[6] R. K. Yedavalli, "Improved measures of stability robustness for linear state space models," *IEEE Trans. Automat. Contr.*, vol. AC-30, pp. 577–579, June 1985.
[7] R. K. Yedavalli and Z. Liang, "Reduced conservatism in stability robustness bounds by state transformation," *IEEE Trans. Automat. Contr.*, vol. AC-31, pp. 863–866, Sept. 1986.
[8] K. M. Zhou and P. P. Khargonekar, "Stability robustness bounds for state-space modes with structured uncertainty," *IEEE Trans. Automat. Contr.*, vol. AC-32, pp. 621–623, July 1987.
[9] S. R. Kolla, R. K. Yedavalli, and J. B. Farison, "Robust stability bounds on time-varying perturbations for state-space models of discrete-time systems," *Int. J. Contr.*, vol. 50, no. 1, pp. 151–159, 1989.
[10] M. E. Sezer and D. D. Šiljac, "A note on robust stability bounds," *IEEE Trans. Automat. Contr.*, vol. 34, pp. 1212–1214, Nov. 1989.
[11] Z. Bien and J. D. Kim, "A robust stability bound of linear systems with structured uncertainty," *IEEE Trans. Automat. Contr.*, vol. 37, pp. 1549–1551, Oct. 1992.
[12] P. Dorato and R. K. Yedavalli, Eds., *Recent Advances in Robust Control*. New York: IEEE Press, 1990.
[13] V. L. Kharitonov, "Asymptotic stability of an equilibrium position of a family of systems of linear differential equations," *Differential'nye Uravneniya*, vol. 14, pp. 2086–2088, 1978.
[14] ——, "The Routh-Hurwitz problem for families of polynomials and quasipolynomials," *Izvetiy Akademii Nauk Kazakhskoi SSR, Seria fiziko-matematicheskaia*, no. 26, pp. 69–79, 1979.
[15] A. C. Bartlett, C. V. Hollot, and H. Lin, "Root locations of an entire polytope of polynomials: It suffices to check the edges," *Math. Contr. Signals. Syst.*, vol. 1, pp. 61–71, 1988.
[16] R. J. Minnichelli, J. J. Anagnost, and C. A. Desoer, "An elementary proof of Kharitonov's stability theorem with extensions," *IEEE Trans. Automat. Contr.*, vol. 34, pp. 995–998, 1989.
[17] B. R. Barmish, *New Tools for Robustness of Linear Systems*. New York: Macmillan, 1994.
[18] S. J. Xu, M. Darouach, and J. Schaefers, "The expansion of $\det(A+B)$ and the robustness analysis of uncertain state space systems," *IEEE Trans. Automat. Contr.*, vol. 38, pp. 1671–1675, 1993.
[19] ——, "Expansion of $\det(A+B+C)$ and robustness analysis of discrete-time state space systems," *IEEE Trans. Automat. Contr.*, vol. 40, pp. 936–942, 1995.
[20] A. Rachid, "Robustness of pole assignment in a specified region for perturbed systems," *Int. J. Syst. Sci.*, pp. 579–585, 1990.
[21] K. Furuta and S. B. Kim, "Pole assignment in a specified disk," *IEEE Trans. Automat. Contr.*, vol. AC-32, pp. 423–427, 1987.
[22] S. S. Wang and W. G. Lim, "On the analysis of eigenvalue assignment robustness," *IEEE Trans. Automat. Contr.*, vol. 37, pp. 1561–1564, 1992.
[23] Y. G. Fang, "The analysis of eigenvalue assignment robustness," *IEEE Trans. Automat. Contr.*, vol. 37, pp. 1561–1564, 1995.
[24] Y. T. Juang, Z. C. Hong, and Y. T. Wang, "Robustness of pole-assignment in a specified region," *IEEE Trans. Automat. Contr.*, vol. 34, p. 158, 1989.
[25] ——, "Lyapunov approach to robust pole-assignment analysis," *Int. J. Contr.*, vol. 49, no. 3, pp. 921–927, 1989.
[26] Y. T. Juang, "Eigenvalue assignment robustness for systems with structured perturbation," *IEEE Trans. Automat. Contr.*, vol. 38, pp. 1697–1700, 1993.
[27] R. K. Yedavalli, "Robust root clustering for linear uncertain systems using generalized Lyapunov theory," *Automatica*, vol. 29, no. 6, pp. 237–240, 1993.
[28] W. Bakker, J. S. Luo, and A. Johnson, "Perturbation bounds for root-clustering of linear systems in a specified second order subregion," *IEEE Trans. Automat. Contr.*, vol. 40, pp. 473–478, 1995.
[29] P. C. Parks and C. Kahn, *Stability Theory*. Englewood Cliffs, NJ: Prentice Hall, 1993.
[30] B. R. Barmish and H. I. Kang, "A survey of extreme point results for robustness of control systems," *Automatica*, vol. 29, no. 1, pp. 13–35, 1993.

