Proceedings of the 33rd
Conference on Decision and Control
Lake Buena Vista, FL - December 1994

Efficient Algorithms for Globally Optimal Trajectories

John N. Tsitsiklis
Laboratory for Information and Decision Systems
M.I.T.
Cambridge, MA 02193, U.S.A.
jnt@mit.edu

Abstract

We present serial and parallel algorithms for solving a system of equations that arises from the discretization of the Hamilton-Jacobi equation associated to a trajectory optimization problem of the following type. A vehicle starts at a prespecified point x_0 and follows a unit speed trajectory $x(t)$ inside a region in $\Re^m$, until an unspecified time T that the region is exited. A trajectory minimizing a cost function of the form $\int_0^T r(x(t))\,dt + q(x(T))$ is sought. The discretized Hamilton-Jacobi equation corresponding to this problem is usually solved using iterative methods. Nevertheless, assuming that the function r is positive, we are able to exploit the problem structure and develop one-pass algorithms for the discretized problem. The first algorithm resembles Dijkstra's shortest path algorithm and runs in time $O(n \log n)$, where n is the number of grid points. The second algorithm uses a somewhat different discretization and borrows some ideas from Dial's shortest path algorithm; it runs in time $O(n)$, which is the best possible, under some fairly mild assumptions. Finally, we show that the latter algorithm can be efficiently parallelized: for two-dimensional problems, and with p processors, its running time becomes $O(n/p)$, provided that $p = O(\sqrt{n}/\log n)$.

1. Introduction

Consider a vehicle that is constrained to move in a subset G of $\Re^m$. The vehicle starts at an initial point x_0 and moves according to $dx/dt = u(t)$, subject to the constraint $\|u(t)\| \leq 1$, where $\|\cdot\|$ denotes the Euclidean norm. At some unspecified time T, the vehicle reaches the boundary of G and incurs a terminal cost $q(x(T))$. We also associate a traveling cost $\int_0^T r(x(t))\,dt$ to the trajectory followed by the vehicle. We are interested in a numerical method for finding a trajectory that minimizes the sum of the traveling and the terminal cost. We assume that $\inf_{x \in G} r(x) > 0$, which forces the vehicle to exit G in finite time.

This problem formulation allows us to enforce a desired destination x_f: for example, we may let $G = \Re^n - \{x_f\}$ and $q(x_f) = 0$. It can also incorporate "hard obstacles"; for example, if a subset F of G corresponds to an obstacle, we can redefine G, by removing F from G and by letting $q(x)$ be very large at the boundary of F.

Related research

Problems of this type have been considered in the computer science literature for some cases where the function r is piecewise constant [M]. They can also be addressed using deterministic optimal control methods but these are only guaranteed to produce locally optimal trajectories [AF, FR]. However, if a globally optimal trajectory is desired, then methods based on dynamic programming are required. The solution to the problem is furnished, in principle, by the Hamilton-Jacobi (HJ) equation. Since an exact solution of the HJ equation is usually impossible, the problem has to be discretized and solved numerically. After discretization, one needs to solve a system of nonlinear equations whose structure resembles the structure of the original HJ equation. This approach raises two types of issues:

(a) Does the solution to the discretized problem provide a good approximation of the solution to the original problem?

(b) How should the discretized problem be solved?

Questions of the first type have been studied extensively and in much greater generality elsewhere - see, e.g. [CD, CDF, CDI, F, GR, KD, S] and references therein. We bypass such questions and focus on the purely algorithmic issues.

The usual approaches for discretizing the HJ equation are finite-difference or, more generally, finite-element methods. The discretized problem is equivalent to solving a *stochastic* optimal control problem for a finite state controlled Markov chain; the number of states of the Markov chain is equal to the number of grid points used in the discretization [KD]. Thus, the discretized problem can be solved by standard methods such as successive approximation or policy iteration [B1]. This is somewhat unfortunate: one would hope that the discretized version of an optimal trajectory problem would be a deterministic shortest path problem on a finite graph, that can be solved efficiently, say using Dijkstra's algorithm. In contrast, a method such as successive approximation can require a fair number of iterations, does not have good guarantees on its computational complexity (because the number of required iterations is not easy to bound), and can be much more demanding than Dijkstra's algorithm. The contribution of this paper is to show that, for the particular problem under consideration and for certain discretizations, Dijkstra-like methods can be used, resulting to fast algorithms. In particu-

0-7803-1968-0/94$4.00©1994 IEEE

lar, we will show under mild assumptions, that there is an algorithm whose complexity is proportional to the number of grid–points. Our starting point is the discretized HJ equation, which we take for granted and whose structure we then exploit; our development is completely independent from the rich analytical theory that deals with the justification of the HJ equation and its discretizations.

