Information Sciences and Computer Engineering, Vol. 1, No. 2, (2010) 75–85

PHARUS

International Journal of
Information Sciences and Computer Engineering

journal homepage: http://www.ijisce.org

Constrained Navigation with Mandatory Waypoints in Uncertain Environment

François Lucas[a], Christophe Guettier[a], Patrick Siarry[b,*], Anne-Marie Milcent[a], Arnaud de La Fortelle[c]

[a]*Sagem Defense and Security, 27 rue Leblanc, 75012 Paris, France*
[b]*Paris XII University, LiSSi Laboratory, 61 avenue du Général de Gaulle, 94010 Créteil, France*
[c]*Mines ParisTech, CAOR Robotics Laboratory, Mathématiques et Systèmes, 60 boulevard Saint Michel, 75006 Paris, France*

Abstract– **This paper presents a hybrid solving method for vehicle path planning problems. As part of the vehicle system architecture (vetronic), planning is dynamic and has to be activated on-line, which requires response times to be compatible with mission execution. The proposed approach combines constraint solving techniques with an Ant Colony Optimization (ACO). The hybridization relies on a static probing technique which builds up a search strategy using a distance information between problem variables and a heuristic solution. Various forms of this approach are compared and evaluated on real world scenarios. Preliminary results exhibit response times close to vehicle control requirements, on realistic problem instances.**

Keyword: Path planning, ant colony optimization, probing, TSP, constraint programming, search.

1. Introduction

Path planning has been a major challenge for decades. It comes up in various fields such as *mobile robot mission planning* where the itinerary to a goal must be minimized, *video games* where character trajectory determination must fit with reactivity demands or *logistics* where complex resource management problems can directly affect company profits.

This paper focuses on the problem of constrained navigation with mandatory waypoints in uncertain environment. We are considering the case of a military vehicle on mission, which is given a final goal and a list of intermediate objectives to reach, whose sequence is not defined. Several constraints have to be taken into account: overall mission time, itinerary length, energy consumption, coordination with other vehicles, etc. Military mission planning has been so far considered separately from navigation issues and defined at mission preparation time. However, with modern on-board hardware and communication architecture (called *vetronic*), mission uncertainty can be managed on-line. In particular, it is possible to provide planning alternatives when contingencies occur.

The goal is to provide driver decision support by advising the best route to follow under specified mission constraints. System efficiency criteria are *solution optimality* and *reactivity*: it must bring the best solution in execution times close to that of human reflexes. This application can be extended to unmanned vehicles, where navigation plans must be updated whenever mission objectives change, the environment evolves significantly or the expected amount of resources is not sufficient. In this paper, we present a new hybrid approach mixing a complete constraint solving with a metaheuristic-based guiding method. Experiments made on representative problems (urban or open environment) show interesting performances.

The problem is described formally in section 2. Section 3 summarizes recent state of the art in the different research fields. The proposed approach is detailed in section 4 and tested in section 5, where results are analyzed.

2. Problem Formulation and Modeling

2.1. The problem

The problem consists in finding a path from the current vehicle location to an objective waypoint. Intermediate mandatory waypoints can be imposed to fulfill secondary objectives. According to the user experience, two kinds of plan optimization are interesting: minimizing overall duration and maximizing mission safety. However, various other operational metrics can be imposed as hard constraints. In the following, we will consider only the vehicle energy as an additional constraint.

This problem is halfway between several well-known instances of the literature:

- *Optimal path finding*, for which the problem is to find a fast, efficient and economic method to connect two points in a graph;
- *Travelling Salesman Problem* (TSP) which is the problem of finding the less expensive hamiltonian cycle over a series of nodes in a compact graph;
- *Vehicle Routing Problem* (VRP) where costs (distance, time) of multi-vehicle routing with capacity constraints must be optimized;

*Corresponding author:
Email address: siarry@univ-paris12.fr, Ph: +33 145171567

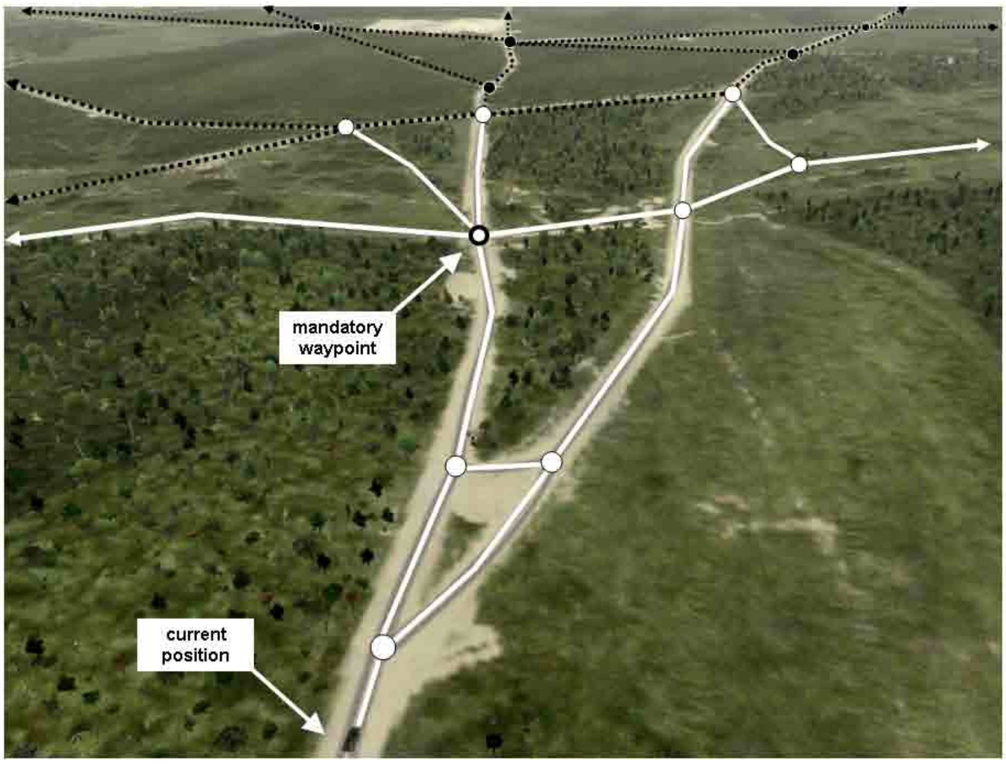


Fig. 1. Example of alternative paths for a ground vehicle horizon, and corresponding to the line of sight of a cooperating *Unmanned Aerial Vehicle* (UAV).

- *Constraint Satisfaction Problems* where the goal is to find correct assignments to problem variables satisfying a set of constraints.

