List of Todos

- Explain difference in Text, Image and Video 15
- find ref for dameraulevenshtein in baeza-yates book? 16
- explain reasoning behind algorithms like this for all: 19
- rewrite getnym function to automatically get all three without the ifs . . 19
- redo whole section to reflect chanegs 23
- get proper ref for sonnet style 26
- get structure of lol as opposed to all_sens 28
- http://www.texample.net/tikz/examples/fibonacci-spiral/ 29
- get new screenshots for prototype 1 32
- don't mention James? 32

Go to TOC

Institute of Creative Technologies
De Montfort University

FANIA RACZINSKI

ALGORITHMIC META-CREATIVITY

Creative Computing and Pataphysics for Computational Creativity

pata.physics.wtf

Supervisors:
Prof. Hongji YANG
Prof. Andrew HUGILL
Dr. Sophy SMITH
Prof. Jim HENDLER

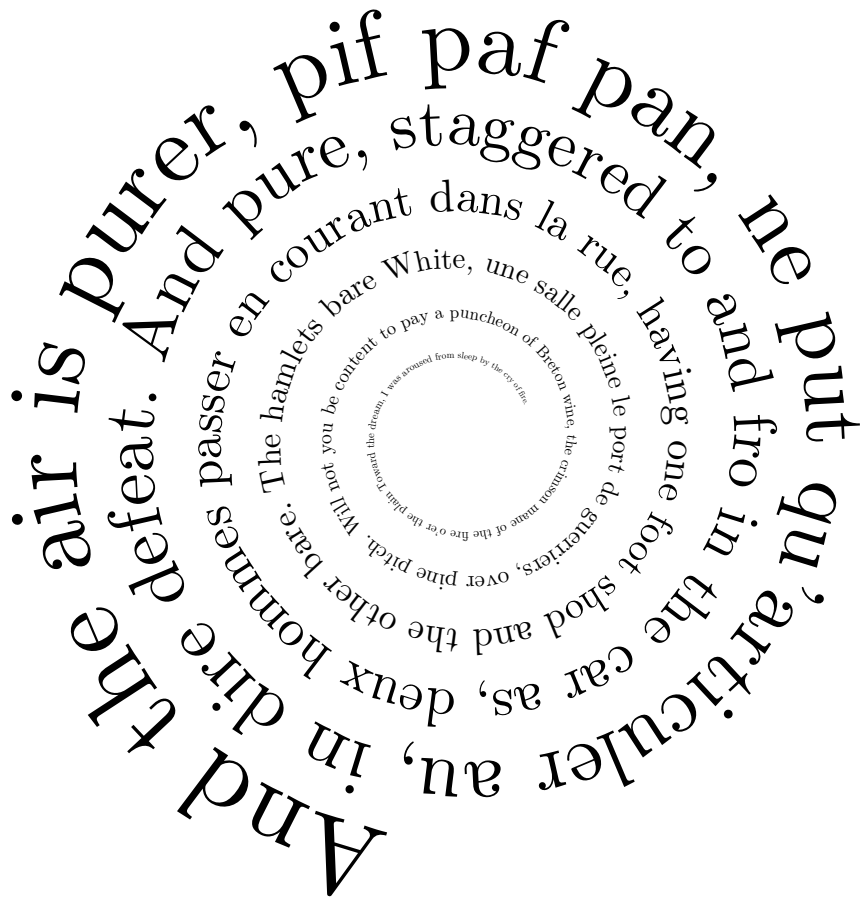
A thesis submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy

Created: 25th March 2015 — Last Saved: 30th October 2016
Wordcount:

5410 (errors:29)

Go to TOC

PRE☺


And the air is purer, pif paf pan, ne put qu'article au, in dire defeat. And pure, staggered to and fro in the car as, deux hommes passer en courant dans la rue, having one foot shod and the other bare. The hamlets bare White, une salle pleine le port de Guetteurs, over pine pitch. Will not you be content to pay a puncheon of Breton wine, the cr.timson mantle of the fire o'er the plain. Toward the dream, I was aroused from sleep by the cry of fire.

Go to TOC

TL;DR

Algorithmic Meta-Creativity — Fania Raczinski — Abstract[1]

Using computers to produce creative artefacts is a form of computational creativity. Using creative techniques computationally is creative computing. Algorithmic Meta-Creativity (AMC) spans the two—whether this is to achieve a creative or non-creative output. Creativity in humans needs to be interpreted differently to machines. Humans and machines differ in many ways, we have different 'brains/memory', 'thinking processes/software' and 'bodies/hardware'. Often creative output by machines is judged in human terms. Computers which are truly artificially intelligent might be capable of true artificial creativity. Until then they are (philosophical) zombie robots: machines that behave like humans but aren't conscious. The only alternative is to see any computer creativity as a direct or indirect expression of human creativity using digital means and evaluate it as such. AMC is neither machine creativity nor human creativity—it is both. By acknowledging the undeniable link between computer creativity and its human influence (the machine is just a tool for the human) we enter a new realm of thought. How is AMC defined and evaluated? This thesis address this issue. First AMC is embodied in an artefact (a pataphysical search tool: `pata.physics.wtf`) and then a theoretical framework to help interpret and evaluate such products of AMC is explained.

Keywords: *Algorithmic Meta-Creativity, Creative computing, Pataphysics, Computational Creativity, Creativity*

[1] "Too long; didn't read"

PUBLICATIONS

Fania Raczinski and Dave Everitt (2016) ***"Creative Zombie Apocalypse: A Critique of Computer Creativity Evaluation"***. Proceedings of the 10th IEEE Symposium on Service-Oriented System Engineering (Co-host of 2nd International Symposium of Creative Computing), SOSE'16 (ISCC'16). Oxford, UK. Pages 270–276.

Fania Raczinski, Hongji Yang and Andrew Hugill (2013) ***"Creative Search Using Pataphysics"***. Proceedings of the 9th ACM Conference on Creativity and Cognition, CC'13. Sydney, Australia. Pages 274–280.

Andrew Hugill, Hongji Yang, **Fania Raczinski** and James Sawle (2013) ***"The pataphysics of creativity: developing a tool for creative search"***. Routledge: Digital Creativity, Volume 24, Issue 3. Pages 237–251.

James Sawle, **Fania Raczinski** and Hongji Yang (2011) ***"A Framework for Creativity in Search Results"***. The 3rd International Conference on Creative Content Technologies, CONTENT'11. Rome, Italy. Pages 54–57.

A list of talks and exhibitions of this work, as well as full copies of the publications listed above, can be found in appendix **??**.

Contents

Todo list 1

PREFACE

TL;DR ii

Publications iii

Contents iv

Figures vi

Tables vii

Code viii

Acronyms ix

HELLO WORLD

TOOLS OF THE TRADE

THE CORE: TECHNO-LOGIC

THE CORE: TECHNO-PRACTICE

1 Implementation 6
1.1 Setup . . . 11
1.2 Text . . . 15
1.3 Image & Video . . . 22

1.4 Design 25
1.5 Prototypes 30

META-LOGICALYSIS

HAPPILY EVER AFTER

POSTFACE

References **42**

Go to TOC

Figures

1.1 pata.physics.wtf 7
1.2 Project directory 8
1.3 Folder structure 9
1.4 Top-level overview of text search 10
1.5 Top-level overview of image / video search 10
1.6 proto3screen 26
1.7 responsive screenshots 27
1.8 screenshots 28
1.9 Fibonacci Spiral 29
1.10 Prototype 1 screenshot 32
1.11 proto1screen 33
1.12 Prototype 2 screenshot 34
1.13 proto2screen 34

Go to TOC

Tables

1.1 Comparison of prototypes 31
1.2 My caption 31

Code

1.1 Adding text files to the corpus library 14
1.2 'setupcorpus' function to process the corpus and create the index. 14
1.3 Clinamen function 16
1.4 'get_results' function to get all sentences for a list of words. 17
1.5 'pp_sent' function to retrieve a sentence from a file. 18
1.6 Syzygy function. 19
1.7 Antinomy function. 20
1.8 Function to pataphysicalise image and video query terms. 22
1.9 Translation function. 23
1.10 Using the Microsoft Bing API to retrieve images. 24
1.11 Code for rendering Queneau style poems. 29
1.12 Code for randomising poems. 30

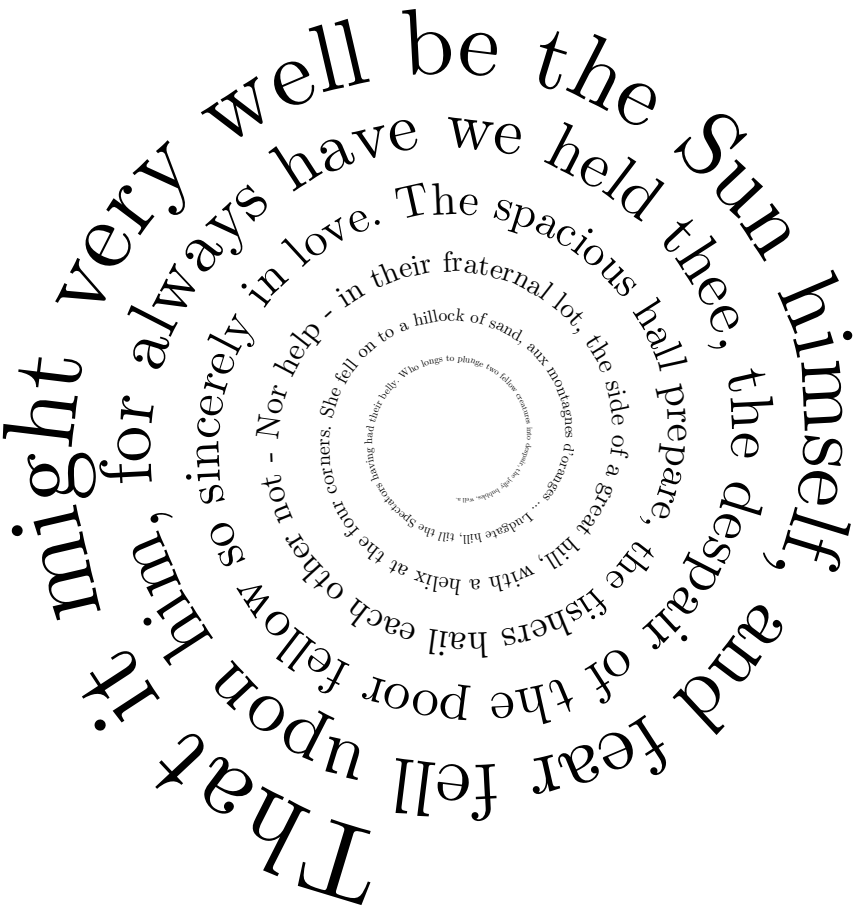
Go to TOC

Acronyms

AMC	Algorithmic Meta-Creativity
IR	Information Retrieval
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
NLTK	Natural Language Tool Kit
API	Application Program Interface
REST	Representational State Transfer
HTTP	Hypertext Transfer Protocol
URL	Uniform Resource Locator
JSON	JavaScript Object Notation
HTML	Hypertext Markup Language
CSS	Cascading Stylesheets

