Space-Time Group Motion Planning

Ioannis Karamouzas, Roland Geraerts, and A. Frank van der Stappen

Abstract. We present a novel approach for planning and directing heterogeneous groups of virtual agents based on techniques from linear programming. Our method efficiently identifies the most promising paths in both time and space and provides an optimal distribution of the groups' members over these paths such that their average traveling time is minimized. The computed space-time plan is combined with an agent-based steering method to handle collisions and generate the final motions of the agents. Our overall solution is applicable to a variety of virtual environment applications, such as computer games and crowd simulators. We highlight its potential on different scenarios and evaluate the results from our simulations using a number of quantitative quality metrics. In practice, our system runs at interactive rates and can solve complex planning problems involving one or multiple groups.

1 Introduction

In this work, we address the problem of planning the paths and directing the motions of agents in an environment containing static obstacles. The agents are organized into groups having similar characteristics and intentions, such as an army of soldiers in a real-time strategy game or virtual pedestrians that cross paths at an intersection. Each group has its own start and goal position (or area), and each group member will traverse its own path. Our formulation is designed for large groups and the main objective is to steer the group agents towards their destinations as quickly as possible and without collisions with obstacles or other agents in the environment. This task would have been simple, if the paths of the agents were independent of each other. However, due to space restrictions, congestions may appear and waiting or finding alternative paths may be more efficient for some agents. Furthermore, the time dimension introduces additional complexity into the planning process;

Ioannis Karamouzas
Department of CS&E, University of Minnesota
e-mail: ioannis@cs.umn.edu

Roland Geraerts · A. Frank van der Stappen
Utrecht University, The Netherlands
e-mail: {roland,frankst}@cs.uu.nl

E. Frazzoli et al. (Eds.): *Algorithmic Foundations of Robotics X*, STAR 86, pp. 227–243.
DOI: 10.1007/978-3-642-36279-8_14

congestions only appear at certain moments in time, while at other moments the same area may be completely empty allowing the agents to navigate through it without delay.

To resolve the aforementioned problems, we present a novel algorithm for planning and directing group motions that is based on *linear programming*. Our solution translates the group planning problem into a *network flow* problem on a graph that represents the free space of the environment. It plans in both space and time and uses the *column generation* technique [3] from the operations research theory to efficiently identify the most promising paths in the graph. The planner computes an optimal distribution of the agents over these paths such that their average traversal time is minimized. The computed space-time plan is then given as an input to an underlying agent-based method in order to handle collisions between the agents and generate their final motions.

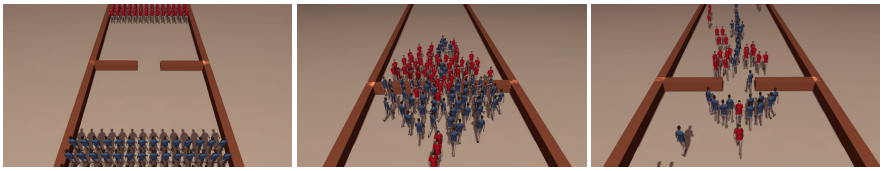
As compared to prior solutions, our method offers the following characteristics:

- Space-time planning of multiple groups of heterogeneous agents based on a linear programming approach;
- Optimal distribution of the agents over alternative paths to resolve congestions;
- Coordination of the movements of the agents to prevent deadlocks that typically occur near bottlenecks (e.g. narrow passages) in the environment;
- Explicit waiting behavior that leads to convincing simulations of high-density crowds with no observed oscillatory behavior;
- Seamless integration of global path planning with existing schemes for local collision avoidance;
- Real-time performance in complex and challenging scenarios.

2 Related Work

The most common approach to simulate groups of autonomous agents is the flocking technique of Reynolds [19]. Although flocking leads to natural behavior, the group members act based only on local information and thus, they can get stuck in cluttered environments. To remedy this, Bayazit *et al.* [1] combined flocking techniques with probabilistic roadmaps, whereas Kamphuis and Overmars [8] used the concept of path planning inside corridors so that the group members can stay coherent.

In general, two-level planning approaches have been widely used in the graphics and animation community for multi-agent navigation and crowd simulation. In these approaches, a high-quality roadmap governs the global motion of each agent [5, 31], whereas a local planner allows the agent to avoid collisions with other agents and obstacles. Over the past few years, numerous models have been proposed for local collision avoidance including force-based approaches [7, 11] and variants of velocity-based methods [28, 27]. These approaches are known to exhibit emergent behaviors. Nevertheless, since they are separated from the global planner, they are susceptible to local-minima problems. In highly dynamic and densely packed environments, this normally leads to deadlock situations (see, e.g., Fig. 1(b)) that can only be resolved by rather unnatural motions. In such environments, even replanning is often insufficient to generate a feasible trajectory. Note that, similar to our



(a) (b) (c)

Fig. 1 (a) Two opposing groups, of 56 members each, need to pass through a narrow bottleneck. (b) Agent-based methods lead to congestion and deadlocks. (c) Our technique can successfully handle such challenging scenarios by planning in both space and time.

technique, the recent work of Singh *et al.* [23] resolves deadlocks by planning on a space-time graph. However, their formulation provides a localized space-time plan for each agent and cannot give any guarantees on the overall behavior of the agents.

Multi-agent path planning has also been extensively studied in robotics. Centralized planners, such as [15, 20], compute the motions of all agents simultaneously, whereas decoupled techniques [17, 22] plan a path for each agent independently and try to coordinate the resulting motions. However, both solutions are too computationally expensive for real-time interactive applications. Prioritized approaches have also been proposed that plan paths sequentially according to a certain priority scheme [29, 24]. This reduces the multi-agent planning problem to the problem of computing a collision-free trajectory for a single agent in a known dynamic environment, which can be addressed by roadmap-based space-time planners [30]. Even though prioritized techniques guarantee inter-agent collision avoidance, the growing complexity of the planning space limits the number of agents that they can simulate.