When relaxing operational constraints, our vehicle navigation problem complexity is NP-hard in worst cases as it can be reduced in polynomial time to a TSP. The core difficulty is to find the optimal sequence of mandatory waypoints to visit with near real-time performances, despite a realistic problem size.

2.2. Example

As an illustrative example, let us consider the following situation inspired from a real case (figure 1). During mission execution, the planner has only a partial awareness of its environment. The knowledge horizon is determined by direct ground observation, but additional knowledge can also be provided by vehicle team, sensor networks or external surveillance systems. White dots and lines represent waypoints and feasible paths between waypoints respectively. Black dots and dashed lines represent waypoints and uncertain paths that are situated beyond the observability horizon. Arrows show lines that continue beyond the limits of the picture. Lastly, one or more waypoints may be imposed along the vehicle route.

2.3. Basic constraints

In our problem, we need to represent the ground topography of the vehicle area of interest. A graph model is used to represent the space of possible paths, where vertices are geographical points (for example crossroads) and edges are progression axes (for example a road or a meadow) that can be taken by vehicles. They are determined before the beginning of the mission using roadmaps, digital terrain models and UAV or satellite observations.

Formally, the graph G is defined as a couple $G = (V, E)$ where V is the set of vertices and E is the set of edges. V_m is a subset of V that represents the set of mandatory waypoints. A valid path starts from vertex v_{start} and reaches vertex v_{end} after having passed through all mandatory waypoints. A set Φ of variables $\varphi_e \in \{0, 1\}$ is defined, where each variable is associated to an edge e in order to model a possible path from v_{start} to v_{end}. An edge e does not belong to a feasible path when $\varphi_e = 0$. This is formulated as the following constraint, where $\omega(v)$ represents the set of incoming and outgoing edges of vertex v (incoming for ω^-, outgoing for ω^+). The graph is undirected, so that $\omega^+(v) = \omega^-(v)$ for each $v \in V$:

$$\sum_{e \,\in\, \omega^+(v_{start})} \varphi_e = 1, \quad \sum_{e \,\in\, \omega^-(v_{end})} \varphi_e = 1, \tag{1}$$

$$\forall v \in V \setminus \{v_{start}, v_{end}\}, \sum_{e \,\in\, \omega^+(v)} \varphi_e = \sum_{e \,\in\, \omega^-(v)} \varphi_e \leq 1 \tag{2}$$

Nodes v_{start} and v_{end} represent current position and primary objective for the vehicle respectively. Equation (2) ensures path connectivity and unicity while equation (1) imposes limit conditions on path start and end. These constraints give a linear chain alternating positions and mobility actions along the graph.

2.4. Capability metrics

Assuming a given date D_v associated with a position v, we are using a path length formulation (3) often considered in Operational Research (OR) [1]. Variable D_v expresses the time at which the vehicle reaches position v (see example in figure 2).

Assuming that constants $d_{(v,v')}$ represent the time taken to perform a movement from location v to v', we have:

$$\forall v \in V, \quad D_{v'} = \sum_{(v,v') \in \omega^-(v')} \varphi_{(v,v')}(D_v + d_{(v,v')}) \tag{3}$$

Constants $d_{(v,v')}$ are critical decision variables in the problem and make constraints (3) non linear. Finally, the mission schedule can be represented as $\Delta = \{(v, D_v)| \ v \in V, \ D_v > 0\}$. An equivalent constraint-based formulation is also used for other mission metrics (figure 2), such as energy or capacity.

2.5. The challenges

In traditional architecture design, the notion of *fast reactivity* is itemized as a system requirement. However, as described above, our problem is TSP-like and consequently of NP complexity. To keep tractability, the number of mandatory waypoints must remain realistic: on mission, rarely more than a ten of waypoints are imposed. In addition, in our approach, on-line planning is solved over a limited horizon from the current vehicle position. It corresponds to the terrain on which the vehicle vetronic has enough detailed information to characterise its trafficability. Therefore, the number of mandatory waypoints is relatively small. They correspond to short-term objectives or narrow manoeuvres executed by the vehicle.

Note that if a plan cannot be quickly solved, the vehicle may stop if this is possible. It is an ultimate solution that is not satisfactory from an operational point of view. Finally, due to vetronic processing resources, computation load, memory usage and response time must be reduced as much as possible.

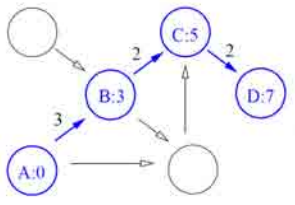
In the following, we present how a new hybrid approach, that mixes a complete method with a metaheuristic pre-processing mechanism, delivers compliant performances over real-case scenarios.

3. State of the Art

3.1. Mission Planners

3.1.1. Generic planners

Much research has been carried out on generic planning problems, often motivated by mission preparation in defense area. Domain-independent planners [2] and formalisms such as PDDL 2.0 [3] have emerged to tackle these complex problems. The related search methods can be complemented by pre-processing or dedicated heuristics to fit specific domain problems. However, these approaches may not match on-line requirements of reactive embedded systems.



Graph in 2.4 is a spatial representation of possible moves (edges) and positions (nodes). Moves, that correspond to the set of positive values $\Phi = \{(A, B), (B, C), (C, D)\}$, are represented with bold arrows. We are assuming an edge distance metric for optimization. Other operational constraints, such as protection, vulnerabiliy, available energy and security are similarly formulated in different experiments.

Fig. 2. Illustrating a path with pass-by dates over a graph of locations and progression axes.

3.1.2. Domain-dependent planners

Much research has been done for both military or civilian purposes, relying on specific planning frameworks such as Hierarchical Task Network (HTN)[4]. Some specific planning techniques fulfill on-line requirements, such as in [5], but may not encompass the spectrum of operational constraints.

3.1.3. CP planners

Constraint solving in planning has been integrated into various frameworks: this is the case of Ix-TeT [6] and HSTS [7]. In Reactive Model-based Programming Language (RMPL) [8], an evolution of Concurrent Constraint (CC) languages, the same paradigm is used to dynamically constrain planning representations of one or more remote agents.

3.2. Operational Research

Our problem can be considered using OR algorithms, based on flow models. As explained in section 2.1, it can be relaxed as a TSP, for which numerous algorithms have been proposed. Firstly, it can be tackled with a deterministic approach (see [1] for example). Local search methods have also been widely used for TSP problems [9]. To solve larger TSP problem instances, where deterministic approaches may take exponential time to solve, new metaheuristics have been developed. The most renown algorithms are Simulated Annealing [10], Genetic Algorithms [11] and Ant Colony Optimization (ACO) [12], for which various versions exist.