Most of the proofs and some of the details are omitted; they can be found in the full paper, available from the author.

2. Problem formulation and a finite difference discretization

The purpose of this section is purely to motivate the structure of the discretized HJ equation that will be studied in the rest of the paper; the reader is referred to the literature for rigorous and more precise statements.

Let G be a bounded connected open subset of $\Re^m$ and let ∂G be its boundary. We are also given two cost functions $r : G \mapsto (0,\infty)$ and $q : \partial G \mapsto (0,\infty)$. A *trajectory* starting at $x_0 \in G$ is a continuous function $x : [0,T] \mapsto \Re^m$ such that $x(t) \in G$ for all $t \in [0,T)$ and $x(T) \in \partial G$. A trajectory is called *admissible* if there exists a measurable function $u : [0,T] \mapsto \Re^m$ such that $x(t) = x(0) + \int_0^t u(s)\,ds$ and $\|u(t)\| \leq 1$ for all $t \in [0,T]$, where $\|\cdot\|$ stands for the Euclidean norm. The cost of an admissible trajectory is defined to be $\int_0^T r\big(x(t)\big)\,dt + q\big(x(T)\big)$. The optimal cost–to–go function $V^* : G \cup \partial G \mapsto \Re$ is defined as follows: if $x \in \partial G$, we let $V^*(x) = q(x)$; if $x \in G$, we let $V^*(x)$ be the infimum of the costs of all admissible trajectories that start at x.

A formal argument [FR] indicates that V^* should satisfy the Hamilton–Jacobi equation

$$\min_{\{v \in \Re^m \mid \|v\| \leq 1\}} \{r(x) + \langle v, \nabla V^*(x)\rangle\} = 0, \qquad x \in G. \tag{2.1}$$

Furthermore, for any $x \in \partial G$, V^* should satisfy

$$\limsup_{y \to x} V^*(y) \leq V^*(x), \tag{2.2}$$

where the limit is taken with y approaching x from the interior of G. If the problem data are smooth enough and if V^* is differentiable, it can be argued rigorously that V^* must satisfy Eqs. (2.1)–(2.2). Furthermore, V^* can be characterized as the maximal solution of Eqs. (2.1)–(2.2). Unfortunately, the assumptions needed for V^* to be differentiable are too strong for many practical problems. Equations (2.1)–(2.2) can be still justified, under much weaker assumptions, if V^* is interpreted as a "viscosity" solution of Eq. (2.1) [CDF, CL, FS].

We now describe a discretized version of the HJ equation. While this discretization is closely related or a special case of the discretizations described in [CDF, F, GR, KD], we provide a self–contained heuristic argument based on Bellman's principle of optimality. Once more, no rigorous results are derived or stated; our only purpose is to indicate the origin of the discretized HJ equation that will be studied later.

Let h be a small positive scalar representing the fineness of the discretization (the discretization step). Let S and B be two disjoint finite subsets of $\Re^m$, all their elements being of the form (ih, jh), where i and j are integers. The sets S and B are meant to represent a discretization of the sets G and ∂G, respectively. (For example, S could be the set of all grid points inside G and B could be the set of all gridpoints outside G that neighbor an element of S.)

Let $e_1, \ldots, e_m$ be the unit vectors in $\Re^m$. For any point $x \in S$, we define the set $N(x)$ of its *neighbors* by letting $N(x) = \{x + h\alpha_i e_i \mid i \in \{1,\ldots,m\},\ \alpha_i \in \{-1,1\}\}$. The assumption that follows states that B contains the "boundary" of S, in keeping with the intended meaning of these sets.

Assumption 2.1: For every $x \in S$, we have $N(x) \subset S \cup B$.

Let α be an element of $\mathcal{A} = \{-1,1\}^m$. To every $\alpha = (\alpha_1, \ldots, \alpha_m) \in \mathcal{A}$, we associate a *quadrant*, namely, the cone generated by the vectors $\alpha_1 e_1, \ldots, \alpha_m e_m$. Let Θ be the unit simplex in $\Re^m$; that is, $(\theta_1, \ldots, \theta_m) \in \Theta$ if and only if $\sum_{i=1}^m \theta_i = 1$ and $\theta_i \geq 0$ for all i.

