University of Zurich^UZH

Bachelorthesis Mai 12. 2017

# SSE - An Automated Sample Size Extractor for Empirical Studies

**Kürsat Aydinli**
of Grabs SG, Switzerland

Student-ID: 13-926-910
kuersat.aydinli@uzh.ch

Advisor: **Patrick de Boer**

Prof. Abraham Bernstein, PhD
Institut für Informatik
Universität Zürich
http://www.ifi.uzh.ch/ddis

# SSE - An Automated Sample Size Extractor for Empirical Studies

**Kürsat Aydinli**
Department of Informatics
University of Zurich
kuersat.aydinli@uzh.ch

**Patrick de Boer**
Department of Informatics
University of Zurich
pdeboer@ifi.uzh.ch

**Abraham Bernstein**
Department of Informatics
University of Zurich
bernstein@ifi.uzh.ch

## ABSTRACT

This thesis proposes a novel method for automatically retrieving the sample size for a variety of empirical studies. The studies being targeted cover a wide range of research fields, including but not limited to health sciences, computer-human interaction, psychlogy and management sciences.

SSE is composed of a three-level pipelined algorithmic framework. At the first stage pattern matching harnessing regular expressions is utilized to extract eligible sentence fragments most likely containing the information of interest. The second stage is responsible for rule-based filtering of the matches identified at step 1. This ensures that only sample sizes are dealt with during subsequent steps. The third level of the algorithm is concerned with applying manually designed heuristics in order to further filter the entries passing stage 2 and return the requested information.
The algorithm achieves a good accuray while showing promising performance with respect to competitor systems.

### Author Keywords

Sample Size Extraction, Pattern Matching, Rule-Based Filtering, Case-Specific Heuristics, Valid Research

## INTRODUCTION

When conducting and designing empirical studies (e.g medical trials), attaching considerable importance to a confident number of individuals to include in the analysis - the sample size of the study - is a key step in ensuring statistical reliability. Due to the fact that investigators perform studies on a limited number of participants rather than the whole population, the sample size is closely tied to the scientific relevance of a study.
Any empirical study involves gathering data from study participants on different variables. Applying statistical methods (*e.g* t-test) at various stages of a study requires mostly a minimum number of participants. Ensuring these prerequisites helps in terms of sketching an appropriately designed experiment leading to reliable results from which meaningful interpretations can be drawn [2].
The usefullness of a study along with the data presented is partially determined by the sample size [15].
The main objective of the research at hand is to extract sample size information from a given study article.
This paper evolved in the context of a comprehensive statistics project conducted by the *Dynamics and Distributed Information Systems group (DDIS)* at the University of Zurich. The main motivation of the project is to assess and evaluate the adequate usage of statistical methods and their assumptions in various research disciplines. As a result, *PaperValidator* was developed - an open-source statistics validation tool which allows the automated validation of statistics in puplications. At its current state, PaperValidator mainly focuses on the valid usage of statistical methods in particular checking whether underlying assumptions are considered and reported as such [14].
As an advancement to PaperValidator, SSE was developed in order to include sample size analysis. This component is insofar worth considering as it contributes to the scientific significance of a study.

Our information extraction task is heterogenous with many respects and poses several challanges. The input texts (study publications) for our framework differ greatly in both the writing style of different authors as well as textual discrepancies across various research fields. This makes the information of interest - the sample size - be expressed in many dissimilar ways. In general, number disambiguation is a defiance in terms of sample size extraction as numbers are included as descriptors for a variety of study characteristics like the number of subjects, participant age or different variable quantities (e.g participants having diseas XY).
Finally, in most cases there are many textual fragments containing identifiers likely to represent a sampe size. This can lead to a large number of potentially interesting values whereby discrepancies between them have to be resolved. To overcome these issues, we have developed a pipelined methodology incorporating rule-based filtering steps as well as various heuristics.

## RELATED WORK

The problem of automatically recognizing and extracting the sample size from unstructured text falls under the general category of *Information Extraction* (IE). IE is defined as the process of reconstructing disambiguated quantifiable data from natural language text snippets. In contrast to IE, *Information Retrieval* (IR) indicates the gathering of relevant information resources from a wide collection of available information resources (*e.g* Google search) [6].

There exist several approaches in the literature with regard to the automated retrieval of study specifications. In the majority of cases attention is devoted to the extraction of study characteristics from biomedical articles by analysizing their abstract sections.

Cassidy and Hui [3] describe a system for extracting study design parameters (*e.g* age of subjects, study duration, number of subjects) from nutritional genomoics abstracts. Their approach basically consists of extracting potentially relevant sentence fragments using regular expressions suiting the individual parameter types followed by additional rules to filter the results and return the most adequate finding.

The *Trial Bank* project conducted by Bruijn *et al.* [7] presents a two-stage architecture for extracting key study information elements from randomized controlled trials (RCT) including information related to the study population. Their methodology follows a machine learning approach in the first step while annotating relevant sentences according to their informational content. In a second step, an automatic extractor was designed with a set of rules includig regular expressions to extract snippets of required information from the sentences identified in the first step. Their framework is different from the one described previously in the sense that it is applicable for running on full-text articles whereas the work of Cassidy and Hui [3] focuses on the abstract section.

Hansen *et al.* [8] employs a SVM classifier in order to extract the sample size from abstracts describing RCTs with the assumption that the largest number found is the correct sample size. In addition, their objective lies on extracting the initial number of enrolled participants into the study before any exclusions or allocations to different study arms.

Hara and Matsumoto [9] propose a system for extracting controlled trials design information from MEDLINE abstracts. They utilize NLP techniques consisting of base noun-phrase (NP) chunking followed by pattern matching and subsequent filtering in order to extract target sentences.

*ExaCT* is a comprehensive system proposed by Kiritchenko *et al.* [11] for the automated extraction of coltrolled trial characteristics from journal publications. ExaCT primarily searches the text with a statistical text classifier to locate sentences best describing trial characteristics. Next, the IE engine applies simple rules to the selected sentences to extract the requested information. Furthermore it proivdes a web-based user interface for additional modifications of the suggestions.

Xu *et al.* [18] developed a mechanism to extract subject demographic information from RCT abstracts. They employ text classification in order to identify sentences containing subject demographics. Finally, NLP techniques are utilized to extract relevant information.

It becomes obvious that recent literature in this research field concentrates mainly on clinical trials due to their significant roles as most important sources of evidence for medical practice and the design of new trials. These systems indeed provide an easy to handle way dealing with the time consuming issue of manually reviewing a large amount of medical publications on the web.

Compared to the previously described techniques for extracting study information including the sample size, our work extends and enhances previous research efforts in two main directions. First, the approach of this work can be seen as an attempt applicable not only to medical research publications but to a wide range of research fields.
Second, analysis is performed on full-text publications whereas literature operating in this field focuses mostly on abstracts. Entire articles present more challenges yet allow us to extract information typically not found in abstracts/summaries.

## METHODS

### Data Source

In this study we used a random sample of 86 full-text articles from a variety of scientific journals from 2014:

- American Journal of Sociology
- BMJ - British Medical Journal
- Journal of Management
- ACM CHI Conference
- Cognitive Psychology
- Management Science

A first set of 25 papers was used and analyzed in order to design the heuristical techniques employed by SSE. The remaining set of 61 papers constitutes the ground truth which is used for examining the effectiveness of SSE.

