Spring 2011

Optimization of textual affect entity relation models

Ajith Cherukad Jose

Department:

OPTIMIZATION OF TEXTUAL AFFECT ENTITY RELATION MODELS

by

AJITH CHERUKAD JOSE

A THESIS

Presented to the Faculty of the Graduate School of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE IN COMPUTER SCIENCE

2011

Approved by

Dr. Daniel Tauritz, Advisor
Dr. Jack Schryver
Dr. Ali Hurson

ABSTRACT

Affective computing is a recent research area in computer science which deals with the design and development of systems that can recognize, interpret and process human affects/emotions. Various research projects in the past have focused on affect sensing and processing raw textual data. One such research effort conducted at the Oak Ridge National Laboratory (ORNL) has introduced an affect propagation algorithm which can generate affective relationships between entities contained in a given textual document. The algorithm depends upon a set of real-valued numeric parameters for which the best possible values are unknown.

This thesis describes three different contributions to ORNL's research project. Firstly, the affect propagation algorithm was implemented along with a visualization environment. Secondly, an experimental framework was created for comparison of different algorithms to optimize the affect propagation algorithm parameters. A benchmark system was established for this purpose. Thirdly, different optimization algorithms were implemented to optimize the affect propagation algorithm. The optimization algorithms included variants of stochastic hill climbing, simulated annealing and evolutionary algorithms.

This thesis explores the use of a diversity maintained evolutionary algorithm to find the optimal parameter set for the affect propagation algorithm. A fitness sharing scheme has been adopted to maintain population diversity of the evolutionary algorithm. Statistical experimental studies are presented which show that the diversity maintained evolutionary algorithm performs best, followed by the adaptive simulated annealing algorithm, with respect to the best fitnesses achieved.

ACKNOWLEDGMENTS

Firstly, I would like to thank my advisor Dr. Daniel Tauritz for all the help and guidance he has provided throughout the research.

The internship I did at the Oak Ridge National Laboratory was the motivating factor behind the inception of this thesis. I am thankful to my mentor, Dr. Jack Schryver for all the support and directions he provided during my term at the laboratory. I would also like to thank all the scientists who have worked with me.

Finally, I would like to thank all of my family and friends for all the encouragement they gave me.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

1. INTRODUCTION

This section introduces the ORNL research project, its motivation, related work and the optimization problem under consideration in this thesis. Section 1.1 explains the motivation behind the research. Section 1.2 introduces the problem statement. Section 1.3 elaborates previous research work done in this area. Section 1.4 explains the challenges involved in the optimization problem pertaining to the research project. Section 1.5 describes the contributions of this thesis towards the research project.

1.1. MOTIVATION

In contrast to popular beliefs, it has been found that human affects and emotions play a pivotal role in rational thinking and social decision making [23, 28].

Quoting from the best-selling book "Descartes Error" by neuroscientist Antonio Damasio [7],

"Before you reason toward the solution of any given problem, something quite important happens: when the bad outcome connected with a given response option comes into your mind, however fleetingly, you experience an unpleasant feeling. The prefrontal cortices of the human brain consists of somatic markers - special instance of feelings generated from secondary emotions - plays a major role here. The somatic markers forces attention on the negative outcome to which a given action may head, and functions as an automated alarm signal which says : beware of danger ahead if you choose the option which leads to this outcome. The signal may lead you to reject, immediately, the negative course of action and thus make you choose among other alternatives. Similarly, when a positive somatic marker is juxtaposed, it becomes a beacon of incentive."

It has been observed from Damasio's case studies that people with impaired pre-frontal cortices, i.e., having damaged emotional systems, display gross defects of planning, judgement and social appropriateness. These defects were caused due to their inability to respond emotionally to the content of their thoughts.

The following two paragraphs are adopted with minor modifications from an ORNL technical report [27] which was co-authored by the author of this thesis.

