analytics

MDPI

Article

occams: A Text Summarization Package

Clinton T. White [1], Neil P. Molino [2], Julia S. Yang [1] and John M. Conroy [2,*]

[1] Department of Defense, USA; ctwhit2@super.org (C.T.W.); jsyang@super.org (J.S.Y.)
[2] IDA Center for Computing Sciences, 17100 Science Drive, Bowie, MD 20740, USA; npmolin@super.org
* Correspondence: conroy@super.org

Abstract: Extractive text summarization selects a small subset of sentences from a document, which gives good "coverage" of a document. When given a set of term weights indicating the importance of the terms, the concept of coverage may be formalized into a combinatorial optimization problem known as the budgeted maximum coverage problem. Extractive methods in this class are known to be among the best of classic extractive summarization systems. This paper gives a synopsis of the software package `occams`, which is a multilingual extractive single and multi-document summarization package based on an algorithm giving an optimal approximation to the budgeted maximum coverage problem. The `occams` package is written in Python and provides an easy-to-use modular interface, allowing it to work in conjunction with popular Python NLP packages, such as `nltk`, `stanza` or `spacy`.

Keywords: text summarization; extractive; multilingual; budgeted maximum coverage

Citation: White, C.T.; Molino, N.P.; Yang, J.S.; Conroy, J.M. `occams`: A Text Summarization Package. *Analytics* **2022**, *2*, 546–559. https://doi.org/10.3390/analytics2030030

Academic Editor: R. Jordan Crouser

Received: 31 January 2023
Revised: 6 May 2023
Accepted: 15 June 2023
Published: 30 June 2023

1. Introduction

Text summaries give a reader an overview of a document or a collection of documents on the same topic. Summaries are a convenient tool to give an *indication* of the overall content and may also *inform* the reader of details of interest to them. The need for quality summaries has grown with the dramatic rise of information. Many individual documents have good-quality summaries; however, these summaries may not focus on the specific information of interest to a user.

There are two general approaches to address a user's needs for a summary of one or more documents: *extractive* and *abstractive*. Extractive methods primarily seek to find good sentences from the source documents that can be concatenated to form a summary; abstractive methods may use words and phrases that do not appear in the original documents and try to present them meaningfully for a more human-level summary. In this paper, we present the software package `occams`, an extractive summarization system using a combinatorial optimization approach whose performance is demonstrated to be among the best extractive methods [1]. For a survey of the summarization problem and classical extractive summarization methods, see [2,3].

Until recently, abstractive summarization methods lagged behind extractive methods, as systems could not generate fluent context containing relevant information. One of the first successful approaches used a recurrent neural network (RNN) along with a *pointer generating network* [4]. RNNs were used to encode a document, turning the sequence of token embeddings into a state vector. Conversely, *decoding* takes the state vector and converts it into a sequence of tokens, forming the summary. This approach cleverly augmented the decoding of the RNN-encoded document with a "copy mechanism" to include extracted words or even segments of the document.

With the invention of the transformer model [5] later that year, another big change came about. The transformer model then led to deep pre-trained language models, such as generative pre-trained transformers (GPT) [6] and bidirectional encoder representations from transformers (BERT) [7], both of which improved the performance on a range of

natural language processing tasks, including recognizing relevant content to generate summaries. When these models are coupled as was done in bidirectional and auto-regressive transformers (BART) [8], sequence-to-sequence models were trained to learn to transform a document into an abstractive summary. Over the last three years, the models have grown in size and scope for text summarization and other natural language processing tasks, including the introduction of sophisticated chatbots, such as ChatGPT [9], which have the ability to answer questions, summarize, and more generally generate text based on a prompt (for example, a prompt could be "Summarize the following text with 100 words or less" https://openai.com/blog/chatgpt, accessed on 8 December 2022).

Despite these great advances, there is still a need for classic extractive summarization. Abstractive methods suffer from "hallucinations", factual errors or statements not supported or readily explained by the original text [10]. Secondly, large language models can require great computing resources, and most have limited input sizes. Due to these concerns, an extractive summarization system can provide an alternative or be used in conjunction with an abstractive system as done by others, e.g., [11].

The primary goal of `occams` is to provide a state-of-the-art multilingual extractive summarization method using the first principles of the statistics of natural language in conjunction with an optimal approximation of a combinatorial covering algorithm. The covering problem approximates the integer linear programming problem formulation [12]. The previous implementation of the approximation algorithm was reported [13] to give over 90% of the optimal coverage with an order of magnitude speed up. The current implementation maintains this coverage approximation and has improved its performance.

The system's name evokes the principle of Occam's razor, which can be summarized as "entities should not be multiplied beyond necessity". The principle is attributed to William Ockham (Occam), a 13th-century English Franciscan friar and philosopher (although this statement of principle occurs later, and the principle itself could be argued to date back to Aristotle [14]).

We apply this principle first to aim toward an extractive versus abstractive summarization. While neural-based summarization systems have produced great advances in text summarization, and human-quality abstractive summaries would be preferred, in many cases, a simple extractive method suffices. In addition to being computationally cheaper, extractive systems often benefit from greater flexibility and control of input and output lengths. They can be applied to various multi-document summarization tasks across various document genres. As mentioned, abstractive methods suffer from hallucinations and present a challenge to explain their output. In contrast, an extractive method, by definition, selects a subset of sentences from the document, so it is easy to find the source of information used in the generated summary.

The `occams` package decomposes the extractive summarization task into three steps.

1. The input document or documents (the input text) is segmented into sentences and terms.
2. Term weights are computed, indicating the relative importance of the terms in the input text.
3. Given a target summary length in characters or words, an optimal approximation algorithm is used to maximize the weighted coverage of terms by selecting a subset of sentences, the sum of whose lengths does not exceed the budgeted target length.

