Parallel Processing of Remote Sensing Time Series Applied to Land-Use and Land-Cover Classification

Roberto U. Paiva, Sávio S. T. Oliveira, Luiz M. L. Pascoal, Leandro L. Parente, Wellington S. Martins

Universidade Federal de Goiás, Brazil
{urzedabr,wsmartins}@ufg.br
{savioteles,luizmlpascoal,leal.parente}@gmail.com

Abstract. The increase in satellite launches into Earth's orbit in recent years has generated a huge amount of remote sensing data. These data, in the form of time series, have been used in automated classification approaches, generating land-use and land-cover (LULC) products for different landscapes around the world. Dynamic Time Warping (DTW) is a well-known computational method used to measure the similarity between time series. It has been used in many algorithms for remote sensing time series analysis. These DTW-based algorithms are capable of generating similarity measures between time series and patterns. These measures can be used as meta-features to increase the accuracy results of classification models. However, DTW-based algorithms require a lot of computational resources and have a high execution time, which makes them difficult to use in large volumes of data. This article presents a parallel and fully scalable solution to optimize the construction of meta-features through remote sensing time series (RSTS). In addition, results of the application of the generated meta-features in the training and evaluation of classification models using Random Forest are presented. The results show that the proposed approaches have led to improvements in execution time and accuracy when compared to traditional strategies.

Categories and Subject Descriptors: I. [**Computing Methodologies**]: Miscellaneous; D.1 [**Programming Techniques**]: Miscellaneous; H.2 [**Applied computing**]: Miscellaneous

Keywords: Classification, Land-Use and Land-Cover, Parallel Processing, Remote Sensing, Time Series

1. INTRODUCTION

The Land-Use and Land-Cover (LULC) present dynamics of changes that can be monitored through the analysis of Remote Sensing Time Series (RSTS) [Foody 2002]. Due to a large number of satellites orbiting the planet and the technological advancement of space sensors, monitoring initiatives are producing large volumes of Earth observation data. These data are used to generate products for mapping LULC, which assist in decisions related to food security, environmental conservation, sustainability, greenhouse gas emissions, and combating deforestation [Nepstad et al. 2014; Bonan 2008; Bala et al. 2007; Dewan and Yamaguchi 2009].

Dynamic Time Warping (DTW) is a classic method of computer science, introduced in the 1970s that makes use of dynamic programming to measure the similarity between time series [Sakoe and Chiba 1978; Rabiner et al. 1978]. These similarity measures built from time series can be used as meta-features. Some algorithms based on DTW were developed to use similarity measures between time series in the mapping of LULC changes [Guan et al. 2016; Romani et al. 2010; Maus 2016]. Among these algorithms, Time-Weighted Dynamic Time Warping (TWDTW) [Maus et al. 2016] stands out for creating the Time-Weighted function, which makes the method sensitive to seasonal climate changes in natural and cultivated vegetation. The TWDTW is used in conjunction with the k-Nearest Neighborhood (k-NN) algorithm in LULC classifications based on the similarities of

RSTS [Dadi 2019; Oliveira et al. 2019; Manabe et al. 2018].

Although TWDTW is a DTW-based algorithm that is effective in analyzing time series, it faces problems that affect its performance, such as the high cost of computational resources and high execution time, factors that increase according to the size of the data entry. This is due to the quadratic complexity of the DTW method, that makes it difficult to use this solution with large volumes of data, RSTS products with large temporal and spatial resolutions, or even in large areas. In addition, in some regions, the integration between TWDTW and k-NN has low accuracy in the classification of LULC [Dadi 2019].

Recent works have explored the parallel processing of RSTS, looking for more efficient ways to generate meta-features based on similarity. Some works also seek to carry out the classification of LULC through meta-features in conjunction with more sophisticated machine learning algorithms, to obtain improvements in the accuracy of the classification [Oliveira et al. 2018; Oliveira et al. 2019]. However, these solutions have deficiencies with processing idleness and scalability limitations. Considering technological advances for new space sensors and the emergence of satellites with high temporal and spatial resolutions (e.g., PlanetScope[1]), these limitations tend to hamper the use of these solutions in a near future.

This article presents the Rapid-DTW algorithm, a new parallel solution to the problem of processing large volumes of RSTS data. The Rapid-DTW is a DTW-based solution designed specifically to work with remote sensing data, using the Time-Weighted function. The proposal was developed for processing in GPUs, using the CUDA architecture, due to the massive parallelism capacity of these types of processors, and the efficient energy consumption with low investment costs. The Rapid-DTW was able to significantly reduce processing idleness and eliminate scalability problems when compared to previous solutions. The experiments carried out showed that Rapid-DTW obtained better results in terms of performance than the bests current solutions.

The work also presents the application of the generated meta-features, for samples from the region of Mato Grosso - Brazil, in models of classification of LULC using the Random Forest algorithm. The meta-features were used in some scenarios, in order to assess the impact of their use on the accuracy of these models. The results of this experiment showed that in all the proposed scenarios, the use of meta-features based on similarity of RSTS was able to improve the accuracy results in the classification of LULC.

This article is an extended version of [Paiva et al. 2020], presented in XXI Brazilian Symposium on GeoInformatics (GEOINFO 2020). This new version better describes the proposed Rapid-DTW algorithm and brings the following new contributions: a thread utilization analysis for the proposed parallel algorithm, a methodology for evaluating the use of meta-features in LULC classification models, a feature importance study, and experimentation with three scenarios and different feature spaces. The results show that the meta-features can be incorporated in models already consolidated, with a positive impact on the accuracy of classification, mainly in complex classes. The article is organized as follows. Section 2 presents a brief presentation of related works. Section 3 presents the Rapid-DTW algorithm. Section 4 presents the results of the tests and experiments carried out to evaluate the performance of Rapid-DTW and the application of meta-features in LULC classification models. Finally, in section 5 we present the conclusions and future work.

2. RELATED WORKS

This section presents the works that served as a basis for the development of this research. Section 2.1 presents the TWDTW algorithm, highlighting the development of the Time-Weighted function. Section 2.2 presents the parallel solution SP-TWDTW [Oliveira et al. 2018; Oliveira et al. 2019], with

[1]`https://www.planet.com/`

a critical analysis addressing positive points and problems. The section 2.3 introduces some at the works related to the use of RSTS and meta-features for the classification of LULC.

2.1 Time-Weighted Dynamic Time Warping (TWDTW)

DTW-based algorithms can measure the similarity between a time series and a pattern (even if they have different sizes or are displaced in time), calculating meta-features that indicate how far a time series is from a pattern.A pattern represents the expected behavior of a time series over a given period. The TWDTW is a DTW-based algorithm designed to work with remote sensing time series. The TWDTW presents a logistical function capable of dealing with different seasonality in data, for example, the seasonality during a crop cultivation. This function, called Time-Weighted, is responsible for creating a penalty score in similarities between time series and recognized patterns that are displaced in time.

The TWDTW algorithm computes a cost matrix $\Psi_{n,m}$ given the pattern $U = (u_1, ..., u_n)$ and time series $V = (v_1, ..., v_m)$. The elements $\psi_{i,j}$ of $\Psi_{n,m}$ are computed by adding the temporal cost ω, becoming $\psi_{i,j} = |u_i - v_j| + \omega_{i,j}$. To calculate the temporal cost ω, the Time-Weighted function is used, presented in the Equation 1. The logistic weight has midpoint $\beta = 100$ days and steepness $\alpha = 0.1$.

$$\omega_{i,j} = \frac{1}{1 + e^{-\alpha(g(t_x,t_y)-\beta)}}, \tag{1}$$

The Time-Weighted technique is a a logistic function based on the difference in days $g(t_x, t_y)$ between the date of the pattern and time series observations. From the result of the weight matrix Ψ, the algorithm calculates an accumulated cost matrix D (Equation 2), using a recursive sum of the minimum dynamics. Then the algorithm uses the D matrix to find the path with the lowest cost, thus generating the measure of similarity between the pattern and the time series.

