Optimal formation Switching with collision avoidance and allowing variable agent velocities*

Dalila B.M.M. Fontes and Fernando A.C.C. Fontes and Amélia Caldeira

Abstract We address the problem of dynamically switching the geometry of a formation of a number of undistinguishable agents. Given the current and the final desired geometries, there are several possible allocations between the initial and final positions of the agents as well as several combinations for each agent velocity. However, not all are of interest since collision avoidance is enforced. Collision avoidance is guaranteed through an appropriate choice of agent paths and agent velocities. Therefore, given the agent set of possible velocities and initial positions, we which to find their final positions and traveling velocities such that agent trajectories are apart, by a specified value, at all times. Among all the possibilities we are interested in choosing the one that minimizes a predefined performance criteria, e.g. minimizes the maximum time required by all agents to reach the final geometry. We propose here a dynamic programming approach to solve optimally such problems.

Dalila B.M.M. Fontes
LIAAD - INESC Porto L.A. and Faculdade de Economia, Universidade do Porto
Rua Dr. Roberto Frias, 4200-464 Porto, Portugal.
e-mail: fontes@fep.up.pt

Fernando A.C.C. Fontes
Instituto de Sistemas e Robótica do Porto and Faculdade de Engenharia, Universidade do Porto
Rua Dr. Roberto Frias, 4200-465 Porto, Portugal.
e-mail: faf@fe.up.pt

Amélia C.D. Caldeira
Departamento de Matemática, Instituto Superior de Engenharia do Porto,
R. Dr. Ant. Bernardino de Almeida, 431, 4200-072 Porto, Portugal.
e-mail: acd@isep.ipp.pt

* Research supported by FCT and FEDER through Project PTDC/EEA-CRO/100692/2008 Perception-Driven Coordinated Multi-Robot Motion Control

1 Introduction

In this paper, we study the problem of switching the geometry of a formation of undistinguishable agents by minimizing some performance criterion. The questions addressed are, given the initial positions and a set of final desirable positions, which agent should go a specific final position, how to avoid collision between the agents, and which should be the traveling velocities of each agent between the initial and final positions. The performance criterion used in the example explored is to minimized the maximum traveling time, but the method developed – based on dynamic programming — is sufficiently general to accommodate many different criteria.

Formations of undistinguishable agents arise frequently both in nature and in mobile robotics. The specific problem of switching the geometry of a formation arises in many cooperative agents missions, due to the need to adapt to environmental changes or to adapt to new tasks. An example of the first type is when a formation has to go through a narrow passage, or deviate from obstacles, and must reconfigure to a new geometry. Examples of adaption to new tasks arise in robot soccer teams: when a team is in an attack formation and looses the ball, it should switch to a defence formation more appropriate to the new task. Another example is in the detection and containment of a chemical spillage, the geometry of the formation for the initial task of surveillance, should change after detection occurs, switching to a formation more appropriate to determine the perimeter of the spill.

Research in coordination and control of teams of several agents (that may be robots, ground, air or underwater vehicles) has been growing fast in the past few years. Application areas include unmanned aerial vehicles (UAVs) [4, 18], autonomous underwater vehicles (AUVs) [16], automated highway systems (AHSs) [3, 17] and mobile robotics [20, 21]. While each of these application areas poses its own unique challenges, several common threads can be found. In most cases, the vehicles are coupled through the task they are trying to accomplish, but are otherwise dynamically decoupled, meaning the motion of one does not directly affect the others. For a survey in cooperative control of multiple vehicles systems, see for example the work by Murray [11]. Regarding research on the optimal formation switching problem, it is not abundant, although it has been addressed by some authors. Desai *et al.*, in [5], model mobile robots formation as a graph. The authors use the so-called "control graphs" to represent the possible solutions for formation switching. In this method, for a graph having n vertices there are $n!(n-1)!/2^{n-1}$ control graphs, and switching can only happen between predefined formations. The authors do not address collision or velocity issues. Hu and Sastry [9] study the problems of optimal collision avoidance and optimal formation switching for multiple agents on a Riemannian manifold. However, no choice of agent traveling velocity is considered. It is assumed that the underlying manifold admits a group of isometries, with respect to which the Lagrangian function is invariant. A reduction method is used to derive optimality conditions for the solutions. In [19] Yamagishi describes a decentralized controller for the reactive formation switching of a team of autonomous mobile robots. The focus is on how a structured formation of agents can reorganize into a non-rigid formation based on changes in the environment. The controller uti-

lizes nearest-neighbor artificial potentials (social rules) for collision-free formation maintenance and environmental changes act as a stimulus for switching between formations. A similar problem, where a set of agents must perform a fixed number of different tasks on a set of targets, has been addressed by several authors. The methods developed include exhaustive enumeration (see Rasmussen *et al.* [13]), branch-and-bound (see Rasmussen and Shima [12]), network models (see Schumacher *et al.* [14, 15]), and dynamic programming (see Jin *et al.* [10]). None of these works address velocity issues.

