GPS Solutions (2026) 30:60
https://doi.org/10.1007/s10291-025-02020-6

RESEARCH

Efficient clustering of GNSS stations for processing using double differences

Shane P. Grigsby[1,2] · **Demián D. Gómez**[2]

Received: 10 July 2025 / Accepted: 30 December 2025 / Published online: 21 January 2026

Abstract
The rapid growth of GNSS networks poses significant challenges for efficiently processing large datasets using double-difference techniques. In this study, we introduce a novel clustering algorithm, *qmeans*, which is based on bisecting k-means, to partition GNSS networks into smaller, manageable subnetworks or clusters for double-difference processing. We explore the trade-offs between cluster size, computational cost, and solution quality using a comprehensive dataset of approximately 1200 stations distributed across México, the United States, and Canada. Our results demonstrate that partitioning the network into clusters of 20–30 stations with 6 overlap stations between clusters can reduce processing time by ~20%, while larger clusters of 40–50 stations with 10 overlap stations slightly improve solution precision. We show that the number of shared stations between clusters impacts both the computational efficiency and the precision of the final solution, with higher counts leading to better precision but also increased processing time. The *qmeans* algorithm is integrated into the open-source Parallel.GAMIT software, offering a scalable, flexible solution that can be applied to large GNSS networks. Our work sets a foundation for selecting optimal subnetwork sizes based on specific needs of a GNSS processing project, enabling faster processing without significantly sacrificing solution quality.

Keywords GNSS processing · Double-difference processing · Clustering · GNSS network · GNSS solution scatter · Segmentation

Introduction

Global GNSS networks providing open data have grown geometrically since the early 1990s from a few hundred continuously operating stations to over 20,000 known stations today. This number is even larger when considering intermittently occupied sites for survey campaigns and private stations that do not share their data openly. The vast quantity of GNSS observations generated daily is essential for numerous scientific, engineering, and mapping applications, requiring automated processing to manage the computational burden efficiently (Gómez et al. 2024).

GNSS double-difference processing techniques remain a standard approach for high-precision positioning and network adjustments. To our knowledge, network design for large scale differential processing is a topic that has not been thoroughly discussed in geodesy. Yet, one of the main challenges in processing large GNSS networks at Ohio State University (OSU) has been efficiently dividing the network into subnetworks for processing in GAMIT/GLOBK (Herring et al. 2018). Currently, our GNSS database has over 6100 stations adding up to ~19 M station-days, with a typical GNSS processing project at OSU containing about 2000 simultaneous stations. Current computational power restricts double-difference processing to a maximum of approximately 80 simultaneous stations per session. Additionally, processing time increases as s^2, where s is the number of sites in the processing session, creating a significant bottleneck in the computation of large network solutions. As GNSS networks continue to expand, this constraint poses real challenges for large-scale analyses, requiring partitioning strategies,

Shane P. Grigsby and Demián D. Gómez have contributed equally to this work.

✉ Shane P. Grigsby
shane.grigsby@colorado.edu

[1] Cooperative Institute for Research in Environmental Sciences, University of Colorado, Boulder, CO, USA

[2] Division of Geodetic Science, School of Earth Sciences, Ohio State University, Columbus, OH, USA

hierarchical processing schemes (International Organization for Standardization 2020), or alternative methodologies to integrate observations from thousands of stations without excessive computational cost. While other alternatives to differential processing exist, such as Precise Point Positioning and undifferenced processing (Geng and Mao 2021), we concentrate on the processing constraints imposed by the double differences method, which still provides the best accuracy and precision for daily GNSS solutions (Gómez et al. 2022). There are also method to more efficiently compute solutions in parallel (e.g. Cui et al. 2021) but in these cases the problem of splitting stations into subnetworks still remains.

The increasing size of GNSS networks requires scalable solutions that balance computational feasibility with the need for accurate and consistent geodetic products. Addressing these limitations is crucial for maintaining the reliability and improving GNSS-derived positioning, velocity fields, and global reference frame realizations, such as the International Terrestrial Reference Frame (ITRF, Altamimi et al. 2023), as well as regional realizations like the Geodetic Reference System for the Americas (Alves Costa et al. 2022). In this paper, we present a new algorithm we term *qmeans* clustering, based on the sklearn (Pedregosa et al. 2011) bisecting-kmeans, to partition a network of GNSS stations into subnetworks of fewer stations for efficient processing. Our results show that subnetworks of 20–30 stations provide an optimal partitioning to minimize the processing time while slightly increasing the scatter level. In contrast, subnetworks with 40–50 stations increase the computation time with a slight improvement in scatter. These results establish a foundation for selecting subnetwork sizes based on the objectives of a GNSS processing project, enabling a balance between processing speed and solution precision.

Throughout this work, we refer to subnetworks of GNSS stations as 'clusters', and stations that are shared across clusters as *overlap stations*, in contrast to the more typical *tie station* language frequently used by the GNSS community. While the definition of *tie stations* in the context of tying differential solutions is loosely constrained, typically, *tie stations* refer to an exchange of stations between clusters, while our use of *overlap* refers to expansion of a given cluster to include stations from neighboring clusters. The distinction here is in the mutual reciprocity implied by *tie stations*–i.e., 'tying' clusters A and B together with *tie stations* would involve adding N stations from A into B, while also adding N stations from B back into A. In contrast, *overlap stations* may or may not have any reciprocal relationship between specific clusters: if cluster A 'overlaps' into cluster B by N stations, cluster B may then instead overlap into another cluster that isn't A. We use *overlap stations* because we are able to define guaranteed algorithmic performance when discussing the processing behavior in the context of single clusters and their expansion, and further note that the number of *overlap stations* for a single cluster can be conceptually thought of as half of the number of analogous *tie stations*.

The remaining sections of this manuscript are divided as follows: Methods describes the *qmeans* clustering algorithm and the methodology used to connect neighboring clusters. Results show a one-year ~ 1200 station network GAMIT run using two cluster sizes and three levels of station overlap, and the impact of cluster size and overlap in the weighted root mean square (wrms) scatter of the solutions. Discussion develops and discusses a model to predict the execution time of a double-difference processing session of s stations–and allows us to understand the trade-off between cluster size, shared stations between neighboring clusters, and session execution time.

Methods

Segmentation of points in space is a problem typically addressed using unsupervised learning, specifically unsupervised clustering. The problem is unsupervised because we do not have an a priori set of labels to train from, and need to accomplish the segmentation into clusters using only the information inherent in the location data of our stations at run-time. This also means that our cluster labeling and assignment is independent between runs, although the inherent spatial structure may lead to some persistent clusters that are forced to reform due to geometry constraints imposed by coasts, national borders, and other human or natural factors that influence station placement and density. For our particular use case in defining subnetworks of GNSS networks, we have an additional constraint that is not typically present in other clustering problems: we need to define our partitions with overlapping point membership between the subnetworks, so that the reference frame maintains consistency between adjacent subnetworks.

Existing clustering techniques include k-means variants, spectral methods, agglomerative techniques, and density-based methods. None of these techniques address our problem completely. The k-means techniques split points into groupings of equal variance, minimizing the within-cluster sum of squares (termed 'inertia') across the dataset (Forgy 1965; Lloyd 1982)—however, k-means requires that the number of output clusters be specificized in advance as the 'k' parameter. Spectral clustering similarly requires an a priori number of clusters to be provided in advance, and additionally struggles as the number of clusters to segment grows (Ng et al. 2001). Agglomerative techniques initiate each observation as a cluster and iteratively

'link' neighboring clusters together as merges; while these methods are hierarchical, they also tend to lead to unbalanced cluster membership, a behavior that is at odds with our downstream processing goals. Density based methods include popular algorithms such as DBSCAN (Ester et al. 1996; Schubert et al. 2017) and OPTICS (Ankerst et al. 1999), and can be conceptually thought of as convolving a density kernel over the input data points, and then passing a plane through the output heatmap topology of density where the isolated 'peaks' form distinct cluster objects—however, this leaves a subset of points (in the valleys) that are not assigned to any cluster and are simply labeled as noise. Other clustering algorithms such as Mean Shift and Affinity Propagation have poor computational complexity, and none of the above methods address the need for overlapping membership between clusters.