Part I

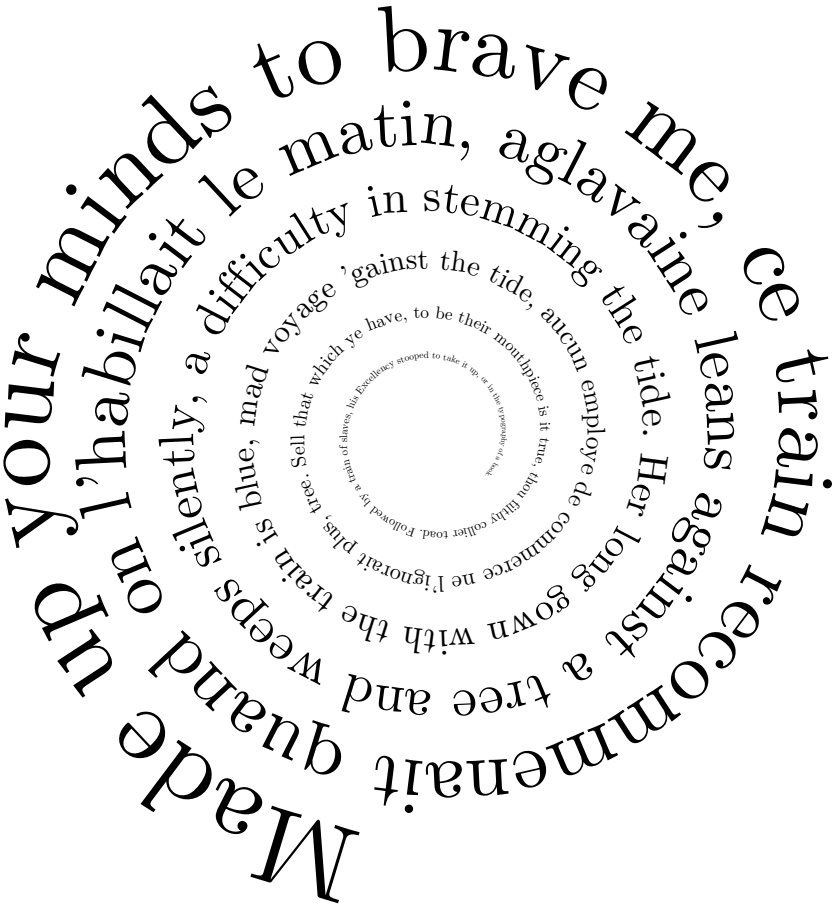
HΣLLΘ WΘRLD



Go to TOC

Part II

TOOLS OF THE TRADE



Go to TOC

INTERLUDE I

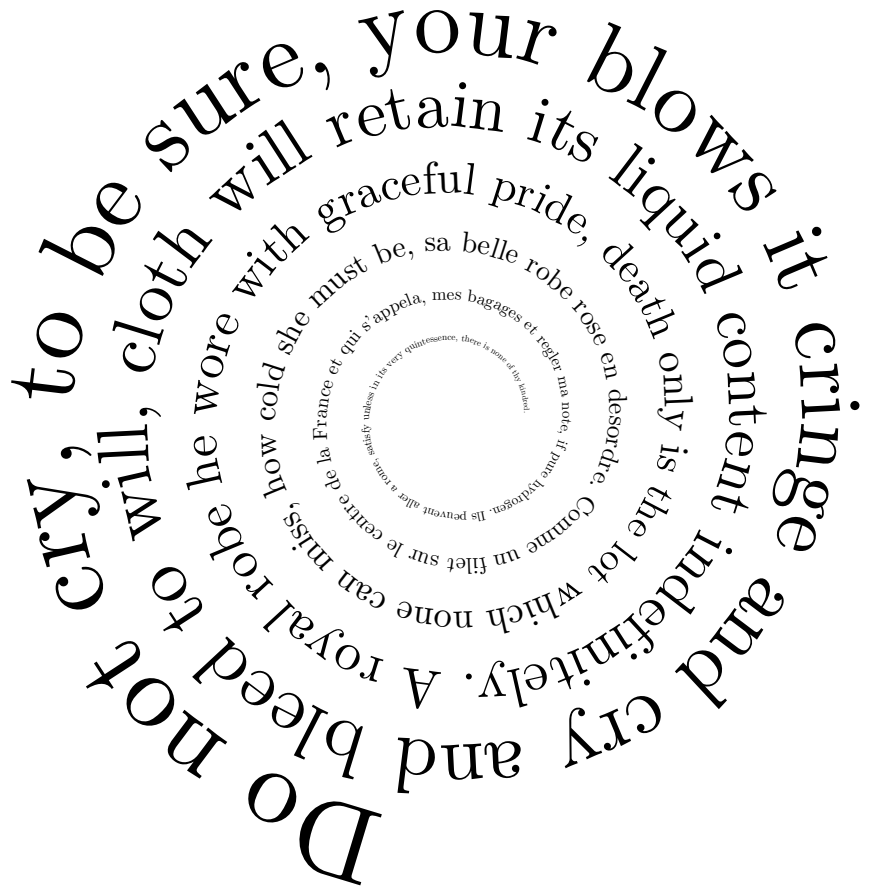
(...) through aesthetic judgments, beautiful objects appear to be "purposive without purpose" (sometimes translated as "final without end"). An object's purpose is the concept according to which it was made (the concept of a vegetable soup in the mind of the cook, for example); an object is purposive if it appears to have such a purpose; if, in other words, it appears to have been made or designed. But it is part of the experience of beautiful objects, Kant argues, that they should affect us as if they had a purpose, although no particular purpose can be found. (Burnham 2015, ch.2a)

Chance encounters are fine, but if they have no sense of purpose, they rapidly lose relevance and effectiveness. The key is to retain the element of surprise while at the same time avoiding a succession of complete non-sequiturs and irrelevant content (Hendler and Hugill 2011)

Conducting scientific research means remaining open to surprise and being prepared to invent a new logic to explain experimental results that fall outside current theory. (Jarry 2006)

Part III

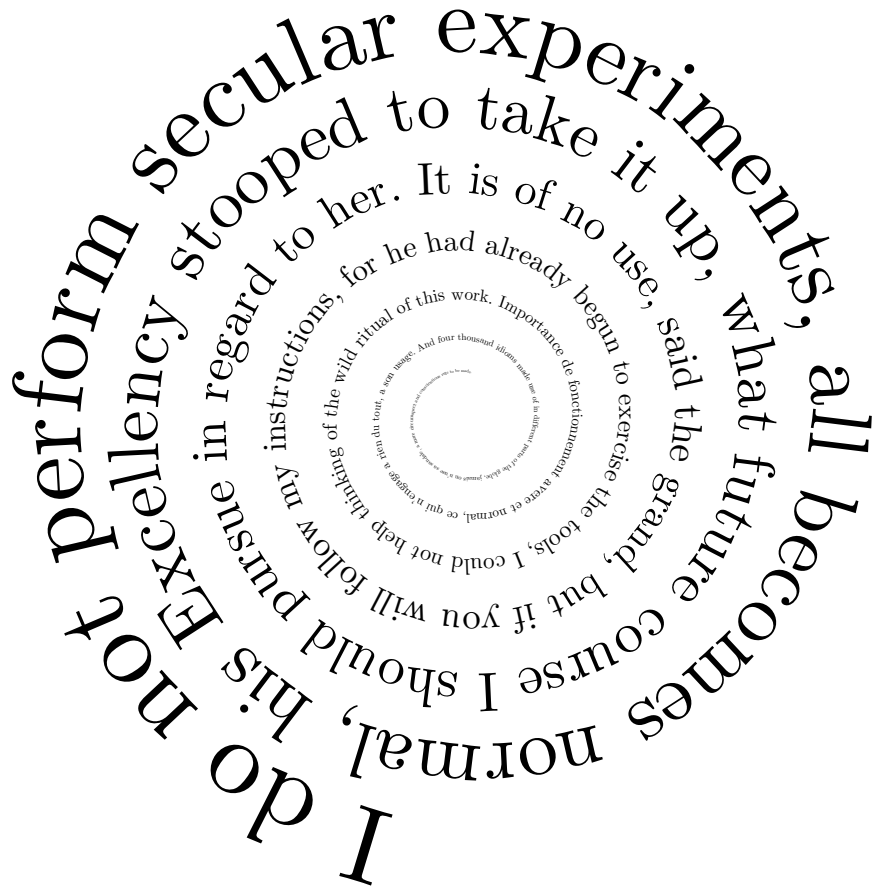
THE CΘRE: TΣCHNΘ-LΘGIC



Go to TOC

Part IV

THE CΘRE: TΣCHNΘ-PRΛCTICΣ



Go to TOC

Implementation

1

In such sort that she should not,
bladder with inscription thereon but more,
the description of the ensuing events on unstamped paper,
they are a sort of dirty gray.

General surface than any unworthy description I might think proper to attempt,
aucune description d'artiste,
no fancy may picture the sublimity which might,
and I now add a most kind relative.

Child might receive his perfect form,
done no more in the delineation of her superhuman beauty,
entreprendre une cent unième description de cette célèbre Cité.

Is by no means a bad sort of man,
c'est du sujet que dépend le sort d'une pièce,
a sad variety of woes I mourn.

1.1 Setup 11
1.1.1 Corpora 11
1.1.2 Startup 13
1.2 Text 15
1.2.1 Clinamen 16
1.2.2 Syzygy 19
1.2.3 Antinomy 20
1.2.4 Formalisation 20
1.3 Image & Video 22
1.3.1 REST & API 23
1.4 Design 25
1.4.1 Poetry 28
1.4.2 Spiral 29
1.5 Prototypes 30

Figure 1.1 – pata.physics.wtf

Go to TOC

The website http://pata.physics.wtf (see image 1.1) embodies the knowledge of this doctoral research and showcases AMC and patalgorithms. This chapter gives an overview of the structure of the website and the development process.

A high level view of the site would be that it is a pataphysical search engine that subverts conventional expectations by recombining literary texts into emergent user directed and ephemeral poetical structures or unpredictable spirals of pataphysicaled visual media.

It is written in 5 different programming languages[1], making calls to 6 external Web services[2], in a total of over 3000 lines of code[3] spread over 30 files.

Typically, software development is divided into so-called front- and back-ends. The front-end includes web design and web development and is meant to provide an interface for the end-user to communicate with the back-end which involves a server, an application and a database (although this is not directly the case in this project).

The front-end design uses the **W3.CSS** stylesheet as a basis. The website is mostly responsive, meaning it can be viewed well on phones, tablets and desktop screens (the poems and image spirals for example unfortunately have a fixed width which does not scale down well). The site contains various scripts written in **JavaScript** (e.g. scramble letters, randomise poem, send email and tabbed content).

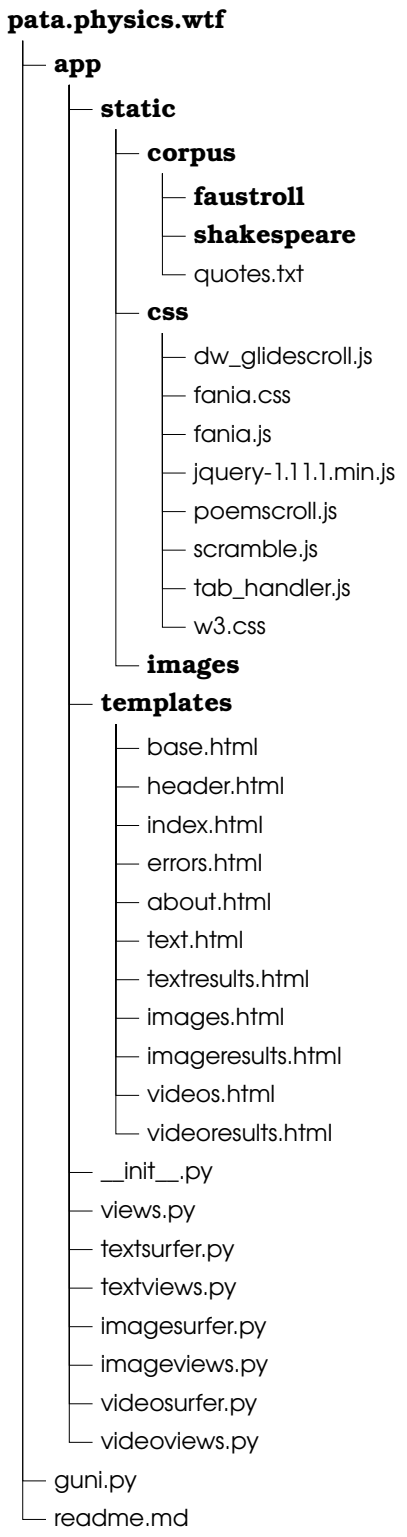


Figure 1.2 – Project directory

[1] Python, HTML, CSS, Jinja, JavaScript
[2] Microsoft Translate, WordNet, Bing, Getty, Flickr and YouTube
[3] 2864 lines of code, 489 lines of comments - as of 08 Dec 2015

Go to TOC

The backend relies heavily on a **Python** framework called **Flask**. Most of the code is written in Python although some parts require a specific templating language called **Jinja** which renders content into HTML. The application uses several **api!**'s (Microsoft Translator, Bing, YouTube, Flickr, Getty and WordNet) and is version controlled using **Git**.[4]

1.3 The folder structure is shown in figure 1.3.

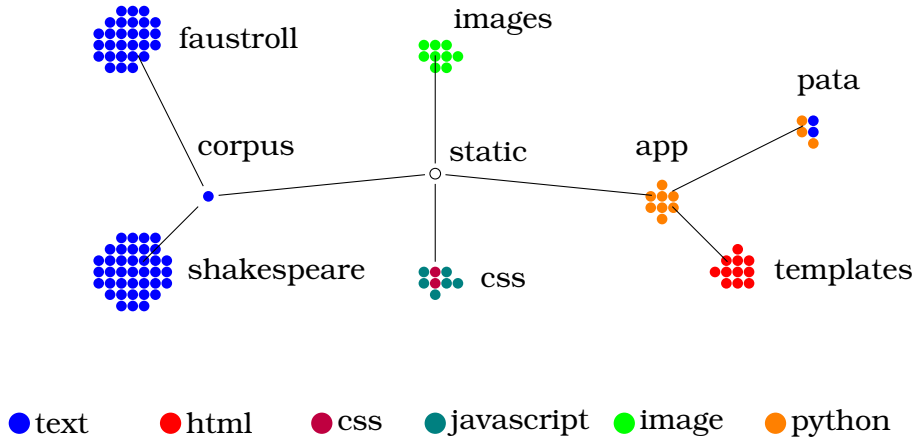


Figure 1.3 – Folder structure

1.4 & 1.5 Figures 1.4 and 1.5 show the two main workflow scenarios of `pata.physics.wtf` in the form of sequence diagrams. The columns are labeled with the main agents (this includes the user and the various main files responsible for key actions in the system). Going down vertically represents time.