A problem of formation switching has also been addressed in [6] and [7] using dynamic programming. However, the possible use of different velocities for each agent was not addressed. But the possibility of slowing down some of the agents might, as we will show in an example, achieve better solutions while avoiding collision between agents. We propose a dynamic programming approach to solve the problem of formation switching with collision avoidance and agents velocities selection, that is the problem of deciding which agent moves to which place in the next formation guaranteeing that at any time the distance between any two of them is at least some predefined value. In addition, each agent can also explore the possibility of modifying its velocity to avoid collision, which is a main distinguishing feature from previous work. The formation switching performance is given by the time required for all agents to reach their new position, which is given by the maximum traveling time amongst individual agent traveling times. Since we want to minimize the time required for all agents to reach their new position, we have to solve a minmax problem. However, the methodology we propose can be used with any separable performance function. The problem addressed here should be seen as a component of a framework for multiagent coordination, incorporating also the trajectory control component [8], that allows to maintain or change formation while following a specified path in order to perform cooperative tasks.

This paper is organized as follows. In the next section, the problem of optimal reorganization of agent formations with collision avoidance is described and formally defined. In Section 3, a dynamic programming formulation of the problem is given and discussed. In Section 4, we discuss computational implementation issues of the dynamic programming algorithm, namely an efficient implementation of the main recursion as well as efficient data representations. A detailed description of the algorithms is also provided. Next, an example is reported to show the solution modifications when using velocities selection and collision avoidance. Some conclusions are drawn in the final section.

2 The Problem

In our problem a team of N identical agents has to switch from their current formation to some other formation (i.e agents have a specific goal configuration not related to the positions of the others), possibly unstructured, with collision avoidance. To address collision avoidance, we impose that the trajectories of the agents

must satisfy the separation constraint that at any time the distance between any two of them is at least ε, for some positive ε. The optimal (joint) trajectories are the ones that minimize the maximum trajectory time of individual agents.

Our approach can be used either centralized or decentralized, depending on the agents capabilities. In the latter case, all the agents would have to run the algorithm, which outputs an optimal solution, always the same if many exist, since the proposed method is deterministic.

Regarding the new formation, it can be either a pre-specified formation or a formation to be defined according to the information collected by the agents. In both cases, we do a pre-processing analysis that allows us to come up with the desired locations for the next formation.

This problem can be restated as the problem of allocating to each new position exactly one of the agents, located in the old positions, and determine each agent velocity. From all the possible solutions we are only interested in the ones where agent collision is prevented. Among these, we want to find one that minimizes the time required for all agents to move to the target positions, that is an allocation which has the least maximum individual agent traveling time.

To formally define the problem, consider a set of N agents moving in a space $\mathbb{R}^d$, so that at time t, agent i has position $q_i(t)$ in $\mathbb{R}^d$ (we will refer to $q_i(t) = (x_i(t), y_i(t))$ when our space is the plane $\mathbb{R}^2$). The position of all agents is defined by the N-tuple $Q(t) = [q_i(t)]_{i=1}^N$ in $\mathbb{R}^{d\times N}$. We assume that each agent is holonomic and that we are able to choose its velocity, so that its kinematic model is a simple integrator

$$\dot{q}_i(t) = \vartheta_i(t) \quad a.e.\ t \in \mathbb{R}^+.$$

The initial positions at time $t = 0$ are known and given by $A = [a_i]_{i=1}^N = Q(0)$. Suppose a set of M (with $M \geq N$) final positions in $\mathbb{R}^d$ is specified as $F = \{f_1, f_2, \ldots, f_M\}$.

The problem is to find an assignment between the N agents and N final positions in F. That is, we want to find a N-tuple $B = [b_i]_{i=1}^N$ of different elements of F, such that at some time $T > 0$, $Q(T) = B$ and all $b_i \in F$, with $b_i \neq b_k$. There are $\binom{M}{N} \cdot N!$ such N-tuples (the permutations of a set of N elements chosen from a set of M elements) and we want to find a procedure to choose an N-tuple minimizing a certain criterion that is more efficient than total enumeration.

The criterion to be minimized can be very general since the procedure developed is based on dynamic programming which is able to deal with general cost functions. Examples can be minimizing the total distance traveled by the agents

$$\text{Minimize} \sum_{i=1}^{N} \|b_i - a_i\|,$$

the total traveling time

$$\text{Minimize} \sum_{i=1}^{N} \|b_i - a_i\|/\|\vartheta_i\|,$$

or the maximum traveling time

$$\text{Minimize} \max_{i=1,\ldots,N} \|b_i - a_i\|/\|\vartheta_i\|.$$

We are also interested in selecting the traveling velocities of each agent. Assuming constant velocities, these are given by

$$\vartheta_i(t) = \vartheta_i = v_i \frac{b_i - a_i}{\|b_i - a_i\|},$$

where the constant speeds are selected from a discrete set $\Upsilon = \{V_{min}, \ldots, V_{max}\}$.