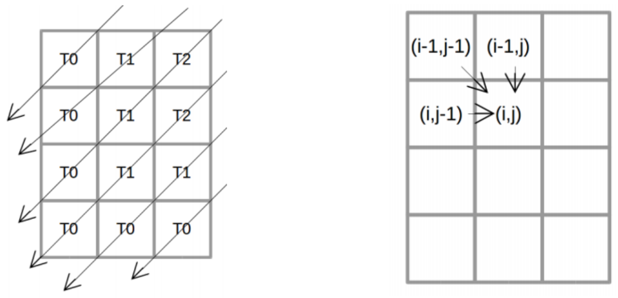
$$d_{i,j} = \psi_{i,j} + min\{d_{i-1,j}, d_{i-1,j-1}, d_{i,j-1}\}, \tag{2}$$

The TWDTW is generally used in conjunction with the k-NN algorithm in LULC classifications. However, with time and space complexity $O(n^2)$ (i.e., the bigger the series and patterns, the bigger the matrix to be computed), high demand for computational resources, and high execution time, due to the increase in the volume of RSTS data, the use of the TWDTW algorithm has become challenging, especially in large areas, given the large volume of data to be processed.

2.2 Spatial Parallel TWDTW

The ST-TWDTW algorithm is the result of the first attempt in exploring GPU-based parallelism to optimize parallel processing of RSTS. This solution is based on the traditional strategy of computing elements in diagonals in parallel wavefront (Figure 1a) [Oliveira et al. 2018]. Each diagonal is processed in parallel, given the dependency on previous elements, this approach is used in the D matrix. In this matrix, the computation of each (i, j) element depends on the $(i - 1, j)$, $(i, j - 1)$ and $(i - 1, j - 1)$ elements previously calculated, as illustrated in Figure 1b.

SP-TWDTW target a highly multi-threaded GPU, and calculates one element per thread at each step when computing the D matrix. To ensure correctness, the number of threads in each block is determined as the size of the main diagonal, which is equal to the minimum value of the size of the pattern and the size of the time series. Since this operation is very simple and presents a very low workload for each thread (i.e., each thread computes only one element of the matrix per step of



(a) Diagonal strategy (b) Data depence

Fig. 1. SP-TWDTW strategy to compute D matrix.

the algorithm), it causes a large amount of processing idleness during its execution, thus decreasing performance. Also, the GPU/CUDA programming architecture is currently limited to 1024 threads for each block [NVIDIA 2020]. Given this limitation, the SP-TWDTW is not able to work when patterns and time series are both greater than 1024.

2.3 LULC classification based on meta-features

The use of machine learning algorithms for the classification of LULC is essential to create models that enable automated and recurrent mapping. The DTW-based algorithms can generate similarity measures that can be used as meta-features to carry out the classification of LULC. These meta-features are agnostic to the classification model and have been used with some well-known classification algorithms. The work of [Maus 2016] proposes the TWDTW algorithm in conjunction with k-NN, in an approach to classify LULC based on RSTS similarity measures. The work of [Oliveira et al. 2019] uses meta-features as a complement to the classification methodology based on the SVM classifier proposed in [Picoli et al. 2018].

To evaluate the meta-features generated by Rapid-DTW, this article used as a basis the methodology of classification of LULC presented in [Picoli et al. 2018]. The authors have proposed the use of time series as input vectors for the SVM classifier. Originally, the method does not transform the time series data to generate new features. This article expands this approach, adding the meta-features to train classifiers based on the Random Forest algorithm. Also, a new Random Forest model classification is presented using the meta-features generated by the Rapid-DTW to improve accuracy when compared to the TWDTW with k-NN approach presented in [Oliveira et al. 2019].

Random Forest is one of the most widely used machine learning algorithms for LULC classification [Pal 2005; Belgiu and Drăguţ 2016; Gislason et al. 2006]. It is possible to observe several current works that use Random Forest for mapping large areas, showing this ability to deal with large volumes of data [Parente and Ferreira 2018; Tsai et al. 2018; Parente et al. 2019; Ayala-Izurieta et al. 2017]. According to [Pal 2005] Random Forest's popularity occurs due to its capacity to achieve an accuracy similar to SVM, together with ease of use, with few parameters to be configured by the user and adaptation to remote sensing data.

3. RAPID-DTW: AN EFFICIENT SOLUTION TO GENERATE META-FEATURES

The Rapid-DTW[2] is a new DTW-based algorithm that calculates the elements of the dynamic programming matrix D using windows of variable size. At each step of the algorithm, windows of elements

[2]`https://github.com/urzedabr/RAPID-DTW`

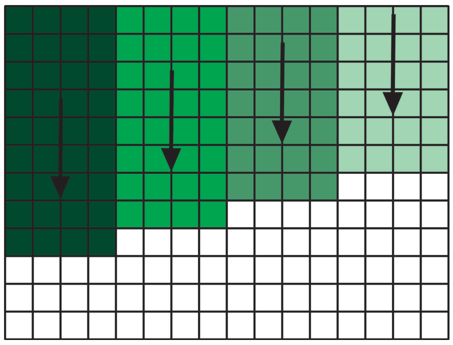


Fig. 2. Wavefront of the "D" matrix with Rapid-DTW.

are computed in parallel. The sizes of these windows can be defined according to the hardware architecture and the size of the data entry. The main idea is to determine the workload performed by each processing unit in each step. This approach allows the method to experiment with different window sizes to obtain an ideal configuration for specific instances of the problem. In this way, it is possible to reduce processing idleness, which improves the use of GPU resources, enabling better performance at runtime.

The Rapid-DTW changes the wavefront in which the D matrix is computed. Rather than performing the computation of the elements in a diagonal wavefront, as the SP-TWDTW does, the window strategy performs the computation in elements within a given window. This results in the computation wavefront to be carried out vertically, allowing the dependence of data internally in each window in a sequential manner while guaranteeing the dependence of data between the windows of elements in parallel. Figure 2 illustrates the computation wavefront given the new strategy.

As illustrated in Figure 3, within each window the processing unit performs its work in a sequential manner, ensuring data dependency, while windows with the same color can be processed in parallel. The larger the defined window, the greater the workload of each processing unit, which implies less processing idleness. The capacity to choose the size of the window allows the diversification of the processing load, aiming at optimizing the use of resources, according to the size of the input data. This characteristic makes Rapid-DTW able to compute measures of similarity between patterns and time series without worrying about scalability limitations.

Designed to work specifically with RSTS, the Rapid-DTW performs 4 steps to generate meta-features. First, the algorithm concatenates patterns and time series in queues in a coordinated way, avoiding exceeding the CPU and GPU memory sizes. Then the global weight matrix Ψ is computed with an embarrassingly parallel approach (i.e., requires no effort to separate the problem into parallel tasks, as there exists no data dependency), using the Time-Weighted technique. From the Ψ weight matrix, the dynamic programming matrix D is computed according to the number of threads and

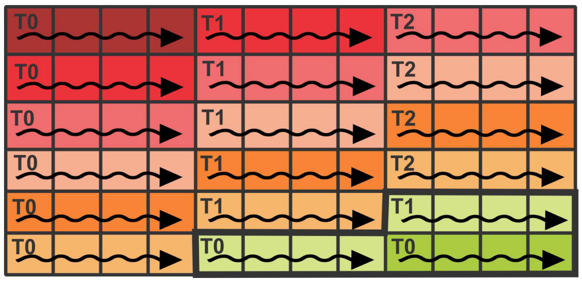


Fig. 3. Behavior of processing units inside the window.

window size defined, and finally, the lowest cost path for each pair of pattern and time series is calculated.