### Sample Size Types

We developed SSE with the aim of extracting three types of sample sizes which will be referred to as follows:

- **OL1**: Initial sample size - number of participants initially enrolled by the investigators before any exclusions or assignments to study arms
- **NL1**: Actual sample size - number of participants included in the data analysis after excluding ineligible participants
- **L2**: Group sample size - number of participants in each study arm if present. Typically *L2* sample sizes sum up to the *NL1* sample size

For the purpose of evaluating the final system, the sample sizes comprising the golden standard were labelled as follows: *OL1*, *NL1*, *L2* or *FALSE*. The following sample extract demonstrates the annotation:

```
'Initially, we screened 1000 people
who met the inclusion criteria. After
excluding 200 participants, 800 persons
```

```
were included in the data analysis. Of
the subjects who underwent randomization,
200 were randomly allocated to the group
X, 250 were assigned to group Y and 550
were allocated to group Z. The group X
consists of 110 males and 90 females
where as group Y has ...’
```

In this case the particular entries in the ground truth would be labeled as follows:

Table 1. Labeling of identified sample sizes

| PDF Name | ID | N | Comment | Label |
|---|---|---|---|---|
| SampleStudy.txt | 1 | 1000 | 1000 screened | OL1 |
| SampleStudy.txt | 2 | 200 | 200 exluded | FALSE |
| SampleStudy.txt | 3 | 800 | 800 included in the data analysis | NL1 |
| SampleStudy.txt | 4 | 200 | 200 allocated to ... | L2 |
| SampleStudy.txt | 5 | 250 | 250 assigned to ... | L2 |
| SampleStudy.txt | 6 | 550 | 550 allocated to ... | L2 |
| SampleStudy.txt | 7 | 110 | 110 males | FALSE |
| SampleStudy.txt | 8 | 90 | 90 females | FALSE |

The main reason for emphasizing on these three types of sample sizes rather than the actual one is that empricial studies mostly report all of them in their papers. On the other hand, the classification of these types facilitates the comparison of SSE with the systems described in the related work section.

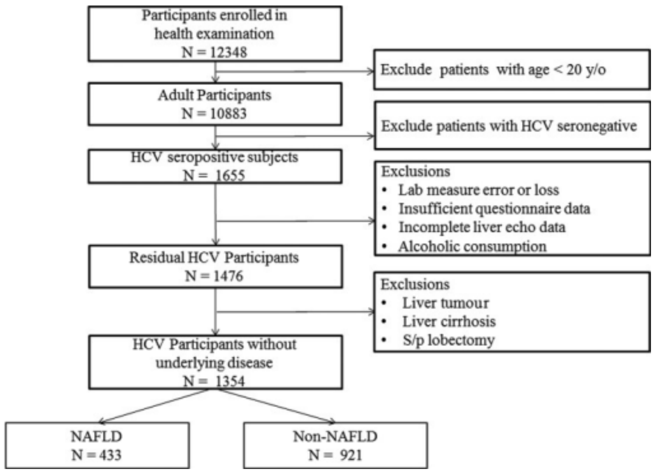
## Typical Participant Flow

Typically the sample size falls during the progress of a study. At the very first stage, a certain number of participants is contacted and invited. After the screening process, some of them are excluded due to not fitting the study criteria. If the study intends to compare different treatments or intervensions on different groups (especially in medical research), than the number of eligible persons is further divided into different study arms (see Figure 1).

Within the scneario outlined by Figure 1, SSE would aim to extract following information elements: *12348* (OL1), *1354* (NL1), *433* ($L2_1$) and *921* ($L2_2$).

## Approach SSE

SSE follows a multi-stage pipelined approach on extracting and calssifying the various sample size types found in a study artcle. The first stage involves 5 different pattern matching modules in order to capture appropriate sentence fragments. Each module contains patterns for extracting a specific kind of sample size. In this



Participants enrolled in health examination N = 12348
Exclude patients with age < 20 y/o
Adult Participants N = 10883
Exclude patients with HCV seronegative
HCV seropositive subjects N = 1655
Exclusions
• Lab measure error or loss
• Insufficient questionnaire data
• Incomplete liver echo data
• Alcoholic consumption
Residual HCV Participants N = 1476
Exclusions
• Liver tumour
• Liver cirrhosis
• S/p lobectomy
HCV Participants without underlying disease N = 1354
NAFLD N = 433
Non-NAFLD N = 921

Fig. 1. Sample Participants Flow Chart

stage, the system is basically divided into two subsystems which focus on different kinds of sample sizes. The first subsystem aims at extracting all kinds of sample sizes from the article whereas the second subsystem is devoted to the extractio of following sample size categories:

- **OL1**: Initial sample size
- **NL1**: Actual sample size
- **!NL1**: Excluded sample size - Typically difference between *OL1* and *NL1*
- **L2**: Group sample size

Once the sentence fragements according to the regular expressions in the 5 modules are extracted, rule-based filtering is applied in the second stage in order to filter clauses accidentally matching a particular pattern without representing a sample size. The completion of stage 2 results in 5 pools of integer values with each containing plausible sample sizes according to their types. The most important part of SSE is embedded in stage 3 where manually crafted heuristics are applied to the 5 sample size pools in order to extract the correct ones. The overall architecture of the system is shown in Figure 2.

## Stage 1: Pattern Matching

The first phase of the SSE pipeline consists of 5 different pattern matching modules utilizing regular expressions. At this stage, the system is further divided into two different tracks (see Figure 2). The first track (Module *TS*) aims at extracting all sorts of sample sizes from a paper, thus having more general patterns. The second track (Modules *OL1, NL1, !NL1* and *L2*) intends to pool the sample sizes into 4 different categories (see Figure 2).