Implementation of Efficient Algorithms for Globally Optimal Trajectories

L. C. Polymenakos, D. P. Bertsekas, and J. N. Tsitsiklis

***Abstract*—We consider a continuous-space shortest path problem in a two-dimensional plane. This is the problem of finding a trajectory that starts at a given point, ends at the boundary of a compact set of $\Re^2$, and minimizes a cost function of the form $\int_0^T r(x(t))\,dt + q(x(T))$. For a discretized version of this problem, a Dijkstra-like method that requires one iteration per discretization point has been developed by Tsitsiklis [10]. Here we develop some new label correcting-like methods based on the Small Label First methods of Bertsekas [2] and Bertsekas *et al.* [6]. We prove the finite termination of these methods, and we present computational results showing that they are competitive and often superior to the Dijkstra-like method and are also much faster than the traditional Jacobi and Gauss–Seidel methods.**

***Index Terms*—Label correcting, label setting, optimal control, shortest paths.**

I. Introduction: Problem Formulation

We consider a continuous-space shortest path problem and its discretization. This problem has been addressed by Tsitsiklis [10], and our presentation follows that reference closely. We are given a bounded open subset G of $\Re^2$ and a point $x(0) \in G$. A *trajectory* starting at $x(0)$ is a continuous function x: $[0,T] \mapsto \Re^2$, where T is some positive scalar such that $x(t) \in G$ for all $t \in [0,T)$ and $x(T) \in \partial G$, where ∂G is the boundary of G. A trajectory is called *admissible* if there exists a positive scalar T and a measurable function u: $[0,T] \mapsto \Re^2$ such that

$$x(t) = x(0) + \int_0^t u(s)\,ds$$

and

$$\|u(t)\| \le 1, \qquad \forall\, t \in [0,T]$$

where $\|\cdot\|$ stands for the Euclidean norm. The cost of an admissible trajectory is defined as

$$\int_0^T r(x(t))\,dt + q(x(T))$$

where r: $G \mapsto (0,\infty)$ and q: $\partial G \mapsto (0,\infty)$ are given cost functions. We want to find an admissible trajectory of least cost. Note that we have considered a two-dimensional space for simplicity. The algorithms and the analysis of this paper admit straightforward generalizations to higher-dimensional spaces.

We consider a method for discretization of this problem described and analyzed by Kushner and Dupuis [7], who give several earlier references. The advantage of this method is that it does not require the explicit discretization of the control space. Our methodology focuses exclusively on the discretized version of the problem and does not

Manuscript received December 4, 1995; revised September 24, 1996. This work was supported by the NSF under Grant 9300494-DMI.

L. C. Polymenakos is with IBM Research, Yorktown Heights, NY USA.
D. P. Bertsekas and J. N. Tsitsiklis are with the Laboratory for Information and Decision Systems, M.I.T., Cambridge, MA 02139 USA.

Publisher Item Identifier S 0018-9286(98)01175-1.

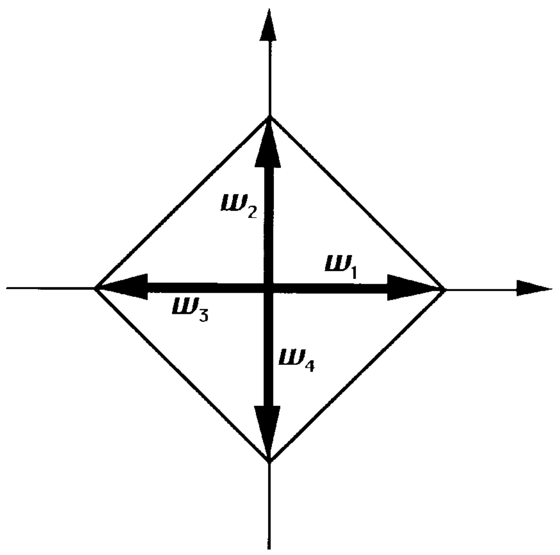

Fig. 1. A square centered at the origin and the definition of the vectors $w_1, \cdots, w_4$ of length h.

address or depend on the relation between the discretized and the continuous versions. We form a discretization grid using a square centered at the origin whose corners are vectors w_1, w_2, w_3, w_4 of length h, as shown in Fig. 1. This grid consists of two disjoint finite sets S and B such that for each $x \in S$, the set of *neighbors* of x, defined by

$$N(x) = \{x + w_i | i = 1, 2, 3, 4\}$$

is a subset of $S \cup B$. The set S should be viewed as a discretization of the interior set G, and the set B should be viewed as a discretization of the boundary set ∂G. We also have two functions, $f\colon B \mapsto (0, \infty)$ and $g\colon S \mapsto (0, \infty)$, that represent discretizations of the cost functions q and r of the original problem, respectively. The function g usually can be defined by $g(x) = r(x)$ for every $x \in S$.

