Proceedings of the 16th Conference on Computer Science and Intelligence Systems pp. 153–162
DOI: 10.15439/2021F18
ISSN 2300-5963 ACSIS, Vol. 25

Algorithms for the Safe Management of Autonomous Vehicles

Mourad Baïou
LIMOS Lab. CNRS/UCA,
Clermont-Ferrand, France

Aurelien Mombelli, Alain Quilliot
Labex IMOBS3, LIMOS Lab.
CNRS/UCA, Clermont-Ferrand, France

Lounis Adouane
HEUDYASIC Lab.CNRS and
UTC, Compiègne, France

Zhengze Zhu
LIMOS and InstitutPascal Labs
UCA/CNRS, Clermont-Ferrand, France

***Abstract*—We deal here with a fleet of autonomous vehicles which is required to perform internal logistics tasks inside some protected area. 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. We focus here on the top level, while introducing a time dependent estimation of the risk induced by the traversal of any arc at a given time. We set a model, state some structural results, and design, in order to route and schedule the vehicles according to a well-fitted compromise between speed and risk, a bi-level algorithm and a A* algorithm which both relies on a reinforcement learning scheme.**

I. Introduction

INTELLIGENT vehicles, provided with an ability to move with some level of autonomy, are the new hot spot in Mobility [1]. 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.

Fig. 1 An Autonomous Vehicle

Instead they foresee that the use of those vehicles is likely to be restricted to protected areas and professional purposes: moving free access vehicles inside large parking areas, performing rural or urban logistics or replacing too constrained AGV (*Autonomous Guided Vehicles*) inside warehouses or industrial structures (see [1]).

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

When it comes to the management of such systems, current trend is to the implementation of a hierarchical supervision architecture, relying on 2 or 3 levels:

- The first level, or *embedded* level, is defined by the monitoring and sensing devices which are embedded inside the vehicles, compute the trajectories in real time and adapt them to the possible presence of obstacles: currently, most effort from the robotics community remains devoted to this embedded level (see [9, 12, 13, 17]), which mostly involves optimal control and artificial perception techniques;
- The second one, or *midle* one, is in charge of the supervision of small *tricky* areas, like for instance crossroads or loading/unloading spots (see Figure 2).

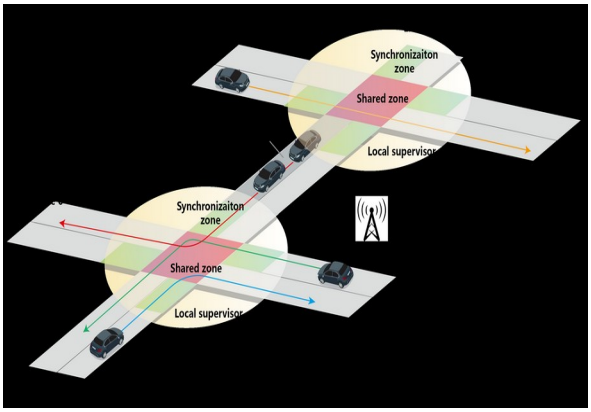

Fig. 2 A Hierarchical Supervision Architecture

Working as a mediator agent, it sends signals and instructions to the vehicles in order to regulate their transit and to avoid them to collide when they get through those areas. This level have been motivating a

This work was supported by ANR, Labex IMOBS3

rise in interest for the last years (see [5, 11, 15]), and sometimes a confusion with the *embedded* level: in many cases, hypothesis is set that all vehicles involved are run by the same embedded software and exchange perfect information; this become equivalent to supposing the existence of a local external mediator.

- The third one, or *global* one, consists in tactical dynamic planning and routing of the fleet, and the distribution of *Pick up and Delivery* (PDP) tasks among the vehicles (see [4, 7, 12, 21]).

Depending on the complexity and the size of the system, the second level may merge with either the first one or the last one. In any case, a true challenge is about the synchronization of those monitoring levels, which correspond to distinct time scales and purposes, and the design of communication protocols which will allow them to interact.

Our goal here is to deal with the *Global Monitoring* level. By some aspects, related problems may be viewed as cases of well-known *Pick up and Delivery* problem (see [4]), since in most cases a task will consist for a vehicle in moving to some place, performing some loading or unloading transaction and keep on. But two specific features are going to bring its specificity to this PDP variant:

- The time horizon of autonomous or semi-autonomous vehicles is usually somewhat short: decisions have to be taken fast, in a dynamic context, and decisional processes must take into account the communication infrastructure [20] 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 or semi-autonomous vehicles are involved, safety is at stake (see [2, 16, 17, 18]). The global supervisor must compute and schedule routes in such a way that not only tasks are going to be performed fast (standard industrial efficiency) but also that local and embedded supervisors will perform their job more easily. In other words, risk minimization should be a criterion for a good schedule.

A consequence is that performing the top level supervision of a fleet of autonomous vehicles requires disposing at any time of an accurate representation of the current state of the system and its short term evolution. This representation should enable us able to quantify the risk induced by an additional vehicle, which enters into the transit network and is asked to follow a given trajectory. We are not going to directly address this issue, which is complex (see [18, 23]). Instead, we are going to suppose that, at the time when we are trying to schedule this vehicle, we are provided with a procedure which, to any arc (x, y) of the transit network and any time value t, computes an estimation of the risk resulting from the presence of our vehicle on arc e at time t. Then our goal becomes to compute and schedule the route Γ of our vehicle, in such a way that its riding time is minimized and that induced risk estimation does not exceed some threshold *Risk_Max*. For the sake of simplicity, we shall limit ourselves to a one task tour, which means that Γ will be constrained by its starting point o and its destination point d. Described this way, our problem might be view as the search for a *constrained shortest path* [7]. But the fact that both risk and arc traversal times are time dependent makes the problem significantly more difficult (see [7]). By the same way, the *on line* feature of a system such that an autonomous vehicle fleet keeps us from relying on heavy mathematic machinery like MILP models or time expanded networks [9], and impose us to design ad hoc fast heuristics.