We assume that we have two functions $f : B \mapsto (0,\infty)$ and $g : S \mapsto (0,\infty)$ that represent discretizations of the cost functions q and r in the original problem. The function g can be usually defined by $g(x) = r(x)$ for every $x \in S$. The choice of f can be more delicate because B can be disjoint from ∂G even if B is a good approximation of ∂G. In that case, some delicate analytical issues arise but are beyond the scope of this discussion.

We finally introduce a function $V : S \cup B \mapsto \Re$ which is meant to provide an approximation of the optimal cost–to–go function V^*. The discretized HJ equation is the following system of equations in the unknown V:

$$V(x) = \min_{\alpha \in \mathcal{A}} \min_{\theta \in \Theta} \left[hg(x)\tau(\theta) + \sum_{i=1}^m \theta_i V(x + h\alpha_i e_i) \right], \tag{2.3}$$

if $x \in S$, and

$$V(x) = f(x), \qquad x \in B, \tag{2.4}$$

where

$$\tau(\theta) = \|\theta\| = \sqrt{\sum_{i=1}^m \theta_i^2}, \qquad \theta \in \Theta. \tag{2.5}$$

We now explain the form of Eqs. (2.3)–(2.4). Suppose that the vehicle starts at some $x \in S$ and that it moves, at unit speed, along a direction d. This direction is determined by specifying the quadrant α to which d belongs and by then specifying the relative weights θ_i of the different vectors $\alpha_i e_i$ that generate this quadrant. Assume that the vehicle moves along the direction d until it hits the convex hull of the points

$x + h\alpha_i e_i$, $i = 1, \ldots, m$. At that time, the vehicle has reached point $x + h\sum_{i=1}^{m} \theta_i \alpha_i e_i$. Since the vehicle travels at unit speed, the amount of time it takes is equal to

$$\left\| h \sum_{i=1}^{m} \theta_i \alpha_i e_i \right\| = h\tau(\theta);$$

see Fig. 2.1. Since $g(x)$ represents travel costs per unit time (in the vicinity of x), the traveling cost is equal to $hg(x)\tau(\theta)$. To the traveling cost we must also add the cost-to-go from point $x + h\sum_{i=1}^{m} \theta_i \alpha_i e_i$ and, invoking the principle of optimality, we obtain

$$V^*(x) \approx \min_{\alpha \in \mathcal{A}} \min_{\theta \in \Theta} \left[hg(x)\tau(\theta) + V^*\left(x + h \sum_{i=1}^{m} \theta_i \alpha_i e_i \right) \right]. \tag{2.6}$$

We approximate V^* by a linear function on the convex hull of the points $x + h\alpha_i e_i$, to obtain

$$V^*\left(x + h \sum_{i=1}^{m} \theta_i \alpha_i e_i \right) \approx \sum_{i=1}^{m} \theta_i V^*(x + h\alpha_i e_i). \tag{2.7}$$

Using the approximation (2.7) in Eq. (2.6), we are led to Eq. (2.3).

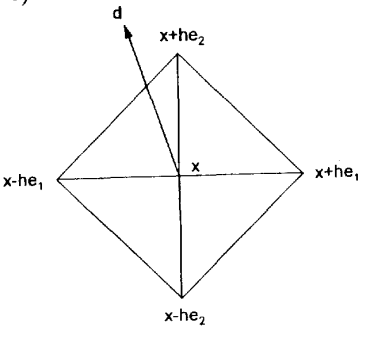

Figure 2.1. Illustration of the discretization of the HJ equation. Here, the vehicle moves along the direction d, in the quadrant defined by $-e_1$ and e_2; that is, $\alpha = (\alpha_1, \alpha_2) = (-1, 1)$.

The above discussion gives some plausibility to the claim that the solution V of Eqs. (2.3)-(2.4) can provide a good approximation of the function V^*. (Of course, some smoothness assumptions are required for this to be the case.) This motivates our main objective: providing an efficient algorithmic solution of Eqs. (2.3)-(2.4).