As our base clustering model, we selected *bisecting-kmeans* clustering, a hierarchical partitioning variant of traditional kmeans (Steinbach et al. 2000), which we modified and improved to adjust it to our needs. More specifically, we choose the variant from sklearn, which itself uses and modifies *kmeans++* (Arthur and Vassilvitskii 2006). Our algorithm modifies the existing sklearn implementation with a new set of termination conditions which eliminates the *k* parameter requirement of specifying the number of output clusters at algorithm run-time. Since the '*k*' in 'kmeans' refers to the number of output clusters, we term our modified clustering algorithm *qmeans* to distinguish it from its parent algorithms. Our *qmeans* algorithm conceptually operates opposite to the agglomerative techniques: rather than starting with all points as independent clusters that are iteratively merged, we instead start with a single cluster of all stations that is iteratively bisected. We then supplement our new *qmeans* algorithm with two post processing steps: an *overcluster* routine to expand cluster membership to adjacent station clusters, and then a *prune* step to filter subnetwork clusters that are fully redundant after cluster expansion. Note that we adopt the convention of italicizing names of *functions*, *classes*, and *parameters* that we use, inherit from, or modify.

Qmeans cluster formation

Since *qmeans* is a modification of *bisecting-kmeans*, which is itself a variant of *kmeans++* and kmeans clustering, it is useful to review the parent algorithms to understand how *qmeans* works. As mentioned above, kmeans works by dividing *N* points (i.e., our stations) into *K* non-overlapping clusters *C*, and minimizing inertia which is defined as:

$$\sum_{i=0}^{n} \min_{\mu_j \in C} \left(\|x_i - \mu_j\|^2 \right) \quad (1)$$

Where μ_j is the centroid of proposed cluster *j*, and x_i is a point coordinate from the set of all coordinates proposed for assignment within cluster *j*. In the original algorithm, this is done by selecting *k* centroid centers, and then perturbing those centers (and adjacent point membership) to iteratively minimize inertia; the sklearn implementation uses *kmeans++* which accelerates convergence by using random seeding, an enhancement which is of marginal impact when *k* is low.

For both *qmeans* and *bisecting-kmeans*, there are two nested loops iterating: an inner-loop iterating to converge on the centroid and membership proposal that will optimally bisect a given cluster 'node' within the recursion node tree, and an outer-loop iterating over nodes to bisect. For $k=N$, the *bisecting-kmeans* algorithm sets $k_{inner}= 2$, and splits the dataset after optimizing cluster centroid location and membership to minimize inertia as measured in (1). This bisection process is repeated recursively in the outer-loop by selecting the cluster with either the largest inertia or largest point membership, and splitting that cluster (using $k_{inner}= 2$), with the recursion terminating when the total number of clusters is equal to the overall number of output clusters requested by *k*. As a consequence, the *bisecting-kmeans* algorithm is hierarchical, reduces run-time complexity for high values of *k*, and produces clusters with more uniform membership. These latter two properties make *bisecting-kmeans* especially appealing for our GAMIT processing pipeline, as having lower variance in cluster membership number allows us to further optimize for run-time.

The base kmeans algorithm must effectively recalculate the entire clustering structure from scratch whenever the *k* parameter is incremented. In contrast, the bisecting variant maintains the prior *i*-th outer-loop iteration cluster centroids and membership labels in a recursion tree while swapping the current highest inertia (or largest membership) cluster node within a given outer-loop iteration for two new subcluster nodes that bisect that parent. This outer-loop iterative structure for the bisecting algorithm allows us to remove the *k* parameter entirely, and instead replace it with our own boundary conditions. Our resulting *qmeans* algorithm replaces the *k* parameter with a single new *qmax* parameter for maximum cluster size. The *qmax* parameter is a hard boundary condition that provides an algorithmic guarantee that the output clustering will include only clusters of size at or below the *qmax* parameter setting—if a node within the *qmeans* recursion tree is larger than *qmax* it is bisected, if the node is smaller than *qmax* it is not, and the algorithm terminates when there are no remaining nodes left in the tree to bisect. Our *qmeans* algorithm produces clusters with different properties and streamlines the implementation code when compared with the parent *kmeans++* and *bisecting-kmeans* algorithms. As mentioned above, *bisecting-kmeans*

chooses the next node to bisect by selecting the node with either the largest inertia or the largest point membership, and will produce a different output clustering depending on which of the two metrics is selected. For *qmeans*, the distinction is meaningless and does not impact the output clustering at all. Deciding on which cluster to bisect matters in *bisecting-kmeans* because the algorithm increments the total number of clusters at each outer-loop iteration and will therefore terminate after $k - 1$ total outer-loop iterations–this means that even though the bisection of any particular node will produce the same output subnodes from applying (1), the choice of which order to apply the fixed k bisections will produce different results. In contrast, *qmeans* is not iteration bound for the outer-loop and will apply bisections, using (1) in the inner-loop to determine output nodes until there are no nodes above *qmax* remaining to bisect. One drawback of kmeans algorithms in general is that they perform best when clusters are convex; because the technique bisects points into groupings of equal variance, this family of algorithms handles irregular shapes poorly. Low membership bisections can occur on our data inputs because of variable density and sparse station coverage–for example, across oceans with widely separated island stations in a geometry that yields a small member cluster for the tightly grouped GNSS stations over islands such as Hawai‘i, and single member clusters at distant atoll stations such as Guam. While both low cluster membership and high cluster membership present problems for our GAMIT runs, cluster expansion for overlap stations ensures that even single station clusters such as Guam will have multiple stations to form a subnetwork for a GAMIT session. Since clusters with high station membership can cause non-recoverable crashes in our processing runs due to memory errors, by design we provide algorithmic guarantees for a ceiling on the station count membership during both cluster formation from *qmeans* and expansion via *overcluster*. A final additional post processing by the *prune* function (Sect. Cluster pruning) is implemented to remove any redundant clusters that may be formed entirely by stations that overlap with other clusters.

Cluster expansion

When processing subnetworks within GAMIT, we require there to be overlapping stations between the subnetworks, and we use the *overcluster* function to ensure this overlap. The *overcluster* function takes three required parameter arguments: *overlap*, *nmax*, and *rejection threshold*. The *rejection threshold* is a distance set by default to 5000 km, and rejects any points that fall outside of that distance—this is mainly present as a sanity check for very isolated stations in polar regions or oceans, and ensures that GAMIT can form double differences on adjacent stations. While the *rejection threshold* was enabled to be user modified, we have had no reason to do so and it remains fixed for all of our processing runs. The *overlap* parameter specifies how many additional stations we want to expand our cluster by, while *nmax* sets a maximum number of stations to include from any given adjacent cluster. The cluster expansion is recursive, and uses a *balltree* spatial index from sklearn to calculate nearest neighbors (Omohundro 1989). More explicitly, we take all of the GNSS stations except for the cluster being expanded and build a spatial index, using the index to calculate the $k = 1$ nearest neighbors from each station outside of the cluster, to each member of the cluster. This intersection of cluster and non-cluster members results in a sorted list of station distances to our cluster's member stations, and we add the external station with the shortest distance as an overlap point. We then update the cluster and non-cluster membership to include/exclude this new station and rerun the nearest neighbor analysis. As we add points from neighbors to our cluster, we also track the original label of the added stations; when the sum of a given label's occurrence is equal to the *nmax* parameter, we remove that entire neighboring cluster from future consideration. This ensures that we get broad connectivity and representation from neighboring clusters. The *overcluster* expansion algorithm terminates when the number of stations added to the cluster under expansion reaches that value set by the *overlap* parameter—unless there are no stations to add under the *distance threshold* parameter and the algorithm terminates early, a condition which was not triggered for any of the results we present given our other parameter choices.