Prior work in the graphics community has also focused on the synthesis of realistic group motions mainly for offline simulations of crowds [14, 13]. Recently, a number of algorithms have been proposed to simulate the local behavior of pedestrian groups [18, 12], but they do not address the issue of path planning. An alternative approach has been proposed in [25] that guides large homogeneous groups using continuous density fields. This approach unifies global planning and local collision avoidance into a single framework and can become expensive when simulating a large number of groups. More recently, Patil *et al.* [16] proposed a novel technique to steer and interactively control groups of agents using smooth, goal-directed navigation fields. Although their approach can effectively handle challenging scenarios, it requires user input to guide the agent flows and successfully resolve congestion, as compared to our model that automatically accounts for time-consuming situations.

Finally, a different solution to the group path finding problem has emerged from the operations research community. Van den Akker *et al.* [26] formulated this problem as a *dynamic multi-commodity flow* problem. Their approach takes as input a graph resembling the free space of the environment and several groups of units, each having its own start and goal position. A *capacity* is associated with each arc on the graph representing the maximum number of units that can traverse this arc per unit time. The objective is to find paths that minimize the average arrival times of all units. Since this problem is known to be NP-hard, they proposed a new heuristic based on techniques from linear programming.

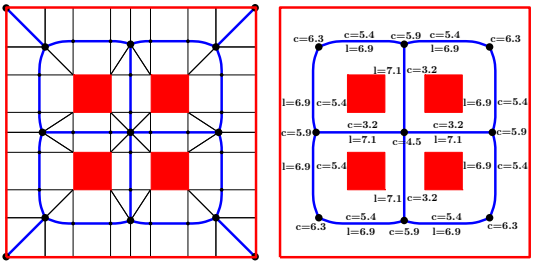
Our model is inspired by the work of van den Akker *et al.*, since we use a similar linear program to coordinate the global motions of the groups (Sect. 4). However, we extend their formulation to allow the simulation of *heterogeneous* groups. We also take into account capacity constraints on the *nodes* of the graph, which provides a more accurate solution to the problem and inhibits the oscillations we observed in [26] when multiple agents have to wait on a certain node. Moreover, van den Akker's model uses a directed graph to capture the free workspace. This significantly restricts the movements of the agents; in real life, for example, pedestrians can walk in either direction along a sidewalk or simultaneously enter/exit a building through the same door. To resolve this, our method uses *undirected* edges and thus, additional constraints are considered. Finally, van den Akker *et al.* only present a theoretical framework for global planning and do not discuss simulation. We address this with a simple, yet effective, *steering* algorithm that guides the agents towards their goals without collisions (Sect. 5).

3 Problem Formulation and Overall Solution

In our problem setting, we are given a geometric description of a virtual environment in which k groups of agents must move. Each group $C_i, i \in [1,k]$ has its own start s_i and goal position t_i and consists of d_i agents that need to navigate from a specified start to a specified goal area defined around the group's origin and destination point respectively. Furthermore, each group C_i is assigned a desired speed U_i^{des} indicating the speed at which its members prefer to move. Given a group C_i, we denote an agent belonging to C_i as A_{ij}, where $j \in [1,d_i]$. For simplicity, we assume that each agent A_{ij} is modeled as a disc having radius r_{ij} and is subject to a maximum speed $\upsilon_{ij}^{\max}$. We further assume that A_{ij} keeps a certain psychophysical [7] distance $\rho_{ij} \geq r_{ij}$ from other agents and static obstacles in order to feel comfortable. This distance defines the *personal space* of A_{ij} modeled as a disc centered at the agent. The task is then to steer the group members A_{ij} toward their corresponding goal areas as quickly as possible and without collisions with the environment and with each other.

We propose a two-level framework to solve the aforementioned planning problem. In the first phase, we formulate the group planning problem as a *dynamic multi-commodity flow problem* and employ an Integer Linear Programming (ILP) approach to solve it. The ILP plans in both time and space by taking into account capacity constraints on the edges and the nodes of a graph that is based on the *medial axis* of the environment. The variables in the ILP represent the number of agents using a certain path. Such a path is described by the graph nodes that the agent should visit along with the times at which these nodes should be reached. Furthermore, waiting behavior can be encoded in the path.

In the second phase of our framework, the paths computed by the ILP are given as input to an agent-based steering algorithm in order to generate the final motions of the agents. The algorithm creates a number of *lanes* across the medial axis edges and computes for each agent a trajectory along these lanes that respects the space-time



(a) (b)

Fig. 2 (a) A medial axis graph MG enhanced with proximity information to nearest obstacles. (b) Its corresponding capacitated graph G.

plan provided by the ILP. At every step of the simulation, a local collision avoidance method is also used to guarantee collision-free navigation for the agents.

In the following sections, we elaborate on our method. For a more detailed explanation, including theorems, proofs and additional experiments, we refer to [9].

4 Path Planning for Groups

This section outlines the first phase of our framework that formulates the multi-group path planning problem as a dynamic multi-commodity flow problem.

4.1 Creating a Capacitated Graph

To solve the problem, we first need a graph that resembles the free space of the environment. We base this graph on the edges and vertices of the medial axis (MA), as it provides maximum clearance from the obstacles and includes all cycles present in the environment. In particular, we precompute a MA graph $MG = (V, E)$ by sampling *event points* (nodes of degree 2 or more) along the MA based on the approach proposed in [4]. The resulting structure fully covers the free space of the environment using a minimum amount of sample points. Also, for each sample point, its minimum clearance is stored along with its corresponding nearest obstacle points. Figure 2(a) shows an example of such a graph. Note that the graph is undirected.

