How Do API Selections Affect the Runtime Performance of Data Analytics Tasks?

Yida Tao[1], Shan Tang[1], Yepang Liu[2], Zhiwu Xu[1], Shengchao Qin[3]

[1] Shenzhen University

[2] Southern University of Science and Technology

[3] Teesside University

Motivation

Developers often leverage alternative implementations, which invoke different APIs, yet still produce the same output given the same task inputs, to speed up their data analytics tasks

Motivation

Developers often leverage alternative implementations, which invoke different APIs, yet still produce the same output given the same task inputs, to speed up their data analytics tasks

Iterating over rows of a DataFrame	APIs	Speedup
❶ `[row['a'] for _,row in df.iterrows()]` ❷ `[row.a for row in df.itertuples()]`	`pandas.DataFrame.iterrows` `pandas.DataFrame.itertuples`	88x (1M rows)

Motivation

Developers often leverage alternative implementations, which invoke different APIs, yet still produce the same output given the same task inputs, to speed up their data analytics tasks

Iterating over rows of a DataFrame	APIs	Speedup
❶ `[row['a'] for _,row in df.iterrows()]` ❷ `[row.a for row in df.itertuples()]`	`pandas.DataFrame.iterrows` `pandas.DataFrame.itertuples`	88x (1M rows)

Computing the magnitude of vectors	APIs	Speedup
❶ `[np.linalg.norm(x) for x in a]` ❷ `np.sqrt((a*a).sum(axis=1))`	`numpy.linalg.norm` `numpy.ndarray.sum -> numpy.sqrt`	147x (1M rows)

Focus of this paper

- How can we effectively identify alternative implementations?

Focus of this paper

- How can we effectively identify alternative implementations?

Challenges

- Where to find?
- How to determine?
- How to scale?

Alternative implementations can be extracted from the comparative structures in Stack Overflow discussions

Alternative implementations can be extracted from the comparative structures in Stack Overflow discussions

Nature of comparison

Things that can be compared should have something in common, otherwise the comparison will be meaningless

Alternative implementations can be extracted from the comparative structures in Stack Overflow discussions

Nature of comparison

Things that can be compared should have something in common, otherwise the comparison will be meaningless

Rewarding mechanism of Stack Overflow

Stack Overflow enforces users to post answers that directly address the question. Unconstructive or irrelevant answers might be downvoted or removed

Alternative implementations can be extracted from the comparative structures in Stack Overflow discussions

Nature of comparison

Things that can be compared should have something in common, otherwise the comparison will be meaningless

Rewarding mechanism of Stack Overflow

Stack Overflow enforces users to post answers that directly address the question. Unconstructive or irrelevant answers might be downvoted or removed

If two implementations are compared in an SO answer post, they are very likely to be alternative solutions to the task proposed in the corresponding question post

Alternative implementations can be extracted from the comparative structures in Stack Overflow discussions

1 Consecutive profiling statements in code blocks

Although all of the above return a view, there are some timing differences. So, the preferred way of doing this (for efficiency) would be:

```
In [124]: arr = np.arange(3*3*5).reshape(3, 3, 5)

In [125]: %timeit np.swapaxes(arr, -1, 1)
456 ns ± 6.79 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)

In [126]: %timeit np.transpose(arr, (0, 2, 1))
458 ns ± 6.93 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)

In [127]: %timeit np.reshape(arr, (3, 5, 3))
635 ns ± 9.06 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)

In [128]: %timeit np.moveaxis(arr, -1, 1)
3.42 µs ± 79.6 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

https://stackoverflow.com/questions/56099793/reshaping-data-in-python-list-array/56099854#56099854

Alternative implementations can be extracted from the comparative structures in Stack Overflow discussions

1 Consecutive profiling statements in code blocks

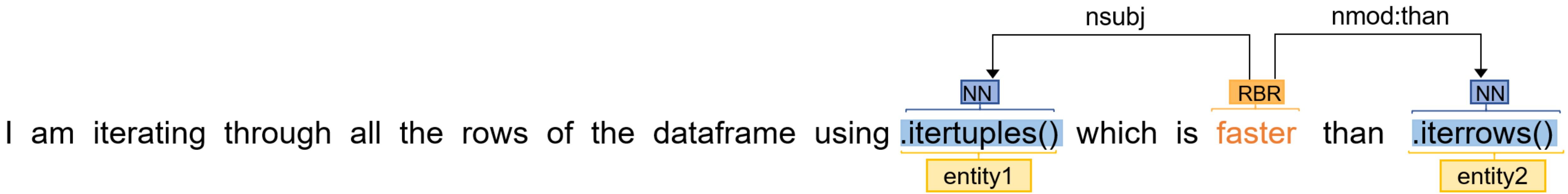
Although all of the above return a view, there are some timing differences. So, the preferred way of doing this (for efficiency) would be:

