Multi-Unit Tactical Pathplanning

Haiqing Wang, Omar N. Malik, and Alexander Nareyek

Abstract—**The task of pathfinding has received considerable attention in research as well as practice. Most of the approaches focus on minimizing the length of the path as well as optimizing the efficiency of the algorithm. Less attention has been paid to the problem of incorporating rich global conditions that require more than simply reaching a specific location. In this paper, we analyze elements of richer pathplanning problems, propose an enhanced specification format, and provide a first simple mechanism to solve such problems.**

I. Introduction

Pathfinding problems play an important role in many application areas, ranging from electronic gaming and military simulations to robotics and logistics. Popular pathfinding algorithms include A* [1], D* [5], and the use of simple look-up tables [7] for smaller problems. In general, the problem can be cast as finding the shortest route between a start and a target node within a network of connected nodes.

In many situations, the resulting length of a path, however, is only one of many considerations (or not important at all), and the path needs to serve a much more complex context. Especially in gaming, where we are setting up and manipulating a stage for a player, the movement of non-player characters has to follow complex interaction patterns. For example, we might want to let unit A, B and C trap unit D in region X for 20 seconds, and thereafter, have unit A and B move away from region X, and let unit C follow unit D while keeping a distance of 30 meters from D. Such problems cannot easily be mapped into existing pathfinding algorithms.

In the following, we will analyze how to best specify these more complex problems, discuss solving strategies, and demonstrate a simple solving mechanism. Generally, we assume that we have centralized control over the units to be positioned – in contrast to problems of multi-agent problem solving.

A. Nareyek is with the Department of Electrical & Computer Engineering, National University of Singapore, Singapore 117576 (phone: +65 9223 1756; fax: +65 6779 1103; e-mail: alex@ai-center.com).

H. Wang (e-mail: wang.haiqing@alumni.nus.edu.sg) and O. N. Malik (e-mail: u0509635@nus.edu.sg) were students at the department and recently graduated.

This work was supported by the Singapore National Research Foundation Interactive Digital Media R&D Program under research grant NRF NRF2007IDM-IDM002-051, as well as by a joint grant of Singapore's Ministry of Education / Academic Research Fund and the National University of Singapore under research grant RG-263-001-148.

II. Problem Specification

A specification could for example be based on a general temporal logic and be solved by a general-purpose problem solver. Indeed, there are some approaches in this direction that mainly focus on robotics, but are nonetheless slightly related to the tactical pathplanning problems discussed here (e.g., see [2] – even though the richness of specifications of their approach is seriously limited). Such a solution, however, comes at great computational costs, and the high-performance real-time environments of most games are completely out of range. Similarly, at least conventional action-planning approaches are not particularly promising for such domains as time/distance is not the main optimization criterion [3] and a lot of specialized data structures and heuristics can potentially be exploited for the path computations.

In [6], an approach inspired by board games is used to plan paths of a set of units, utilizing a form of minmax optimization and decomposition into a dynamic hierarchy of subsystems. Each subsystem can be tackled independently to construct intra-paths (paths within a subsystem) and inter-paths (paths between subsystems) to reduce computation time and make it usable in real-time situations. However, again, hardly any interactions between units can be expressed and planned for.

Just like in real-world robot movement problems, many gaming applications face the problem of planning movements of units without collisions. Solutions include [4], but avoiding collisions is only a very small fraction of the functionality that we are trying to achieve.

The question then arises what the basic functionality components are that should be captured. In this section, we will first look into sample case scenarios and then abstract and consolidate these use cases to define a specification format.

A. Case Examples

Character actions and interactions often translate into movements. The relation of static objects, other movable units, and regional/terrain features needs to be considered for such movements. Considering situations that represent combat as well as storytelling situations, some case examples are:

- A villager is fleeing from a monster. He needs to reach a distance of at least 1,000 meters before sunset so he can rest in safety.
- Several soldiers want to trap an enemy in a particular

region and catch him. The soldiers need to split and move toward the enemy from all directions to make sure the enemy cannot escape. The soldiers cannot venture more than 10,000 meters away from their base because they have limited supplies.