Based on MG, we compute a *capacitated graph* $G = (N, E)$ as follows. We first copy all vertices and edges from MG to G. We then determine all nodes of degree 1 in the graph. Such nodes denote dead-ends in the environment and are removed from the set N in G. Their corresponding edges are also removed from the edge set E of G. For each remaining edge $e \in E$, we define its traversal time l_e as the time that an agent requires to traverse this edge and compute it as the edge length divided by the maximum speed $U^{\max}$ among all groups' desired speeds. A capacity c_e is also assigned to each edge denoting the maximum number of agents that can traverse the edge at the same time while walking next to each other. The edge capacity is computed as the minimum clearance along the edge divided by the maximum personal space $\rho^{\max}$ among all agents. Similarly, a capacity c_n is associated with each node n in the graph indicating the maximum number of agents that can simultaneously be

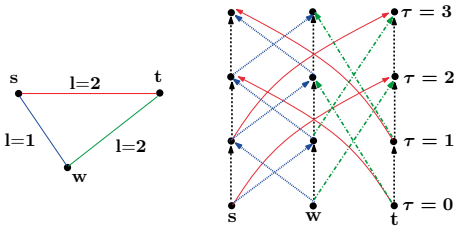


Fig. 3 (a) A simple capacitated graph with integral traversal times. (b) Its corresponding time-expanded graph with time horizon $T = 4$ and time step $\Delta t = 1s$. Note that for clarity, we omit the capacity information from both graphs.

on the node. We refer the reader to Fig. 2(b) for the capacitated graph corresponding to the MA graph depicted in Fig. 2(a).[1]

The generated graph captures different paths that the agents can follow in order to reach their destinations. Nevertheless, to properly avoid congestions, we must not only know which nodes are visited but also at which times these nodes should be reached. In addition, waiting at certain nodes may be more efficient for some of the agents. For that reason a condensed *time-expanded graph* is defined as proposed in [2] that adds the time dimension to the graph G. The basic idea here is to round up the traversal time l_e of each edge e in G to the nearest multiple of an appropriately chosen time step Δt, i.e. $l_e^* = \lceil l_e/\Delta t \rceil$. The capacity of the edge is also multiplied by Δt in order to define the maximum number of agents that can traverse the edge in one time unit, i.e. $c_e^* = \lfloor c_e \cdot \Delta t \rfloor$. In a similar manner, the discrete capacity of each node n is expressed as $c_n^* = \lfloor c_n \cdot \Delta t \rfloor$.

Let $G^T = (N^T, A^T)$ denote the time-expanded graph of G, where T defines the time horizon, that is the total number of discrete time steps. G^T is directed and is constructed from the static graph G based on its discrete capacities and traversal times. The set N^T contains a copy of each node n of the static graph G for each time step $\tau = 0...T-1$. Such a copy is denoted as $n(\tau)$. In addition, for every edge $e = \{n,m\}$ in the graph G, two arcs will be created in G^T, one from $n(\tau)$ to $m(\tau + l_e^*)$ and one from $m(\tau)$ to $n(\tau + l_e^*)$. Finally, waiting arcs are also added to A^T from each node $n(\tau)$ to the same node one time-step later $n(\tau+1)$, for all $\tau < T$. Such an arc allows an agent to stay at node n for one time-step period, i.e. $l^* = 1$. Its capacity is defined as the number of agents that can simultaneously wait at the node and is the same as the discrete capacity c_n^* of the static node n.

An illustration of a time-expanded graph is given in Fig. 3. Such a graph is not explicitly constructed. Instead, time-expanded nodes and arcs are created on-the-fly as further discussed in Sect. 4.3. In addition, an upper bound on the time horizon T is set as explained in [9].

[1] If the start s_i or goal t_i position of a group C_i is not included in the graph G, it is added to the set N as an extra node. The new node is also connected to G introducing additional edges in the set E.

4.2 ILP Formulation

In our problem formulation, we are given the origin s_i, destination t_i and the size d_i of each group C_i. For now, let us assume that we know all valid paths in the time-expanded graph G^T corresponding to each origin-destination pair. A path is valid, if it starts at a node $s_i(0) \in N^T$ for some $i \in [1,k]$ and ends at node $t_i(\tau)$ for the same i. Let $\mathcal{P}$ denote the set of all these valid paths in G^T. For every path $p \in \mathcal{P}$ we introduce a decision variable f_p which denotes the number of agents using the path p. Then, we can model our path planning problem as an Integer Linear Programming (ILP) problem as follows.

Our goal is to minimize the average arrival time of all agents, or, equivalently, the sum of the travel times of all agents. We use l_p to denote the cost (length) of path p, which equals to the arrival time of p at its destination. This leads to the objective function shown in (1). We also formulate demand constraints to guarantee that all agents will reach their targets as expressed in (2), where $\mathcal{P}_i$ is the set of paths in G^T starting at s_i and ending at t_i given a group C_i.

Restrictions are also needed to ensure that the arc capacity constraints are obeyed. In our problem setting, these constraints are much harder to model than in a normal multi-commodity flow problem, since the static capacitated graph G is undirected. Consequently, an edge can be traversed in both directions at the same point in time. To avoid having to add an enormous number of constraints, we introduce a non-negative decision variable u_e for every edge e in the graph G. We call one direction on the edge forward and the other one backward. Let $F(e) \subset A^T$ be the forward arcs in the time-expanded graph G^T emerging from e, and $B(e) \subset A^T$ the corresponding backward arcs. Then, the capacity c_e^* of the edge is divided between the two directions. We do not fix the capacity distribution beforehand, but we make it time-independent by putting the capacity of the forward arcs equal to u_e and the capacity of the backward arcs equal to $c_e^* - u_e$. This adds (3) and (4) to the ILP, where $\mathcal{P}_a$ denotes the subset of paths using arc $a \in A^T$.