The *overlap* parameter provides us with a simple algorithmic guarantee for our processing pipeline. The maximum size for any cluster provided to GAMIT will be:

$$Max\ cluster\ size <= qmax + overlap \tag{2}$$

The *nmax* parameter ensures that the cluster overlap doesn't heavily favor a single neighboring cluster. If $nmax = 1$, then the number of clusters that are expanded over will be equal to what the *overlap* parameter is set to. Using the default $nmax = 2$, the number of clusters that are expanded over will be bounded between *overlap* and *overlap*/2. It is important to note that while we are guaranteed these minimum overlaps, in practice many central clusters will have more redundancy than this. Asymmetry in reciprocity means that some central clusters will have substantial additional redundancy when the subnetworks are reconciled. While redundancy improves the scatter of the GNSS solutions, some clusters are either so small or so central that their membership is entirely redundant, with the stations fully represented across

other clusters. Thus, we implement a final postprocessing step to *prune* redundant clusters and improve run-time.

Cluster pruning

The output from the *overcluster* function is a Boolean *M* by *N* matrix, where the *M* rows correspond to *M* clusters, and the *N* columns correspond to *N* total stations. Each row has columns marked with True values to indicate which stations belong to that cluster after the cluster has been expanded. Depending on the GNSS station distribution geometry and *overcluster* parameters, there will be varying redundancy in the membership of the clusters. We require that each station is represented at least once; while we expect that some stations will be represented multiple times due to overlaps, we proactively remove cases where the full cluster itself is redundant because that entire subnetwork membership is fully present in one or more other subnetwork clusters. To check for this redundancy, we take the sum of each column to verify if that GNSS station is captured one or more times, and then iterate through the matrix by temporarily removing a given row. As we remove a row / cluster, we recalculate the column totals; if removing that row causes us to lose representation of any station in our input dataset, then we pass on to the next row and cluster. However, if removing a cluster leaves all GNSS stations present in at least one other subnetwork, then we delete the cluster and update the *M* by *N* matrix encoding before continuing to recursively check the remaining rows and clusters for if they have any fully redundant coverage. Using our production parameters, a linear scan through the cluster encodings trims approximately 20–30% of overall GNSS stations that are ultimately fed into GAMIT, with a concurrent drop in overall processing time. We expect the impact to processing time is disproportionately impacted by the extreme tail of smaller sized clusters as explained in Sect. 4, therefore, while we maintain the option to execute the *prune* function as a simple linear scan, our current default method will monotonically sort the rows for the Boolean *overcluster* matrix according to the row sum, and then preferentially eliminate redundant rows by removing the smallest redundant clusters first. The successive steps of *qmeans*, *overcluster*, and *prune* are highlighted for a part of a global processing run in Fig. 1.

Computational considerations and software availability

The bulk of the processing time is in the GAMIT runs, which is why we seek an efficient cluster structure—so we can more efficiently process daily GNSS solutions on

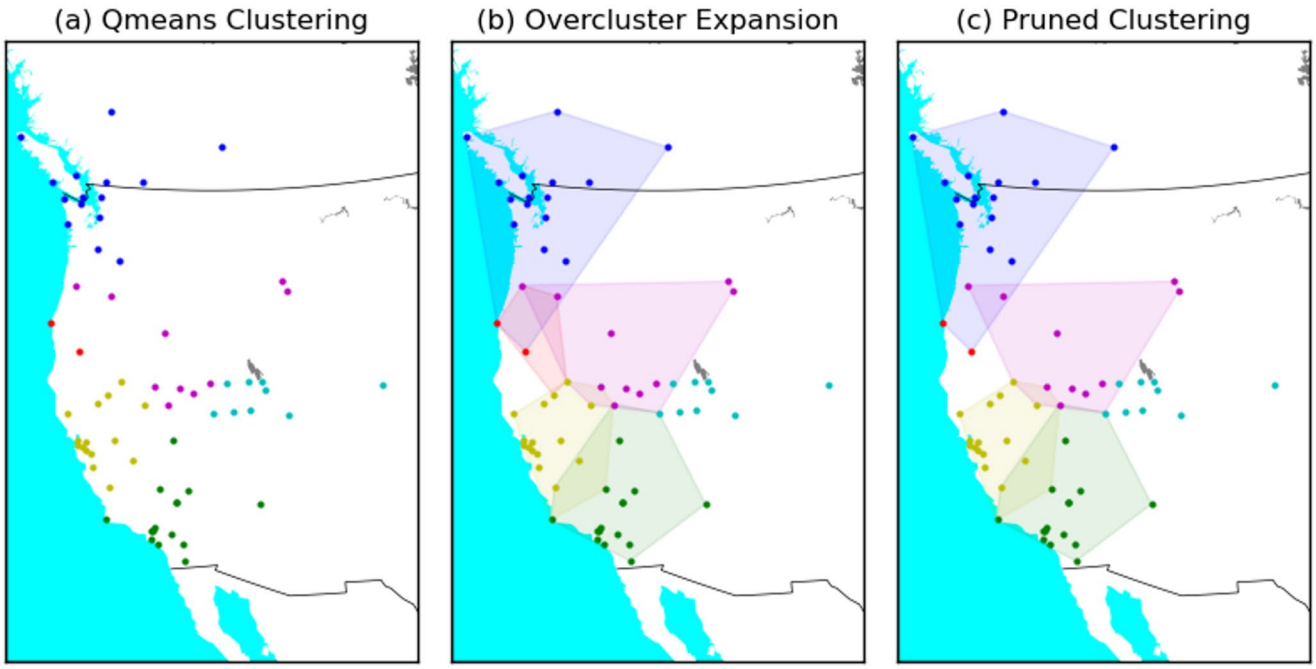

Fig. 1 Successive steps of *qmeans*, *overcluster*, and *prune* **a** Initial *qmeans* clustering using *qmax=16* for a subset of 1008 stations from January 5th, 2022; the degenerate *red* cluster of two stations was split off from the *yellow* cluster, which had 17 stations prior to the split. **b** *Overcluster* using *overlap=4* and *nmax=2*, shown as the hull of original & added stations (cyan cluster hull excluded for readability). The *blue* cluster adds 2 *purple* and 2 *red* stations; *red* cluster adds 2 *purple* and 2 *yellow*; *purple* adds 2 *yellow* and 2 *cyan*; *yellow* adds 2 *purple* and 2 *green*; *green* adds 2 *yellow*, 1 *purple*, and 1 *cyan*. **c** Final post *prune* output, with degenerate *red* cluster removed since it had complete overlap with the *blue* cluster. Note the symmetry of overlap between the *yellow–green* and *purple–yellow* clusters, and the asymmetric overlap for the *blue–purple* and *green–purple* clusters

the time scale of years. The algorithms presented here are computationally efficient; the processing of a full day of ~1000 GNSS stations may take ~20 h in GAMIT, while our clustering processing and postprocessing adds only a few seconds per day, depending on station count. The bisecting *qmeans* algorithm is stochastic, initiating random centroid locations as it searches for an optimum solution. However, to ensure that our results are reproducible, we typically fix the seed in the random number generator so that we have deterministic behavior when comparing runs with different parameter values. The *qmeans* algorithm, as well as the *overcluster* and *prune* functions, are open-source functions integrated into Parallel.GAMIT, a BSD licensed open-source library maintained by the authors to parallelize GAMIT runs. Our library is available through GitHub (https://github.com/demiangomez/Parallel.GAMIT), and is additionally installable from the pypi community repository under the package name *pgamit* when using the python 'pip install' command. While GAMIT (http://www-gpsg.mit.edu/gg/) is available from its authors on request, distribution is limited to non-commercial research use, which also means that it is distributed separately from the *pgamit* package. The clustering functions we develop are freely available, and can be run both within the Parallel.GAMIT parallelization routines, with clustering functions (within the pgamit.cluster module) following the sklearn documentation and API style so that they can be used separately as needed.