Moreover, we are also interested in avoiding collision between agents. We say that two agents i, k (with $i \neq k$), do not collide if their trajectories maintain a certain distance apart, at least ε, at all times. The non-collision conditions is

$$\|q_i(t) - q_k(t)\| \geq \varepsilon \qquad \forall t \in [0,T] \tag{1}$$

where the trajectory is given by

$$q_i(t) = a_i + \vartheta_i(t)t, \quad t \in [0,T].$$

We can then define a logic-valued function c as

$$c(a_i, b_i, v_i, a_k, b_k, v_k) = \begin{cases} 1 \text{ if collision between } i \text{ and } k \text{ ocurs} \\ 0 \text{ otherwise} \end{cases}$$

With these considerations, the problem (in the case of minimizing the maximum traveling time) can be formulated as follows

$$\min_{b_1,\ldots,b_N,v_1,\ldots,v_N} \max_{i=1,\ldots,N} \|b_i - a_i\|/v_i,$$

Subject to

$$\begin{aligned} & b_i \in F && \forall i, \\ & b_i \neq b_k && \forall i,k \text{ with } i \neq k, \\ & v_i \in \Upsilon, && \forall i, \\ & c(a_i, b_i, v_i, a_k, b_k, v_k) = 0, && \forall i,k \text{ with } i \neq k, \end{aligned}$$

Instead of using the set F of d-tuples, we can define a set $J = \{1,2,\ldots,M\}$ of indexes to such d-tuples, and also a set $I = \{1,2,\ldots,M\}$ of indexes to the agents. Let j_i in J be the target position for agent i, that is $b_i = f_{j_i}$. Define also the distances $d_{ij} = \|f_j - a_i\|$ which can be pre-computed for all $i \in I$ and $j \in J$. Redefining, without changing the notation, the function c to take as arguments the indexes to the agent positions instead of the positions (i.e. $c(a_i, f_{j_i}, v_i, a_k, f_{j_k}, v_k)$ is simply represented as $c(i, j_i, v_i, k, j_k, v_k)$), the problem can be reformulated into the form

$$\min_{j_1,\ldots,j_N,v_1,\ldots,v_N} \max_{i=1,\ldots,N} d_{ij}/v_i,$$

Subject to

$$\begin{aligned} & j_i \in J && \forall i \in I, \\ & j_i \neq j_k && \forall i,k \in I \text{ with } i \neq k, \\ & v_i \in \Upsilon, && \forall i \in I, \\ & c(i,j_i,v_i,a_k,j_k,v_k) = 0, && \forall i,k \text{ with } i \neq k, \end{aligned}$$

3 Dynamic programming formulation

Dynamic Programming (DP) is an effective method to solve combinatorial problems of a sequential nature. It provides a framework for decomposing an optimization problem into a nested family of subproblems. This nested structure suggests a recursive approach for solving the original problem using the solution to some subproblems. The recursion expresses an intuitive *principle of optimality* [2] for sequential decision processes; that is, once we have reached a particular state, a necessary condition for optimality is that the remaining decisions must be chosen optimally with respect to that state.

3.1 Derivation of the dynamic programming recursion: the simplest problem

We start by deriving a DP formulation for a simplified version of problem: where collision is not considered and different velocities are not selected. The collision avoidance and the selection of velocities for each agent are introduced later.

Consider that there are N agents $i = 1,2,\ldots,N$ to be relocated from known initial location coordinates to a target locations indexed by set J. We want to allocate exactly one of the agents to each position in the new formation. In our model a stage i contains all states S such that $|S| \geq i$, meaning that i agents have been allocated to the targets in S. The DP model has N stages, with a transition occurring from a stage $i-1$ to a stage i, when a decision is made about the allocation of agent i.

Define $f(i,S)$ to be the value of the best allocation of agents $1,2,\ldots,i$ to i targets in set S, that is, the allocation requiring the least maximum time the agents take to go to their new positions. Such value is found by determining the least maximum agent traveling time between its current position and its target position. For each agent, i, the traveling time to the target position j is given by d_{ij}/v_i. By the previous definition, the minimum traveling time of the $i-1$ agents to the target positions in set $S \setminus \{j\}$ is given by $f(i-1, S \setminus \{j\})$. From the above, the minimum traveling time of all i agents to the target positions in S they are assigned to, given that agent i travels at velocity v_i, without agent collisions, is obtained by examining all possible target locations $j \in S$ (see Fig. 1).

The *dynamic programming recursion* is then defined as

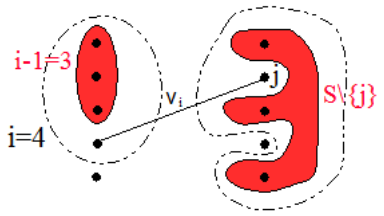

Fig. 1 Dynamic Programming Recursion for an example with $N = 5$ and stage $i = 4$.

$$f(i,S) = \min_{j \in S} \left\{ d_{ij}/v_i \vee f(i-1, S \setminus \{j\}) \right\}, \tag{2}$$

where $X \vee Y$ denotes the maximum between X and Y.

The *initial conditions* for the above recursion are provided by

$$f(1,S) = \min_{j \in S} \left\{ d_{1j}/v_1 \right\}, \quad \forall S \subseteq J, \tag{3}$$

and all other states are initialized as not yet computed.

