Handling Multiple Certificate Failures in Kinetic Data Structures

Reuben Diaz* Veer Panchal*
Duru Türkoğlu*

October 21, 2016

Abstract. Kinetic Data Structures (KDSs) provide a framework for maintaining certain properties of geometric objects moving along predefined trajectories. A KDS identifies a combinatorial structure that does not depend on the exact position of the objects, but rather on their relationships to one another. These relationships, known as certificates, prove the correctness of the combinatorial structure; as long as they are valid, the KDS will be valid as well. The known trajectories allow us to identify when each certificate would fail, enabling us to advance time up to the next certificate failure without damaging the structure's integrity. When a certificate finally fails, a KDS provides kinetic update methods to repair the broken combinatorial structure, and to update the set of certificates so that the new certificate set proves the validity of the repaired structure. Since their introduction, it is widely assumed that certificate failures must be handled individually. This assumption places a restriction requiring total ordering of the certificates as they fail. In this paper, we remove this restriction by developing an approach to resolve multiple failures at once. We also evaluate the effectiveness of our approach experimentally, and show that when the number of certificate failures in a kinetic update changes, the number of operations per affected point remains logarithmic.

Introduction. Since their introduction almost two decades ago by Basch, Guibas and Hershberger [2], many KDSs have been developed and analyzed. In their work, they note that to handle multiple certificate failures, a KDS must identify and use partially correct structures. Unfortunately, identifying such structures has been challenging, and as a result, almost all KDSs rely entirely on handling one certificate failure at a time [1]. This is necessary so that the KDS can rely on the correctness of the rest of the structure. However, this requirement is problematic, failures can occur so close to one another that ordering their failure times can be very costly, and even impossible when multiple certificates fail at exactly the same time.

In this paper, we offer an approach to handle multiple certificate failures simultaneously. We define a timeline for fixing the structure, so that we can partially repair the broken structure at each timestep, and prove some invariants about the partially repaired structures. More specifically, we identify the objects which have failed certificates, and using our timeline, we schedule the broken objects to be repaired at their corresponding timesteps. When processing the objects scheduled at a particular timestep, we first repair them, possibly causing other parts of the structure to break down. We then schedule those broken parts of the structure alongside the already broken objects at their corresponding timesteps. We ensure that this procedure functions properly by maintaining the invariant that any structure broken down while conducting repairs at a given timestep does not require going back in time, and thus can always be repaired at a timestep later in the timeline. Once we complete the repairs at every timestep, the entire structure will be correct, following from our invariants.

Kinetic Tournaments. We illustrate our approach on the kinetic tournament data structure, which maintains the largest number in a given set of numbers [2]. The kinetic tournament employs a divide-and-conquer algorithm, in which the set is recursively divided into halves until reaching subsets of size one, which form the leaves of a binary tree. The numbers then "compete" against one another — the larger number in a pair at level ℓ "wins" the competition and becomes a node at level $\ell+1$ which is the parent of both numbers in the competition. The comparisons from each competition become the certificates, proving the correctness of the structure. Restricted to a method that only handles single certificate failures, we would need to advance time to the first certificate failure, i.e., when a competition's winner is no longer correct. Then, we would repair the structure by correcting the outcome of that competition, and any other incorrect competitions along the path to root. To handle multiple certificate failures, an intuitive approach would be to fix each failed certificate one at a time. This would require us to traverse the structure once for each failure. Instead, we identify each level in the tournament as a timestep in our kinetic update algorithm. We then schedule each failed certificate at the timestep associated with its level and repair the entire tournament bottom-up, using only one traversal. Given these timesteps, the partially-correct structure that we identify is captured in the following invariant: *after level ℓ is completed, for any number x present at level $\ell+1$, the subtree rooted at x is a valid kinetic subtournament.*