Most of CP frameworks are useful to design hybrid search techniques, by integrating OR and Linear Programming algorithms [13]. However, only a few ones such as [14], have explored on-line planning requirements.

3.3. Optimal path planning

A large number of heuristic-based search methods have been developed, mainly on the basis of the well-known A* [15]. Research has been done on *memory size limitation* [16], often on the principle of iterative-deepening searches [17]. Memory saving conditions are another family [18] that allow deleting the less interesting evaluated states. Others are dealing with the problem of *anytime solution availability* [19] to face situations with uncertain execution time. Finally, much work has been done to tackle *dynamic and uncertain environments*. Two families have emerged : incremental heuristic searches and real-time heuristic searches. Their properties are significantly different. Incremental searches consider the whole environment, and are optimal: to be efficient, they reuse information from previous searches. If a contingency occurs, propagation methods allow to update information and to prevent from reconsidering the entire problem. The most optimized incremental search algorithm is currently Dynamic FSA* [20]. On the other hand, real-time heuristic searches use a different approach by considering only a local portion of the environment. They are consequently suboptimal but very efficient on highly dynamic problem instances. Most recent works are hierarchical methods [21].

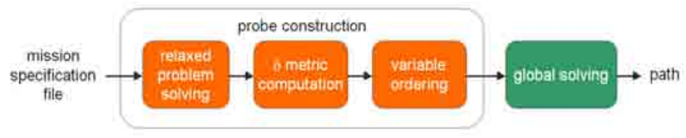


Fig. 3. Diagram of the complete solver using probing techniques.

4. Proposed Method

4.1. Global Solving

The global solving techniques use a Constraint Programming (CP) framework which combines a high level of expressiveness and powerful constraint solving techniques. The $\{0,1\}$ flow problem (see formulas (1) and (2)) is expressed with variables and arithmetical constraints. The formulation can support multiple distance metrics over paths, even non-linear ones (formula (3)). This approach is very interesting to represent tactical mobility (positions, progression axes, objectives, visibility...), and vehicle abilities (fuel, trafficability...). Problem formulation and global solving method have been implemented with the $CLP(FD)$ domain of SICStus Prolog library. It uses state-of-the-art in discrete constrained optimization techniques:

- Arc Consistency-5 (AC-5) [22] for constraint propagation, that is managed by $CLP(FD)$ predicates. When a variable domain get reduced, AC propagates domain variables until a fixed point is reached.
- Variable filtering with correct values, using specific labelling predicates to instantiate problem domain variables. AC being incomplete, value filtering guarantees the search completeness.
- Tree search with standard backtracking when variable instantiation fails.
- Branch and Bound (B&B) for cost optimization, using *minimize* predicate.

The global solving techniques under consideration guarantee search completeness, solution optimality and proof of optimality. Designing a good solving method consists in finding the right variables ordering and values filtering, using domain or generic heuristic, and in general implemented with some specific labelling predicates.

Other possibilities exist to reinforce global solving. Arc consistency can integrate domain-specific consistency rules, while global optimization can be improved in many ways (generating bounds, branch and cut, branch and price, iterative deepening...). These techniques are not in the scope of the probing hybridization described in this paper.

4.2. The Probing Method

4.2.1. Overview

The goal of hybridizing global solving with stochastic approaches is to save the number of backtracks and to quickly focus the search towards good solutions. It consists in designing the tree search according to problem structure, revealed by the probe.

The idea is to use the prober to statically order problem variables, as a pre-processing. Instead of dynamic probing with tentative values such as in [23], this search strategy uses a static prober which orders problem variables to explore according to the relaxed solution properties. Then, the solving follows a standard CP search strategy, combining variable filtering, AC-5 and B&B.

As shown in figure 3, the probing technique proceeds in three steps (the three blocks on the left). The first one is to establish the solution to the relaxed problem. As a reference, we can for example compute the shortest path between starting and ending vertices, abstracting away mandatory waypoints. The next step is to establish a minimal distance $\delta(v)$ between any problem variable and the solution to the relaxed problem. This step can be formally described as follows. Let $V_s \subset V$ be the set of vertices that belong to the relaxed solution. The distance is given by the following evaluation:

$$\forall v \in V, \delta(v) = \min_{v' \in V_s}(min\_distance(v, v')) \tag{4}$$

where the distance is the number of vertices between v and v'. The last step uses the resulting partial order to sort problem variables in ascending order. At the global solving level the relaxed solution is useless, but problem variables are explored following this order.

4.2.2. Interests

Two interesting probe properties can be highlighted:

- *probe complexity*: since computation of minimum distance is polynomial between a vertex and any node is polynomial thanks to Dijsktra or Bellman-Ford algorithms, the resulting probe construction complexity is still polynomial in worst cases. The complexity of quicksort can in practice be neglected (see below for further details).
- *probe completeness*: since the probe does not remove any value from variable domains and the set of problem variables remains unchanged, the probe still guarantees global solving completeness.

Complexity analysis. Let γ be the cardinality of V_s and n the one of V. The complexity of probe construction is:

- worst case performance: $O(n^2)$;
- average case performance: $O(\gamma.n.\log(n))$.

Sketch of the proof. The probing method first determines the minimal distance between all vertices $v' \in V'$ where $V' = V \setminus V_s$ and any vertex $v_s \in V_s$. A Dijkstra algorithm run over a vertex v_s allows to compute the distance to any point of V' with $O(n.\log(n))$ worst case complexity where n is the number of nodes in V. This has to be run over each vertex of V_s and a comparison with previous computed values must be done for every vertex v', to keep the lowest one. Thus, the resulting complexity is $O(\gamma.n.\log(n))$. Variables must finally be sorted with a quicksort-like algorithm. The worst case complexity of this sort is $O(n^2)$ but is generally computed in $O(n.\log(n))$

(average case performance). Hence, the worst case complexity of the probing method is $O(n^2)$, but in practice behaves in $\max\{O(\gamma.n.\log(n)), O(n.\log(n))\} = O(\gamma.n.\log(n))$.

4.2.3. Pseudocode

Algorithm 1 synthesizes probe construction mechanisms. Firstly, a vector L_d of size n (n being the number of nodes in V) is created and initialized with infinite values. At the end of the execution, it will contain a value associated to each vertex, corresponding to the minimal distance between this vertex and the solution to the relaxed problem. To do so, a Dijkstra algorithm is run over each node of the solution. During a run, distances are evaluated and replaced in L_d if lower than the existing value (in the pseudocode, comparisons are made at the end of a run for easier explanation). Once minimal distances are all computed, they are used to rank the set of vertices V in ascending order (to be used by the complete solver).