The discretization (2.3)-(2.4) is a special case of those considered in [GR, KD]. It is also related to those in [CD, F], except that the latter references involve a fixed time step, whereas our time step $\tau(\theta)$ is variable. It should also be pointed out that our choice of a particular discretization out of the multitude of choices allowed by [GR, KD] is not arbitrary; under most choices, the arguments in subsequent sections fail to go through.

References [GR] and [KD], which deal with more general types of problems, suggest the use of the successive approximation method, possibly an accelerated version. The computational complexity of each iteration is proportional to the number of grid-points. However, even for deterministic shortest path problems, the number of iterations is proportional to the diameter of the grid-graph, which is usually of the order of $1/h$. The number of iterations can be reduced using Gauss-Seidel relaxation (as in [GR], for example), but no theoretical guarantees are available. This is in contrast to Dijkstra-like algorithms that solve deterministic shortest path problems with essentially a single pass through the grid points.

3. A Dijkstra-like algorithm

Dijkstra's algorithm is a classical method for solving the shortest path problem on a finite graph. Its running time, for bounded degree graphs, is $O(n \log n)$, where n is the number of nodes, provided that it is implemented with suitable data structures [B2]. The key idea in Dijkstra's algorithm is to generate the nodes in order of increasing value of the cost-to-go function. This is done in n stages (one node is generated at each stage) and the $O(\log n)$ factor is due to the overhead of deciding which node is to be generated next. We will now show that a similar idea can be applied to the solution of Eqs. (2.3)-(2.4) and that the elements of $S \cup B$ can be generated in order of increasing values of $V(x)$.

Throughout this section, we reserve the notation $V(x)$ to indicate the unique solution of Eqs. (2.3)-(2.4). The key to the algorithm is provided by the following lemma that states that the cost-to-go $V(x)$ from any node x can be determined from knowledge of $V(y)$ for those nodes y with strictly smaller cost-to-go.

Lemma 3.1: Let $x \in S$, and let $\alpha \in \mathcal{A}$, $\theta \in \Theta$, be such that $V(x) = hg(x)\tau(\theta) + \sum_{i=1}^{m} \theta_i V(x + h\alpha_i e_i)$. Let $\mathcal{I} = \{i \mid \theta_i > 0\}$. Then, $V(x + h\alpha_i e_i) < V(x)$ for all $i \in \mathcal{I}$.

We now proceed to the description of the algorithm. Let x_1 be an element of B at which $f(x)$ is minimized. Using the Markov Decision Problem interpretation of Eqs. (2.3)-(2.4), it is evident that $V(x) \geq f(x_1) = V(x_1)$, for all $x \in S \cup B$. Thus, x_1 is a point with a smallest value of $V(x)$, and this starts the algorithm.

We now proceed to a recursive description of a general stage of the algorithm. Suppose that during the first k stages ($1 \leq k < n$) we have generated a set of points $P_k = \{x_1, \ldots, x_k\} \subset S \cup B$ with the property

$$V(x_1) \leq V(x_2) \leq \cdots \leq V(x_k) \leq V(x), \qquad \forall x \notin P_k.$$

Furthermore, we assume that the value of $V(x)$ has been computed for every $x \in P_k$. (The set P_k is like the set of *permanently labeled* nodes in Dijkstra's algorithm.)

We define $\overline{V}_k(x)$ by letting

$$\overline{V}_k(x) = \begin{cases} V(x), & \text{for } x \in P_k \cup B, \\ \infty, & \text{otherwise.} \end{cases}$$

We then compute an estimate $\hat{V}_k$ of the function V by essentially performing one iteration of the successive approximation algorithm, starting from $\overline{V}_k$. More precisely, for $x \in B$, let $\hat{V}_k(x) = V(x)$ and $x \in S$ let

$$\hat{V}_k(x) = \min_{\alpha \in \mathcal{A}} \min_{\theta \in \Theta} \left[hg(x)\tau(\theta) + \sum_{i=1}^{m} \theta_i \overline{V}_k(x + h\alpha_i e_i) \right], \tag{3.1}$$

In this equation, and throughout the rest of the paper, we use the interpretation $0 \cdot \infty = 0$. Since $\overline{V}_k(x) \geq V(x)$, a comparison of Eqs. (3.1) and (2.3) shows that

$$\hat{V}_k(x) \geq V(x), \qquad \forall x \in B \cup S.$$

The variable $\hat{V}_k(x)$, for $x \notin P_k$ is similar to the *temporary labels* in Dijkstra's algorithm.