Hence, the optimal value for the performance measure, that is the minimum traveling time needed for all N agents to assume their new positions in J, is given by

$$f(N,J). \tag{4}$$

3.2 Considering collision avoidance and velocities selection

Recall function c for which $c(i,j,v_i,a,b,v_a)$ takes value 1 if there is collision between pair of agents i and a traveling to positions j and b with velocities v_i and v_a, respectively, and takes value 0 otherwise. To analyze if the agent traveling through a newly defined trajectory collides with any agent traveling through previously determined trajectories, we define a recursive function. This function checks the satisfaction of the collision condition, given by equation (1), in turn, between the agent which had the trajectory defined last and each of the agents for which trajectory decisions have already been made. We note that by trajectory we understand not only the path between the initial and final positions but also a timing law and an implicitly defined velocity.

Consider that we are in state (i,S) and that we are assigning agent i to target j. Further let v_{i-1} be the traveling velocity for agent $i-1$. Since we are solving state (i,S) we need state $(i-1, S \setminus \{j\})$, which has already been computed. (If this is not the case, then we must compute it first.) In order to find out if this new assignment is possible, we need to check if at any point in time agent i, traveling with veloc-

ity v_i, will collide with any of the agents $1,2,\ldots,i-1$ for which we have already determined the target assignment and traveling velocities.

Let us define a recursive function $\mathscr{C}(i,v_i,j,k,V,S)$ that assumes the value one if a collision occurs between agent i traveling with velocity v_i to j and any of the agents $1,2,\ldots k$, with $k<i$, traveling to their targets, in set S, with their respective velocities $V=[v_1 v_2 \ldots v_k]$ and assumes the value zero if no such collisions occurs. This function works in the following way (see Fig. 2):

1. first it verifies $c(i,v_i,j,k,v_k,\mathscr{B}_j)$, that is, it verifies if there is collision between trajectory $i \longrightarrow j$ at velocity v_i and trajectory $k \longrightarrow \mathscr{B}_j$ at velocity v_k, where $\mathscr{B}_j$ is the optimal target for agent k when targets in set $S\setminus\{j\}$ are available for agents $1,2,\ldots,k$. If this is the case it returns the value 1.

2. Otherwise, if they do not collide, it verifies if trajectory $i \longrightarrow j$ at velocity v_i collides with any of the remaining agents. That is, it calls the collision function $\mathscr{C}(i,v_i,j,k-1,V',S')$, where $S'=S\setminus\{\mathscr{B}_j\}$ and $V=[V' v_k]$

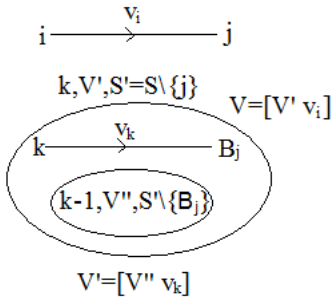

Fig. 2 Collision recursion.

The collision recursion is therefore written as:

$$\mathscr{C}(i,v_i,j,k,V,S)=\left\{c(i,v_i,j,k,v_k,\mathscr{B}_j)\vee\mathscr{C}(i,v_i,j,k-1,V',S')\right\} \tag{5}$$

where $\mathscr{B}_j=Best_j(k,V',S')$, $V=[V'\ v_k]$, $S'=S\setminus\{j\}$

The initial conditions for recursion (5) are provided by

$$\mathscr{C}(i,v_i,j,1,v_1,\{k\})=\{c(i,v_i,j,1,v_1,k)\},$$

$\forall i\in I;\forall j,k\in J$ with $j\neq k;\forall v_i,v_1\in\Upsilon$. All other states are initialized as not yet computed.

The dynamic programming recursion for the minimal time switching problem with collision avoidance and velocities selection is then

$$f(i,V,S) = \min_{j \in S} \left\{ d(i,j)/v_i \vee f(i-1,V',S') \vee M * \mathscr{C}(i,v_i,j,i-1,V',S') \right\}.$$

where $V = [V' v_i]$, $S' = S \setminus \{j\}$, and $\mathscr{C}$ is the collision function.

The initial conditions are given by

$$f(1,v_1,\{j\}) = \{d(1,j)/v_1\}, \forall j \in J \text{ and } \forall v_1 \in \Upsilon.$$

All other states being initialized as not computed (∞).

To determine the optimal value for our problem we have compute

$$\min_{\text{all N-tuples V}} f(N,V,J)$$

.

4 Computational implementation

The DP procedure we have implemented exploits the recursive nature of the DP formulation by using a backward-forward procedure. Although a pure forward Dynamic Programming (DP) algorithm can be easily derived from the DP recursion, equations (2) and (6), such implementation would result in considerable waste of computational effort since, generally, complete computation of the state space is not required. Furthermore, since the computation of a state requires information contained in other states, rapid access to state information should be seek.

The main advantage of the backward-forward procedure implemented is that the exploration of the state space graph, i.e. the solution space, is based upon the part of the graph which has already been explored. Thus, states which are not feasible for the problem are not computed, since only states which are needed for the computation of a solution are considered. The algorithm is dynamic as it detects the needs of the particular problem and behaves accordingly.