The paper is divided into three main sections according to these three parts. We then give a short example demonstrating how to use the package to extract a summary, and conclude with results compared to other summarization systems.

2. Document Segmentation

Document segmentation is a critical part of natural language processing. The quality of the resulting downstream tasks, summarization included, is affected by this first step. Document segmentation for extractive summarization consists of splitting the text of a

document into sentences and tokenizing those sentences into smaller units of meaning, typically words. This process is language dependent.

As we will see in more detail in Section 3, `occams` views a document as a bipartite graph on the sets of sentences and *terms*, with adjacency indicating that a given term occurs in a given sentence. Terms are user specified and can represent more abstract notions, such as concepts, as originally proposed by [15]. Typically, we form terms as unigrams or bigrams of tokens, where tokens consist of words, word stems, or word lemmas. One might also want to apply some kind of text normalization, such as collapsing consecutive whitespace characters to a single ASCII space, removing punctuation, removing digits, or lowercasing characters. There is quite a bit of choice in this process.

Included in `occams` is the subpackage `occams.nlp`, which deals with these problems. This subpackage serves two roles for users. First, it defines an abstraction in the form of the two Python dataclasses `Sentence` and `Document`. These dataclasses are a means of bundling the result of document segmentation such that it is independent of any particular NLP library's API. `Sentence` has two attributes, `text` and `terms`; the former, a string, is the literal text of the sentence, while the latter is a list of the terms which appear in that sentence. `Document` also has two attributes: `sentences` is a list of `Sentence` objects, and `language` is a string indicating the language of the document. Thus, a `Document` object encodes the bipartite graph resulting from segmenting the text of a document. The summarizer class, which we will discuss in Section 3, takes as input a list of `Document` objects. In this way, we decouple the APIs of NLP libraries that may be used for the document segmentation step from the API of the summarizer class, giving users the flexibility to use whatever tools they wish for the document segmentation step without requiring those tools to be directly supported by `occams`.

On the other hand, as we said above, there is quite a bit of choice involved in segmenting documents, from the choice of a library, to what constitutes a token, to whether or how to normalize the text. The second role of the `occams.nlp` subpackage is to alleviate some of this burden for users who would prefer a more convenient and streamlined process. We provide a simple `DocumentProcessor` class, which can be used to transform the text into `Document` objects. By default, this class uses `nltk` [16] for document segmentation: `nltk.tokenize.sent_tokenize()` to split the document into sentences, `nltk.tokenize.word_tokenize()` to form word tokens, and one of `nltk`'s Porter or Snowball stemmers to produce stemmed word tokens. We choose `nltk`, as it is the most widely used Python package for natural language processing, it is computationally efficient, and gives strong support for most European languages. To provide convenient support for a broader set of languages, we also include a class that uses `stanza` [17] for document segmentation, as `stanza` supports 70 languages.

These classes have a few options to allow users to control some of the aspects of the document segmentation noted above. However, they are also meant to be somewhat opinionated about the process to save users from having to make too many choices. Users with their own opinions can quite easily segment documents however they wish and bundle the results into `Sentence` and `Document` objects. Finally, `occams.nlp` contains a helper function, `process_document()`, that instantiates a `DocumentProcessor` object with its default options and uses it to process text into a `Document` object. We demonstrate the user of this wrapper in Section 5.

Now that our text, which may consist of one or more documents, is segmented into sentences and terms, we can now formulate the text summarization problem as a combinatorial optimization problem.

3. Extractive Summarization as a Combinatorial Optimization Problem

The mathematical formulation of the problem used in `occams` is adapted from [12,15]. We let $A = (a_{ij})$ be the term-sentence incidence matrix of size m by s encoding the relationship among all of the terms and sentences found in the documents to be summarized so that $a_{ij} = 1$ if and only if term i appears in sentence j, and 0 otherwise.

The objective function of the optimization problem credits each unique term a sentence that contributes to the summary based on its term weight, as discussed in Section 4, which we denote as $w_i^{(\tau)}$ and which represents the relative importance of term i for the topic τ of the set of documents to be summarized. The constraint for the optimization problem is the target length L of the summary. Let ℓ_j denote the length of sentence j in whatever units are desired, e.g., words or characters.

The optimization problem can be formalized as follows:

$$y^* = \text{argmax}_y \sum_i y_i w_i^{(\tau)}$$

subject to

$$\sum_j x_j \ell_j \leq L; \tag{1}$$

$$y_i - \sum_j a_{ij} x_j \leq 0; \tag{2}$$

$$a_{ij} x_j - y_i \leq 0; \tag{3}$$

$$y_i \in \{0,1\},\ x_j \in \{0,1\}. \tag{4}$$

Inequality (1) says that the sum of the lengths of the sentences selected for the summary does not exceed the target length L, while inequalities (2)–(4) together say that the set of terms "covered" by the summary is precisely the union of the sets of terms, which occurs in the sentences selected for the summary.

The above formulation is an NP-hard problem [15]. Still, this binary integer programming problem can be viewed as a budgeted maximum coverage problem, and solved with provably optimal approximate algorithms [15,18,19] or treated as a problem in the more general class of sub-modular approximation algorithms [20]. These algorithms are *optimal*, in that they are guaranteed to achieve an optimal fraction of the best solution. The `occams` package uses an optimal approximation algorithm for the budgeted maximum coverage problem and a dynamic programming algorithm for the knapsack problem. These methods were shown empirically to exceed the guaranteed lower bound of approximation ($1 - 1/\sqrt{e} \approx 0.39$) and achieve about 94% accuracy [13] and are about 14 times faster than the corresponding integer programming algorithm. The new implementation, which we present in this paper, achieves or exceeds the same accuracy and has both improved runtime and memory requirements. (See Appendix A for an example comparing the former implementation and the current one.)