We now choose a node with the smallest temporary label to be labeled permanently. Formally, we choose some x_{k+1} that minimizes $\hat{V}_k(x)$ over all $x \notin P_k$. The following lemma asserts that this choice of x_{k+1} is sound.

Lemma 3.2: (a) $V(x_{k+1}) = \hat{V}_k(x_{k+1})$.
(b) For every $x \notin P_k$, we have $\hat{V}(x_{k+1}) \leq V(x)$.

The description of the algorithm is now complete. The algorithm terminates after n stages and produces the values of $V(x)$ for all $x \in S \cup B$, in nondecreasing order. In order to determine the complexity of the algorithm, we will bound the complexity of a typical stage. Throughout this analysis, we view the dimension m of the problem as a constant, and we investigate the dependence of the complexity on n.

Let us first consider what it takes to compute $\hat{V}_k(x)$. There are $O(1)$ different elements α of $\mathcal{A}$ to consider and for each one of them, we have to solve, after some normalization, a convex optimization problem of the form

$$\min_{\theta \in \Theta} \left[\sqrt{\sum_{i=1}^{m} \theta_i^2} + \sum_{i=1}^{m} \theta_i V_i \right] \tag{3.2}$$

No matter what method is used to solve the problem (3.2), the computational effort is independent of the number n of grid points; it depends, of course, on the dimension m, but we are viewing this as a constant. Thus, we can estimate the complexity of computing $\hat{V}_k(x)$, for any fixed x, according to Eq. (3.1), to be $O(1)$. (In the full paper, it is also shown that $\hat{V}_k(x)$ can be computed using a finite number of arithmetic operations and square root calculations.)

We now notice that $\overline{V}_k(x) = \overline{V}_{k+1}(x)$ for every $x \neq x_{k+1}$. This means that if x is not a neighbor of x_{k+1}, then $\hat{V}_k(x) = \hat{V}_{k+1}(x)$. Thus, $\hat{V}_{k+1}(x)$ only needs to be computed for the $O(1)$ neighbors of x_{k+1}. We conclude that once $\hat{V}_k$ is computed, the evaluation of $\hat{V}_{k+1}$, at the next stage of the algorithm, only requires $O(1)$ computations.

At each stage, we must also determine the next point x_{k+1}, by minimizing $\hat{V}_k(x)$ over all $x \notin P_k$. Comparing $O(n)$ numbers takes $O(n)$ time, which leads to $O(n)$ time for each stage, and a total $O(n^2)$ running time. In a better implementation, the values $\hat{V}_k(x)$ can be maintained in a binary heap, in which case x_{k+1} can be determined in $O(\log n)$ time; see [B2] for the use of binary heaps in shortest path algorithms. We conclude that each stage of the algorithm can be implemented with $O(\log n)$ computations. We now summarize:

Theorem 3.1: The algorithm of this section solves the system of equations (2.3)–(2.4). Assuming that square roots can be evaluated in unit time, it can be implemented so that it runs in time $O(n \log n)$.

Some more comments are in order. We have been using a uniform grid. If we were to use a nonuniform grid instead, there would be some minor changes in the form of Eq. (2.3). The general structure would still be the same. However, Lemma 3.1 would cease to hold. Similarly, if the cost function $g(x)$ were to become direction dependent, e.g., of the form $g(x, \alpha, \theta)$, Lemma 3.1 would again fail to hold.

The algorithm of this section is inherently serial. This is because the elements of S are generated one at a time, in order of increasing values of $V(x)$. To obtain a parallelizable algorithm, we should be able to generate the values of $V(x)$ for several points x simultaneously. This is accomplished by the method developed in the next section.

4. An algorithm with optimal complexity

The algorithm of Section 3 achieved $O(n \log n)$ running time by mimicking Dijkstra's shortest path algorithm.

A property that would lead to a fast solution of Eqs. (2.3)–(2.4) is the following:

Property P: There exists a constant $\delta > 0$ such that if $V(x) = hg(x)\tau(\theta) + \sum_{i=1}^{m} \theta_i V(x + h\alpha_i e_i)$ and if for some j we have $\theta_j > 0$, then $V(x) \geq V(x + h\alpha_j e_j) + \delta$.