```
In [124]: arr = np.arange(3*3*5).reshape(3, 3, 5)

In [125]: %timeit np.swapaxes(arr, -1, 1)
456 ns ± 6.79 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)

In [126]: %timeit np.transpose(arr, (0, 2, 1))
458 ns ± 6.93 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)

In [127]: %timeit np.reshape(arr, (3, 5, 3))
635 ns ± 9.06 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)

In [128]: %timeit np.moveaxis(arr, -1, 1)
3.42 µs ± 79.6 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

https://stackoverflow.com/questions/56099793/reshaping-data-in-python-list-array/56099854#56099854

Alternative implementations can be discovered from the comparative structures in Stack Overflow discussions

2 Comparative sentences from natural-language text



Evaluation

- **Dataset**: Stack Overflow threads tagged with *NumPy*, *Pandas* and *SciPy*.

Evaluation

- **Dataset**: Stack Overflow threads tagged with *NumPy*, *Pandas* and *SciPy*.
- **Validation**: programmatically verify the task-specific input/output equivalence of the extracted candidate implementation pairs

```
import numpy as np
arr = np.array([['B', 'B', 'B', 'A', 'B’,
               'A', 'A', 'A', 'B', 'A’]])

o1 = np.sum(arr == 'A')

o2 = np.count_nonzero(arr == 'A’)

assert o1 == o2
```

Extracted implementation pair

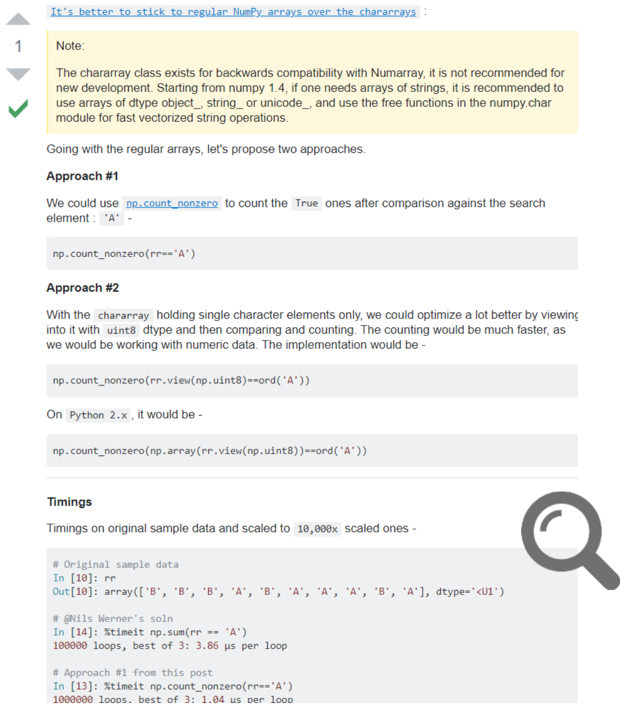
Evaluation

- **Dataset**: Stack Overflow threads tagged with *NumPy*, *Pandas* and *SciPy*.
- **Validation**: programmatically verify the task-specific input/output equivalence of the extracted candidate implementation pairs

```
import numpy as np
arr = np.array([['B', 'B', 'B', 'A', 'B’,
                 'A', 'A', 'A', 'B', 'A’]])


o1 = np.sum(arr == 'A')

o2 = np.count_nonzero(arr == 'A’)


assert o1 == o2
```

Instantiating the task input by inspecting the SO post

Extracted implementation pair


1

It's better to stick to regular NumPy arrays over the chararrays :

Note:

The chararray class exists for backwards compatibility with Numarray, it is not recommended for new development. Starting from numpy 1.4, if one needs arrays of strings, it is recommended to use arrays of dtype object_, string_ or unicode_, and use the free functions in the numpy.char module for fast vectorized string operations.

Going with the regular arrays, let's propose two approaches.

