A min-max version of Dijkstra's algorithm with application to perturbed optimal control problems

Marcus von Lossow *

Mathematisches Institut, Universität Bayreuth, 95440 Bayreuth, Germany.
This work was supported by the international doctorate program "Identification, Optimization and Control with Applications in Modern Technologies" of the Elite Network of Bavaria, Germany.

In this paper the computational complexity of a newly developed min-max version of Dijkstra's algorithm for solving the shortest path problem in a directed weighted hypergraph is analyzed. Together with a set oriented discretization of the state space this algorithm provides a new numerical approach for the construction of an optimally stabilizing feedback under perturbations.

1 Problem formulation

Definition 1.1 Let $\mathcal{P}$ be a finite set, the set of *nodes*. Let E be a subset of $\mathcal{P} \times 2^{\mathcal{P}}$, the set of *hyperedges*.
The pair $(\mathcal{P}, E)$ is called *directed hypergraph*.
A *weighted directed hypergraph* is a hypergraph together with weights $\mathcal{G} : E \to [0, \infty)$.

This hypergraph concept generalizes the usual directed graph: In a directed graph an edge leads from one node to another node. In a directed hypergraph a hyperedge leads from one node to a set of nodes.

The shortest path problem in a weighted directed hypergraph is to find a shortest path from every node to a set of destination nodes with maximization over all edges of a hyperedge (cf. [4]).

2 Min-max version of Dijkstra's algorithm

Dijkstra's algorithm (cf. [5]), developed in the 1950s, is one of the most efficient known algorithms for the shortest path problem in a directed weighted graph (cf. [6, p. 103]). In [3] the following extended version of Dijkstra's algorithm for the shortest path problem in a directed weighted hypergraph was proposed:

Algorithm 2.1 Let $(\mathcal{P}, E)$ be a directed hypergraph with hyperedge weights $\mathcal{G} : E \to [0, \infty)$ and $\mathcal{D} \subset \mathcal{P}$ the set of destination nodes.

1. for each P in $\mathcal{P}$ set $V(P) := \infty$
2. for each $P \in \mathcal{D}$ set $V(P) := 0$
3. $\mathcal{Q} := \mathcal{P}$
4. while $\mathcal{Q} \neq \emptyset$
5. $\quad P := \operatorname{argmin}_{P' \in \mathcal{Q}} V(P')$
6. $\quad \mathcal{Q} := \mathcal{Q} \backslash \{P\}$
7. $\quad$ for each $(Q, \mathcal{N}) \in E$ with $P \in \mathcal{N}$
8. $\quad\quad$ if $\mathcal{N} \subset \mathcal{P} \backslash \mathcal{Q}$
9. $\quad\quad\quad$ if $V(Q) > \mathcal{G}(Q, \mathcal{N}) + V(P)$ then
10. $\quad\quad\quad\quad V(Q) := \mathcal{G}(Q, \mathcal{N}) + V(P)$

3 Computational complexity

The computational complexity of this algorithm depends on the used data structures. For the efficiency of the algorithm it is necessary to have an efficient approach to determine the minimum in step 5. Storing the set $\mathcal{Q}$ in a heap is suitable for this problem.

In the following the computational complexity of Algorithm 2.1 is analyzed using a binary heap and a Fibonacci heap (cf. [6, p. 99]).

Theorem 3.1 *Let $(\mathcal{P}, E)$ be a directed weighted hypergraph and M the maximum number of edges in the hyperedges. Then Algorithm 2.1 has the following computational complexity:*

- $O(|\mathcal{P}| \cdot \log(|\mathcal{P}|) + |E| \cdot \log(|\mathcal{P}|) + |E| \cdot M^2))$ *using a binary heap*
- $O(|\mathcal{P}| \cdot \log(|\mathcal{P}|) + |E| M^2)$ *using a Fibonacci heap*

* Corresponding author E-mail: marcus.vonlossow@uni-bayreuth.de, Phone: +49 921 55 3282, Fax: +49 921 55 5361

P r o o f. The initializing steps 1–3 require an effort of $O(|\mathcal{P}|)$. For every $P \in \mathcal{P}$ step 5 and 6 are executed once. The effort for restoring the heap property is $O(\log(|\mathcal{P}|))$. Step 7 is executed for every hyperedge at most M times. Checking the condition in step 8 has an effort of $O(M)$. For every hyperedge $\mathcal{N}$ condition 9 is satisfied at most once, so there are at most $|E|$ reductions. The effort for restoring the heap property is $O(log(|\mathcal{P}|))$ using a binary heap and $O(1)$ using a Fibonacci heap. Altogether the computational complexity of Algorithm 2.1 is the following:

- $O(|\mathcal{P}|+|\mathcal{P}|\cdot\log(|\mathcal{P}|)+|E|\cdot M^2+|E|\cdot\log(|\mathcal{P}|)) = O(|\mathcal{P}|\cdot\log(|\mathcal{P}|)+|E|\cdot\log(|\mathcal{P}|)+|E|\cdot M^2))$ using a binary heap
- $O(|\mathcal{P}|+|\mathcal{P}|\cdot\log(|\mathcal{P}|)+|E|\cdot M^2+|E|)) = O(|\mathcal{P}|\cdot\log(|\mathcal{P}|)+|E|M^2)$ using a Fibonacci heap

□

Remark 3.2

1. If $M = 1$, this means the hypergraph is a graph, then the computational complexity of Algorithm 2.1 is the same as of the classical Dijkstra's algorithm.
2. A M^2 appears in the formula of the computational complexity of Algorithm 2.1, but in most cases M is very small compared to $|\mathcal{P}|$ and $|E|$.

4 Application to perturbed optimal control problems and a numerical example

Algorithm 2.1 can be applied to perturbed optimal control problems in the following way. Every state (or every box after discretization, if the state space is not finite) corresponds to a node. One state and one control and some perturbations correspond to a hyperedge, the weight of a hyperedge is the cost of the control. For more details confer [3]. This approach is an extension of a new numerical method for the construction of an optimally stabilizing feedback developed in [1, 2].

For numerical computations it is necessary to use testpoints in order to compute the image of the boxes. Minimizing over the testpoints leads to a lower bound of the optimal value function, maximizing over the testpoints to an upper bound of the optimal value function.

The following example from [7], the "Homicidal chauffeur game", is a differential game. After discretization in space and time a hyperedge corresponds to one control of the first player and all possible controls of the second player. The intention of the first player, a chauffeur in a car, whose controls are denoted by $u \in [-1, 1]$, is to catch a pedestrian, the second player, whose controls are denoted by $w \in [-\pi, \pi]$. This system has the following equation:

$$\begin{aligned} \dot{x_1} &= -ux_2 + \alpha\sin(w) \\ \dot{x_2} &= -1 + ux_1 + \alpha\cos(w) \end{aligned}$$

The following pictures show a numerically computed lower and upper bound for the game value function, the time necessary to catch the pedestrian depending on the starting position.

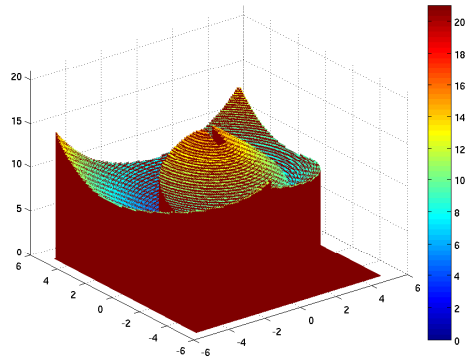

Fig. 1 lower bound, using 512^2 boxes

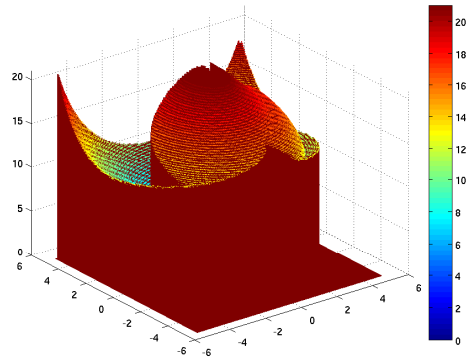

Fig. 2 upper bound, using 512^2 boxes

References

[1] O. Junge and H. M. Osinga, ESAIM Control Optim. Calc Var. **10**, 259–270 (2004).
[2] L. Grüne and O. Junge, Systems Control Lett. **54**, 169–180 (2005).
[3] L. Grüne and O. Junge, JOTA **136** (2008, to appear).
[4] G. Gallo, G. Longo, S. Nguyen, and S. Pallottino, Disc. Appl. Math. **42**, 177–201 (1993).
[5] E. W. Dijkstra, Numer. Math. **1**, 269–271 (1959).
[6] A. Schrijver, Combinatorial Optimization, Vol. A, Algorithms and Combinatorics, Vol. 24 (Springer, Berlin, 2003).
[7] T. Başar and G. J. Olsder, Dynamic Noncooperative Game Theory, second edition, Classics in Applied Mathematics (SIAM, Philadelphia, 1999).