Algorithm 1 Probe construction

```
1: Initialize a vector $L_d$ of distances (with infinite values)
2: Get $P$ the best solution of the relaxed problem
3: for each node $v_i$ of $P$ do
4:     $L'_d \Leftarrow$ Run Dijkstra algorithm from $v_i$
5:     $L_d = \min(L_d, L'_d)$ (value by value)
6: end for
7: Sort $V$ using $L_d$ order
8: return the newly-ordered $V$ list
```

4.3. A Stochastic Relaxed Problem Solver

Instead of considering a blind shortest path to solve the relaxed problem, the proposed algorithm implements an Ant Colony Optimization (ACO) search [12] that uses a similar model of the environment.

4.3.1. The Ant Colony algorithm

ACO belongs to the family of swarm intelligence metaheuristics. It has been initially developed to solve TSP instances, and is more generally well defined for discrete and possibly dynamic problems. It spreads a population of ants through the state space and iteratively reuses collective memory to improve the search. Similarly to the notion of generations in Genetic Algorithms, ACO deploys a series of search cycles. During a cycle, each ant builds a solution thanks to a probability law using both a guiding heuristic and the collective memory information. The latter is defined as an edge weight that varies over time and represents the *pheromone rate*. In nature, ants disseminate this chemical substance to remind the path. The shorter the path, the sooner the path will be taken by other ants and consequently the higher the pheromone rate will be. In the ACO algorithm, the pheromone model is updated at the end of a cycle : solutions are compared and best ones are used to improve collective memory by reinforcing related edges.

Formally, an ant builds a path through the state space by electing iteratively the next vertex to take to reach the goal. To do so, it uses the following formula. For an ant k currently at vertex v, the probability for choosing a reachable vertex v' as its next waypoint is given by formula (5).

$$\forall (v,v') \in E, v' \in \omega^+(v), P_{v'}(k) = \frac{\tau^{\alpha}_{v,v'}\eta^{\beta}_{v'}}{\sum\limits_{v''\in\omega^+(v)} \tau^{\alpha}_{v,v''}\eta^{\beta}_{v''}} \tag{5}$$

$P_{(v,v')}$ is a probability and thus belongs to $[0,1]$. The $\eta_{v'}$ parameter is the guiding heuristic, which is described below. The $\tau_{(v,v')}$ parameter is the pheromone edge weight to go from vertex v to vertex v'. It represents the experience acquired during previous search cycles and tends to choose edges that belong to known good solutions. α and β are calibration variables that balance the importance given to τ and η parameters. It has an impact on algorithm convergence, as a search with a strong β value will be very orientated but may not allow a correct space exploration and thus an unexpected better solution discovery. The pheromone model update is made using the following formula:

$$\forall (v,v') \in E, \tau_{v,v'}(c+1) = \rho.\tau_{v,v'}(c) + \Delta\tau_{v,v'} \tag{6}$$

where ρ is the conservation factor ($1-\rho$ corresponds to the pheromone *evaporation* factor, in analogy with real ants). It allows decreasing pheromone weight over time to remove attraction from bad paths. $\Delta\tau_{v,v'}$ equals to:

$$\Delta\tau_{v,v'} = \begin{cases} \frac{1}{L_{LB}} & \text{if } (v,v') \text{ belongs to the} \\ & \text{local best solution} \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

where L_{LB} is the length of the local best solution. Some improvements of the initial ACO algorithm can be made using [24, 25] but they are not discussed in this paper.

4.3.2. Adaptation to the problem

In the original TSP implementation of ACO, initial ant positions are randomly defined. The guiding heuristic $\eta_{v'}$ returns the distance inverse from the current point to v': the closer this point, the higher its probability to be selected.

In our implementation, ACO is not only used to solve the TSP-like problem (which is the sequence of mandatory points). It is also used to find the shortest path between each mandatory point. Consequently, the original guiding heuristic definition is not satisfactory as the shortest path between two points may be a long but *straightforward* edge instead of several small but derivating ones. That's why a new definition has been given to $\eta_{v'}$: it is henceforth the distance inverse to go from the candidate vertex v' to the goal. Thus, the algorithm will tend to choose the vertex that is the closest to the goal. In addition, all searches are starting either from the current point or from the final objective (the path is not a loop anymore).

The guiding heuristic orientates the search towards the final goal, but mandatory waypoints may not be aligned with the start-goal axis. Thus the probability that a path contains all the mandatory waypoints is very low, and the search will have a poor success rate. To counter this problem, intermediate goals are iteratively elected during an ant search. Currently at point v_m, the election probability of a v'_m point is made using formula (8).

$$\forall v'_m \in V_m, P_{v'_m} = \frac{\eta_{v_m,v'_m} + D_{v'_m}}{\sum\limits_{v''_m\in V_m} \eta_{v_m,v''_m} + D_{v''_m}} \tag{8}$$

η_{v_m,v'_m} is the distance inverse to go from v_m to v'_m and tends to choose the closest point as the next intermediate goal. $D_{v'_m}$ is the distance proportional function to go from v'_m to the goal and tends to avoid from keeping isolated points along the search.

4.3.3. Pseudocode of the algorithm

Algorithm 2 ACO algorithm

```
1:  Initialize pheromone model
2:  Set global best path $P_{GB}$ to *null*
3:  **for** $c \Leftarrow 1$ to $C$ **do**
4:      **for** each ant of the population **do**
5:          Run single ant search
6:      **end for**
7:      Get local best solution $P_{LB}$ over all searches
8:      **if** $P_{LB} \leq P_{GB}$ **then**
9:          $P_{GB} \Leftarrow P_{LB}$
10:     **end if**
11:     Update pheromone model
12: **end for**
13: **return** the best solution $P_{GB}$
```

Algorithm 3 Run single ant search

```
1:  Initialise path $P \Leftarrow \{v_{start}\}$
2:  Define the current position $v_{current} \Leftarrow v_{start}$
3:  Define $M$ the set of mandatory waypoints
4:  **while** $M \neq \emptyset$ **do**
5:      Elect an intermediate goal $v_{goal}$ from $M$
6:      Remove $v_{goal}$ from $M$
7:      Reach objective $v_{goal}$
8:  **end while**
9:  Reach objective $v_{end}$
10: **return** $P$
```

Algorithm 4 Reach objective v_{obj}

```
1: **while** $v_{current} \neq v_{obj}$ **do**
2:     Elect next waypoint $v_{next}$
3:     Verify $v_{next}$ validity
4:     Add $v_{next}$ to $P$
5:     $v_{current} \Leftarrow v_{next}$
6: **end while**
7: **return**
```

