2014 IEEE International Conference on Robotics & Automation (ICRA)
Hong Kong Convention and Exhibition Center
May 31 - June 7, 2014. Hong Kong, China

Path Planning for Coherent and Persistent Groups

Tianyu Huang[1] Mubbasir Kapadia[2] Norman I. Badler[3] Marcelo Kallmann[4]

Abstract— **This paper addresses the problem of group path planning while maintaining group coherence and persistence. Group coherence ensures that a group minimizes both longitudinal and lateral dispersion, and is achieved with the introduction of a deformation penalty to the cost formulation. When the deformation penalty is significantly high, a group may split and later merge. Group persistence is modeled by introducing split and merge actions in the action space, and adding a split penalty to the cost measure. We formulate the problem domain (state, action space, and cost formulation), present our path planning approach for coherent and persistent groups, and provide empirical results demonstrating the capabilities of our method on a variety of challenging scenarios.**

I. INTRODUCTION

Global navigation for autonomous agents in complex environments is well studied, with many proposed solutions. However, path planning for groups is still an active research with many open problems that are yet to be addressed. Agents within a group share a common target and must try to stay together by satisfying constraints on lateral and longitudinal dispersion, thus maintaining *group coherence*. Additionally, a group must remain *persistent* unless the environment demands group splitting and reformation. These dispersion constraints and the ability to split and reform need to be modeled at the global planning level, producing an optimal navigation strategy that minimizes distance traveled, group deformation, and split penalty.

This paper presents a path planning approach for coherent and persistent groups in arbitrarily complex environments. The navigable regions in the environment are represented using a triangulated navigation mesh [1] with precomputed local clearance information. A group is represented as a shape constrained area, which incurs a deformation penalty when it deviates from its rest shape, and can split (and reform) in necessary situations. The group action space is extended to include split and merge actions, and a deformation cost and a split penalty are introduced into the cost formulation of the search. The cost due to deformation is modeled as the lateral and longitudinal dispersion of the group from its rest shape, while split penalty is computed using the current split status of the group (number of splits and distance between subgroups). As a result, the proposed method plans paths for coherent and persistent groups by optimizing a three-tuple cost vector (distance, deformation, and split cost).

*This work was supported by NSFC 61202243

[1]Tianyu Huang is with the School of Software, Beijing Institute of Technology, Beijing 100081, China huangtianyu at bit.edu.cn

[2]Mubbasir Kapadia is with the Center for Human Modeling and Simulation, University of Pennsylvania, Philadelphia, PA 19104, USA mubbasir at seas.upenn.edu

[3]Norman I. Badler is with the School of Engineering and Applied Science, University of Pennsylvania, Philadelphia, PA 19104, USA Badler at seas.upenn.edu

[4]Marcelo Kallmann is with the School of Engineering, University of California, Merced, CA 95343, USA mkallmann at ucmerced.edu

II. RELATED WORK

Traditional reactive approaches to simulate group flocking behavior [2], [3] produce natural results in open areas, but are limited to locally optimal decisions in cluttered environments. Probabilistic roadmaps (PRMs) [4], [5] are widely used in robotics, with extensions developed for multiple entities [6]. Potential field based methods [7] and continuum approaches [8] produce local collision avoidance behavior in large crowds, but have not addressed enforcement of group constraints. The work in [9] uses social potential fields for group formation in robots, with additional work [10] focusing on predefined group forms. Discrete search methods [11] can produce optimal paths for an underlying environment representation and can incorporate additional problem-specific constraints. However, centralized approaches don't scale well as the number of agents increases, while decentralized techniques [12], [13] have not addressed coherent and persistent groups.

Crowd density information [14] helps characters avoid congested routes that could lead to traffic jams. A deformable mesh [15] has also been used to split and merge a crowd at cross points. Our work does not rely on density and instead introduces the concept of controlling groups according to coherence and persistence constraints. The work in [16], [17] presents an approach for group motion planning using the concept of a backbone path with clearance constraints, while group coherence is enforced using local planning. The work in [18] extends RVO to maintain team coherence while guaranteeing collision-free motions and the work in [19] uses a multi-level graph that allows group members to switch formations and reach their goals. Neither of the two approaches focuses on group merging or splitting. The work in [20] manages group motion in dynamic environment using a centralized approach for group coherence and formation, but is computationally infeasible for a large number of groups. The explicit corridor map (ECM) [14], [21], [22] produces shortest paths in the medial axis of an environment, ensuring clearance constraints, and can account for crowd density by periodically replanning to avoid congestion.

Comparison to prior work. While some of these works have proposed heuristic strategies for group coherence, our work enforces the shape of the group at the global planning layer and is the first to introduce a persistence constraint that causes groups to split and merge based on the extent

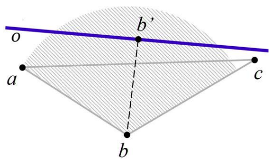


Fig. 1. Illustration of traversal τ_{abc}. If constrained edge o is the closest constraint crossing traversal sector, $\mathbf{cl}(a,b,c) = \mathbf{dist}(b,o) = \mathbf{dist}(b,b')$.

of shape deformation imposed by the environment. Furthermore, contrary to prior work on group path planning, we achieve control over group shapes by treating groups as deformable shapes instead of fixed-size disks or collections of individuals.

III. ENVIRONMENT REPRESENTATION

We represent navigable regions in the environment using a triangulation-based navigation mesh [1] with precomputed local clearance information in edge traversals, which enables the efficient computation of locally shortest paths with arbitrary clearance. We refer the readers to a detailed description in [1] and present a brief overview here.