The computation of the D matrix is described in detail in the Algorithm 1. Between the second and fourth lines, constant variables are defined, this is done to prevent loop operations from occurring several times, optimizing the execution of the algorithm. In line 2, the window size is calculated according to the number of threads. The aux and $auxj$ variables assist in locating the elements of each window, while the $base$ variable is responsible for keeping each thread within the specified window size.

Algorithm 1: Rapid-DTW - Computation of D matrix

```
   Data: $\Psi$ Weight matrix
   $y$: number of lines
   $x$: number of columns
   $num\_threads$: number of threads
   Result: $D$ Matrix
 1 $tid \leftarrow$ thread id
 2 $windowSize \leftarrow x/num\_threads$
 3 $tidWindow \leftarrow tid * windowSize$
 4 $tidWindowaux \leftarrow tid * (windowSize - 1)$
 5 for $(si = 0; si < y; si++)$ do
 6     $base \leftarrow tidWindow + (si - tid) * x$
 7     $auxj \leftarrow tidWindowaux$
 8     if $(tid \leq min(si, x - 1))$ then
 9         All threads in paralalel do:
10         for $(index = base; index < base + windowSize; index++)$ do
11             $i \leftarrow si - tid$
12             $j \leftarrow tid + auxj$
13             $update\_element(i, j)$
14             $auxj \leftarrow auxj + 1$
15         end
16     end
17     $sync\_barrier$
18 end
19 $si \leftarrow (y - 1 - tid) * x$
20 $auxj \leftarrow 0$
21 for $sj \leftarrow ((x/windowSize) - 2; sj \geq 0; sj--)$ do
22     $base \leftarrow tidWindow + si + windowSize + auxj$
23     $aux \leftarrow 0$
24     $auxj \leftarrow auxj + windowSize$
25     if $(tid \leq min(sj, y - 1))$ then
26         All threads in paralalel do:
27         for $(index = base; index < base + windowSize; index++)$ do
28             $i \leftarrow y - tid - 1$
29             $j \leftarrow x - (windowSize * sj) - windowSize + aux + tidWindow$
30             $update\_element(i, j)$
31             $aux \leftarrow aux + 1$
32         end
33     end
34     $sync\_barrier$
35 end
```

The loop between lines 5 and 18 computes the top of the matrix, indexed according to the number of lines. Within this loop, in lines 6 and 7 we have the variables $base$ and $auxj$. It is necessary to ensure that each thread works, in each iteration, only within the defined window. The $base$ variable is responsible for keeping the thread within the specified window size. Meanwhile, within each iteration, the variable j can take on different values according to the size of the window. For this, we use the variable $auxj$, which will update the value of j according to the size of the window.

Line 8 defines a condition that controls the number of threads working simultaneously in each iteration, as it is necessary to ensure that each thread only starts its computation after the previous threads have finished computing its window. At the end of the condition, in line 17, a synchronization call is made using the *sync_barrier* function. This barrier synchronization ensures that all threads end their operations together, respecting the data dependency. In line 10 we have the loop to perform the calculations of the matrix elements, using the variable *base* to define where the window starts according to the thread id. In this loop, the elements i and j are defined to be updated by the function $update\_element(i, j)$, which updates each element according to the equation 2 presented in section 2.1.

Between lines 21 and 35, the bottom part of the matrix is computed, similarly to the top part. In lines 19 and 20, the variables *si* and *auxj* are reset. The *si* variable is necessary so that *base* can control the size of the window, as previously done. The *aux* variable will assist in defining the initial computation index for each thread, considering that when computing the lower diagonal elements, the initial value of the variable j is different for the thread. Unlike computation at the top, where the variable j could assume a fixed amount of different values at each interaction equal to the size of the window, at the bottom, the number of different values becomes the subtraction of the number of columns by the size of the window.

Line 21 has the same function as line 5 previously presented, however, now the control is done based on the number of columns. Lines 22, 23, and 24 initialize the *base*, *aux* and *auxj* variables that will, at each interaction, control the amount of work for each thread according to the size of the window, and the values (i, j) to be computed. Between lines 25 and 33, the algorithm works exactly as in the computation of the top part presented above, with only changes in the indexing equations of the elements.

4. EXPERIMENTATION AND EVALUATION

This section presents the experiments and results obtained. In this article, we defined two types of experiments. Firstly the performance of Rapid-DTW in the generation of meta-features is compared to the SP-TWDTW and the sequential version of TWDTW in C++ language. In a second experimentat, meta-features generated by Rapid-DTW are applied to LULC classification models based on the Random Forest algorithm. All experiments were performed on a computer with an Intel Core i7-9700 processor (3.2 GHz and 8 MB Cache), 16GB DDR4 RAM, and NVIDIA GeForce GTX 1660 Ti video card with 6 GB GDDR6 of memory with Turing architecture, 1536 CUDA cores, 1770 MHz of frequency.

4.1 Performance Analysis of Rapid-DTW

In this section, experiments are presented that aim to evaluate the performance of Rapid-DTW in relation to other solutions of the literature. The algorithms were executed 10 times in each experiment performed. The results presented are the arithmetic average of the 10 runs. A statistical analysis of the results was also performed, showing standard deviations and confidence intervals with 95% confidence, normal, and t-student. Having in mind that the final result of the algorithms is the same (exact solution), and the impact on the execution time of the D matrix computation is related to the number of observations of the time series, to carry out experiments with time series of different sizes, synthetic input databases were generated from the MOD13Q1 database presented in [Oliveira et al. 2019].

The Rapid-DTW computes the D matrix in element windows that can be sized according to the size of the data entries. Thus, during the experiments, the window sizes were varied between 2 and 384. A standard window size configuration pattern was used in all tests, always seeking to have blocks with 32 threads. In the CUDA architecture, when a large number of synchronizations is necessary, it

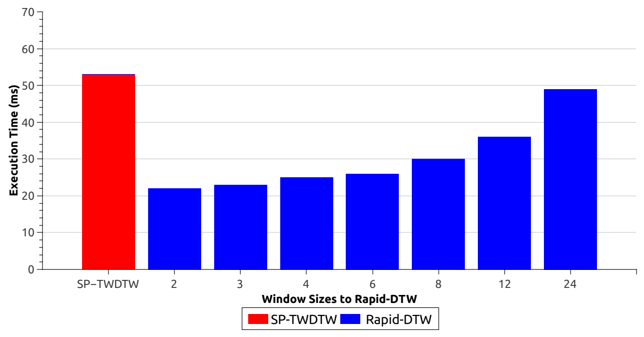


Fig. 4. Execution time of the D matrix with Rapid-DTW and SP-TWDTW. Patterns size 24 and time series size 50.

is recommended to work with blocks that have a smaller number of threads. Multiple-sized blocks of 32 also perform better, due to the warp instructions defined by the architecture [NVIDIA 2020]. In this way, the experiments evaluate both the performance and the adaptability of the algorithms to the data set.

Initially, tests were performed to evaluate the performance of Rapid-DTW on data sets with sizes commonly used in the literature for classification using MOD13Q1 data [Picoli et al. 2018; Maus et al. 2016]. Thus, 50 patterns and 1000 time series were generated, with 24 observations for the patterns and 50 observations for the time series. For this type of experiments, the syntactically generated patterns represent time series in a given period of time, and not necessarily real world LULC classes behaviors. In this experiment, only the execution time of the D matrix was evaluated. Figure 4 presents a graph with the execution time results of this step, with the comparison between the execution time of the D matrix by the SP-TWDTW and Rapid-DTW algorithms, the latter with different sizes of element windows.