Results

To test *qmeans* and evaluate the time-precision trade-off, we processed one year of data (2008) for a GNSS network in North America—composed of ~1200 unique stations in México, the United States, and Canada (Fig. 2a)—and compared the run-times and wrms of four different cluster configurations: *qmax=20* with 4, 6, and 10 overlap stations respectively, and *qmax=40* with 10 overlap stations which is our baseline case for comparisons. This network yields the station-baseline length distribution shown in Fig. 2b, where we show the histogram of all possible baselines for our case study. Since *qmeans* cannot guarantee clusters with a uniform number of stations, Fig. 2c shows the stations per cluster distribution for the test year using *qmax=40* with 10 overlap stations, which results in a membership floor for clusters (at least 11 stations), as well as a ceiling (no more than 50 stations). Figure 2d shows that for *qmax=20* with 10 overlap stations, the total cluster count per day rises as expected, and the clusters are distributed across a tighter range between 11 and 30 stations per cluster. For a graphical

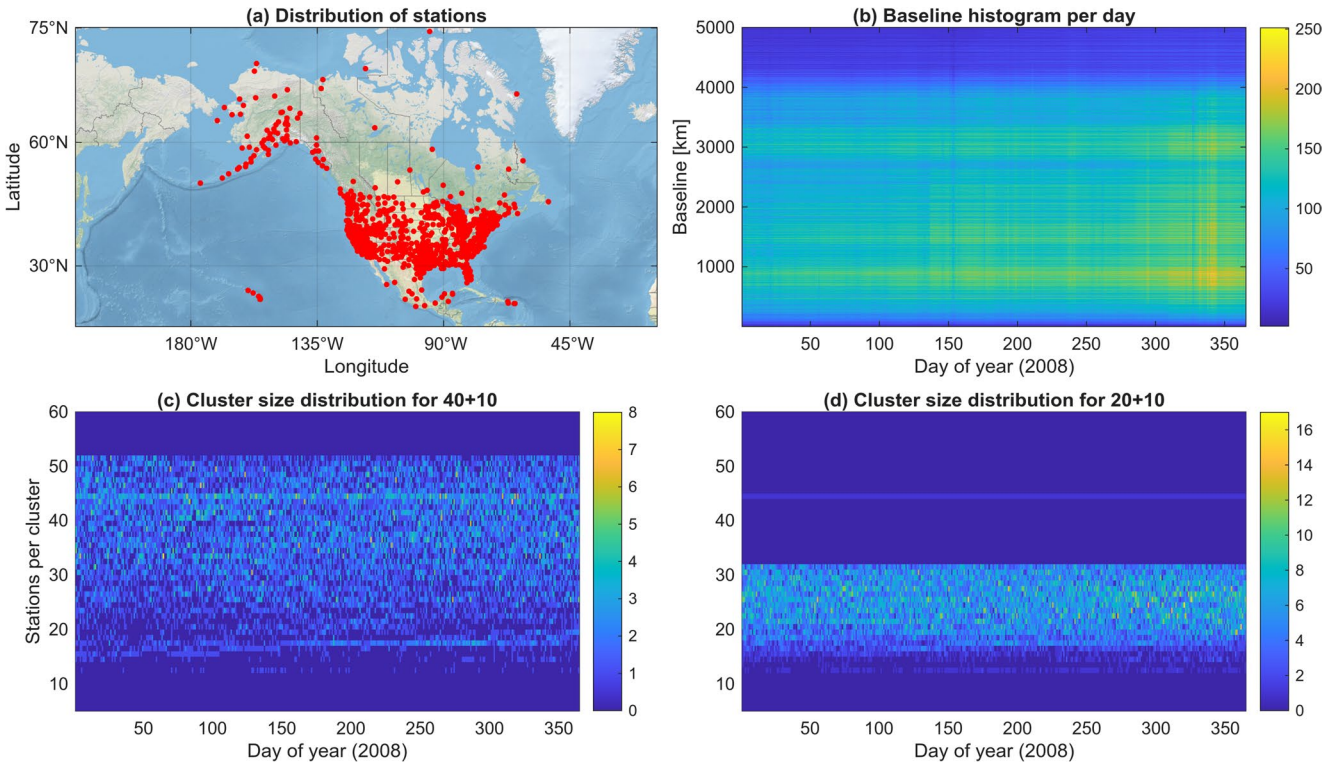

Fig. 2 Cluster size distribution for test year 2008 for two configurations. **a** Station distribution for the network used in this study. **b** Baseline length histogram as a function of time for year 2008. **c** *qmax=40* with 10 overlap stations. **d** *qmax=20* with 10 overlap stations. The line of clusters with 45 stations corresponds to the backbone networks which are always calculated for each day regardless of *qmax* parameter settings

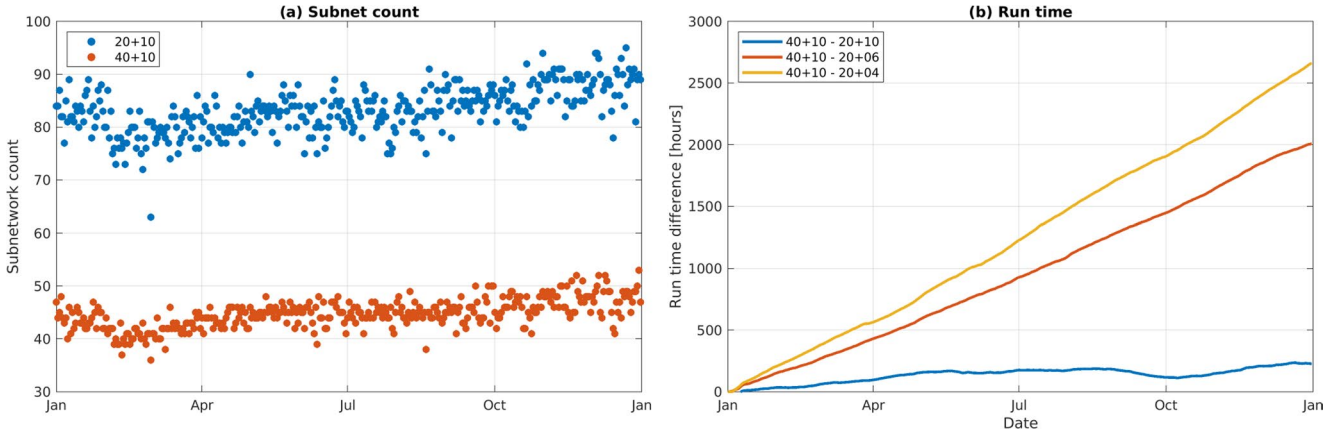

Fig. 3 **a** subnet count for *qmax=20* and *qmax=40* configurations, with 10 overlap stations. **b** Cumulative run-time difference between reference *qmax=40* with 10 overlap stations, and *qmax=20* with 10 (blue line), 6 (red line), and 4 (yellow line) overlap stations respectively

Table 1 Average total stations to process after *overcluster* and *prune*

	Overlap=4	Overlap=6	Overlap=10
qmax=40	1320	1408	1608
qmax=20	1480	1617	2010

representation of the clusters using *qmax=40*, see Fig S1 in the Supporting Information, and Fig S2 for cluster size and overlap station count histograms for a single day.

Clustering performance of the qmeans algorithm

In terms of the number of clusters produced for a Parallel. GAMIT session, Fig. 3a shows that with 10 overlap stations, *qmax=20* results in nearly double the number of clusters compared to the *qmax=40* configuration. It is important to note that the number of clusters produced by *qmeans* is invariant with respect to the *overlap* parameter specified, as the partitioning in the *qmeans* process is independent of the *overcluster* function, which does not add or remove clusters. Although the *prune* function is impacted by structure and number of overlap stations when determining redundant clusters to remove, the output number of clusters is primarily determined by the *qmax* parameter of *qmeans*. Figure 3b shows the observed cumulative time difference between our reference case of *qmax=40* with 10 overlap stations, and *qmax=20* with 10, 6, and 4 overlap stations respectively. For the case where only *qmax* differed and overlap stations were held at 10, our results showed that *qmax=40* took ~250 more hours to run relative to *qmax=20*. Based on the total processing time for *qmax=40* (~8500 h), 250 h represents only a 3% reduction of run-time. A more significant run-time reduction was observed for *qmax=20* when varying overlap stations to 6 and 4, where the total run-time difference from our *qmax=40* reference reached 2000 and ~2600 h (23% and 30% reduction), respectively.