According to this purpose, we propose here 2 algorithms: The first one is a bi-level heuristic one, whose structure may be compared to the structure of *Split* algorithms which are used in *Route First Cluster Second* algorithms for *Vehicle Routing* problems. This algorithm iteratively acts on the topology of the route Γ and next schedule the vehicle along this route according to a filtered dynamic programming procedure. The second one is a A* tree search algorithm [14], which explores a *risk expanded network*.

Both algorithms are designed in order to induce small computing costs and both perform a kind of auto-adaptative reinforcement learning scheme [3, 10, 13, 22], which aim at estimating a good conversion ratio *time versus risk*.

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 bilevel local search heuristic [19], while in Section IV we describe the A* like algorithm. Section V is devoted to numerical experiments.

II. The Model

A. Transit Network and Risk Function

Transit Network and Risk Function. We suppose that our fleet of vehicles moves inside a simple planar transit network $G = (N, A)$, N denoting the nodes of G, and A its arcs, likely to represent for instance a warehouse (see figure 3 below).

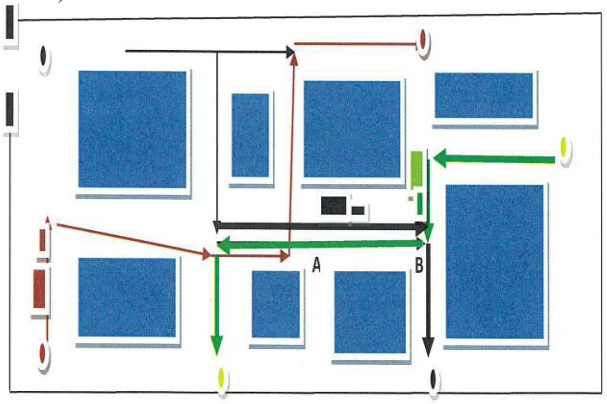

Fig. 3. A Warehouse like Transit Network.

Every arc $e = (x, y)$ is provided with a maximal speed V_Max_e and a length D_e. We denote by *TIME* the shortest path distance induced by shortest traversal time values D_e/V_Max_e. At the time $t = 0$ when the global supervisor of the fleet needs to take a decision about target vehicle *VEH*, he knows about routes followed by the other vehicles and the tasks they are going to perform. So he is provided, for any arc $e = (x, y)$ and any future instant $t > 0$, with an estimation of the number of vehicles and obstacles which are going to be located in e at time t. This allows him to derive a risk estimation $\Pi^e(t)$ whose meaning is:

- For any small value dt, $\Pi^e(t).dt$ is the *Expected Damage* between time t and time $t+dt$ in case *VEH* moves at maximal speed V_Max_e along e during this period.

Since we practically derive [18] any function Π^e from a finite (small) set of possible activity configurations related to arc e, we suppose that this function, which translates those configurations into risk, is piecewise linear (see figure 4). We call *break points* of $\Pi^e(t)$ the values t when the value of $\Pi^e(t)$ changes.

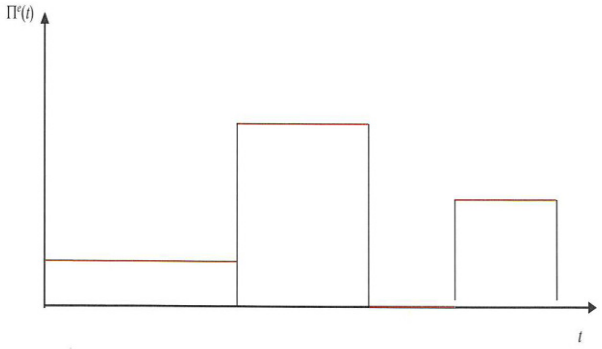

Fig. 4. A Piecewise Function Π.

Then we assume that, if *VEH* traverses arc e during some interval $[t, t+ dt]$ at speed $v \le V\_Max_e$, then related *Expected Damage* is given by a formula:

- $Risk^e(v, t) = \Phi(v/V\_Max_e).\Pi^e(t).dt$, where Φ is a convex increasing function with values in [0, 1] and such that for any value u, $\Phi(u)$ is significantly smaller than u. Those condition are imposed in order to confirm the intuition which tells that that the slower the vehicle moves, the smaller is resulting risk. In our experiments, we shall use function $\Phi(u) = u^2$.

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

$\int_{[T,\ T+\ \delta]} \Phi(v(t)/V\_Max_e).\Pi^e(t)\ dt$.

B. Routing Strategies and SPRC Model

Let us suppose now that origin o and destination d are given, as nodes of the transit network $G = (N, A)$. A *routing strategy* for our vehicle, is going to be a pair (Γ, v), where Γ is a path in the network G, and v is a *speed function*, which, to any time value $t \ge 0$, makes corresponds the speed $v(t)$ of the vehicle. Clearly, if at time t, *VEH* in located on arc $e \in \Gamma$, then $v(t)$ must not exceed V_Max_e.

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 $T(i)$ the time when *VEH* arrives to the end-node of e_i, then values $T(i)$ are completely determined speed function $t \to v(t)$. Then we set:

- $G\_Time(\Gamma, v) = T(n) =$ *global duration* of the routing strategy (Γ, v);
- $G\_Risk(\Gamma, v) = \Sigma_i \int_{[T(i-1),\ T(i)]} \Phi(v(t)/ V\_Max_e).\Pi^e(t)\ dt$ = *global risk* of the routing strategy (Γ, v).