It can be observed that the Rapid-DTW algorithm had a response time of 2.4 times less than the SP-TWDTW with the window of size 2, which is the smallest multiple of 24. As the size of the window increases, the difference between execution times decreases. This was due to the number of threads per block in this experiment being less than 32. As CUDA architecture makes use of warps, minimum sets of 32 threads that share instructions and memory, decrease the number de threads in this scenario has a greater weight than the gain with the decrease of processing idleness. With few threads working inside the block, the degree of parallelism decreases, causing a loss of performance in the algorithm. So the best result was obtained with a window size equal to 2, implying 16 threads per block. Still, it is possible to observe that Rapid-DTW performed better in all cases.

To experiment with larger time series, aiming to demonstrate the Rapid-DTW's ability to process time series of the next generations of satellites, another data set was generated, with 50 patterns and 1000 time series of sizes varying from 48 to 768 observations for patterns and 100 to 1600 observations for time series. In this experiment, the complete execution times of the TWDTW in C ++, SP-TWDTW, and Rapid-DTW in the generation of meta-features were evaluated. The results of these tests are illustrated in the graph shown in Figure 5a. Given these results, it is important to note that as we increase the size of the problem entry, the performance of Rapid-DTW concerning improves. This is due to the possibility of adapting the size of the window to the size of the entries, because by increasing the size of the window, we decrease the number of idleness threads during execution, allowing achieve better results in larger data sets. As all versions are evaluated in this experiment, the statistical analysis, containing the time of all 10 runs, the standard deviation, and the confidence intervals is shown in the Table I. According to the statistical results, the standard deviation and confidence intervals are small, indicating a low variability of the measurements and consequently a small margin of error.

Table I. Statistical analysis. Ex1 to Ex10 informs the execution time of each test. All information in the table are presented in milliseconds.

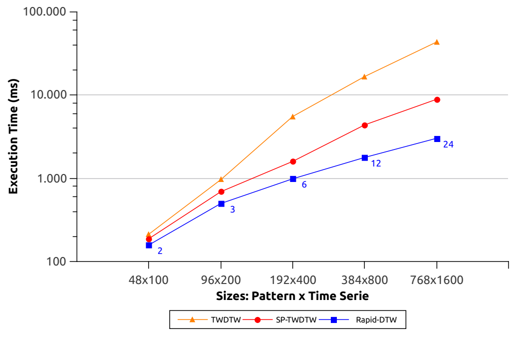
Rapid-DTW Results														
Data input	**Ex1**	**Ex1**	**Ex3**	**Ex4**	**Ex5**	**Ex6**	**Ex7**	**Ex8**	**Ex9**	**Ex10**	**Average**	**Standard deviation**	**Normal confidence interval**	**T-student confidence interval**
48x100	154	159	160	158	161	152	161	158	158	159	158	2.91	1.80	2.08
96x200	496	499	492	501	502	498	499	495	499	500	498.1	3.00	1.86	2.14
192x400	988	991	985	987	986	988	985	988	990	985	987.3	2.11	1.31	1.51
384x800	1788	1784	1780	1788	1780	1781	1782	1777	1781	1782	1782.3	3.50	2.17	2.50
768x1600	3025	3025	3027	3030	3033	3029	3024	3040	3028	3022	3028.3	5.21	3.23	3.73

P-TWDTW Results														
Data input	**Ex1**	**Ex1**	**Ex3**	**Ex4**	**Ex5**	**Ex6**	**Ex7**	**Ex8**	**Ex9**	**Ex10**	**Average**	**Standard deviation**	**Normal confidence interval**	**T-student confidence interval**
48x100	185	185	185	189	191	192	190	189	188	188	188.2	2.53	1.57	1.81
96x200	682	685	684	686	689	690	694	690	690	690	688	3.62	2.24	2.59
192x400	1594	1597	1597	1597	1596	1599	1600	1602	1597	1599	1597.8	2.25	1.40	1.61
384x800	4385	4385	4386	4386	4387	4390	4392	4393	4393	4395	4389.2	3.82	2.37	2.74
768x1600	8910	8910	8910	8911	8911	8912	8912	8915	8914	8915	8912	2.00	1.24	1.43

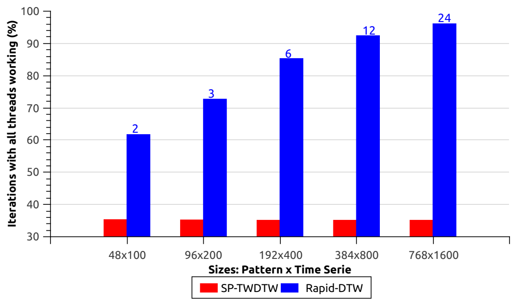
TWDTW Results														
Data input	**Ex1**	**Ex1**	**Ex3**	**Ex4**	**Ex5**	**Ex6**	**Ex7**	**Ex8**	**Ex9**	**Ex10**	**Average**	**Standard deviation**	**Normal confidence interval**	**T-student confidence interval**
48x100	210	212	210	213	215	214	214	213	215	213	212.9	1.79	1.11	1.28
96x200	959	961	964	963	963	963	963	964	965	964	962.9	1.73	1.07	1.24
192x400	5536	5538	5538	5538	5538	5538	5539	5538	5538	5539	5538	0.82	0.51	0.58
384x800	16736	16736	16738	16738	16738	16737	16739	16741	16738	16739	16738	1.49	0.92	1.07
768x1600	43516	43518	43517	43519	43519	43519	43521	43522	43524	43526	43520.1	3.14	1.95	2.25

To better understand the optimization of the use of threads, it is possible to abstract from these tests the percentage of iterations in which all threads work simultaneously during the computation of the D matrix. This data is obtained through the ratio between the number of iterations with all threads running in parallel and the total iterations for computing the entire matrix. This data is illustrated in the graph of Figure 5b. It is observed that SP-TWDTW always has approximately 35.5% iterations with all threads working simultaneously in any data sets, while Rapid-DTW reaches 96.1% of iterations with all threads working in the 768×1600 entry set and size window 24.

The quadratic time and space complexity of DTW-based algorithms, when combined with large input values for n, implies high execution times and memory storage usage. Faced with this challenge, a final set of data was created to test patterns and time series simulating spatial sensors with extremely high temporal resolution. For this experiment, only 5 patterns and 10 time series were generated, with sizes ranging from 1536 to 12288 observations for patterns and 3200 to 24800 observations for the time series. The SP-TWDTW was not used in this last data set, due to its scalability limitation. Figure 6a shows the graph of this experiment. As the speedup calculation of a parallel solution is obtained by comparing it with the best sequential solution, the graph in Figure 6b presents Rapid-DTW speedup to the TWDTW implemented in the $C++$ language.

