{\frac{3}{4}}, \lambda_{max}]$ in such a way that:
  - $1/4$ of decisions λ_i belong to interval $[\lambda_{min}, \lambda_1 4]$;
  - $1/4$ of decisions λ_i belong to interval $[\lambda^{\frac{1}{4}}, \lambda^{\frac{1}{2}}]$;
  - $1/4$ of decisions λ_i belong to interval $[\lambda^{\frac{1}{2}}, \lambda^{\frac{3}{4}}]$;
  - $1/4$ of decisions λ_i belong to interval $[\lambda^{\frac{3}{4}}, \lambda_{max}]$.

Then restricted Λ becomes the set $\{\lambda^{\frac{1}{4}}, \lambda^{\frac{1}{2}}, \lambda^{\frac{3}{4}}\}$.

4.2 Bounding States

In order to filter state set $State[i]$ related to a given node i and impose a prefixed lower bound S on the cardinality of $State[i]$, several techniques may be applied. One may for instance consider as equivalent 2 states (T, R) and (T', R') if $|T - T'| + |R - R'|$ does not exceed some rounding threshold. We are not going to follow this approach which does not guaranty that we are going to maintain the cardinality of $State[i]$ below imposed threshold S. Instead, we are going to do as if there were existing a natural conversion rate ω which turns risk into time. According to this, we are going to rank pairs (T, R) $State[i]$ according to increasing values $\omega T + R$ and keep on with the S best ones according to this ordering, while killing the others.

Key issue here becomes about the value of ω. Intuitively, ω should be equal to $\frac{R_{max}}{T^o}$, where T^o is the optimal **SPR** value, and we should be able to learn this value as a function of the main characteristics of **SPR** instances: most relevant characteristics seem to be risk threshold R_{max}, the length (expected length) L^* of path Γ, the mean value Δ of functions $\Pi^e, e \in A$, and the frequency B of the breakpoints of those functions. We may notice that in case all functions Π^e are constant and equal to some value Δ, then speed u is going to be constant and equal to $\frac{R_{max}}{L^*\Delta}$, and so that time value T^o will be equal to $\frac{L^*}{u} = \frac{\Delta L^{*2}}{R_{max}}$. This will lead to initialize ω as $\omega = \frac{R_{max}^3}{\Delta L^{*2}}$.

In order to refine this initial choice for ω we retrieve, for any instance, the optimal decision sequence $\{\lambda_1, \ldots, \lambda_n\}$, related state sequence $s_1 = (T_1, R_1), \ldots, s_n = (T_n, R_n)$. Then, for any such an instance, we look for the value ω which statistically makes states s_i be always among the best ones for the ranking related to quantity $\omega T + R$. As a matter of fact, while performing numerical experiments, we focus on an estimation of optimal value T^o as a function of R_{max}, L^*, B and Δ, and next test the ranking of states s_i among the state subsets $State[i]$ for $\omega = \frac{R_{max}}{T^o}$.

Bounding States through Reinforcement Learning. Still, this way of performing learning may induce distortions. A lack of flexibility in the pruning procedure associated with a non fully well-fitted value ω may yield, for a given node i, a collection $State[i]$ poorly balanced, in the sense that one would expect related values (T, R) to distribute themselves as a wide Pareto set. More precisely, we may qualify a pair (T, R) as risky if $\frac{R}{\sum_{j \geq i+1} L_j}$ is large with respect to $\frac{R_{max}}{L^*}$, or cautious if the converse holds. Then it may happen that our pruning technique yields pairs (T, R) which, taken as a whole, are either too risky or too cautious. In order to control this kind of side-effect, we make ω value become auto-adaptive.

More precisely, we start, as previously explained, from some pre-learned ω value, and make it evolve through Reinforcement Learning, that means throughout an (or several) execution the **DP_Evaluate** (or **A*_SPR** or **LS_SPR**). In order to explain it better, we focus on the case of **DP_Evaluate** algorithm, while supposing that state threshold S has been fixed and that initial value ω has been computed as described above. So let us suppose that, at some time during the process, we just dealt with arc e_i and so computed current state set $State[i]$, while updating value ω. Applying decisions of Λ and filtering resulting states (T, R) through Bellman principle provides us with a state subset $State[i+1]$ whose size is likely to exceed S. Then we rank states (T, R) of $State[i+1]$ according to $\omega T + R$ values. Ideally, states (T, R) ordered this way should make S best states (T, R) be balanced in the sense that risky states should get along with cautious ones, or, in other words, that the ratio $\frac{R}{R_{max}}$ should be centered around the ratio $\frac{L^*_{0,i}}{L^*}$. If, for instance, those values are centered significantly above this ratio, then we are moving in a too risky way and must make ω decrease. Con-

versely, if those best values are centered below this ratio, then we are too careful and must make ω increase.