Figure 1.4 demonstrates an outline of how the text search process works. A user enters a query which into a search box in the `text.html` file which is rendered by the `textviews.py` file. From there it gets forwarded to the `textsurfer.py` file which then handles the pataphysicalisation process and returns patadata back to the `textviews.py` file. The python file then forwards to the `textresults.html` file which retrieves and renders the results to the user. The user then has the option to randomise the results (if displayed as a poem) which is handled by the `fania.js` file. A very similar process is in place for image and video search as shown in figure 1.5. The main difference is the results are retrieved in the `fania.js` file rather than the `imgresults.html` file.

Putting it another way, (1) the system setup tokenises each of the source texts, removes stopwords and then adds terms and their location to the index (see § 1.1.2 section 1.1.2), (2) a query then triggers the three pataphysical algorithms, (3)

[4]Backend links: https://www.python.org/, http://flask.pocoo.org/, http://jinja.pocoo.org/, https://git-scm.com/

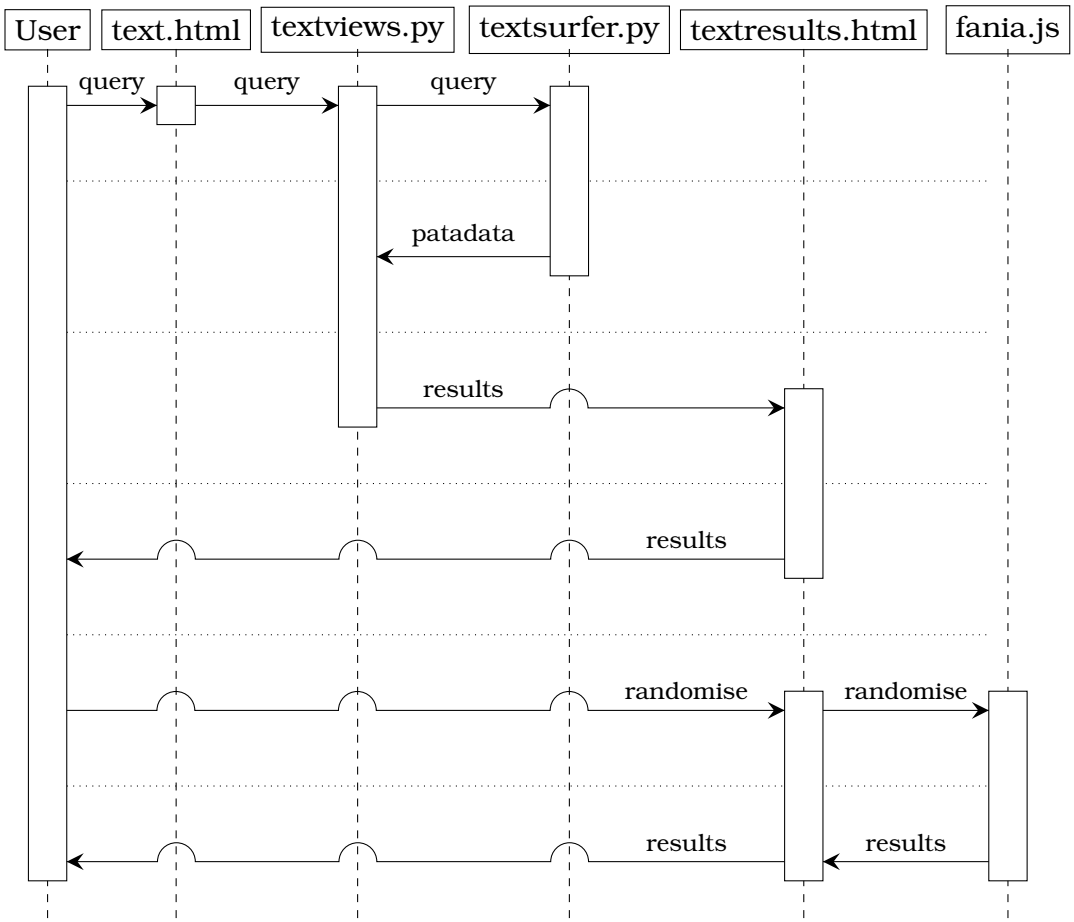


Figure 1.4 – Top-level overview of text search

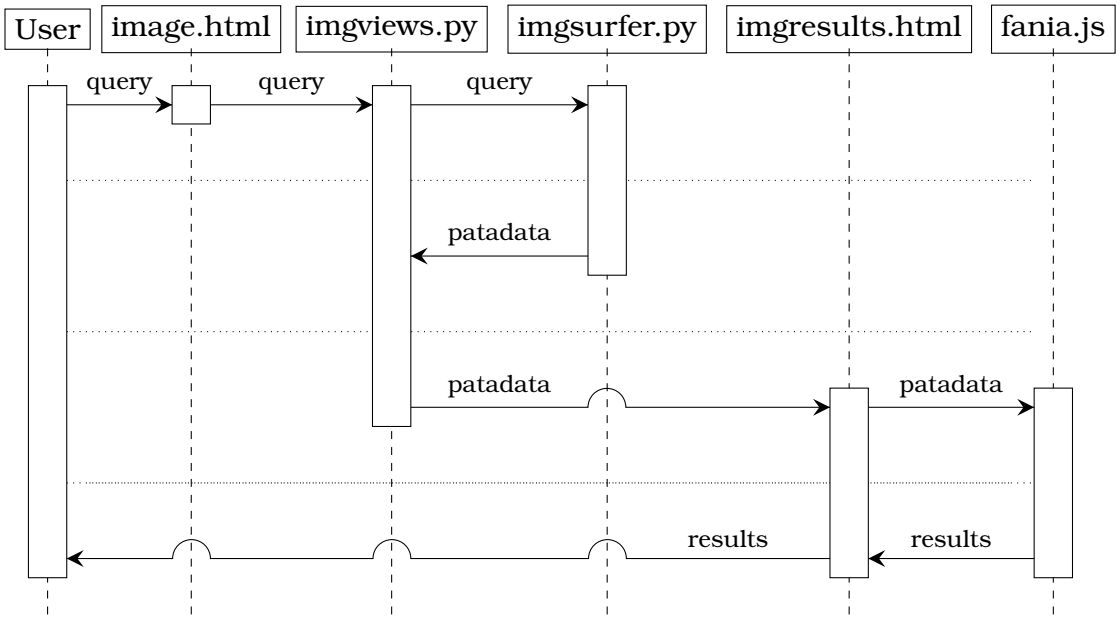


Figure 1.5 – Top-level overview of image / video search

Go to TOC

§ 1.2 each algorithm finds results for the query (see section 1.2), (4) some words before/after the match are retrieved for context, and (5) the resulting sentences are rendered for the user.

This chapter explains how `pata.physics.wtf` was created and how it operates technically. Specifically it will discuss the intitial setup of the system when it is first started up, the text search algorithms, the image and video Application Program Interface (API) calls and the main design elements (text poetry and image spirals).

1.1 Setup

1.1.1 Corpora

Instead of crawling the Internet `pata.physics.wtf` uses a local collection of texts for its text search. Setting up a custom web crawler would require a lot more resources (in terms of hardware, time and money) than practical for this project. There are two corpora containing 65 text files together.

The first corpus resembles the fictional library of 'equivalent books' from Alfred Jarry's *Exploits and Opinions of Dr. Faustroll, 'Pataphysician* (1996). In principle the corpus is just a folder within the tool's directory structure which contains the following files:

0. Alfred Jarry: *Exploits and Opinions of Dr. Faustroll, 'Pataphysician*
1. Edgar Allen Poe: *Collected Works*
2. Cyrano de Bergerac: *A Voyage to the Moon*
3. Saint Luke: *The Gospel*
4. Leon Bloy: *Le Desespere* (French)
5. Samuel Taylor Coleridge: *The Rime of the Ancient Mariner*
6. Georges Darien: *Le Voleur* (French)
7. Marceline Desbordes-Valmore: *Le Livre des Meres et des Enfants* (French)
8. Max Elskamp: *Enluminures* (French)
9. Jean-Pierre Claris de Florian: *Les Deux Billets* (French)
10. *One Thousand and One Nights*
11. Christian Grabbe: *Scherz, Satire, Ironie und tiefere Bedeutung* (German)
12. Gustave Kahn: *Le Conte de l'Or et Du Silence* (French)
13. Le Comte de Lautreamont: *Les Chants de Maldoror* (French)
14. Maurice Maeterlinck: *Aglavaine and Selysette*

Go to TOC

15. Stephane Mallarme: *Verse and Prose* (French)
16. Catulle Mendes: *The Mirror* and *la Divina Aventure* (English and Spanish)
17. Homer: *The Odyssey*
18. Josephin Peladan: *Babylon* (EMPTY FILE)[5]
19. Francois Rabelais: *Gargantua and Pantagruel*
20. Jean de Chilra: *L'Heure Sexuelle* (EMPTY FILE)[5]
21. Henri de Regnier: *La Canne de Jaspe* (EMPTY FILE)[5]
22. Arthur Rimbaud: *Poesies Completes* (French)
23. Marcel Schwob: *Der Kinderkreuzzug* (German)
24. Alfred Jarry: *Ubu Roi* (French)
25. Paul Verlaine: *Poems*
26. Emile Verhaeren: *Poems*
27. Jules Verne: *A Journey to the Centre of the Earth*

§ **??** The original list as it appears in 'Faustroll' is shown in chapter **??**. Three of the items have not been found as a resource. Some others have been approximated by using another text by the same author for example. Most of these were sourced from **Project Gutenberg**[6,7] in their original languages. The decision to get foreign language texts was partially due to the lack of out-of-copyright translated versions and partially because the original library in 'Faustroll' was also multi-lingual.

The second corpus is a collection of 38 texts by William Shakespeare (**Shakespeare2011**).

1. *The Sonnets*
2. *Alls Well That Ends Well*
3. *The Tragedy of Antony and Cleopatra*
4. *As You Like It*
5. *The Comedy of Errors*
6. *The Tragedy of Coriolanus*
7. *Cymbeline*
8. *The Tragedy of Hamlet, Prince of Denmark*

[5] I have not been able to find any source texts online.

[6] See https://www.gutenberg.org/

[7] **A note on copyright:** Duration of copyright: §5. 'For literary, dramatic, musical or artistic works 70 years from the end of the calendar year in which the last remaining author of the work dies.' (https://www.copyrightservice.co.uk/ukcs/docs/edupack.pdf) Maurice Maeterlinck and Marguerite Vallette-Eymery (a.k.a. Rachilde or Jean de Chilra) died less than 70 years ago and their work should still be under copyright. Alfred Jarry in the Simon Watson Taylor translation is a derivative work and is probably also still protected. (http://www.copyrightservice.co.uk/copyright/p22_derivative_works) ***Fair dealing***: §7. 'Private and research study purposes', so for the purposes of this project copyright should not apply.

Go to TOC

9. *The First Part of King Henry the Fourth*
10. *The Second Part of King Henry the Fourth*
11. *The Life of Kind Henry the Fifth*
12. *The First Part of Henry the Sixth*
13. *The Second Part of Henry the Sixth*
14. *The Third Part of Henry the Sixth*
15. *King Henry the Eigth*
16. *King John*
17. *The Tragedy of Julius Caesar*
18. *The Tragedy of King Lear*
19. *Love's Labour's Lost*
20. *The Tragedy of Macbeth*
21. *Measure for Measure*
22. *The Merchant of Venice*
23. *The Merry Wives of Windsor*
24. *A Midsummer Night's Dream*
25. *Much Ado About Nothing*
26. *The Tragedy of Othello, Moor of Venice*
27. *King Richard the Second*
28. *Kind Richard III*
29. *The Tragedy of Romeo and Juliet*
30. *The Taming of the Shrew*
31. *The Tempest*
32. *The Life of Timon of Athens*
33. *The Tragedy of Titus Andronicus*
34. *The History of Troilus and Cressida*
35. *Twelfth Night or What You Will*
36. *The Two Gentlemen of Verona*
37. *The Winter's Tale*
38. *A Lover's Complaint*

1.1.2 Startup

When the server is first started various setup functions (such as the creation of the index) are executed before any HTML is rendered. The search algorithms are triggered once a user enters a search term into the query field on any of the text, image or video pages.