Algorithm 5 Elect next waypoint

```
1: Get the set $V'$ of $v_{current}$ neighbors
2: Compute probability for each $v' \in V'$
3: Elect the next waypoint $v_{next}$
4: **return** The next waypoint $v_{next}$
```

Algorithm 2 is the main instance, and returns the best path found. The input parameters are the graph G, the list of mandatory waypoints M, and the starting and ending nodes. At first, the pheromone model P is created and initialized with constant values (that is, every edge is given a same weight). Then a loop is used to execute the C search cycles. In this loop, a vector of length N is created (N being the number of ants), corresponding to the list of potential solutions built by ants. This vector is filled iteratively using a second loop that each time runs a single search. Once all searches are achieved, the best local solution P_{LB} (i.e. the shortest one) is found, then compared to the current best solution P_{GB} and saved if better. And before a new series of searches is done, P is updated using the most interesting solutions locally found. The pheromone model update procedure is not detailed here but follows from formula (6) rules.

Algorithm 3 builds a single path and returns it (or may fail) as a candidate solution. A path is a list of vertices that can be reached two-two (it only contains the starting point at initialization). To ensure passing through all intermediate objectives, one is picked from M using formula 8 and a single search (on line 7) is launched to reach it. As long as the M list is not empty, a new intermediate goal is elected and the path is completed. Finally, a search is conducted to reach the final goal.

Algorithm 4 is a search that builds a path from the current point to a specified objective. To do so, the "ant" repeatedly chooses the next waypoint to pass through (on line 2). As the path may have taken a vertex that belongs to the M list, a check is proceeded. Consequently, if there are x mandatory waypoints in M, at most $x + 1$ single searches will be performed. Note that a search may fail if the ending node is expanded before all mandatory waypoints have been reached.

Algorithm 5 details operations done to choose a next waypoint. Firstly, the list of neighbors (accessible nodes) is built. Then, respective probabilities to be elected are computed using formula (5).

In the current implementation, C and N are constants and defined statically in a configuration file.

5. Results

This section focuses on the comparison of our hybrid approach with two other methods. The first one is only based on the complete solver (presented in section 4.1) without probing. We call it the *reference* algorithm in the following. The second method is the complete solver coupled with the probing mechanism that uses a basic shortest path algorithm (described in section 4.2). We simply call it the *shortest path* algorithm in the following.

5.1. Benchmarks

To make our comparison, the three methods are tested over three distinct benchmarks that correspond to real-world scenarios. They are representative of vehicle planning for modern peace-keeping missions, both in urban and open environments. The figure 5.1 gives an idea of problem complexity.

For each benchmark, five series of executions are done: between two series, a new mandatory waypoint is added (there are consequently 1 to 5 waypoints per series). For each series, ten distinct runs are led (distinct means that the couple $(start, end)$ changes).

5.2. ACO calibration

5.2.1. Discussion

There is no universal rule to parameterize the ACO algorithm. It depends on the problem, essentially in terms of graph size and

	Bench1	Bench2	Bench3
Environment	urban	urban	open
Vertices	23	22	22
Edges	76	74	68
Variables	723	654	702
Constraints	1944	1750	1886

Above: table of benchmark characteristics (the number of edges is considered over the directed graph).
On the right: graph of benchmark 2. Various mandatory waypoints (gray nodes) can be imposed. Black circles represent possible starting or ending nodes.

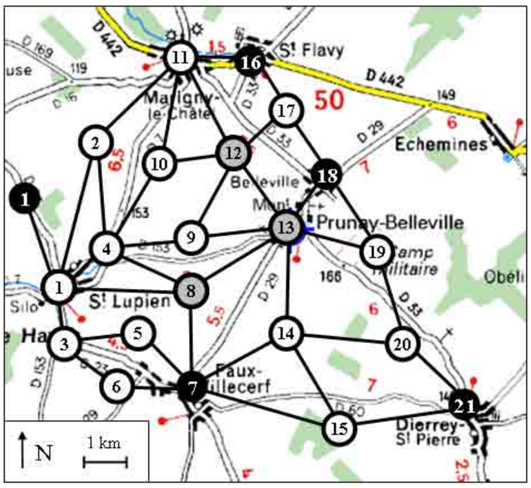


Fig. 4. Benchmark overview

connexity. The choice of α and β can be decisive if there is a high risk for the search to follow a dead-end path. In this case, by privileging the η term (thus by setting β higher than α), our research will have more chance to fail (or to find bad solutions) if it takes a path in direction to the objective but which does not lead to this one. An analogy can be made with A-star algorithm, whose worse case is a labyrinth in which the only way to reach the objective is opposite to the location's direction. In the other cases, it can be judicious to take more consideration to η that can fastly lead to good solutions. The N parameter depends on the size of the problem. The larger the problem, the more the ant population (represented by N) should be important, as the number of possible solutions becomes high. The C parameter (number of search cycles) more particularly depends on the learning mechanisms. As we know, the quality of the research is a compromise between state space exploration and convergence speed. According to the chosen strategy, C should be low if a fast convergence is wished (the reinforcing edge parameters should then evolve fastly), or high in the contrary (and reinforcing should be mild to maintain alternative paths).

5.2.2. *Parameter values*

For this test, we calibrated the ACO algorithm with the following parameters:

- $C = 10$;
- $N = 10$;
- $\alpha = 1$;
- $\beta = 5$;
- $\tau_{init} = 0.5$;
- $\rho = 0.9$.

These values were set empirically after comparing performances on several tries.

5.3. *Comparison criteria*

The performance of a method is evaluated depending on several criteria:

- the time to find the optimal solution;
- the time to prove the optimality;
- the number of backtracks done by over the branch & bound method;
- the memory space required.

Experiments have been run on a dual core CPU working at 2.53 GHz with 2 GB of memory. The results are presented below.

5.4. *Results*

5.4.1. *Execution time*

During the runs, execution time was limited to ten seconds. It corresponds to the upper limit of satisfiability: over this bound, it is considered as unacceptable. In the following, a "X" time value indicates a computation overrun.

The table in figure 5 brings the results in terms of execution time and backtracking. In the columns, the three algorithms are compared : REF is the reference algorithm, $SPATH$ is the shortest-path-based algorithm, and $ANTS$ is the ACO-based algorithm. For each method, the table presents:

- (a): the time needed to find the optimal solution respectively;
- (b): the time needed to prove optimality;
- (c): the number of backtracks needed to find the optimal solution.