We implement this principle by performing a kind of statistical analysis of those best values in $State[i+1]$, in order to derive, from those S best states (T,R), an indicator $Risk\_Balance$, which takes symbolic values $\{Risky, Normal, Careful\}$ depending on the way the mean $\frac{R}{R_{max}}$ value is located with respect to $\frac{L^*_{0,i}}{L^*}$. Then our *filter&learn* **Filter_Learn** works as follows:

Filter_Learn Procedure:
Rank states (T,R) of $State[i+1]$ according to $\omega T + R$ values;
Select S best pairs (T,R) according to this ranking and compute $Risk_Balance$;
If $Risk_Balance = Normal$ then
  Keep only the S best states in $State(i+1)$;
If $Risk_Balance = Risky$ then
  Split $State[i+1]$ into 2 subsets S_1 and S_2 with same size: S_1 is made of the best states (S,R) according to our tanking and $S_2 = S - S_1$;
  Keep only the $\frac{S}{2}$ best states in S_1 and S_2 in $State(i+1)$; Make ω decrease;
If $Risk_Balance = Cautious$ then
  proceed as in previous case, while making ω increase.

Adaptation ot the State Bounding Scheme to A*_SPR Algorithm. In the case of the **A*_SPR** algorithm, we apply the same principle, with the idea that, in the list $\overline{LE} \wedge LPivot$, the number of states (i,T,R) related to a given i should not exceed some target threshold S. We structure elements of LE and $LPivot$ according to lists $LE[i]$ and $LPivot[i]$, where each list $LE[i]$ and $LPivot[i]$ is a list of states (T,R) in the **DP_Evaluate** sense and apply the *learn&filter* process only to the $LE[i]$ lists.

5 Numerical Experiments

Goal: We perform numerical experiments with the purpose of studying

- the behavior of static **DP_Evaluate**, **LS_SPR**, **A*_SPR** algorithms of Section 3. We pay special attention to the dependence of those algorithms to the choice of the decision mode (Mean Speed, Risk versus Time, Risk versus Distance), to the characteristics of decision set Λ;
- the way we may efficiently turn those static algorithms into efficient dynamic algorithms through the use of statistical an reinforcement learning techniques described in Section 4.

Technical Context: Algorithms were implemented in Python3.7.6 in a Docker environment on an Intel i5-9500 CPU at 4.1GHz.

Instances: We generated networks (N,A) as connected symmetric partial grids, which means grids $n * n$, modified through removal of 25% of its arcs. Those partial grids are summarized through their number $|N|$ of nodes and their number $|A|$ of arcs. Length values $L_e, e \in A$, are uniformly distributed

between 3 and 10. Function Φ is taken as function $u \mapsto \Phi(u) = u^2$. Function Π^e are generated by fixing a time horizon T_{max}, a mean frequency B of break points t_i^e, and an average value Δ for value $\Pi^e(t)$: More precisely, values Π^e are generated within a finite set $\{2\Delta, \frac{3\Delta}{2}, \Delta, \frac{\Delta}{2}, 0\}$. As for threshold R_{max}, we notice that if functions Π^e are constant with value Δ and if we follow a path Γ with length L_{diam}, where L_{diam} is the diameter of network G, at speed $\frac{1}{2} = \frac{v_{max}}{2}$, then the expected risk is $\frac{L_{diam}\Delta}{2}$. It comes that we generate R_{max} as a quantity $\alpha\frac{L_{diam}\Delta}{2}$, where α is a number between 0.2 and 2. Finally, since an instance is also determined by origin/pair (o, d), we denote by L^* the value $L^*_{o,d}$.

Table 1 presents a package of 12 instances with their characteristics.