Let $O = \{o_1, o_2, ..., o_n\}$ be the set of obstacle segments of an environment and let $T = \mathbf{CDT}(O)$ be its Constrained Delaunay Triangulation. Let π be a free path in T, and t be a triangle of the channel of π. τ_{abc} is the traversal of t if a, b, c are the vertices of t and the free path crosses t by entrance edge ab and exit edge bc. Given a traversal τ_{abc}, its sector clearance $\mathbf{cl}(a,b,c)$ is defined as the distance between the traversal corner b and the closest vertex or constrained edge intersecting the traversal sector. As shown in Fig. 1, $\mathbf{cl}(a,b,c) = \mathbf{dist}(b,o) = \mathbf{dist}(b,b')$, where b' is the orthogonal projection of b on o and $\mathbf{dist}$ denotes the Euclidean distance, if constrained edge o is the closest constraint crossing traversal sector τ_{abc}. A traversal τ_{abc} in T has local clearance if it does not have disturbances. A Local Clearance Triangulation $\mathbf{LCT}(O)$ is a $\mathbf{CDT}(O)$ with all traversals having local clearance.

Once the LCT of the planar environment is available, a single graph search can be performed over the adjacency graph of the triangulation in order to obtain a channel with enough clearance to connect the start and end points. The search continuously expands triangle traversals from the current lowest cost edge until the triangle containing the end point is found. To guarantee the desired path clearance, triangle traversals are only accepted if the respective traversal clearance is greater or equal to the needed clearance.

Our group search approach is developed in a given LCT graph. A group is informally defined as a set of agents, satisfying collective spatial and temporal constraints while trying to achieve a common goal. We model a group as a deformable and splittable area preserving shape. The efficiency of the group search is determined by three factors: path length, deformation minimization, and spitting minimization. The two new characteristics of our group path planning search are therefore to maintain group coherence and persistence. Group coherence is modeled by deformation cost

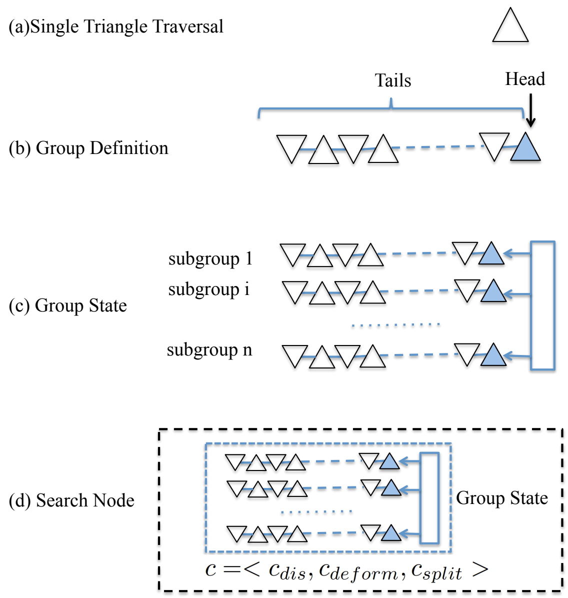


Fig. 2. Illustration of (a) traversal, (b) group, (c) group state comprising multiple subgroups, and (d) search node.

to preserve the group desired shape, while group persistence is modeled by determining split and merge conditions, and introducing the ability to plan paths for multiple subgroups within a single planning instance.

IV. PROPOSED PLANNER

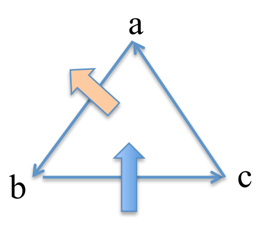
Our group path planning problem is formulated with:

$$\sum = \langle S, A, c(s,s'), h(s) \rangle, \tag{1}$$

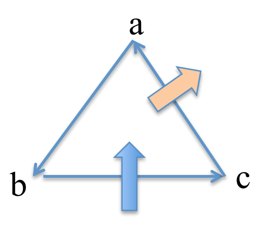
where S is the state space of a group, A is the set of possible actions, $c(s,s')$ is the cost of transitioning from s to s', where $s, s' \in S$, and $h(s)$ is the heuristic estimate of reaching the goal g.

The space occupied by a group is described by the collection of triangles containing it. The triangle containing the front of the group is referred to as the head traversal of the group, while the rest of the group is represented as a series of tail triangle traversals, as shown in Fig. 2(b). A group can additionally split into one or more subgroups, as shown in Fig. 2(c). A node in the search expansion tree comprises the state s of the group, and its associated cost. The action space $A = \{$`LEFT`,`RIGHT`,`SPLIT`,`PAUSE`, `MERGE`$\}$ describes the set of possible movement behaviors of each subgroup:

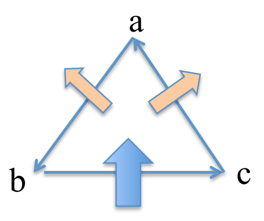
1) `LEFT`: Group traversal from edge bc to ba (Fig. 3(a)).
2) `RIGHT`: Group traversal from edge cb to ca (Fig. 3(b)).
3) `SPLIT`: Group arrives by edge bc, then splits in two subgroups that exit by edges ba and ca (Fig. 3(c)).
4) `PAUSE`: A subgroup stays in place in order to merge with another subgroup or when the cost associated with the other actions is prohibitive due to deformation or split penalties.
5) `MERGE`: Two subgroups entering from two distinct edges merge as one group that exits through the 3rd edge (Fig. 3(d)).



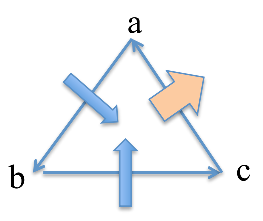
(a) Action LEFT



(b) Action RIGHT



(c) Action SPLIT



(d) Action MERGE

Fig. 3. Group actions. (a) LEFT, (b) RIGHT, (c) SPLIT, (d) MERGE. The blue arrow is the arrival direction of a group, and the orange arrow is the exit direction of group.