Human affects mediates group communication and they have the innate capability of influencing social network processes - group formation, group recruitment, intergroup conflicts, intergroup threats, group schism and dissolution. Shared beliefs and attitudes increase group cohesion and group loyalty. Similary, in-group bias, out-group antipathy and factionalization are the results of emotional disruptions inside groups. In recent experimental tests, it has been found that sociopolitical concepts are affectively charged and that this affective charge gets activated within a short time period. Most citizens of a country, especially those with strong political attitudes, are biased information processors [13].

A significant proportion of information coming from groups and social networks, especially communications from internet sources such as blogs, email, forums, tweets and chat, is textual. The ability to extract from text, topical information usable in social network analysis, is the first step in deep social network analysis. Since affects contain important facts pertaining to group states and processes, much of our communications relies on the successful transmission, reception, identification, and interpretation of affective states. Therefore, a critical element of deep social network analysis will be the automated extraction and classification of affects toward various entities of interest, based on an a priori defined basic affect set.

1.2. PROBLEM STATEMENT

For a given text document, we need to determine the affective relationships between the entities referred inside the document.

1.2.1. Named Entities. Named entities as defined by the CoNLL-2003 Shared task [26] are proper names in the document which correspond either to

- Organization : named corporate, governmental, or other organizational entity
- Person : named person or family
- Location : name of politically or geographically defined location (cities, provinces, countries, international regions, bodies of water, mountains, etc.)

Besides named entities, we consider events, topics, and issues also as simply 'entities'.

1.2.2. Affect Categories. According to the Ortony, Clore and Collins (OCC) emotion model [20], human affects can be classified into 22 categories. The

different affect categories are shown in Table 1.1. Binary affects are the ones that act between two entities while the unary ones applies only to a single entity. Half the affect categories have positive attitude and the remaining half is negative.

Table 1.1: Table of affect categories

Unary		**Binary**	
Positive	**Negative**	**Positive**	**Negative**
Joy	Distress	Happy-for	Resentment
Pride	Shame	Pity	Gloating
Gratification	Remorse	Admiration	Reproach
Satisfaction	Disappointment	Love	Hate
Relief	Fear-confirmed	Gratitude	Anger
		Hope	Fear

1.2.3. Affect Entity Relationships. The different entities contained in a document can be related with each other in terms of the affect categories. Depending on whether the relationship applies to a single entity or two, the affect entity relationship can be either unary or binary. The affect propagation algorithm developed at ORNL is an algorithm which employs a mathematical approach to derive affect entity relationships from a given input text document.

1.2.4. Need for Optimization. The affect propagation algorithm is controlled by a set of real-valued numeric parameters. A set of manually set parameter values were assigned to the parameters for testing. However, the entity relationship diagrams generated from the manual parameters consisted of a large number of irrelevant entity relations. Every entity in the input document was related to every other entity in the document. Based on this it was postulated that to achieve meaningful relationships, the numeric parameters of the affect propagation algorithm need to be optimized.

1.3. RELATED WORK

This section is adopted with minor modifications from an ORNL technical report [27] which was co-authored by the author of this thesis.

Textual semantic analysis can be broadly classified into two categories: denotation and connotation. Denotation refers to the direct meaning of text whereas connotation refers to the affective or associational meaning of text. The connotative aspect of text could be understood only by a precise understanding of the words involved, the implicit commonsense knowledge and the affective relationship shared by different entities in the text.

Early work on connotative analysis has been in the area of sentiment analysis which is the assessment of opinion polarity with respect to a whole document or a particular topic [21]. An important application of this research is opinion mining: the automated classification of customer feedback on products [21]. The analysis of complex group dynamics cannot be sufficiently understood by polarity alone. A detailed analysis of affective meaning is required to fully understand group processes underlying textual data.

There has been some effort put to extract affective meaning from text. An important development was WordNet-Affect, which extends WordNet by defining a hierarchy for affective meaning [30, 31]. The Linguistic Inquiry and Word Count (LIWC) [22] was a significant attempt to provide in a software program a psycholinguistic summary of text characteristics at the document level. LIWC performs keyword spotting of affective processes that includes positive emotions in addition to negative emotions such as anxiety, anger, and sadness. Some investigators have attempted to identify affect in text [2, 3, 17, 4, 11, 12] using either an affect lexicon or supervised learning techniques. These efforts have mostly been directed at document-level assessment of affect. Abbasi and Chen [1] studied the presence of violence and hate-related affect in web forums operated by extremist political groups. Their research elaborates the relevance of affect extraction techniques to the objectives of intelligence analysis.