Table 1. Instances' characteristics

Instance	$\|N\|$	$\|A\|$	B	Δ	α	L^*
1	16	88	3	2.02	0.2	34.6
2	16	60	3	2.04	1	35.7
3	16	76	3	2.02	2	42.7
4	16	80	9	1.98	0.2	32.3
5	16	76	9	2.00	1	30.2
6	16	76	9	2.00	2	43.3
7	100	560	3	1.99	0.2	108.6
8	100	580	3	1.97	1	109.5
9	100	544	3	2.01	2	113.9
10	100	520	9	2.01	0.2	124.2
11	100	528	9	2.01	1	107.6
12	100	548	9	2.00	2	104.5

Outputs related to the behavior of the procedure DP_Evaluate

For every instance, L^* value is the length of path Γ in the L sense. We apply **DP_Evaluate** while testing the role of parameters $\lambda = \lambda^{RD}, \lambda^{SP}, \lambda^{RS}$, as well as K and δ. So, for every instance, we compute:

- in **Table 2**: The risk value R_DP^{mode}, the time value T_DP^{mode}, and the CPU times (in s.) CPU^{mode}, induced by application of **DP_Evaluate** with $\lambda^{mode} = \lambda^{RD}, \lambda^{SP}, \lambda^{RS}$, $K = 10$, $\rho = 4$;
- in **Table 3**: For the specific mode λ^{RD}, related number State of states per node i, together with time value T^{RD}, when $K = 1, 3, 5, 7, 10$ and $\rho = 4$;
- in **Table 4**: For the specific mode λ^{RD}, related number State of states per node i, together with time value T^{RD}, when $K = 10$ and $\rho = 1.5, 2, 3, 4, 8$.

CPU times are in seconds.

Table 2. Impact of λ^{mode}, with $K = 10$ and $\rho = 4$

Instance	R^{RD}	T^{RD}	cpu^{RD}	R^{SP}	T^{SP}	cpu^{SP}	R^{RS}	T^{RS}	cpu^{RS}
1	6.74	104.0	0.48	6.90	117.7	0.73	4.58	112.6	0.47
2	36.10	37.2	0.62	35.66	38.7	0.52	34.00	38.9	0.78
3	85.48	41.3	0.72	77.57	44.7	0.86	81.28	41.7	1.02
4	5.42	99.4	0.51	6.41	105.0	1.57	3.96	102.5	0.49
5	30.07	41.5	0.97	30.18	45.0	1.99	30.18	43.7	2.03
6	84.62	34.5	1.10	79.93	37.1	1.89	86.67	36.1	2.22
7	7.99	362.8	4.02	21.58	390.2	5.66	6.27	370.1	5.28
8	107.73	144.6	5.88	107.68	177.6	7.27	91.08	164.1	9.05
9	227.87	98.3	6.61	227.57	99.6	8.47	228.39	98.2	9.26
10	11.44	410.5	8.42	74.17	423.1	13.07	8.52	388.7	13.59
11	106.38	143.7	10.16	107.61	167.6	20.06	88.08	157.5	19.35
12	203.65	87.5	10.55	202.81	91.7	19.53	207.34	86.3	23.54

Table 3. Impact of K, with $\rho = 4$

K	1		3		5		7		10	
instance	T^{RD}	*States*	T^{RD}	*States*	T^{RD}	*States*	T^{RD}	*States*	T^{RD}	*States*
1	111	10.67	108	28.33	128	36.67	130	44.00	104	60.50
2	46	17.00	38	38.17	40	47.50	37	59.67	37	66.00
3	43	13.00	41	38.17	41	49.17	41	62.67	41	72.33
4	99	13.50	98	24.83	100	35.50	100	42.00	99	49.00
5	51	15.83	44	37.50	42	48.17	52	60.33	42	78.33
6	43	15.83	35	31.50	35	49.83	35	56.67	34	68.83
7	362	23.17	365	34.50	365	49.56	366	52.89	363	65.78
8	148	25.17	178	41.11	175	57.83	175	63.50	145	103.83
9	111	25.06	107	46.39	99	63.50	99	75.50	98	91.50
10	420	24.10	392	48.00	466	60.90	466	74.80	410	94.20
11	171	28.22	147	50.72	164	60.67	157	70.22	144	107.44
12	103	26.11	85	47.67	88	66.06	84	74.83	88	94.11

Table 4. Impact of ρ, with $K = 10$

ρ	1.5		2		3		4		8	
instance	T^{RD}	*States*	T^{RD}	*States*	T^{RD}	*States*	T^{RD}	*States*	T^{RD}	*States*
1	132	21.50	120	39.33	128	36.67	108	42.17	112	37.83
2	54	31.83	50	42.33	40	47.50	40	47.33	46	42.83
3	43	46.83	42	49.50	41	49.17	41	45.50	42	42.33
4	113	18.67	107	27.33	100	35.50	96	37.67	96	34.83
5	53	45.33	52	42.83	42	48.17	46	45.17	49	42.00
6	38	41.83	36	50.67	35	49.83	36	42.50	37	39.50
7	416	22.39	411	37.06	365	49.56	355	55.39	348	51.78
8	182	49.28	179	53.44	175	57.83	172	60.28	168	55.11
9	136	53.39	109	59.22	99	63.50	104	57.56	128	56.72
10	474	40.40	387	63.40	466	60.90	368	74.85	367	72.15
11	202	52.61	195	53.83	164	60.67	163	73.06	163	61.89
12	103	55.11	87	62.83	88	66.06	95	61.94	113	55.06