A. Action Evaluation Function

The cost function $c(s, s')$ is the cost of performing an action which transitions the group from s to s', and governs action selection. Our cost function is formulated as a three-tuple vector:

$$c(s, s') = \langle c_{dis}(s, s'), c_{def}(s, s'), c_{split}(s, s')\rangle, \tag{2}$$

where $c_{dis}(s, s')$ is the distance cost from state s to s', $c_{def}(s, s')$ is the deformation cost from state s to s', and $c_{split}(s, s')$ is the split penalty from state s to s'. The heuristic function $h(s)$ is an admissible estimate of the cost from the current state s to the goal state g, accounting for potential deformation:

$$h(s) = \langle h_{dis}(s), h_{def}(s), 0\rangle, \tag{3}$$

where $h_{dis}(s)$ computes the straight line distance to the goal and $h_{def}(s)$ estimates the deformation cost from the current state of the group to the goal state g and is computed as $c_{def}(s, g)$.

Let $f(s)$ be the cost estimate from the start state to the goal state, constrained to go through s. $f(s) = g(s) + h(s)$, where $g(s)$ is the accumulated cost of all traversed states from the start state to s, represented as a three-tuple $\langle g_{dis}, g_{def}, g_{split}\rangle$. Each component of $g(s)$ is defined as:

$$g_{dis}(s') = g_{dis}(s) + c_{dis}(s, s'), \tag{4}$$

$$g_{def}(s') = g_{def}(s) + c_{def}(s, s'), \tag{5}$$

$$g_{split}(s') = g_{split}(s) + c_{split}(s, s'). \tag{6}$$

The planner will generate a sequence of states, from the initial to the goal state, while optimizing the evaluation function:

$$f = w_1 \cdot f_{dis} + w_2 \cdot f_{def} + w_3 \cdot f_{split}, \tag{7}$$

where $f_{dis}, f_{def}, f_{split}$ accumulate their respective costs from the initial to the goal state. The weights $\{w_i | 1 \leq i \leq 3\}$, are normalized and can be varied for achieving different behaviors. f represents the summation of distance costs, deformation costs and split penalties. There is a split penalty only when new splits happen. Group splits are introduced to reduce the deformation cost in subsequent states at the expense of incurring a split penalty.

B. Cost Computation

Distance cost. Let **dist** $(gr_i,gr_i^{'})$ denote the Euclidean distance between the traversal exit edges of the heads of the subgroup states gr_i and $gr_i^{'}$. The distance between two traversal edges can be computed based on the center of the respective triangles, the center of the edges etc. We use the edge centers and incorporate a visibility criterion [1] to compute the distance because it guarantees that straight line solutions are not missed. The distance cost $c_{dis}(s, s')$ of transitioning between group states s and s' is then computed as the average distance between the subgroups at each state. For n subgroups in the current state s:

$$c_{dis}(s, s') = \frac{\sum_{i=1}^{i \leq n} \textbf{dist}(gr_i, gr_i^{'})}{n}. \tag{8}$$

The cost computation above has to handle special situations in the case of merges and splits. In case of a merge, the subgroups that merge will both be corresponded with the merged subgroup. In the case of a subgroup splitting, the two new subgroups are corresponded with the original one. This ensures consistent cost average values.

Deformation cost. Deformation cost measures group dispersion while passing through the environment corridors, leveraging the precomputed clearance values in the LCT triangulation. The deformation cost between two states is computed as:

$$c_{def}(s, s') = \sum_{i=1}^{i \leq n} \textbf{deform}(gr_i, gr_i^{'}), \tag{9}$$

where **deform** measures the deformation of each subgroup. Let n, m be the number of traversals in gr and gr' respectively. Let $\{cl_j | 1 \leq j \leq n\}$, $\{cl_k | 0 \leq k < m, m-1 \leq n\}$ be the sequence of traversal clearances of gr and gr' respectively, as illustrated in Fig. 4. These clearance values approximately capture the width of the corridor areas where the group is passing and are used to compute the deformation between the two subgroup states as follows:

$$\textbf{deform}(gr, gr') = \sqrt[m]{\prod_{k=0}^{m} \rho_k} \cdot \textbf{dist}(gr, gr'), \tag{10}$$

where

$$\rho_k = \begin{cases} cl_k / cl_{k+1} & \text{, if } cl_k < cl_{k+1}, k < n \\ 1 & \text{, otherwise.} \end{cases} \tag{11}$$

The above deformation definition needs to be computed for the head and tail traversals for each step, which is

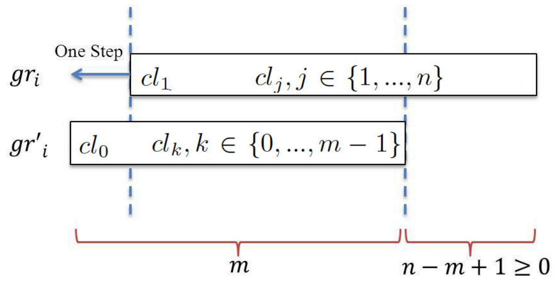


Fig. 4. Illustration of clearance sequences of current state of the group gr and next state of the group gr'.

time consuming. For the sake of computation efficiency, we consider only the head of each subgroup when computing the deformation cost. This works well in our experiments because the tail of a group is guaranteed to go through the head, thus the aggregate deformation of a group over a sequence of traversals can be computed as the net deformation of its head over the path:

$$\mathbf{deform}(gr, gr') = max\{0, (width - cl_0)/width\}, \quad (12)$$

where $width$ is the desired group width. If there is no splitting, it's the initial size of the group.Note that the longitudinal dispersion factor is ignored for the above reason. We use this cost formulation to compute the deformation heuristic $h_{def}(s) = c_{def}(s, g)$ as the estimated cost of deforming from s to g. This generates a plan where the group deforms from its initial rest shape to the desired goal shape, while minimizing net deformation (i.e., preserving group coherence).

