MULTI-GOAL PATH-FINDING FOR AUTONOMOUS AGENTS IN VIRTUAL WORLDS

Philippe Codognet
University of Paris 6, LIP6, case 169, 8 rue du Capitaine Scott, 75 015 Paris, France, Philippe.Codognet@lip6.fr

Abstract: We are interested in complex path-planning, where several goals have to be satisfied by an autonomous agent. The agent has to pass through several destination points in a virtual environment, minimizing some criterion (e.g., the length of the overall trajectory), we call this problem multi-goal path-finding. The problem appears for artificial creatures in virtual environments and for computer-controlled characters in 2D or 3D games, which have to exhibit so-called opportunistic behaviors. We present a simple framework and an effective algorithm based on goal reordering using local search techniques.

Key words: Interactive entertainment, artificial intelligence, game AI, autonomous agents, virtual reality, path-finding, constraint solving.

1. Introduction

Our aim is to design a high-level language for describing behaviors for autonomous agents in virtual worlds, together with an efficient run-time algorithm to implement those behaviors. Virtual worlds are becoming increasingly popular due to the definition of standard formats for describing 3D scenes on the web (e.g., VRML97, Java3D and X3D) but also, in the computer games community, because of the availability of tools to design new maps and agent behaviors for popular 3D action games, such as Unreal, Half-life, or Quake, not to mention general tools, such as Discreet's Gmax. Nevertheless, it is difficult to "populate" such worlds with virtual agents

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35660-0_65
R. Nakatsu et al. (eds.), *Entertainment Computing*

representing life-like creatures which could autonomously navigate and react to their changing environment, and also possibly interact with users. Path-planning problems are thus a key issue in the so-called "Game AI" domain [2], and have been tackled up to now with traditional AI techniques such as the classical A* algorithm or more modern extensions [1], heavily relying on the complete knowledge of the complete virtual environment to define an optimal trajectory for computer-controlled characters. Such path-finding is usually limited to a single goal point and basically amount to compute an optimal or near-optimal trajectory avoiding collision with known obstacles. We are interested in this work in more complex path-planning, where several goals have to be satisfied by the autonomous agent, which has thus to pass through *several* destination points in the virtual environment, and we call this problem *multi-goal path-finding*. This problem appears in a virtual environment or 3D game when, for instance, the user ask some helper agent to go and pick several items and to meet him at some further point. This could also be the case for an autonomous creature who has to "survive" in an unknown environment and for instance look for food but also for water, and so on. A last example in the First-Person Shooting Games (FPSG) setting would be a virtual character who has to take some ammunitions, one or several medikits to recover from injuries and a few energy packs to be able use some defensive device. The order in which these goals have to be satisfied could also be constrained by some temporal precedence constraints (e.g., point A should be visited before point B). Moreover, one will usually have a criterion to satisfy together with the goals to be achieved, and therefore some optimality issue arises: for instance, one will try to minimize the overall time to complete the goals, or to minimize the proximity of danger, etc. If the virtual environment is discretized in small areas, this problem could in fact be seen as an instance of the well-known Traveling Salesman Problem (TSP), where one has to find the shortest Hamiltonian path in a complete graph (that is, a path visiting all nodes only once). Many algorithms have been proposed to solve the TSP, and the most successful approaches seem today to be based on Local Search techniques [3]. We will thus propose a general framework for solving the multi-goal path-finding problem based on a new local search algorithm that we call *Adaptive Search*. This method, although simple and thus easy to implement and integrate in the core behavior of autonomous agents, is nevertheless rather efficient to solve complex combinatorial problems [4]; it extends our previous work on single-point path-finding [5] for autonomous agents in virtual worlds.