Table 1 provides a summary of the mean number of stations per cluster configuration for an average network size of ~1200 stations, and highlights the fundamental tradeoff between cluster size, total number of stations to process, and run-time impact. Using 10 overlap stations, the total number of input stations for GAMIT to process when *qmax=40* is 2010 versus 1608 for *qmax=20*; however, the total cumulative run-time is 3% less for *qmax=20*, despite processing 402 additional stations, which is 25% more stations than the reference *qmax=40*. This is expected given the polynomial behavior of run-time as a function of cluster size (discussed below in Sect. 3.4), but optimization is complicated because of how overall station count varies as a function of both number of clusters (itself a function of *qmax*) and number of overlap stations. As Table 1 shows, since smaller clusters are more numerous, increasing the number of overlap stations has an outsized effect on overall number of cumulative stations to process– going from 10 overlap stations to 6 overlap stations reduces cumulative station count by only 200 for *qmax=40*, but by over 400 for *qmax=20* since there are approximately double the clusters produced to add overlap stations to. Given that *qmax=40* with 10 overlap stations and *qmax=20* with 6 overlap stations have approximately the same cumulative number of stations to process (1608 versus 1617), the red line from Fig. 3b presents an interesting comparison on run-time, a difference driven entirely by the size distribution of the clusters.

3.2 Qmeans under a sparse network

To show the performance of *qmeans* with a sparse network, we applied our clustering algorithm to part of the Argentine GNSS network (*Red Argentina de Monitoreo Satelital Continuo*, RAMSAC, Piñón et al. 2018) for year 2015. Figure 4a shows the station distribution for the last day of 2015, while Fig. 4b shows that the network is much smaller in

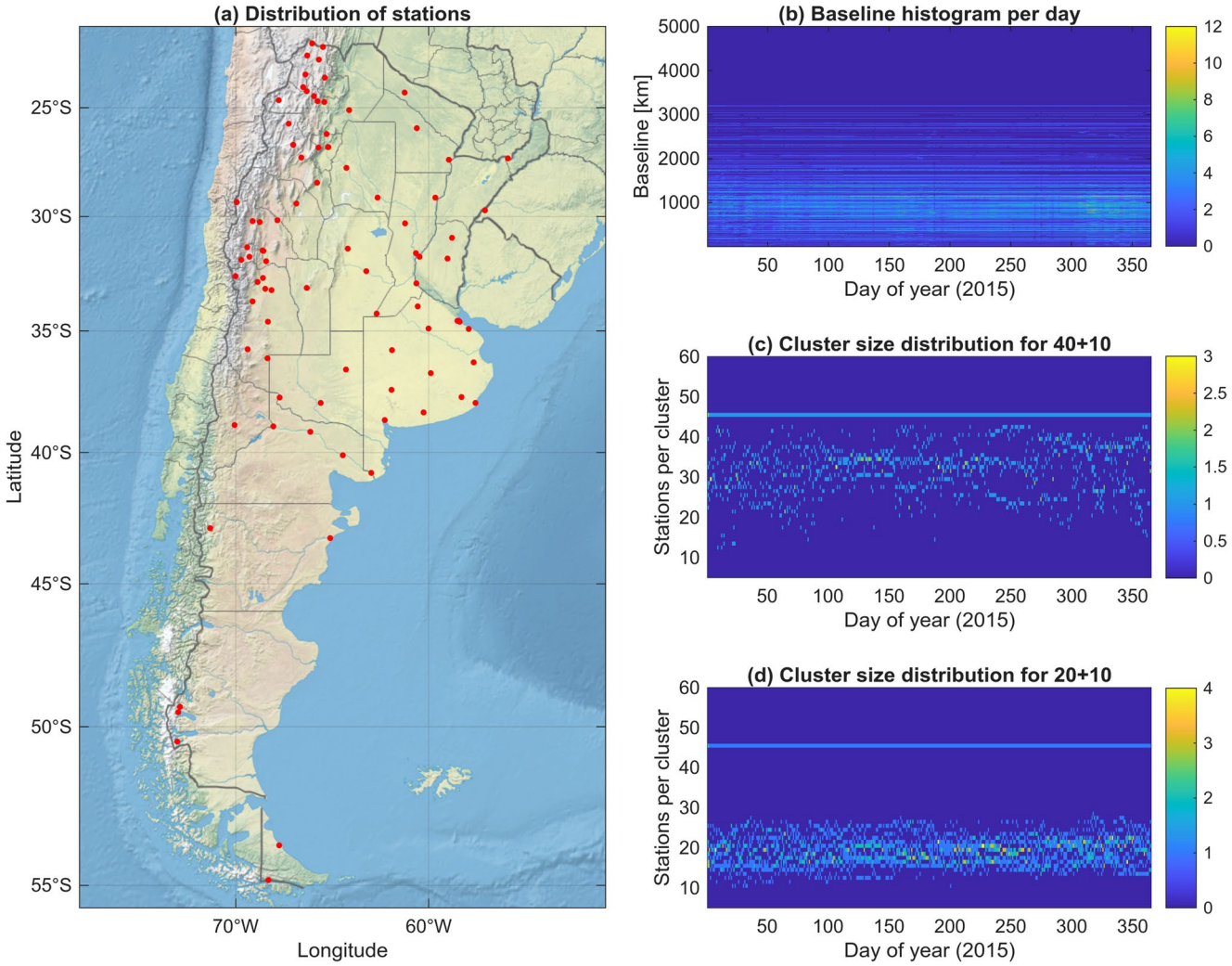

Fig. 4 Cluster size distribution for test year 2015 for part of the Argentine GNSS network for two cluster configurations. **a** Station distribution for the sparse network used in this study. **b** Baseline length histogram as a function of time for year 2015. **c** *qmax=40* with 10 overlap stations. **d** *qmax=20* with 10 overlap stations. The line of clusters with 45 stations corresponds to the backbone networks which are always calculated for each day regardless of *qmax* parameter settings

aperture and in station count (compared to its North American counterpart in Fig. 2). This sparse network is composed, on average, by ~80 stations (enough to at least require two clusters when some stations go offline for some time). Similarly to the case study in Fig. 2c, d. Figure 4c, d show the effect of the stations per cluster parameter on the ceiling of the histogram.

Setting *qmax*=40 and *overlap*=10 yields between 3 and 4 clusters per day (with a total station count to process of ~90–100), while using *qmax*=20 yields between 7 and 9 clusters (with a total station count of ~110–140, see SI for the corresponding network diagram configurations). Thus, as for the dense network case, the number of clusters almost doubles. In terms of processing time, we note that the difference in total run time is not as significant as for dense network configuration, and the difference between *qmax*=40, *overlap*=10 and *qmax*=20 becomes *negative*, that is, *qmax*=40 required ~450 h to complete while *qmax*=20 required a cumulative time of ~500 h. Given the order of magnitude difference in percentage of overall network that is added for overlap (>10% for ~80 station network vs. <1% for the ~1200 station network), we expect that increasing dense network sizes will scale *positive* differences to higher orders of magnitude. The difference for this sparse network is reduced and becomes almost zero when reducing the overlap station count to 4, thus it takes about the same time to run *qmax*=40, *overlap*=10 than *qmax*=20, *overlap*=4; this is expected, since the total number of stations is reduced when the overlap parameter is smaller. Therefore, for sparse networks or small numbers of stations, there is not significant time saved when generating clusters using different *qmax* parameters. Yet, the impact of

overlap stations in solution quality (discussed in the next section) still applies.