Table 2 describes the patterns made use of by each module. The

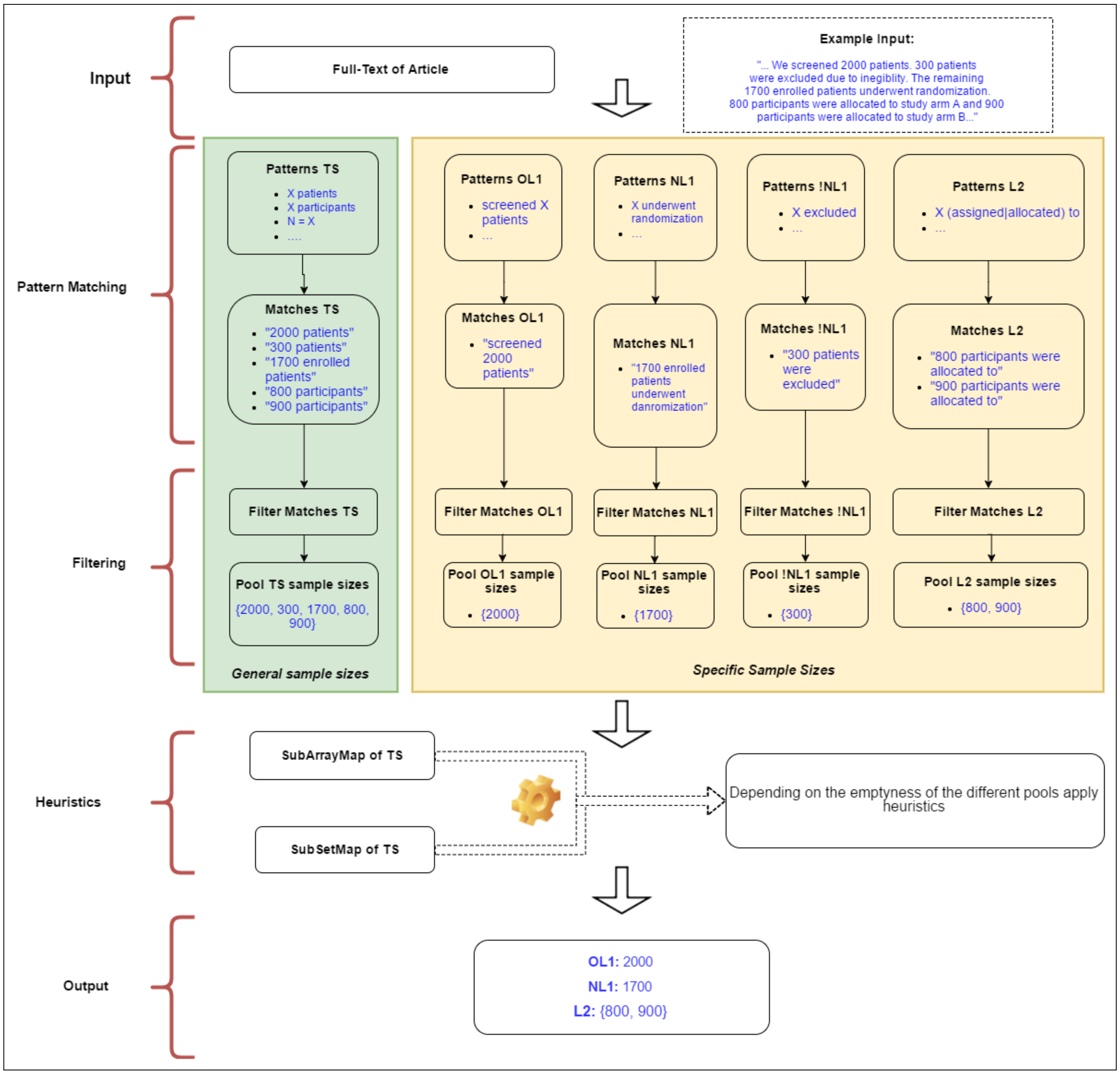


Fig. 2. Design of SSE

regular expressions were developed and adjusted during the investigation of a large set of scientific studies spanning various research disciplines.

## Stage 2: Rule-Based Filtering

Applying the search patterns to the full-text of an article results in 5 different pools containing string matches each. In order to capture distant dependencies between the integer value and the identifier (e.g 'participants'), the regular expressions are designed to allow up to 50 arbitrary characters between these tokens. This regulation allows SSE to match strings of the form '`50 participants`'

Table 2. Patterns of the SSE modules

| Patterns TS | Patterns OL1 | Patterns NL1 |
|---|---|---|
| X women | X invited<br>invited X | X recruited/included<br>recruited/included X |
| X members | X reviewed<br>reviewed X | Of the X recruited/included |
| X cases | X screened<br>screened X | X were randomized |
| X controls | X assessed for eligibility | X underwent randomization |
| X respondents | X met inclusion criteria | X were included in the (data)? analysis |
| X persons | | |
| X participants | | |
| X subjects | | |
| X patients | | |
| X people | | |
| X individuals | | |
| X adults | | |
| X recruited | | |
| N=X | | |
| Total of X | | |
| Study population included X | | |
| X enrolled<br>enrolled X | | |
| Data of / from X | | |

| Patterns !NL1 | Patterns L2 |
|---|---|
| X were not eligible | X in the group |
| X did not meet (inclusion)? criteria | X received |
| X were excluded | X (allocated\|assigned) to |
| | X / Y |

as well as '<u>50</u> alcohol drinking as well as medication XY receiving <u>participants</u>'.

At the other hand, fragments containing numbers not being related to the identifier are also likely to be matched. The fragment '<u>2012</u>. Regarding the health status of the <u>patients</u>' serves as such an example.
In this case the value 2012 probably represents a year instead of a sample size. In order to exclude misleading matches from further analysis, rule-based filtering is performed on the matches of a module. The utilized filterings are outlined in Table 3.

In general, it is attempted to filter matches where the integer value is not related to an identifier token (e.g 'participants'). This may be due to appearance in different sentences or due to seperation by a collection of pre-defined seperating words like 'for', 'and', 'from', 'when', 'among' etc.

Table 3. Filtering constraints on the matches of each module

| Filter TS | Filter OL1 | Filter NL1 |
|---|---|---|
| X years, months, weeks, days, hours etc. | | |
| X and Identifier unrelated | | |
| Contains special symbols | | |
| Contains Non-ASCII | | |
| length of number > 9 | | |
| Contains '%' | '%' $\notin$ X(Y%) | |
| | | Contains 'not' |

| Filter !NL1 | Filter L2 |
|---|---|
| X years, months, weeks, days, hours etc. | |
| X and Identifier unrelated | |
| Contains special symbols | |
| Contains Non-ASCII | |
| '%' $\notin$ X(Y%) | |
| length of number(s) > 9 | |
| | X > Y in X/Y |
| | sum($X_i$) $\neq$ Y in [$X_1$/Y; $X_2$/Y;...] |

Matches of the pool *TS* are not allowed to contain '%' whereas the other pools may contain this sign. In case a '%' is encountered it has to appear in clauses where the sample size preceeds '%'.
For instance the clause '... 400(45%) patients ...' is a valid match for *OL1, NL1, !NL1* and *L2*.
The pattern 'X/Y' from the pool *L2* poses further restrictions since it risks capturing irrelevant clauses from tables or even the references section (*e.g* as part of a doi identifier). The first part of this pattern ('**X**/Y') only represents a subgroup for as long as it is smaller than the second part of the pattern ('X/**Y**').
Furthermore it is assumed that in the case the subgroups are represented using this pattern, SSE should be able to capture the subgroups all together. More specifically, summing up every '$X_i$' in '$X_i$/Y' should be equal to 'Y'.
Following fragment is representative for not filtering the 'X' matches of the pattern 'X/Y':

> 'In total we enrolled 200 subjects. The subjects underwent randomization as follows: 30/200 were allocated to group A. 60/200 were allocated to group B and 110/200 were allocated to group C.'

After processing this clause, the pool *L2* would provide following matches: 30/200, 60/200 and 110/200. In this case it is assured that

$$\sum_{i=1}^{3} X_i = 200$$

Matches of this pattern exhibiting such a behaviour are kept in the *L2* pool.

Moreover, all numbers with a length of > 9 are discarded since such huges numbers are not likely to represent sample sizes encountered in empirical studies. As a matter of fact, this restriction ensures a successful typecast to `Int`.

Once irrelevant matches are filtered from the pools, the respecive integer values are extracted. Passing this second stage, the SSE pipeline enters the last major stage of the framework which deals with heuristical calculations on the 5 pools.

## Stage 3: Case-Specific Heuristics

At this point SSE has populated 5 integer buckets containing potential sample sizes. The manually crafted heuristics which are applied in this stage have some underlying assumptions. Due to the sophisticated elaboration of regular expressions and subsequent filtering of misleading matches it is assumed that the numbers in each pool contain more or less reasonable sample sizes of the corresponding module type - in particular:

- **Pool TS:** contains all kinds of sample sizes
- **Pool OL1:** contains primary sample sizes of enrollment
- **Pool NL1:** contains actual sample sizes included in the data analysis
- **Pool !NL1:** contains sample sizes excluded from the study
- **Pool TS:** contains sample sizes from study arms

Furthermore it is assumed that the pool *TS* always contains some sample sizes on which the heuristics are applied. As mentioned earlier in the paper, the pool *TS* intends finding every possible sample size from the text.
The other 4 pools are utilized as an aid on categorizing and filtering the sample sizes from TS.

Due to the limited patterns of the other 4 modules it is not guaranteed that each pool is non-empty. Because the heuristics build upon the assumption of dealing with different kinds of sample sizes, it needs to be aware of which pool is available for the calculations. The information about the emptyness of each pool thus has a crucial impact on the calculations.
For instance, a reasonable attempt would be to sum up values from *L2* and check if a number in *NL1* matches the sum. Another effort would be to substract values of *!NL1* from the entries in *OL1* and check if a value in *NL1* matches the difference and so forth.
Before executing any heuristical calculations, SSE therefore needs to check which of the 4 pools are available. By doing so, SSE distinguishes between 16 cases denoted in Table 4.
SSE applies cascades of calculations on the pools depending on the case encountered. Each cascade consists of a sequence of calculations with decreasing priority. If the first calculation does not lead to a result than the next one is executed and so forth. If a calculation leads to reasonable outcomes than the involved sample sizes are stored in a list containig potentially correct results.