Approach #1

We could use `np.count_nonzero` to count the `True` ones after comparison against the search element : `'A'` -

```
np.count_nonzero(rr=='A')
```

Approach #2

With the `chararray` holding single character elements only, we could optimize a lot better by viewing into it with `uint8` dtype and then comparing and counting. The counting would be much faster, as we would be working with numeric data. The implementation would be -

```
np.count_nonzero(rr.view(np.uint8)==ord('A'))
```

On `Python 2.x`, it would be -

```
np.count_nonzero(np.array(rr.view(np.uint8))==ord('A'))
```

Timings

Timings on original sample data and scaled to `10,000x` scaled ones -

```
# Original sample data
In [10]: rr
Out[10]: array(['B', 'B', 'B', 'A', 'B', 'A', 'A', 'A', 'B', 'A'], dtype='<U1')

# @Nils Werner's soln
In [14]: %timeit np.sum(rr == 'A')
100000 loops, best of 3: 3.86 µs per loop

# Approach #1 from this post
In [13]: %timeit np.count_nonzero(rr=='A')
1000000 loops, best of 3: 1.04 µs per loop
```

Evaluation

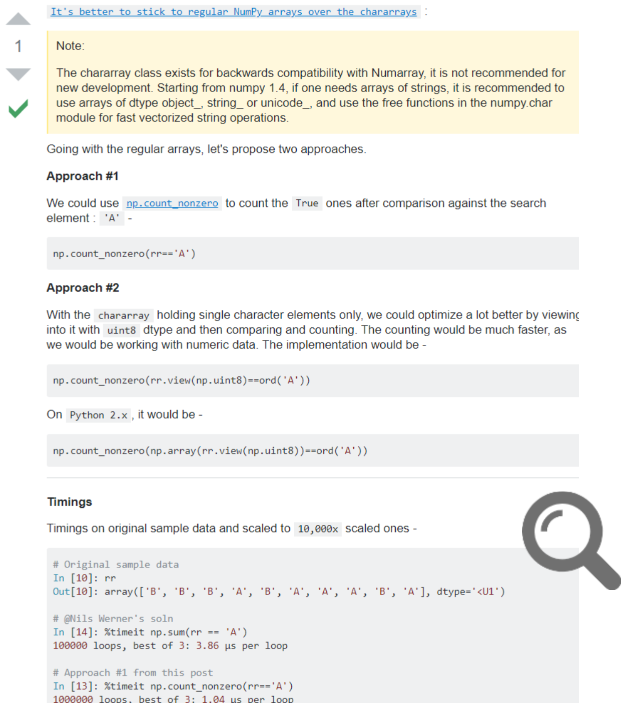
- **Dataset**: Stack Overflow threads tagged with *NumPy*, *Pandas* and *SciPy*.
- **Validation**: programmatically verify the task-specific input/output equivalence of the extracted candidate implementation pairs

```
import numpy as np
arr = np.array([['B', 'B', 'B', 'A', 'B’,
                'A', 'A', 'A', 'B', 'A’]])

o1 = np.sum(arr == 'A')

o2 = np.count_nonzero(arr == 'A’)

assert o1 == o2
```

Instantiating the task input by inspecting the SO post

Extracted implementation pair

Verify the output equivalence



It's better to stick to regular NumPy arrays over the chararrays :

1

Note:

The chararray class exists for backwards compatibility with Numarray, it is not recommended for new development. Starting from numpy 1.4, if one needs arrays of strings, it is recommended to use arrays of dtype object_, string_ or unicode_, and use the free functions in the numpy.char module for fast vectorized string operations.

Going with the regular arrays, let's propose two approaches.

Approach #1

We could use `np.count_nonzero` to count the `True` ones after comparison against the search element : `'A'` -

```
np.count_nonzero(rr=='A')
```

Approach #2

With the `chararray` holding single character elements only, we could optimize a lot better by viewing into it with `uint8` dtype and then comparing and counting. The counting would be much faster, as we would be working with numeric data. The implementation would be -

```
np.count_nonzero(rr.view(np.uint8)==ord('A'))
```

On `Python 2.x` , it would be -

```
np.count_nonzero(np.array(rr.view(np.uint8))==ord('A'))
```