Each plain text file in the corpus is added to the internal library one by one. Source 1.1 shows how this is done. The `PlaintextCorpusReader` is a feature of the Natural Language Tool Kit (NLTK) Python library (Project 2016) for Natural Language Processing.

Go to TOC

```
1 library = PlaintextCorpusReader(corpus_root, '.*\.txt')
2 l_00 = library.words('00.faustroll.txt')
3 l_01 = library.words('01.poe1.txt')
4 ...
5 l_27 = library.words('27.verne.txt')
```

Code 1.1 – Adding text files to the corpus library

The `setupcorpus` function (see source 1.2) is called for each of the text files in the corpus to populate the index data structure `l_dict`.

```
l_dict = dictionary { dictionary { list [ ] } }
```

A dictionary in Python is what is known as an 'associative array' in other languages. Essentially they are unordered sets of **key: value** pairs. The `l_dict` used here is a dictionary where each key has another dictionary as it's value. Each nested dictionary has a list as the value for each key.

```
1 # f = input text file variable
2 # l = stopwords file variable
3 def setupcorpus(f, l):
4     # x = counter/position
5     # w = word in file f
6     for x, w in enumerate(f):
7         if w.isalpha() and (w.lower() not in l):
8             y = 'l_' + (re.search(r"((\d\d).(\w)+.txt)",
              ↪  f.fileid)).group(2)
9             l_dict[w.lower()][y].append(x)
```

Code 1.2 – 'setupcorpus' function to process the corpus and create the index.

Line 6 in source 1.2 starts looping through file `f`. Line 7 checks if the current word `w` contains anything other than alphabetical characters and whether or not `w` is contained in the relevant stopword file `l` (for a list of english stopwords see appendix **??**). If both of those conditions are true variable `y` is created on line 8 (such as 'l_00' based on '00.faustroll.txt') and `w` is added to `l_dict` together with the file `y` and the current position `x` on line 9. After all files are processed, the index looks like this:

```
{
  word1: {fileA: [pos1, pos2, ...], fileB: [pos], ...},
  word2: {fileC: [pos1, pos2], fileK: [pos], ...},
```

Go to TOC

```
  ...
}
```

Using one of the terms from figure **??** on page **??**, here are their entries in the index file (the files are represented by their number in the corpus (see page **??**), i.e. **l_00** is the 'Faustroll' file, **l_01** is the 'Poe' file, etc.). An excerpt from the actual `l_dict` can be found in the appendix **??**.

```
{
  doctor: {
    l_00: [253, 583, 604, 606, 644, 1318, 1471, 1858, 2334, 2431,
          ↪  2446, 3039, 4743, 5034, 5107, 5437, 5824, 6195, 6228,
          ↪  6955, 7305, 7822, 7892, 10049, 10629, 11055, 11457,
          ↪  12059, 13978, 14570, 14850, 15063, 15099, 15259,
          ↪  15959, 16193, 16561, 16610, 17866, 19184, 19501,
          ↪  19631, 21806, 22570, 24867],
    l_01: [96659, 294479, 294556, 294648, 296748, 316773, 317841,
          ↪  317854, 317928, 317990, 318461, 332118, 338470,
          ↪  340548, 341252, 383921, 384136, 452830, 453015,
          ↪  454044, 454160, 454421, 454596, 454712, 454796,
          ↪  454846, 455030, 455278, 455760, 455874, 456023,
          ↪  456123, 456188, 456481, 456796, 457106, 457653,
          ↪  457714, 457823, 457894, 458571, 458918, 458998,
          ↪  459654, 459771, 490749],
    l_02: [11476, 12098, 28151, 36270],
    l_10: [53085, 53118, 53220, 53266, 53364, 53469, 53573, 53592,
          ↪  53621, 53718, 54873, 55262, 55525, 55577, 55614,
          ↪  55683, 55741, 56058, 62709, 113969, 114131, 114405,
          ↪  114794],
    l_19: [14928, 15702, 49560, 82710, 167218, 180210, 189817,
          ↪  189908, 190020, 190235, 190905, 199430, 226663,
          ↪  275454, 275928, 278097, 287375, 291383, 304731,
          ↪  306055, 324757, 330488],
    l_27: [16270, 79245]
  }, ...
}
```

1.2 Text

After the setup stage is completed and the webpage is fully loaded, user input in the form of a text query is required to trigger the three pataphysical algorithms.

Image and Video search do not use all three algorithms — where relevant this is highlighted in each section. Generally the following descriptions refer to the text search functionality.

Go to TOC

Explain difference in Text, Image and Video

1.2.1 Clinamen

The clinamen is the unpredictable swerve that Bök calls 'the smallest possible aberration that can make the greatest possible difference' (**Boek2002**).

> Like all digitally encoded information, it has unavoidably the uncomfortable property that the smallest possible perturbations —i.e. changes of a single bit— can have the most drastic consequences. (**Dijkstra1988**)

In simple terms, the clinamen algorithm works in two steps:

1. get clinamen words based on dameraulevenshtein and faustroll,
2. get sentences from corpus that match clinamen words.

find ref for dameraulevenshtein in baeza-yates book?

It uses the 'faustroll' text by Alfred Jarry (1996) as a base document and the Damerau-Levenshtein algorithm (**Damerau1964**; **Levenshtein1966**), which measures the distance between two strings (with 0 indicating equality), to find words that are similar but not quite the same. The distance is calculated using insertion, deletion, substitution of a single character, or transposition of two adjacent characters. This means that we are basically forcing the program to return matches that are of distance two or one, meaning they have two or one spelling errors in them.

```
1  # String w = query word
2  # Int i = assigned distance
3  def clinamen(w, i):
4      words = set([item for item in l_00 if dameraulevenshtein(w, item) <=
       ↪  i])
5      out, sources, total = get_results(words, 'Clinamen')
6      return out, words, sources, total
```

Code 1.3 – Clinamen function

Source 1.3 line 4 creates the set of clinamen words using a list comprehension. It retrieves matches from the 'faustroll' file `l_00` with the condition that they are of Damerau-Levenshtein distance `i` or less to the query term `w` (see appendix **??**). Duplicates are removed. Line 5 then makes a call to the generic `get_results`

Go to TOC

function to get all relevant result sentences, the list of source files and the total number of results.

```
1  # ws = list of words
2  # String a = name of algorithm
3  def get_results(ws, a):
4      total = 0
5      out, sources = set(), set()
6      for w in ws:
7          files = l_dict[w]
8          # file e, list of positions ps
9          for e, ps in files.items():
10             f = get_title(e)
11             sources.add(f)
12             sent = pp_sent(w.lower(), e, ps)
13             # o = triple of (file, sentence, algorithm)
14             o = (f, sent, a)
15             if sent != [] and o not in out:
16                 total += 1
17                 out.add(o)
18     return out, sources, total
```

Code 1.4 – 'get_results' function to get all sentences for a list of words.

The `get_results` function (see source 1.4) is used by all three algorithms (clinamen, syzygy and antinomy). Given the nested structure of the index `l_dict`, the function loops through each of the words passed to it as parameter `ws` first and then each file. Line 7 retrieves the dictionary of files from `l_dict`. Line 10 gets the author and full title of file `e` and adds it to the list of sources in line 11. Line 12 makes use of yet another function called `pp_sent` (see source 1.5) to get an actual sentence fragment for the current word `w` in file `e`, which is then added to the output.

In function `pp_sent` (source 1.5) line 6 is important to note because it is a key functionality point. Even though the index `l_dict` stores a full list of all possible positions of a given word in each file, the `pp_sent` function **only retrieves the sentence of the very first occurance of the word** rather than each one. This decision was taken to avoid overcrowding of results for the same keyword.

Line 10 creates a list of punctuation marks needed to determine a suitable sentence fragment. Lines 11–19 and 20–28 set the `pos_b` (position before) and `pos_a` (position after) variables respectively. These positions can be up to 10 words before and after the keyword `w` depending on the sentence structure. In line 30 the actual sentence fragment up to the keyword is retrieved, while in line

Go to TOC

```
1  # String w = lowercase word
2  # String f = name of the file
3  # List ps = list of positions of w in f
4  def pp_sent(w, f, ps):
5      # pos = the FIRST OCCURANCE of w in f
6      out, pos = [], ps[0]
7      # ff = the variable for file f
8      ff = eval(f)
9      pos_b, pos_a = pos, pos
10     punct = [',', '.', '!', '?', '(', ')', ':', ';', '\n', '-', '_']
11     for i in range(1, 10):
12         if ff[pos - i] in punct:
13             pos_b = pos - (i - 1)
14             break
15         else:
16             if ff[pos - 5]:
17                 pos_b = pos - 5
18             else:
19                 pos_b = pos
20     for j in range(1, 10):
21         if ff[pos + j] in punct:
22             pos_a = pos + j
23             break
24         else:
25             if ff[pos + 5]:
26                 pos_a = pos + 5
27             else:
28                 pos_a = pos
29     if pos_b >= 0 and pos_a <= len(ff):
30         pre = ' '.join(ff[pos_b:pos])
31         post = ' '.join(ff[pos+1:pos_a])
32         out = (pre, w, post)
33     return out
```

Code 1.5 – 'pp_sent' function to retrieve a sentence from a file.

31 the fragment just after the keyword is retrieved. `ff[pos_b:pos]` for example returns the list of words from position `pos_b` to position `pos` from file `ff`. The built-in Python `.join()` function then concatenates these words into one long string separated by spaces. On line 32 a triple containing the pre-sentence, keyword and post-sentence is set as the output and then returned.

The image/video searches don't use the clinamen function at all.

Go to TOC

1.2.2 Syzygy

The syzygy surprises and confuses. It originally comes from astronomy and denotes the alignment of three celestial bodies in a straight line. In a pataphysical context it is the pun. It usually describes a conjunction of things, something unexpected and surprising. Unlike serendipity, a simple chance encounter, the syzygy has a more scientific purpose.

In simple terms, the syzygy algorithm works in two steps:

1. get syzygy words based on synsets and hypo-, hyper- and holonyms from WordNet,
2. get sentences from corpus that match syzygy words.

```
1  # w = input query term
2  def syzygy(w):
3      words = set()
4      wordsets = wn.synsets(w)
5      for ws in wordsets:
6          words.update(get_nym('hypo', ws))
7          words.update(get_nym('hyper', ws))
8          words.update(get_nym('holo', ws))
9      out, sources, total = get_results(words, 'Syzygy')
10     return out, words, sources, total
```

Code 1.6 – Syzygy function.

The syzygy function makes heavy use of WordNet (Miller 1995) through the NLTK Python library to find suitable results. Specifically, as shown in source 1.6, the algorithm fetches the set of synonyms (synsets) on line 4. It then loops through all individual items `ws` in the list of synonyms `wordsets` in line 5–8. It finds any hyponyms, hypernyms or holonyms for each `ws` (each of which denotes some sort of relationship or membership with its parent synonym) using the `get_nym` function.

explain reasoning behind algorithms like this for all:

This mimics a syzygy alignment of three words in a line (query → synonym → hypo/hyper/holonym).

Line 9 makes use of the `get_results` function (see source 1.4) in the same was as the clinamen function does.

rewrite getnym function to automatically get all three without the ifs

Go to TOC

The image and video searches both use the syzygy function as part of their `pataphysicalise` function (see source 1.8).

1.2.3 Antinomy

The antimony, in a pataphysical sense, is the mutually incompatible.

In simple terms, the antinomy algorithm works in two steps:

1. get antinomy words based on synsets and antonyms from WordNet,
2. get sentences from corpus that match antinomy words.

```
# w = input query term
def antinomy(w):
    words = set()
    wordsets = wn.synsets(w)
    for ws in wordsets:
        anti = ws.lemmas()[0].antonyms()
        if len(anti) > 0:
            for a in anti:
                if str(a.name()) != w:
                    words.add(str(a.name()))
    out, sources, total = get_results(words, 'Antinomy')
    return out, words, sources, total
```

Code 1.7 – Antinomy function.

For the antinomy we simply used WordNet's antonyms (opposites) (see source 1.7). This algorithm is very similar to the algorithm for the syzygy. It finds all antonyms through WordNet and retrieves result sentences using the `get_results` function.