- A group of students is exploring a national park and want to get to a specific lake. They need to keep away from a dangerous cave that is on their path.
- A boy and a girl are in love and stay together wherever they go.

Scenarios with intentionally random movements, like monsters hit by a confusion spell, are not considered here because a realization does not require advanced movement planning.

B. Consolidation into a Specification Format

From the case examples, we can derive some general features to describe potential scenarios.

Movement Types: Movements are related to at least one specific position or object. We call such a position a *reference position*. For example, for the villager, the reference position is the monster, and for the students, the reference position is the lake. The reference position can be static or dynamically changing. We break up an overall situation description as given in the case examples into a set of such movements.

The second important feature is the relative change of a unit's position toward the reference position. We call this the *change of relative distance*, distinguishing between 'increasing', 'decreasing' and 'maintaining'.

According to the type of reference position and change of relative distance, we define six types of basic movement specifications. This is shown in Table 1, with a graphical representation in Fig. 1.

TABLE I
MOVEMENT TYPES

	Reference Position	
Relative Distance	Static	Dynamic
Increasing	LEAVE	ESCAPE
Decreasing	REACH	INTERCEPT
Maintaining	MONITOR	FOLLOW

Parameters: There are a number of more specific parameters that help fleshing out the details of a movement specification:

- *Time:* A pair of time points, providing a time limit range until when the movement has to be completed.
- *Distance:* A pair of numerical values, specifying the allowed minimal and maximal distance to the reference position. For movement types with changing distances, this range is to be reached at the end of the movement.
- *Direction:* A pair of numerical values, specifying the minimal and maximal angle from the reference position at which a unit is to be positioned at the end of the movement. A graphical example of the direction range is shown in Fig. 2 with a reference position R, and unit A within [90, 180] from R, and unit B within [270, 360] from R.

For example, the villager needs to escape from the monster before sunset (time factor), and the distance from the monster should be at least 1,000 meters (distance factor). In the trapping example, each solder needs to approach the enemy from a specific direction (direction factor).

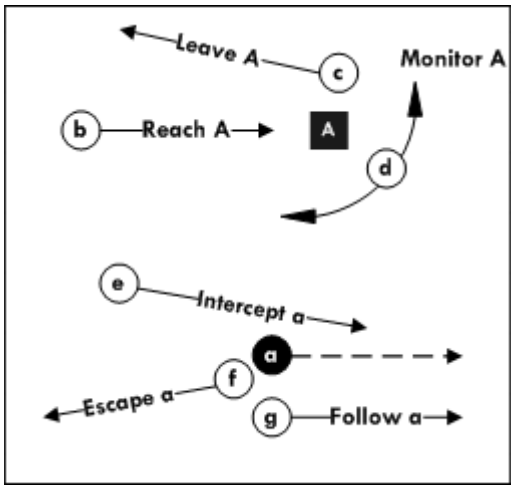

Fig. 1. Graphical Representation of Movement Types. The filled square is a static reference position, and the filled circle is a dynamic reference position. The other circles (b, c, d, e, f, g) are moving units, and the arrows represent their paths.

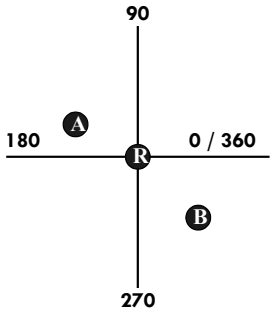

Fig. 2. Directional Conditions.

Interaction Restrictions: In many cases, conditions to stay together or to stay away from each other apply for more than only two units. For example, the group of solders should spread out and keep away from each other when they try to trap the enemy. This can theoretically be modeled by 'monitor' movement types for every pair of units, but

because of the exponential scaling for a larger number of units, we introduce a special type of *interaction restrictions*.

Interaction restrictions apply to a set of units, can be of type TOGETHER or AVOID, and take the same parameters as the basic movement types.

Terrain Restrictions: A set of units or movements can be constrained to avoid or only stay in specific areas. Again, a temporal range is introduced to factor in conditions that only last for a subset of complete movements.