Solution quality impact

Cluster size and number of overlap stations is expected to imply a trade-off both between execution time, and solution quality, at least when using large numbers of sites as in our North America case study. To assess this trade-off in solution quality, we analyzed the daily GNSS repeatability using *qmax=40* with 10 overlap stations as a reference solution and compared it against solutions using *qmax=20* with station overlaps of 4, 6, and 10 for the same test year and the same network. Daily repeatabilities are obtained by transforming consecutive GNSS solutions onto one another using six-parameter Helmert transformations (Bevis and Brown 2014). Following the reweighing procedure of Sobrero et al. (2024), during the daily solution alignment we downweight any stations with large residuals to ensure that these outliers will not affect our parameter estimation and statistics. After all solutions have been aligned, the differences between individual station coordinates are computed and a wrms value in North, East, and Up provides an estimate of the scatter level of the solution for each individual site. The daily wrms for each component is computed as.

$$wrms = \hat{\sigma}_s \sqrt{\frac{\mathbf{v^T \cdot P \cdot v}}{n-r}}$$

where $\hat{\sigma}_s$ is the nominal uncertainty of the solution dataset (obtained iteratively during the reweighing), $\mathbf{v}$ is the vector of residuals (in North, East, or Up), $\mathbf{P}$ is the weight matrix after downweighting outliers accordingly, and $n - r$ is the degrees of freedom of the system.

Figure 5a–c show the cumulative empirical distribution function for the daily repeatability wrms scatter for the North, East, and Up components for all stations in the analysis. A clear pattern emerges from these plots, which is most pronounced in the East component, where we note that the base reference case (*qmax=40* with 10 overlap stations) outperforms the other solutions using *qmax=20* across all overlap parameter values, with *overlap=4* producing the worst results, while solutions for *overlap=6* and *overlap=10* closely follow each other. Interestingly, the vertical component in Fig. 5c shows that solutions with *qmax=40* or *20* and *overlap=10* closely follow each other. Although the smallest subnetwork clusters with the least overlap are consistently outperformed by other configurations, as shown by the 0.1 and 0.3 mm shaded regions of scatter for the horizontal and vertical components in Fig. 5a–c, all tested cluster configurations fall within these tight regions and therefore can be considered statistically equal in terms of expected GNSS solution noise. Despite these results, the level of agreement between the wrms of these experiments cannot be always guaranteed, given that other effects such as network geometry or latitude location can affect the overall solution quality. Hence, whenever minimum scatter is desired, cluster size and number of overlap stations should be increased to achieve needed precision.

Processing time as function of station count from historical data

As part of our parallel processing system for GAMIT, we save all of the relevant statistics for each run to a Postgres database, including execution time, overlap station count, and solution quality indicators. Since the deployment of Parallel.GAMIT at OSU in 2019, we have performed a total of ~2.5 M individual GAMIT runs for various projects. Using this Postgres database, we selected a subset of GAMIT runs for execution time statistics over an average GNSS project of ~1000 stations. We did not use the entire 2.5 M runs in order to avoid double counting the same network distributions and duplicate runs, since many of the

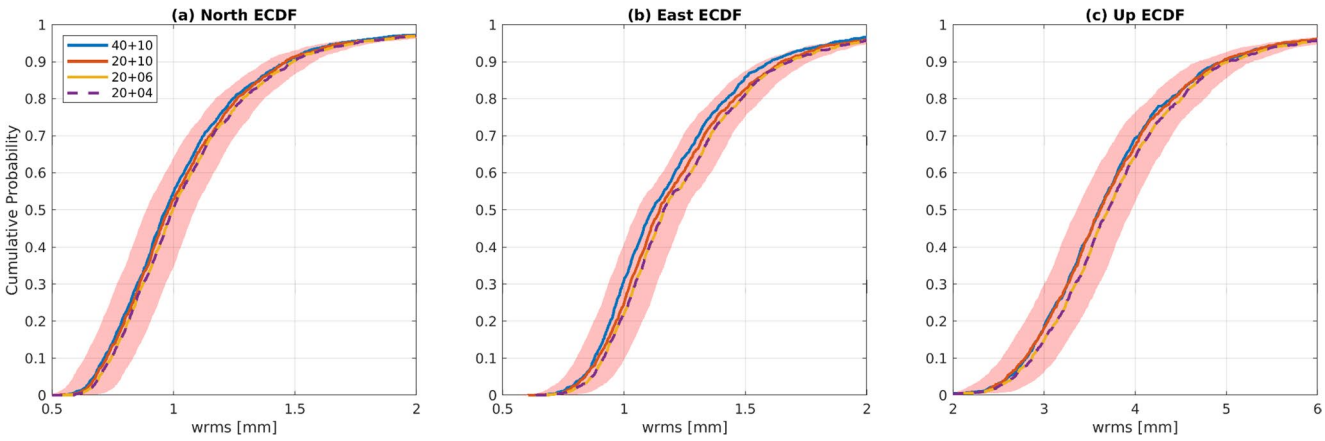

Fig. 5 Empirical cumulative distribution function for repeatability analysis of four cluster configurations **a** Weighted root mean square scatter for the North component. Red shaded area represents the ±0.1 mm region. **b** Same as **a** for the East component. **c** Same as **b** for the Up component. Red shaded area represents the ±0.3 mm region

sessions are densifications or reruns using different models and orbits. After removing any 'outlier' runs (those that took more than 80 min to complete) that could alter our statistics, the dataset used in this work is composed of ~360 k GAMIT executions. The removed long execution times account for less than 0.5% of the dataset, and were mostly related to external events not associated with the GNSS processing, such as computer network outages and other technological difficulties encountered during the processing.

Figure 6a shows the histogram of execution times for all the selected GNSS sessions from 1994 to 2022. For simplicity and consistency of the results, we performed all statistics and analysis using GPS-only solutions (rather than multi-constellation). The station clusters for these runs were created using an ad-hoc, non-optimized script (deployed from 2019 to 2024, prior to developing *qmeans*) which creates station clusters by proximity and splits the resulting subnetworks if the number of stations in the processing session is above a threshold of ~60. Once each cluster is determined, the clusters contain sets of unique stations to process and overlap stations from neighboring clusters, which are used to merge the solutions with GLOBK upon finishing all the processing jobs. Additionally, the clusters are also tied together using a 'backbone' network of evenly distributed sites across the entire area of interest to provide a 'rigid support' for the chainmail-like network of smaller clusters (see Appendix A for a description of the backbone network algorithm) and guarantee a robust combination.

Figure 6b shows the cluster size histogram where we note two peaks, one around 15 and a second one near 40 stations. This distribution is produced due to the impossibility of splitting the stations into clusters with an exact number of stations. The data for execution time as a function of station count (Fig. 6c) can be fitted using a quadratic expression to obtain an estimate of the execution time; to account for other factors affecting the execution time at lower station counts, such as overhead time to read observation and metadata files, models, and others, we fit a second-degree polynomial rather than a pure quadratic. We performed this fit and found that the execution time t in minutes can be expressed as:

$$t = 6.04 \ - 0.10 \cdot s \ + 0.021 \cdot s^2 \tag{3}$$

where s is the number of stations in the processing session. Given the large number of data points used in the fit, the weighted root mean square error of our polynomial fit (95% confidence interval) was ~2 min.

Discussion

Although determining the exact reason behind the solution quality improvement with increasing cluster size and overlap station count is outside the scope of this work, we surmise that clusters with more stations have, on average, a larger spatial aperture, and also larger number of double differences. Larger aperture and more double differences will improve the determination of the zenith tropospheric delay parameters which, in turn, improves both horizontal and vertical component scatter. Additionally, increasing the overlap station count increases the redundancy of the network, which helps mitigate the solution scatter when combining all clusters together.