Finally, to account for the throughput limitation of each node v of the time-expanded graph, (5) is added to the ILP constraints, where $\mathcal{P}_v$ denotes the subset of paths in the time-expanded graph using the expanded node v. The above considerations result in the following ILP problem:

$$\min \sum_{p \in \mathcal{P}} l_p f_p \tag{1}$$

$$s.t. \sum_{p \in \mathcal{P}_i} f_p \geq d_i \qquad \forall i = 1...k \tag{2}$$

$$\sum_{p \in \mathcal{P}_a} f_p - u_e \leq 0 \qquad \forall e \in E, a \in F(e) \tag{3}$$

$$\sum_{p \in \mathcal{P}_a} f_p + u_e \leq c_e^* \qquad \forall e \in E, a \in B(e) \tag{4}$$

$$\sum_{p \in \mathcal{P}_v} f_p \leq c_v^* \qquad \forall v \in V^T \tag{5}$$

$$f_p \geq 0 \text{ and integral} \qquad \forall p \in \mathcal{P} \tag{6}$$

$$u_e \geq 0 \text{ and integral} \qquad \forall e \in E \tag{7}$$

Obviously, we do not know the entire set of paths $\mathcal{P}$ and enumerating it would be impractical, since there are infinite paths in the time-expanded graph. Therefore, we will make a selection of paths that we consider 'possibly useful', that is, paths that might improve the value of our objective function, and we will solve the ILP for this small subset. We determine these paths by considering the LP-relaxation of the ILP, which is obtained by removing the integrality constraints from the decision variables f_p and u_e. We solve the LP-relaxation through the technique of *column generation*, which was first described by Ford and Fulkerson [3].

4.3 Column Generation and Path Finding

The basic idea of column generation is to solve the LP problem for a restricted set of variables (the set of valid paths in G^T in our case) and then add variables that may improve the objective value until no additional variables can be found anymore. In case of a minimization problem, it is well known from the theory of column generation that the addition of a variable will only improve the solution if its *reduced cost* is negative. The reduced cost of a path $p \in \mathcal{P}_i$ for a group C_i is equal to

$$l_p + \sum_{a \in \alpha(p)} \mu_a + \sum_{v \in \lambda(p)} \phi_v - \psi_i, \tag{8}$$

where $\alpha(p)$, $\lambda(p)$ denote the arcs and nodes respectively in G^T used by path p, and $\psi_i, \mu_a, \phi_v \geq 0$ are the dual variables for the demand, arc and node constraints respectively that derive from the *dual* of the current LP [2,3]. Consequently, for each group C_i we need to find a path $p \in \mathcal{P}_i$ such that

$$l_p + \sum_{a \in \alpha(p)} \mu_a + \sum_{v \in \lambda(p)} \phi_v - \psi_i < 0. \tag{9}$$

[2] Every linear programming problem, referred to as a primal problem, can be converted into a dual problem. For a primal problem expressed in its symmetric form $\{min\ \mathbf{c}^T\mathbf{x} : A\mathbf{x} \geq \mathbf{b}, \mathbf{x} \geq \mathbf{0}, \}$, the corresponding symmetric dual problem is given by $\{max\ \mathbf{b}^T\mathbf{y} : A^T\mathbf{y} \leq \mathbf{c}, \mathbf{y} \geq \mathbf{0}\}$, where $A \in \mathbb{R}^{m \times n}$, $\mathbf{c}, \mathbf{x} \in \mathbb{R}^n$, $\mathbf{b} \in \mathbb{R}^m$ and $\mathbf{y} \in \mathbb{R}^m$.

[3] Given a linear program in its symmetric form, the reduced cost can be computed as $\mathbf{c} - A^T\mathbf{y}$, where $\mathbf{y}$ denotes the variables of the dual of the linear program. In our case, our LP formulation results in a dual variable ψ_i for the demand constraint (2) corresponding to the group C_i, a dual variable μ_a for the constraints (3)-(4) corresponding to the arc a and a dual variable ϕ_v for the constraint (5) corresponding to the node v.

This inequality can actually be rewritten to gain more insight into its meaning. The cost l_p of a path p is equal to the sum of the length of the arcs contained in p, that is $l_p = \sum_{a \in \alpha(p)} l_a^*$, where l_a^* denotes the discrete length (traversal time) of arc $a \in A^T$ (see Sect. 4.1). Note, though, that the traversal time l_a^* is computed based on the maximum desired speed $U^{\max}$ among all groups. However, since each group C_i has its own desired speed U_i^{des}, we define the correct length of the arc a for $p_i \in \mathcal{P}$ as $l_a'^* = \lceil l_a^* \cdot U^{\max} / U_i^{\text{des}} \rceil$. Regarding the $\sum_{v \in \lambda(p)} \phi_v$ term of (9), it is clear that if a node $v \in N^T$ is contained in the path p, the path also contains precisely one arc that terminates at this node (with the exception of the origin node). Consequently, the effective contribution of the path's nodes to the reduced cost of p becomes $\sum_{a \in \alpha(p)} \phi_m$, given that $a = (n, m)$. Finally, reorganizing (9) we obtain:

$$\sum_{a \in \alpha(p)} (l_a'^* + \mu_a + \phi_m) < \psi_i. \tag{10}$$

Deciding whether (10) holds defines the *pricing problem* and can be efficiently solved for each group C_i as follows. We compute the shortest path for C_i in the time-expanded graph G^T, where each arc $a \in A^T$ has a modified length $l_a'^* + \mu_a + \phi_m$. We use an A* search to efficiently find such a path from the origin $s_i(0)$ of the group to some copy of its destination $t_i(\tau)$. In A*, explored nodes of the time-expanded graph are created and added to an open set based on a function $f(v) = g(v) + h(v)$ that determines how promising each node v is. The cost $g(v)$ of reaching the current node v from the origin $s_i(0)$ is based on the modified arc lengths. Regarding the heuristic $h(v)$, we use the shortest path length (expressed in discrete time units) from v to t_i in the static graph G. This heuristic is clearly admissible, since the path length in a time-expanded graph may be enlarged due to the modified lengths of its arcs and the possible presence of waiting arcs. As our proposed heuristic is also consistent, our search algorithm retains the optimality of an A* search and can efficiently compute the shortest path for the group C_i. If the length of this path is shorter than ψ_i, we can add it to the LP and iterate. Otherwise the optimum has been found, since adding the path will not decrease the objective value of the LP.