For each, three values are given:

- MIN: the minimal value over the ten runs;
- MAX: the maximal value over the ten runs;
- $TOTAL$: the total value over the ten runs (i.e. the sum of the value for each run).

If a series of runs contains at least one value that exceeds the ten seconds limit (i.e. is marked with a "X"), then the $TOTAL$ value must not be considered as exact. It should be greater in real case because a run that failed finding a solution within the bound is saved as an execution time of ten seconds (it would normally be more). A series of runs is represented as a row. Far left is the number of the benchmark. Each benchmark is evaluated using

			REF			SPATH			ANTS		
			(a)	(b)	(c)	(a)	(b)	(c)	(a)	(b)	(c)
BENCH1	1MW	MIN	16	2984	2	0	297	0	31	140	0
		MAX	7656	x	23594	422	1750	719	375	3360	659
		TOTAL	28172	59716	91319	1139	8828	1716	938	11640	659
	2MW	MIN	0	1453	0	32	328	20	62	250	0
		MAX	5906	x	24320	453	2156	919	188	1734	180
		TOTAL	16045	48016	52761	1500	10843	2478	1048	7719	454
	3MW	MIN	2547	7109	9363	0	359	1	63	532	0
		MAX	x	x	30860	187	2313	6271	141	2969	72
		TOTAL	71704	93687	208666	702	11719	6963	863	18641	82
	4MW	MIN	1312	1890	3518	0	282	1	63	329	0
		MAX	x	x	20204	141	1703	195	407	1781	2103
		TOTAL	46875	61485	107614	845	8577	964	1312	12781	2841
	5MW	MIN	3578	5813	9411	16	594	1	78	703	0
		MAX	9454	x	24691	359	1672	6969	219	2078	225
		TOTAL	56003	84313	153560	1436	11343	9135	1404	14421	939
	ALL	MIN	0	1453	0	0	282	0	31	140	0
		MAX	x	x	30860	453	2313	6969	407	3360	2103
		TOTAL	218799	347217	613920	5622	51310	21256	5565	65202	4975
BENCH2	1MW	MIN	0	735	0	0	125	0	31	156	0
		MAX	7297	9797	17107	297	625	527	79	641	0
		TOTAL	13859	27064	36000	685	3484	1081	580	4157	0
	2MW	MIN	0	750	0	16	156	1	32	156	0
		MAX	8609	x	19429	375	796	745	359	1218	531
		TOTAL	11796	23872	30504	1138	3858	1835	1034	6078	705
	3MW	MIN	0	734	0	47	141	16	62	219	0
		MAX	4016	6063	8337	343	687	624	94	1063	46
		TOTAL	5406	17812	13385	1465	3891	2394	797	5486	48
	4MW	MIN	16	922	32	16	156	3	46	328	0
		MAX	5937	7453	13397	391	891	671	156	1172	54
		TOTAL	12279	26090	26936	736	4095	844	905	7030	101
	5MW	MIN	94	735	158	31	188	3	78	391	0
		MAX	3125	4312	6695	234	703	339	547	1906	786
		TOTAL	8313	19298	17677	985	4014	1077	1750	9047	1351
	ALL	MIN	0	734	0	0	125	0	31	156	0
		MAX	8609	x	19429	391	891	745	547	1906	786
		TOTAL	51653	114136	124502	5009	19342	7231	5066	31798	2205
BENCH3	1MW	MIN	15	453	0	15	344	0	47	406	0
		MAX	2204	2735	5070	94	1750	133	79	1750	12
		TOTAL	4907	11766	11067	250	8874	170	658	8767	14
	2MW	MIN	16	203	20	46	125	18	62	110	0
		MAX	594	984	1104	2734	3782	5403	640	3282	1127
		TOTAL	2612	6266	5209	7515	13173	15340	1935	16205	2266
	3MW	MIN	47	454	16	15	250	11	47	328	0
		MAX	719	1984	1649	1656	3329	3839	922	2797	1650
		TOTAL	2691	10533	5298	3015	12940	6178	1719	14470	1766
	4MW	MIN	31	375	41	15	218	4	93	547	2
		MAX	1281	1703	2694	1609	2390	3108	547	2469	797
		TOTAL	4719	8265	9383	4484	9483	8767	1806	14548	1749
	5MW	MIN	31	281	46	15	234	4	94	547	1
		MAX	984	1297	2092	860	1359	1535	984	2031	1822
		TOTAL	4265	7263	8628	2874	7109	5258	3125	13000	4945
	ALL	MIN	15	203	0	15	125	0	47	110	0
		MAX	2204	2735	5070	2734	3782	5403	984	3282	1822
		TOTAL	19194	54093	39585	18138	51579	35713	9243	66990	10740

Fig. 5. Benchmark results over a total of 450 runs.

	REF	SPATH	ANTS
BENCH1	284	162	**76**
BENCH2	215	167	**131**
BENCH3	236	320	**234**

Fig. 6. Maximum memory space required for a search (in MB).