For example, the soldiers should not be away more than 10,000 meters from their base, and the students should not go to the dangerous cave.

C. An Example Specification

We use a variation of the case examples here to demonstrate an example with a variety of different specifications. Consider the following description:

> A monster is moving towards a town. A villager (speed 3) is escaping from the monster (be away from the monster with a distance of at least 5 in 3 minutes) and tries to reach the town nearby (be within distance 8 of town in 3 minutes). There is a city guard (speed 4) who needs to return to the town (be within distance 3 to 5 in 3 minutes) to keep the region safe. There is a dangerous cave (map position <1,1>) in the north-west corner, and the villager should avoid the area within a distance of 2 around cave. The villager and the guard do not like each other, and always keep some distance (with a distance of at least 4).

The path of the monster is fixed. The problem is to plan and find paths for the villager and city guard. According to the description, the scenario can be specified as:

Movements Specifications:

1 Villager: {REACH(town), distance[0,8], time[3,3], direction[0,360]}
2 Villager: {ESCAPE(monster), distance[5,∞], time[3,3], direction[0,360]}
3 City guard: {MONITOR(town), distance[3,5], time[3,3], direction[0,360]}

Interaction Restrictions:

4 {Villager, Monster}: AVOID, distance[1,∞], time[0,∞], direction[0,360]
5 {Villager, City guard}: AVOID, distance[4,∞], time[0,∞], direction[0,360]

Terrain Restrictions:

6 Villager: AVOID, area(position<1,1>, range[2]), time[0,∞]

A graphical representation of the initial scenario is shown in Fig. 3.

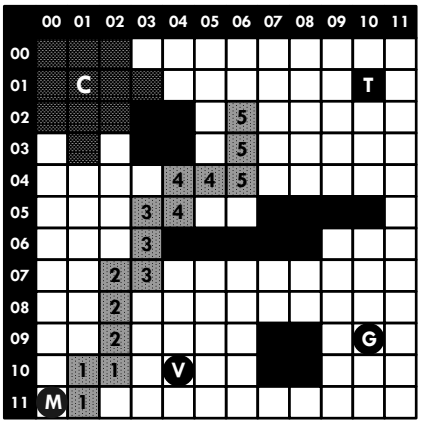

Fig. 3. Graphical Representation of Initial Scenario.

III. A Simple Solving Strategy

Solving problems that are given in the specification format, i.e., finding paths for the single units that fulfill the given restrictions, is not a trivial matter. Games typically have very strong real-time requirements, and most certainly there is no single solution that suits all platforms and game environments.

As an initial study, we adopt a relatively brute-force approach. We will first find all potential end positions for the units' movements, then filter these positions according to various restrictions, apply an enhanced pathfinding algorithm to find actual solution paths for one unit after the other, and overlay this procedure with a backtracking mechanism that guarantees completeness.

A. Critical Positions

In the first step, we search for all potential end positions for the movements of each unit. We call these end positions 'critical positions' in the following.

Potential critical positions can be calculated by the reference position, the movement type, the distance factor and the direction factor. For example, to meet specification 2, potential critical positions have a distance greater than 5

from the monster at time 3. The nodes with a distance of 5 from the monster are the minimum requirement to meet this ESCAPE specification. This means that all positions lying outside a 5 meter radius from the monster's position at time 3 will be potential critical positions for the villager.

In a similar manner, critical positions for all 6 movement types can be calculated. In general, according to a movement specification with a reference position R and parameters distance[d1,d2], time[t1,t2], and direction[D1,D2], the critical positions are computed as all the positions within a distance d1 and angle of D1 and D2 from the position of R at time t1 in cases of REACH and INTERCEPT. For ESCAPE and LEAVE specifications, they are positions lying outside the circle of radius d1. In cases of MONITOR and FOLLOW, the critical positions lie between two circles of radii d1 and d2 respectively from the position of R at time t1. After the computation of the critical positions for all movement specifications and units, we filter these targets according to a subset of the restrictions.