Lemma 3.1 established that property P holds with $\delta = 0$. Unfortunately, property P is not true for Eqs. (2.3)–(2.4) when we let δ be positive. To see this, let us focus on the first quadrant, let ϵ be a small positive number and consider the case where $m = 2$, $h = g(x) = 1$, $V(x + e_1) = 1 - \epsilon$, $V(x + e_2) = 0$. For any positive ϵ, the optimal value of θ_1 can be computed and is positive. On the other hand, the value of $V(x)$ is bounded above by 1 and the difference $V(x) - V(x + e_1)$ is no larger than ϵ. Since this is true for every $\epsilon > 0$, property P does not hold.

In this section, we show that property P becomes true if a somewhat different discretization is used. Unfortunately, the discretization that we introduce is more cumbersome and is unlikely to be useful when the dimension is higher than 3. We describe here our method when the dimension m is 2; the full paper also deals with the case where $m = 3$.

Let H be the boundary of a square centered at the origin and whose edge length is equal to $2h$. We define the vectors $w_1, \ldots, w_9$ as shown in Fig. 4.1. We use $x + H$ to denote the translation of H so that it is centered at x.

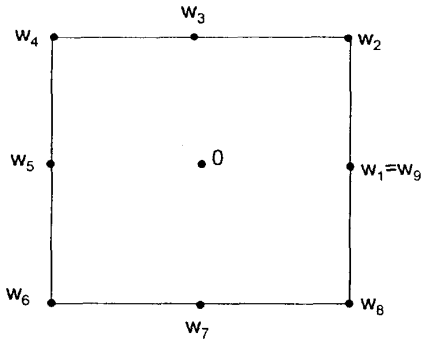

Figure 4.1. A square of size $2h \times 2h$ centered at the origin and the definition of the vectors $w_1, \ldots, w_9$.

As in Section 2, let S and B be two disjoint finite subsets of $\Re^m$, all of their elements being of the form (ih, jh), where i and j are integers. We assume that we are given functions $f : S \mapsto (0, \infty)$ and $g : B \mapsto (0, \infty)$. For any point $x \in S$, let $N(x)$, the set of its be neighbors, be $N(x) = \{x + w_i \mid i = 1, \ldots, 8\}$. As in Assumption 2.1, we assume that for every $x \in S$, we have $N(x) \subset S \cup B$.

We now motivate the discretization of the HJ equation that will be used in this section. Suppose that the vehicle starts at some $x \in S$ and moves along a direction d, for some time τ, until it hits the set $x + H$. The direction d is in the cone generated by w_α and $w_{\alpha+1}$ for some suitable choice of α. The point at which the vehicle meets H is of the form $(1-\theta)w_\alpha + \theta w_{\alpha+1}$, for some $\theta \in [0, 1]$. We will thus parametrize the choice of direction d by a parameter $\alpha \in \{1, \ldots, 8\}$ that specifies a particular cone and then by a parameter $\theta \in [0, 1]$ that picks a particular element of that cone. Let $h\tau_\alpha(\theta)$ be the travel time along the direction determined by α and θ, until the set $x + H$ is reached. It is easily seen that

$$\tau_\alpha(\theta) = \|(1-\theta)w_\alpha + \theta w_{\alpha+1}\|$$

$$= \begin{cases} \sqrt{1+(1-\theta)^2}, & \text{if } \alpha \text{ is even,} \\ \sqrt{1+\theta^2}, & \text{if } \alpha \text{ is odd.} \end{cases}$$

Using the principle of optimality, as in Section 2, and by approximating V by a linear function on the segment joining w_α and $w_{\alpha+1}$, we obtain the following system of equations:

$$V(x) = \min_{\alpha=1,\ldots,8} \min_{\theta\in[0,1]} \Big[hg(x)\tau_\alpha(\theta)$$

$$+(1-\theta)V(x+w_\alpha)+\theta V(x+w_{\alpha+1})\Big], \qquad x \in S, \quad (4.1)$$

$$V(x) = f(x), \qquad x \in B. \quad (4.2)$$

Equations (4.1)–(4.2) are again a special case of the finite element discretizations studied in [GR, KD]. Once more, they admit a Markov Decision Process interpretation, have a unique solution, and we reserve the notation $V(x)$ to denote such a solution.

Recall that the cost per stage g in the discretized problem has been assumed to be positive. In the following, we assume a lower bound of unity for g and proceed to establish property P.

Assumption 4.1: For every $x \in S$, we have $g(x) \geq 1$.