We now consider a finite-state optimal control problem, the states of which are the points $x \in S \cup B$, also referred to as *nodes*. This problem fits within the framework of the stochastic shortest path problems discussed in [4] and [5], which cite several earlier references. The problem is defined as follows: at a state $x \in S$, we must choose a quadrant spanned by the vectors w_α and $w_{\alpha+1}$, where $\alpha \in \{1, 2, 3, 4\}$, and then choose a parameter $\theta \in [0, 1]$ that specifies an element $\theta w_\alpha + (1 - \theta) w_{\alpha+1}$ of the line segment connecting w_α and $w_{\alpha+1}$ (the indexing of w is modulo four, so that $w_5 = w_1$). The next state is $x + w_\alpha$ with probability θ and $x + w_{\alpha+1}$ with probability $1 - \theta$. The cost of the choice (α, θ) is $hg(x)\tau(\theta)$, where

$$\tau(\theta) = \sqrt{\theta^2 + (1 - \theta)^2}$$

so that

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

is the distance traveled from x to the point $x + \theta w_\alpha + (1 - \theta) w_{\alpha+1}$. Also, if the state $x \in B$ is reached, then a terminal cost $f(x)$ is incurred and the process terminates. The optimal cost-to-go function $V^*(x)$ of the original continuous-time problem, which is the infimum of the costs of all admissible trajectories that start at x, is approximated by the optimal cost-to-go function $\{V(x) | x \in S \cup B\}$ of the discretized problem.

It can be shown under our assumptions that the optimal cost-to-go $V(x)$ of the discretized problem is finite for all x and that the function V is the unique solution of the following Bellman equations:

$$V(x) = \min_{\alpha=1,2,3,4} \min_{\theta \in [0,1]} \left[\underbrace{hg(x)\tau(\theta)}_{\text{traveling cost}} + \underbrace{\theta V(x + w_\alpha) + (1 - \theta) V(x + w_{\alpha+1})}_{\text{expectedcost-to-go}} \right], \quad x \in S \tag{1}$$

$$V(x) = f(x), \qquad x \in B. \tag{2}$$

To see this, note that from every nonboundary state $x \in S$ we can go to the four neighbors w_α, where $\alpha \in \{1, 2, 3, 4\}$. Therefore, since S is assumed to be a finite set, it is possible to go from each nonboundary state $x \in S$ to at least one boundary state, and this implies that the optimal cost-to-go of the discretized problem (which cannot be negative since $g(x) > 0$ for all x) must be finite from every initial state. Furthermore, since $g(x) > 0$ for all $x \in S$, all policies that do not reach the boundary with positive probability result in infinite cost. This implies that the assumptions of the theory of stochastic shortest path problems (see [3]–[5]) are satisfied. This theory implies that the Bellman equations have the optimal cost-to-go function V as their unique solution. The theory also guarantees that the value iteration method will converge to the solution of the above equations, but does not guarantee finite termination. However, here we have a special structure that is implied by the positivity of the cost $g(x)$ and the shortest path character of the problem. A key property in this regard is given in the following proposition, first proved in [10].

Proposition 1: Let V be the solution of the Bellman equations (1) and (2). Let $x \in S$, and let $\theta \in [0, 1]$ and $\alpha \in \{1, 2, 3, 4\}$ be such that $V(x) = hg(x)\tau(\theta) + \theta V(x + w_\alpha) + (1 - \theta) V(x + w_{\alpha+1})$. If $\theta > 0$, then $V(x) > V(x + w_\alpha)$. If $1 - \theta > 0$, then $V(x) > V(x + w_{\alpha+1})$.

Using the above proposition, it was shown in [10] that a Gauss–Seidel algorithm that cycles through the nodes terminates finitely. Furthermore, a Dijkstra-like algorithm that requires only one iteration per node was proposed in [10] and was shown to be much faster in theory than the Jacobi and Gauss–Seidel methods that are typically used to solve stochastic shortest path problems. It has been confirmed by our experiments that the Dijkstra-like algorithm is also much faster in practice than the Jacobi and Gauss–Seidel methods.

The key property implied by Proposition 1 is that there exists an optimal policy under which, from any state, we can only go to a state of lower cost. Such policies are called *consistently improving* and are discussed in more detail in [3, p. 90]. When a consistently improving policy exists, a Dijkstra-like algorithm as well as other finitely terminating value iteration algorithms of the type considered in this paper can be used (see [3, p. 127]).

The objective of this paper is twofold. First, we introduce a broad class of value iteration methods, which resemble the label-correcting methods used to solve deterministic shortest path problems. These methods can be viewed as Gauss–Seidel methods with the node order used for iteration being arbitrary (not necessarily cyclical). We show that these methods terminate (see Proposition 2 in the next section), although they may require more iterations than the Dijkstra-like algorithm of [10]. Second, we develop several new label correcting-like methods, which use special rules to define the node order used for iteration and which try to approximate the operation of the Dijkstra-like algorithm with smaller overhead per iteration. These methods are patterned after the Smallest Label First (SLF) method of [2] and the Smallest Label First–Last Label Last (SLF–LLL) method of [6], which have been shown experimentally to be very effective for deterministic shortest path problems. We provide computational results showing that these new methods are competitive and often superior to the Dijkstra-like algorithm and are also much faster than the traditional Jacobi and Gauss–Seidel methods. The new methods also lend themselves better for parallelization than the Dijkstra-like algorithm.

II. Generic Label-Correcting Algorithm

In this section we describe a general algorithm for solving the stochastic shortest path problem corresponding to the discretization discussed in Section I. The algorithm, referred to as *generic*, is patterned after a generic label-correcting method for deterministic shortest path problems; see for example [8] and [1]. It maintains a list of nodes L called *the candidate list* and a *label* $V(x)$ for each node $x \in S \cup B$. Each label is either a real number or ∞. Initially

$$\begin{aligned} L &= B \\ V(x) &= f(x), \qquad \forall x \in B \\ V(x) &= \infty, \qquad \forall x \in S. \end{aligned}$$

For convenience, we also keep track of the direction $(\alpha(x), \theta(x))$ along which the current label of x was calculated. The algorithm proceeds in iterations and terminates when L is empty. The typical iteration of the algorithm (assuming that L is not empty) is as follows.