Table 4. Cases for different heuristics

| Case | OL1 | NL1 | !NL1 | L2 |
|---|---|---|---|---|
| 1 | Non-Empty | Non-Empty | Non-Empty | Non-Empty |
| 2 | Empty | Non-Empty | Non-Empty | Non-Empty |
| 3 | Non-Empty | Empty | Non-Empty | Non-Empty |
| 4 | Non-Empty | Non-Empty | Empty | Non-Empty |
| 5 | Non-Empty | Non-Empty | Non-Empty | Empty |
| 6 | Empty | Empty | Non-Empty | Non-Empty |
| 7 | Empty | Non-Empty | Empty | Non-Empty |
| 8 | Empty | Non-Empty | Non-Empty | Empty |
| ... | ... | ... | ... | ... |
| ... | ... | ... | ... | ... |
| 16 | Empty | Empty | Empty | Empty |

This means that each processed paper is assigned three list data structures for each sample size type (*OL1*, *NL1* and *L2*). In case a calculation was successful the involved sample sizes are added to the corresponding lists. It is noteworthy that the list containing promising *L2* sample sizes is a list embedded in a list. Each list item stores the single *L2* values.
For instance, passing all the heuristics may result in following final lists:

- **OL1**: `List = [3456, 6517, 3000, 6517]`
- **NL1**: `List = [576, 893, 45]`
- **L2**:
  - `List = [List = [250,326]`
  - `List = [400,493]`
  - `List = [15,20,10]`

Each entry in a list was added due to successfully passing some rule-based calculations which will be discussed in detail further down the paper. Furthermore, the list items in the embedded *L2* list each correspond to a split-up of one *NL1* entry.
This is due to SSE seeking to find *NL1* and *L2* sample sizes in combination. In case this is not possible, only the *NL1* entry is retrieved. In the rest of the paper, these three lists will be referred to as *OL1_potential*, *NL1_potential* and *L2_potential* respectively.

At this point it is worth of mention that two additional map data structures are generated for the *TS* pool: *SubArrayMap* and *SubSetMap*:

- **SubArrayMap**: `Map` $[\mathtt{TS}_i \texttt{ -> } \mathtt{List}[\mathtt{TS}_j, \mathtt{TS}_{j+1}, \mathtt{TS}_{j+2}, \ldots]$ `with`
$$\sum_{j,j+1,j+2,\ldots} \mathtt{TS} = TS_i$$
- **SubSetMap**: `Map` $[\mathtt{TS}_i \texttt{ -> } \mathtt{List}[\mathtt{TS}_k, \mathtt{TS}_l, \mathtt{TS}_m, \ldots]$ `with`
$$\sum_{k,l,m,\ldots} \mathtt{TS} = TS_i$$

The *SubArrayMap* contains for each $TS_i$ value a subarray from the TS pool of length $\geq 2$ excluding itself which sums up to $TS_i$. In similar fassion, the *SubSetMap* contains a subset from the TS pool of length $\geq 2$ excluding $TS_i$ with sum equal to $TS_i$.
The difference is that *SubArrayMap* forces to find sequential pattern matches whereas *SubSetMap* may contain matches which are not in sequential ordering. A sample representaion would be the following scenario:

- **TS Pool**: `List = [310, 5, 9000, 20, 11250, 89, 800, 196, 150, 160]`
- **SubSetMap**: `Map = [310 -> List[5,20,89,196]]`
- **SubArrayMap**: `Map = [310 -> List[150,160]]`

The idea behind these data structures is to capture dependencies between the values of *TS*. If at some point the calculations on the other specific pools return a potentially interesting sample size `X`, then using these maps it can be investigated whether `X` represents a composition of other sample sizes.

The next part of this section continues with the specific heuristics for each of the 16 cases which can be encountered (see Table 4).
The row headers of the cases are supported through colouring the 'emptyness' of the pools *OL1, NL1, !NL1* and *L2*. Green cellcolor indicates a non-empty pool whereas a red cell indicates an empty pool. Non-empty pools are those which are available and thus can be used in the single calculations. Remember that the *TS* pool is assumed to be non-emtpy.
The sub-cases (e.g 1a, 1b,...) of each main case constitute a cascade. If the first calculation does not lead to reasonable results, then the next calculation is triggered.
Executing sub-cases aims to find values likely to represent *NL1* and *L2* sample sizes which are then stored in *NL1_potential* or *L2_potential* respectively. Besides these sub-cases, there are *roll-back* procedures which are executed irrespective of the sub-cases. They are primarily meant to extract *OL1* sample sizes and storing them in *OL1_potential*.
The following section will discuss case 1 in detail. Cases 2-16 are described in the appendix.

| **Case 1** | **OL1** | **NL1** | **!NL1** | **L2** |
|---|---|---|---|---|

- <u>Case (1a):</u> ∃ $\mathtt{OL1}_i$ ∧ subset of `!NL1` such that:
  - $\mathtt{OL1}_i$ - subset(`!NL1`) = $\mathtt{NL1}_i$
  - SubArrayMap.keySet.contains($\mathtt{NL1}_i$) or
  - SubSetMap.keySet.contains($\mathtt{NL1}_i$)

  $\Longrightarrow$ Add $\mathtt{NL1}_i$ to *NL1_potential* and its subArray resp. subSet to *L2_potential*

Case (1a) shows the calculation with the highest priority for case 1. This sub-case asks if an entry of the *NL1* pool can be composed of substracting 'excluding' sample sizes from a potentially initial sample size. If such a $\mathtt{NL1}_i$ exists, then the *TS* pool is investigated to check wheter $\mathtt{NL1}_i$ can be composed of some other sample sizes.

For this purpose *SubArrayMap* and *SubSetMap* are iterated over to assess if they contain a key equal to $\mathtt{NL1}_i$. If this is the case it means that $\mathtt{NL1}_i$ can be put together by at least 2 other sample sizes. This positive control results in adding $\mathtt{NL1}_i$ to the list of possible *NL1* sample sizes of the paper. In analogeous fashion the corresponding subArray or subSet is added to the list of potential *L2* sample sizes.

In case the map data structures are checked, priority is given to *SubArrayMap* over *SubSetMap*. The intuition behind this handling is that the authors of a study maily break down the actual sample sizes in its components right after referring to it. Following sentence demonstrates the scenario:

```
'In total we enrolled 800 persons whereby
300 persons were allocated to treatment
group X and 500 persons were assigned
to group Y'
```

The SubArrayMap would contain inter alia following entry:

```
(800) -> List = [300,500]
```

It is not guaranteed that the SubSetMap contais this exact sequence. Thus, if SubArrayMap returns an entry than this entry is stored in the lists of potential sample sizes of a paper.

- <u>Case (1b):</u> SubArrayMap ≠ ∅∨ SubSetMap ≠ ∅ such that:
  - SubArrayMap.keySet.contains($\mathtt{NL1}_i$)
  - SubSetMap.keySet.contains($\mathtt{NL1}_i$)

  $\Longrightarrow$ Add $\mathtt{NL1}_i$ to *NL1_potential* and its subArray resp. subSet to *L2_potential*

If case (1a) did not lead to some outcome, i.e did not fill *NL1_potential* or *L2_potential*, then case (2b) is triggered.
Here it is asked whether *TS* contains a subArray or subSet summing up to $\mathtt{NL1}_i$.

- <u>Case (1c):</u> ∃ subSet of the *L2* such that:
  - sum(subSet) = $\mathtt{NL1}_i$

  $\Longrightarrow$ Add $\mathtt{NL1}_i$ to *NL1_potential* and subset(L2) to *L2_potential*

(1c) checks whether there is a $\mathtt{NL1}_i$ composed of a subset of the *L2* pool.

- <u>Case (1d):</u> ∃ $\mathtt{NL1}_i$ = max(pool *NL1*) such that:
  - SubArrayMap.keySet.contains ($\mathtt{NL1}_i$) or
  - SubSetMap.keySet.contains($\mathtt{NL1}_i$)

  $\Longrightarrow$ Add $\mathtt{NL1}_i$ to *NL1_potential* and its subArray resp. subSet to
  *L2_potential*

This case looks for an entry in the map structures comprising the largest value in the pool *NL1*.