B. Filtering

Positions that are too far for a unit to reach practically (according to pure Euclidean distance) based on the time available, and those lying in AVOID regions are removed from the critical position set. The resulting set is called the **Valid Critical Position (VCP)** set, and is passed over to the pathfinder for pathfinding. If no critical positions are left for a unit after the filtering process, the algorithm aborts and returns failure.

C. Sequentialization

As handling rich global constraints makes the pathfinding process computationally expensive, a heuristic is used to assign priorities to the units. The pathfinding algorithm then plans paths for units sequentially based on this priority, and the already existing paths are taken as constraints for the units' paths that still need to be planned. To guarantee completeness, in case of unsuccessful pathfinding, a backtracking mechanism over the sequence/priority order is triggered. The heuristic assigns priorities based on the time that a unit has available to plan its path (most constrained first).

D. Enhanced Pathfinding

An enhanced version of the A* algorithm is used to search for valid paths sequentially for all the units. The enhancements have been made to handle the scenario specifications and to further filter out invalid constellations. The enhancements are discussed below.

Planning Node: As unit paths need to be evaluated with respect to a specific node, a planning node is identified at the beginning of the pathfinding process. This planning node belongs to the Valid Critical Position set of a unit.

For example, in the case of the guard monitoring the town centre, its valid critical positions will be all the positions lying between two circles of radii 3 meters and 5 meters respectively from the town centre (assuming these positions do not lie in AVOID regions and the guard can get to them based on its speed). However, the planning node will be the town centre lying at the centre of the surrounding valid critical positions. It is with respect to this one planning node that the path of the unit will be planned.

As the valid critical positions surround the planning node, if a unit can get to its planning node, it can also get to one of the valid critical positions. In the case of the MONITOR, FOLLOW, and REACH specifications, there will always be one planning node that will be available in the specification.

However, for the LEAVE and ESCAPE specification, this node will not be available. Therefore, one node that belongs to the VCP set of the unit and is within its *reachability radius* (positions that units can get to based on their speed) will be randomly assigned. As a unit will be unable to get to positions outside this radius, there is no point in finding a path to them. Furthermore, for the INTERCEPT scenario, one unit's path will act as potential planning node for the other unit (as long as it is within the unit's reachability radius).

For LEAVE, ESCAPE, and INTERCEPT specifications, if no path can be calculated for a planning node, a backtracking mechanism over all possible planning nodes needs to be applied. However, since valid critical positions surround the planning node, there is a good chance that one of these positions will be reached while calculating a path to the current planning node, thereby fulfilling the scenario specification.

Actual Travel Time: During the pathfinding process, the length of the path can be computed accurately and the actual travel time is known (in contrast to the previous filtering by Euclidean distance). If the actual travel time is longer than the time limit, this path fails to meet the specification and the plan will be marked as failed.

Terrain Restrictions: According to the terrain restrictions, the same node may be accessible for some time and inaccessible for some other time for a unit. Similar to the D* algorithm, a time attribute is added to every node to deal with the dynamic changes of the map. This time attribute stores the availability of a node for every time period. If the node is not accessible at a requested time, the search process will not incorporate that node into a path.

Collision Avoidance: Searching for a path with collision avoidance is similar to finding a path with terrain restrictions. If a unit occupies a node at a certain time, the node is marked as inaccessible at that particular time.

Interaction Restrictions: By considering the position of the other units at a particular time, it is possible to check the interaction restrictions during the pathfinding process. For

example, if the next node is too close to another unit with an AVOID interaction restriction at a certain time, this node will be considered inaccessible.

As an illustration of this dynamic filtering, for specification 5 of the example, Fig. 5 shows a situation in which a path for the villager is already found and a path for the city guard to position <4,8> is searched for. When node <4,9> is evaluated, the algorithm will find that this position is too near to the villager (less than distance 4) and the city guard cannot move to it (even if this is a shorter path to the target) but has to move to <3,10> instead.

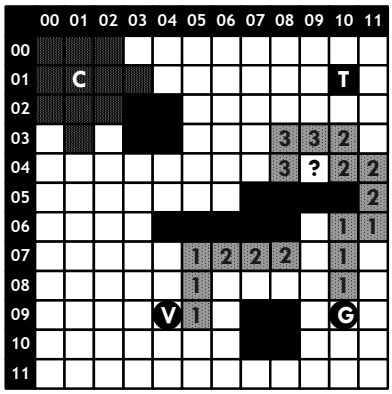

Fig. 5. Illustration of an Interaction Specification Check.