1.2.4 Formalisation

A formal description of the `pata.physics.wtf` system in terms of an Information
§ ?? Retrieval (IR) model described in chapter **??** is unsuitable. It assumes for example the presence of some sort of ranking algorithm $R(q_i, d_j)$. It also assumes a set set of queries—our doesn't have that, queries come in as the user sends them.

Remember, (**Yates2011**) defines an IR model is a quadruple $[D, Q, F, R(q_i, d_j)]$ where:

D is the set of documents,
Q is the set of queries,

Go to TOC

F	is the framework,
$R(q_i, d_j)$	is the ranking function, where $q_i \in Q$ and $d_j \in D$,
t	is the number of index terms in a document collection,
V	is the set of all distinct index terms $\{k_1, \ldots, k_t\}$.

Making relevant changes, for the Faustroll corpus text search, an approximate system description could be this:

D	is the set of documents $\{d_1, \ldots, d_m\}$,
m	is the number of all documents in D ($\|D\| = 28$),
V	is the set of all distinct terms $\{v_1, \ldots, v_n\}$ in D, not including stopwords,
n	is the number of all distinct terms in V ($\|V\| = 78893$),
v	is a vocabulary entry of form $\{d(v) \mapsto [l(v)]\}$, where $l(v)$ is the location of term v in text d
q	is the user query,
F	is the set of patalgorithms Clinamen, Syzygy and Antinomy $\{f_C, f_S, f_A\}$,
P	is the set of pataphysicalised query terms $\{p_1, \ldots, p_u\}$,
u	is the number of terms in P,
$P(q)$	is the set of patadata $\{P(q)_C \cup P(q)_S \cup P(q)_A\}$ for query q,
R	is the set of results $\{r_1, \ldots, r_o\}$,
o	is the number of results in R,
$R(P(q))$	is the set of results $\{R(P(q)_C) \cup R(P(q)_S) \cup R(P(q)_A)\}$, produced by each algorithm in F,
r	is a result of form (d, sentence, f).

$$P(q)_C = \{p \in v_1 : 0 < \text{dameraulevenshtein}(q, p) \leq 2\} \tag{1.1}$$

dameraulevenshtein(q, p) in equation 1.1 is the Damerau-Levenshtein algorithm as described in section 1.2.1 and v_1 is the Faustroll text.

§ 1.2.1

$$\begin{gathered} P(q)_S = \{p \in V : p \in \text{nyms}(s),\ \forall s \in \text{synonyms}(q)\} \\ \text{where nyms}(s) = \text{hypos}(s)\ \cup\ \text{hypers}(s)\ \cup\ \text{holos}(s)\ \cup\ \text{meros}(s) \end{gathered} \tag{1.2}$$

synonyms(q) in equation 1.2 is the WordNet function to retrieve all synsets for the query q and the four 'nym' functions return the relevant hyponyms, hypernyms, holonyms or meronyms for each of the synonyms.

$$P(q)_A = \{p \in V : p \in \text{antonyms}(s),\ \forall s \in \text{synonyms}(q)\} \tag{1.3}$$

Go to TOC

Similarly, in equation 1.3 the synonyms(q) function returns WordNet synsets for the query and the antonyms(s) function returns WordNet antonyms for each of the synonyms.

$$R(P(q)) = \{(d \in D,\ sent(p) \in d,\ f \in F) : \forall\ p \in P(q)_f))\} \tag{1.4}$$

$R(P(q))$ in equation 1.4 returns a list of triples containing the source text (d), the sentence $sent(p)$ and the algorithm f. For each pataphysicalised query term p one sentence is retrieved per file d.

1.3 Image & Video

In simple terms, the image and video search works in three steps:

1. pataphysicalise query terms using syzygy algorithm
2. translate each pataphysicalised term
3. retrieve images/videos using **api!** calls

The `pataphysicalise` function (see source 1.8) transforms the original query terms ready for the next step. In line 5 the `syzygy` algorithm (source 1.6) is used to make this transformation. Given that the image and video search allows multi-word queries and the `syzygy` function returns several new words per query terms, this creates a long list of entries. On top of that the output is the inner product (line 8) of all these results. The purpose of producing so many pataphysicalisations is to find more results using the **api!** (**api!**).

```
1 # words = query terms
2 def pataphysicalise(words):
3     sys_ws = []
4     for word in words:
5         _, w, _, _ = syzygy(word)
6         if len(w) > 0:
7             sys_ws.append(list(w))
8     out = itertools.product(*sys_ws)
9     return list(out)
```

Code 1.8 – Function to pataphysicalise image and video query terms.

For example, running the pataphysicalise function with the terms ‘clear’ and ‘sky’ will produce two intermediary lists (shortened here for the demonstration) which are then combined into one list using the Cartesian product:

```
["disembarrass", "bear", "judge", "remove", "elucidate", "modify",
      ↪  "free", "approve", "certify", "determine", "strip",
      ↪  "empty", "purge", "vanish", "disappear", "sell", "pay",
      ↪  "make", "take", "disforest", "formalize", "okay", "allow",
      ↪  ...],
["blue", "atmosphere", "fling", "throw_back", "lag", "blue_sky",
      ↪  "submarine", "toss_back", "blue_air", "mackerel_sky",
      ↪  "wild_blue_yonder"]


[("disembarrass", "blue"), ("disembarrass", "atmosphere"), ...,
      ↪  ("strip", "fling"), ..., ("empty", "submarine"), ...,
      ↪  ("allow", "mackerel_sky"), ("allow", "wild_blue_yonder")]
```

The next step is to translate the pataphysicalised search terms as shown in source 1.9 before any API calls are made.

```
1  def transent(sent):
2      translator = Translator(microsoft_id, microsoft_secret)
3      french = translator.translate(sent, "fr")
4      japanese = translator.translate(french, "ja")
5      patawords = translator.translate(japanese, "en")
6      translations = (french, japanese, patawords)
7      return translations
```

Code 1.9 – Translation function.

redo whole section to reflect chanegs

1.3.1 REST & API

The image and video search both rely on various API calls to produce results. Currently used are Microsoft Translate, Bing Image Search and YouTube.

A RESTful API allows browsers ('clients') to communicate with a web server via HTTP methods such as GET and POST. The idea is that a given service, like the Microsoft Bing search API, can be accessed in a few simple steps using a library like **Requests**[8]. These are:

1. Construct the Uniform Resource Locator (URL) (see, source 1.10 lines 5,6,7 and 11)
2. get an API key (see, source 1.10 line 4)
3. send URL and key using GET method (see, source 1.10 line 12)

[8] http://docs.python-requests.org/en/latest/

Go to TOC

4. receive and process response in requested format (e.g. JavaScript Object Notation (JSON)[9])

```
def get_Bing(words):
    out = []
    trans = ''
    bing_key = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
    base = "https://api.datamarket.azure.com/Bing/Search/"
    params = "Image?$format=json&Query='"
    after = "'"
    for x in words:
        y = ' '.join(x)
        z = transent(y)  # (french, japanese, patawords)
        url = ''.join([base, params, z[2], after])
        bing_img = requests.get(url, auth=HTTPBasicAuth(None, bing_key))
        if bing_img.json()['d']['results']:
            trans = z
            for result in bing_img.json()['d']['results']:
                phototitle = result['Title']
                photoimg = result['MediaUrl']
                photolink = result['SourceUrl']
                out.append((phototitle, photoimg, photolink))
            break
        else:
            out = []
    return out, trans
```

Code 1.10 – Using the Microsoft Bing API to retrieve images.

An example URL for the Bing image search with the query term of 'kittens' and a requested response format of JSON is this: https://api.datamarket.azure.com/Bing/Search/Image?$format=json&Query='kittens'. There are many other parameters that can be specified, such as 'Adult' (which can be set to 'Moderate' for example) and 'ImageFilters' (which allows users to specify size or aspect ratio)[10].

Bing will then send back the response in JSON format. One entry of the list of results looks like this (with whitespace formatting added for convenience). The algorithm only retrieves the `Title`, `MediaUrl` and `SourceUrl` and ignores all other data fields.

```
"d": { "results": [
  { "__metadata":
```

[9] http://www.json.org/

[10] see https://datamarket.azure.com/dataset/bing/search#schema

```
    { "uri":
          ↪  "https://api.datamarket.azure.com/Data.ashx/Bing/Search/Image?Query=\u0
      "type": "ImageResult"
    }, // __metadata
    "ID": "e09072a2-faf3-47ac-b77d-46a8df8941aa",
    "Title": "Cute Kittens - Pictures - The Wondrous Pics",
    "MediaUrl":
          ↪  "http://wondrouspics.com/wp-content/uploads/2011/12/Cute-Kitten2.jpg",
    "SourceUrl": "http://wondrouspics.com/cute-kittens-pictures/",
    "DisplayUrl": "wondrouspics.com/cute-kittens-pictures",
    "Width": "1440",
    "Height": "900",
    "FileSize": "238015",
    "ContentType": "image/jpeg",
    "Thumbnail":
    { "__metadata":
      { "type": "Bing.Thumbnail"
      },
      "MediaUrl":
            ↪  "http://ts2.mm.bing.net/th?id=OIP.M5692e5d79242507e30600fd54639316cH0
      "ContentType": "image/jpg",
      "Width": "480",
      "Height": "300",
      "FileSize": "13856"
    } // Tumbnail
  }, ...
  ], // results
  "__next":
        ↪  "https://api.datamarket.azure.com/Data.ashx/Bing/Search/Image?Query=\u002
} // d
```


1.4 Design

Once the three algorithms have produced their respective results, the page displaying these results can be rendered. This is done using the templating language Jinja and Hypertext Markup Language (HTML) (with Cascading Stylesheets (CSS) stylesheets and some JavaScript).

'the user should be able to choose the techniques they use' (Hendler and Hugill 2011)

The text results page has three options for how the results are presented, with 'Poetry — Queneau' being the default.

Poetry

Displayed in sonnet style (two quatrains and two tercets) if possible, al-

Figure 1.6 – proto3screen

though no rhyming pattern is used.[11]

- Queneau — Each line can be changed manually.
- Random — The whole poem can be randomised.

Sources
: Ordered by source text.

Algorithms
: Ordered by algorithm.

get proper ref for sonnet style

The image and video results pages work the same way. They both have two display options, with the 'Spiral' option being the default. The spirals are modelled on the idea of Fibonacci spirals.

Spiral
: Displayed square images/videos as a golden spiral.

List Displayed as a simple list.

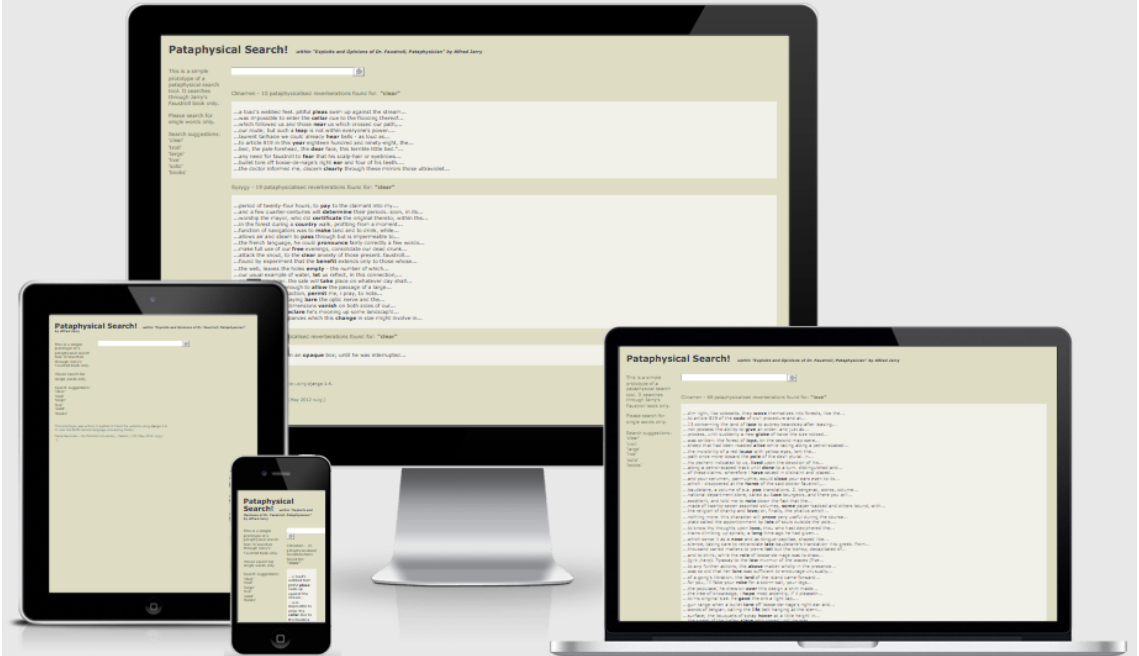
1.4.1 Poetry

[11] https://en.wikipedia.org/wiki/Sonnet

Go to TOC


PATA. ꟻHYSICS. SEARCH
the virgin, the bright, and the beautiful today
DEFINITIONS
(MIS-)UNDERSTANDING PATAPHYSICS
SYZYGY
ANTINOMY
ALFRED JARRY
PATA. PHYSICS. SEARCH
SAINT CADO.
PATA. SEARCH


PATAꟻHYSICAL SƎARCH
(MIS-)UNDERSTANDING PATAPHYSICS
DEFINITIONS
PATAPHYSICAL SEARCH
¶ WELCOME!