- Case (1e): SubArrayMap $\neq \emptyset \vee$ SubSetMap $\neq \emptyset \implies$ Add max(SubArrayMap.keySet) or max(SubSetMap.keySet) to *NL1_potential* and the subArray resp. subSet values to *L2_potential*

Case (1e) attempts to take the entry with the largest key from *SubArrayMap* or *SubSetMap*

- Rollback Pool *OL1* such that:
  - OL1.filter($\mathtt{OL1}_i$ > max(*NL1*)) $\neq \emptyset$

  $\implies$ Add max(*OL1*) to *OL1_potential*

These sub-cases including the rollback procedure represent the single calculation steps in order to fill the potential sample size pools for an article. They are arranged in a modular way with decreasing priority. The order of the sub-cases was arranged by intuition and research.
In a similar fashion all other sub-cases per 'superior' case are handled. They are described in more detail in the appendix.

## Evaluation

In order to evaluate the performance for all three sample size types, a test set comprising 61 full-text articles from various scientific research fields was used. The identified sample sizes from the papers were manually annotated with either *OL1*, *NL1*, *L2* or *FALSE*.
The labelled dataset was considered to represent the ground truth. For the purpose of quantifying the performance, following metrics were utilized:

$$\frac{\text{Sample Sizes found and correct}}{\text{Sample Sizes found}} = Precision$$

$$\frac{\text{Sample Sizes found and correct}}{\text{Sample Sizes correct}} = Recall$$

$$\frac{\text{Precision * Recall}}{\text{Precision + Recall}} \times 2 = F - Score$$

Furthermore, following evaluation criteria were categorized:

- **Exact Match:** Outcome of SSE is fully in accordance with the annotation in the ground truth
- **Partial Match:** Any overlap between outcome of SSE and annotation in the ground truth

The *exact match* criterion is applicable to all three sample size types (*OL1, NL1, L2*) meaning that the returned value for a specific sample size class by SSE can fully correspond to the annotation in the ground truth.

The *partial match* criterion is only applicable to the *L2* sample size since in this case the list of group sample sizes identified by SSE can overlap with the sample sizes in the testset labeled as L2. The partial match criterion is not applicable to the other sample size types since there exists at most only one per paper which can not be retrieved partially.

The sample application of SSE in table 5 demonstrates the difference between exact and partial match performances. The left column contains *L2* sample sizes which are assumed to be annotated as such in the ground truth. The right column contains *L2* sample sizes assumed to be returned by SSE.

Table 5. Example Exact (EM) vs. Partial Match (PM)

| L2 in Ground Truth | L2 found by SSE | EM | PM |
|---|---|---|---|
| `List = [40, 50, 80]` | `List = [40, 60, 70]` | | X |
| `List = [300, 400]` | `List = [300, 400]` | X | X |
| `List = [110, 120, 130]` | `List = [350, 5, 5]` | | |
| `List = [45, 55]` | `List = [90, 55]` | | X |
| `List = [789, 801, 900]` | `List = [789, 801, 400, 500]` | | X |
| `List = [85,75]` | | | |
| `List = [370,350]` | | | |

The scenario outlined by table 5 shows one exact match vs. 4 partial matches with respect to the sample sizes found by SSE. Comparing these criteria the precision of exact vs. partial match would be `20% (1/5)` and `80% (4/5)` respectively. In a similar way the recall metric could be calculated as `14% (1/7)` and `57% (4/7)` respectively.

The results of the actual evaluation of SSE on the testset comprising 61 full-text articles are shown in table 6.

Table 6. Evaluation results of SSE on ground truth

| | Exact Match | | | Partial Match | | |
|---|---|---|---|---|---|---|
| | **Prec.** | **Rec.** | **F-Score** | **Prec.** | **Rec.** | **F-Score** |
| **OL1** | 0.71 | 0.4 | 0.52 | - | - | - |
| **NL1** | 0.73 | 0.69 | 0.71 | - | - | - |
| **L2** | 0.33 | 0.31 | 0.32 | 0.71 | 0.67 | 0.69 |

The evaluation of *L2* demonstrates that loosening the restrictions on the matches increases the overall performance roughly by the factor of over 2. In addition it is noteworthy that the main contribution of SSE is to extract the actual sample size (in our case *NL1*). For this purpose it achieves the best performance.

Primary causes of errors included violations of assumptions built into the heuristics (*e.g* a specific pool containing values intended to be in another pool), overlooking of important patterns or simply

errors in the design of the regular expressions (*e.g* missing tolerance for spacing or capital letters).
Due to the pipelined architecture of SSE, an error in stage 1 or 2 has significant implications on the outcome of stage 3.

In order to put the performance of SSE into a relative perspective, SSE was also evaluated on two different test datasets used by related extractors.
We contacted the authors of the papers referenced in the *Related Work* section for requesting their testset used for evaluation. Two of them returned with a positive reply. The testsets were reconstructed whereby a subsample was chosen to be examined by SSE.
Both of the compared systems refer to the *NL1* sample size in SSE.

Hara and Matsumoto [9] designed their system for the application on abstracts describing phase III randomized controlled trials (RCTs). SSE was run on a sample of 50 manually annotated abstracts from their dataset. The results of the comparison is shown in table 7.

Table 7. Comparing SSE with Hara and Matsumoto

| | **Hara and Matsumoto** | **SSE** |
|---|---|---|
| **Precision** | 0.803 | 0.936 |
| **Recall** | 0.794 | 0.898 |
| **F-Score** | 0.8 | 0.92 |

It is obvious that SSE outperforms the approach of Hara and Matsumoto. Because abstracts usually do not contain a large number of potential sample size matches, SSE only has to find the correct number from a small set of numbers. In most cases, these abstracts report the *NL1* sample size along with its breakdown into subgroups if present.
Thus, in the majority of cases the *SubArrayMap* and *SubSetMap* data structures of the *TS* pool should be sufficient to find the correct number.

The second dataset used to draw comparison was provided by Kiritchenko *et al.* [11]. Their extractor was employed on full-text articles representing RCT publications. A sample of 20 papers was examined by SSE. Table 8 shows the results of the evaluation.

Table 8. Comparing SSE with Kiritchenko *et al.*

| | **Kiritchenko *et al.* - Sentence Level** | **Kiritchenko *et al.* - Fragment Level** | **SSE** |
|---|---|---|---|
| **Precision** | 0.77 | 0.89 | 0.79 |
| **Recall** | 0.68 | 0.87 | 0.75 |
| **F-Score** | 0.72 | 0.88 | 0.77 |

Kiritchenko *et al.* distinguish two types of system performances.
*Sentence level* performance is concerned with the ability of the system to identify sentences carrying relevant parameter information - in this case the sample size.
*Fragment level* performance deals with the capability of extracting the correct infromation element from within the relevant sentence.
Comparing SSE with the efforts of Kiritchenko *et al.*, it can be stated that the performance of SSE lies inbetween the two performances of the compared system.
The more granular fragment level performance outperformes SSE. This observation may be due to the fact that the system of Kiritchenko *et al.* is specialized in parsing full-text RCT trials whereas SSE is designed to handle studies covering a wide range of research disciplines.
The general purpose application landscape of SSE poses a natural restriction on the specification of the patterns in stage 1 and therefore influencing subsequent performance of the other 2 stages.
If, for instance, SSE would only be applied in the environment of medical RCT papers than the extraction pipeline could be more strictly bound to structural circumstances encountered in RCTs and probably a higher overall performance could be achieved.

The comparison with the two competitors demonstrates the ability of SSE being able to keep up with comparable systems operating in the medical field.

## Discussion

We have introduced a method for automatically extractig the number of participants included in a study. The extraction is leveraged by a three-level pipelined architecture whose third stage utilizes manually crafted heuristics based on the pattern matching and subsequent filtering outcome of the first two stages.
This section discusses various purposes for which SSE can be used. One apparent application is to enhance PaperValidator - an open-source tool for automated assessment of valid statistics usage in study papers by including sample size analysis into the framework.