Lemma 4.1: Fix some $x \in S$ and let α, θ attain the minimum in Eq. (4.1), that is,
$V(x) = hg(x)\tau_\alpha(\theta)+(1-\theta)V(x+w_\alpha)+\theta V(x+w_{\alpha+1})$.
If $\theta < 1$, then $V(x) \geq V(x + w_\alpha) + (h/\sqrt{2})$. If $\theta > 0$, then $V(x) \geq V(x + w_{\alpha+1}) + (h/\sqrt{2})$.

Let $\delta = h/\sqrt{2}$. Let $Q_k = \{x \mid (k-1)\delta \leq V(x) < k\delta\}$ and $R_k = \cup_{i=0}^{k} Q_i = \{x \mid V(x) < k\delta\}$. Suppose that at some stage of the algorithm, we have computed $V(x)$ for all $x \in R_k$. We define $\overline{V}_k(x)$ to be equal to $V(x)$ if $x \in R_k$ and infinity otherwise. Let, for $x \in S$,

$$\hat{V}_k(x) = \min_\alpha \min_{\theta\in\Theta} \Big[hg(x)\tau_\alpha(\theta)$$

$$+(1-\theta)\overline{V}(x+w_\alpha)+\theta\overline{V}(x+w_{\alpha+1})\Big], \qquad (4.3)$$

where we are again following the convention $0 \cdot \infty = 0$. If $V(x) \geq (k+1)\delta$, then $\hat{V}_k(x) \geq V(x) \geq (k+1)\delta$. If on the other hand $V(x) < (k+1)\delta$, Lemma 4.1 shows that if the minimizing θ in Eq. (4.3) is positive, we must have $V(x+w_{\alpha+1}) < k\delta$ and therefore $x+w_{\alpha+1} \in R_k$ and $V(x + w_{\alpha+1}) = \overline{V}_k(x + w_{\alpha+1})$. By a similar argument, if the minimizing θ in Eq. (4.3) is less than 1, then $V(x + w_\alpha) = \overline{V}_k(x + w_\alpha)$. This implies that $\hat{V}_k(x) = V(x)$. Thus, we have computed $V(x)$ for every $x \in R_{k+1}$, and we are ready to start the next stage of the algorithm.

We implement the algorithm by using buckets that hold all of the grid points x for which $(k-1)\delta \leq \hat{V}_k(x) < k\delta$. Since each x has a bounded number of "neighboring points" $x+w_\alpha$, a point x may move from one bucket to another and the value of $\hat{V}_k(x)$ may need to be recomputed only $O(1)$ times. Each time that $\hat{V}_k(x)$ is recomputed, we need to solve the optimization problem in Eq. (4.3). This can be done with a finite number of operations, provided that square root computations are counted as single operations. Thus, the complexity estimate becomes $O(n)$ plus the number of buckets employed. The number of buckets can be bounded in turn by $O(L/\delta) = O(L/h)$, where L is an upper bound on $\max_{x\in S} V(x)$.

Theorem 4.1: Let Assumption 4.1 hold and assume that square roots can be evaluated in unit time. Then, a solution of Eqs. (4.1)–(4.2) can be computed in time $O(n + L/h)$, where L is an upper bound for $\max_{x\in S} V(x)$. The same complexity estimated is also obtained for three-dimensional problems.

We now interpret the complexity estimate of Theorem 4.1 in terms of the original continuous trajectory optimization problem. We assume that the underlying cost function r (cf. Section 1) is bounded below by some positive constant. For a problem involving trajectories in $\Re^m$, the number of grid-points is

$n = O(h^{-m})$, where h is the grid–spacing. On the other hand $\hat{V}(x)$ should converge to $V^*(x)$, the cost–to–go for the original continuous problem, which is independent of h. In particular, the factor L in Theorem 4.1 can be taken independent of h. For $m > 1$, the term $O(n)$ is the dominant one in the complexity estimate $O(n + L/h)$. We conclude that, as long as the problem data in a trajectory optimization problem are regular enough for our discretizations to be justified, we have algorithms whose complexity is proportional to the number of grid–points involved, which is the best possible.