Pataphysical Search!
Pataphysical Search!
Pataphysical Search!

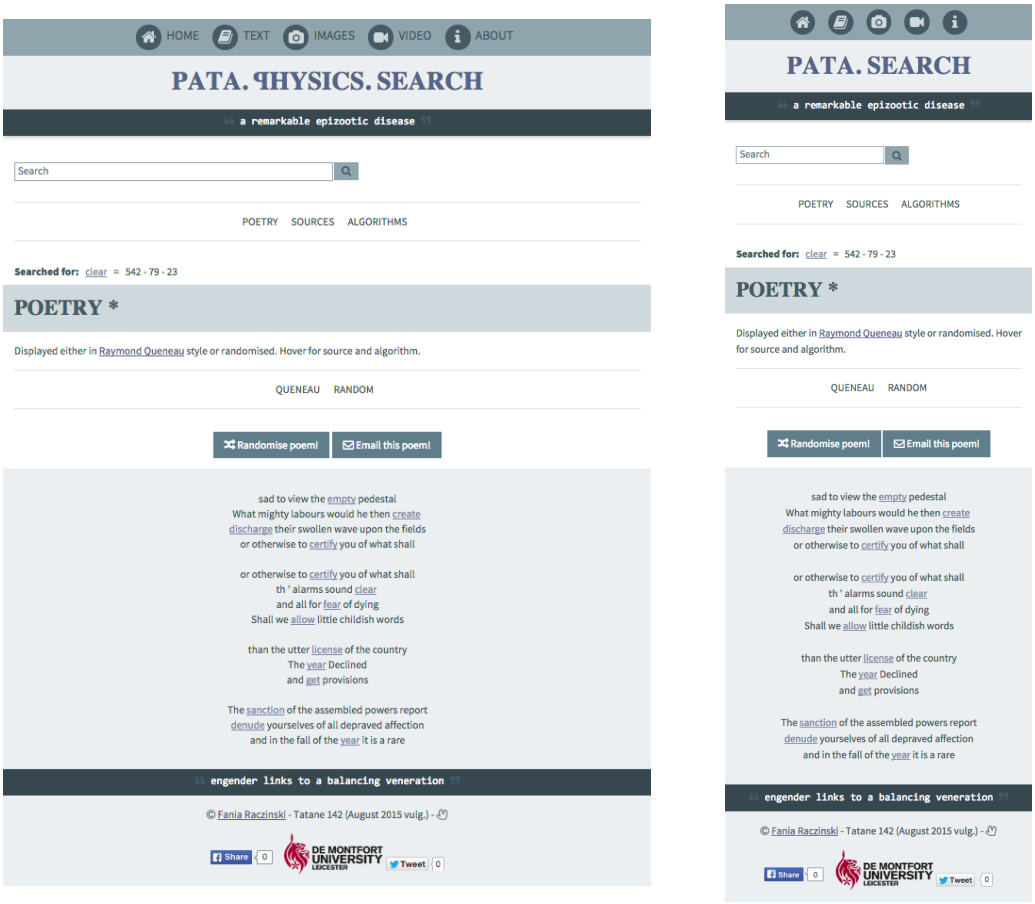


Figure 1.8 – Poetry results screenshot & mobile

Source 1.11 shows the segment of HTML/Jinja code that renders the Queneau Poetry. Lines 2-6 creates a button for sending the currently showing poem per email. Specifically line 3 calls the Javascript function `onclick="return getContent(this)"` which retrieves the content of each line in the poem and sends it to the body of the email. Lines 7-22 render the 4 stanzas of the poem. This is done using two nested Jinja 'for' loops (line 8 and line 16). Line 8 loops through the (ideally) 14 lines of the poem. `lol` can be considered a masterlist of all sublists for each poem line.

get structure of lol as opposed to all_sens

```
# all_sens list:
  [(title, (pre, word, post), algorithm), ...]
# lol list:
  [all_sens[0], all_sens[1], ...]
```

Go to TOC

```
    <div class=``subtab_content'' id=``q_tab''>
      <p class=``w3-center''>
        <a class=``emailbutton w3-btn w3-blue-grey'' href=``#''
         ↪  onclick=``return getContent(this)''>
          Email this poem!
        </a>
      </p>
      <div class="poetry w3-container w3-theme-l5">
        {% for n in range(1, lol|length + 1) %}
          {% set wid = ['wn', n|string]|join %}
          {% set lid = ['lyr', n|string]|join %}
          {% set sid = ['scrollLinks', n|string]|join %}
          {% set aid = lol[n-1] %}
          <div id="poems">
            <div id="{{wid}}" class="wn">
              <div id="{{lid}}" class="lyr">
                {% for sens in aid %}<span title="{{ sens[0] }}, {{
                 ↪  sens[2] }}">{{ sens[1][0] }} <form class="inform"
                 ↪  action="../textresults" method="post"><input
                 ↪  class="inlink" type="submit" name="query" value="{{
                 ↪  sens[1][1] }}" onclick="loading();"></input></form>
                 ↪  {{ sens[1][2] }}</span>{% endfor %}
              </div>
            </div>
            <div id="{{sid}}" class="scrollLinks"></div>
          </div>
        {% endfor %}
      </div>
    </div>
```

Code 1.11 – Code for rendering Queneau style poems.

1.4.2 Spiral

http://www.texample.net/tikz/examples/fibonacci-spiral/

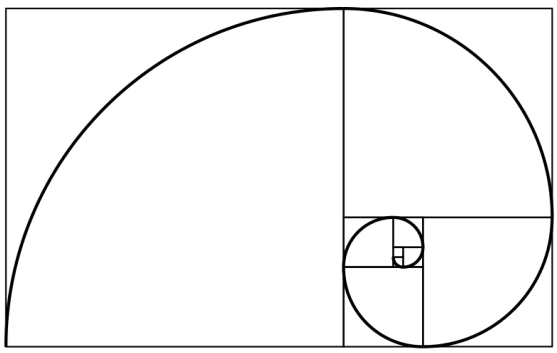


Figure 1.9 – Fibonacci Spiral[12]

Go to TOC

```
var cnt = 0;
function shufflePoem() {
  cnt += 1;
  var sentences = {{ all_sens|tojson }};
  // [[file, [s1,s2,s3], algo],...]
  var n = {{ all_sens|length }};
  var rlist = [];
  for (var i = 0; i < 14; i++) {
    var r = Math.floor(Math.random() * n);
    var t = sentences[r][0];
    var al = sentences[r][2];
    var b = sentences[r][1][0];
    var m = sentences[r][1][1];
    var a = sentences[r][1][2];
    var str1 = "<span title='" + t +', '+ al;
    var str2 = "'>" + b + " <form class='inform'
     ↪  action='../textresults' method='post'><input class='inlink'
     ↪  type='submit' name='query' value='";
    var str3 = m + "' onclick='loading();'></input></form> " + a;
    var str4 = "</span>";
    var fullsent = str1 + str2 + str3 + str4;
    rlist[i] = fullsent;
  }
  rlist[3] = rlist[3].concat('<br>');
  rlist[7] = rlist[7].concat('<br>');
  rlist[10] = rlist[10].concat('<br>');
  var output = rlist.join('<br>');
  document.getElementById('clickcount').innerHTML = cnt;
  document.getElementById('random_poem').innerHTML = output;
  return false;
}
```

Code 1.12 – Code for randomising poems.

1.5 Prototypes

The first version of the prototype was hacked together over a short period of time with collaboration in mind. It was originally build to demonstrate the three algorithms in action before James' architecture was finished. The design of the website was simple and plain.

Results were displayed in three sets per algorithm. Each keyword was preceded and followed by exactly 5 words.

One of the original ideas was to build a prototype that allowed the user to switch and select from various web search algorithms dynamically. The system architecture was never built. My prototype was built with the intention to show the

Go to TOC

Table 1.1 – Comparison of prototypes

	Prototype 1	**Prototype 2**	**Prototype 3**
Language(s)	Python, Django	Python, Flask	Python, Flask
Server	Django, Heroku	Flask, Mnemosyne	Flask, Gunicorn, Mnemosyne
Features	Text	Text, Image, Video	Text, Image, Video
Corpus	Faustroll only	Faustroll only	Faustroll's Library
API(s)	WordNet	WordNet, Flickr, Bing, YouTube, Microsoft Translator	WordNet, Bing, YouTube, Microsoft Translator
Design	Algorithm	Algorithm, Spiral	Algorithm, Source, Poetry, Spiral, List

Table 1.2 – My caption

Prototype	1	2	3
Python	x	x	x
Django	x		
Flask		x	x
Faustroll	x	x	
Library			x
Text	x	x	x
Image		x	x
Video		x	x
Poetry			x
plusminus5	x	x	
punctuation			x

Go to TOC

algorithms in action before the full implementation of the surrounding architecture was finished. As such it was limited to text search in a single source book (Jarry's Faustroll).

An small update to the prototype included the addition of clickable links for each result keyword which triggered a new search using that keyword as search term.

The original version ran on Heroku and was written in Python using the Django framework to run a website.

get new screenshots for prototype 1

don't mention James?

Figure 1.10 – Prototype 1 screenshot

The main differences between prototype 1 and prototype 2 are:

Go to TOC

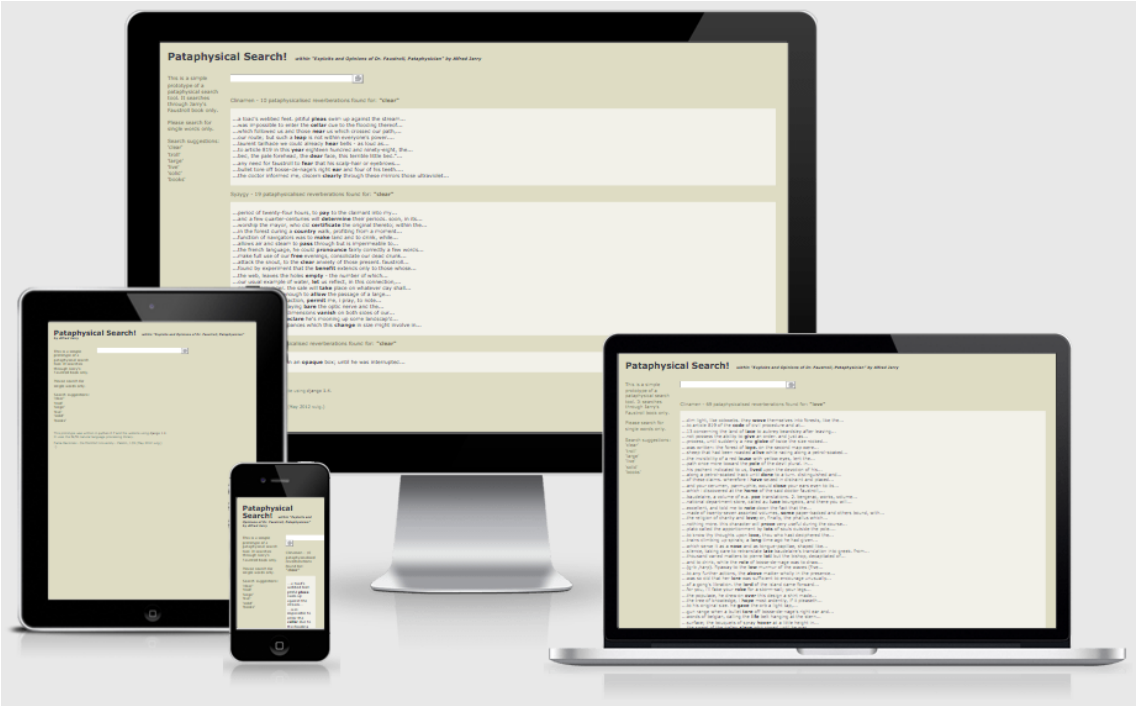


Figure 1.11 – proto1screen

- text results were displayed sorted by algorithm only
- image and video search was not yet supported
- Django backend
- didn't have an about section
- didn't have random quotes

—

This version introduced the move from Django to Flask. It also included the first major re-design of the website. Flask made things simpler than Django.