Typical Iteration of the Generic Label-Correcting Algorithm

Remove a node x from the candidate list L. For each neighbor y of x that also belongs to S, if $V(y) > V(x)$, calculate

$$\begin{aligned} \tilde{V}(y) = \min_{\{\alpha=1,2,3,4 \mid x \in \{y+w_\alpha, y+w_{\alpha+1}\}\}} \min_{\theta\in[0,1]} [hg(y)\tau(\theta) \\ + \theta V(y+w_\alpha) + (1-\theta)V(y+w_{\alpha+1})]. \end{aligned}$$

Let $(\tilde{a}, \tilde{\theta})$ be the direction for which the minimum value $\tilde{V}(y)$ is obtained. If $V(y) > \tilde{V}(y)$, then set $V(y) = \tilde{V}(y), a(y) = \tilde{a}, \theta(y) = \tilde{\theta}$ and add y to L if it is not already in L.

It can be seen that in the course of the algorithm the labels are monotonically nonincreasing and that $V(x) < \infty$ if and only if x has entered the candidate list at least once. The following lemma gives the main properties of the algorithm.

Proposition 2:

1) At the end of each iteration the following conditions hold.

a) $V(x) = f(x)$ for all $x \in B$. Furthermore, nodes in B do not re-enter the candidate list once removed.

b) For all $x \in S$, if $V(x) < \infty$, then

$$\begin{aligned} V(x) \geq hg(x)\tau(\theta) + \theta(x)V(x+w_{\alpha(x)}) \\ + (1-\theta(x))V(x+w_{\alpha(x)+1}). \qquad (3) \end{aligned}$$

c) If for a node $x \in S$ there is a quadrant α such that $x + w_\alpha \notin L$ and $x + w_{\alpha+1} \notin L$, then we have

$$\begin{aligned} V(x) \leq \min_{\theta\in[0,1]} [hg(x)\tau(\theta) + \theta V(x+w_\alpha) \\ + (1-\theta)V(x+w_{\alpha+1})]. \qquad (4) \end{aligned}$$

2) The algorithm terminates. The set of labels $\{V(x) \mid x \in S \cup B\}$ obtained upon termination solves the Bellman equations (1) and (2).

Proof:

1) Condition a) holds since by the rules of the algorithm the labels of the border nodes cannot change, and only nodes in S can re-enter the candidate list L. We prove b) as follows: just after the label of x is reduced, we have

$$\begin{aligned} V(x) = hg(x)\tau(\theta) + \theta(x)V(x+w_{\alpha(x)}) \\ + (1-\theta(x))V(x+w_{\alpha(x)+1}). \end{aligned}$$

Since the labels of the neighboring nodes of x are nonincreasing, we see that (3) holds in subsequent iterations until the value of x is recalculated.

To prove c), note that initially the nodes not in L have infinite labels. Therefore, c) holds trivially at the beginning of the algorithm. Let us now fix a node x and its two neighbors $x + w_\alpha, x + w_{\alpha+1}$ of some quadrant α. If neither of the nodes $x + w_\alpha, x + w_{\alpha+1}$ enters the candidate list L throughout the algorithm, then c) holds since the label of x is nonincreasing. Otherwise, at least one of the nodes $x + w_\alpha, x + w_{\alpha+1}$ enters L at some time. Consider now an iteration k of the algorithm where node $x + w_\alpha$ or node $x + w_{\alpha+1}$, exits L, and as a result both nodes are not in L. At this iteration, the label of x is recalculated and (4) is satisfied. Furthermore, (4) is satisfied in all subsequent iterations of the algorithm until either termination is reached or one of the nodes $x + w_\alpha, x + w_{\alpha+1}$ re-enters L at some iteration $k' > k$. This is because for all iterations performed after k until either termination or iteration k' is reached, the labels of $x+w_\alpha, x+w_{\alpha+1}$ remain unchanged while the label of x is nonincreasing. We conclude that (4) holds throughout the algorithm.

2) We assume that the algorithm does not terminate in order to reach a contradiction. Let I be the set of nodes that enters L a positive but finite number of times, and let $\overline{I}$ be the set of nodes that enters L an infinite number of times. Also let J be the set of nodes that never enters L and whose labels are infinite throughout the algorithm. The sets I, $\overline{I}$, and J, and the labels of the nodes in I remain unchanged after some iteration denoted $\overline{m}$. Furthermore, the sets I and $\overline{I}$ are nonempty since from Proposition 2.1-a), $B \subset I$, implying that I is nonempty and the algorithm does not terminate, implying that $\overline{I}$ is nonempty.