Liu, Lieberman, and Selker [12] recognized the significance of commonsense knowledge in extracting the connotative/affective meaning from text. Their approach utilized affective knowledge contained in the Open Mind Common Sense (OMCS) database in the construction of their affect lexicon. In particular, they were interested in the kind of real-world knowledge that revealed common-place affective stances toward situations, things, people, organizations, concepts, and events. Sen-

tences that contained affective meaning in OMCS were identified by spotting the "emotion ground" signified by affect-saturated keywords. These emotion grounds were used in models or templates to subsequently extract affect from documents. A similar approach grounded in an affect lexicon augmented with commonsense knowledge from OMCS was followed by [11]. However, their efforts to incorporate commonsense knowledge were also burdened by the relatively unstructured sentence structures found in the OMCS database.

The affect propagation algorithm used for the thesis research has utilized a cognitive theory of emotion (affect) as the framework for modeling affective meaning in text. According to the appraisal theory [19], affect is the response to cognitive evaluations made by individuals and groups to outcomes associated with self, agents, objects, and events. For example, admiration/reproach is a valenced reaction to an approved/disapproved action of another agent. Pride/shame result from a similar evaluation of an action focusing on the self as an agent. O'Rorke and Ortony's taxonomy considers 23 affects resulting from different types of appraisal, but of course this number is somewhat arbitrary and the set can be enlarged.

All previous work in the extraction of affect in text has focused on classification of documents or sentences into one of several affect categories, e.g., basic emotions. We take this analysis a step further by considering the affective relationships between entities in a document. Affects and extracted entities are represented as an affect-entity relation network.

1.4. OPTIMIZATION CHALLENGE

The affect propagation algorithm is controlled by 11 real-valued parameters. The search space of the parameter set is continuous (each parameter is in the range <0,1> with infinite possible values) so the number of possible states is infinite.

Consider a brute-force search with 13 possible values (0.001, 0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8., 0.9, 0.99, 0.999) for each parameter. Since there are 11 parameters in total, this would mean that the total number of evaluations would be 13^11 = 1792160394037. Say for example, if a single evaluation takes 0.2 seconds (on a Pentium IV 3.2 GHz machine with 3 GB RAM running Windows 7 Operating System), the total number of hours taken for the complete execution would be 99564466 hours.

Therefore, a pure brute-force approach is not practical here.

Now, let us consider n number of search points in the search space and from these arbitrary points, let us perform hill climbing search to find a parameter set with better fitness. From experimental study (discussed in Section 5.2) it has been found that these n number of hill climbers (starting from n random search points) do not lead to a single point on the search space, indicating a multi-modal search scenario.

Therefore, a pure hill-climbing approach is inappropriate.

It has also been seen from experimental study (discussed in observations from Section 4.5) that the eleven parameters are dependent on each other. Therefore, we cannot find an optimal solution to the problem by greedily optimizing a single parameter at a time.

Thus, the complexity of the optimization problem we are facing here is combinatorial in nature. This indicates the use of computationally intelligent algorithms like simulated annealing and evolutionary algorithms to find the best solution.

1.5. RESEARCH CONTRIBUTIONS

Affect entity relation modelling is a broad area of research. The contribution of this thesis is towards three different aspects of the ORNL research project.

Firstly, the affect propagation algorithm was implemented in Java. The different steps of the algorithm consist of complex mathematical formulae. The Java programming language was chosen for implementation because of the availability of a wide range of open source APIs and packages, especially for drawing graphical diagrams and user interfaces. The open source package JUNG [18] was used to draw graphical networks on the application GUI. Appropriate data structures for better performance of the mathematical equations were also needed. For this purpose, most of the Java data structures used involved hashing. However, an initial implementation of the algorithm had a bad performance overhead and it took more than a minute for a single execution of the affect propagation algorithm. Performance tuning of the application was done by improving the execution times of parts of the algorithm. Algorithm steps which took execution times of $O(n^3)$ and $O(n^2)$ were modified to have execution times of $O(n^2)$ and $O(n)$ respectively, resulting in 30 second evaluation time.