The SPRC: ***Shortest Path Under Risk Constraint* Model**: Then our purpose becomes in a natural way to make vehicle *VEH* move from o to d while achieving small $G\_Time(\Gamma, v)$ and $G\_Risk(\Gamma, v)$ values. This looks a kind of bi-objective formulation. As a matter of fact, risk and time play very different roles inside a real industrial system, and so the risk is usually managed as a constraint: some threshold *Risk_Max* is given and the trajectory (Γ, v) of vehicle *VEH* is required to be such that resulting risk $G\_Risk(\Gamma, v)$ does not exceed threshold *Risk_Max*. It comes that our SPRC: *Shortest Path Under Risk Constraint* model comes in a natural way as follows:

> **SPRC: *Shortest Path Under Risk Constraint***: {Given the threshold *Risk_Max*, compute a routing strategy (Γ, v) such that $G\_Risk(\Gamma, v) \le Risk\_Max$ and $G\_Time(\Gamma, v)$ is the smallest possible}

C. Structural Results

The time dependence of the transit network together with the proximity of the SPRC model with *Shortest Path Constraint* models suggests that SPRC is a complex problem. As a matter of fact, we may state:

Proposition 1: *SPRC is NP-Hard. Even if Γ is fixed, computing speed function t -> v(t) is also NP-Hard.*

Sketch of the proof: SPRC can be reduced to the *Constrained Shortest Path* problem [6]. If we fix Γ, then we can reduce resulting problem to Knapsack. □

Still, as we shall see now, SPRC may be simplified. Let us suppose that we are provided with an optimal routing strategy (Γ, v). One easily checks that:

Proposition 2: *If VEH is running along some arc e during time T and time T + δ, and if $\Pi^e(t)$ is constant between T and T + δ, then we may do in such a way that optimal speed v(t) is constant on]T, T + δ[.*

It comes that we may impose function v to be piecewise constant, with *break points* which follow the arc e of Γ and the *break points* of functions $t \to \Pi^e(t)$.

Also, we may notice that in general, (Γ, *v*) will achieve exactly the risk threshold *Risk_Max*:

Proposition 3: *If it happens, at some time t, that VEH is running inside an arc e in such a way that* $v(t) \neq V\_Max_e$, *then* $G\_Risk(\Gamma, v) = Risk\_Max$.

Sketch of Proof: We suppose the converse, and check that it is possible to make *G_Risk(Γ, v)* decrease by augmenting the speed, and so the resulting risk, on some arc *e* of Γ. □

As it is the case in multi-objective optimization, a natural question arises now about a possible conversion of risk into time, which could allow us to deal with a mono-objective problem. When talking about risk into time conversion, we mean a coefficient α which would tell us that adding *dr* to *G_Risk*(Γ, *v*) would be equivalent to adding α.*dt* to *G_Risk*(Γ, *v*). If it were existing, coefficient α would be a risk per time coefficient, that means a *risk speed*. Such a conversion is not possible in the general case (else our problem would be almost time-polynomial). Still, it is possible in a *local* way, that means inside any given arc $e \in \Gamma$ such that $v(t) \neq V\_Max_e$, and also in the *stationary* case when every function $t \rightarrow \Pi^e(t)$ is constant. More precisely, if, at some time *t*, we are located inside an arc *e*, then we define what we call the *Risk Speed* $rt^e(t)$ (*rt* as risk per time) of our routing strategy (Γ, *v*):

- $rt^e(t) = \Phi(v(t)/V\_Max_e).\Pi^e(t)$.

Then we may state:

Proposition 4: *If, at some time t, VEH is running inside an arc e at speed* $v(t) \neq V\_Max_e$, *and if t is not a break point for piecewise function* Π^e *then the quantity* $rt^e(t)$ *is independent on t (but not on e). Besides, in the specific case when functions* Π^e *are constant for any arc e in Γ, then* $rt^e(t)$ *is independent on t and e, as soon as constant speed* $v_e = v(t)$ *is different from* V_Max_e.

Sketch of the Proof: It is a matter of applying Kuhn-Tucker local optimality conditions for constrained optimization, to the gradient vectors of quantities *G_Risk*(Γ, *v*) and *G_Time*(Γ, *v*). □

Remark 1: Above value $rt^e(t)$, computed for *t* such that is *VEH* is located on *e* at time *t* with $v(t) \neq V\_Max_e$, is independent on *t* but dependent on *e*, as we may see through the following example (Figure 5):

- Path Γ contains 2 arcs, e_1 and e_2, both with length 1 and maximal speed 2. Function Π^{e2} is constant and equal to 1. Function Π^{e1} takes value 2 for $0 \leq t \leq 1$, and a very large value *M* (for instance 100) for $t > 1$ (see figure 5). *Risk_Max* = 3/4; Function Φ is: $u \rightarrow \Phi(u) = u^2$.

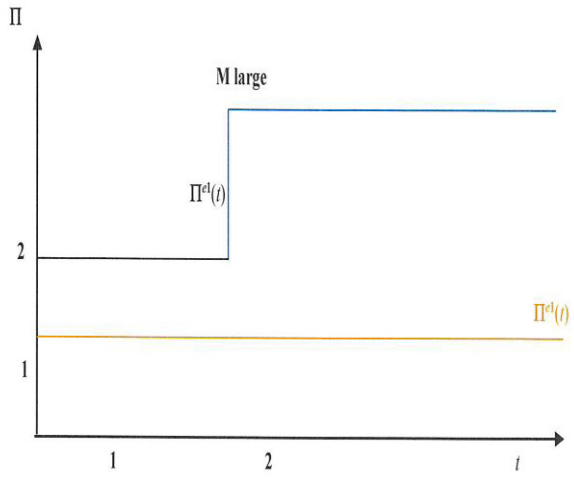

Fig. 5. Functions Π^{e1} and Π^{e2}.

- Then we see that *VEH* 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 speed* is equal to ½. Next it puts the brake, in the sense that its speed remains equal to 1 but its *risk speed* decreases to ¼. It is easy to check that this routing strategy is the best one, with *G_Risk*(Γ, *v*) = ¾ and *G_Time*(Γ, *v*) = 2.