$$h_{def}(s) = \sum_{i=1}^{i \leq n} \mathbf{deform}(gr_i, g). \quad (13)$$

Split penalty. The split penalty is the cost for the group to split, which is taken into account only when a new split happens. Penalizing splits favors group persistence as far as possible, where splits are introduced only when the cost due to deformation is too high. The split penalty is computed as a function of the number of subgroups and distances between subgroups. The split cost is based on a hierarchical distance measure where the distance between two splitting subgroups is computed as the Euclidean distance from the center of their heads.

Assume two subgroups gr_i and gr_j are generated as a result of a split, with $l - 1$ split ancestors where l is the current splitting level. The ancestor of a split group is the node before this split. The split penalty is computed as the aggregate distance between all subgroup pairs generated as a result of a split, weighted by the split level. δ is a scaling constant.

$$c_{split}(s, s') = \delta \sum_{k=1}^{k \leq l} k \cdot dist(gr_i, gr_j). \quad (14)$$

We can limit the maximum number of splits that can be generated by placing an upper bound on the split level. This aggregate measure of split penalty accounts for the relative split depth, and mitigates the need of a costly recursive computation during the search.

C. State Transition

A node in the search graph represents the state of one group. At each step, the search generates new nodes, computed as the successors of the node with lowest cost. For a group with only one subgroup, this corresponds to moving to its adjacent triangles. For a group with multiple subgroups, this refers to each subgroup independently taking an action which may be a normal traversal, a split, or a merge. For k subgroups, there are a maximum of 2^k possible traversals which can be generated. Each subgroup may also choose to `PAUSE` when it reaches the goal, or when it chooses to merge with another subgroup.

`SPLIT`. Let s' and s'' be the next states from s with and without a split respectively. A split node s' is expanded as a valid successor if one of the following conditions is satisfied: (1) The deformation cost without a split is greater than a threshold: $c_{def}(s, s'') > T_{split}$. (2) The next state produced as a result of a split is more likely to reach the goal: $f(s, s'') > f(s, s')$. (3) The deformation cost without a split is greater than sum of the deformation cost in the case of a split, and the resulting split penalty: $c_{def}(s, s'') > c_{def}(s, s') + c_{split}(s, s')$. We evaluate the two traversal exits of the head triangle to see if both exits are too narrow to determine the need for a split. In a channel, if the deformation cost value is too high, the channel entry triangle may be a valid split candidate. A look-ahead method may be used to better evaluate the potential benefits of a split.

`MERGE`. For each subgroup gr_i, we compute the distance between its head and the heads of the other subgroups. If the minimum distance to another subgroup is less than T_{merge}, or if the two subgroup heads are in adjacent triangles, a valid merge action is identified. We currently assume that only two subgroups can merge at a time. A merge between three subgroups would thus require 2 merge actions. `MERGE` is considered for the following cases: (1) The exits of two subgroup traversals belong to the same triangle, causing them to merge. (2) The exits of two subgroup traversals lie in adjacent triangles, with four possible cases (Fig. 5). In this case, one subgroup can pause to wait for the other subgroup, then merge. (3) The distance between two subgroups is within the threshold and in this case one subgroup can also wait for the other. This condition may produce a false positive when there is a thin obstacle separating two subgroups, in which case performing a wait will result in the distance exceeding the threshold, thus invalidating the merge.

Once two subgroups are chosen to merge, the one with smaller h_{dis} value will receive the `PAUSE` action and the other subgroup will move `LEFT` or `RIGHT` towards the paused subgroup. The wait-to-merge strategy mitigates

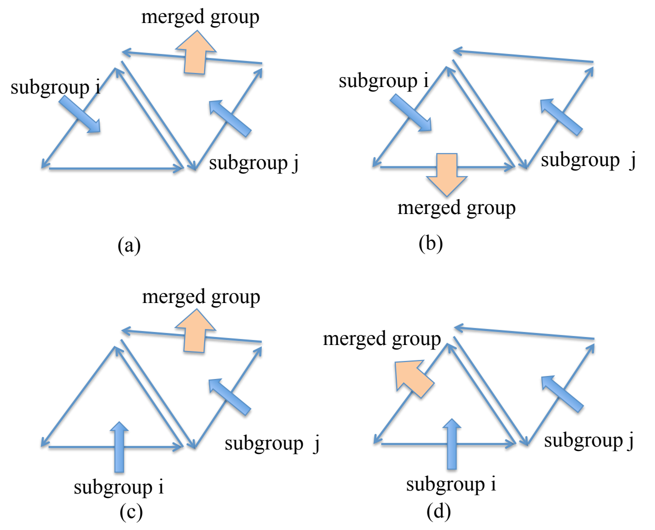


Fig. 5. Illustration of MERGE when two subgroups are in neighbor triangles. There are four different cases (a-d).

synchronization problems between the merging groups. Using this procedure, there is only one valid action combination for the two merging subgroups gr_i and gr_j. For example, suppose that gr_i stops to wait for gr_j, and gr_j moves LEFT, then the only valid action combination is A(gr_i) = PAUSE, A(gr_j) = LEFT, invalidating other action combinations: { A(gr_i) = PAUSE, A(gr_j) = RIGHT }, and { A(gr_i) = PAUSE and A(gr_j) = SPLIT }.