The algorithms for solving the budgeted maximum coverage problem and the knapsack problem have been implemented in `occams` as a Python extension module for better performance. Originally, these were written in C and Cython, but the extension module was recently rewritten in the Rust programming language. We choose Rust because it is a modern, memory-safe systems programming language that compiles to native machine code and achieves comparable speed and memory usage to C. Users of `occams` benefit from the speed and memory safety of Rust while retaining a familiar Python API.

The algorithm employed for solving the budgeted maximum coverage problem is a modified greedy algorithm given in [18], which is more efficient. In [18], each step of the algorithm greedily finds the sentence from those remaining, which maximizes the normalized marginal weight (the sum of the weights of the new terms it would add to the solution divided by its cost). If this optimal sentence is affordable with what remains of the budget, it is added to the solution; otherwise, it is thrown out. This process continues until every sentence has either been selected for the solution or thrown away. This requires s iterations, where the i-th iteration maximizes the normalized marginal weight over the remaining $s - i$ sentences.

Our modification reverses the order in which weight and cost are considered. Rather than maximizing the normalized marginal weight over *all* remaining sentences, we only consider affordable sentences with what remains of the budget. Our algorithm terminates when no more sentences can be afforded with the remaining budget and adds new terms not already in the solution. This requires at most L steps since the length of the sentence is an integer and at least 1. In practice, typical sentence lengths measured in words or characters well exceed 1. Our implementation of the greedy algorithm for the budgeted maximum coverage problem requires space of order $O(n)$ since the incidence matrix is stored as a sparse matrix and has a running time that is at most $O(nL)$, where n is the number of non zeros in the term–sentence matrix and L is the budgeted length for the summary.

As well as the budgeted maximum coverage problem, `occams` also models the sentence selection problem as a knapsack problem; see [19] for details on the correspondence. However, our new Rust extension module has an improved implementation of a dynamic programming algorithm for the knapsack problem from that given in the reference and implemented in the original C extension module. Items and costs index the dynamic programming table, whereas previously, it was indexed by items and profits. Using costs instead of profits to index one dimension of the table has two advantages. First, by the very nature of the summarization problem, the maximum total knapsack cost, which is the budgeted summary length, is inherently small. Previously, our knapsack algorithm suffered from requiring space and time of order $O(s^2 P_{\max})$, where $P_{\max}$ is the maximum sum of the weights of the terms of a sentence. Our new implementation requires space and time of order $O(sL)$, which is typically much smaller and, importantly, independent of the scaling of the term weights. Second, in the summarization problem, the costs are positive integers, whereas profits are floating point values. Indexing the table with profits requires approximating the profits with integers.

Users of `occams` do not directly interact with the extension module and its functions implementing the above algorithms. Instead, the `occams` API provides a high-level interface to these algorithms in the form of the `SummaryExtractor` class. This class is responsible for collecting the required inputs for the summarization problem, converting those inputs to the more abstract formulation used in the combinatorial optimization algorithms, interacting with the extension module implementing those algorithms, and finally converting the results back into concrete, natural language constructs to produce the desired summary. The input to create an instance of `SummaryExtractor` is primarily the data discussed in Sections 2 and 4, namely, (a) documents segmented into sentences and terms in the form of `Document` objects, and (b) term weights.

In addition to the base `SummaryExtractor` class, `occams` provides several subclass specializations, each with its baked-in notion of how to compute term weights. These `WeightedSummaryExtractors` simplify the setup for a user by computing term weights automatically from the segmented documents the user provides. We discuss one of these subclasses, `TermFrequencySummaryExtractor`, in Section 4.

4. Term Weighting Methods

The linear objective function of the `occams` algorithm is the sum of the weights of the terms covered by an extract summary. These term weights are critical to the quality of the summary produced. The `occams` package provides a number of different ways of computing term weights. Some of these depend only on the counts of the number of sentences or documents containing each of the terms. Such term weight methods are collected as members of a Python `enum` class called `TermFrequencyScheme`, described in more detail below. However, we emphasize that users are not limited to these provided schemes for computing term weights. In the end, users may assign term weights however they wish and simply hand these term weights to `SummaryExtractor`, the summarizer base class, in the form of a dictionary mapping the terms to their weights.

The following are members of the `TermFrequencyScheme` enumeration:

1. `LOG_COUNTS`: The logarithm of the Laplace smoothed number of occurrences of term i; $w_i = \log(1 + \sum_j a_{ij})$, where a_{ij} is 1 if term i occurs in sentence j and 0 otherwise.
2. `ENTROPY`: The scaled smoothed entropy over the sentences; $w_i = c_i \log(1 + c_i)$, where $c_i = \sum_j a_{ij}$.
3. `POSITIONAL_FIRST`: A variation of `LOG_COUNTS` which gives double weight to the first sentence in each document, as inspired by [12]. Formally, $w_i = \log(1 + \sum_j a_{ij} + \sum_{j \in \alpha} a_{ij})$ where α is the set of first sentences in the documents.
4. `POSITIONAL_DENSE`: A variation of `POSITIONAL_FIRST` which replaces the use of the first sentence with the first sentence above the median score for the document. Formally, $w_i = \log(1 + \sum_j a_{ij} + \sum_{j \in \beta} a_{ij})$ where β is the set of positional dense sentences described below.
5. `POSITIONAL_MEAN`: The mean count of the number of times a term occurs in the list of documents to be summarized. This scheme is inspired by the work of [13].
6. `POSITIONAL_MAX`: The maximum count of the number of times a term occurs in the list of documents to be summarized.
7. `POSITIONAL_MIN`: The minimum count of the number of times a term occurs in the list of documents to be summarized.
8. `CORE_TERMS`: The principle left singular vector of the term-sentence matrix. Assuming A is irreducible, we let u be the left principal singular value of A. u can be chosen to be non-negative and used to form term weights. The matrix A can be assured to be irreducible by adding a small constant, currently set to 0.01, to each entry. This scheme, as well as the next, are inspired by the success of [21,22] as well as the recent paper [23], which gives a theoretical justification. Specifically, spectral clustering of a graph's adjacency matrix will tend to expose the "core–periphery" structure. Computing the left singular vectors of A will then tend to partition the graph to separate the key terms from the least essential terms. A scale of this singular vector is added to the overall term counts, and the logarithm of the entries is used for term weights.
9. `CORE_SENTENCE`: The principle right singular vector of the term-sentence matrix. Assuming A is irreducible, we let v be the right principal singular value of A. v can be chosen to be non-negative and used to form term weights. The entries of v are used as sentence weights. Sentences above the median value are marked as "core" and contribute to a log count term weight. The sentence weights are converted to term weights by computing the matrix–vector product $w = Av$.