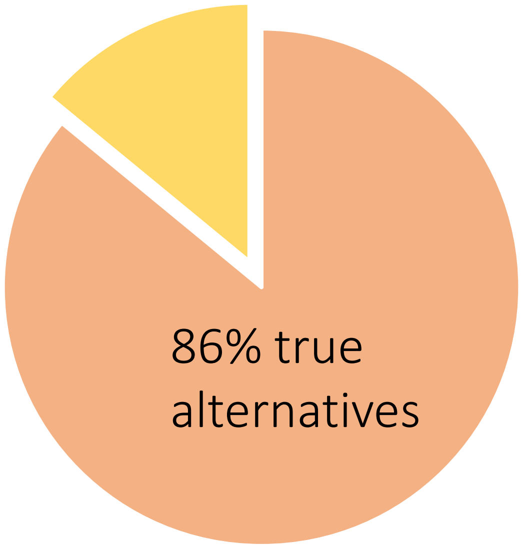
Timings

Timings on original sample data and scaled to `10,000x` scaled ones -

```
# Original sample data
In [10]: rr
Out[10]: array(['B', 'B', 'B', 'A', 'B', 'A', 'A', 'A', 'B', 'A'], dtype='<U1')

# @Nils Werner's soln
In [14]: %timeit np.sum(rr == 'A')
100000 loops, best of 3: 3.86 µs per loop

# Approach #1 from this post
In [13]: %timeit np.count_nonzero(rr=='A')
1000000 loops, best of 3: 1.04 µs per loop
```

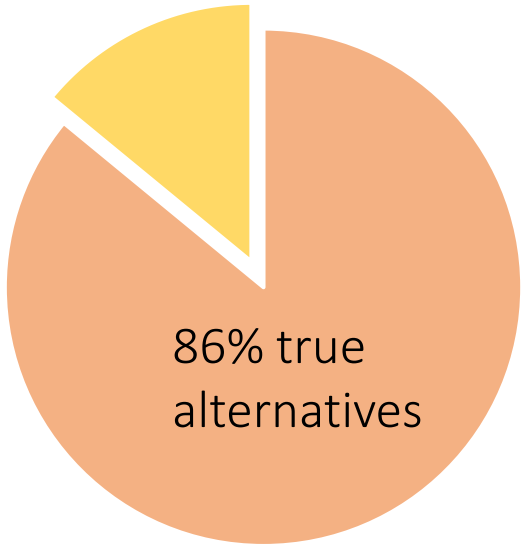
Results

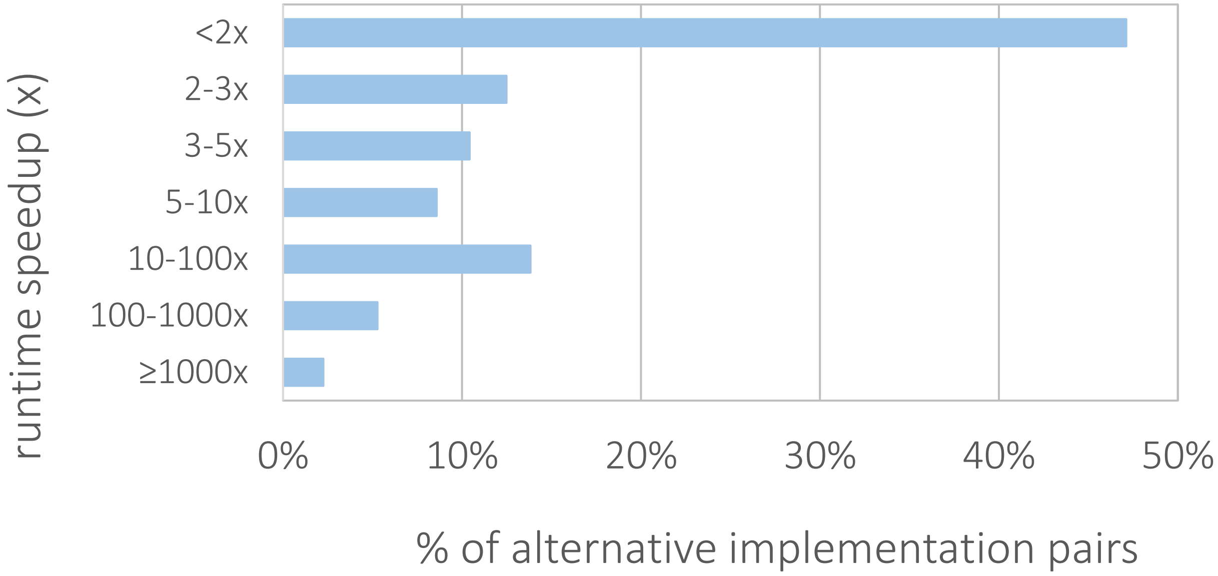
- We extracted 5575 candidate pairs, 4786 (86%) are validated as true alternative implementations



Results

- We extracted 5575 candidate pairs, 4786 (86%) are validated as true alternative implementations
- Alternative implementations do improve task runtime performance, and sometimes the improvement is quite significant.





Future Directions

- The idea of exploiting comparative structures to reveal programming alternatives should also be applicable to other libraries and other programming languages

- The approach provides a new perspective for API recommendation and performance optimization