D. Risk Driven Reformulation of the SPRC Model

Above results allow us to significantly simplify our SPRC model by replacing the search for speed $t \rightarrow v(t)$, likely to be volatile, by the search for *risk speed* $e \rightarrow rt^*_e$, where is the *risk speed* value which corresponds to arc *e* in Γ according to the first part of Proposition 4. It comes that we define a *risk driven routing strategy* as a pair (Γ, *rt**) 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*;
- *rt** associates, with any arc *e in* Γ, related *risk speed* value rt^*_e which is the unique value $\Phi(v(t)/V\_Max_e).\Pi^e(t)$ for any *t* such that *VEH* is located inside arc e_i and $v(t) \neq V\_Max_e$. Notice that if $v(t) = V\_Max_e$ then $rt(t) = \Pi^e(t)$.

Reconstructing a routing strategy (Γ, *v*) from a *risk driven routing strategy* (Γ, *rt).** Let us suppose that we know value rt^*_e related to arc *e* of Γ. Then, at any time *t* when *VEH* is inside arc *e*, and which is not a *break point* for function Π^e, we have: (E1)

- $v(t) = V\_Max_e$ and $dR(t)/dt = rt(t) = \Pi^e(t)$, where *R*(*t*) denotes the cumulative risk between 0 and *t*.
  or
- $v(t) = V\_Max_e$ and $rt^*_e = \Phi(v(t)/V\_Max_e).\Pi^e(t)$ and $dR(t)/dt = rt(t) = rt^*_e$.

Speed *v*(*t*) is obtained by solving the equation $rt^*_e = \Phi(v(t)/V\_Max_e).\Pi^e(t)$ and next comparing it with *V_Max*. Since both *v*(*t*) and *rt*(*t*) are piecewise constant on *e*, we see that we may scan the arc sequence $\{e_1,\ldots, e_n\}$and get,

through a simple iterative process, both time $T(i)$ when *VEH* arrives at the end-node of e_i and related cumulative risk $R(T(i))$. That means that the knowledge of (Γ, rt^*) allows us to reconstruct standard routing strategy (Γ, v^*).

According to this and proposition 4, SPRC may be rewritten as follows (we extend previous notations $G\_Time(\Gamma, v^*)$ and $G\_Risk(\Gamma, v^*)$ by denoting by $G\_Time(\Gamma, rt^*)$ and $G\_Risk(\Gamma, rt^*)$ respectively the time value and risk value of a *risk driven routing strategy* (Γ, rt^*):

SPRC *Risk Driven* Reformulation: {Compute *risk driven routing strategy* (Γ, rt^*) such that $G\_Risk(\Gamma, rt^*) \leq Risk\_Max$ and $G\_Time(\Gamma, rt^*)$ is the smallest possible}.

III. A FIRST BILEVEL ALGORITHM

We discuss here a bi-level heuristic algorithm [19] whose main iterative loop works in 2 steps:

***BL_RCSP* Algorithm**.
Intialize some path Γ from o to d; Not *Stop*;
While Not *Stop* do
1st step: *Schedule* Γ; (*Low level step*)
2nd step: *Improve* Γ; (*Top level step*)
If Fail(*Improve*) then *Stop*;
Keep the best solution Γ ever obtained.

The *Schedule* step considers path Γ as being fixed, and deals with the problem of computing values rt^*_e, e in Γ. Let us recall (Proposition 1) that this problem is NP-Hard. As a matter of fact, this *Schedule* step will contain the most important features of the ***BL_RCSP*** algorithm, namely those related to reinforcement learning. We shall describe it in section III.2. Meanwhile, we are going to briefly describe the *Improve* step, designed in order to modify Γ and improve its quality, and which works in a more classical way.

A. Top Level Improve Step

We suppose that some proximity threshold *S_Prox* has been fixed, and that for any two nodes x, y of the transit network G such that $TIME(x, y) \leq S\_Prox$, we are provided with a collection $Path(x, y)$ of elementary path Ω^j, $j \in J(x, y)$ from x to y. Construction of collections $Path(x, y)$, $x, y \in X$ such that $TIME(x, y) \leq S\_Prox$, may have been previously achieved though some preprocess. This allows us to introduce the following local transformation operator $Detour(\Gamma, x, y, j)$, which acts on any path Γ through parameters x, y and j: x and y are 2 nodes of Γ, such that $TIME(x, y) \leq S\_Prox$ and x is located before y on Γ; j belongs to $J(x, y)$.

- Then *Detour* replace the restriction $\Gamma_{x,y}$ of Γ from x to y by path $\Omega^j \in Path(x, y)$.

Performing the pre-process which perform the constructions of path collections $Path(x, y)$, $x, y \in X$ such that $TIME(x, y) \leq S\_Prox$, allows us not to take care about path search when trying to modify Γ, and so speed the *Improve* step in a significant way.

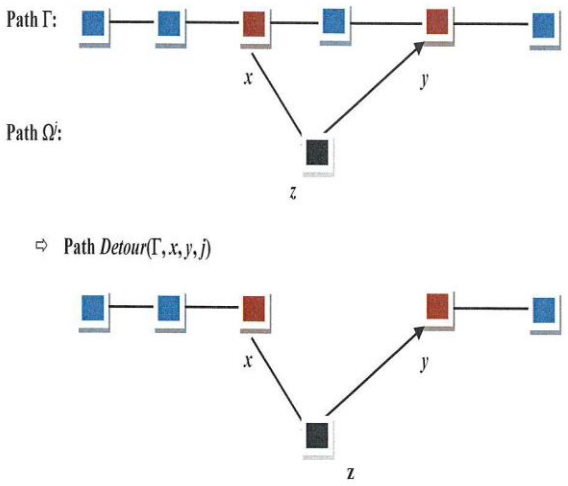

Fig. 6. *Detour* Operator

Improve step is going to drive operator *Detour* according to some standard *descent* process:

***Improve* Step**:
Not *Stop1*;
While Not *Stop1* do
Generate 3-uple (x, y, j),
Schedule Detour(Γ, x, y, j);
If *Detour*(Γ, x, y, j) is better than Γ then
Stop1;
Replace Γ by *Detour*(Γ, x, y, j)
Else Update *Stop1*;

What remains to be told is the way we generate parameters (x, y, j) for the operator *Detour*. Once Γ has been scheduled, we are provided, for any node x in the support $X(\Gamma)$ of Γ, with:

- The time T_x, when *VEH* arrives in x, together with related cumulated risk value R_x;

Then, above *Generate* instruction focuses on 3-uples (x, y, j) such that:

- Ratio $(R_y - R_x)/(T_y - T_x)$ is large with respect to $G\_Risk(\Gamma, rt^*)/G\_Time(\Gamma, rt^*)$, which suggests that sub-path $\Gamma_{x,y}$ is somewhat *crowded*;
- Path Ω^j is not very *crowed* between time T_x and time T_y, or in other words, the sum, for the arcs e of Ω^j, of mean $\Pi^e(t)$ value between time T_x and time T_y is significantly smaller than the same quantity for sub-path $\Gamma_{x,y}$.