Secondly, testing the effectiveness of the algorithm and the optimization of its

parameters required a benchmark system to be created. The affect entity relation models extracted using the affect propagation algorithm had to be compared with benchmark entity relation models manually tagged by expert analysts. To facilitate the tagging process, a software application, namely the affect entity tagger, was developed. Using this application, an analyst could manually draw affect entity relationship diagrams for a given benchmark set of documents. These relationship diagrams could further be saved as xml files for a comparative analysis with the algorithm generated diagrams. Further details of the benchmark system are provided in Section 3.1.

Thirdly, and most importantly, the contribution of this thesis is towards the optimization of the affect propagation algorithm. Various optimization strategies were investigated after analysing the search space using random search and grid search heuristics (see Section 4). Among the different optimization algorithms included are versions of hill-climbing, simulated annealing and evolutionary algorithms. Section 5 elaborates on the optimization algorithms used. The time taken for fitness evaluations was expensive, each being around 30 seconds. So, the experimental setup for running the optimization algorithms also required a high-speed environment. For this reason, high speed computing clusters were used.

2. AFFECT PROPAGATION ALGORITHM

This section explains with an example the affect propagation algorithm. Section 2.1 gives an example of an Entity Relationship Model applied to an example document. Section 2.2 explains the affect propagation algorithm using the example document. Section 2.3 describes the implementation details and Section 2.4 elaborates on the parameters used in the algorithm.

2.1. EXAMPLE AFFECT ENTITY RELATIONSHIP MODEL

Let us consider the text shown in Appendix B.

After reading the textual content, a document analyst can deduce United States, Russia, Chechens and Putin as the named entities. Besides this the author of the document is a default entity.

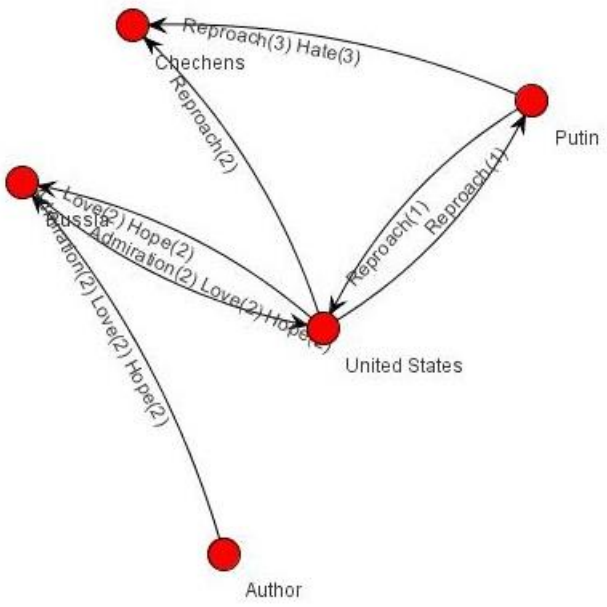


Figure 2.1: Example affect entity model

Overall, the textual content depicts a hopeful affective relationship between the two countries, United States and Russia. It also discusses a reproach feeling towards the Chechens. The author expresses admiration, hope and love towards Russia. Considering all possible affective evaluations, the analyst could produce an affect entity relationship diagram as shown in Figure 2.1.