ActionMask and ***Mutex***. We introduce *ActionMask* and *Mutex* to efficiently compute possible transitions at each state. *ActionMask* is used to compute all action combinations, and illegal action combinations are excluded using *Mutex*.

$$\textit{ActionMask} = \begin{cases} 1, & \text{If the action is valid} \\ 0, & \text{If the action is invalid.} \end{cases} \qquad (15)$$

Note that PAUSE and MERGE are mutually exclusive actions, and are defined as one element. *ActionMask* is a quadruple: e.g., *ActionMask* $= \{1, 1, 1, 0\}$ means that the possible actions are: LEFT, RIGHT, SPLIT while PAUSE/MERGE is invalid. To compute *ActionMask*, we do the four following checks when expanding a given state: (1) if it has reached the goal, (2) if any of the traversals are possible (3) if it satisfies splitting conditions, and (4) if it satisfies merging conditions. *Mutex* is used to omit invalid action combinations of possible actions with MERGE. We compute the next possible action combinations for multiple subgroups by permutations and combinations, then delete invalid ones based on Mutex Criteria. This will be computed while judging if there is any merging.

Consider the following example of a state with 5 subgroups with the following action possibilities: (1) All subgroups can go either left or right. (2) Two possible merges: gr_0 PAUSE to wait gr_1 moving RIGHT, and gr_2 MERGE with gr_3 by moving LEFT. (3) gr_4 is a valid split candidate. The action mask each sub group is defined as follows: A(gr_0)={L, R, P}, A(gr_1)={L, R}, A(gr_2)={L, R}, A(gr_3)={L, R, M}, A(gr_4)={L, R, S}. There are 75 possible action combinations, e.g. LLLLL, but the mutex efficiently prunes unnecessary actions to dramatically reduce the number of feasible actions to be considered by the planner. Action combinations, PL*** and **RM*, are invalid according to the Mutex Criteria. Here each * can be replaced by any possible action.

D. Search Procedure

Our path planning method for agent groups is based on the A* algorithm using the problem domain described above. Lowest cost nodes from the OPEN list are successively expanded until the goal state is reached. In our case, each group state can occupy multiple triangles in the environment which poses the challenge of marking nodes that have been previously visited. Triangles representing free space in the environment are marked as one of the following: (1) *not visited*: no generated state visited this triangle, (2) *edge visited*: a node in OPEN list visited this triangle, and (3) *visited*: a node in CLOSED list visited this triangle. A triangle would be accessible, if it was not previously visited, or if it was accessed from a different edge than its current visited edge. Else, it is inaccessible. The planner searches for a sequence of group states from the start to the goal, where a group may split into multiple subgroups, each subgroup may choose a separate path before merging to reach the target in a coherent and persistent fashion.

E. Complexity Analysis

Space complexity. There are four possible actions for each subgroup. Suppose there is at most k subgroups in a group state and the search depth is d, then the upper bound of the size of OPEN is 4^{kd}. If there is no limit for k, then k may increase as the depth d increases. For practical purposes, the effective branching factor of the search is much lower due to the efficient pruning of unnecessary actions using *Mutex*, and we limit the value of k to a reasonable value (see results below).

Time complexity. Suppose the search space size is N, and the size of the open list is $O(N)$, then the time complexity is $O(NlogN)$. If N=4^{kd}, the time complexity becomes $O(dk4^{kd})$, where d is the depth of the search. In practice, we have observed an average effective branching factor of 1.32 from experiments run on three different maps with a maximum depth of 100. These experiments are described in Section V.

V. EXPERIMENTS

We have implemented our path planning algorithm for agent groups on top of Tripath Toolkit [23]. We demonstrate the validity and efficiency of our approach on a variety of benchmarks, as described below. In the experiments below, Env1 is the map of Fig. 11, Env2 is the map of Fig. 6, Env3 is the map of Fig. 12, Env4 is the map of Fig. 7, Env5 is a larger scale of map of the one in Fig. 7.

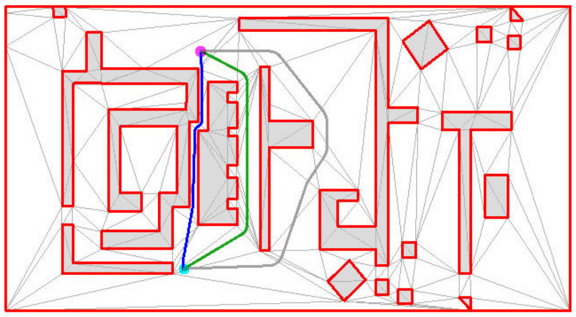


Fig. 6. Paths obtained with varied cost weights: the blue path has distance weight w_1 =1.0 and deformation weight w_2 =0.0, the green path has w_1 = 0.4 and w_2 = 0.6, and the gray path has w_1 = 0.2 and w_2 = 0.8.

We use the term *group size* to measure the initial dispersion of a given group. In our experiments, an initial group dispersion is set as a square with side of *group size*. We set 1 *group size* to be the average width of the passages in each environment being used. Note that the final shape at the end of a path will be restricted by the goal area. In the simulations below, the initial and goal group are visualized as one point.

A. Deformation and Splitting

Deformation cost. To evaluate the impact of the deformation cost we run the planner with varied cost weights. As shown in Fig. 6, the deformation cost models the degree of shape deformation achieved while moving along the paths. If the deformation weight w_2 is set to 0, the distance cost dominates resulting in shortest paths returned by A*, shown as the blue path in Fig. 6. Increasing the deformation cost weight produces longer paths with less deformation. The green path in Fig. 6 is computed with the distance weight w_1 = 0.4, the deformation weight w_2 = 0.6, and the gray path in Fig. 6 is computed with the distance weight w_1 = 0.2, the deformation weight w_2 = 0.8.