Term weights may be optionally scaled, via an affine transformation, to be between 1 and 100.

We now give more detail and motivation of the `POSITIONAL_DENSE` scheme. The goal of this term weight, which is the default for `occams`, is to produce a robust method that works at least as well as `POSITIONAL_FIRST` and will avoid giving double weight to the first sentence in a document if it has little content. Such low-content sentences in the first position may arise from an error in sentence splitting or a failure to remove a dateline, byline, or boilerplate sentence.

The relative importance of a sentence is given by its per-term log-likelihood score. A sentence is considered "positional dense" if it is the first sentence in a document whose score is above a specified quantile, currently set to 0.1. If all the sentences in a document are below the threshold, then no sentence from this document is chosen. (Such a situation only occurs for multi-document summarization.) If we let p_i be the maximum likelihood estimated probability of term i, then the average likelihood score for a sentence is given by

$$o(j) = \frac{1}{n_j} \sum_i \log(m p_i) a_{ij},$$

where m is the number of terms in the list of documents and $n_j = \sum_i a_{ij}$ is the number of terms in sentence j.

5. A Simple Example

Before we give an evaluation of `occams` on a few common summarization datasets, let us illustrate the software with a simple example. We will use the abstract of this paper as the text to be summarized, about 130 words, and generate an extractive summary of at most 50 words. The code below generates such a summary.

```
from occams.nlp import TermOrder, process_document
from occams.summarize import (
    SummaryUnits,
    TermFrequencyScheme,
    extract_summary,
)

# Set the text to be the abstract for this paper.
text = "Extractive text summarization selects a small subset ..."

# Parse the document: split into sentences, tokenize and stem words,
# and form terms as bigrams.  This will use nltk for segmentation,
# tokenization, and stemming.
document = process_document(text, TermOrder.BIGRAMS, language="english")

# Summarize the document in at most 50 words, using POSITIONAL_DENSE
# term weights.
extract = extract_summary(
    documents=[document],
    budget=50,
    units=SummaryUnits.WORDS,
    scheme=TermFrequencyScheme.POSITIONAL_DENSE,
)

# Print the summary.
print(extract.summary())
```

The first step is to parse the document, a process which (a) segments the document as a list of sentences; (b) word tokenizes each sentence and stems the word tokens; and (c) forms bigrams as overlapping pairs of tokens. This information is collected into the dataclasses `Document` and `Sentence`, which we described in Section 2. For example, the list of terms in the first sentence is given as follows:

```
[('extract', 'text'), ('text', 'summar'), ('summar', 'select'),
('select', 'a'), ('a', 'small'), ('small', 'subset'), ('subset', 'of'),
('of', 'sentenc'), ('sentenc', 'from'), ('from', 'a'), ('a', 'document'),
('document', 'which'), ('which', 'give'), ('give', 'a'), ('a', 'good'),
('good', 'coverag'), ('coverag', 'of'), ('of', 'a'), ('a', 'document')].
```

Note that bigrams are denoted as 2-tuples of Porter stemmed and lower-cased tokens, which is the default.

Once the document is prepared, we can create an extract of the document by calling the `extract_summary()` function. Note that `extract_summary()` takes a list of documents as input. Here, we give it a list with only one element, but more generally, a list of any number of documents may be provided to compute a multi-document summary. We ask for term weights to be computed with the `POSITIONAL_DENSE` scheme (which is the default). This method of computing term weights gives double weight to terms in the first sentence in each document, which is *dense* in the sense that its log probability score exceeds the bottom quantile of 10% (the quantile is computed by `numpy.quantile`, which uses linear interpolation to estimate the quantile by default). The resulting `Extract` object contains various information about the sentences that were selected. Calling its `summary()` method returns those sentences concatenated to form a single string.

For our example, the resulting summary includes the first sentence and one additional sentence, and has a combined total length exactly matching our budget of 50 words:

> Extractive text summarization selects a small subset of sentences from a document which give a good "coverage" of a document. The `occams` package is written in

Python and provides an easy-to-use, modular interface, allowing it to work in conjunction with popular Python NLP packages, such as `nltk`, `stanza` or `spacy`.

This summary, while not fluent, does give a good indication of the content of the abstract.

The package also includes a command line interface. Assuming the text we wish to summarize is in the file `input.txt`, we can reproduce the results of the example above by running the `occams` command from a shell.

The CLI offers many of the options available in the Python package. See Appendix B for the full usage of the program.

6. ROUGE Evaluation of Summaries

The `occams` package employs `py_rouge` (https://github.com/Diego999/py-rouge, accessed on 8 December 2022), a Python implementation of the ROUGE metric for the evaluation of summaries with English language data. We also provide support for multilingual summary evaluation. This is accomplished by processing the summaries with the same `DocumentProcessor` class and then computing a ROUGE score based on the overlap of token n-grams in the machine and human summaries. In this section, we illustrate the performance of the `occams` summarizer on several classic summarization datasets.

Table 1 gives results on the Document Understanding Conference 2004 task 2 using `occams` with the default term weighting scheme, `POSITIONAL_DENSE`, (labeled `occams`) and compares it to top systems, including the previous version of OCCAMS (labeled `OCCAMS_V`) as reported in [1]. The results demonstrate that `occams` produces summaries among the best scoring on this widely studied dataset, giving the highest ROUGE-2 score.