B. Low Level Schedule Step

As told before, it means the key point inside our algorithm. Basically, it consists in a dynamic programming procedure ***DP_Schedule*** whose main features come as follows:

- We denote by $e_1,\ldots, e_n$ the arcs of current path Γ, and by $x_0, \ldots, x_n$ related nodes of the node support $X(\Gamma)$ of Γ.
- So the *time space* of ***DP_Schedule*** comes in a natural way as the set $\{0, 1,\ldots, n\}$ and a *state* (or *label*) at i is a pair (T, R), where T means the time when *VEH* arrives in x_i, and R the cumulated risk at this time. For any i, we shall denote as $State(i)$ the set of *states* computed in relation with i. Those states will be used in order to move along arc e_i. Clearly, initial state is (0, 0) and final state ($G\_Time(\Gamma, v^*)$, $G\_Risk(\Gamma, v^*)$) is going to be the pair (T, R) in $State(n)$ with smallest T value and such that $R \leq Risk\_Max$.
- Then a *decision* at i comes as a value rt^*_e with arc $e = e_{i+1}$. We denote by $DEC(i)$ he set of decisions which are tried at i. Resulting transition derives from equation (E1), which allows us to compute time value $T1$ and risk value $R1$ when we arrive in x_{i+1}. Clearly, decision rt^*_e will be feasible only if $R1 \leq Risk\_Max$.
- According to this, applying Bellman principle means eliminating from $State(i+1)$, states $(T1, R1)$ which are not Pareto optimal, that means which are such that there exists $(T2, R2)$ in $State(i+1)$, such that $T2 \leq T1$ and $R2 \leq R1$, one at least of those inequalities being strict.

The Filtering Issue: **A Learning through Reinforcement Device** ([3, 10, 13, 22]). As told in the introduction, SPRC puts computing costs are at stake. Above low level **DP*_Schedule*** procedure should run very fast, and so $State(i)$, as well as the set $DEC(i)$ of tried decisions rt^*_e should remain (very) small, and this in spite of the fact that the number of potential vales rt^*_e may be high, and even infinite if we suppose that we are dealing with rational numbers. In order to handle this issue, we use the second part of proposition 4 and the fact that, in perfect cases, rt^*_e might be considered as a **risk per time price**, taking a same *theoretical* value *rt_perfect* all along path Γ. This suggests us to do as if such a theoretical value *rt_perfect* were existing and try to *learn* it through the *Reinforcement Principle* (see [2, 17]), that means while moving along path Γ and performing (possibly several times) our ***BL_RCSP*** algorithm. More precisely:

- We fix the number M of possible decisions rt^*_e, and impose a threshold *State_Max* on the size of any state subset $State(i)$. Those 2 values M and *State_Max* become parameters of the ***BL_RCSP*** algorithm.
- According to this, we manage, all along the process, two quantities *rt_min* and *rt_max*, respectively *pessimistic* and *optimistic* estimations of ideal value *rt_perfect*, and which are going to be the target of the learning process. We do in such a way that, for any value i during the DP process, decisions rt^*_e which are going to be tried can be written $rt^*_e = rt\_min + m.rt\_max/(M-1)$, $m = 0,\ldots, M-1$.
- Then we drive *rt_min* and *rt_max* values in an auto-adaptative way (*learning through reinforcement*): applying decisions rt^*_e from state subset $State(i)$ and filtering them through Bellman principle provides us with a state subset $State(i+1)$ whose size is likely to exceed *State_Max*. Since our interpretation of *rt_min* and *rt_max* values is that *r-midst* = (*rt_min* + *rt_max*)/2 might be considered as the kind of theoretical *risk versus time* price *rt_perfect* we have just been talking above, we rank states (T, R) of $State(i+1)$ according to $rt\_midst.T + R$ values. Ideally, states (T, R) ordered this way should make best states (T, R) be balanced in the sense that the ratio $R/Risk\_Max$ should be centered around the ratio $TIME(o, x_{i+1})/TIME(o, d)$ and that the *entropy* of those best states should not be too large. If, for instance, those values are centered significantly above this ratio, then we deduce that we are moving in a too risky way and must make *rt_min* and *rt_max* decrease. Conversely, if those best values are centered above this ratio, then we are too *careful*. More precisely, we perform a kind of statistical analysis of those best values in $State(i+1)$, and derive, from those *best states* (T, R), several indicators:
  - *Risk_Balance*: It takes values {*Risky*, *Normal*, *Careful*} depending on the way the mean $R/Risk\_Max$ *value* is located with respect to
    $$TIME(o, x_{i+1})/TIME(o, d).$$
  - *Entropy*: It takes values {*Large*, *Normal*, *Small*} depending on the scope of $R/Risk\_Max$ values.

Then ***Clean_Learn*** procedure below performs the *filtering&learning* process:

***Clean_Learn* Procedure**:
Rank states (T, R) of $State(i+1)$ according to $rt\_midst.T + R$ values;
Select best *State_Max* (T, R) according to this ranking and compute *Risk_Balance* and *Entropy*;
If *Risk_Balance* = *Normal* then
  Keep only the *State_Max* best states in $State(i+1)$;
  If *Entropy* = *Small* (*Large*) then *Enlarge* (*Shorten*) the interval [*rt_min*, *rt_max*] while keeping *rt_midst* unchanged;
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 ranking and $S_2 = S - S_1$;
  Clean $State(i+1)$ in order to keep the *State_Max*/2 best states in both S_1 and S_2;
  Make *rt_max* and *rt_min* decrease;
  If *Entropy* = *Small* (*Large*) then *Enlarge* (*Shorten*) the interval [*rt_min*, *rt_max*] while keeping *rt_midst* unchanged;

If *Risk_Balance* = *Careful* then proceed the same way as in previous case, while making *rt_min* and *rt_max* increase.