Split penalty. The search space grows as the number of splits increases. Split penalty is used to control the tendency of a group to split. A *split limit* is used in order to control the maximum number of splits allowed. In this experiment, we applied different split limits in order to test the size of the search space by adjusting the split penalty. The benchmark for this test is a 1k-LCT polygonal map, which is a particularly challenging scenario for splitting, as shown in Fig. 7. To evaluate the relationship between the memory requirements of the search and the split limit, we perform several random searches of varying path lengths with different split limits, as shown in Fig. 8. A split limit of 4 limits the size of the `OPEN` list to a maximum of $50,000$ nodes, even for very long paths.

Our algorithm is capable of planning varied routes for subgroups based on the specified weights in our cost function. Different split and merge results can be obtained, as shown in Fig. 9. Examples (a)-(d) in the figure have the initial *group size* of 2.5, and examples (e)-(h) have the initial *group size* of 3.Note that although there are only two subgroups

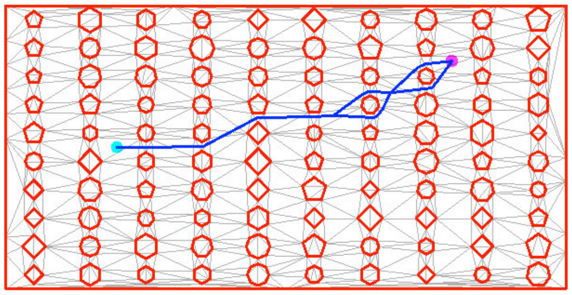


Fig. 7. Group search on 1k-LCT matrix map with split constraint = 2.

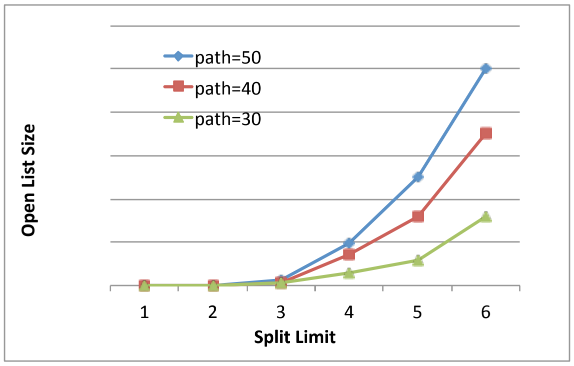


Fig. 8. Relation between memory requirements (size of `OPEN`) and the maximum number of splits. We generated paths to random goals and picked the first paths that were obtained with approximate length of 30, 40 and 50.

generated for a split due to the triangulation mesh, an arbitrary number of subgroups can be achieved by multiple splits. A case of multiple splits is shown in example (d). Fig. 10 illustrates, in left-right, top-down order the dynamic group state sequences of the example in Fig. 9(b). The blue channels represent the individual states of the solution group path, which includes one split and one merge.

B. Effective Branching Factor and Search Horizon

To compute the effective branching factor for our problem domain, we perform searches on three different environment scales and compute the size of the open list for different search depths. The results are given in Table I. The main values in the table are the effective branching factors, the values in parenthesis are the corresponding open list sizes. The split constraint is 4, and the average effective branching factor is 1.32.

TABLE I

OBSERVED EFFECTIVE BRANCHING FACTORS.

Depth	Env1	Env2	Env3	Env4
6	1.62 (18)	1.59 (16)	1.63 (18)	1.49 (11)
12	1.53 (165)	1.43 (72)	1.44 (81)	1.35 (36)
24	1.40 (3026)	1.36 (1526)	1.39 (2761)	1.22 (109)
36	1.28 (6538)	1.26 (3876)	1.28 (7621)	1.20 (700)
60	1.18 (20371)	1.18 (16721)	1.17 (18421)	1.19 (27881)
100	1.13 (190246)	1.12 (21641)	1.12 (98314)	1.12 (64977)
avg	1.36	1.32	1.34	1.26

Fig. 11 visualizes all the triangles that were visited during an example search by highlighting them in blue. When

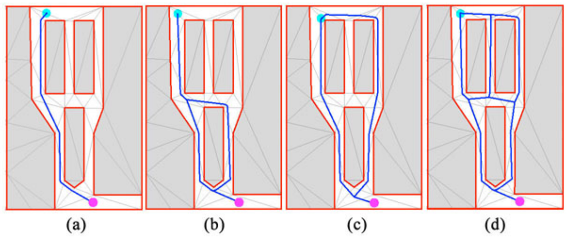
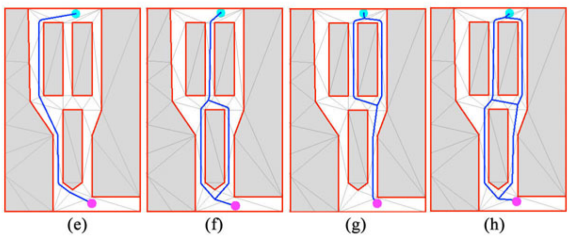



Fig. 9. Example of split and merge cases including multiple splits and merges, using different weights in our cost function. The cost weights (w_1,w_2,w_3) used in each example were a:(0.8,0.1,0.1), b:(0.5,0.4,0.1), c:(0.4,0.5,0.1), d:(0.2,0.8,0), e:(0.4,0.4,0.2), f:(0.5,0.4,0.1), g:(0.4,0.5,0.1), h:(0.3,0.7,0).