Execution time model

Due to memory limits and also because execution time increases quadratically, GNSS sessions with $N > 60$–80 stations need first to be partitioned into C clusters of s stations each. As seen in Sect. 3, our GAMIT runs occur on datasets where the cluster sizes are not uniform; the compute runtime for these production sessions can be approximated by modifying Eq. 3:

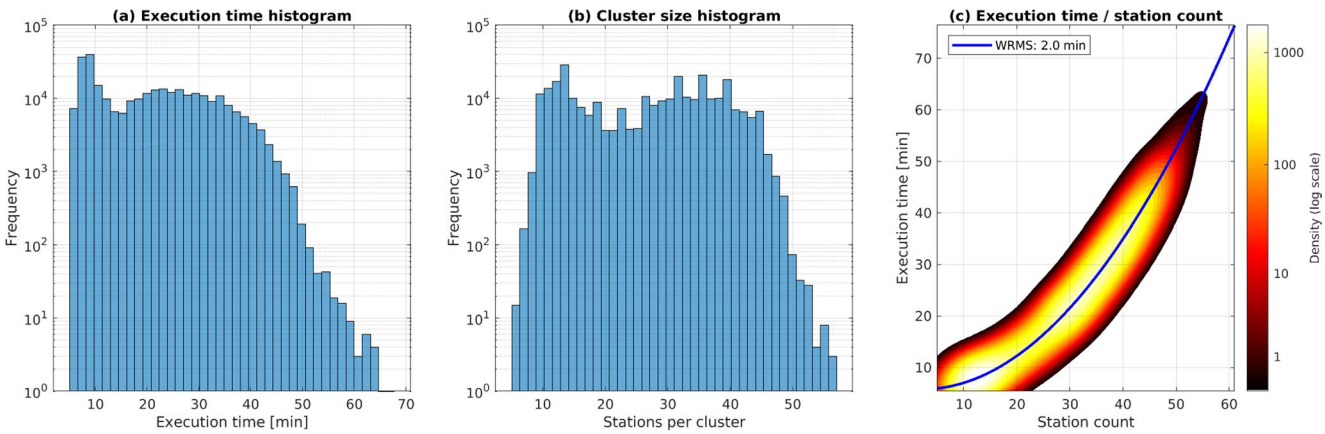

Fig. 6 Empirical GAMIT execution time **a** Histogram of execution times used to obtain the polynomial quadratic parameters. **b** Same as **a** but for the cluster sizes. **c** Density of data points and fitted polynomial quadratic curve (3), execution time as a function of station count

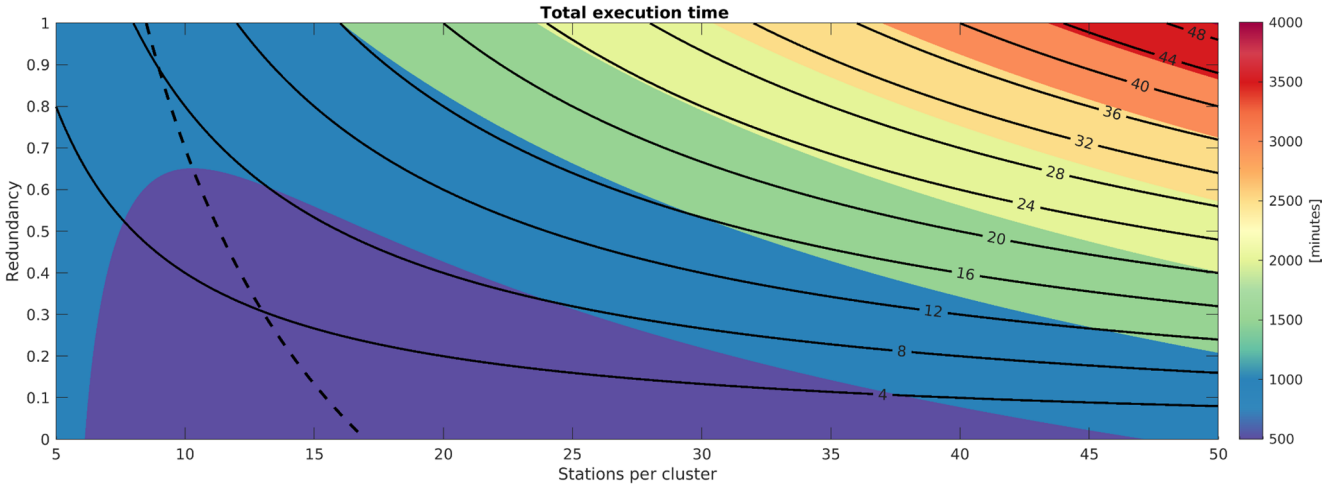

Fig. 7 Color contour plot with 500-minute intervals of total execution time for a network of 1000 stations. Black dashed line shows the minimum run-time for each redundancy value as a function of stations per cluster. Black contours show the overlap station count per cluster, with redundancy values on the y-axis

$$T = \sum_{i=1}^{C} \left(6.04 \ - 0.10 \cdot (s_i + e) \ + 0.021 \cdot (s_i \ + e)^2\right) \tag{4}$$

Where e is the *overlap* parameter value (constant for all clusters), C is the actual number of clusters formed, and s_i is the cluster size of cluster at index i. Equation (4) does not account for geometry considerations within GAMIT or other background processes that impact run-time; still, we find that the time estimate T has only slight bias in predicting overall run-time when compared to metadata within our postgres database which records the empirical run-time.

Equation 4 is of little practical value by itself given that we already log the actual run-time of the GAMIT processing runs, and don't need predictive estimation of individual subnetwork run-time for our operational processing. More useful is generalizing Eq. 4 to visualize the behavior of the subnetworks and help illustrate parameter choices that can guide us in optimizing our compute pipeline. The ratio between parameters e and s, $R = \frac{e}{s}$, can be defined as the 'redundancy' of a given Parallel.GAMIT session, where $R = 0$ indicates that there are no shared stations between clusters (this is an undesired condition) and $R = 1$ indicates that all stations are processed separately in two or more clusters. To idealize the compute run-time of a given GNSS network, the total number of clusters to be processed is $C = \frac{N}{s}$, assuming N total session stations are divisible by s, where stations per cluster s is now fixed at a single value for all clusters rather than varying per cluster. Assuming this idealized uniform distribution of clusters, the total execution time T, including the redundancy, can be estimated as:

$$T = \sum_{i=1}^{C} t_i = C \cdot \left(6.04 \ - 0.10 \cdot s \cdot (1 + R) \ + 0.021 \cdot [s \cdot (1 + R)]^2\right) \tag{5}$$

Equation (5) uses R instead of e, as redundancy is defined as the ratio between e and s, which simplifies the interpretation of contour plot in Fig. 7.

Figure 7 shows a contour plot of T (for N=1000) for Eq. (5) as a function of s and R, also including the contours of equal overlap stations, which vary with s for a constant R. For clarity, we plotted a continuous contour field regardless of the value of $C = \frac{N}{s}$, although the idealized execution time model is only valid when C is an integer. We also show, as a black dashed contour, the minimum run-time for each value of R and its corresponding cluster size. It should be noted that this line marks the limit of the cluster mediated computation efficiency: cluster sizes to the left of this dashed line will increase the run-time, decreasing efficiency rather than improving it. In other words, the execution time needed to complete a multi-GAMIT run session decreases concurrently with cluster size (i.e., as the number of clusters formed per session goes up) until the limit denoted by the intersection of the black dashed contour. Thus, if for example one uses 4 overlap stations, a session divided into uniform clusters of, say, 20 stations, takes ~400 min less to finish than if it were divided into uniform clusters of 50 stations, although the number of clusters is larger for the former case. This tendency of decreasing time with lower cluster size, however, is less significant with increasing number of overlap stations. Figure 7 shows that for increasing overlap stations, the overlap contours become increasingly parallel to the execution time contours, meaning that moving along overlap contours (and changing the cluster size) does not significantly change the execution time. We find that Fig. 7 is mostly useful to examine the execution time behavior purely as a function of the overlap parameter, and help select an ideal cluster size to target.

As we mentioned before, run-time is not the only consideration that a user has to account for when choosing a given partitioning. From Fig. 7 it is clear that the choice of cluster configuration can have a considerable impact in the time needed to compute the solutions. This time difference will translate into a more or less significant 'wall time', i.e., the actual time needed to compute the solutions, depending on the number of compute nodes used to process the data. For instance, to process the totality of the GNSS stations in México, the United States, and Canada from 1994 to 2025 using clusters of size 20 with 6 overlap station clusters requires a total computation time of 8223 days, or ~33 days at 250 cores. If the same project is partitioned into clusters of size 50 with the same 6 station overlap, then the required run-time is 9600 days, or ~38 days at 250 cores, a wall time difference of 5 days. The wall time difference becomes more significant with a smaller compute cluster, and in the case of using only 100 cores, the wall-time difference between these two example runs is close to 14 days.