Despite the utilization of SSE within PaperValidator, it can also be employed as an independent module. For instance, it can drastically decrease the time consuming effort of manually collecting studies from the web enrolling a minimum number of participants. Especially within the field of evidence-based medicine (EBM), SSE can simplify the retrieval of relevant documents. EBM argues that making novel decisions about the care of patients should build upon the best medical research available at that time [17]. Necessarily, this implies the assessment of a large number of articles with a confident sample size being representative for the current examination. Within this scenario, SSE could assist in quickly filtering irrelevant studies.

Taking a step back and observing SSEs ability of operating on various kinds of research documents poses yet another interesting application.
Society in general can and should benefit from scientific discoveries found by researchers through conducting empirical studies. Nevertheless, caution is recommended with respect to fully trusting study outcomes only due to the fact that the investigation was performed by researchers. An additional factor in relying on experimental findings should consider the sample size included in the respective study.

Regardless of how innovative or groundbreaking a study finding might seem to be, it would possibly loose conviction in the light of a small number of investigated subjects.

Being able to rapidly scan numerous empirical studies from different disciplines with respect to the sample size information introduces an attractive utility.
For this purpose, SSE was run on a large sample of the corpus comprising studies of the journals outlined in the *Data Source* section. The intention behind this is to more or less shed light on the distribution of the sample sizes included in various studies.
The results of the following analysis have to be interpreted with caution. Due to the fact that SSE was evaluated on the ground truth with an F-Score of `0.71` (see Evaluation), its performance cannot be projected to the whole corpus the more so as the corpus likely contains studies not representative with regard to the ground truth or the training set mainly used to develop SSE.
Figure 3 shows the sample size distribution of the ground truth whereby studies with a *NL1* value > 1000 were excluded.
The chart demonstrates that the pool containing sample sizes of ≤ 20 is the largest one.

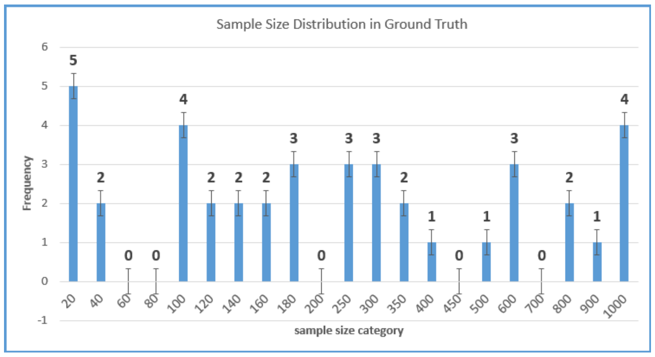


Fig. 3. Sample Size Distribution in the Ground Truth

This behaviour becomes clearer when processing a large sample of the corpus. As with the previous analysis, studies having a *NL1* value > 1000 were discarded. The final dataset contains roughly 3000 papers covering all investigated journals. The distribution of the sample sizes is outlined in the histogram of Figure 4.

The presented overview is in no manner representative with the actual values in the corpus. It only demonstrates the application of SSE on a large set of empirical studies.
Nevertheless, an interesting point can be made about the distribution. According to the histogram, approximately 40% of the studies exhibit a sample size of ≤ 20. This finding may call reasonable research into question.

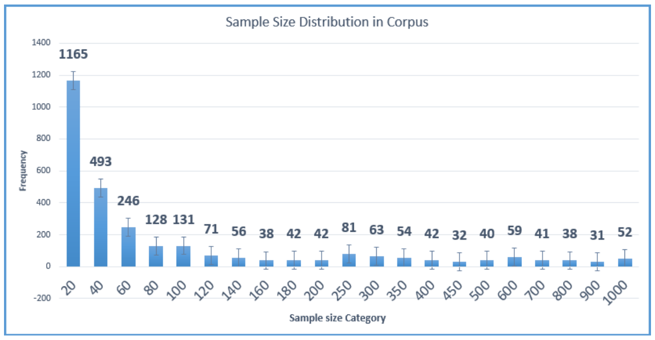


Fig. 4. Sample Size Distribution in the Corpus

## Limitations

The design and purpose of SSE accounts for several limitations. The framework at its current state is primarily designed to be applicable on studies describing one experiment and thus having at most one *NL1* sample size. Studies documenting several experiments were discarded from the evaluation.
Furthermore, only studies having at least either a *NL1* or *L2* sample size were considered in the development as well as in the evaluation.

## Conclusion

The present thesis outlined an approach on automatically extracting the most important types of sample sizes from a full-text study paper. The methodology used differentiates in many aspects from concepts discussed in the *Related Work* section.
While recent literature in this field mostly employs NLP techniques in combination with machine learning, SSE follows a more conservative approach. SSE is designed in a modular fashion clearly seperating the single stages in the pipeline from each other.
Therefore it can be extended at any time (*e.g.* by adding new regular expressions in stage 1 or refining the case-specific heuristics).
Alltogether, SSE proposes a promising methodology on retrieving sample size information from research in general.

## Acknowledgement

I would like to show my gratitude especially to Patrick de Boer for his valuable feedbacks on the development ideas as well as his patient support.

## References

[1] D. J. Biau, S. KernÃĺis, and R. Porcher. *Statistics in Brief: The Importance of Sample Size in the Planning and Interpretation of Medical Research.* Clinical Orthopaedics and Related Research, 2008.

[2] V. S. Binu, S. S. Mayya, and M. Dhar. *Some basic aspects of statistical methods and sample size determination in health science research.* Ayu, 2014.

[3] K. Cassidy and Y. Hui. A system for extracting study design parameters from nutritional genomics abstracts. *Journal of Integrative Bioinformatics*, 2013.

[4] Pierre Charles, Bruno Giraudeau, Agnes Dechartres, Gabriel Baron, and Philippe Ravaud. Reporting of sample size calculation in randomised controlled trials: review. *BMJ: British Medical Journal*, 338(7705):1256–1259, 2009.

[5] Grace Yuet-Chee Chung. Towards identifying intervention arms in randomized controlled trials: Extracting coordinating constructions. *Journal of Biomedical Informatics*, 42(5):790 – 800, 2009. Biomedical Natural Language Processing.

[6] H. Cunninghan. Information extraction, automatic. *Encyclopedia of Language and Linguistics*, 2005.

[7] B. De Bruijn, S. Carini, S. Kiritchenko, J. Martin, and I. Sim. Automated information extraction of key trial design elements from clinical trial publications. *AMIA Annual Symposium Proceedings*, pages 141–145, 2008.

[8] Marie J Hansen, Nana Rasmussen, and Grace Chung. A method of extracting the number of trial participants from abstracts describing randomized controlled trials. *Journal of Telemedicine and Telecare*, 14(7):354–358, 2008. PMID: 18852316.

[9] Kazuo Hara and Yuji Matsumoto. Extracting clinical trial design information from medline abstracts. *New Generation Computing*, 25(3):263–275, 2007.

[10] Maurits Kaptein and Judy Robertson. Rethinking statistical analysis methods for chi. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 1105–1114, New York, NY, USA, 2012. ACM.

[11] Svetlana Kiritchenko, Berry de Bruijn, Simona Carini, Joel Martin, and Ida Sim. Exact: automatic extraction of clinical trial characteristics from journal publications. *BMC Medical Informatics and Decision Making*, 10(1):56, 2010.

[12] Robert V. Krejcie and Daryle W. Morgan. Determining sample size for research activities. *Educational and Psychological Measurement*, 30(3):607–610, 1970.

[13] Russell V. Lenth. Some practical guidelines for effective sample size determination. *The American Statistician*, 55(3):187–193, 2001.

[14] R. Manuel and de B. Patrick. Papervalidator - towards the automated validation of statistics in publications. https://github.com/pdeboer/PaperValidator, 2016.