Deformable Spanners. The deformable spanners kinetically maintain a $(1+\varepsilon)$-spanner of an input point set $\mathcal{S}$ [3]. A deformable spanner is a hierarchy of representatives of $\mathcal{S}$ at geometrically varying radii: for a given radius 2^ℓ, the representative set, called the *discrete centers* and denoted $\mathcal{S}_\ell$, is a maximal subset of $\mathcal{S}_{\ell-1}$ such that any $p, q \in \mathcal{S}_\ell, p \neq q$ satisfy $|pq| > 2^\ell$ ($\mathcal{S}_\lambda = \mathcal{S}$ for some small enough λ). Intuitively, this (non-unique) hierarchy of discrete centers defines samplings at varying resolutions. To ensure the $(1+\varepsilon)$ stretch factor, a deformable spanner defines two types of edges: *neighbor* and *parent-child*. A neighbor edge connects any two close enough points at the same level, i.e., $p, q \in \mathcal{S}_\ell$ are neighbors if $|pq| \leq c \cdot 2^\ell$, where $c = 4 + 16/\varepsilon$. A parent-child edge describes a connection between two points in consecutive levels, every point $p \in \mathcal{S}_{\ell-1} \backslash \mathcal{S}_\ell$ chooses a parent $q \in \mathcal{S}_\ell$ within 2^ℓ distance; this relationship is described as p being a child of q.

A deformable spanner is constructed by inserting the points in $\mathcal{S}$ one at a time. During insertion, a point p is represented at every level of the structure (inserted into the discrete centers) for location purposes. First p is located top-down by identifying all of its neighbors at each level. Then, p is assigned as a child of its lowest possible parent at level ℓ. After this assignment, insertion

*School of Computing, DePaul University, Chicago, IL

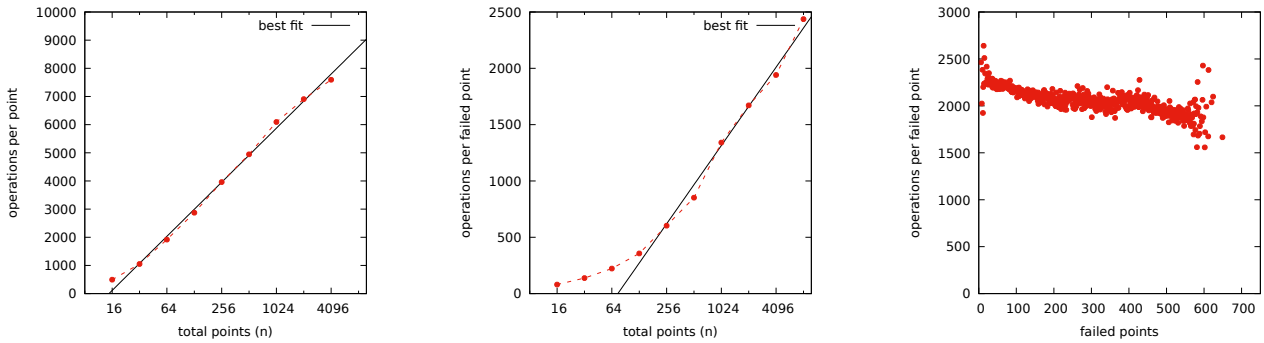

Figure 1: (a) operations per inserted point (b) operations per failed point (c) multiple certificate failures (n=8192)

of p is finalized by deleting all representations of p at levels $\geq \ell$. Four types of certificates prove the validity of the structure: *separation* certificates ensure any two points at the same level are far enough (no parent-child edge between them), *parent-child* and *neighbor* certificates ensure that existing edges are valid, and *potential neighbor* certificates ensure that two nonneighbor points (whose parents are neighbors) are far enough apart.

The authors describe the kinetic update for each type of certificate failure [3]. If a *potential neighbor* certificate fails, a neighbor edge is drawn between the two points that certificate refers to. If an *edge* certificate fails, the edge in question is deleted. If a *parent-child* certificate fails, the associated edge is deleted, and the child is raised along the path to root until it finds a suitable parent. If a *separation* certificate fails, one of the two points in question becomes the child of its former neighbor, and its children are raised as though their parent-child certificates had failed.