Table 1. Document Understanding Conference (DUC) 2004 task 2 results. Bold values indicate the best (highest) scoring systems.

System	R1-Recall	R2-Recall
LexRank	35.95	7.47
Greedy-KL	37.98	8.53
CLASSY-04	37.62	8.96
DPP+	**39.79**	9.62
OCCAMS_V	38.50	9.76
ICSISumm	38.41	9.78
`occams`	38.50	**10.24**

While the DUC 2004 automatic evaluation is noteworthy, it is instructive to note that when human summaries are compared, their summaries bracket all but two of the systems listed. For ROUGE-2, the lowest-scoring human summary has a score of about 8.6 and the highest about 10.7. While ROUGE correlates well with content coverage of a summary, it is only a surrogate for human evaluation. It is helpful to illustrate two summaries, one generated from the `occams` and one from an abstractive summarization system, ChatGPT (text generated by ChatGPT, 21 December 2022, OpenAI, https://chat.openai.com/chat: the prompt was "summarize the following text", and then when requested, the text was pasted into the chat). Here, we have both systems summarize document set d30010t from DUC 2004, a set of 10 documents on the Israeli–Palestinian peace negotiations, the Wye River Accord.

`occams`:
A defiant Prime Minister Benjamin Netanyahu said on Saturday that Israel would continue to build Jewish neighborhoods throughout Jerusalem, including at a controversial site in the traditionally Arab sector of the city. An Islamic Jihad official in the Syrian capital of Damascus

ChatGPT:
The Israeli Cabinet has announced that it will postpone a vote on whether to ratify the Wye River Accord until the Palestinian Authority cracks down further on terrorism. The Cabinet also insisted that Yasser Arafat convene the Palestinian National Council to abrogate the Palestinian

confirmed that the group's leader Ramadan Abdullah Shallah claimed responsibility for Friday's bombing in an interview with the Paris-based Radio Monte Carlo. The radical group Islamic Jihad claimed responsibility. The militant Palestinian movement Islamic Holy War said Saturday that it carried out the suicide bombing in a Jerusalem market on Friday, which prompted arrests by the Palestinian Authority overnight.

National Charter, which calls for Israel's destruction. The decision follows a market bombing in Jerusalem in which two people were killed and 21 were wounded. Hamas, which has tried to stop the peace agreement, claimed responsibility for the attack. In response, the Israeli Cabinet suspended ratification of the Wye agreement until "there is verification that the Palestinian authority is indeed fighting terrorism", and announced plans to build houses in the controversial Jewish settlement of Har Homa in east Jerusalem.

We make a few observations about the two automatically generated summaries. First, both contain good content and are informative. However, the ChatGPT abstractive summary flows more naturally than the `occams` extractive counterpart. The `occams` summary satisfies the specified word limit of 100 words, whereas the ChatGPT summary uses 120 words. When the prompt was changed ("please limit the summary size to 100 words") to specify that the length should be less than 100 words, ChatGPT generated 80 words. This is an example of length being more difficult to control with a neural system. On the other hand, abstracts will generally be more concise than extracts.

Finally, we note that there are two hallucinations in the ChatGPT summary. First, the summary gives a partial quote:

> In response, the Israeli Cabinet suspended ratification of the Wye agreement until "there is verification that the Palestinian authority is indeed fighting terrorism", ...

While this reads well and is largely consistent with the text, the word "verification" does not appear in any quotes from the document set. The closest matches, found by reading the documents, are the following:

- APW19981106.0520: "The government of Israel will resume the discussion of the agreement after it verifies that the Palestinian Authority is taking vigorous steps for a relentless fight against terrorist organizations and their infrastructure", a Cabinet statement said.
- APW19981106.0572: The Cabinet said in a statement that it will only reconvene after "it verifies that the Palestinian Authority takes vigorous steps for an all-out war against terrorist organizations and their infrastructure".

Secondly, the ChatGPT summary says the following:

> The Cabinet also insisted that Yasser Arafat convene the Palestinian National Council to abrogate the Palestinian National Charter, which calls for Israel's destruction.

However, the original text states the following:

- APW19981106.0520: The Cabinet also demanded clarifications from Palestinian leader Yasser Arafat on the procedure for removing sections of the PLO charter calling for Israel's destruction.
- APW19981106.0572: The Cabinet also demanded clarifications from Palestinian leader Yasser Arafat on the procedure for revoking clauses in the PLO founding charter calling for Israel's destruction.

We note that *abrogating* (repealing) the entire charter is much stronger language than removing specific sections of the charter. Additionally, demanding clarifications about *procedures* differs from insisting that a *council* convene.

The second dataset we include is the Multi-News dataset [24]. The default term weighting, `POSITIONAL_DENSE`, holds up well against the neural abstractive systems as shown in Table 2, achieving the third highest ROUGE-2-F1 score as of the writing of this paper [25].

(These ROUGE-F1 scores were computed using the ROUGE package provided by `huggingface` (https://huggingface.co/spaces/evaluate-metric/rouge, accessed on 8 December 2022), as `occams`' automatic evaluation is designed to use only ROUGE-R (recall), and summaries are limited to a bound length, which parallels the `occams` summarization design, in keeping with the "Recall Oriented [Understudy for Gisting Evaluation]" origin of this metric). The alternative approaches are supervised methods; tuning parameters of the currently supported term weighting methods or full supervised term weights for `occams` are a logical next step.

Table 2. Multi-news results.