[15] J. Gail Neely, Ron J. Karni, Samuel H. Engel, Patrick L. Fraley, Brian Nussenbaum, and Randal C. Paniello. Practical guides to understanding sample size and minimal clinically important difference (mcid). *OtolaryngologyâĂŞHead and Neck Surgery*, 136(1):14–18, 2007. PMID: 17210326.

[16] Robert A. Parker and Nancy G. Berman. Sample size: More than calculations. *The American Statistician*, 57(3):166–170, 2003.

[17] David L Sackett, William M C Rosenberg, J A Muir Gray, R Brian Haynes, and W Scott Richardson. Evidence based medicine: what it is and what it isn't. *BMJ*, 312(7023):71–72, 1996.

[18] Rong Xu, Yael Garten, Kaustubh S. Supekar, Amar K. Das, Russ B. Altman, and Alan M. Garber. Extracting subject demographic information from abstracts of randomized clinical trial reports. In Klaus A. Kuhn, James R. Warren, and Tze-Yun Leong, editors, *MedInfo*, volume 129 of *Studies in Health Technology and Informatics*, pages 550–554. IOS Press, 2007.

## Appendix

Following section describes the inherent heuristics for each of the 2-16 cases.

| **Case 2** | **OL1** | **NL1** | **!NL1** | **L2** |
|---|---|---|---|---|

- Case (2a): SubArrayMap $\neq \emptyset \vee$ SubSetMap $\neq \emptyset$ such that:
  - SubArrayMap.keySet.contains($\mathtt{NL1}_i$)
  - SubSetMap.keySet.contains($\mathtt{NL1}_i$)

  $\Longrightarrow$ Add $\mathtt{NL1}_i$ to *NL1_potential* and its subArray resp. subSet to *L2_potential*
- Case (2b): $\exists$ subSet of the *L2* such that:
  - sum(subSet) = $\mathtt{NL1}_i$

  $\Longrightarrow$ Add $\mathtt{NL1}_i$ to *NL1_potential* and subset(L2) to *L2_potential*
- Case (2c): $\exists$ $\mathtt{NL1}_i$ = max(pool *NL1*) such that:
  - SubArrayMap.keySet.contains ($\mathtt{NL1}_i$) or
  - SubSetMap.keySet.contains($\mathtt{NL1}_i$)

  $\Longrightarrow$ Add $\mathtt{NL1}_i$ to *NL1_potential* and its subArray resp. subSet to *L2_potential*
- Case (2d): SubArrayMap $\neq \emptyset \vee$ SubSetMap $\neq \emptyset \Longrightarrow$ Add max(SubArrayMap.keySet) or max(SubSetMap.keySet) to *NL1_potential* and the subArray resp. subSet values to *L2_potential*

| **Case 3** | **OL1** | **NL1** | **!NL1** | **L2** |
|---|---|---|---|---|

- Case (3a): $\exists$ $\mathtt{OL1}_i$ $\wedge$ subset of $\mathtt{!NL1}$ such that:
  - $\mathtt{OL1}_i$ - subset($\mathtt{!NL1}$) = $\mathtt{TS}_i$
  - SubArrayMap.keySet.contains($\mathtt{TS}_i$) or
  - SubSetMap.keySet.contains($\mathtt{TS}_i$)

$\Longrightarrow$ Add $\mathtt{TS}_i$ to *NL1_potential* and its subArray resp. subSet to *L2_potential*

- Case (3b): $\exists$ subSet of the *L2* such that:
  - sum(subSet) = $\mathtt{TS}_i$

  $\Longrightarrow$ Add $\mathtt{TS}_i$ to *NL1_potential* and subset(L2) to *L2_potential*
- Rollback: $\Longrightarrow$ Add max(Pool *OL1*) to *OL1_potential*

| Case 4 | 🟩 OL1 | 🟩 NL1 | 🟥 !NL1 | 🟩 L2 |
|---|---|---|---|---|

- Case (4a): SubArrayMap $\neq \emptyset \vee$ SubSetMap $\neq \emptyset$ such that:
  - SubArrayMap.keySet.contains($\mathtt{NL1}_i$)
  - SubSetMap.keySet.contains($\mathtt{NL1}_i$)

  $\Longrightarrow$ Add $\mathtt{NL1}_i$ to *NL1_potential* and its subArray resp. subSet to *L2_potential*
- Case (4b): SubArrayMap $\neq \emptyset \vee$ SubSetMap $\neq \emptyset \Longrightarrow$ Add max(SubArrayMap.keySet) or max(SubSetMap.keySet) to *NL1_potential* and the subArray resp. subSet values to *L2_potential*
- Case (4c): $\exists$ subSet of the *L2* such that:
  - sum(subSet) = $\mathtt{NL1}_i$

  $\Longrightarrow$ Add $\mathtt{NL1}_i$ to *NL1_potential* and subset(L2) to *L2_potential*
- Case (4d): $\exists\, \mathtt{NL1}_i$ = max(pool *NL1*) such that:
  - SubArrayMap.keySet.contains ($\mathtt{NL1}_i$) or
  - SubSetMap.keySet.contains($\mathtt{NL1}_i$)

  $\Longrightarrow$ Add $\mathtt{NL1}_i$ to *NL1_potential* and its subArray resp. subSet to *L2_potential*
- Rollback Pool *OL1* such that:
  - OL1.filter($\mathtt{OL1}_i$ > max(*NL1*)) $\neq \emptyset$

  $\Longrightarrow$ Add max(*OL1*) to *OL1_potential*

| Case 5 | 🟩 OL1 | 🟩 NL1 | 🟩 !NL1 | 🟥 L2 |
|---|---|---|---|---|

- Case (5a): $\exists\, \mathtt{OL1}_i \wedge$ subset of $\mathtt{!NL1}$ such that:
  - $\mathtt{OL1}_i$ - subset($\mathtt{!NL1}$) = $\mathtt{NL1}_i$
  - SubArrayMap.keySet.contains($\mathtt{NL1}_i$) or
  - SubSetMap.keySet.contains($\mathtt{NL1}_i$)

  $\Longrightarrow$ Add $\mathtt{NL1}_i$ to *NL1_potential* and its subArray resp. subSet to *L2_potential*
- Case (5b): SubArrayMap $\neq \emptyset \vee$ SubSetMap $\neq \emptyset$ such that:
  - SubArrayMap.keySet.contains($\mathtt{NL1}_i$)
  - SubSetMap.keySet.contains($\mathtt{NL1}_i$)

  $\Longrightarrow$ Add $\mathtt{NL1}_i$ to *NL1_potential* and its subArray resp. subSet to *L2_potential*
- Case (5c): $\exists\, \mathtt{NL1}_i$ = max(pool *NL1*) such that:
  - SubArrayMap.keySet.contains ($\mathtt{NL1}_i$) or
  - SubSetMap.keySet.contains($\mathtt{NL1}_i$)

  $\Longrightarrow$ Add $\mathtt{NL1}_i$ to *NL1_potential* and its subArray resp. subSet to *L2_potential*
- Rollback Pool *OL1* such that:
  - OL1.filter($\mathtt{OL1}_i$ > max(*NL1*)) $\neq \emptyset$

  $\Longrightarrow$ Add max(*OL1*) to *OL1_potential*

| Case 6 | 🟥 OL1 | 🟥 NL1 | 🟩 !NL1 | 🟩 L2 |
|---|---|---|---|---|

- Case (6a): $\exists$ subSet of the *L2* such that:
  - sum(subSet) = $\mathtt{TS}_i$

  $\Longrightarrow$ Add $\mathtt{TS}_i$ to *NL1_potential* and subset(L2) to *L2_potential*

| Case 7 | 🟥 OL1 | 🟩 NL1 | 🟥 !NL1 | 🟩 L2 |
|---|---|---|---|---|

- Case (7a): SubArrayMap $\neq \emptyset \vee$ SubSetMap $\neq \emptyset$ such that:
  - SubArrayMap.keySet.contains($\mathtt{NL1}_i$)
  - SubSetMap.keySet.contains($\mathtt{NL1}_i$)