To handle multiple certificate failures, one must devise an algorithm which can repair the structure after a set of certificate failures which may include multiple types of certificates. To accomplish this, we define our timeline with the following five stages: *reconnect*, *locate*, *raise*, *delete*, *certify*. In the *reconnect* stage, we process failed parent-child certificates to reconnect the children to valid parents, by raising the children along the path to root until a valid parent is found. We ensure that after the reconnect stage at level ℓ is completed, all parent-child edges are valid for levels ℓ and above, and any point $p \in \mathcal{S}_\ell$ is connected to the root via parent-child edges.

We then proceed to the *locate* stage, where we process every point associated with a failed certificate level by level in a top-down fashion. When processing locate for level ℓ, we ensure that points in $\mathcal{S}_\ell$ will be separated. When processing a point p, if we can, we push all very close points down one level, and schedule them for the *raise* stage. Otherwise, we push p down and assign it and its children as children of the point which was not safe to remove from level ℓ. If p is not pushed down and remains at level ℓ, we then identify new neighbors of p, and schedule all children of p for the next timestep so that the broken structure can be repaired further. After the locate stage at level ℓ is completed, we ensure that any two points at level ℓ are separated and all neighbor edges are identified. We also prove a proximity result for those invalid parent-child relationships: for a parent $p \in \mathcal{S}_\ell$ with child q ($\in \mathcal{S}_{\ell-1}$), $|pq| \leq 2 \cdot 2^\ell$.

Next, we process the *raise* stage in a bottom-up fashion. We repair every point p whose parent is invalid after the locate stage concludes. When processing stage for level ℓ, we look for a suitable parent of p at level $\ell + 1$. If our search fails, we raise p, establish any new neighbor relationships, and schedule p for the raise stage at level $\ell + 1$. After the raise stage at level ℓ is completed, we ensure that the substructures at and below level ℓ are valid deformable spanners containing the subset of points reachable from their root via parent-child edges. Finally, in the *delete* stage, we trim any excess nodes, and in the *certify* stage, we update the set of certificates to prove the updated deformable spanner structure.

Implementation and Experiments. We implement the aforementioned kinetic update algorithm for the deformable spanners and set $\varepsilon = 1$. We use Sturm sequences to identify the certificate failure times, and we define a smallest time increment δ to establish a lattice for advancing time forward. By varying δ, we provide experimental results that can handle tens of thousands of certificate failures at once, all capable of handling each failure per point in logarithmic time. Our experiments verify that insertion of a point takes logarithmic time (see Figure 1(a)). Our experiments also verify that a kinetic update involving very few certificate updates takes logarithmic time (see Figure 1(b)). Furthermore, for a fixed size of 8192 points, when multiple certificates fail, the average number of operations per failed point remains fairly constant as the number of failed points increases (see Figure 1(c)).

References

[1] Pankaj K. Agarwal, Leonidas J. Guibas, Herbert Edelsbrunner, Jeff Erickson, Michael Isard, Sariel Har-Peled, John Hershberger, Christian Jensen, Lydia Kavraki, Patrice Koehl, Ming Lin, Dinesh Manocha, Dimitris Metaxas, Brian Mirtich, David Mount, S. Muthukrishnan, Dinesh Pai, Elisha Sacks, Jack Snoeyink, Subhash Suri, and Ouri Wolfson. Algorithmic issues in modeling motion. *ACM Comput. Surv.*, 34(4):550–572, 2002.

[2] Julien Basch, Leonidas J. Guibas, and John Hershberger. Data structures for mobile data. *Journal of Algorithms*, 31(1):1–28, 1999.

[3] Jie Gao, Leonidas J. Guibas, and An Nguyen. Deformable spanners and applications. *Computational Geometry: Theory and Applications*, 35(1):2–19, 2006.