System	R1-F1	R2-F1
PRIMER	49.9	21.1
LongT5	48.2	19.4
`occams`	44.8	16.0
CTF+DPP	45.8	15.9

We conclude with an example of `occams`' support for multilingual summarization as discussed in Section 2 via `nltk` [16], `ersatz` [26], and `stanza` [17]. The performance is illustrated using the default term weighting on the MultiLing 2015 multi-document summarization evaluation (MMS 2015) [27] dataset. For both the `stanza` and `ersatz` options, data are sentence split and tokenized. Terms are formed as bigrams of word lemmas or stems, and the `POSITIONAL_DENSE` term weighting scheme is employed. Note that for the `ersatz` approach, this multilingual sentence splitter is followed by `nltk` for tokenization. Of the 10 languages in the MMS 2015 dataset, `nltk` supports English, French, and Spanish. For the remaining languages, tokenization is achieved using `nltk` English tokenization. In Table 3, we give the ROUGE-2 Recall results for human-to-human as well as the system from West Bohemia University (WBU) [28] from MMS 2015 for comparison with `occams`. WBU was the highest scoring system in 2015 [27,29].

Overall, the performance of `occams` varies greatly in Arabic, Chinese, and Romanian. The variation is largely due to the quality of sentence splitting. As an example, `stanza` splits the first 10 documents of MMS 2015 Arabic into only 18 sentences, whereas `ersatz` finds 89. Some of the sentences are longer than the 250 word budget and cannot be chosen, as `occams` summaries are designed to consistently produce a summary not exceeding the target length. On the other hand, the ROUGE scores for Chinese are lower for `ersatz`, as `nltk` uses white space to tokenize. So regardless of the quality of the sentence splits, the resulting sentence will be broken down into tokens poorly. The `occams(ersatz)` results would improve by using a Chinese tokenizer, for example, `jieba` (https://github.com/fxsjy/jieba accessed on 8 December 2022), or even single-character tokenization.

Table 3. Human-to-human and WBU results from MultiLing 2015 compared with new results for `occams`, using ROUGE-2, Recall. WBU was the highest-scoring automatic system in 2015.

Language	Human	WBU	occams (stanza)	occams (ersatz)
Arabic	0.192	0.204	0.137	0.187
Chinese	0.333	0.172	0.237	0.133
Czech	0.241	0.212	0.214	0.198
English	0.202	0.183	0.181	0.173
French	0.267	0.272	0.242	0.254
Greek	0.207	0.192	0.205	0.202
Hebrew	0.188	0.216	0.156	0.158
Hindi	0.224	0.140	0.157	0.156
Romanian	0.193	0.194	0.163	0.184
Spanish	0.266	0.242	0.269	0.272

7. Conclusions

In this paper, we introduced `occams`, a fast, robust, and flexible software package for multi-lingual, single, and multi-document extractive summarization. `occams` allows users to choose their own NLP library for the document segmentation step, but it also

comes with ready-to-go, built-in support for `nltk` and `stanza`, the latter of which supports 70 languages. The package provides a number of means of computing term weights; we gave an overview of one family of these methods and went into detail on the default term weight method `POSITIONAL_DENSE`. We explained the budgeted maximum coverage problem used by `occams` to model the extractive summarization problem and an optimal approximation algorithm for solving it. A new Python extension module for `occams` written in Rust contains efficient implementations of this optimal approximation algorithm as well as a dynamic programming algorithm for the knapsack problem.

We gave a simple example showing how to use `occams` with its default options to extract a summary from the text. Then, we illustrated the performance of `occams` with its default term weighting method on DUC 2004, MultiLing 2015, and Multi-News. In each case, the approach was shown to be very competitive with the state-of-the-art methods, including neural-net-based methods, which have more demanding computational requirements as well as the need for training data. Finally, we illustrated with an example that while large neural language models generate more fluent summaries, hallucinations are possible and sometimes subtle. In the near term, extractive summarization provides an alternative to abstractive summarization and could be used in conjunction with abstractive methods to facilitate fact checking. User interfaces will need to be adapted to support this hybrid approach.

Author Contributions: Conceptualization, J.M.C., N.P.M., C.T.W. and J.S.Y.; methodology, J.M.C., N.P.M., C.T.W. and J.S.Y.; software, J.M.C., N.P.M., C.T.W. and J.S.Y.; formal analysis, J.M.C., N.P.M., C.T.W. and J.S.Y.; data curation, J.M.C., N.P.M., C.T.W. and J.S.Y.; writing—original draft preparation, J.M.C., N.P.M., C.T.W. and J.S.Y.; writing—review and editing, J.M.C., N.P.M., C.T.W. and J.S.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement:

1. DUC 2004: https://www-nlpir.nist.gov/projects/duc/data.html
2. Multi-News: https://huggingface.co/datasets/multi_news
3. MultiLing 2015: http://multiling.iit.demokritos.gr/pages/view/1540/task-mms-multi-document-summarization-data-and-information
4. MultiLing Single Document and Headline Summaries: http://multiling.iit.demokritos.gr/pages/view/1624/task-single-document-summarization http://multiling.iit.demokritos.gr/pages/view/1651/task-headline-generation

Acknowledgments: The fourth author would like to thank all the collaborators who have worked with him over the past two decades in text summarization. Among these collaborators, he especially would like to thank Sashka Davis, Jeff Kubina, Dianne O'Leary, Mary Ellen Okurowski, and Judith Schlesinger.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Comparison of OCCAMS Implementations

We present a quick comparison of the earlier C (`OCCAMS_V`) and newer Rust implementations of the greedy algorithm for the budgeted maximum coverage problem and the dynamic programming algorithm for the knapsack problem. To illustrate the difference in performance on a large problem, we used the text of *Moby Dick* by Herman Melville. We treated this as a multidocument summarization problem, where each chapter of the novel was a single document. Using `nltk` for the document segmentation step and forming terms as bigrams of stemmed work tokens, we end up with 8579 sentences and 100,918 terms.

We compare the 2 implementations by asking each of them for extractive summaries of no more than 500 words.