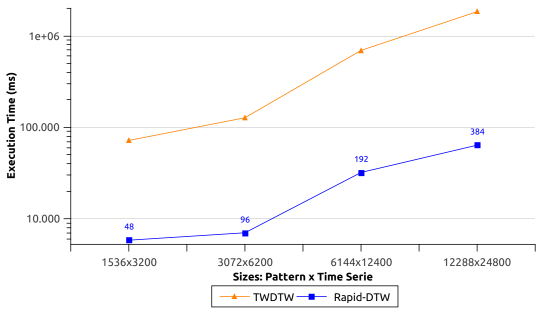


(a) Execution times with different input sizes

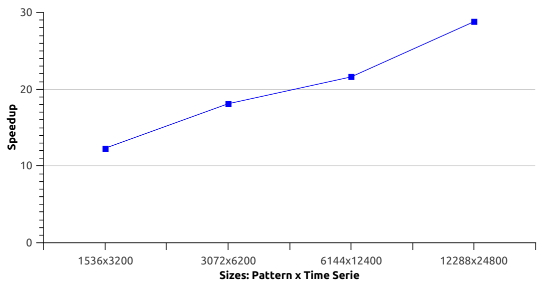


(b) Percentage of iterations in which all threads work simultaneously

Fig. 5. Execution time comparison between TWDTW, SP-TWDTW and Rapid-DTW. The blue numbers next to the Rapid-DTW results correspond to the size of the window used in the experiment.



(a) TWDTW and Rapid-DTW execution times for very large input sizes



(b) Rapid-DTW speedup compared to the best sequential TWDTW version

Fig. 6. Execution time comparison between TWDTW and Rapid-DTW. The blue numbers next to the Rapid-DTW results correspond to the size of the window used for Rapid-DTW.

The results of this experiment behave in the same way as the results of the previous experiment. The greater the volume of input data, the better the performance of the algorithm concerning the other solutions. In tests where it was possible to compare Rapid-DTW with SP-TWDTW, the greatest performance gain was obtained in the largest set of tests, reaching a run time of 2.9 times less than SP-TWDTW. Compared to the sequential version, Rapid-DTW achieved a speedup of up to 28.8 times on the largest data set evaluated. This happens due to the increase in the execution time of the D matrix inherent to the increase in the size of the data of patterns and time series. In this case, the computation time of the D matrix is significantly longer than the rest of the steps, reaching 87% of the execution time in the largest data set evaluated.

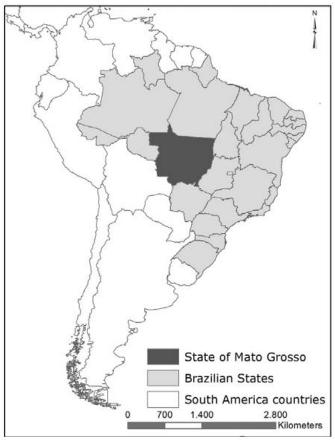
4.2 Effectiveness of Meta-features in LULC Classification

The meta-features generated by Rapid-DTW can be used in various analyzes, including the LULC classification. To evaluate the meta-features generated by Rapid-DTW, this research used as a basis the LULC classification methodology presented in [Picoli et al. 2018]. For comparison purposes, the same data set was used. The database presents MOD13Q1 data extracted from the region of Mato Grosso - Brazil (Figure 7a), to classify nine different classes of LULC. The annual patterns of the classes are represented in Figure 7b.

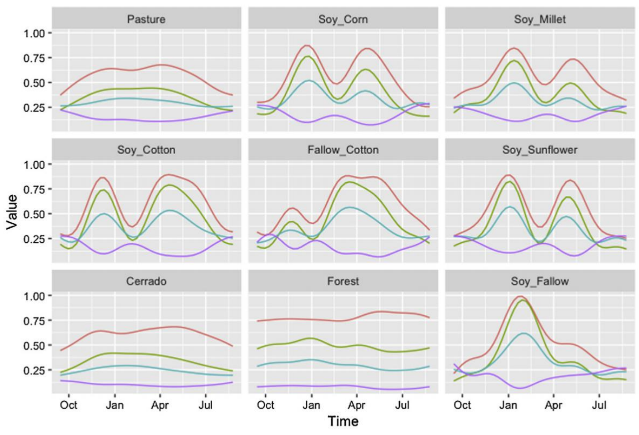
The samples are distributed as follows: Cerrado (400), Cotton-fallow (34), Forest (138), Pasture (370), Soy-corn (398), Soy-cotton (399), Soy-fallow (88), Soy-millet (235), and Soy-sunflower (53), totaling 2115 samples with 23 observations of near-infrared (NIR), mid-infrared (MIR), Enhanced Vegetation Index (EVI) and Normalized Difference Vegetation Index (NDVI) bands each sample. The complexity of classifying this data set is high, due to the similar behavior of the time series associated with agriculture [Picoli et al. 2018; Oliveira et al. 2019].

The meta-features were generated for each point of the samples using Rapid-DTW. These similarity measures represent how close the behavior of each time series is to each of the nine patterns analyzed. Therefore, for each pixel analyzed, there are 92 characteristics (i.e., two bands and two indices of 23 observations per pixel) and 9 meta-features, one for each class. Table II presents a summed up example of a feature space having 3 samples with 2 observations for each band and index and 3 meta-features, for the classes Cerrado, Cotton-fallow and Forest.

To analyze and evaluate the impact of using similarity-based meta-features generated from the Rapid-DTW algorithm in LULC, a methodology was developed to carry out experiments using different feature spaces. With the first stage consolidated, training and validation of the classifiers are carried out, ending with a stage of analysis and evaluation of the results. A flowchart of the method-



(a) Mato Grosso, Brazil, location of the samples database.



(b) Annual patterns of each class.

Fig. 7. Study area and annual behavior patterns of each class. Adapted from [Picoli et al. 2018].

Table II. Feature space example

Sample	**ndvi1**	**ndvi2**	**evi1**	**evi2**	**nir1**	**nir2**	**mir1**	**mir2**	**meta-feature Cerrado**	**meta-feature Cotton-fallow**	**meta-feature Forest**
1	0.39	0.49	0.25	0.28	0.32	0.27	0.31	0.17	3.53	4.03	4.00
2	0.50	0.49	0.26	0.33	0.23	0.36	0.14	0.16	2.62	4.27	2.91
3	0.35	0.34	0.19	0.16	0.23	0.20	0.20	0.15	1.70	4.08	3.34

ology used to carry out the tests and evaluation is presented in Figure 8. Through this methodology, it was possible to compare the impact of meta-features in different scenarios (A, B, and C), described in sections 4.2.1, 4.2.2 and 4.2.3.

To analyze and evaluate the impact of using similarity-based meta-features generated from the Rapid-DTW algorithm in LULC, different feature spaces were created. A methodology was developed that starts with the selection of different features spaces composed in different ways. With the first stage consolidated, training and validation of the classifiers are carried out, ending with a stage of analysis and evaluation of the results. A flowchart of the methodology used to carry out the tests and evaluation is presented in Figure 8. Through this methodology, it was possible to compare the impact of meta-features in different scenarios (A, B, and C), described in sections 4.2.1, 4.2.2 and 4.2.3.

The Random Forest's models were trained using 500 trees, as there was no improvement in the results with the increase in the number of trees. To estimate accuracy, all experiments were performed using the K-Fold cross-validation technique, with K = 5. Finally, confusion matrices were generated with the producer accuracy (PA), the user accuracy (UA), and overall accuracy (OA) from the results obtained. The Kappa coefficient was also calculated to present a baseline classification results.