C. Greedy Algorithm GR_RCSP

We turn above ***BL_RCSP*** algorithm into a *greedy* one, by removing the top level ***Improve*** loop. That means that we choose Γ as the shortest path from *o* to *d* in the *TIME* sense, and apply ***DP_Schedule***.

IV. A A* LIKE ALGORITHM

Let us first recall that well-know A* algorithm [14] is an extension of Dijsktra algorithm for the search of a shortest path in a graph, which was introduced in order to deal with very large networks. Nodes of such a network are usually defined in an implicit way, as possible configurations for the state of a system (for instance a robot). It is typically our case here, since we are searching a path in a *risk expanded* network, whose nodes are all pairs (x, R), x beings a node of the transit network $G = (N, A)$ and R a risk value between 0 and *Risk_Max*.

As in III, we are still willing to design a fast and flexible algorithm. But our approach is different from Section III, in the sense that: Algorithm ***A*_RCSP*** is going to work while simultaneously looking for a path Γ and a schedule *rt** for this path. Roughly, at any time during ***A*_RCSP*** process, we are going to be provided with:

- A current value *T_Curr*, computed by ***GR_RCSP***.
- An *expansion* list *LS* of *state* 3-uples (x, T, R), where:
  - x is a node of the transit network.
  - T and R are respectively the time and risk value which were required in order to arrive in x.
  - *LS* is ordered according to values
    $$V = T + TIME(x, d). \quad \text{(E2)}$$
    The first element in *LS* is called current *Pivot* state.
    Explanation of (E2): $TIME(x, d)$ is a lower bound for the time that the vehicle *VEH* has to spend running before achieving its journey. So V is a lower bound for value $G\_Time(\Gamma, rt^*)$ in case routing strategy (Γ, rt^*) extends current position (x, T, R).
- A list *L_PIVOT*, which contains all 3-uple (x, T, R) which have formerly been used a *Pivot* state.

We do in such a way that:

- All elements in $LS \cup L\_PIVOT$ are Pareto optimal in the sense that, for a given x, there does not exist $T, R, T1, R1$ such that both (x, T, R) and $(x, T1, R1)$ are in $LS \cup L\_PIVOT$, with $T \leq T1$ and $R \leq R1$. (E3)

Then Algorithm ***A*_RCSP*** removes $Pivot = (x_0, T_0, R_0)$ from *LS*, puts it into *L_Pivot*, and perform the *expansion* step, that is:

- For any arc $e = (x_0, x)$, with origin in x_0, it generates a set DEC^e_{Pivot} of decisions rt_e^* (*risk speeds*) in the same sense as in ***BL_RCSP*** algorithm; (*DECIDE*)
- For any arc $e = (x_0, x)$ and any decision rt_e^* (*risk speed*) it generates resulting state (x, T, R); Then ***A*_RCSP*** inserts state (x, T, R) into *LS* and manages in such a way that (E2) be satisfied;
- For any x such that (x_0, x) is an arc of the transit network, it filters states (x, T, R) which are currently in *LS*, in order to meet requirement (E3). (*FILTER*)

The Filtering Issue: ***Learning through Reinforcement.*** Once again, since we want our algorithm to run fast, we impose a threshold M on the number of possible decisions rt_e^*. Besides, we impose a parameter *State_Max*, with the meaning: (E4)

- For any x, the number of states (x, T, R) which are contained into $LS \cup L\_PIVOT$ never exceeds parameter *Max_State*.

In order to go further with this filtering issue, we must now explain the way instructions *FILTER* and *DECIDE* work, since we see that, as for the ***BL_RCSP*** algorithm, the key point in this ***A*_RCSP*** algorithm lies on the way we perform those instructions. Since we are provided with a current solution *T_Curr*, we may of course apply standard Branch/Bound filtering technique, and kill candidate state (x, T, R) if related value $V = T + TIME(x, d) \geq T\_Curr$. But it is clearly not enough in order to ensure that (E4) is satisfied. So we proceed the same way as in the case of ***BL_RCSP***:

- At any time during the process, we are provided with two quantities *rt_min* and *rt_max*;
- Then we use *rt_min* and *rt_max* in order to generate (Instruction *DECIDE*) decisions *rt** exactly as in ***BL_RCSP***:
  - We rank candidate states (x, T, R), resulting from all decisions rt^*_e, $e = (x_0, x)$ as in III.2, while using $rt\_midst = (rt\_max + rt\_min)/2$.
  - We compute the *Risk_Balance* and *Entropy* quantities.
  - Then we update values *rt_min* and *rt_max* as in III.2, and apply the same technique in order to ensure that, for any x, the number of states (x, T, R) in $LS \cup L\_PIVOT$ does not exceed *State_Max*.

V. NUMERICAL EXPERIMENTS

Goal: We have been performing numerical experiments with the purpose of getting information about the following points:

- The ability of the different algorithms to get good solutions under small computational costs, and the dependence of their behavior to the size of the transit network;

- The sensitivity of those algorithms to the parameter *State_Max* and *M*, which bounds, for every algorithm, the numbers of possible states and decisions;
- The sensitivity of our algorithms to the structure of the piecewise constant functions Π^e, and on the intensity of current traffic inside the transit network at the time when the algorithms are applied.

In order to do it, we used the *A** like algorithm, run with large *State_Max* and *M* values as an almost exact algorithm, which provided us with reference results.

Technical Context: Algorithms were implemented in C++, on a computer running Windows 10 Operating system with an IntelCore i5-6500@3.20 GHz CPU, 16 Go RAM and Visual Studio 2017 compiler.