`OCCAMS_V` implemented the greedy algorithm for the budgeted maximum coverage problem by closely following the algorithm as given in [18]. While this formulation of the algorithm was useful there for proving an optimality bound, it is not computationally efficient. The poor performance is primarily due to the terminating condition as we indicated in Section 3. On this problem from *Moby Dick*, the main loop has 8579 iterations. Our new implementation, on the other hand, terminates as soon as the remaining budget is strictly less than the shortest remaining unused sentence. For this problem and budget, it ends up terminating after only 69 iterations. When combined with a few other improvements that benefit the new implementation, we measured the run time speedup factor of the Rust version over that of `OCCAMS_V` on this problem instance to be greater than 700.

In `occams`, we also use a dynamic programming algorithm for the knapsack problem. The current Rust-based reformulation uses a dynamic program, in which one dimension of the table is indexed by the length of the summary instead of the coverage score of the summary as was done in the C implementation. The advantage of this approach is that the length of a summary is often bounded by a small integer, e.g., 100 or 250 words or perhaps 250 or 665 characters. This means the table used in the dynamic program can be smaller so that the algorithm requires less memory and runs faster. Applied to the summarization problem derived from *Moby Dick*, the resulting table has 501 columns in the new implementation compared to 367,525 columns in the `OCCAMS_V`, or a factor of 733 times smaller (since the tables have the same number of rows).

While these comparisons on this large example are extreme, in practice, the Rust version is demonstrated to be faster and require less memory on all summarization problems tested.

Appendix B. Command Line Interface

The package includes a command `occams` with two subcommands, one for summarizing documents and one for comparing generated summaries to model summaries by computing ROUGE scores. The usage for the main program and each subcommand follows.

```
$ occams --help
usage: occams [-h] {summarize,rouge} ...

positional arguments:
  {summarize,rouge}  the subcommand to run
    summarize        extract a summary from one or more documents
    rouge            Score a generated summary against model summaries with
                     Rouge

optional arguments:
  -h, --help         show this help message and~exit


$ occams summarize --help
usage: occams summarize [-h] [-p PATHS [PATHS ...]] -b BUDGET
                        (--words | --chars) (-1 | -2) [-M MAX_SENTENCES]
                        [-m MIN_LENGTH] [-w WRAP] [--list-sentences]
                        [-t TERM_WEIGHTS] [--highlight] [--lang LANG]
                        [--download] [-H [HUMAN_SUMMARIES ...]] [--profile]

optional arguments:
  -h, --help            show this help message and exit
  -p PATHS [PATHS ...], --paths PATHS [PATHS ...]
                        the paths of files or directories to summarize
  -b BUDGET, --budget BUDGET
                        the length of the summary to produce
  --words               measure sentence and summary lengths in words
  --chars               measure sentence and summary lengths in characters
  -1, --unigrams        form terms as unigrams of tokens
  -2, --bigrams         form terms as bigrams of tokens
```

```
  -M MAX_SENTENCES, --max-sentences MAX_SENTENCES
                        set upper bound on the number of sentences used by
    the
                        summarizer
  -m MIN_LENGTH, --min-length MIN_LENGTH
                        set lower bound on the length of a sentence
  -w WRAP, --wrap WRAP  wrap summary to the given width
  --list-sentences      Write each extracted summary sentence on its own line
  -t TERM_WEIGHTS, --term-weights TERM_WEIGHTS
                        specify the method of computing frequency term
    weights
  --highlight           print the full text of the documents, highlighting
    the
                        summary sentences
  --lang LANG           set the language of the input text (default: english)
  --download            download any missing language model files if not
                        present
  -H [HUMAN_SUMMARIES ...], --human-summaries [HUMAN_SUMMARIES ...]
                        compare extracted summary to human summaries with
                        Rouge
  --profile             display execution times for summary~extraction


$ occams rouge --help
usage: occams rouge [-h] -b BUDGET (--words | --chars) [--max-n MAX_N]
                    [--lang LANG] [--download]
                    summary models [models ...]

positional arguments:
  summary               the peer summary to score
  models                model summaries to use when scoring the peer~summary

optional arguments:
  -h, --help            show this help message and exit
  -b BUDGET, --budget BUDGET
                        the length of the summary to produce
  --words               measure sentence and summary lengths in words
  --chars               measure sentence and summary lengths in characters
  --max-n MAX_N         compute Rouge-n scores for n in 1..MAX_N (default: 4)
  --lang LANG           set the language of the input text (default: english)
  --download            download any missing language model files if not
                        present
```

Appendix C. Terms of Use

We plan to release the software using one of the standard open-source licenses, e.g., MIT. In the meantime, we are open to requests to share the code.

References

1. Hong, K.; Conroy, J.; Favre, B.; Kulesza, A.; Lin, H.; Nenkova, A. A Repository of State of the Art and Competitive Baseline Summaries for Generic News Summarization. In Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14), Reykjavik, Iceland, 26–31 May 2014; pp. 1608–1616.
2. Nenkova, A.; McKeown, K., A Survey of Text Summarization Techniques. In *Mining Text Data*; Springer: Boston, MA, USA, 2012; pp. 43–76. [CrossRef]
3. Gambhir, M.; Gupta, V. Recent Automatic Text Summarization Techniques: A Survey. *Artif. Intell. Rev.* **2017**, *47*, 1–66. [CrossRef]
4. See, A.; Liu, P.J.; Manning, C.D. Get To The Point: Summarization with Pointer-Generator Networks. *arXiv* **2017**, arXiv:1704.04368.
5. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, L.u.; Polosukhin, I. Attention is All you Need. In *Proceedings of the Advances in Neural Information Processing Systems*; Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2017; Volume 30.
6. Radford, A.; Narasimhan, K.; Salimans, T.; Sutskever, I. *Improving Language Understanding by Generative Pre-Training*; 2018. Available online: https://openai.com/research/language-unsupervised (accessed on 8 December 2022).
7. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*; Association for Computational Linguistics: Minneapolis, MN, USA, 2019; pp. 4171–4186. [CrossRef]