4.2.1 *Scenario A.* This scenario aims to analyze the approach presented by [Picoli et al. 2018] applied to the Random Forest classifier in conjunction with the meta-features. In a first experiment, the classifier is trained with a feature space that contains all 23 annual observations of the NIR and MIR bands and the NDVI and EVI indices, totaling a feature space of size 92. Then, a second experiment is performed, adding to this feature space the 9 meta-features generated by Rapid-DTW, totaling 101 features per pixel. Given the results, the confusion matrices of the 2 experiments are analyzed, to observe points in which the addition of the meta-features had a positive or negative

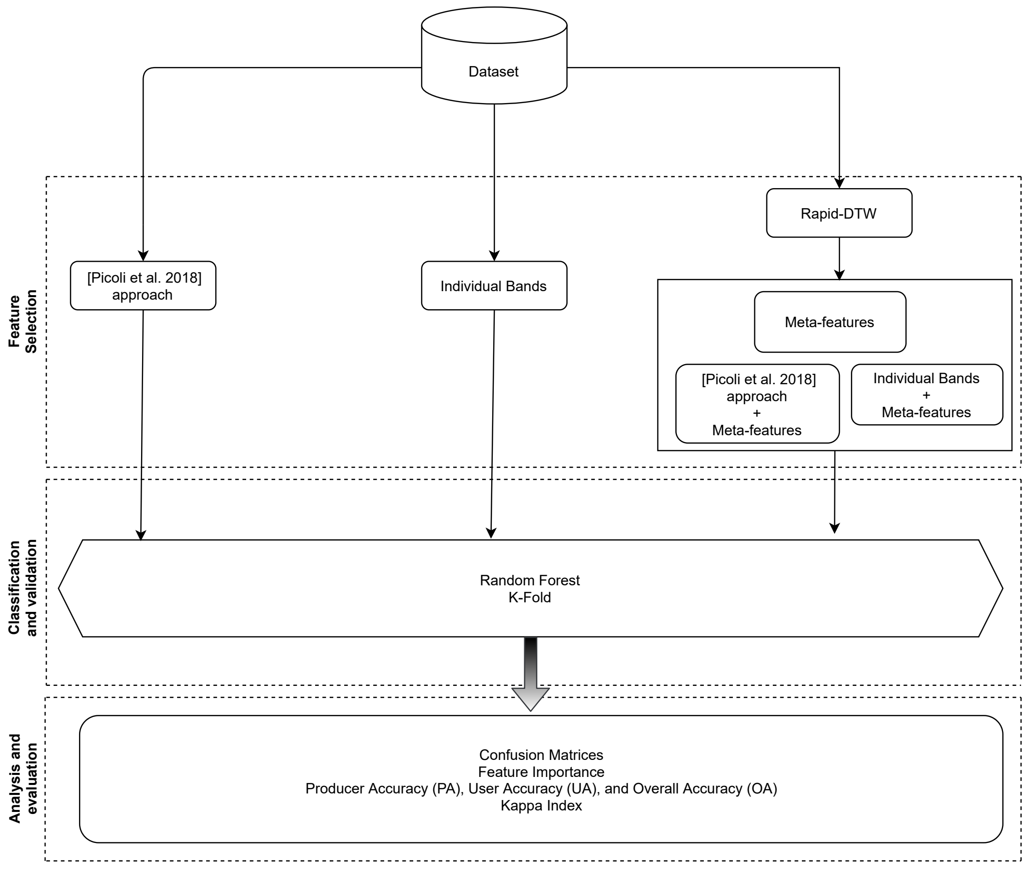

Dataset
Rapid-DTW
Feature Selection
[Picoli et al. 2018] approach
Individual Bands
Meta-features
[Picoli et al. 2018] approach + Meta-features
Individual Bands + Meta-features
Classification and validation
Random Forest K-Fold
Analysis and evaluation
Confusion Matrices
Feature Importance
Producer Accuracy (PA), User Accuracy (UA), and Overall Accuracy (OA)
Kappa Index

Fig. 8. Methodology used to evaluate the application of meta-features in LULC models

impact. In this scenario, an analysis of the feature importance was also carried out, to identify the relevance of meta-features in the LULC classification with Random Forest.

The results of the first experiment in this scenario can be seen in Table III. In this experiment, the Random Forest classifier was trained with 23 observations of the bands (NIR, MIR) and indices (EVI, NDVI), totaling 92 characteristics per pixel. Among the results, the experiment got an overall accuracy of 92.48% (Kappa 0.91). In this experiment, the classifier achieved good accuracy (i.e., greater than 90% for PA) for the Cerrado, Forest, Pasture, Soy-corn, Soy-cotton, and Soy-fallow classes. There was a lot of confusion between the classes Soy-sunflower and Soy-corn due to the similar behavior of the time series of these classes. The classifier was not able to differentiate between them, causing many samples of Soy-Sunflower to be classified as Soy-Corn. The same occurred between the Soy-corn and Soy-millet classes, making the Soy-millet class also does not have a PA above 90%. The similar behavior of classes Cotton-fallow and Soy-cotton hindered the performance of the classification of class Cotton-fallow, causing it to obtain only a PA of 73.53%.

When adding the 9 meta-features per pixel, generating a feature space of size 101, the accuracy results were improved. These results can be seen in Table IV. In this experiment, the overall accuracy obtained was 93.05% (Kappa 0.92). In this scenario, the contribution of the meta-features was more significant in the hard classes (i.e., classes that, because they have similar behavior in the time series, end up confusing the classifier). Tracing a comparison between the experiments carried out, we had an increase in the accuracy of the producer of 8.82% in the Cotton-fallow class, 2.13% in Soy-millet, and 3.78% in Soy-sunflower. This result shows that meta-features can contribute to the classification of complex classes, being able to increase the accuracy in different classes of agriculture.

Table III. [Picoli et al. 2018] approach confusion matrix

	1	2	3	4	5	6	7	8	9	**UA (%)**
1 - Cerrado	**398**	0	1	8	1	0	0	0	0	97.50%
2 - Cotton-fallow	0	**25**	0	0	0	2	0	0	0	94.12%
3 - Forest	2	0	**136**	1	0	0	0	0	0	97.83%
4 - Pasture	0	3	1	**359**	5	1	0	9	0	94.86%
5 - Soy-corn	0	0	0	0	**360**	26	0	27	16	82.66%
6 - Soy-cotton	0	6	0	1	8	**366**	0	3	0	95.49%
7 - Soy-fallow	0	0	0	0	0	0	**87**	1	0	98.86%
8 - Soy-millet	0	0	0	1	23	4	1	**195**	7	84.68%
9 - Soy-sunflower	0	0	0	0	1	0	0	0	**30**	98.11%
PA (%)	99.50%	73.53%	98.55%	97.03%	90.45%	91.73%	98.86%	82.98%	56.60%	**OA = 92.48%**

Table IV. [Picoli et al. 2018] approach + meta-features confusion matrix

	1	2	3	4	5	6	7	8	9	**UA(%)**
1 - Cerrado	**396**	0	1	6	0	0	0	0	0	98.25%
2 - Cotton-fallow	0	**28**	0	0	0	1	0	0	0	97.06%
3 - Forest	2	0	**136**	1	0	0	0	0	0	97.83%
4 - Pasture	2	1	1	**361**	5	1	0	6	0	95.68%
5 - Soy-corn	0	0	0	0	**360**	23	0	26	14	84.17%
6 - Soy-cotton	0	4	0	1	8	**369**	0	2	0	96.24%
7 - Soy-fallow	0	0	0	1	0	1	**86**	1	0	96.59%
8 - Soy-millet	0	1	0	0	24	4	2	**200**	7	83.83%
9 - Soy-sunflower	0	0	0	0	1	0	0	0	**32**	98.11%
PA (%)	99.00%	82.35%	98.55%	97.57%	90.45%	92.48%	97.73%	85.11%	60.38%	**OA = 93.05%**

To assess the impact of meta-features during this classification experiment, an importance function (Gini Importance) was used to rank the importance of each feature during the classification carried out by Random Forest. This function uses the gini impurity metric to rank the features that are most capable of decreasing the degree of impurity during classification [Belgiu and Drăguţ 2016]. The analysis allows to identify that the importance of these 9 meta-features per pixel is significantly greater when compared to the pure data of bands and indices. Figure 9 presents the feature importance graph. It is assessed that the meta-features have a high representation of the samples, making them more important during the classification. In this scenario, meta-features were able to increase the accuracy result of a methodology that already had high accuracy values.