Algorithm 1 summarizes our column generation algorithm. Note that, initially, the LP has no columns and hence, we need to introduce an initial set of paths, one for each group C_i. In general, we can start with any set, as long as it constitutes a feasible solution [9].

Algorithm 1. Column Generation

```
1: Find initial paths and add them as columns to the LP
2: repeat
3:    Solve LP-relaxation
4:    for each group C_i do
5:       Find a shortest (s_i(0) − t_i(τ))-path in G^T w.r.t. the modified arc lengths
6:       if length of the path is less than ψ_i then
7:          add the path as a new column to the LP
8:       end if
9:    end for
10: until no path has been added
```

4.4 Obtaining an Integral Solution

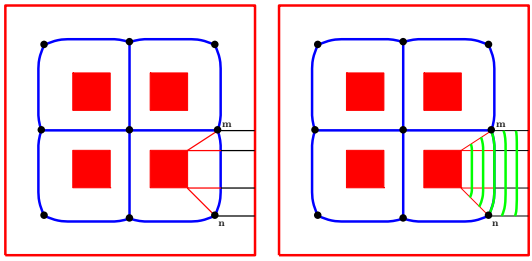
At the end of the column generation algorithm, we have an optimal solution for the LP. Most likely, though, some of the variables f_p in our solution will have a fractional value and hence, we have to ensure that we end up with an integral solution, since we cannot send fractions of agents along a path. We alleviate the problem as follows. Since we have generated useful paths when we solved the LP, we include all these paths in the ILP. We then round down their decision variables f_p, which leads to an integral solution that satisfies the capacity constraints. Because of the rounding down, though, some of the agents will be left without paths. For these agents, we construct additional paths using the Cooperative A* algorithm [21]. These paths are added to the ILP, which then can be solved to optimality. Note that an alternative approach would be to artificially increase the sizes d_i of the groups while solving the LP-relaxation problem. Therefore, after rounding down the f_p variables all agents will be assigned a path.

5 Agent Motion Planning

In this section, we elaborate on the second phase of our framework. The goal here is to generate the final motion of each agent A_{ij} based on its path p_{ij} computed by the ILP. Such a path is defined in the time-expanded graph G^T and can be described by the arcs that it uses. A time-expanded arc is either a waiting arc or a traversal arc that corresponds to an edge in the MA graph MG. Therefore, a straightforward approach for generating the final trajectory of the agent A_{ij} would be to let A_{ij} follow the MA edges, while adjusting its speed in order to adhere to the time-constraints of its ILP path. Note, though, that multiple agents may have the same path in G^T or share the same edge or node at the same time. Since all paths produced by the ILP satisfy the capacity constraints, this means that there is enough clearance for these agents to walk or stand next to each other. Based on this observation, we can utilize the maximum capacity of each edge and discretize the Voronoi regions of the environment into a number of *lanes*. The agents will use these lanes to spread over the environment and reach their goals, respecting at the same time the space-time constraints provided by the ILP paths.

5.1 Constructing Lanes

We construct the lanes by exploiting the properties of the MA graph $MG = (V,E)$. Let $e = \{n,m\}$ denote an edge in the set E, where both n and m are not dead-end vertices. Let us also define a pseudo-orientation for the edge, e.g. from n to m. As described in Sect. 4.1, an edge consists of a number b of sample points B_i, $1 \leq i \leq b$. For each sample point its minimum clearance is also stored along with its closest points to the obstacles' boundaries. Given the orientation of the edge, let l_i and r_i denote the left and right closest obstacle points assigned to each sample B_i.



(a) (b)

Fig. 4 (a) An edge can be expressed as a sequence of portals. (b) Lanes are created along the edge by placing waypoints on the portals.

Then the edge e can be expressed as a sequence of *portals* where each portal is described by the tuple (B_i, r_i, l_i) (see Fig. 4(a)).

Lanes can be easily constructed along the edge by placing waypoints on the navigation portals. Let c_e^* be the integral capacity of the edge in the capacitated graph G. This capacity defines the maximum number of agents that can simultaneously traverse the edge and hence, c_e^* lanes will be created. For each lane $L_k, k \in [1, c_e^*]$, a waypoint is generated on each of the b portals of the edge based on a parameter $\omega_k = k/(c_e^* + 1), \omega_k \in (0,1)$. In case the waypoint is located at the right side of the portal ($\omega_k \leq 0.5$), its exact position w_i is given by: $w_i = r_i + 2\omega_k(B_i - r_i)$. If the waypoint is at the left side of the portal, its position between l_i and B_i linearly depends on $2 - 2\omega_k$. We refer the reader to Fig. 4(b) for the lanes corresponding to the edge e, given that $c_e^* = 5$. Since each edge can also be traversed in the opposite direction, for each lane its reverse one L_k' is also added using the same procedure as above.

5.2 Trajectory Synthesis for an Agent

Given the lanes as constructed above and the ILP path p_{ij} corresponding to the agent A_{ij}, we now describe how we can determine a trajectory for A_{ij}. Let p_{ij} consists of q arcs. We first express this path as a sequence of tuples $(a_1, T_1)....(a_q, T_q)$, where for the θ^{th} tuple, $a_\theta = (n(\tau), m(\tau + l'^*_{a_\theta}))$ denotes an arc in the time-expanded graph and T_θ is the time instant at which the task of the arc (waiting or traversing a lane) should be completed, that is, $T_\theta = (\tau + l'^*_{a_\theta})\Delta t$.