5. Parallel implementation

The algorithm of Section 4 can be parallelized as follows. Let us concentrate on the computations required during a typical stage of the algorithm. Suppose, for example, that $\hat{V}_k(x)$ is available for all points x, so that Q_{k+1} can be determined. Let N_{k+1} be the set of points that have a neighbor belonging to Q_{k+1}. For every $x \notin N_{k+1}$, we have $\hat{V}_{k+1}(x) = \hat{V}_k(x)$ and no computation is required to obtain $\hat{V}_{k+1}(x)$. Thus, a high–level description of a typical stage of the algorithm of Section 4 is as follows:

1. Use the values of $\hat{V}_k(x)$ to determine the set Q_{k+1}.
2. Determine the set N_{k+1}.
3. For every $x \in N_{k+1}$, compute, in parallel, the value of $\hat{V}_{k+1}(x)$.

By splitting the computations at each stage between p processors, and taking into account some overhead required in order to do some load balancing, the algorithm can be implemented in $O(n^{1/2} \log n)$ time using a shared memory parallel computer with $O(n^{1/2}/\log n)$ processors. See the full paper for more details.

6. Numerical results

We report here on some preliminary numerical experiments designed and carried out by L. C. Polymenakos. The problems considered dealt with the unit square discretized using a 150×150 grid of points. The running cost was chosen to be a concave quadratic function. A few line obstacles were also added in the interior of the square. Depending on the number of obstacles, a reasonable implementation of the Gauss–Seidel variant of the successive approximation algorithm was found to be between 10 and 50 times slower than our Dijkstra–like method.

Acknowledgments

The author wishes to thank Mr. Lazaros Polymenakos for carrying out the computational experiments mentioned in Section 6. This research was supported by the the ARO under contract DAAL03–92–G–0115.

References

[AF] Athans, M., and Falb, P. L., *Optimal Control*, McGraw Hill, New York, 1966.

[B1] Bertsekas, D. P., *Dynamic Programming: Deterministic and Stochastic Models*, Prentice Hall, Englewood Cliffs, New Jersey, 1987.

[B2] Bertsekas, D.P., *Linear Network Optimization*, Prentice Hall, Englewwod Cliffs, NJ, 1991.

[BT] Bertsekas, D. P., and Tsitsiklis, J. N., "An Analysis of Stochastic Shortest Path Problems", *Mathematics of Operations Research*, Vol. 16, No. 3, August 1991, pp. 580–595.

[CD] Capuzzo Dolcetta, I., "On a discrete approximation of the Hamilton–Jacobi equation of dynamic programming", *Applied Mathematics and Optimization*, Vol. 10, 1983, pp. 367–377.

[CDF] Capuzzo Dolcetta, I., Falcone, M., "Viscosity solutions and discrete dynamic programming", *Annales Institut H. Poincare*, Analyse non lineaire, 6 (supplement), 1989, pp. 161–183.

[CDI] Capuzzo Dolcetta, I., Ishii, H., "Approximate solutions of the Bellman equation of deterministic control theory", *Applied Mathematics and Optimization*, Vol. 11, 1984, pp. 161–181.

[CL] Crandall, M. G., Lions, P.–L., "Viscosity solutions of Hamilton–Jacobi equations", *Transactions of the American Mathematical Society*, Vol. 277, No. 1, 1983, pp. 1–42.

[F] Falcone, M., "A numerical approach to the infinite horizon problem of deterministic control theory", *Applied Mathematics and Optimization*, 15, 1987, pp. 1–13; corrigenda, 23, 1991, pp. 213–214.

[FR] Fleming, W., and Rishel, R., *Deterministic and Stochastic Optimal Control*, Springer–Verlag, New York, 1975.

[FS] Fleming, W. H., Soner, H. M., *Controlled Markov Processes and Viscosity Solutions*, Springer–Verlag, New York, 1993.

[GR] Gonzalez, R., and Rofman, E., "On deterministic control problems: an approximation procedure for the optimal cost, I, the stationary problem", *SIAM J. on Control and Optimization*, 23, 2, 1985, pp. 242-266.

[KD] Kushner, H. J., Dupuis, P. G., *Numerical Methods for Stochastic Control Problems in Continuous Time*, Springer–Verlag, New York, 1992.

[M] Mitchell, J. S. B., "Planning shortest paths", PhD thesis, Dept. of Operations Research, Stanford University, Stanford, California, 1986.

[S] Souganidis, P. E., "Approximation schemes for viscosity solutions of Hamilton–Jacobi equations", *J. of Differential Equations*, Vol. 59, 1985, pp. 1–43.