2. Path-Finding without Maps

The usual path-finding algorithms are based on some variants of the A* algorithm, introduced in game applications by [1], or its more modern extensions such as LTRA* [6]. Today, all games include some refinement of the A* algorithm for defining the trajectories of the computer-controlled characters. However, such an algorithm can only be used if one has a global view of the environment, that is a full map including the obstacle and the domain costs in order to compute the best (shortest) trajectory between two points. It is not possible to apply this method in the context of autonomous agents since they perceive only a limited part of environment and act accordingly. In the last decade many AI researchers adopted the notion of *reactive agent* originated from robotics in the late 1980's. We use a definition given by R. Arkin, one of the pioneers of this approach : "A reactive robotic system couples perception to action without the use of intervening abstract representation or time history". Reactive agents are thus simple entities that receive percepts from their environment and can act on the environment by performing actions through some effectors/actuators. The simplest one is to issue some motor command to navigate effectively in the external or virtual world. Reactive agents have no symbolic model of the world they live in, but rather use sensory-action control loops in order to perform tasks in a robust manner.

In this context, we previously defined in previous work a path-finding algorithm for reactive agents [5] that produces acceptable (life-like) if not optimal trajectories. Moreover, the method is fully reactive to changing environments (e.g., moving obstacles and moving goals), which is not so for A*-based techniques. What could thus be done for multi-goal path-finding? How is it possible to plan intelligently to reach certain goals if the agent has only a limited knowledge of the environment?
The core ideas are:

1. to develop a heuristic function that is able to compute, from the current situation, some approximated degree of satisfaction of the goals (e.g., the distance still needed to be walked to reach some goal point);
2. to update and refine these heuristic functions as the agent explores the environment and discovers new information, such as the position of some obstacles. Then the necessity to perform some detour exists.

Let us consider the simple goals of the form $go(A_1)$, ..., $go(A_N)$, where A_1,..., A_N are the locations to be reached. Moreover let us perform some goal actions, and let A_0 be the current position of the agent. A heuristic function for the goal $go(A_i)$, called an *error function* [5], is defined as the air-distance

between A_0 and A_i , noted $|A_0\ A_i|$. This is obviously an optimistic approximation, as there might be some obstacles in the route between A_0 and A_i and therefore the actual distance on the trajectory taken by the agent will be larger. But this heuristic will be used to guide the agent towards the most direct route to the goal whenever possible, that is, when there is no obstacle. As detailed in [5], at each time step the agent will compute the cost of the heuristic function for each neighboring position (excluding positions obstructed by obstacles) and choose to move to the position that has the best (smallest) heuristic cost. Observe that the heuristic (air-distance) is computed at each time-step, and thus a moving target position is handled with no modification of the algorithm, which is therefore fully reactive.

3. Multi-Goal Path-Finding

The notion of a constraint representing goals that the agent is trying to achieve is the basis of our framework. The agent thus maintains a set of goal constraints. Now we define a simple but effective action selection mechanism that selects at each time-step the best action to reduce the discrepancy between the current state and the overall satisfaction of the goals. It is worth noticing that behaviors are stated in an implicit way (by giving a set of constraint goals) and not in an explicit way (e.g., by giving a precise trajectory), this makes it possible to adapt reactively the agent behavior to a changing real-time environment.

3.1 Basic Method

Consider now the initial problem of finding the best (shortest) path linking point $\{A_0,..., A_N\}$ in the virtual environment, where A_0 is the current position of the agent and $\{A_1,..., A_N\}$ are the destination goals to be reached. This could be rephrased as finding a permutation sequence of $\{A_1,..., A_N\}$ that minimizes the length of the overall trajectory, that is the sum of the distances between two consecutive goal points of the trajectory :

$$\text{distance}(A_1,..., A_N) = |A_0\,A_1| + |A_1\,A_2\,| + ... + |A_1\,A_N\,|$$