It is still available online at `pata.fania.eu`.

A responsive design was created. Image and video search functionality was added.

Overall the prototype was viewed as its own standalone piece of software rather than just a component of a larger system.

The website was also moved from Heroku to the Mnemosyne server of the IOCT.

The main differences between the current version and prototype 2 are:

- the corpus consisted of the faustroll text only

Go to TOC

Figure 1.12 – Prototype 2 screenshot

Figure 1.13 – proto2screen

Go to TOC

- results were keyword ± 5 words per line
- text results were displayed sorted by algorithm only
- image and video results were displayed as spiral only

—

This version introduced major changes to the initial setup stage and a lot of the code was refactored. Another design update was also implemented. To the user the most obvious change will be the presentation of results. There are now various display choices. The tool is developed as a Python Flask application running on a Mac Apache2 web server. The flask development server is started using the ‘python dev.py‘ command. This mode is set up for debugging and will give detailed error messages. Starting the live gunicorn server on apache2 use ‘guni ..... guni.py‘. This uses several threads etc. The stylesheet is based on the **w3.css**.

Go to TOC

INTERLUDE II

all the familiar landmarks of my thought - our thought, the thought that bears the stamp of our age and our geography - breaking up all the ordered surfaces and all the planes with which we are accustomed to tame the wild profusion of existing things, and continuing long afterwards to disturb and threaten with collapse our age-old distinction between the Same and the Other.

(Foucault 1966)—taking about Borges

Only those who attempt the absurd achieve the impossible.

(attributed to M.C. Escher)

A great truth is a truth whose opposite is also a great truth. Thomas Mann

(as cited in Wickson, Carew and Russell 2006)

Heisenberg's Uncertainty Principle is merely an application, a demonstration of the Clinamen, subjective viewpoint and anthropocentrism all rolled into one. (Jarry 2006)

Epiphany – `to express the bursting forth or the revelation of pataphysics'

Dr Sandomir (Hugill 2012, p.174)

Machines take me by surprise with great frequency. (Turing 2009, p.54)

The view that machines cannot give rise to surprises is due, I believe, to a fallacy to which philosophers and mathematicians are particularly subject. This is the assumption that as soon as a fact is presented to a mind all consequences of that fact spring into the mind simultaneously with it. (Turing 2009, p.54)

Opposites are complementary.
It is the hallmark of any deep truth that its negation is also a deep truth.
Some subjects are so serious that one can only joke about them. Niels Bohr

There is no pure science of creativity, because it is paradigmatically idiographic — it can only be understood against the backdrop of a particular history.
(Elton 1995)

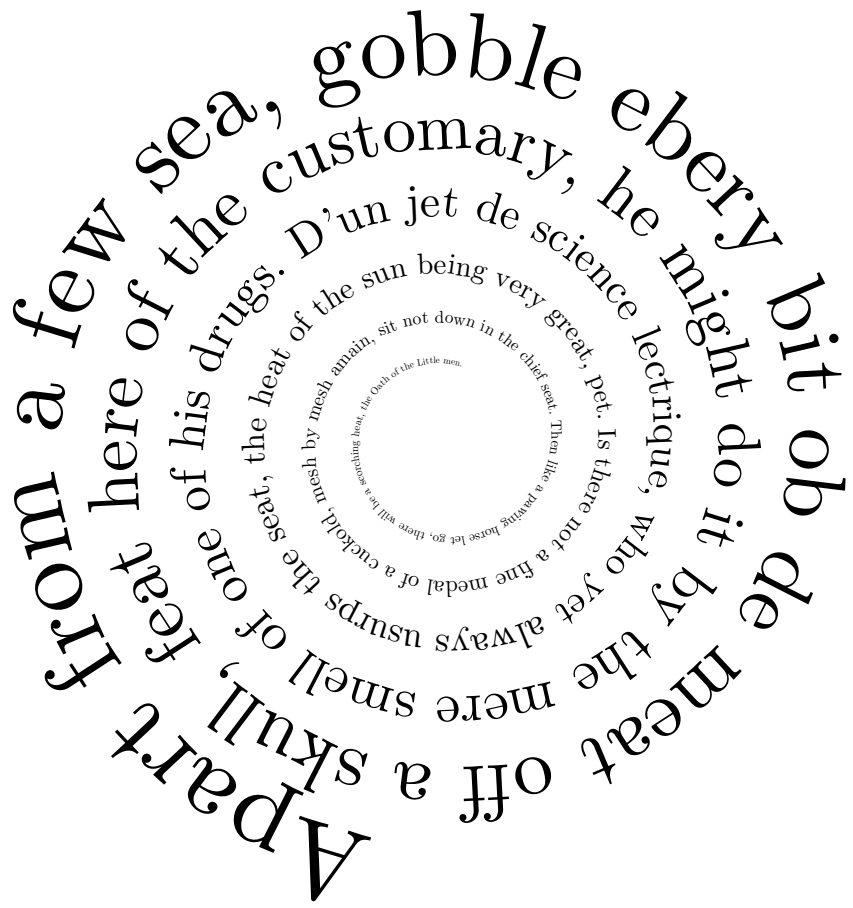
Tools are not just tools. They are cognitive interfaces that presuppose forms of mental and physical discipline and organization. By scripting an action, they produce and transmit knowledge, and, in turn, model a world. (Burdick et al. 2012, p.105)

Humanists have begun to use programming languages. But they have yet to create programming languages of their own: languages that can come to grips with, for example, such fundamental attributes of cultural communication and traditional objects of humanistic scrutiny as nuance, inflection, undertone, irony, and ambivalence.
(Burdick et al. 2012, p.103)

Go to TOC

Part V

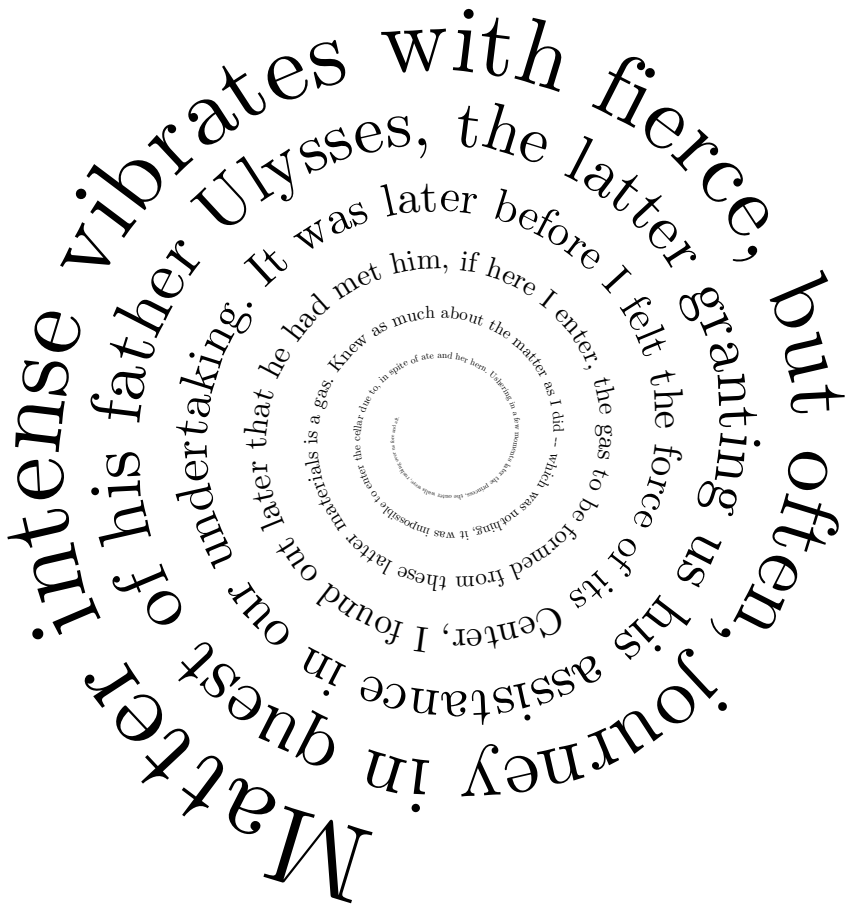
MΣTA-
LΘGICALYSIS



Go to TOC

Part VI

HAPPILY EVER AFTER

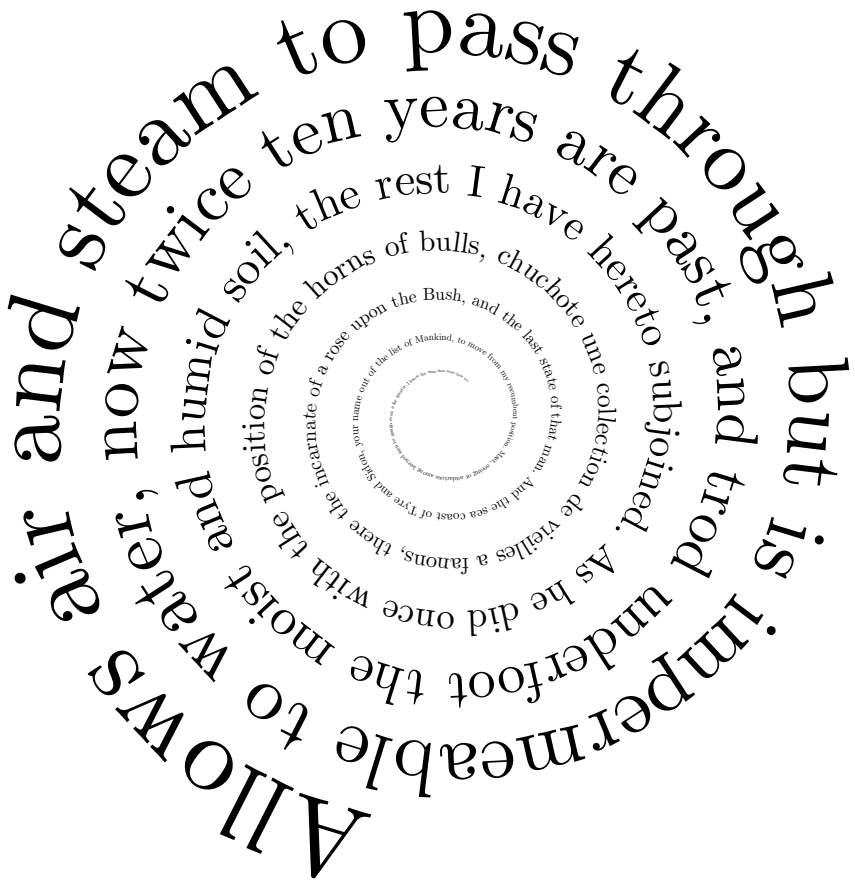


Go to TOC

INTERLUDE III

Part VII

POST☹



Go to TOC

References

Agichtein, Eugene, Eric Brill and Susan Dumais (2006). 'Improving web search ranking by incorporating user behavior information'. In: ***ACM SIGIR conference on Research and development in information retrieval***. New York, New York, USA: ACM Press, p. 19.

Baeza-Yates, Ricardo and Berthier Ribeiro-Neto (2011). ***Modern Information Retrieval: The Concepts and Technology Behind Search***. Addison Wesley.

Baidu (2012). ***Baidu About***.

Baldi, Pierre and Laurent Itti (2010). 'Of bits and wows : A Bayesian theory of surprise with applications to attention'. In: ***Neural Networks*** 23, pp. 649–666.

Bao, Shenghua et al. (2007). 'Optimizing Web Search Using Social Annotations'. In: ***Distribution***, pp. 501–510.

Bastos Filho, Carmelo et al. (2008). 'A novel search algorithm based on fish school behavior'. In: ***IEEE International Conference on Systems, Man and Cybernetics***, pp. 2646–2651.

Bharat, Krishna and George Mihaila (2000). 'Hilltop: A Search Engine based on Expert Documents'. In: ***Proc of the 9th International WWW***. Vol. 11.

Bing, Microsoft (2016). ***Meet our crawlers***.

Bird, Steven, Ewan Klein and Edward Loper (2009). ***Natural Language Processing with Python***. Sebasopol, CA: O'Reilly Media.

Boden, Margaret (2003). ***The Creative Mind: Myths and Mechanisms***. London: Routledge.

Brin, Sergey and Larry Page (1998a). 'The anatomy of a large-scale hypertextual Web search engine'. In: ***Computer Networks and ISDN Systems*** 30.1-7, pp. 107–117.