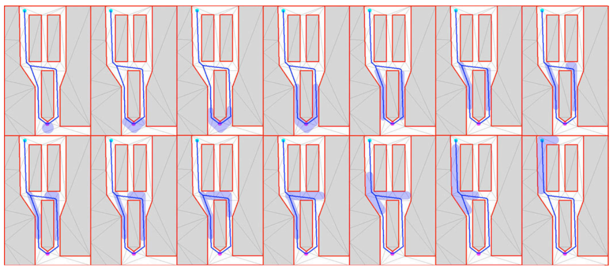
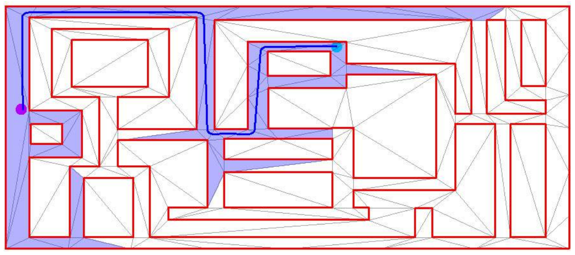


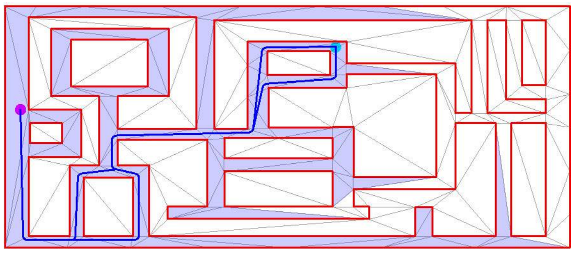
Fig. 10. States of the solution path obtained for the example in Fig. 9(b).

the deformation cost plays a greater role in the evaluation function, the number of visited triangles increases as well as the sizes of the OPEN and CLOSED lists. A balance between different cost weights should be considered for practical applications. When w_2 =0 and w_3 =0, the search degenerates into A*, as shown in Fig. 11(a). Comparing Fig. 11(a) and Fig. 11(b), the number of visited triangles in (a) is less than (b). The path in (a) is shorter but requires more shape deformation. Conversely, the path in (b) is longer but requires less deformation with the use of splits. Additionally, as the initial size of the group increases, the open list size and the number of expanded channels also increases due to the increased possibility of splitting.

The number of visited triangles and the size of the open list depend on the initial *group size* and the search depth. Using the Fig. 12 map containing 650 LCT triangles, we perform tests with start and end points giving shortest path lengths (number of traversals in the path) of 15, 20, 25, 30, 35, and 40 with A*, and with the group search algorithm running initial *group size* of 1.0, 2.0, 3.0, and 4.0. The number of visited triangles relative to the *group size* is shown in Fig. 13. The associated results of open list size are given in Table II. Larger groups require more node expansions during the search to produce paths that satisfy deformation constraints, and may require additional splits in narrow passages, where smaller groups may be able to pass



(a) Parameters: w_1 =1.0, w_2 =0, w_3 =0. Search: depth=23, expanded triangles=50, open list size=84, closed list size=63.



(b) Parameters: w_1 =0.5, w_2 =0.5, w_3 =0. Search: depth=30, expanded triangles=91, open list size=427, closed list size=336.

Fig. 11. Illustration of the visited triangles during our group search method.

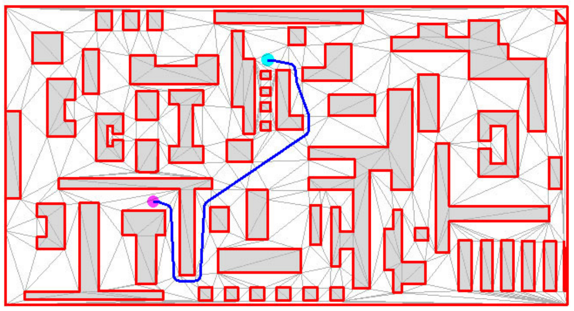


Fig. 12. Environment used for extracting the number of visited triangles shown in Fig. 13 and the open list size shown in Table II.

through.

TABLE II
SIZE OF THE OPEN LIST DURING SEARCH.

	path length					
	15	20	25	30	35	40
A*	66	97	107	173	289	493
group size=1.0	93	154	249	413	851	1248
group size=2.0	492	687	1571	1630	1680	2992
group size=3.0	872	1322	2896	4142	4823	12431
group size=4.0	821	2783	2843	5080	9053	20972

C. Computational Efficiency

Table III illustrates the efficiency of our group search method. For five environments, different split constraints are applied on different path lengths. The average path length and running time over 30 trials of random initial and goal locations are given in the table. For each trial, we present the numbers obtained with 0, 1 and 2 splits. The length of a split path is computed by taking its longest branch. Once the group is allowed to split and merge, the split groups may find

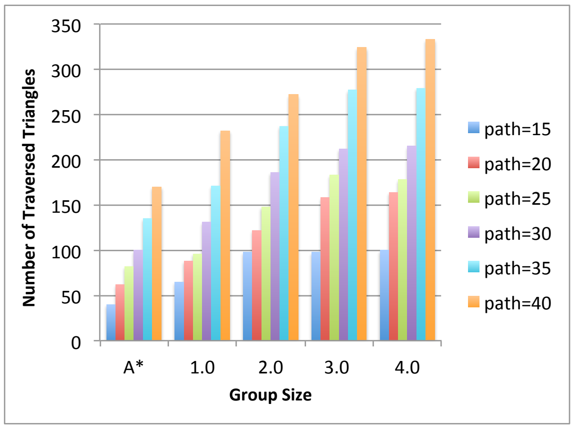


Fig. 13. Number of traversed triangles (vertical axis) versus the *group size* used during the search (horizontal axis).

a shorter path to reach the target, thus resulting in a shorter average path length.