Having determined the times T corresponding to each arc a, we can steer the agent towards its goal as follows. Let $\mathbf{x}_{ij}$ denote the current position of the agent, $\mathbf{v}_{ij}$ its current velocity, and $\mathbf{v}_{ij}^{\text{pref}}$ its preferred velocity. Let also (a_θ, T_θ) denote the next tuple to be processed. If a_θ is a waiting arc, we set the preferred velocity to zero for the next $T_\theta - t$ seconds, where t is the current time of the simulation; note that if $T_\theta \leq t$, we simply ignore the waiting arc and query the next tuple in the list. In case a_θ is a traversal arc, we determine its corresponding edge e in MG along with the edge's lanes $\mathcal{L}$ that have the same orientation as the arc a_θ. The agent needs to select one of these lanes and traverse it within T_θ.

Choosing a Lane: We first determine all lanes in $\mathcal{L}$ that are *free*. A lane L_k is considered as free, if no other agent has entered it at the current time t and if its reverse lane L'_k is unoccupied. Among the free lanes, we select the one that is closest to the agent. Given a lane L_k, the distance $D(L_k, \mathbf{x}_{ij})$ between the agent's current position $\mathbf{x}_{ij}$ and L_k can be computed by projecting $\mathbf{x}_{ij}$ into the segments defined by the waypoints of the lane. In case no free lanes are available[4], with each lane L_k, we dynamically assign a cost that depends on the distance $D(L_k, \mathbf{x}_{ij})$ and the biomechanical energy $E(L_k)$ that the agent needs to spend in order to traverse L_k. This energy is approximated as in the work of Guy *et al.* [6] and is based on a series of experimental studies regarding the relationship between the walking speed and the energy expenditure of real humans. Given $E(L_k)$, the total cost of the lane L_k can now be defined as: $cost(L_k) = c_D D(L_k, \mathbf{x}_{ij}) + c_E E(L_k)$, where c_D, c_E are weights specifying the relative importance of each cost term. Based on the above function, the agent A_{ij} retains the lane $L_s \in \mathcal{L}$ with the lowest cost.

Moving Along a Lane: The agent A_{ij} uses the selected lane L_s as an *indicative route* in order to traverse its current arc. More precisely, an attraction point moves along L_s and attracts the agent forward as defined in [10]. Given the attraction point and the distance that A_{ij} still has to traverse within the time $T_\theta - t$ that has at its disposal, the preferred velocity $\mathbf{v}_{ij}^{\text{pref}}$ of A_{ij} can then be estimated. For additional details on the notion of indicative routes, we refer the reader to [9].

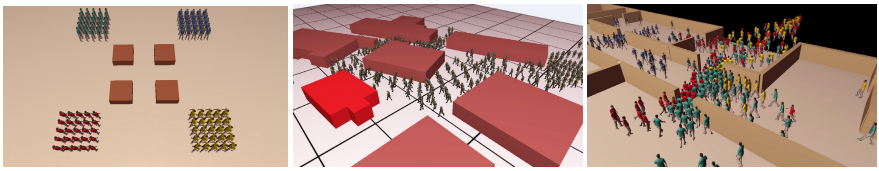
Local Collision Avoidance: At each simulation time-step, the preferred velocity $\mathbf{v}_{ij}^{\text{pref}}$ of the agent A_{ij} is given as an input to a local collision avoidance method in order to compute a collision-free velocity for the agent and update its position. Note that simply using $\mathbf{v}_{ij}^{\text{pref}}$ may not be sufficient for a collision-free motion; the agent may still need to resolve collisions with its neighboring agents and the static part of the environment while advancing along its selected lane or waiting at a specific location. In general, any local method that is based on collision prediction can be used. In our implementation, we consider the model in [11] that tackles the collision avoidance problem using social forces.

6 Experimental Results

We demonstrate the applicability of our approach in four challenging scenarios as can be seen in Figs. 1 and 5. The resulting simulations can be found at http://sites.google.com/site/ikaramouzas/ilp.

Quality Evaluation: Our approach identifies areas with low capacity and divides the agents over different space-time paths allowing them to avoid congestions and

[4] An agent, due to interactions with other agents, may deviate from the time plan provided by the ILP and arrive at its next arc later than the expected time T_θ. Thus, in practice, there may be no free lanes to follow. However, a more relaxed time-plan can be provided for each agent by including an additional time t_{add} to T_θ. This will give the agent more time to complete its task and resolve challenging interactions.



(a) 4-Blocks (b) Military (c) Office

Fig. 5 Example scenarios: (a) Four groups, of 25 agents each, in opposite corners of the environment exchange positions. (b) An army of 400 soldiers needs to move towards a designated goal area in a village-like environment. (c) 700 agents organized into 7 groups have to evacuate an office building through two narrow exits.

quickly reach their destinations. In the 4-blocks scenario, for example, the agents anticipate that a congestion will occur at the center of the environment and avoid it by moving around the obstacles. Our method also prevents deadlocks from occurring by regulating the starting times of the agents, incorporating explicit waiting behavior and successfully dealing with opposing flows of agents through the use of lanes, as can be seen in the narrow bottleneck scenario (Fig. 1). Furthermore, by controlling the starting and waiting times of the agents, no oscillatory behavior is observed when dense crowds of agents thread through narrow spaces, like the doorways in the office scenario.

Our method is also able to generate well-known crowd phenomena which have been noted in the pedestrian literature (see [7] for an overview), such as lane formations (e.g. 4-blocks and military scenarios), queuing and following behaviors (narrow bottleneck and office), as well as slowing down and waiting behaviors to resolve challenging interactions (narrow bottleneck, military and office). Furthermore, in the narrow bottleneck scenario, arch-like blockings are formed at either side of the passage when the two opposite flows meet (Fig. 1(c)). This phenomenon is in accordance with real-life behavior [7].