States at stage 1 are either nonexistent or initialized as given in equation (6). The DP recursion, equation (2), is then implemented in a backward-forward recursive way. It starts from the final states (N,V,J) and while moving backward visits, without computing, possible states until a state already computed is reached. Initially, only states in stage 1, initialized by equation (6) are already computed. Then, the procedure is performed in reverse order, i.e. starting from the state last identified in the backward process, it goes forward through computed states until a state (i,V',S') is found which has not yet been computed. At this point, again it goes backward until a computed state is reached. This procedure is repeated until the final states (N,V,J) for all V are reached with a value that cannot be improved by any other alternative solution. From these we choose the minimum one. The main advantage of this backward-forward recursive algorithm is that only intermediate states needed are visited and from these only the feasible ones that may yield a better solution are computed.

As said before, due to the recursive nature of equation (2), state computation implies frequent access to other states. Recall that a state is represented by a number, a sequence, and a set. Therefore, sequence operations like adding or removing an element and set operations like searching, deletion, and insertion of a set element must be performed efficiently.

Sequence representation and operation:

A sequence is a n-tuple with k possible values for each element, where n is the number of agents and k the number of possible velocity values. Therefore, there are k^n possible sequences to be represented. If sequences are represented by integers in the range $0 \sim k^n - 1$ then it is easy to implement sequence operations such as partitions. Thus, we represent a sequence as a numeral with n digits in the base k. The partition of a sequence with l digits that we are interested on is the one corresponding to the first $l-1$ digits and the last digit. Such a partition can be obtained by performing the integer division in the base k and taking the remainder of such division.

Example: Consider a sequence of length $n = 4$ with $k = 3$ possible values v_0, v_1, and v_2. This is represented by numeral with n digits in the base k as

$$[v_1, v_0, v_2, v_1] \text{ is represented by} 1\ 0\ 2\ 1_3 = 1 \cdot 3^3 + 0 \cdot 3^2 + 2 \cdot 3^1 + 1 \cdot 3^0 = 34$$

Partition of this sequence by the last element can be performed by integer division in the base k and taking the remainder of such division,

$$V = 1\ 0\ 2\ 1_3 = 34 \text{ can be split into } [V'\ v_i] \text{ as follows}$$

$$V' = 1\ 0\ 2_3 = 1 \cdot 3^2 + 2 \cdot 3^0 = 11 = 34 \text{ DIV } 3$$

and

$$v_i = 1_3 = 1 = 34 \text{ MOD } 3$$

Set representation and operation:

A computationally efficient way of storing and operating sets is the bit-vector representation, also called the boolean array, whereby a *computer word* is used to keep the information related to the elements of the set. In this representation a universal set $U = \{1, 2, \ldots, n\}$ is considered. Any subset of U can be represented by a binary string (a computer word) of length n in which the ith bit is set to 1 if i is an element of the set, and set to 0 otherwise. So, there is a one-to-one correspondence between all possible subsets of U (in total 2^n) and all binary strings of length n. Since there is also a one-to-one correspondence between binary strings and integers, the sets can be efficiently stored and worked out simply as integer numbers. A major advantage of such implementation is that the set operations, *location*, *insertion* or *deletion* of

a set element can be performed by directly addressing the appropriate bit. For a detailed discussion of this representation of sets see, for example, the book by Aho *et al.* [1].

Example: Consider the Universal set U=$\{1,2,3,4\}$ of $n = 4$ elements. This set and any of its susbsets can be represented by a binary string of length 4, or equivalently its representation as an integer in the range $0 \sim 15$.

$$U = \{1,2,3,4\} \text{ is represented by } 1111_B = 15.$$

$$\text{A subset } A = \{1,3\} \text{ is represented by } 0101_B = 5$$

The flow of the algorithm is managed by Algorithm 1, which starts by labeling all states (subproblems) as not yet computed, that is, it assigns to them a ∞ value. Then, it initializes states in stage 1, that is subproblems involving 1 agent, as given by equation (6). After that, it calls algorithms 2 with parameters (N,V,J). Algorithm 2, that implements recursion (2), calls algorithm 3 to check for collisions every time it attempts to define one more agent-target allocation. This algorithm is used to find out whether the newly established allocation satisfies the collision regarding all previously defined allocations or not, feeding the result back to algorithm 2. Algorithm 4, called after Algorithm 2 has finished,also implements a recursive function with which the solution structure, i.e. agent-target allocation, is retrieved.

Algorithm 1: DP for finding agent-target allocations and corresponding velocities.

Input: The agent set, locations and velocities, the target set and locations, and the distance function;

Compute the distance for every pair agent-target (d_{ij});

Label all states as not yet computed;
$f(n,V,S) = \infty$;
for all $n = 1,2,\ldots,N$, all V with n components, $S \in J$;

Initialize states at stage one as
$f(1,V,\{j\}) = \{d_{1j}/v_1\}, \quad \forall V \in \Upsilon, j \in J.$

Call *Compute*(N,V,J) for all sequences V with N components;
Output: Solution performance;

Call *Allocation*$(N,V*,J)$;
Output: Agent targets and velocities;

Algorithm 2 is a recursive algorithm that computes the optimal solution cost, i.e. it implements equation (2) . This function receives three arguments: the agents to be allocated, their respective velocity values, and the set of target locations available to them, all represented by integer numbers. It starts by checking whether the specific state (i,V,S) has already been computed or not. If so, the program returns to the point where the function was called, otherwise the state is computed. To compute state (i,V,S), all possible target locations $j \in S$ that might lead to a better subproblem solution are identified. The function is then called with arguments $(i-1,V',S')$, where $V' = V\,DIV\,nvel$ (subsequence of v containing the first $i-1$ elements and $S' = S \setminus \{j\}$, for every j such that allocating agent i to target j does not lead to any collision with previously defined allocations. This condition is verified by algorithm 3.