8. Lewis, M.; Liu, Y.; Goyal, N.; Ghazvininejad, M.; Mohamed, A.; Levy, O.; Stoyanov, V.; Zettlemoyer, L. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, Online, 5–10 July 2020; pp. 7871–7880. [CrossRef]
9. ChatGPT Language Model (Version 3.5). Available online: https://openai.com (accessed on 21 December 2022).
10. Maynez, J.; Narayan, S.; Bohnet, B.; McDonald, R. On Faithfulness and Factuality in Abstractive Aummarization. *arXiv* **2020**, arXiv:2005.00661.
11. Liu, P.J.; Saleh, M.; Pot, E.; Goodrich, B.; Sepassi, R.; Kaiser, L.; Shazeer, N. Generating Wikipedia by Summarizing Long Sequences. In Proceedings of the 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, 30 April–3 May 2018; Conference Track Proceedings. OpenReview.net.
12. Gillick, D.; Favre, B. A Scalable Global Model for Summarization. In Proceedings of the Workshop on Integer Linear Programming for Natural Langauge Processing, Boulder, CO, USA, 4 June 2009; Association for Computational Linguistics: Minneapolis, MN, USA, 2009; ILP '09, pp. 10–18.
13. Conroy, J.M.; Davis, S.yT. Section Mixture Models for Scientific Document Summarization. *Int. J. Digit. Libr.* **2018**, *19*, 305–322. [CrossRef]
14. Ariew, R. Did Ockham Use His Razor? *Francisc. Stud.* **1977**, *37*, 5–17. [CrossRef]
15. McDonald, R. A Study of Global Inference Algorithms in Multi-Document Summarization. In *Advances in Information Retrieval, Proceedings of the 29th European Conference on IR Research, ECIR 2007, Rome, Italy, 2–5 April 2007*; Amati, G., Carpineto, C.; Romano, G., Eds.; Springer: Berlin, Heidelberg, 2007; pp. 557–564.
16. Bird, S.; Loper, E.; Klein, E. *Natural Language Processing with Python*; O'Reilly Media, Inc.: Sebastopol, CA, USA 2009.
17. Qi, P.; Zhang, Y.; Zhang, Y.; Bolton, J.; Manning, C.D. Stanza: A Python Natural Language Processing Toolkit for Many Human Languages. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations, Online, 5–10 July 2020.
18. Khuller, S.; Moss, A.; Naor, J.S. The Budgeted Maximum Coverage Problem. *Inf. Process. Lett.* **1999**, *70*, 39–45. [CrossRef]
19. Davis, S.T.; Conroy, J.M.; Schlesinger, J.D. OCCAMS—An Optimal Combinatorial Covering Algorithm for Multi-Document Summarization. In Proceedings of the 2012 IEEE 12th International Conference on Data Mining Workshops, Brussels, Belgium, 10–13 December 2012; pp. 454–463. [CrossRef]
20. Lin, H.; Bilmes, J. Multi-document Summarization via Budgeted Maximization of Submodular Functions. In Proceedings of the Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Los Angeles, CA, USA, 2–4 June 2010; pp. 912–920.
21. Steinberger, J.; Jezek, K. Using Latent Semantic Analysis in Text Summarization and Summary Evaluation. In Proceedings of the 7th International Conference ISIM, Ostrava, Czech Republic, 19–22 April 2004.
22. Steinberger, J.; Jezek, K. Update Summarization Based on Novel Topic Distribution. In Proceedings of the ACM Symposium on Document Engineering, Munich, Germany, 16–18 September 2009; pp. 205–213.
23. Priebe, C.E.; Park, Y.; Vogelstein, J.T.; Conroy, J.M.; Lyzinski, V.; Tang, M.; Athreya, A.; Cape, J.; Bridgeford, E. On a Two-Truths Phenomenon in Spectral Graph Clustering. *Proc. Natl. Acad. Sci. USA* **2019**, *116*, 5995–6000. [CrossRef] [PubMed]
24. Fabbri, A.; Li, I.; She, T.; Li, S.; Radev, D. Multi-News: A Large-Scale Multi-Document Summarization Dataset and Abstractive Hierarchical Model. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, Florence, Italy, 28 July–2 August 2019; pp. 1074–1084. [CrossRef]
25. Multi-Document Summarization on Multi-News: Leaderboard. Available online: https://paperswithcode.com/sota/multi-document-summarization-on-multi-news?metric=ROUGE-2 (accessed on 8 December 2022).
26. Wicks, R.; Post, M. A Unified Approach to Sentence Segmentation of Punctuated Text in Many Languages. In Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), Online, 1–6 August 2021; pp. 3995–4007. [CrossRef]
27. Giannakopoulos, G.; Kubina, J.; Conroy, J.M.; Steinberger, J.; Favre, B.; Kabadjov, M.A.; Kruschwitz, U.; Poesio, M. MultiLing 2015: Multilingual Summarization of Single and Multi-Documents, On-line Fora, and Call-Center Conversations. In Proceedings of the SIGDIAL 2015 Conference, the 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue, Prague, Czech Republic, 2–4 September 2015; pp. 270–274. [CrossRef]
28. Steinberger, J.; Lenkova, P.; Kabadjov, M.; Steinberger, R.; van der Goot, E. Multilingual Entity-Centered Sentiment Analysis Evaluated by Parallel Corpora. In Proceedings of the International Conference Recent Advances in Natural Language Processing 2011, Hissar, Bulgaria, 12–14 September 2011; pp. 770–775.
29. Giannakopoulos, G. MultiLing 2015 Data. Available online: http://multiling.iit.demokritos.gr/pages/view/1516/multiling-2015 (accessed on 8 December 2022).