Besides the visual inspection of the simulations, we are also interested in a quantitative evaluation of our model. As shown in Table 1, by planning in both time and space and exploiting the notion of lanes, interactions among agents are efficiently solved resulting in smooth trajectories and faster traveling times as compared to existing agent-based systems. In the 4-blocks scenario, for example, using the RVO method in combination with a global roadmap [31] led to an average traveling time of 37.06 s, whereas the latest arrival time was 52.59 s. Similarly, in the narrow bottleneck example, it took 219.4 s for the last agent to arrive at its goal and the average traverse time was 139.46 s. Such high traveling times are expected, since the agents' motions are not taken into account during the global planning and hence, congestions and deadlock situations arise. In contrast, using our approach, such situations are predicted and avoided. In the 4-blocks, even when we used a dynamic roadmap to account for congestion as in [6] along with the ORCA method [27], the maximum travel time was 45.79 s, as the agents were still prone to local minima problems.

Table 1 Results for the four scenarios. The optimality error is the average percentage error between the actual and the estimated arrival times of the agents.

Scene	Max. travel time (s)	Avg. travel time (s)	Optimality error (%)
4-Blocks	32.41	30.01	1.29
Bottleneck	186.90	129.75	6.36
Military	149.78	117.61	4.69
Office	189.10	123.59	8.63

To adequately assess the quality of our approach, we also need to determine the efficiency of the steering algorithm used in the second phase of our framework. For that reason, we compared the *actual* time that an agent requires in order to reach its destination with its *expected* arrival time. The latter is simply the length of the agent's time-expanded path computed by the ILP. The *percentage error* between the actual and the expected arrival time can then be used as a measure of the optimality of the agent's trajectory. Clearly, since the agent needs to avoid collisions with other agents and is also subject to dynamic constraints, its actual traveling time is higher than the expected one. However, as can be seen in the last column of Table 1, in all of our experiments, the optimality error is less than 10% indicating that our steering algorithm generates near time-optimal trajectories.

Performance: Table 2 summarizes the performance of our approach on the four benchmark scenarios. All computations were run on a 2.4 GHz Core 2 Duo CPU (on a single thread). The eighth column of the table indicates the total running time of the first phase of our approach. This time is split into the time used by the column generation part of our algorithm, that is the time for solving the LP-relaxation problem (t_{LP}) and for finding paths using A* in the time-expanded graph (t_{path}), the time needed for finding additional paths due to the missing capacities (t_{add}) and the time to solve the final ILP problem (t_{ILP}). The last column of the table indicates the average computation time of the second phase of our approach, that is the average time taken per simulation step to steer the agents and resolve collisions. As can be inferred, our steering algorithm combined with the underlying collision avoidance scheme is very efficient. This is expected, since most of the collisions are taken into account during the ILP planning. Thus, only a limited number of interactions need to be resolved at every step of the simulation.

Regarding the total planning time of the first phase, it is clear that the LP part of the column generation dominates the overall runtime performance. As can be observed, the *difficulty* of the problem seems to have a high impact on the computational cost of the LP. See, for example, the difference in the running times between the 4-blocks and the narrow bottleneck scenarios. The difficulty of the problem is directly related to the arc and node capacities of the time-expanded graph and can be controlled by the size of the time step Δt used for the discretization of the graph. Shorter time steps improve the accuracy of the graph at the expense of higher computational times. In contrast, large time steps result in faster running times; the capacities of the arcs and the nodes of the graph are scaled as well, reducing the

Table 2 The performance of our approach on the four example scenarios. Reported times are in msec. The average simulation time is expressed in msec/frame. The third column indicates the number of nodes and edges of the underlying capacitated graph.

Scene	#Agents	Graph	t_{LP}	t_{path}	t_{add}	t_{ILP}	Total planning time	Avg. sim. time
4-Blocks	100	(9, 12)	0	15.6	0	0	15.6	0.28
Bottleneck	112	(4, 3)	109.2	46.8	78	124.8	358.8	0.29
Military	400	(42, 56)	343.3	140.4	0	31.2	514.8	0.37
Office	700	(218, 235)	561.6	205.2	62.4	327	1156.2	0.72

number of potential bottlenecks in the environment and leading to simpler LP problems. In general, the running time is inversely proportional to the size of the time step. This is confirmed by a number of experiments we conducted in different scenarios and for different values of Δt. In our example scenarios, the time step Δt was set to 1 s with the exception of the office scenario. Here, to obtain running times that are sufficiently low, we sacrificed the optimality of the LP solution and set $\Delta t = 2$ s.

7 Conclusion and Future Work

We presented a novel approach for planning and directing groups of virtual characters by combining techniques from Integer Linear Programming with an agent-based steering framework. We have demonstrated the applicability of our method through a wide range of challenging scenarios. Despite the computational complexity raised by the ILP formulation, our system runs at interactive rates by using the column generation algorithm and choosing an appropriate discretization of time.

Our approach makes some simplifying assumptions. Most notably, we assume that the characters choose paths so as to minimize their average traveling time. However, in real-life, other parameters can also affect the path choices that people make, such as the safety of the area that the path crosses, its attractiveness, etc. We should also point out that our system is specifically designed to model large groups. While in-group members can have different behavior characteristics, during the ILP planning, we assume that they prefer to move with the same desired speed. This enables our method to account for dynamic congestion and other time-consuming situations.

Our current work focuses on crowds of virtual characters, but it can be easily generalized to address multi-robot planning and coordination problems. Other virtual environment applications can benefit from it as well. An interesting extension, for example, would be to use our formulation for traffic simulation. In the future, we would also like to address dynamic and interactive environments, since currently we assume that the virtual world remains static and its dynamics are perfectly known a priori. Note, though, that in dynamically changing environments, the agents may not have the time to plan and replan optimal paths across the time-expanded graphs. To this end, we can trade-off optimality for performance by quitting the column generation algorithm before we have solved the LP-relaxation problem to optimality.