Each time a node enters L, its label is smaller than the preceding time it entered L. Since the label of a node is bounded below by zero, we conclude that the labels of the nodes in $I \cup \overline{I}$ converge. For a node $x \in I \cup \overline{I}$, let $V^\infty(x)$ denote the limiting value of the label of x, and let $\alpha^\infty(x)$ and $\theta^\infty(x)$ be such that

$$\begin{aligned} V^\infty(x) = hg(x)\tau(\theta^\infty(x)) + \theta^\infty(x)V^\infty(x+w_{\alpha^\infty(x)}) \\ + (1-\theta^\infty(x))V^\infty(x+w_{\alpha^\infty(x)+1}). \qquad (5) \end{aligned}$$

Note that $\alpha^\infty(x)$ and $\theta^\infty(x)$ exist by the compactness of the sets where α and θ take values. We also define the set $\mathcal{P}(x)$ of *desired nodes of* x as follows:

$$\mathcal{P}(x) = \begin{cases} \{x + w_{\alpha^\infty(x)}\}, & \text{if } \theta^\infty(x) = 1 \\ \{x + w_{\alpha^\infty(x)+1}\}, & \text{if } \theta^\infty(x) = 0 \\ \{x + w_{\alpha^\infty(x)}, x + w_{\alpha^\infty(x)+1}\}, & \text{otherwise.} \end{cases}$$

We observe that in view of (5), we have

$$\mathcal{P}(x) \cap \overline{I} \neq \emptyset, \qquad \forall x \in \overline{I}$$

since, after iteration $\overline{m}$, for each $x \in \overline{I}$, the labels of the nodes in $\mathcal{P}(x) \cap I$ remain constant, while the label of x decreases infinitely many times; if all nodes of $\mathcal{P}(x)$ were in I, (5) would be violated. Thus, each node $x \in \overline{I}$ has at least one desired node in $\overline{I}$. Consequently, there exists a cycle of nodes of $\overline{I}$, say $(x_1, x_2, \cdots, x_k, x_{k+1})$, such that $x_{k+1} = x_1$ and x_{i+1} is a desired node of x_i, for $i = 1, \cdots, k$. From Proposition 1 we have that $V^\infty(x_i) > V^\infty(x_{i+1})$ for all $i = 1, \cdots, k$, which is a contradiction. Thus the algorithm terminates.

Next we show that all nodes $x \in S$ will exit L at least once so that they must be finite upon termination. In particular, let us consider a node $x_1 \in S$ and a sequence of nodes $(x_1, \cdots, x_k, x_{k+1})$ such that $x_{k+1} \in B$ and $x_{i+1} \in N(x_i)$, for all $i = 1, \cdots, k$. As discussed in Section I, such a node sequence exists for every $x_1 \in S$. Since the algorithm terminates, the boundary node x_{k+1} must exit L, since it belongs to L initially. When node x_{k+1} exits L, the label of node x_k is calculated and, if x_k has never before entered L, its label becomes finite and it enters L. Repeating this argument using x_k in place of x_{k+1}, and proceeding similarly, we can prove that each of the nodes $x_{k-1}, \cdots, x_1$ will exit L at least once.

We finally note that upon termination, the list L is empty so that both (3) and (4) hold for all $x \in S$, implying that the labels of the nodes satisfy the Bellman equations (1) and (2).$\mathbf{Q.E.D.}$

Different algorithms can be derived from our generic label-correcting algorithm by using different ways to choose the node that exits the candidate list at each iteration. In particular, if a node with the smallest label is chosen to exit the list, then we obtain an analog of Dijkstra's algorithm. The key property of this algorithm is that once a node exits the list, it never re-enters it, as shown in [10]. Other possibilities are to organize the list according to the schemes described in [2] and [6]. In Section III we discuss implementations of such schemes and give some computational results.

III. Implementing Dijkstra-Like and Label-Correcting-Like Algorithms

In this section we compare the Dijkstra-like algorithm discussed above and two recently developed label-correcting algorithms (the SLF–LLL and the SLF–LLL–Threshold methods to be presented below). The Dijkstra algorithm performs at most one iteration per node but requires some extra overhead per iteration, in order to determine a smallest-label node within the candidate list. In the label-correcting methods, the number of iterations is larger than for the Dijkstra algorithm, but the overhead is smaller per iteration. The computational results show that there are cases where the label-correcting algorithms outperform the Dijkstra-like algorithm. This is encouraging for one more reason: label-correcting algorithms are parallelizable, whereas the Dijkstra algorithm is not. The recent paper of Bertsekas *et al.* [6] explores the parallelization of the label-correcting algorithms we consider for the case of deterministic shortest path problems. Their computational results show that parallelization leads to excellent speedup. In any case, the algorithms that we present here are much faster than the traditional algorithms described by Kushner and Dupuis in [7], where the nodes are picked for iteration according to some fixed order.

A. The Algorithms

We describe the algorithms that we tested, and we discuss several issues related to their efficient implementation.

The Dijkstra Algorithm: In our implementation, we maintain the candidate list as a binary heap where the top node is the node with the smallest label. At each iteration, this node is removed from the list, and as discussed above, this node becomes permanently labeled and never enters the heap again. If the label of a neighbor is reduced, this neighbor is included in the binary heap at the appropriate position. Observe that if a node i becomes permanently labeled, then at most three of its neighbors may enter the binary heap, since at least one of its neighbors, the one having smaller label than node i, is permanently labeled. The iterations proceed until the list is empty. Our binary heap code is based on the SHEAP code for deterministic shortest paths of Gallo and Pallotino [8]. In the presentation of our results we will refer to the implementations of the Dijkstra-like algorithm with the prefix "DIJ-".