A configuration state in our search space is defined as a permutation of $\{A_1,...,A_N\}$ corresponding to a path visiting the goals in a certain order. This could be represented as a combinatorial problem by having N variables $\{A_1,..., A_N\}$ with domains of possible values $\{1,..., N\}$, together with the constraint that all variables are assigned to distinct values, modeling this problem in terms of a Constraint Satisfaction Problem (CSP). This could also be represented as a local search problem for which the Adaptive Search

algorithm can be used [4]. Hence we have a second application of this agent-based search algorithm in finding, among all possible permutation sequences, the best overall trajectory. The Adaptive Search algorithm relies on the iterative-repair method, which is based on cost and goal constraint heuristic information. For this algorithm we define a set of "neighbors" of the current configuration, which consists of all configurations obtained by swapping two elements within the current (ordered) configuration. The basic idea of iterative improvement is to select the neighbor which minimizes the total cost; we continue until an optimal value is found. Our method includes an adaptive memory too (as in Tabu Search) in order to avoid being trapped in local minima. The basic idea is as follows : as soon as a configuration is discovered to be a local minimum (i.e. it has a heuristic cost smaller than all its neighbors, but does not still satisfy the goal completely) it is marked as "Tabu" for a certain number of iterations and could not be chosen as the next action to perform. Hence another action, viz. one that will degrade the current value of the heuristic function, will be chosen making the agent escape the dead-end.

3.2 An Example

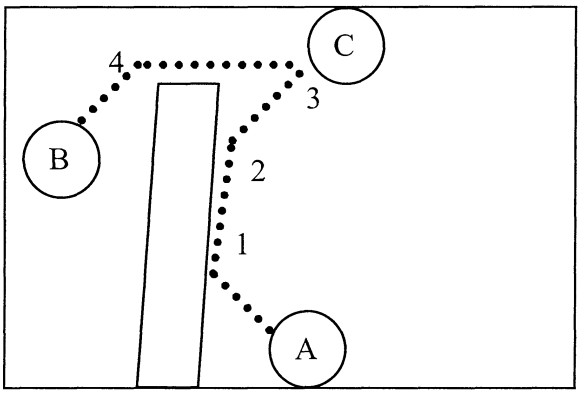

Figure 1. An example of opportunistic behavior

The agent *A* has goals *go(B), go(C)*. As B is closer to A than C, the agent chooses to perform the goal sequence (B,C) and goes towards B by the most direct route. At point (1) it discovers a wall and starts a detour going right. At point (2), it realizes that C is closer than B and changes its goal sequence to (C,B), exhibiting an opportunistic behavior. It then reaches C (point 3) and reduces it goal sequence to (B). The agent then goes directly to B after

avoiding the wall (point (4)). If other obstacles would have been in the route to B, it would have detour accordingly.

3.3 Multi-Goal Path-Finding Algorithm

We consider a discretized trajectory for an agent A which is moving from a position to another with a given time step δt. For simplicity, we consider the agent to have a constant speed and moves from a given distance d at each time step. We assume that the agent only perceives its immediate environments, that is, the character of the domain (clear or obstacle) for a finite set of neighboring positions around him (4 for a square grid, 6 for an hexagonal grid, or in general case n, depending on the precision of the underlying map). In the case of a continuous map, we assume that the agent will sample a finite set of positions around him.

In the algorithm of figure 2 we have an agent at location A_0, a sequence of goal positions $S = (A_1,..., A_N)$, and the heuristic function :

$$h(A_0,S) = |A_0 A_1| + |A_1 A_2| + ... + |A_1 A_N|$$

A_0 = "agent position"
$S = (A_1,..., A_N)$

while $S \neq \varnothing$
  if $A_0 = A_1$ **then** $S = (A_2,..., A_N)$
  compute $N = \{ N_1 , \dots , N_p \}$, the set of neighboring positions from A_0
  forall $N_k \in N$
    if N_k is not Tabu and is not an obstacle
    then compute $h(N_k , S)$
    else $h(N_k , S) = +\infty$
    compute S_k = {sequences $S_k(i)$ / $S_k(i)$ is a permutation of S
                with A_1 and A_i swapped }
    forall $S_k(i) \in S_k$
        compute $h(N_k , S_k(i))$
  choose $S' \in S_k$ and $N_0 \in N$ such that $h(N_0 , S_k)$ is minimal
  if $N_0 = A_0$
  then mark N_0 as Tabu
  else move the agent from A_0 to N_0
      $A_0 = A_0$
      $S = S'$

Figure 2. The multi-goal path-finding algorithm.