IV. Empirical Analysis

In the following, we provide some empirical data and analysis of run times to get a rough understanding of the behavior of our solving strategy. The results are obtained by using a PC with an Intel(R) Core(TM) 2 Duo Processor T5750 (2.00 GHz, 2.00 GHz) and 2 GB RAM. The first level Pac-Man map (with a dimension 29×26) is used as pathplanning environment. Random specifications were applied to three randomly placed units on the map.

To study the efficiency of the algorithm in a variety of cases, various test cases were considered (using a total of 100 scenarios). For one third of the test cases, a high specification time bound was used to make sure that units find a path in the first priority order. Another third of the test runs uses medium time bound that makes sure that backtracking is triggered. The last third uses a very low time bound such that a failed result was returned.

Time to Find Valid Critical Positions: The time to calculate Valid Critical Positions was very short, i.e., an average of 68 ms was needed (with an average of 38 valid critical positions in each case).

Time to Find Paths without Interactive Specifications: In this part of the experiment, an average of 6 movement and 3 environmental specifications were applied on the units. The results are reflected in Table 2 and Fig. 6.

TABLE 2
Pathfinder Experiment Results – Case A

Best Case Calculation Time(s)	0.18
Average Calculation Time(s)	0.89
Worst Case Calculation Time(s)	3.53

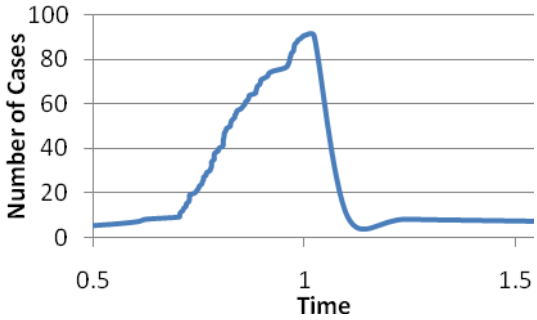

Fig. 6. Distribution of Test Cases against Pathfinding Time (in seconds) – Case A.

The worst cases clearly show the impact of our test cases with very low bounds and full search space exploration before returning a fail. Moreover, in all the worst cases, the movement specifications turn out to be predominantly of type LEAVE, INTERCEPT, and ESCAPE. These cases involve a large number of potential planning nodes. However, for the cases that include only REACH, MONITOR, and FOLLOW specifications, the pathfinding time is under 0.5 seconds.

The number of backtracks being made was entirely random since it largely depended upon the type of path being planned for a unit and the number of conflicts in the path. On average, the pathfinding calculation time took between 0.7 to 1.1 seconds for the majority of the cases (83.0%).

Time to Find Paths with Interactive Specifications: In this part of the experiment, 3 interactive specifications were added to each of the 100 test cases (Table 3 and Fig. 7).

TABLE 3
Pathfinder Experiment Results – Case B

Best Case Calculation Time(s)	1.34
Average Calculation Time(s)	3.39
Worst Case Calculation Time(s)	7.72

The number of backtracks increased by a large factor because many more conflicts were encountered with the interactive specifications in place. As before, calculation times increased substantially for scenarios with LEAVE, INTERCEPT, or ESCAPE specifications.

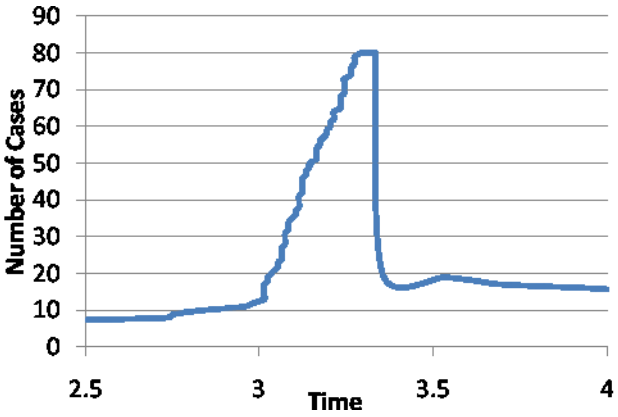

Fig. 7. Distribution of Test Cases against Pathfinding Time (in seconds) – Case B.