Algorithm 2: Recursive function: compute optimal performance.

Recursive $Compute(i,V,S)$;
if $f(i,V,S) \neq \infty$ **then**
  return $f(i,V,S)$ to caller;
end
Set $min = \infty$;
for *each* $j \in S'$ **do**
  $S' = S \setminus \{j\}$; $V' = V\,DIV\,nvel$; $v_i = V\,MOD\,nvel$;
  Call $Collision(i, v_i, j, i-1, V', S')$
  if $Col(i,j,i-1,S') = 0$ **then**
    Call $Compute(i-1,V',S')$;
    $t_{ij} = d_{ij}/v_i$;
    $aux = max(f(i-1,V',S'), t_{ij})$;
    if $aux \leq min$ **then**
      $min = aux$; $best_j = j$;
    end
  end
end
Store information: *target* $\mathscr{B}_j(i,V,S) = best_j$; *value* $f(i,V,S) = min$;
Return: $f(i,V,S)$;

Algorithm 3 is a recursive algorithm that checks the collision of a specific agent-target allocation traveling at a specific velocity with the set of allocations and velocities previously established, i.e. it implements equation (5) . This function receives six arguments: the newly defined agent-target allocation $i \longrightarrow j$ and its traveling velocity v_i and the previously defined allocations and respective velocities to check with, that is agents $1,2,\ldots,k$, their velocities and their target locations S. It starts by checking the collision condition, given by equation (1), for the allocation pair $i \longrightarrow j$ traveling at velocity v_i and $k \longrightarrow \mathscr{B}_j$ traveling at velocity v_k, where $\mathscr{B}_j$ is the optimal target for agent k when agents $1,2,\ldots,k$ are allocated to targets in S. If there is collision it returns 1; otherwise it calls itself with arguments $(i, v_i, j, k-1, V', S \setminus \{\mathscr{B}_j\})$.

Algorithm 3: Recursive function: find if the trajectory of the allocation $i \rightarrow j$ at velocity v_i collides with any of the existing allocations to the targets in S at the specified velocities in V.

Recursive $Collision(i, v_i, j, k, V, S)$;

if $Col\,(i, v_i, j, k, V, S) \neq \infty$ **then**
  return $Col\,(i, v_i, j, k, V, S)$ to caller;
end
$B_j = \mathscr{B}_j(k, V, S)$;
if *collision condition is not satisfied* **then**
  $Col\,(i, v_i, j, k, V, S) = 1$;
  return $Col\,(i, v_i, j, k, V, S)$ to caller;
end
$S' = S \setminus \{B_j\}$;
$V' = V\ DIV\ nvel$;
$v_k = V\ MOD\ nvel$;
Call $Collision(i, v_i, j, k-1, V', S')$;

Store information: $Col\,(i, v_i, j, k, V, S) = 0$;

Return: $Col\,(i, v_i, j, k, V, S)$;

Algorithm 4 is also a recursive algorithm and it backtracks through the information stored while solving subproblems, in order to retrieve the solution structure, i.e. the actual agent-target allocation and agent velocity. This algorithm works backward from the final state (N, V^*, J), corresponding to the optimal solution obtained, and finds the partition by looking at the agent traveling velocity $v_N = V^*\ MOD\ nvel$ and at the target stored for this state $\mathscr{B}_j(N, V^*, J)$, with which it can build the structure of the solution found. Algorithm 3 receives three arguments: the agents, their traveling velocity, and the set of target locations. It starts by checking whether the agent current locations set is empty. If so, the program returns to the point where the function was called; Otherwise the backtrack information of the state is retrieved and the other needed states evaluated.

Algorithm 4: Recursive function: retrieve agent-target allocation and agents velocity.

Recursive $Allocation(i,V,S)$;

if $S \neq \varnothing$ **then**
  $v_i = VMODmvel$;
  $j = target\mathscr{B}_j(i,V,S)$;
  $Vloc(i) = v_i$;
  $Alloc(i) = j$;
  $V' = VDIVnvel$;
  $S' = S \setminus \{j\}$;
  CALL $Allocation(i-1,V',S')$;
end

Return: $Alloc$;

5 An Example

An example is given to show how agent-target allocations are influenced by imposing that no collisions are allowed both with a single fixed velocity value for all agents and with the choice of agent velocities from 3 different possible values. In this example we have decided to use d_{ij} as the Euclidian distance although any other distance measure may have been used.

The separation constraints impose, at any point in time, the distance between any two agent trajectories to be at least 15 points; otherwise it is considered that those two agents collide.

Consider 4 agents, A, B, C, and D with random initial positions as given in Table 1 and four target positions 1, 2, 3, and 4 in a diamond formation as given in Table 2.

	Location	
	x_i	y_i
Agent A	35	185
Agent B	183	64
Agent C	348	349
Agent D	30	200

Table 1 Agents random initial location.