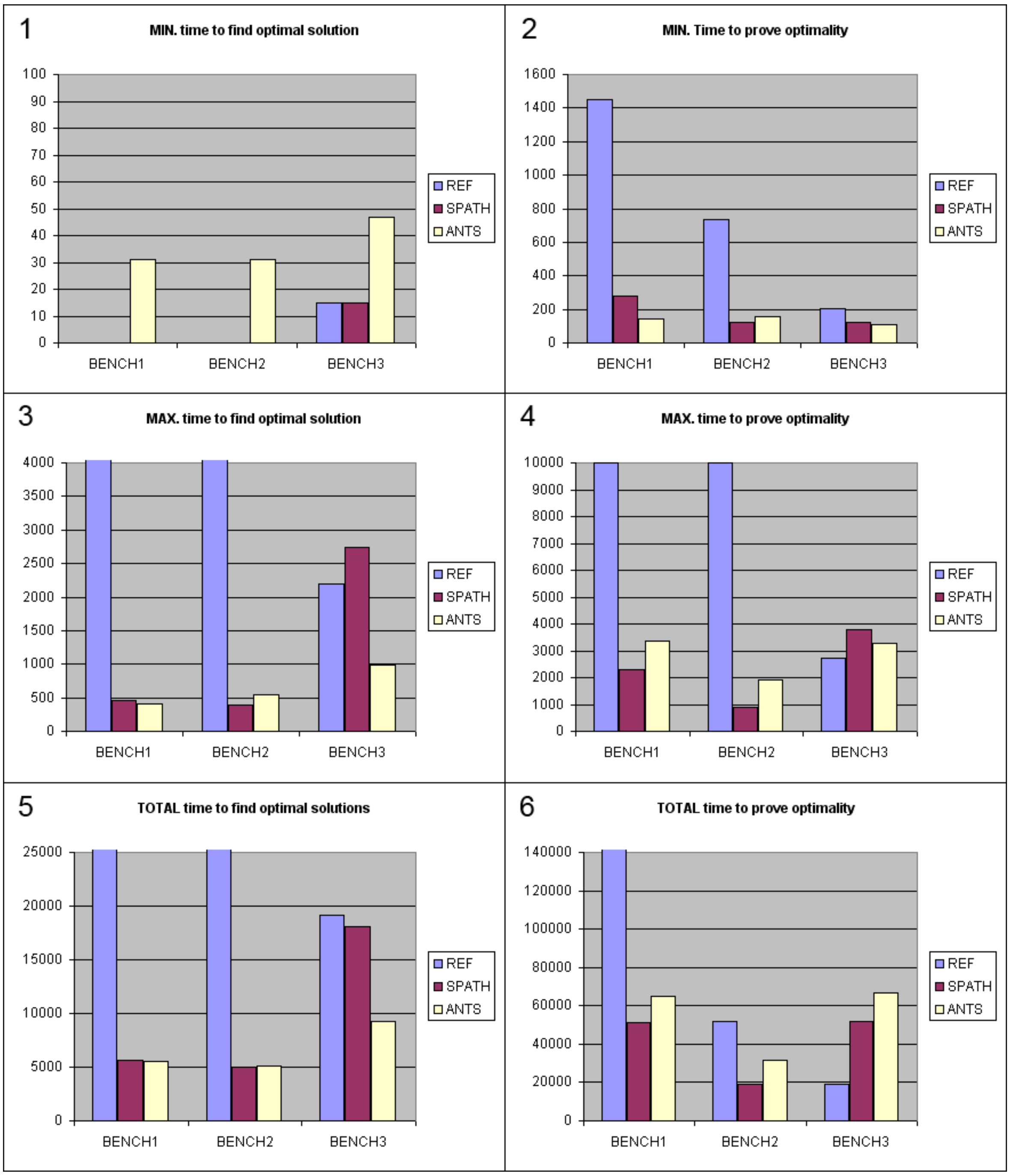


Fig. 7. Graphs illustrating main results from data table.

a different number of mandatory waypoints (MW), from one to five. The ALL rows are a synthesis of the five series over a same scenario (MIN, MAX and $TOTAL$ are consequently evaluated over the fifty runs).

Lastly, execution time is expressed in milliseconds. It is important to notice that these results were obtained using Prolog, that uses operating system function calls. Consequently, time values have a precision (or ”delta”) range of twenty to thirty milliseconds.

5.4.2. Memory consumption

Table 6 summarizes the maximum space required over all runs of each algorithm, for the three benchmarks. As memory is freed between two searches, it is unnecessary to present the memory amount needed for each run. Units are in megabytes, and values are rounded to the nearest greater integer (ceil). Bold values are lowest values over the three algorithms.

5.5. Analysis

We do not focus on the number of backtracks, as the results are strongly correlated with solving time. Here we analyze the results in terms of execution time, whose values are identified by, (a) and (b) in the table of figure 5. To have a clearer view of these results, the figure 7 shows a series of graphs illustrating data from ALL rows of the table in figure 5.

Graph 1 shows the minimal execution time needed to find the optimal solution. As we can see, REF and $SPATH$ may find optimal solutions very fastly, whereas $ANTS$ takes more time. It corresponds to the *overhead*, that is the execution time needed to run ACO searches. Graphs 3 and 5 show the maximal and total execution times needed to find the optimal solution. The latter allows to get an average execution time information (by dividing the total value by fifty, which is the number of runs considered). The first finding is that probing methods are very efficient on benchmarks 1 and 2, whereas the reference method (without probing) is not. The latter even exceeds several times the ten seconds execution time limit, that means it did not find the best solution within this bound. The second finding is that $SPATH$ and $ANTS$ methods are almost equally effective over the benchmarks 1 and 2. However, one can see that $SPATH$ is very inefficient on benchmark 3. It even has lower efficiency than the reference method. It is due to the fact that mandatory waypoint locations were quite distant from the start-goal axis. $ANTS$ is approximately two times faster than $SPATH$ on this benchmark. As the overhead of $ANTS$ is around 70 ms (3500 ms over 50 runs), the investment is about 5% of total $SPATH$ computation time and the gain is about 50%. As a first conclusion, over these three benchmarks, the ACO-based method is proven to be more efficient as it finds the best solution faster than the two other methods. It is more stable as it adapts to various problems without excessive performance variations.

Graphs 2, 4 and 6 respectively show minimal, maximal and total execution time needed to find the best solution and to prove its optimality. Surprisingly, results are quite different from previously. REF algorithm is still very inefficient over benchmark 1. Over benchmark 2, REF is not so inefficient as the total (and consequently the average) execution time remains correct. In fact, the problem is due to the failure of a search in the series with 2 mandatory waypoints (see in table 5). The reference algorithm even reveals to be very efficient in benchmark 3 and outperforms both $SPATH$ and $ANTS$! In addition, $SPATH$ is more efficient than $ANTS$: in terms of total computation time, it spends 39% less time over benchmark 2, and 21% less time over benchmarks 1 and 3. This has to be tempered by the fact that $ANTS$ needs more precomputation time, which leads results of the same order. But the conclusion is that $ANTS$ does not allow to prove the solution faster.

In terms of memory consumption (see table of figure 6), $ANTS$ algorithm reveals to have the best performances over the three approaches, for each benchmark.

6. Conclusion

We presented a hybrid approach that mixes an exact solving method guided by a metaheuristic. The latter uses a stochastic ACO algorithm to solve a relaxed version of the problem to order the variables. This pre-processing step allows the backtracking method of the complete solver to select the most promising variables first. We focused our attention on the adaptation of ACO to this problem and compared it to a deterministic implementation on small problem instances.

Through three realistic series of benchmarks, we showed that the ACO-based approach is as fast as using a brute-force shortest path on easy problems, despite the computation overhead. On most complex cases (when mandatory waypoints are distant from the start-end axis), we recorded a gain of 50% on average computation time, for 5% of extra pre-processing time. Our approach is consequently more stable and could easily solve bigger problem instances while it would rapidly become intractable for the deterministic approach. The ACO-based algorithm also revealed better memory consumption performances, which is very interesting in terms of on-line applications. Additionally, it could have interesting properties in terms of replanning capabilities. However, we also found that the method does not allow to prove optimality faster than the deterministic method.

References

[1] M. Gondran and M. Minoux, "Graphes et algorithmes," 1995.