Instances: We generated networks (N, A) as connected symmetric *partial grids*, which means grids *n*m*, modified through removal of a percentage ρ of nodes and arcs and the introduction of one-way arcs (we break the symmetry of the grid). Those partial grids are summarized through their number $|N|$ of nodes and their number $|A|$ of arcs. The time value $TIME_{x,y}$ of any arc (x, y) is 1, as well as related *v_max* value. Function Φ is taken as function $u \rightarrow \Phi(u) = u^2$. Function Π^e are generated by fixing a time horizon *T_Max*, fixing a mean number B of *break points* t_i^e in [0, *T_Max*] per time unit, and an average value Δ for value $\Phi(u)$. Then, for any e, we randomly generate *break points* t_i^e and values $\Pi^e(t_i^e)$, while imposing those values to belong a finite 5 values set {2Δ, 3Δ/2, Δ, Δ/2, 0}. Finally, we fix the threshold *Risk_Max* value. $TIME_{o,d}$ is also a parameter.

We present here results for 10 instances, whose characteristics come as follows:

TABLE I.
CHARACTERISTICS OF THE INSTANCES

Instance	$\|N\|$	$\|A\|$	*B*	*Δ*	*Risk_Max*	$TIME_{o,d}$
1	22	65	1	0.2	1	6
2	18	61	2	0.6	1	7
3	19	65	3	1	1	5
4	54	159	1	0.2	2	9
5	58	182	2	0.6	2	9
6	51	175	3	1	2	8
7	88	285	1	0.2	3	12
8	92	268	2	0.6	3	11
9	83	250	2	0.6	3	10
10	86	262	3	1	3	11

Outputs: For every instance we compute:

- The Risk value *R_BL*, the Time value *T_BL* computed by the bi-level ***BL_RCSP*** algorithm, the number of iterations *ITER* of its main loop (modification of *I*) and related CPU time *Time_BL*.
- The Risk value *R_A**, the Time value *T_A** computed by the ***A*_RCSP*** algorithm, the number *Node* of visited nodes and related CPU time *Time_BL*.
- The Risk value *R_GR*, the Time value *T_GR* computed by the greedy algorithm ***GR_RCSP*,** together with related CPU time *CPU_GR*.
- Almost exact Risk value and Time value *R_Opt*, *T_Opt* computed by the ***A*_RCSP*** algorithm, performed with large *State_Max* and *M* values, together with related CPU time *CPU_Opt*.

Obtained results are summarized in the following tables: CPU times are in seconds

TABLE 2.
REFERENCE VALUES AND *GR_RCSP* BEHAVIOR (*STATE_MAX* = 10 AND *M* = 5)

Instance	*R_Opt*	*T_Opt*	*CPU_Opt*	*R_GR*	*T-GR*	*CPU_GR*
1	0.98	7.5	459.6	0.91	11.5	0.01
2	0.99	13.0	524.9	0.94	13.8	0.01
3	0.98	15.5	312.4	0.95	18.2	0.01
4	1.97	9.6	988.7	1.78	12.7	0.02
5	1.98	16.9	1044.2	1.92	24.9	0.02
6	1.98	18.4	857.5	1.88	24.1	0.02
7	2.98	11.0	2209.3	2.84	12.4	0.03
8	2.96	16.7	1858.0	2.81	22.7	0.03
9	2.99	18.9	1977.8	2.80	28.6	0.03
10	2.98	37.5	2033.5	2.68	54.2	0.03

TABLE 3.1.
BL_RCSP BEHAVIOR WITH *STATE_MAX* = 10 AND *M* = 5

Instance	*R_BL*	*T-BL*	*CPU_BL*	*ITER*
1	0.92	8.3	0.05	3
2	0.93	14.0	0.02	1
3	0.91	17.1	0.03	2
4	1.84	10.8	0.05	2
5	1.90	17.7	0.14	5
6	1.82	20.7	0.11	4
7	2.74	12.2	0.08	2
8	2.80	19.4	0.13	3
9	2.87	19.5	0.35	8
10	2.78	40.5	0.24	6

TABLE 3.2.
BL_RCSP BEHAVIOR WITH *STATE_MAX* = 50 AND *M* = 10

Instance	*R_BL*	*T-BL*	*CPU_BL*	*ITER*
1	0.96	8.0	0.64	3
2	0.95	13.7	0.79	3
3	0.95	16.3	0.93	4
4	1.88	10.4	0.59	2
5	1.91	17.5	1.3	4
6	1.92	19.4	0.11	5
7	2.87	11.9	2.1	4
8	2.90	18.6	1.5	4
9	2.89	19.4	2.9	6
10	2.85	39.5	3.6	7

TABLE 4.1
*A*_RCSP* BEHAVIOR WITH *STATE_MAX* = 10 AND *M* = 5

Instance	*R_A**	*T_A**	*CPU_A**	*Node*
1	0.95	7.9	0.15	11
2	0.92	13.8	0.13	10
3	0.97	15.9	0.14	10
4	1.80	10.9	0.35	20
5	1.88	17.7	0.39	18
6	1.86	20.0	0.45	19
7	2.75	11.9	0.95	25
8	2.66	18.0	1.0	28
9	2.80	19.5	1.1	30
10	2.84	38.6	1.2	28

TABLE 4.2
*A*_RCSP* BEHAVIOR WITH *STATE_MAX* = 50 AND *M* = 10

Instance	*R_A**	*T_A**	*CPU_A**	*Node*
1	0.98	7.7	1.2	9
2	0.96	13.6	1.3	10
3	0.98	15.7	1.5	10
4	1.87	10.5	3.2	19
5	1.90	17.5	4.2	17
6	1.93	19.2	4.0	17
7	2.85	11.6	9.8	25
8	2.77	17.6	10.5	28
9	2.88	19.2	10.3	29
10	2.87	38.4	9.9	26

Comments: Results obtained through ***GR_RCSP*** are rather erratic, because this algorithm relies on the current state of shortest path Γ from *o* to *d*, which can be bad at the time when we launch the algorithm. ***A*_RCSP*** tends performs better than ***BL_RCSP*** as for the accuracy, but is more time consuming. Depending on the cases, results may be significantly impacted by parameters values *State_Max* and *M*. Finally, we also notice that obtaining almost exact optimal values is rather time costly, even on small instances. In order to improve it, we should find a way to provide a criterion which could identify, at any times, whether a decision rt^*_e has to be tried or not. Notice also that *Risk_Opt* is almost never equal to *Risk_Max*, in spite of proposition 2, because of the bias due to the discretization of the rt^*_e.