– (1998b). 'The PageRank Citation Ranking: Bringing Order to the Web'. In: ***World Wide Web Internet And Web Information Systems***, pp. 1–17.

Burdick, Anne et al. (2012). ***Digital Humanities***. Cambridge, Massachusetts: MIT Press (cit. on p. 37).

Burnham, Douglas (2015). 'Immanuel Kant: Aesthetics'. In: ***Internet Encyclopedia of Philosophy*** (cit. on p. 3).

Candy, Linda (2012). 'Evaluating Creativity'. In: ***Creativity and Rationale: Enhancing Human Experience by Design***. Ed. by J.M. Carroll. Springer.

Colton, Simon (2008a). 'Computational Creativity'. In: ***AISB Quarterly***, pp. 6–7.

– (2008b). 'Creativity versus the perception of creativity in computational systems'. In: ***In Proceedings of the AAAI Spring Symp. on Creative Intelligent Systems***.

Colton, Simon, Alison Pease and Graeme Ritchie (2001). ***The Effect of Input Knowledge on Creativity***.

De Bra, Paul, Geert-jan Houben et al. (1994). 'Information Retrieval in Distributed Hypertexts'. In: ***Techniques***.

De Bra, Paul and Reinier Post (1994a). 'Information retrieval in the World-Wide Web: Making client-based searching feasible'. In: ***Computer Networks and ISDN Systems*** 27.2, pp. 183–192.

– (1994b). 'Searching for Arbitrary Information in the WWW: the Fish Search for Mosaic'. In: ***Mosaic A journal For The Interdisciplinary Study Of Literature***.

Dean, Jeffrey, Luiz Andre Barroso and Urs Hoelzle (2003). 'Web Search for a Planet: The Google Cluster Architecture'. In: ***Ieee Micro***, pp. 22–28.

Deerwester, Scott et al. (1990). 'Indexing by Latent Semantic Analysis'. In: ***Journal of the American Society for Information Science*** 41.6, pp. 391–407.

Ding, Li et al. (2004). 'Swoogle: A semantic web search and metadata engine'. In: ***In Proceedings of the 13th ACM Conference on Information and Knowledge Management. ACM***.

Du, Zhi-Qiang et al. (2007). 'The Research of the Semantic Search Engine Based on the Ontology'. In: ***2007 International Conference on Wireless Communications, Networking and Mobile Computing***, pp. 5398–5401.

Elton, Matthew (1995). 'Artificial Creativity: Enculturing Computers'. In: ***Leonardo*** 28.3, pp. 207–213 (cit. on p. 37).

Foucault, Michel (1966). 'The Order of Things - Preface'. In: ***The Order of Things***. France: Editions Gallimard. Chap. Preface, pp. xv–xxiv (cit. on p. 36).

Garcia-Molina, Hector, Jan Pedersen and Zoltan Gyongyi (2004). 'Combating Web Spam with TrustRank'. In: ***In VLDB***. Morgan Kaufmann, pp. 576–587.

Glover, E.J. et al. (2001). 'Improving category specific Web search by learning query modifications'. In: ***Proceedings 2001 Symposium on Applications and the Internet***, pp. 23–32.

Google (2012). ***Google Ranking***.

– (2016). ***Googlebot***.

Haveliwala, Taher H (2003). 'Topic-Sensitive PageRank: A Context Sensitive Ranking Algorithm for Web Search'. In: ***Knowledge Creation Diffusion Utilization*** 15.4, pp. 784–796.

Hendler, Jim and Andrew Hugill (2011). 'The Syzygy Surfer : Creative Technology for the World Wide Web'. In: ***ACM WebSci 11*** (cit. on pp. 3, 25).

Hersovici, M et al. (1998). 'The shark-search algorithm. An application: tailored Web site mapping'. In: ***Computer Networks and ISDN Systems*** 30.1-7, pp. 317–326.

Hotho, Andreas et al. (2006). 'Information retrieval in folksonomies: Search and ranking'. In: ***The Semantic Web: Research and Applications, volume 4011 of LNAI***. Springer, pp. 411–426.

Hugill, Andrew (2012). ***'Pataphysics: A Useless Guide***. Cambridge, Massachusetts: MIT Press (cit. on p. 36).

Jarry, Alfred (1996). ***Exploits and Opinions of Dr Faustroll, Pataphysician***. Cambridge, MA: Exact Change (cit. on pp. 11, 16).

– (2006). ***Collected Works II - Three Early Novels***. Ed. by Alastair Brotchie and Paul Edwards. London: Atlas Press (cit. on pp. 3, 36).

Jeh, Glen and Jennifer Widom (2002). 'SimRank: A Measure of Structural Context Similarity'. In: ***In KDD***, pp. 538–543.

Jordanous, Anna Katerina (2011). 'Evaluating Evaluation : Assessing Progress in Computational Creativity Research'. In: ***Proceedings of the Second International Conference on Computational Creativity***.

– (2012). 'Evaluating Computational Creativity: A Standardised Procedure for Evaluating Creative Systems and its Application'. PhD thesis. University of Sussex.

Jordanous, Anna Katerina and Bill Keller (2012). 'Weaving creativity into the Semantic Web: a language-processing approach'. In: ***Proceedings of the 3rd International Conference on Computational Creativity***, pp. 216–220.

Jurafsky, Daniel and James H Martin (2009). ***Speech and Language Processing***. London: Pearson Education.

Kamps, Jaap, Rianne Kaptein and Marijn Koolen (2010). ***Using Anchor Text , Spam Filtering and Wikipedia for Web Search and Entity Ranking***. Tech. rep. ?

Kleinberg, Jon M (1999). 'Authoritative sources in a hyperlinked environment'. In: ***journal of the ACM*** 46.5, pp. 604–632.

Kleinberg, Jon M et al. (1999). 'The Web as a graph : measurements, models and methods'. In: ***Computer***.

Luke, Saint (2005). ***The Gospel According to St. Luke***. Ebible.org.

Luo, Fang-fang, Guo-long Chen and Wen-zhong Guo (2005). 'An Improved 'Fish-search' Algorithm for Information Retrieval'. In: ***2005 International Con-***

ference on Natural Language Processing and Knowledge Engineering, pp. 523–528.

Macdonald, Craig (2009). 'The Voting Model for People Search'. In: ***Philosophy***.

Manning, Christopher, Prabhakar Raghavan and Hinrich Schuetze (2009). ***Introduction to Information Retrieval***. Cambridge UP.

Marchionini, Gary (2006). 'From finding to understanding'. In: ***Communications of the ACM*** 49.4, pp. 41–46.

Marchionini, Gary and Ben Shneiderman (1988). 'Finding facts vs. browsing knowledge in hypertext systems'. In: ***Computer*** 21.1, pp. 70–80.

Marcus, Mitchell P, Beatrice Santorini and Mary Ann Marcinkiewicz (1993). 'Building a Large Annotated Corpus of English: The Penn Treebank'. In: ***Computational Linguistics*** 19.2.

Mayer, Richard E (1999). 'Fifty Years of Creativity Research'. In: ***Handbook of Creativity***. Ed. by Robert J Sternberg. New York: Cambridge University Press. Chap. 22, pp. 449–460.

Mayhaymate (2012). ***File:PageRank-hi-res.png***. URL: https://commons.wikimedia.org/wiki/File:PageRank-hi-res.png (visited on 18/10/2016).

Michelsen, Maria Hagsten and Ole Bjorn Michelsen (2016). ***Regex Crossword***. URL: http://regexcrossword.com/ (visited on 19/10/2016).

Microsoft (2012). ***Bing Fact Sheet***.

Miller, George A. (1995). 'WordNet: a lexical database for English'. In: ***Communications of the ACM*** 38.11, pp. 39–41 (cit. on p. 19).

Miyamoto, Sadaaki (1988). ***Information Retrieval based on Fuzzy Associations***.

– (2010). ***Fuzzy Sets in Information Retrieval and Cluster Analysis (Theory and Decision Library D)***. Springer, p. 276.

Miyamoto, Sadaaki and K Nakayama (1986). 'Fuzzy Information Retrieval Based on a Fuzzy Pseudothesaurus'. In: ***IEEE Transactions on Systems, Man and Cybernetics*** 16.2, pp. 278–282.

Nick, Z.Z. and P. Themis (2001). 'Web Search Using a Genetic Algorithm'. In: ***IEEE Internet Computing*** 5.2, pp. 18–26.

Nicole (2010). ***The 10 Most Incredible Google Bombs***.

Pease, Alison and Simon Colton (2011). 'On impact and evaluation in Computational Creativity : A discussion of the Turing Test and an alternative proposal'. In: ***Proceedings of the AISB***.

Pease, Alison, Simon Colton et al. (2013). 'A Discussion on Serendipity in Creative Systems'. In: ***Proceedings of the 4th International Conference on Computational Creativity***. Vol. 1000. Sydney, Australia: University of Sydney, pp. 64–71.

Pease, Alison, Daniel Winterstein and Simon Colton (2001). 'Evaluating Machine Creativity'. In: ***Proceedings of ICCBR Workshop on Approaches to Creativity***, pp. 129–137.

Piffer, Davide (2012). 'Can creativity be measured? An attempt to clarify the notion of creativity and general directions for future research'. In: ***Thinking Skills and Creativity*** 7.3, pp. 258–264.

Polya, George (1957). ***How To Solve It***. 2nd. Princeton, New Jersey: Princeton University Press.

Project, NLTK (2016). ***Natural Language Toolkit***. URL: http://www.nltk.org/ (visited on 18/10/2016) (cit. on p. 13).

Ritchie, Graeme (2001). 'Assessing creativity'. In: ***AISB '01 Symposium on Artificial Intelligence and Creativity in Arts and Science***. Proceedings of the AISB'01 Symposium on Artificial Intelligence, Creativity in Arts and Science, pp. 3–11.

– (2007). 'Some Empirical Criteria for Attributing Creativity to a Computer Program'. In: ***Minds and Machines*** 17.1, pp. 67–99.

– (2012). 'A closer look at creativity as search'. In: ***International Conference on Computational Creativity***, pp. 41–48.

Schmidhuber, Juergen (2006). ***New millennium AI and the Convergence of history***.

Schuetze, Hinrich (1998). 'Automatic Word Sense Discrimination'. In: ***Computational Linguistics***.

Schuetze, Hinrich and Jan Pedersen (1995). ***Information Retrieval Based on Word Senses***.

Shu, Bo and Subhash Kak (1999). 'A neural network-based intelligent metasearch engine'. In: ***Information Sciences*** 120.

Srinivasan, P (2001). 'Vocabulary mining for information retrieval: rough sets and fuzzy sets'. In: ***Information Processing and Management*** 37.1, pp. 15–38.

Sutcliffe, Alistrair and Mark Ennis (1998). 'Towards a cognitive theory of information retrieval'. In: ***Interacting with Computers*** 10, pp. 321–351.

Taye, Mohammad Mustafa (2009). 'Ontology Alignment Mechanisms for Improving Web-based Searching'. PhD thesis. De Montort University.

Turing, Alan (2009). 'Computing Machinery and Intelligence'. In: ***Parsing the Turing Test***. Ed. by Robert Epstein, Gary Roberts and Grace Beber. Springer. Chap. 3, pp. 23–66 (cit. on p. 36).

University, Princeton (2010). ***What is WordNet?*** URL: http://wordnet.princeton.edu (visited on 20/10/2016).

Varshney, Lav R et al. (2013). 'Cognition as a Part of Computational Creativity'. In: ***12th International IEEE Conference on Cognitive Informatics and Cognitive Computing***. New York City, USA, pp. 36–43.

Ventura, Dan (2008). 'A Reductio Ad Absurdum Experiment in Sufficiency for Evaluating (Computational) Creative Systems'. In: ***5th International Joint Workshop on Computational Creativty***. Madrid, Spain.

Verne, Jules (2010). ***A Journey to the Interior of the Earth***. Project Gutenberg.

Vries, Erica de (1993). 'Browsing vs Searching'. In: ***OCTO report 93/02***.

Wickson, F., A.L. Carew and A.W. Russell (2006). 'Transdisciplinary research: characteristics, quandaries and quality'. In: ***Futures*** 38.9, pp. 1046–1059 (cit. on p. 36).

Widyantoro, D.H. and J. Yen (2001). 'A fuzzy ontology-based abstract search engine and its user studies'. In: ***10th IEEE International Conference on Fuzzy Systems*** 2, pp. 1291–1294.

Wiggins, Geraint A (2006). 'A preliminary framework for description, analysis and comparison of creative systems'. In: ***Knowledge Based Systems*** 19.7, pp. 449–458.

KTHXBYE

Go to TOC