[2] M. Fox and D. Long, "Automatic synthesis and use of generic types in planning," *Proceedings of the Artificial Intelligence Planning System*, pp. 196–205, 2000.

[3] M. Fox and D. Long, "Pddl2.1: An extension to pddl for expressing temporal planning domains," *Journal of Artificial Intelligence Research*, vol. 20, 2003.

[4] D. M. R. Goldman, K. Haigh and M. Pelican, "Macbeth: A multi-agent constraint-based planner," *Proceedings of the 21st Digital Avionics Systems Conference*.

[5] D. S. N. Meuleau, C. Plaunt and T. Smith, "Emergency landing planning for damaged aircraft," *Proceedings of the 21st Innovative Applications of Artificial Intelligence Conference*, pp. 3247–3259, 2009.

[6] P. Laborie and M. Ghallab, "Planning with sharable resource constraints," *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1995.

[7] N. Muscettola, *Hsts: Integrating planning and scheduling*. Robotics Institute, Carnegie Mellon University, 1993.

[8] P. K. M. Abramson and B. Williams, "Executing reactive, model-based programs through graph-based temporal planning," *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2001.

[9] E. Aarts and J. Lenstra, *Local Search in Combinatorial Optimization*. 1997.

[10] V. Cerny, "A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm," *Journal of Optimization Theory and Applications*, vol. 45, pp. 41–51, 1985.

[11] D. Goldberg, "Genetic algorithms in search, optimization and machine learning," 1989.

[12] M. Dorigo and L. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, pp. 53–66, 1997.

[13] F. Ajili and M. Wallace, *Constraint and integer programming: Toward a unified methodology*. McGraw Hill, Kluwer Academic Publishers, 2003.

[14] R. K. G. R. S. Chien, B. Engelhardt and R. Sherwood, "Casper: Space exploration through continuous planning," *Journal of IEEE Intelligent Systems*, vol. 16, 2001.

[15] N. N. P. Hart and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems, Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[16] A. Reinefeld and T. Marsland, "Enhanced iterative-deepening search," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 7, pp. 701–710, 1994.

[17] R. Korf, "Depth-first iterative-deepening: An optimal admissible tree search," *Artificial Intelligence*, vol. 27, pp. 97–109, 1985.

[18] S. Russel, "Efficient memory-bounded search methods," *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI)*, pp. 1–5, 1992.

[19] E. Hansen and R. Zhou, "Anytime heuristic search," *Journal of Artificial Intelligence Research*, vol. 28, pp. 267–297, 2007.

[20] X. S. S. Koenig and W. Yeoh, "Dynamic fringe-saving a*," *Proceedings of the 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, vol. 2, pp. 891–898, 2009.

[21] M. M. A. Botea and J. Schaeffer, "Near optimal hierarchical pathfinding," *Journal of Game Development*, vol. 1, no. 1, 2004.

[22] Y. Deville and P. V. Hentenryck, "An efficient arc consistency algorithm for a class of csp problems," in *Proceedings of the 12th International Joint Conference on Artificial intelligence (IJCAI)*, vol. 1, pp. 325–330, 1991.

[23] H. E. Sakkout and M. Wallace, "Probe backtrack search for minimal perturbations in dynamic scheduling," *Constraints Journal*, vol. 5, no. 4, pp. 359–388, 2000.

[24] T. White, S. Kaegi, and T. Oda, "Revisiting elitism in ant colony optimization," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, vol. 2723/2003, p. 199, Springer Berlin / Heidelberg, 2003.

[25] B. Bullnheimer, R. Hartl, and C. Strauß, "A new rank based version of the ant system - a computational study," *Central European Journal for Operations Research and Economics*, vol. 7, no. 1, pp. 25–38, 1997.

François Lucas was born in France in 1985. He received his Master's degree from ESIEE Paris in 2008, with a major in Real-Time Embedded Systems His internships led him to adapt a ground robot into fully-automated driven system (*Sagem Defense & Security*, 2008) and to design an environment simulator aiming at validating various CBTC equipments separately (*Thales Rail Signaling Solution*, 2009). Since December 2008 he is a PhD student in Mines ParisTech. He is working in collaboration with Sagem on efficient solving strategies for constrained path planning to be embedded in military troop transport vehicles. His main research interests are new hybridization techniques of metaheuristics and complete constraint solvers.

Christophe Guettier had his PhD in 1997 from "Ecole des Mines de Paris" in the area of combinatorial optimisation applied to the engineering of large-scale parallel systems. Then, he joined the SME "Axlog Ingénierie" where he developed some researches on autonomous systems for the European Space Agency (spacecraft constellation, deep space probes) and for Dassault Aviation (unmanned combat aerial vehicle). In 2001, he has been recruited by Xerox PARC to work on the DARPA NEST (Networking Embedded System Technology) project led by Air Force Research Lab and Boeing. In 2002, he joined Imperial College London where he developed a research axis on solving constrained routing problems for ad hoc networks. He joined SAGEM DS in 2004 where he leads several new product developments.

Patrick Siarry was born in France in 1952. He received the PhD degree from the University Paris 6, in 1986 and the Doctorate of Sciences (Habilitation) from the University Paris 11, in 1994. He was first involved in the development of analog and digital models of nuclear power plants at Electricité de France (E.D.F.). Since 1995 he is a full professor in automatics and informatics. His main research interests are the applications of new stochastic global optimization heuristics to various engineering fields. He is also interested in the fitting of process models to experimental data, the learning of fuzzy rule bases, and of neural networks.

Anne-Marie Milcent is an Engineer graduated from Ecole Centrale Paris, and has also a Master of Science from Standford University in Management Science and Engineering. She started as a leading-edge R&D engineer in Sagem Defense Security, working on helicopter and airborne observation equipments. During two years, she has been project leader in haptic man - machine interfaces, opening the way of new generation interfaces for soldier systems. Then, she became in charge of R&D group in future combat system programs where she has coordinated several robotic and soldier systems architecture projects. She recently joined the medical industry where she is now a R&D leader in the GUERBET group.

Arnaud de La Fortelle is civil servant at the French Transport Ministry and is both director of the Mines robotics lab (CAOR) and of the Joint Research Unit LaRA (La Route Automatisée - the automated road, an INRIA-Mines ParisTech Consortium). He managed for LaRA several French and European projects (Puvame, Prevent/Intersafe, REACT, COM2REACT...) and is currently coordinator of the European project GeoNet and of the French project AROS. He has a Ph.D. in Applied Mathematics and engineer degrees for the French Ecole Polytechnique and Ecole des Ponts et Chaussées.