VI. CONCLUSION

We have been dealing here with a shortest path problem with risk constraints, which we handled under the prospect of fast, reactive and interactive computational requirements. But, the true practical problem is supposed to be a *pick up and delivery* one, simultaneously involving several tasks and vehicles. It comes that a future challenge is to adapt the algorithms which we just described here to such a more general PDP context. Also, there exist a demand from industrial players to use those algorithms as a tool for strategic decision, in order to estimate convenient size of the AGV fleet, together with the number of autonomous vehicles inside this fleet. We plan addressing those issues in the next months.

ACKNOWLEDGMENT

Present work was funded by French ANR: National Agency for Research, and Labex IMOBS3, as well as by Region AURA: *Auvergne Rhône Alpes*.

REFERENCES

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

[2] C.Artigues, E.Hébrard, A.Quilliot, H.Toussaint: "Models and algorithms for natural disaster evacuation problems". *Proceedings of the 2019 FEDCSIS WCO Conference*, p 143-146, (2019). DOI: http://dx.doi.org/10.15439/978-83-952357-8-8

[3] B. Bakker, S. Whiteson, L. J. Kester, F. Groen: "Traffic light control by multi-agent reinforcement learning systems"; In *Interactive Collaborative Information Systems*, (2010). DOI: 10.1007/978-3-642-11688-9_18

[4] 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). DOI: 10.1007/s11750-007-0009-0

[5] L. Chen, C. Englund: "Cooperative intersection management: a survey"; *IEEE Transactions on Intelligent Transportation Systems* 17-2, p 570-586, (2016). DOI: 10.1109/TITS.2015.2471812

[6] D. Duque, L.Lozano, A.L.Medaglia. An exact method for the biobjective shortest path problemfor large-scale road network. EJOR 242, p 788-795, (2015). http://dx.doi.org/10.1016/j.ejor.2014.11.003

[7] S.Fidanova, O.Roeva, M.Ganzha: " Ant colony optimization algorithm for fuzzy transport modelling ". *Proceedings of the 2020 FEDCSIS WCO Conference*, p 237-240, (2020). DOI: http://dx.doi.org/10.15439/978-83-955416-7-4

[8] 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). DOI: 10.1016/j.ejor.2016.11.026

[9] S. Bsaybes, A.Quilliot, A.Wagler: "Fleet management for autonomous vehicles using multicommodity coupled flows in time-expanded networks"; *17th International Symposium on Experimental Algorithms (SEA 2018) (LIPIcs)* 103, (2018). DOI: 10.4230/LIPIcs.SEA.2018.25

[10] M.Krzyszton: "Adapative supervison: method of reinforcement learning fault elimination by application of supervised learning". *Proceedings of the 2018 FEDCSIS AI Conference*, p 139-149, (2018). DOI: http://dx.doi.org/10.15439/978-83-949419-5-6

[11] J. Kumar, V. V. Ranga: "Multi-robot coordination analysis, taxonomy, challenge and future scope"; *Journal of Intelligent and Robotic Systems*, 102:10, (2021). https://doi.org/10.1007/s10846-021-01378-2

[12] 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). https://doi.org/10.1016/j.ejor.2005.01.036

[13] Y.Li, E.Fadda, D.Manerba, R.Tadei, O.Terzo: " Reinforcement learning algorithms for online single machine scheduling". *Proceedings of the 2020 FEDCSIS WCO Conference*, p 277-283, (2020). DOI: http://dx.doi.org/10.15439/978-83-949419-5-6

[14] Nilsson, J.: *Artificial Intelligence*; SpringerY, (1982). ISBN 978-3-540-11340-9

[15] Philippe, C., Adouane, L., Tsourdos, A., Shin, H.S., Thuilot, B. : "Probability collective algorithm applied to decentralized coordination of autonomous vehicles"; *2019 IEEE Intelligent Vehicles Symp*., 1928–34. IEEE, Paris (2019). DOI:10.1109/IVS.2019.8813827

[16] 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). doi: 10.1016/j.ejor.2016.07.037.

[17] Y. Rizk, M. Awad, E. Tunstel: "Cooperative heterogenous mutlti-robot systems: a survey"; *ACM Computing Surveys* 29, (2019). https://doi.org/10.1145/3303848

[18] C. Ryan, F. Murphy, F., Mullins, M.: "Spatial risk modelling of behavioural hotspots: Risk aware paths planning for autonomous vehicles"; *Transportation Research A* 134, p 152-163 (2020).

[19] DOI: 10.1016/j.tra.2020.01.024

[20] K.Stoilova, T.Stoilov: "Bi-level optimization application for urban traffic management". *Proceedings of the 2020 FEDCSIS WCO Conference*, p 327-336, (2020). DOI: http://dx.doi.org/10.15439/978-83-949419-5-6

[21] 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, (2013)

[22] I. F. Vis: "Survey of research in the design and control of AGV systems". *European Journal of Operations Research 170:677–709*, (2016). DOI: 10.1016/j.ejor.2004.09.020

[23] J. Wojtuziak, T. Warden, O. Herzog: "Machine learning in agent based stochastic simulation: Inferential theory and evaluation in transportation logistics"; *Computer and Mathematics with Applications* 64, p 3658-3665, (2012).

[24] https://doi.org/10.1016/j.camwa.2012.01.079

[25] 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). DOI: 10.1287/mnsc.1080.0916