TABLE III

COMPUTATION TIMES IN DIFFERENT ENVIRONMENTS WITH DIFFERENT SPLIT CONSTRAINTS

		0 split		1 splits		2 splits	
env	size	*len*	*t*(ms)	*len*	*t*(ms)	*len*	*t*(ms)
Env1	248	20.6	0.23	22.8	0.27	25.7	0.44
Env2	304	31.4	0.24	31.8	0.31	34.2	0.63
Env3	650	44.1	0.26	42.6	0.38	45.6	0.97
Env4	1244	50.8	0.27	52.8	0.46	56.2	1.53
Env5	6514	102.7	1.40	104.8	28.60	107.6	315.29

VI. CONCLUSIONS

This paper presents a path planning approach for coherent and persistent groups in arbitrarily complex environments. Our work introduces a planning methodology that allows groups of agents to deform as much as needed and at the same time achieve splitting and merging behaviors in a controlled way. Paths to given goals are computed while optimizing a three-tuple cost vector of distance, deformation, and split costs. The cost vector encodes the desired planning strategy and can produce paths with different characteristics with respect to group coherence and persistence behaviors. Such capabilities have not been demonstrated before.

Limitations. Although paths can be computed efficiently in reasonably complex environments, the introduced possibility of splitting groups can considerably increase the cardinality of the search space. A reasonable tradeoff is to balance the influence of the deformation cost and to impose constraints on the number of possible splits in order to meet performance constraints. While our present work only addresses the group path planning problem, our framework is designed to be easily integrated with local collision-avoidance techniques [24] in order to simulate autonomous agents exhibiting goal-directed navigation while being subject to group constraints. In a split, the agents in a group choose their moving directions (LEFT or RIGHT) according to the branch capacities, which are validated by the clearances stored in LCT mesh. The integration of path following behaviors while avoiding collisions and satisfying group coherence and persistence constraints is left as future work.

REFERENCES

[1] Marcelo Kallmann. Dynamic and robust local clearance triangulations. *ACM Transactions on Graphics*, 2014, to appear.

[2] Craig Reynolds. Flocks, herds and schools: A distributed behavioral model. In *Proc. Conf. on Computer Graphics and Interactive Techniques*, pages 25–34, 1987.

[3] Craig Reynolds. Steering behaviors for autonomous characters. In *Proc. of Game Developers Conference*, pages 763–782, 1999.

[4] Lydia Kavraki and Jean-Claude Latombe. Randomized preprocessing of configuration space for fast path planning. In *IEEE Int. Conf. Robot. Automation*, pages 2138–2145, 1994.

[5] Lydia Kavraki, Petr Svestka, Jean-Claud Latombe, and Mark Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. In *Proc. IEEE ICRA*, pages 566–580, 1996.

[6] Gildardo Sanchez and Jean Latombe. Using a prm planner to compare centralized and decoupled planning for multi-robot systems. In *IEEE Int. Conf. on Robotics and Automation*, 2002.

[7] John Reif and Hongyan Wang. *Social Potential Fields: A Distributed Behavioral Control for Autonomous Robots*. Elsevier, 1999.

[8] Adrien Treuille, Seth Cooper, and Zoran Popović. Continuum crowds. *ACM Trans. Graphics*, 25:1160–1168, 2006.

[9] Tucker Balch and Maria Hybinette. Social potentials for scalable multi-robot formations. In *IEEE ICRA*, pages 73–80, 2000.

[10] Tsai-Yen Li and Hsu-Chi Chou. Motion planning for a crowd of robots. In *IEEE ICRA*, pages 4215–4221, 2003.

[11] Peter Hart, Nils Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Systems Science and Cybernetics*, pages 100–107, 1968.

[12] Latombe Jean-Claude. *Robot Motion Planning*. Kluwer Academic Publishers, Norwell, MA, 1991.

[13] Stephane Leroy, Jean-Paul Laumond, and Thierry Siméon. Multiple path coordination for mobile robots: A geometric algorithm. In *Proc. IJCAI*, pages 1118–1123, 1999.

[14] Wouter Van Toll, Atlas Cook, and Roland Geraerts. Real-time density-based crowd simulation. *CAVW*, pages 59–69, 2012.

[15] Joseph Henry, Hubert Shum, and Taku Komura. Environment-aware real-time crowd control. In *Proc. ACM SIGGRAPH/Eurographics SCA*, pages 193–200, 2012.

[16] Arno Kamphuis and Mark Overmars. Motion planning for coherent groups of entities. In *IEEE ICRA*, pages 3815–3822, 2004.

[17] Arno Kamphuis and Mark Overmars. Finding paths for coherent groups using clearance. In *Proc. ACM SIGGRAPH/Eurographics SCA*, pages 19–28, 2004.

[18] Andrew Kimmel, Andrew Dobson, and Kostas Bekris. Maintaining team coherence under the velocity obstacle framework. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 247–256, 2012.

[19] Athanasios Krontiris, Sushil Louis, and Kostas E Bekris. Multi-level formation roadmaps for collision-free dynamic shape changes with non-holonomic teams. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1570–1575, 2012.

[20] Renato Silveira, Edson Prestes, and Luciana Nedel. Managing coherent groups. *Computer Animation and Virtual Worlds*, pages 295–305, 2008.

[21] Ioannis Karamouzas, Roland Geraerts, and Mark Overmars. Indicative routes for path planning and crowd simulation. In *Proc. of Int. Conf. on Foundations of Digital Games*, pages 113–120, 2009.

[22] Roland Geraerts. Planning short paths with clearance using explicit corridors. In *IEEE ICRA*, pages 1997–2004, 2010.

[23] Marcelo Kallmann. Triangulation and path planning toolkit, http://graphics.ucmerced.edu/software/tripath/, 2010.

[24] Jur Van den Berg, Ming Lin, and Dinesh Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *IEEE Int. Conf. on Robotics and Automation*, pages 1928–1935, 2008.