Outputs related to the behavior of **A*_SPR** and **LS_SPR**. We test the ability **A*_SPR** and **LS_SPR**. to catch optimal solution, and observe the characteristics of resulting path. We rely on $\lambda = \lambda^{RD}$, $K = 10$ and $\rho = 4$. For every instance, we compute:

- in Table 5: The time value T_A*, the risk value R_A*, CPU time (in s.) CPU^{A^*}, the number Node of visited nodes, the number $State^{A^*}$ of generated states, and the deviation Dev^{A^*} between the length of resulting path Γ and L^*, induced by **A*_SPR** with $\lambda^{mode} = \lambda^{RD}$, $K = 10$ and $\rho = 4$;
- in Table 6: The time value T_LS, CPU time (in s.) CPU^{LS}, the number Trial of trials, the number State of generated states, and the deviation Dev^{LS} between the length of resulting path Γ and L^*, which derives from applying **LS_SPR** with $\lambda^{mode} = \lambda^{RD}$, $K = 10$ and $\rho = 4$.

CPU times are in seconds.

Table 5. Behavior of **A*_SPR**, with $\lambda^{mode} = \lambda^{RD}$, $K = 10$ and $\rho = 4$

Instance	T_A^*	R_A^*	$Node$	cpu^{A^*}	$States^{A^*}$	ρ^{A^*}
1	94.1	5.64	14.00	0.41	17.00	4.26
2	38.9	35.95	5.00	0.20	14.60	0.00
3	41.4	84.21	5.00	0.04	1.80	0.00
4	99.5	6.39	12.00	0.62	14.42	0.00
5	42.9	30.14	9.00	0.63	11.89	0.00
6	36.9	82.30	5.00	0.06	1.40	0.00
7	310.4	20.58	97.00	7.80	18.98	10.94
8	132.3	107.55	44.00	5.53	16.09	5.08
9	98.3	228.03	17.00	0.42	1.88	0.00
10	349.2	24.88	96.00	14.16	18.65	1.30
11	132.1	106.97	45.00	12.81	14.24	0.41
12	85.5	208.41	17.00	0.81	1.06	0.00

Table 6. Behavior of **LS_SPR**, with $\lambda^{mode} = \lambda^{RD}$, $K = 10$ and $\rho = 4$

Instance	T_LS	R_LS	$Trial$	cpu^{LS}	$States^{LS}$	ρ^{LS}
1	102.5	5.60	2.00	0.35	31.33	5.09
2	39.5	34.85	1.00	0.31	47.50	0.00
3	41.4	84.07	1.00	0.35	49.17	0.00
4	99.6	4.72	1.00	0.30	35.50	0.00
5	41.8	29.04	2.00	0.86	47.00	0.00
6	34.6	84.14	1.00	0.61	49.83	0.00
7	365.0	6.28	4.00	7.48	46.32	0.00
8	160.9	107.72	10.00	22.47	58.31	18.50
9	99.1	227.33	1.00	3.39	63.50	0.00
10	458.8	9.81	5.00	20.11	60.61	2.48
11	164.3	88.26	1.00	5.43	60.67	0.00
12	88.3	205.68	1.00	5.93	66.06	0.00

Comments:
We see that **A*_SPR** and **LS_SPR** are very close to each other with $\lambda^{mode} = \lambda^{RD}$. Also, they often use the shortest path but not every time especially when working with small R_{max} values.

Outputs related to characteristics of the solutions (Learning preprocess). We apply the learning devices of Section 4 and test the behavior of **DP_Evaluate** with no more than 3 possible decisions and 6 possible states for every $i = 1, \ldots, n$. For every instance, we compute in Table 7, related risk value R_Fast_DP, the time value T_Fast_DP, and CPU times CPU_Fast_DP. CPU times are in seconds.