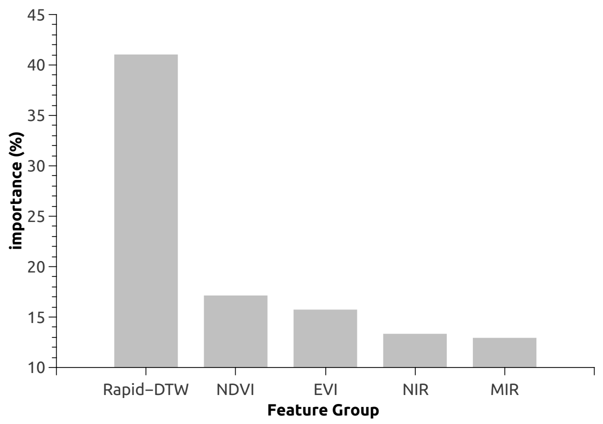


Fig. 9. Feature importance in the Random Forest classification

Table V. Comparison between experiments Indivual bands and Individual bands + meta-features

	Individual bands	**Individual bands + meta-features**
1 NIR	OA = 86.8%	OA = 90.35%
	Kappa = 0.84	Kappa = 0.88
2 MIR	OA = 83.26%	OA = 89.17%
	Kappa = 0.80	Kappa = 0.87
3 EVI	OA = 87.47%	OA = 90.3%
	Kappa = 0.85	Kappa = 0.88
4 NDVI	OA = 86.43%	OA = 90.4%
	Kappa = 0.84	Kappa = 0.88

4.2.2 *Scenario B.* In this scenario, situations are analyzed in which the feature space is smaller, with low accuracy values. For this, experiments were carried out with the bands and indices individually, training 4 classifiers, with features spaces composed only by 23 observations per pixel, one classifier for each available data (NDVI, EVI, NIR, and MIR). Then, the training is performed again, with the addition of 9 meta-features for each pixel, now with feature spaces size 32. For these experiments, only the global accuracy in the classification of the samples for each classifier was compared, the results can be observed in Table V.

With the reduced feature space (i.e., using only the 23 observations of a Band or Index, added to 9 meta-features generated by Rapid-DTW), it was possible to observe an improvement of up to 3.97% in the overall accuracy in the NDVI model, reaching the highest global accuracy of this test scenario, with 90.40%. The experiment that benefited the most from the addition of meta-features was the one with the lowest overall accuracy, the MIR model, achieving an improvement of up to 5.91%. These results demonstrate that these meta-features can be applied in several classification models, which can lead to overall accuracy improvements.

4.2.3 *Scenario C.* All previous scenarios used [Picoli et al. 2018] strategy as a basis for the classification, using time series observation data without any transformation as features and then evaluating the same experiments with the addition of the meta-features generated by Rapid-DTW. In this comparison scenario, a different approach is proposed. Only the 9 similarity-based meta-features generated by Rapid-DTW are used as input for the Random Forest algorithm. This strategy aims to compare the LULC classification methodology using the TWDTW algorithm in conjunction with k-NN [Maus 2016; Oliveira et al. 2019].

Table VI presents the confusion matrix with the results of this experiment. The Random Forest model reached an overall accuracy of 84.02% (Kappa 0.81) using only the 9 meta-features generated by Rapid-DTW. The same experiment, using the same data set, was performed in the work of [Oliveira et al. 2019], in which a combination of the meta-features generated by SP-TWDTW with k-NN was proposed, obtaining an overall accuracy of 78%. In this case, Random Forest managed to obtain a better result, in terms of accuracy, of 6.2% compared to k-NN, without adding new features.

Table VI. Classification confusion matrix using only the 9 meta-features generated by Rapid-DTW

	1	2	3	4	5	6	7	8	9	**UA(%)**
1 - Cerrado	**375**	0	7	16	0	0	0	0	0	94.25%
2 - Cotton-fallow	0	**13**	0	0	0	5	0	0	0	85.29%
3 - Forest	11	0	**126**	4	0	0	0	0	0	89.13%
4 - Pasture	14	0	5	**343**	2	0	0	7	0	92.43%
5 - Soy-corn	0	2	0	1	**326**	48	0	44	35	67.34%
6 - Soy-cotton	0	17	0	1	22	**333**	0	6	2	87.97%
7 - Soy-fallow	0	1	0	0	0	1	**81**	7	0	89.77%
8 - Soy-millet	0	1	0	5	46	12	7	**171**	7	66.81%
9 - Soy-sunflower	0	0	0	0	2	0	0	0	**9**	96.23%
PA(%)	93.75%	38.24%	91.30%	92.70%	81.91%	83.46%	92.05%	72.77%	16.98%	**OA = 84.02%**

Table VII. Classification confusion matrix using only the 9 meta-characteristics simplifying the agricultural classes

	1	2	3	4	**UA (%)**
1 - Cerrado	**375**	7	16	0	94.25%
2 - Forest	11	**127**	3	0	89.86%
3 - Pasture	14	4	**337**	5	93.78%
4 - Agriculture	0	0	14	**1202**	98.84%
PA(%)	93.75%	92.03%	91.08%	99.59%	**OA = 96.50%**

It can be observed that the accuracy values of the producer remained above 90% for the Cerrado, Forest, Pasture, and Soy-Fallow classes. In opposition, the rest of the classes, related to agricultural cultivation, had worse results. This is because these classes exhibit similar behaviors in their time series, making it difficult for the classifier to differentiate, and consequently obtaining good accuracy results [Picoli et al. 2018].

Finally, given the similarities between the classes of agriculture, a new approach of the experiment was carried out unifying the classes Cotton-Fallow, Soy-corn, Soy-cotton, Soy-fallow, Soy-cotton, Soy-millet and Soy-Sunflower in a single class called Agriculture. In this approach, it was possible to observe better results of accuracy. The classifier managed to obtain 96.50% of global accuracy (Kappa 0.94). For the new class of Agriculture, a producer accuracy of 99.59% was obtained. With these results, in scenarios where there is no need to typify the classification of agriculture, the use of meta-features can become a competitive approach. Table VII presents the confusion matrix of this experiment.

5. CONCLUSIONS AND FUTURE WORK

Given a large number of satellites being constantly launched into Earth's orbit, and their increasingly powerful sensors, it is expected that the temporal and spatial resolutions of the RSTS will continue to increase in the future. Therefore, the computational cost to process this large volume of data must also increase proportionately. It is possible to state that the future of Remote Sensing may depend on the exploitation of high-performance computing tools [Houborg and McCabe 2018; Hansen and Loveland 2012; Plaza 2008; Camara et al. 2016].

In this research, a parallel algorithm for the generation of meta-features based on RSTS similarity was presented. These meta-features can be used mainly in the composition of characteristic spaces for automated LULC classification. A feature space that represents well the elements to be classified is crucial for good accuracy of the classifier [Belgiu and Drăguţ 2016]. The meta-features generated by Rapid-DTW are a compilation of information from the entire time series of bands and indices. For this reason, these meta-features provide an excellent representation of the classes, causing a positive impact on the accuracy of the classification.