References

1. Bayazit, O.B., Lien, J.M., Amato, N.M.: Better group behaviors in complex environments using global roadmaps. In: Artificial Life, pp. 362–370 (2003)
2. Fleischer, L., Skutella, M.: Quickest flows over time. SIAM J. on Computing 36(6), 1600–1630 (2007)
3. Ford Jr., L.R., Fulkerson, D.R.: A suggested computation for maximal multi-commodity network flows. Management Science, 97–101 (1958)
4. Geraerts, R.: Planning short paths with clearance using explicit corridors. In: IEEE Int. Conf. on Robotics and Automation, pp. 1997–2004 (2010)
5. Geraerts, R., Overmars, M.H.: The corridor map method: Real-time high-quality path planning. In: IEEE Int. Conf. on Robotics and Automation, pp. 1023–1028 (2007)
6. Guy, S.J., Chhugani, J., Curtis, S., Pradeep, D., Lin, M., Manocha, D.: PLEdestrians: A least-effort approach to crowd simulation. In: Symp. on Computer Animation, pp. 119–128 (2010)
7. Helbing, D., Buzna, L., Werner, T.: Self-organized pedestrian crowd dynamics and design solutions. Traffic Forum 12 (2003)
8. Kamphuis, A., Overmars, M.H.: Finding paths for coherent groups using clearance. In: Symp. on Computer Animation, pp. 19–28 (2004)
9. Karamouzas, I.: Motion planning for human crowds: from individuals to groups of virtual characters. Ph.D. thesis, Utrecht University, the Netherlands (2012)
10. Karamouzas, I., Geraerts, R., Overmars, M.: Indicative routes for path planning and crowd simulation. In: Foundations of Digital Games, pp. 113–120 (2009)
11. Karamouzas, I., Heil, P., van Beek, P., Overmars, M.H.: A Predictive Collision Avoidance Model for Pedestrian Simulation. In: Egges, A., Geraerts, R., Overmars, M. (eds.) MIG 2009. LNCS, vol. 5884, pp. 41–52. Springer, Heidelberg (2009)
12. Karamouzas, I., Overmars, M.: Simulating and evaluating the local behavior of small pedestrian groups. IEEE Trans. Vis. Comput. Graph. 18(3), 394–406 (2012)
13. Kwon, T., Lee, K.H., Lee, J., Takahashi, S.: Group motion editing. ACM Trans. on Graphics 27(3), 1–8 (2008)
14. Lee, K.H., Choi, M.G., Hong, Q., Lee, J.: Group behavior from video: a data-driven approach to crowd simulation. In: Symp. on Computer Animation, pp. 109–118 (2007)
15. Li, T.Y., Chou, H.C.: Motion planning for a crowd of robots. In: IEEE Int. Conf. on Robotics and Automation, pp. 4215–4221 (2003)
16. Patil, S., van den Berg, J., Curtis, S., Lin, M.C., Manocha, D.: Directing crowd simulations using navigation fields. IEEE Trans. Vis. Comput. Graph. 17, 244–254 (2011)
17. Peng, J., Akella, S.: Coordinating multiple robots with kinodynamic constraints along specified paths. Int. J. of Robotics Research 24, 295–310 (2005)
18. Peters, C., Ennis, C.: Modeling groups of plausible virtual pedestrians. IEEE Computer Graphics and Applications 29(4), 54–63 (2009)
19. Reynolds, C.W.: Flocks, herds, and schools: A distributed behavioral model. Computer Graphics 21(4), 24–34 (1987)
20. Sánchez, G., Latombe, J.C.: Using a PRM planner to compare centralized and decoupled planning for multi-robot systems. In: IEEE Int. Conf. on Robotics and Automation, pp. 2112–2119 (2002)
21. Silver, D.: Cooperative pathfinding. In: Artificial Intelligence and Interactive Digital Entertainment, pp. 117–122 (2005)
22. Simeon, T., Leroy, S., Laumond, J.P.: Path coordination for multiple mobile robots: A resolution-complete algorithm. IEEE Trans. on Robotics and Automation 18(1), 42–49 (2002)

23. Singh, S., Kapadia, M., Hewlett, B., Reinman, G., Faloutsos, P.: A modular framework for adaptive agent-based steering. In: ACM Symp. on I3D, pp. 141–150 (2011)
24. Sung, M., Kovar, L., Gleicher, M.: Fast and accurate goal-directed motion synthesis for crowds. In: Symp. on Computer Animation, pp. 291–300 (2005)
25. Treuille, A., Cooper, S., Popović, Z.: Continuum crowds. ACM Trans. on Graphics 25(3), 1160–1168 (2006)
26. van den Akker, M., Geraerts, R., Hoogeveen, H., Prins, C.: Path Planning for Groups Using Column Generation. In: Boulic, R., Chrysanthou, Y., Komura, T. (eds.) MIG 2010. LNCS, vol. 6459, pp. 94–105. Springer, Heidelberg (2010)
27. van den Berg, J., Guy, S.J., Lin, M.C., Manocha, D.: Reciprocal n-body collision avoidance. In: Int. Symp. on Robotics Research, pp. 3–19 (2009)
28. van den Berg, J., Lin, M., Manocha, D.: Reciprocal velocity obstacles for real-time multi-agent navigation. In: IEEE Int. Conf. on Robotics and Automation, pp. 1928–1935 (2008)
29. van den Berg, J., Overmars, M.H.: Prioritized motion planning for multiple robots. In: IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, pp. 2217–2222 (2005)
30. van den Berg, J., Overmars, M.H.: Roadmap-based motion planning in dynamic environments. IEEE Trans. on Robotics and Automation 21, 885–897 (2005)
31. van den Berg, J., Patil, S., Sewall, J., Manocha, D., Lin, M.C.: Interactive navigation of multiple agents in crowded environments. In: ACM Symp. on I3D, pp. 139–147 (2008)