Table 7. Behavior of **Fast-DP_Evaluate**

Instance	R_Fast_DP	T_Fast_DP	T_DP	cpu_Fast_DP
1	5.05	113.20	113.20	0.01
2	34.43	46.10	46.10	0.02
3	81.64	43.70	43.70	0.02
4	2.65	106.00	106.00	0.02
5	26.11	51.80	51.80	0.04
6	84.70	41.20	41.20	0.03
7	7.09	414.40	427.60	0.09
8	71.59	177.50	179.00	0.11
9	192.39	140.70	135.60	0.16
10	9.02	468.40	476.40	0.22
11	62.21	195.70	195.70	0.37
12	161.48	116.60	107.20	0.35

Comments: We can see that **Fast-DP_Evaluate** is, by far, the best method not because of the solution returned by because of how fast it computes a nearly equal solution.

6 Conclusion

We dealt here with a shortest path problem with risk constraints, which we handled under the prospect of fast, reactive and interactive computational requirements. But in practice, a vehicle is scheduled in order to perform some kind of *pick up and delivery* trajectory while performing retrieval and storing tasks. It comes that a challenge becomes to adapt previously described models and algorithms to such a more general context. Also, there exist a demand from industrial players to use our models in order to estimate the best-fitted size of an AGV fleet, and the number of autonomous vehicles inside this fleet. We plan addressing those issues in the next months.

References

1. Amazon.com, inc. amazon prime air. Available :http://www.amazon.com/primeair (2013)

2. B. Berbeglia, J-F. Cordeau, J-F., I. Gribkovskaïa, G. Laporte: "Static pick up and delivery problems : a classification scheme and survey". TOP: An Official Journal of the Spanish Society of Statistics and Operations Research 15, p 1-31, (2007)
3. L. Chen, C. Englund: "Cooperative intersection management: a survey"; IEEE Transactions on Intelligent Transportation Systems 17-2, p 570-586, (2016)
4. A. Franceschetti, E. Demir, D. Honhon, T. Van Woensel, G. Laporte, and M. Stobbe. "A metaheuristic for the time dependent pollution-routing problem"; European Journal of Operational Research, 259 (3) :972 – 991, (2017)
5. S.O.Krumke, A.Quilliot, A.Wagler, J.T.Wegener: "Relocation in carsharing systems using flows in time-expanded networks"; LNCS 8504, 87-98 (Special Issue SEA 2014), (2014)
6. T. Le-Anh, M. B. De Koster:: "A review of design and control of automated guided vehicle systems" European Journal of Operational Research, 171, 1-23, (2006)
7. L.Lozano, A.L.Medaglia. "On an exact method for the constrained shortest path problem". Computer&Operations Research 40, p 378-384, (2013)
8. J. Nilsson: Artificial Intelligence; J.Wiley Ed, N.Y, (1975)
9. I. Park, G. U. Jang, S. Park, J. Lee: "Time dependent optimal routing in micro-scale emergency situations"; In 10 th Int. Conf. on Mobile Data Management, p 714-719, IEEE, (2009)
10. C. Philippe, L. Adouane, A.Tsourdos, H.S. Shin, B.Thuilot: "Probability collective algorithm applied to decentralized coordination of autonomous vehicles"; 2019 IEEE Intelligent Vehicles Symp., 1928–34. IEEE, Paris, (2019)
11. V. Pimenta, A. Quilliot, H. Toussaint, D. Vigo: "Models and algorithms for reliability oriented DARP with autonomous vehicles"; European Journ. of Operat. Res., 257, 2, p 601-613, (2016)
12. C. Ryan, F. Murphy, F., Mullins: "Spatial risk modelling of behavioural hotspots: Risk aware paths planning for autonomous vehicles"; Transportation Research A 134, p 152-163 (2020)
13. K.C. Vivaldini, G. Tamashiro, J. Martins Junior, M. Becker: "Communication infrastructure in the centralized management system for intelligent warehouses". In: Neto, P., Moreira, A.P., et al. (eds.) WRSM 2013. CCIS, vol. 371, pp. 127–136. Springer, Heidelberg (2013)
14. I.F. Vis: (2006): "Survey of research in the design and control of AGV systems"; European Journal Operations Research 170:677–709
15. M. Zhang, R. Batta, R. Nagi R (2008): "Modeling of workflow congestion and optimization of flow routing in a manufacturing/warehouse facility". Management Sciences 55:267–280, (2008)
16. M. Koes and al.: "Heterogeneous multi-robot coordination with spatial and temporal constraints"; In Int. Conf. on Artificial Intelligence, p 1292-1297, (2005)
17. Q. Ren and al. "Cooperation of multi-robots for disaster rescue"; In ISOCC Conf. Proceedings; p 133134, (2017)