In Fig. 3 we give the graphical representation of the optimal agent-target allocation found, when a single velocity value is considered and collisions are allowed and no collisions are allowed, respectively.

	Location	
	x_i	y_i
Target 1	95	258
Target 2	294	258
Target 3	195	169
Target 4	195	347

Table 2 Target locations, in diamond formation.

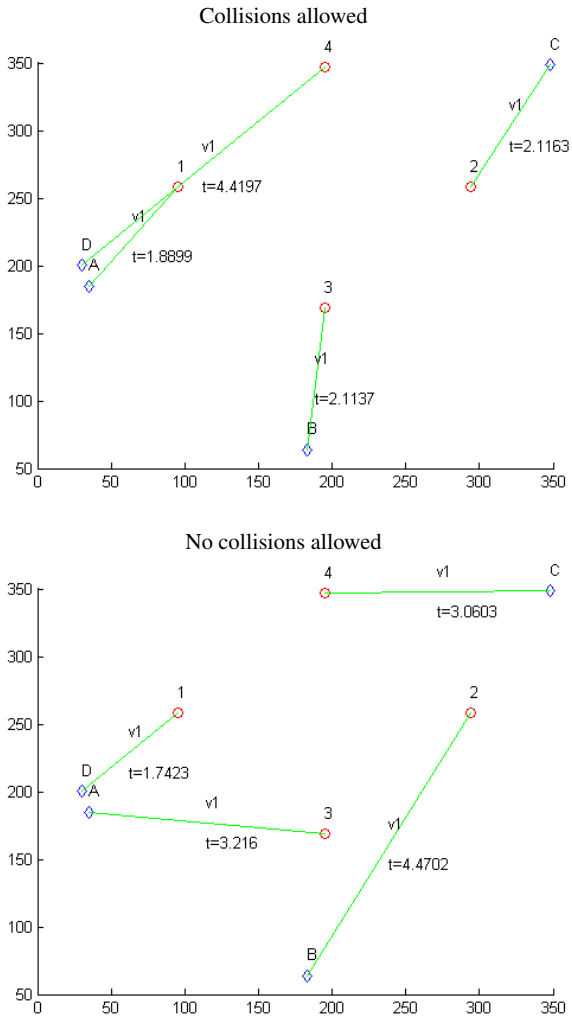

Fig. 3 Comparison of solutions with and without collision for the single velocity case.

As it can be seen in the top part of the Fig. 3, i.e. when collisions are allowed, the trajectory of agents A and D do not remain apart, by 15 points, at all times. Therefore, when no collisions are enforced the agent-target allocation changes with an increase in the time that it takes for all agents to assume their new positions.

In Fig. 4 we give the graphical representation of the optimal agent-target allocation found, when there are 3 possible velocity values to choose from and collisions are allowed and no collisions are allowed, respectively.

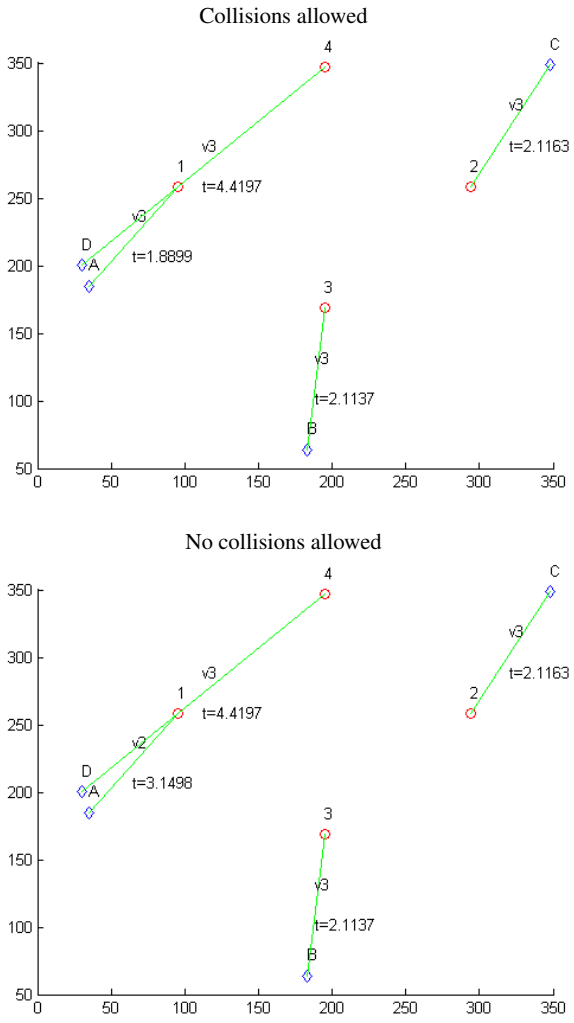

Fig. 4 Comparison of the solutions with and without collision for the velocity choice case.

As it can be seen in the top part of the Fig.4, i.e. when collisions are allowed, the trajectory of agents A and D do not remain apart, by 15 points, at all times, since the agents move at the same velocity. Therefore, when no collisions are enforced although the agent-target allocation remains the same, agent A has it velocity decreased and therefore its trajectory no longer collides with the trajectory of agent D. Furthermore, since agents A trajectory is smaller this can be done with no increase in the time that it takes for all agents to assume their new positions.