Discussion: When interactivity constraints are added, the algorithm efficiency is substantially affected. However, further tests revealed that if only one interactivity constraint is used, the pathfinding time hardly increases (average run times of 1.2 seconds). Pure pathfinding times also increase with an increase in the number of movement specifications. however, this increase in time is not particularly significant.

If specifications of type LEAVE, ESCAPE, or INTERCEPT are predominantly applied on the units, the pathfinding time increases strongly since more paths need to be evaluated with respect to the greater number of planning nodes. On the other hand, the pathfinding time is minimal in the case of only MONITOR, REACH, and FOLLOW specifications since there is only one planning node involved.

V. Conclusion

We have introduced the problem of tactical pathplanning, which tackles more complex and richer scenario descriptions for unit movements. In contrast to a simple fixed script of path sequences, this leads to much more dynamic flexibility for designing game scenarios.

The specification system breaks a scenario into simpler movements and additional restrictions. The basic components are intuitive and easy to understand for people without much technical background, such as game designers. Depending on the specific game, there may be additional factors and conditions that should be part of a specification. Our specification format is thus not meant to be the Holy Grail for specifying tactical pathplanning problems, but more a guideline and inspiration to tackle these problems.

We also provided a simple solving strategy that is using the concept of critical positions and following filtering and pathfinding. Hardly any optimizations were used, which results in a behavior that is hardly acceptable for real-time applications. The actual example implementation was however only meant as a feasibility study, and represents only the starting point for our future work to efficiently tackle this class of problems. For example, the current combination of pathfinding and backtracking is done in a highly uninformed way where the information gained in previous backtracking cases is not retained and the overall pathfinding is basically restarted from scratch every time a conflict is encountered. We expect that more intelligent approaches will have a very significant impact. Better heuristics need to be explored as well.

Generally, however, this problem is simply not as "easy" as the regular shortest-path problem in terms of complexity but comparable to action-planning problems. Given the real-time requirements of domains like computer games, future approaches will thus likely abandon algorithm features like completeness.

With respect to specifications, potential stopping of units (at objects like benches or for collisions) should be considered in the future as well. Another important topic is that of dynamics, i.e., efficiently manipulating the solution when parts of the specification change over time instead of re-computing a solution from scratch. Furthermore, the handling of partial satisfaction of the specification conditions, and analysis methods to return suggestions on which conditions should be relaxed in case no plan can be found, are to be explored as well.

References

[1] P. E. Hart, N. J.Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics* 4(2), pp. 100-107, 1968.

[2] M. Kloetzer and C. Belta, "LTL Planning for Groups of Robots," in *Proceedings of the IEEE International Conference on Networking, Sensing, and Control* (ICNSC), pp. 578-583, 2006.

[3] A. Nareyek, "Beyond the Plan-Length Criterion," in A. Nareyek (ed.), *Local Search for Planning and Scheduling*, Springer LNAI 2148, pp. 55-78, 2001.

[4] D. Silver, "Cooperative Pathfinding," in S. Rabin (ed.), *AI Game Programming Wisdom 3*, Charles River Media, pp. 99-111, 2006.

[5] A. Stentz, "Optimal and Efficient Path Planning for Partially-Known Environments," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '94)*, pp. 3310-3317, 1994.

[6] B. Stilman, "Multiagent Air Combat with Concurrent Motions," *Symposium on LINGUISTIC GEOMETRY AND SEMANTIC CONTROL, Proceedings of the First World Congress on Intelligent Manufacturing: Processes and Systems*, pp. 855-867, Mayaguez, Puerto Rico, Feb. 1995.

[7] W. van der Sterren, "Path Look-Up Tables – Small is Beautiful," in S. Rabin (ed.), *AI Game Programming Wisdom 2*, pp. 115–129, Cengage Delmar Learning, 2003.