The Label Correcting-Like Algorithms:

1) *The SLF–LLL Method*: In this algorithm, the candidate list is maintained as a queue. Nodes enter the list according to the following *(SLF)* criterion first proposed in [2]: *whenever a node i not already in the list enters the list, its label $V(i)$ is compared to the label $V(j)$ of the top node j of the list. If $V(i) \leq V(j)$, node i is inserted at the top of the list; otherwise node i is inserted at the bottom of the list.*

   This criterion for insertion in the list is combined with another simple criterion for the removal of a node from the list. The removal criterion is called *(LLL)* and was proposed in [6]. According to this criterion, the node at the top of the list is considered. If its label is larger than the average of the labels of all the nodes in the list (defined as the sum of the labels divided by the number of nodes), then the node is reinserted at the bottom of the list and its successor node in the list is extracted. This procedure continues until a node with label less than or equal to the average of the labels of all the nodes in the list is found. The labels of the neighboring nodes of the node removed from the list are recomputed, and the corresponding nodes are inserted in the list according to the SLF criterion.

2) *The SLF–LLL-Threshold Method:* This is a combination of the SLF–LLL method with the threshold method of Glover *et al.* [9] (see [6]). Here, the candidate list maintained by the SLF–LLL method is divided in two lists; the nodes on the first list are the ones that have labels less than or equal to some threshold value. Nodes are inserted in both lists according to the SLF criterion. Nodes are removed only from the first list according to the LLL criterion. When the first list becomes empty, the threshold is appropriately increased and nodes with label less than or equal to the new threshold are removed from the second list and inserted to the first list according to the SLF criterion. The method we chose for increasing the threshold is the following: initially, the threshold is set to the minimum node cost $\min_{x \in S} \ g(x)$ plus a user-chosen percentage of the maximum node cost $\max_{x \in S} \ g(x)$. Each time the first list empties, the threshold is increased by the user-chosen percentage of the maximum node cost. If this increase was not sufficient to transfer any nodes of the second list to the first list, the threshold is set equal to the minimum node label in the second list plus the user-set percentage of the maximum node cost. In the presentation of the computational results we will refer to the SLF–LLL-Threshold algorithm with the prefix "SLF–LLL–TH-".

In all the above implementations we have used certain tests that help avoid unnecessary recomputations of the labels of the nodes. The reduction in computation obtained by these tests is significant. Specifically, recall that the label of a node x is computed by performing minimizations along four quadrants (parameter α), and the minimization in a particular quadrant is based on the labels of two neighboring nodes. In particular, we make the following observations.

- When a node exits the list, only its neighbors with larger labels need to have their label recomputed (see Proposition 1).
- The recomputation of the label of a node needs to be made only in the quadrants that involve the neighboring node that exited the list at the beginning of the current iteration.

TABLE I

COMPUTATIONAL RESULTS ON THE MACINTOSH POWERBOOK 170 AND Alpha-Dec. ON EACH PROBLEM LINE, THE NUMBERS ON TOP ARE THE TIMES IN SECONDS ON THE DIFFERENT MACHINES, AND THE NUMBERS AT THE BOTTOM ARE THE ITERATIONS/LABEL CALCULATIONS/SIMPLIFIED LABEL CALCULATIONS. ALL PROBLEMS WERE GENERATED BY THE GRIDQUAD PROGRAM. THE COST ASSIGNED TO A GRID POSITION (x, y) WHICH IS NOT A BOUNDARY POINT OR AN OBSTACLE IS GIVEN BY $1001 - 1000(10(x-x_0)^2 + 40(y-y_0)^2/10(x_0+1)^2 + 40(y_0+1)^2$, WHERE (x_0, y_0) IS THE CENTRAL POINT OF THE GRID