6 Conclusion

We have developed an optimization algorithm to decide how to reorganize a formation of vehicles into another formation of different shape with collision avoidance and agent traveling velocity choice, which is a relevant problem in cooperative control applications. The method proposed here should be seen as a component of a framework for multiagent coordination/cooperation, which must necessarily include other components such as a trajectory control component.

The algorithm proposed is based on a dynamic programming approach that is very efficient for small dimensional problems. As explained before, the original problem is solved by combining, in an efficient way, the solution to some subproblems. The method efficiency improves with the number of times the subproblems are reused, which obviously increases with the number of feasible solutions.

Moreover, the proposed methodology is very flexible, in the sense that it easily allows for the inclusion of additional problem features, e.g. imposing geometric constraints on each agent or on the formation as a whole, using nonlinear trajectories, among others.

References

1. A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, 1983.
2. R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, USA, 1957.
3. JG Bender. An overview of systems studies of automated highway systems. *IEEE Transactions on Vehicular Technology*, 40(1 Part 2):82–99, 1991.
4. L. E. Buzogany, M. Pachter, and J. J. d'Azzo. Automated control of aircraft in formation flight. pages 1349–1370, Monterey, California, 1993. AIAA Guidance, Navigation, and Control Conference and Exhibit.
5. J. P. Desai, P. Ostrowski, and V. Kumar. Modeling and control of formations of nonholonomic mobile robots. *IEEE Transactions on Robotics and Automation*, 17(6):905–908, 2001.
6. D.B.M.M. Fontes and F.A.C.C. Fontes. Optimal reorganization of agent formations. *WSEAS Transactions on Systems and Control*, 3(9):789–798, 2008.
7. D.B.M.M. Fontes and F.A.C.C. Fontes. Minimal Switching Time of Agent Formations with Collision Avoidance. *Springer Optimization and Its Applications*, 40:305–321, 2010.
8. F. A. C. C. Fontes, D. B. M. M. Fontes, and A. C. D. Caldeira. Model predictive control of vehicle formations. In P. Pardalos M.J. Hirsch, C.W. Commander and R. Murphey, editors,

Optimization and Cooperative Control Strategies, Lecture Notes in Control and Information Sciences , Vol. 381 ISBN: 978-3-540-88062-2. Springer Verlag, Berlin, 2009.
9. J. Hu and S. Sastry. Optimal collision avoidance and formation switching on Riemannian manifolds. In *IEEE CONFERENCE ON DECISION AND CONTROL*, volume 2, pages 1071–1076. IEEE; 1998, 2001.
10. Z. Jin, T. Shima, and C. J. Schumacher. Optimal scheduling for refueling multiple autonomous aerial vehicles. *IEEE Transactions on Robotics*, 22(4):682–693, 2006.
11. R. M. Murray. Recent research in cooperative control multivehicle systems. *Journal of Dynamic Systems, Measurement and Control*, 129:571–583, 2007.
12. S. J. Rasmussen and T. Shima. Branch and bound tree search for assigning cooperating UVAs to multiple tasks. Minneapolis, Minnesota, USA, 2006. Institute of Electrical and Electronic Engineers, American Control Conference 2006.
13. S. J. Rasmussen, T. Shima, J. W. Mitchell, A. Sparks, and P. R. Chandler. State-space search for improved autonomous UAVs assignment algorithm. Paradise Island, Bahamas, 2004. IEEE Conference on Decision and Control.
14. C. J. Schumacher, P. R. Chandler, and S. J. Rasmussen. Task allocation for wide area search munitions via iterative network flow. Reston, Virginia, USA, 2002. American Institute of Aeronautics and Astronautics, Guidance, Navigation, and Control Conference 2002.
15. C. J. Schumacher, P. R. Chandler, and S. J. Rasmussen. Task allocation for wide area search munitions with variable path length. New York, New York, USA, 2003. Institute of Electrical and Electronic Engineers, American Control Conference 2003.
16. T.R. Smith, H. Hanssmann, and N.E. Leonard. Orientation control of multiple underwater vehicles with symmetry-breaking potentials. In *IEEE Conference on Decision and Control*, volume 5, pages 4598–4603, 2001.
17. D. Swaroop and JK Hedrick. Constant spacing strategies for platooning in automated highway systems. *Journal of dynamic systems, measurement, and control*, 121:462, 1999.
18. J. D. Wolfe, D. F. Chichka, and J. L. Speyer. Decetntralized controllers for unmanned aerial vehicle formation flight. pages 96–3833, San Diego, California, 1996. AIAA Guidance, Navigation, and Control Conference and Exhibit.
19. M. Yamagishi. Social rules for reactive formation switching. Technical Report UWEETR-2004-0025, Department of Electrical Engineering, University of Washington, Seattle, Washington, USA, 2004.
20. H. Yamaguchi. A cooperative hunting behavior by mobile-robot troops. *The International Journal of Robotics Research*, 18(9):931, 1999.
21. H. Yamaguchi, T. Arai, and G. Beni. A distributed control scheme for multiple robotic vehicles to make group formations. *Robotics and Autonomous systems*, 36(4):125–147, 2001.