4. Adding Precedence Constraints

Additional precedence constraints on the goals (e.g., A_i should be visited before A_j) can be easily integrated in this framework by adding a new constraint $A_i < A_j$. It is done by modifying the cost criterion that is associated to each configuration, which has been defined previously as the overall length of the path (from A_0 to A_N). In order to take into account the additional precedence constraints on the goals, such as A should be visited before B, we need to define an "error" function that will give an indication on how much the constraint is violated in the current configuration. For instance the error function associated to an arithmetic constraint of the form $A < B$ will be *max (0, A-B)*. We will add the total sum of the errors on all constraints of the problem to the configuration cost (multiplied by a suitable coefficient) in order to compute a global cost to be minimized. When tuning the suitable coefficients for precedence costs, one could produce different behaviors according to the constraints. It is possible to have "soft" precedence constraints.

5. Path-Finding and Multi-Goal Behaviors

We can generalize the path-finding techniques to more complex behaviors and benefit from a multi-goal optimization. For instance, let us take the well-known game "The Sims" (Maxis, 2000), which is a character-level simulation of everyday life. The user can interfere at anytime with the characters by giving them some actions to perform (e.g. go to sleep, take a shower, go out and meet neighbors, etc.) in order to improve their current situation. So, a character can have a list of "to do" actions imposed by the player, that he will perform on a first-in-first-out basis. However, some optimization could be achieved by reordering the goals in the "to do" list. For instance, executing two goals in one sequence requires the agent to be in two separate locations close to each other, then reordering the goals for executing priority may be one of the first actions. There is currently no such possibility in the game, but this could easily be done by integrating a multi-goal optimization strategy based on the framework that we propose, thus generalizing from a path-finding application to a generic goal-based planning problem. More generally, we would like to consider opportunistic behaviors, that is, behaviors that could be triggered by some new perception in the environment and that could temporarily override some current behavior. Thus an artificial character that has to go from point A to point B will temporarily perform a detour towards a food source that has been perceived nearby in order to restore its energy level. However, one has to

tackle the issue of considering a trade-off between reactivity (the ability to take into account new incoming events occurring in the environment and re-plan accordingly if necessary) and an increasing complexity and execution time, which could be a crucial resource in the case of the multi-agent simulation of a large population. An interesting possibility is to define time-windows in which the agent follows a given plan without modification and inform only from time to time the agent on environmental events, viz. at the transition point between two time-windows. Reducing the duration of the time-windows will make the agent more environment-aware and therefore more opportunistic, at the expense of greater execution time devoted to reasoning and (re-)planning.

6. Conclusion

We presented a simple framework and an effective algorithm to integrate in a practical manner multi-goal path-planning for autonomous agents in virtual worlds. The basic idea is to reorder goals using local search techniques in order to optimize a given criterion (e.g., the overall time to achieve the set of goals, or the total distance of the overall trajectory). We considered reactive agents without any previous knowledge of an environment map that could be fully reactive and opportunistic: this is the setting of many games, e.g., intelligent "bots" in action games. The framework could be extended from path-finding to general multi-goal behavior and applied in many character-level simulation games.

References

[1] B. Stout. Smart Moves: Intelligent Pathfinding, *Game Developer*, vol. 3, no. 10, October 1996.

[2] S. Woodcock. Game AI: The State of the Industry. *Game Developer*, vol. 7, no. 8, August 2000.

[3] E. Aarts and J. Lenstra (Eds). *Local Search in Combinatorial Optimization*, Wiley, 1997.

[4] P. Codognet and D. Diaz. Yet Another Local Search Method for Constraint Solving. *First Symposium on Stochastic Algorithms: Foundations and Applications*, K. Steinhöfel (Ed.), LNCS 2246, Springer Verlag, December 2001.

[5] P. Codognet. Animating Virtual Creatures by Constraint-based Adaptive Search, *Proceedings of VSMM2000, 4th Int. Conf. on Virtual Systems and Multimedia*, Gifu, Japan, IOS Press 2000.

[6] S. Koening. Agent-Centered Search. *Artificial Intelligence Magazine,* vol. 22, no. 4, December 2001.