The Rapid-DTW exploits parallelism to handle the large volume of remote sensing data, demonstrating the potential of parallel processing strategies. The experiments carried out showed that Rapid-DTW presents significant improvements in performance in comparison with the traditional methods present in the literature. Also, this work proposes the application of meta-features in classification models based on the Random Forest classifier. This approach showed that it is possible to obtain improved accuracy in several models with the addition of meta-features in their feature spaces. Through these results, it was possible to develop strategies that use only the meta-features as input for the classifier, maintaining good accuracy indexes. Despite the high computational cost for the generation of meta-features, once they are generated by Rapid-DTW, they can be used in several classification models.

However, there are some limitations to this work. This methodology does not deal directly with the satellite images. Therefore, all inputted data must be pre-processed in order to create a table with the time series data and their patterns. In future work, we plan to apply of our method in different regions in order to verify its performance in runtime and accuracy. For that, we intend to create a method to directly process the satellite images and input its results to Rapid-DTW into a more complete methodology.

REFERENCES

Ayala-Izurieta, J. E., Márquez, C. O., García, V. J., Recalde-Moreno, C. G., Rodríguez-Llerena, M. V., and Damián-Carrión, D. A. Land cover classification in an ecuadorian mountain geosystem using a random forest classifier, spectral vegetation indices, and ancillary geographic data. *Geosciences* 7 (2): 34, 2017.

Bala, G., Caldeira, K., Wickett, M., Phillips, T., Lobell, D., Delire, C., and Mirin, A. Combined climate and carbon-cycle effects of large-scale deforestation. *Proceedings of the National Academy of Sciences* 104 (16): 6550–6555, 2007.

Belgiu, M. and Drăguţ, L. Random forest in remote sensing: A review of applications and future directions. *ISPRS Journal of Photogrammetry and Remote Sensing* vol. 114, pp. 24–31, 2016.

Bonan, G. B. Forests and climate change: forcings, feedbacks, and the climate benefits of forests. *Science* 320 (5882): 1444–1449, 2008.

Camara, G., Assis, L. F., Ribeiro, G., Ferreira, K. R., Llapa, E., and Vinhas, L. Big earth observation data analytics. In *Proceedings of the 5th ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data - BigSpatial '16*. ACM Press, 2016.

Dadi, M. M. *Assessing the transferability of random forset and time-weighted dynamic time warping for agriculture mapping*. M.S. thesis, University of Twente, 2019.

Dewan, A. M. and Yamaguchi, Y. Land use and land cover change in greater dhaka, bangladesh: Using remote sensing to promote sustainable urbanization. *Applied geography* 29 (3): 390–401, 2009.

Foody, G. M. Status of land cover classification accuracy assessment. *Remote Sensing of Environment* 80 (1): 185–201, 2002.

Gislason, P. O., Benediktsson, J. A., and Sveinsson, J. R. Random forests for land cover classification. *Pattern Recognition Letters* 27 (4): 294–300, 2006.

Guan, X., Huang, C., Liu, G., Meng, X., and Liu, Q. Mapping rice cropping systems in vietnam using an ndvi-based time-series similarity measurement based on dtw distance. *Remote Sensing* 8 (1): 19, 2016.

Hansen, M. C. and Loveland, T. R. A review of large area monitoring of land cover change using landsat data. *Remote Sensing of Environment* vol. 122, pp. 66–74, July, 2012.

Houborg, R. and McCabe, M. F. A cubesat enabled spatio-temporal enhancement method (CESTEM) utilizing planet, landsat and MODIS data. *Remote Sensing of Environment* vol. 209, pp. 211–226, May, 2018.

Manabe, V. D., Melo, M. R., and Rocha, J. V. Framework for mapping integrated crop-livestock systems in mato grosso, brazil. *Remote Sensing* 10 (9): 1322, 2018.

Maus, V. *Land Use and Land Cover Monitoring Using Remote Sensing Image Time Series*. Ph.D. thesis, PhD thesis, Instituto Nacional de Pesquisas Espaciais, Sao Jose dos Campos, 2016.

Maus, V., Camara, G., Cartaxo, R., Sanchez, A., Ramos, F. M., and De Queiroz, G. R. A time-weighted dynamic time warping method for land-use and land-cover mapping. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 9 (8): 3729–3739, 2016.

Nepstad, D., McGrath, D., Stickler, C., Alencar, A., Azevedo, A., Swette, B., Bezerra, T., DiGiano, M., Shimada, J., da Motta, R. S., et al. Slowing amazon deforestation through public policy and interventions in beef and soy supply chains. *science* 344 (6188): 1118–1123, 2014.

NVIDIA. CUDA C++ Programming Guide. http://docs.nvidia.com/cuda/cuda-c-programming-guide/, 2020.

Oliveira, S. S., Cardoso, M. d. C., Bueno, E., Rodrigues, V. J., and Martins, W. S. Exploiting parallelism to generate meta-features for land use and land cover classification with remote sensing time series. *Brazilian Symposium on Geoinformatics (GeoInfo)*, 2019.

Oliveira, S. S., Pascoal, L. M., Ferreira, L., Cardoso, M. d. C., Bueno, E., Rodrigues, V. J., and Martins, W. S. Sp-twdtw: A new parallel algorithm for spatio-temporal analysis of remote sensing images. *Brazilian Symposium on Geoinformatics (GeoInfo)*, 2018.

Oliveira, S. S. T., Martins, W. S., Sacramento, V., Bueno, E., Cardoso, M., and Pascoal, L. A parallel and distributed approach to the analysis of time series on remote sensing big data. *Journal of Information and Data Management* 10 (1): 16–34, 2019.

Paiva, R. U., Oliveira, S. S., Pascoal, L. M., Parente, L. L., and Martins, W. S. An efficient solution to generate meta-features for classification with remote sensing time series. *Brazilian Symposium on Geoinformatics (GeoInfo)*, 2020.

Pal, M. Random forest classifier for remote sensing classification. *International Journal of Remote Sensing* 26 (1): 217–222, 2005.

Parente, L. and Ferreira, L. Assessing the spatial and occupation dynamics of the brazilian pasturelands based on the automated classification of modis images from 2000 to 2016. *Remote Sensing* 10 (4): 606, 2018.

Parente, L., Mesquita, V., Miziara, F., Baumann, L., and Ferreira, L. Assessing the pasturelands and livestock dynamics in brazil, from 1985 to 2017: A novel approach based on high spatial resolution imagery and google earth engine cloud computing. *Remote Sensing of Environment* vol. 232, pp. 111301, 2019.

Picoli, M. C. A., Camara, G., Sanches, I., Simoes, R., Carvalho, A., Maciel, A., Coutinho, A., Esquerdo, J., Antunes, J., Begotti, R. A., et al. Big earth observation time series analysis for monitoring brazilian agriculture. *ISPRS journal of photogrammetry and remote sensing* vol. 145, pp. 328–339, 2018.

Plaza, A. *High performance computing in remote sensing.* Chapman & Hall/CRC, Boca Raton, FL, 2008.

Rabiner, L., Rosenberg, A., and Levinson, S. Considerations in dynamic time warping algorithms for discrete word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing* 26 (6): 575–582, 1978.

Romani, L. A., Goncalves, R., Zullo, J., Traina, C., and Traina, A. J. New dtw-based method to similarity search in sugar cane regions represented by climate and remote sensing time series. In *2010 IEEE International Geoscience and Remote Sensing Symposium.* IEEE, pp. 355–358, 2010.

Sakoe, H. and Chiba, S. Dynamic programming algorithm optimization for spoken word recognition. *IEEE transactions on acoustics, speech, and signal processing* 26 (1): 43–49, 1978.

Tsai, Y. H., Stow, D., Chen, H. L., Lewison, R., An, L., and Shi, L. Mapping vegetation and land use types in fanjingshan national nature reserve using google earth engine. *Remote Sensing* 10 (6): 927, 2018.