  $\Longrightarrow$ Add $\mathtt{NL1}_i$ to *NL1_potential* and its subArray resp. subSet to *L2_potential*
- Case (7b): $\exists$ subSet of the *L2* such that:
  - sum(subSet) = $\mathtt{NL1}_i$

  $\Longrightarrow$ Add $\mathtt{NL1}_i$ to *NL1_potential* and subset(L2) to *L2_potential*
- Case (7c): $\exists\, \mathtt{NL1}_i$ = max(pool *NL1*) such that:
  - SubArrayMap.keySet.contains ($\mathtt{NL1}_i$) or
  - SubSetMap.keySet.contains($\mathtt{NL1}_i$)

  $\Longrightarrow$ Add $\mathtt{NL1}_i$ to *NL1_potential* and its subArray resp. subSet to *L2_potential*
- Case (7d): SubArrayMap $\neq \emptyset \vee$ SubSetMap $\neq \emptyset \Longrightarrow$ Add max(SubArrayMap.keySet) or max(SubSetMap.keySet) to *NL1_potential* and the subArray resp. subSet values to *L2_potential*

| Case 8 | 🟥 OL1 | 🟩 NL1 | 🟩 !NL1 | 🟥 L2 |
|---|---|---|---|---|

- Case (8a): SubArrayMap $\neq \emptyset \vee$ SubSetMap $\neq \emptyset$ such that:
  - SubArrayMap.keySet.contains($\mathtt{NL1}_i$)

  - SubSetMap.keySet.contains($\mathtt{NL1}_i$)

  $\Longrightarrow$ Add $\mathtt{NL1}_i$ to *NL1_potential* and its subArray resp. subSet to *L2_potential*

- Case (8b): $\exists\, \mathtt{NL1}_i$ = max(pool *NL1*) such that:
  - SubArrayMap.keySet.contains ($\mathtt{NL1}_i$) or
  - SubSetMap.keySet.contains($\mathtt{NL1}_i$)

  $\Longrightarrow$ Add $\mathtt{NL1}_i$ to *NL1_potential* and its subArray resp. subSet to *L2_potential*

| | | | | |
|---|---|---|---|---|
| **Case 9** | 🟩 **OL1** | 🟥 **NL1** | 🟥 **!NL1** | 🟩 **L2** |

- Case (9a): $\exists$ subSet of the *L2* such that:
  - sum(subSet) = $\mathtt{TS}_i$

  $\Longrightarrow$ Add $\mathtt{TS}_i$ to *NL1_potential* and subset(L2) to *L2_potential*
- Case (9b): SubArrayMap $\neq \emptyset \vee$ SubSetMap $\neq \emptyset \Longrightarrow$ Add max(SubArrayMap.keySet) or max(SubSetMap.keySet) to *NL1_potential* and the subArray resp. subSet values to *L2_potential*
- Rollback: $\Longrightarrow$ Add max(Pool *OL1*) to OL1_potential

| | | | | |
|---|---|---|---|---|
| **Case 10** | 🟩 **OL1** | 🟥 **NL1** | 🟩 **!NL1** | 🟥 **L2** |

- Case (10a): $\exists\, \mathtt{OL1}_i \wedge$ subset of $\mathtt{!NL1}$ such that:
  - $\mathtt{OL1}_i$ - subset($\mathtt{!NL1}$) = $\mathtt{NL1}_i$
  - SubArrayMap.keySet.contains($\mathtt{NL1}_i$) or
  - SubSetMap.keySet.contains($\mathtt{NL1}_i$)

  $\Longrightarrow$ Add $\mathtt{NL1}_i$ to *NL1_potential* and its subArray resp. subSet to *L2_potential*
- Case (10b): SubArrayMap $\neq \emptyset \vee$ SubSetMap $\neq \emptyset \Longrightarrow$ Add max(SubArrayMap.keySet) or max(SubSetMap.keySet) to *NL1_potential* and the subArray resp. subSet values to *L2_potential*
- Rollback: $\Longrightarrow$ Add max(Pool *OL1*) to OL1_potential

| | | | | |
|---|---|---|---|---|
| **Case 11** | 🟩 **OL1** | 🟩 **NL1** | 🟥 **!NL1** | 🟥 **L2** |

- Case (11a): $\exists\, \mathtt{NL1}_i$ = max(pool *NL1*) such that:
  - SubArrayMap.keySet.contains ($\mathtt{NL1}_i$) or
  - SubSetMap.keySet.contains($\mathtt{NL1}_i$)

  $\Longrightarrow$ Add $\mathtt{NL1}_i$ to *NL1_potential* and its subArray resp. subSet to *L2_potential*
- Rollback Pool *OL1* such that:
  - OL1.filter($\mathtt{OL1}_i$ > max(*NL1*)) $\neq \emptyset$

  $\Longrightarrow$ Add max(*OL1*) to *OL1_potential*

| | | | | |
|---|---|---|---|---|
| **Case 12** | 🟥 **OL1** | 🟥 **NL1** | 🟥 **!NL1** | 🟩 **L2** |

- Case (12a): $\exists$ subSet of the *L2* such that:
  - sum(subSet) = $\mathtt{TS}_i$

  $\Longrightarrow$ Add $\mathtt{TS}_i$ to *NL1_potential* and subset(L2) to *L2_potential*
- Case (12b): SubArrayMap $\neq \emptyset \vee$ SubSetMap $\neq \emptyset \Longrightarrow$ Add max(SubArrayMap.keySet) or max(SubSetMap.keySet) to *NL1_potential* and the subArray resp. subSet values to *L2_potential*

| | | | | |
|---|---|---|---|---|
| **Case 13** | 🟥 **OL1** | 🟥 **NL1** | 🟩 **!NL1** | 🟥 **L2** |

- Case (13a): SubArrayMap $\neq \emptyset \vee$ SubSetMap $\neq \emptyset \Longrightarrow$ Add max(SubArrayMap.keySet) or max(SubSetMap.keySet) to *NL1_potential* and the subArray resp. subSet values to *L2_potential*

| | | | | |
|---|---|---|---|---|
| **Case 14** | 🟥 **OL1** | 🟩 **NL1** | 🟥 **!NL1** | 🟥 **L2** |

- Case (14a): $\exists\, \mathtt{NL1}_i$ = max(pool *NL1*) such that:
  - SubArrayMap.keySet.contains ($\mathtt{NL1}_i$) or
  - SubSetMap.keySet.contains($\mathtt{NL1}_i$)

  $\Longrightarrow$ Add $\mathtt{NL1}_i$ to *NL1_potential* and its subArray resp. subSet to *L2_potential*
- Case (14b): SubArrayMap $\neq \emptyset \vee$ SubSetMap $\neq \emptyset \Longrightarrow$ Add max(SubArrayMap.keySet) or max(SubSetMap.keySet) to *NL1_potential* and the subArray resp. subSet values to *L2_potential*

| | | | | |
|---|---|---|---|---|
| **Case 15** | 🟩 **OL1** | 🟥 **NL1** | 🟥 **!NL1** | 🟥 **L2** |

- Case (15a): SubArrayMap $\neq \emptyset \vee$ SubSetMap $\neq \emptyset \Longrightarrow$ Add max(SubArrayMap.keySet) or max(SubSetMap.keySet) to *NL1_potential* and the subArray resp. subSet values to *L2_potential*
- Rollback: $\Longrightarrow$ Add max(Pool *OL1*) to OL1_potential

| | | | | |
|---|---|---|---|---|
| **Case 16** | 🟥 **OL1** | 🟥 **NL1** | 🟥 **!NL1** | 🟥 **L2** |

- Case (16a): SubArrayMap $\neq \emptyset \vee$ SubSetMap $\neq \emptyset \Longrightarrow$ Add max(SubArrayMap.keySet) or max(SubSetMap.keySet) to *NL1_potential* and the subArray resp. subSet values to *L2_potential*