Problem	DIJ-NO-NEIGH	DIJ-NEIGH	SLF-LLL-NO-NEIGH	SLF-LLL-NEIGH	SLF-LLL-TH-NO-NEIGH	SLF-LLL-TH-NEIGH
150X150	22.02/0.38	20.8/0.52	25.45/0.42	23.65/0.4	19.42/0.316	17.37/0.285
No Obst.	21904/43230/43796	21904/21721/43796	32976/61605/69195	32976/32976/69195	23426/45805/47155	23426/45805/47155
150X150	20.73/0.32	19.60/0.45	20.20/0.33	18.37/0.31	18.58/0.3	16.62/0.278
3 Obst.	21474/41469/42977	21474/20873/42977	26131/49214/53138	26131/29234/53138	22733/43356/45980	22733/23425/45980
100X200	19.27/0.29	18.08/0.35	22.95/0.37	22.18/0.35	17.35/0.278	15.57/0.237
No Obst.	19404/38289/38787	19404/19193/38787	30729/54588/67360	30729/37368/67360	20976/22510/42286	20976/29207/42286
50X250	11.47/0.169	10.73/0.177	14.48/0.244	14.18/0.23	10.52/0.165	9.45/0.137
No Obst.	11904/23232/23794	11904/11650/23794	19505/33574/43556	19505/24553/43556	12787/24674/25706	12787/13639/25706
100X225	21.70/0.33	20.47/0.388	26.67/0.427	26.15/0.4	19.82/0.31	17.82/0.27
No Obst	21854/43057/43719	21854/21609/43719	35777/63405/78777	35777/44971/78777	23997/46674/48424	23997/25879/48424
100X225	20.42/0.32	19.25/0.406	19.60/0.371	18.08/0.346	19.58/0.31	17.8/0.28
3 Obst.	21387/41191/42783	21387/21387/42783	25590/47856/52116	25590/29373/52116	24015/45779/48609	24015/25916/48609
300X300	*/2.05	*/2.63	*/1.73	*/1.71	*/ 1.35	*/1.3
No Obst.	88804/176412/177614	88804/88450/177614	132476/248243/280171	132476/155172/280171	96313/189826/194300	96313/104423/194300
300X300	*/.86	*/2.44	*/1.5	*/1.476	*/1.34	*/1.3
3 Obst	87924/172819/175927	87924/86716/175927	113448/216430/232960	113448/131261/232960	95326/185193/193039	95326/01489/193039
500X500	*/6.99	*/8.49	*/5.25	*/5.38	*/3.95	*/3.9
No Obst.	248004/493991/496035	248004/247471/496035	394289/742420/830920	394289/344023/830920	268465/531269/541007	268465/288964/541007

- If the labels of the two neighboring nodes of node x in a particular quadrant have remained the same since the last time that the label of x was calculated, no minimization using that quadrant is needed. We consider implementations of the algorithms both with and without keeping track of the labels of neighboring nodes. In the presentation of the computational results we will distinguish the two implementations of an algorithm by appending the suffixes "-NEIGH" and "-NO-NEIGH," respectively, to the name of the algorithm.
- Let two neighboring nodes of node x in a particular quadrant be i and j. If i has a greater label than the current label of x, then the simpler label update $V(x) := \min\{V(x), hg(x) + V(j)\}$ can be used. Such a label evaluation will be referred to as *simplified.*

The implementations of all algorithms solve the problem of finding the optimal costs from all nodes $x \in S$ to some border point $\overline{x}$. In particular, we have set $f(\overline{x}) = 0$, and for all border nodes $x \neq \overline{x}$ we have set the cost $f(x)$ to be a very large number.

B. The Test Problems

We implemented a test problem generator called GRIDQUAD. The test problems generated are grids resulting from a square discretization of parallelograms with sides whose length is an integer multiple of the discretization step h. Thus G is the interior of the parallelogram and ∂G is the border of the parallelogram. The corresponding discretizations S and B are easy to obtain. Each node $x \in S$ has an associated cost which is the discretized version of the positive cost function $r(x)$ defined on G. The program GRIDQUAD uses a quadratic function for $r(x)$. The cost of all boundary nodes $x \in B$ is (effectively) infinity except for the two neighbors of the top right-hand corner of the parallelogram border. To make the test problems more interesting, we have introduced *obstacles,* i.e., interior points of the parallelogram with infinite cost. These are considered to be part of ∂G. In our test problems, obstacles occur in rows; all the points on a row have infinite cost except for a segment of length kh where k is some positive integer. Obstacle rows are equally spaced at a distance which is an integer multiple of the discretization step h.

C. Test Results

We present in Table I the results of some of the computational experiments. The test problems are described by giving the dimensions of the grid and the number of obstacles. Costs of interior points are in the range [1–1000] and are generated by a quadratic function which has its maximum at the center of the grid. For each algorithm we tested two implementations, one that keeps track of the values of neighboring nodes in order to avoid unnecessary computations (suffix -NEIGH) and one that does not maintain such information (suffix -NO-NEIGH). We report running times on a Macintosh Powerbook 170 and on an Alpha-Dec computer running a version of UNIX. We also report the number of iterations, label calculations, and simplified label calculations performed by each algorithm to facilitate the comparison between the algorithms.

Our computational experiments indicate that the label-correcting algorithms are competitive with the Dijkstra algorithm. The SLF–LLL–TH algorithm outperforms consistently the Dijkstra algorithm, while the SLF–LLL algorithm outperforms the Dijkstra algorithm on the Alpha-Dec computer and is a bit slower than the Dijkstra algorithm on the Macintosh Powerbook 170. On problems with obstacles, all algorithms have similar running times. This is because the addition of obstacles has an effect similar to breaking the problem to several smaller-sized problems. Therefore, the data structures maintained by the algorithms for the choice of the node to exit the list contain few nodes and thus need less overhead, leading to similar running times. The differences in running times are more evident on problems without obstacles.

We do not present any computational comparisons with traditional Gauss–Seidel methods that cycle through the nodes in a fixed order. Our experiments have confirmed that these methods require much greater computation time than the methods discussed in this paper, often by an order of magnitude or more.

One final observation is that times on the Alpha-Dec seem to favor the SLF–LLL methods more than on the Mac. The extra iterations that the label-correcting algorithms require are outweighed on the Alpha-Dec by the amount of time spent on maintaining the heap in the Dijkstra method. A similar effect is observed when we keep extra data to reduce calculations (-NEIGH versus -NO-NEIGH). Although the number of time-consuming minimizations as in (1) is reduced by keeping labels of neighbors, the running time is not improved. This may be due to a higher penalty for memory access operations than for computations in the Alpha-Dec.