Conclusions

This study presents a novel approach for efficiently partitioning large GNSS networks into subnetworks or clusters using a modified bisecting k-means algorithm, which we called *qmeans*. The primary goal of this work was to develop a method for reducing the computational burden of processing large GNSS datasets while maintaining the precision of the results. Through extensive testing, we demonstrated that *qmeans* provides a robust and scalable solution for dividing GNSS stations into clusters of manageable sizes, facilitating faster double-difference processing in GAMIT/GLOBK.

The *qmeans* algorithm itself offers several advantages. First, the total clustering processing time was found to add only a few seconds per day, a minimal overhead relative to the time savings achieved by efficiently dividing the network. Second, by eliminating the need to predefine the number of clusters (as in traditional k-means clustering), *qmeans* provides a flexible and dynamic solution that can adapt to varying network sizes and spatial configurations. This feature is particularly beneficial when processing GNSS networks where station distributions are irregular, such as in sparsely populated regions or oceanic islands. Moreover, the hierarchical nature of *qmeans*, combined with the post-processing steps (*overcluster* and *prune*), ensures that the final clusters are both computationally efficient and spatially coherent, with sufficient overlap for robust reference frame realization. As expected, the increase in the station count caused by adding additional overlap stations leads to a decrease in computational efficiency, as some stations are processed more than once. However, we have also demonstrated that this reduction in efficiency results in a corresponding increase in precision, which is a desired outcome when processing GNSS data.

Our results show that partitioning the network into clusters of approximately 20–40 stations provides an optimal balance between reducing computational time and preserving solution accuracy. Smaller clusters (20 stations) lead to faster processing times, with slightly increased solution scatter, while larger clusters (40 stations) offer a modest reduction in scatter at the cost of increased processing time. In our experiments, the *qmax=20* with *overlap=6* configuration was shown to cut processing time by nearly 23% compared to the *qmax=40* with *overlap=10* configuration despite processing a similar number of total stations within the session. Similarly, using *qmax=20* with *overlap=4* reduced the processing time by 30%, despite generating twice the clusters– both cases with a trade-off of a slight increase in solution scatter. These findings suggest that for large-scale GNSS projects, careful consideration of cluster size and overlap station count is critical for achieving efficient processing without compromising the quality of the result. By contrast, if one is seeking to obtain a fast solution for testing purposes, reducing the number of stations and overlap stations per cluster significantly reduces the processing time for GNSS solutions.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s10291-025-02020-6.

Acknowledgements We acknowledge Eric Kendrick for his role in maintaining the postgres database of GAMIT run statistics, which was queried by the authors to develop the predictive runtime model.

Author contributions S.G. and D.G. contributed equally to the manuscript. S.G. designed and developed the qmeans, overcluster, and prune functions described in the manuscript, and led integration for adding them to the ParallelGAMIT library. D.G. envisioned experimental design, interpreted the results, and developed the run time predictive model. S.G. wrote the initial draft of the Methods and Discussion sections; D.G. wrote the initial draft of the Introduction and Results sections; both S.G. and D.G. reviewed, edited, and revised the entire manuscript. D.G. added supplementary material, and produced Figs. 2, 3, 4, 5 and 6. S.G. produced Fig. 1, and authored and tested the pgamit.cluster module used to create all figures except Fig. 5. S.G. and D.G. maintain the open source project Parallel.GAMIT (pgamit), and D.G. is the project founder of that project.

Funding The research leading to these results received funding from the National Geodetic Survey (NGS), under Grant Agreement AWD-115866.

Data availability Data is public and available through various websites, including the International GNSS Service data repository, through the EarthScope Facility Archive, and the NOAA Continuously Operating Reference Station (CORS) Network (NCN), managed by NOAA/National Geodetic Survey.

Declarations

Conflict of interest All authors certify that they have no affiliations with or involvement in any organization or entity with any financial interest or non-financial interest in the subject matter or materials discussed in this manuscript. The authors have no financial or proprietary interests in any material discussed in this article.

References

Altamimi Z, Rebischung P, Collilieux X et al (2023) ITRF2020: an augmented reference frame refining the modeling of nonlinear station motions. J Geod 97:47. https://doi.org/10.1007/s00190-023-01738-w

Alves Costa SM, Sánchez L, Piñón D et al (2022) Status of the SIRGAS reference frame: recent developments and new challenges. In: IAG International symposium on reference frames for applications in geosciences. Springer Nature Switzerland, Cham. pp 153–165

Ankerst M, Breunig MM, Kriegel H-P, Sander J (1999) OPTICS: ordering points to identify the clustering structure. ACM SIGMOD Rec 28:49–60. https://doi.org/10.1145/304181.304187

Arthur D, Vassilvitskii S (2006) k-means++: The advantages of careful seeding. Stanford

Bevis M, Brown A (2014) Trajectory models and reference frames for crustal motion geodesy. J Geod 88:283–311. https://doi.org/10.1007/s00190-013-0685-5

Cui Y, Chen Z, Li L, Zhang Q, Luo S, Lu Z (2021) An efficient parallel computing strategy for the processing of large GNSS network datasets. GPS Solutions 25(2):36. https://doi.org/10.1007/s10291-020-01069-9

Ester M, Kriegel H-P, Sander J, Xu X (1996) A density-based algorithm for discovering clusters in large spatial databases with noise. In: KDD. pp 226–231

Forgy EW (1965) Cluster analysis of multivariate data: efficiency versus interpretability of classifications. Biometrics 21:768–769

Geng J, Mao S (2021) Massive GNSS network analysis without baselines: undifferenced ambiguity resolution. J Geophys Res: Solid Earth 126(10):e2020JB021558

Gómez DD, Bevis MG, Caccamise DJ (2022) Maximizing the consistency between regional and global reference frames utilizing inheritance of seasonal displacement parameters. J Geod, 96(2)

Gómez DD, Bevis MG, Caccamise DJ et al (2024) An empirical tool for predicting the presence or absence of coseismic displacements at GNSS stations. GPS Solut 28:214. https://doi.org/10.1007/s10291-024-01758-9

Herring TA, King RW, Floyd MA, McClusky SC (2018) Introduction to GAMIT/GLOBK

International Organization for Standardization (2020) Geographic information - Geodetic references—Part 1: International terrestrial reference system (ITRS) (ISO Standard No. 19161-1:2020(E))

Lloyd S (1982) Least squares quantization in PCM. IEEE Trans Inf Theory 28:129–137

Ng A, Jordan M, Weiss Y (2001) On spectral clustering: analysis and an algorithm. Adv Neural Inf Process Syst 14

Omohundro SM (1989) Five Balltree construction algorithms. International Computer Science Institute

Pedregosa F, Varoquaux G, Gramfort A et al (2011) Scikit-learn: machine learning in python. J Mach Learn Res 12:2825–2830

Piñón DA, Gómez DD, Smalley R Jr, Cimbaro SR, Lauría EA, Bevis MG (2018) The history, state, and future of the Argentine continuous satellite monitoring network and its contributions to geodesy in Latin America. Seismol Res Lett 89(2A):475–482

Schubert E, Sander J, Ester M et al (2017) DBSCAN revisited, revisited: why and how you should (Still) use DBSCAN. ACM Trans Database Syst 42:1–21. https://doi.org/10.1145/3068335

Sobrero FS, Ahlgren K, Bevis MG, Gómez DD, Heck J, Echalar A, Caccamise DJ, Kendrick E, Montenegro P, Batistti A, Contreras Choque L, Catari JC, Sallico T, R., Guerra Trigo H (2024) A robust approach to terrestrial relative gravity measurements and adjustment of gravity networks. J Geod 98(9):86. https://doi.org/10.1007/s00190-024-01891-w

Steinbach M, Karypis G, Kumar V (2000) A comparison of document clustering techniques Michael Steinbach. George Karypis, and Vipin Kumar