D. Conclusions

Our computational results indicate that the methods presented in this paper are very efficient. They are apparently much faster than traditional Gauss–Seidel methods that cycle through the nodes in a fixed order. Furthermore, the label-correcting methods developed are parallelizable as in [6] and will likely lead to efficient parallel algorithms. Finally, the label-correcting methods we presented may also be used in the case where the cost function is of the form $r(x, u)$. In this case the theory of [10] cannot be applied and a Dijkstra-like algorithm is not possible. However, the label-correcting algorithms we proposed can be used as heuristics that specify the order in which the label updates are performed. Their efficient implementation is an interesting subject for further research.

REFERENCES

[1] D. P. Bertsekas, *Linear Network Optimization.* Cambridge, MA: M.I.T. Press, 1991.

[2] ——, "A simple and fast label correcting algorithm for shortest paths," *Networks,* vol. 23, pp. 703–709, 1993.

[3] ——, *Dynamic Programming and Optimal Control,* vol. II. Athena Sci., 1995.

[4] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation.* Englewood Cliffs, NJ: Prentice Hall, 1989.

[5] ——, "An analysis of stochastic shortest path problems," *Math. Ops. Res.,* vol. 16, pp. 580–595, 1991.

[6] D. P. Bertsekas, F. Guerriero, and R. Musmanno, "Parallel asynchronous label correcting methods for shortest paths," LIDS-P-2250, M.I.T., Lab. Inform. Decision Syst. Rep., May 1994; also *J. Optim. Theory Appl.,* vol. 88, pp. 297–320, 1996.

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

[8] G. S. Gallo and S. Pallotino, "Shortest path algorithms," *Ann. Ops. Res.,* vol. 7, pp. 3–79, 1988.

[9] F. Glover, R. Glover, and D. Klingman, "The threshold shortest path algorithm," *Networks,* vol. 14, no. 1, 1986.

[10] J. N. Tsitsiklis, "Efficient algorithms for globally optimal trajectories," *IEEE Trans. Automat. Contr.,* vol. 40, pp. 1528–1538, 1995.

Stable Inversion for Nonlinear Nonminimum-Phase Time-Varying Systems

S. Devasia and B. Paden

***Abstract*— In this paper, we extend stable inversion to nonlinear time-varying systems and study computational issues—the technique is applicable to minimum-phase as well as nonminimum-phase systems. The inversion technique is new, even in the linear time-varying case, and relies on partitioning (the dichotomic split of) the linearized system dynamics into time-varying, stable, and unstable, submanifolds. This dichotomic split is used to build time-varying filters which are, in turn, the basis of a contraction used to find a bounded inverse input-state trajectory. Finding the inverse input-state trajectory allows the development of exact-output tracking controllers. The method is local to the time-varying trajectory and requires that the internal dynamics vary slowly; however, the method represents a significant advance relative to presently available tracking controllers. Present techniques are restricted to time-invariant nonlinear systems and, in the general case, track only asymptotically.**

***Index Terms*—Dynamics, feedforward systems, inverse problems, tracking.**

I. INTRODUCTION

In this paper, the stable inversion problem for nonlinear time-varying systems is solved. The approach is quite novel in that it applies to nonminimum phase systems—even the linear version of our approach is new in the time-varying context. The basic idea is to compute the inverse dynamics, through a contraction, to find an input-state trajectory that achieves a desired output trajectory. To develop output tracking controllers, the input trajectory (found through inversion) can be used as a feedforward input signal in conjunction with any conventional feedback control law that stabilizes the inverse state trajectory [1]. The present work completes a line of research which was motivated by the inversion of time-invariant articulated flexible structure dynamics [2] and extends our work on inversion of general affine-in-control time-invariant nonlinear systems [3].

System inversion is key to recent results in exact-output tracking for autonomous systems [1], [3]–[5]. This paper extends these results to exact-output tracking of time-varying systems. The output tracking problem has a long history marked by the solution of the linear time-invariant regulator problem by Francis [6] and the nonlinear time-invariant generalization made by Isidori and Byrnes [7]. The linear regulator is designed by solving a manageable set of linear matrix equations, whereas the nonlinear regulator requires the nontrivial solution of a first-order partial differential algebraic equation. These approaches asymptotically track any member in a given family of output signals. More recently, there have been refinements of these approaches. Huang and Rugh [8] used a formal Taylor series expansion of the Isidori–Byrnes partial differential equation and gave a sufficient condition for solvability. Krener [9] extended this work by providing necessary and sufficient conditions for the term-by-term

Manuscript received May 29, 1996. This work was supported through National Science Foundation under Grant MSS-9216690, the NASA Ames Research Center under Grant NAG-2-1042, and the Astro Aerospace Corporation.

S. Devasia is with the Mechanical Engineering Department, University of Utah, Salt Lake City, UT 84112 USA (e-mail: santosh@me.utah.edu).

B. Paden is with the Mechanical Engineering Department, University of California, Santa Barbara, CA 93106 USA.

Publisher Item Identifier S 0018-9286(98)01186-6.