2.2. BASE ALGORITHM

The affect propagation algorithm developed at ORNL forms the basis for extracting affective relationships between entities in text documents without manual intervention. In summary, it consists of the following steps:

1. Affective Words and Named Entity Detection

   A seedlist of affective words for the 22 different affective categories from the OCC Emotion Model [20] was created. The seedlist consists of all affective words belonging to the 22 affect categories from the Oxford English dictionary, as determined by a human expert specialised in psycholinguistic human affects at ORNL. The seedlist words are further searched in the lexical database WordNet and a network graph is formed for all the direct synonyms of the seedlist words. For all the 22 affect categories, 22 different affective networks are generated based on synonyms extracted via word sense (meaning or sense of the word usage) detection from WordNet. For example, let us consider the affect category 'Hope, Hopeful'. A partial seedlist of affect words belonging to the category is shown in Appendix C. Each line in the seedlist consists of the affective word with the part of speech in brackets and a number following a hash symbol (#) representing different WordNet word senses of the respective word. These words are searched in the WordNet database with respect to word sense, part of speech and thereby all the direct synonyms are found. Figure 2.2 represents the corresponding network graph generated. Whenever the input document is parsed, the algorithm searches from the network graph, all the affective words appearing inside the document and they are highlighted. In the example document from Appendix B, the words 'expected', 'committed', 'prospect', 'opportunity', 'certain' and 'encourage' are highlighted for the affect category 'Hope, Hopeful'. To determine the intensities of the affect words in the affect network graph,

the PageRank algorithm [6] is used. The PageRank algorithm attaches a score denoting a degree of authority of the affect word on the affect network graph. For example, the word 'opportunity' has a PageRank score of 0.003 (calculated using the PageRank function in the OpenNLP package [34, 29]) inside the affect network graph of 'Hope, Hopeful'. All the PageRank scores in an affect network graph add to 1.

The named entities inside the document are extracted using the Learning Based Java Named Entity Tagger [24].

2. Qualifier and Negation Intensity Calculation

   The qualifiers and negations of the affect words are detected in this step. The effective intensity of the qualifiers and negations on the different affect words is calculated relative to the PageRank score of the affect words from the affect network graph. In the example document, the qualifier 'rare' appears before the word 'opportunity' and it reduces the intensity of the affect 'hope'.

3. Edge Weight Calculation

   A network graph of all words from the input document is generated. Depending on the occurrence of affect words, entities, periods, commas and conjunctions, edge weights are determined for the graph. In general, punctuation reduces edge weights. For example, suppose a comma appears after an affect word in a sentence. The context of the sentence after the comma would be less influenced by the affect word, which is reflected by a lower edge weight. Currently, the graph is generated with words appearing in the order they exist inside the document. Future changes of the algorithm would have aggregations happening at specific word vertices. The second paragraph of the example generates the graph as shown in Figure 2.3. The paragraph contains brackets which reduces the corresponding weights given to the edges.

4. Random Walk With Restart

   A Random Walk With Restart (RWR) algorithm is applied to the word graph generated from the previous step. With each affect word token as the restarting node, forward and reverse random walk ranks of different named entities are

generated from the word graph. The forward and reverse random walk scores are combined to determine unary and binary affective relationship scores of entities. For example, the affect word 'regard' appears as the last word in the second paragraph of the example document. With this word as the restarting node, the random walk ranks for the entities like 'Putin' and 'Russia' are calculated. In the second paragraph, since 'regard' comes after the entities 'Putin' and 'Russia', only the reverse random walk ranks are considered for them.

5. Combine Evidences

   Many documents contain multiple affect tokens representing each affect category. Therefore, when considering the affective relationship between entities with respect to a category, we must aggregate the evidence from each affect token in that category. A model of reasoning with belief functions is used for this purpose [14]. The positive and negative evidence of entity relationship scores for the different affect categories are calculated. The evidence scores are combined and represented as unary/binary relationships between entities. These relationships are represented as edges on the final affect-entity graph network. In the example document, the affect tokens 'expected', 'committed', 'prospect', 'opportunity', 'certain' and 'encourage' represent the 'Hope, Hopeful' category. The positive and negative evidences for each of them are different with respect to the surrounding context. All such evidence is combined and final unary/binary relationship vectors between the different entities are generated for the affect category.

2.3. SOFTWARE IMPLEMENTATION

The software tool which implements the affect propagation algorithm is called TEAMSTER. It is implemented using Java 1.5. The following are used for their respective purposes:

- JUNG Package - drawing network graphs and using graph-walk algorithms [18].
- OpenNLP Package - Tokenizers, Part Of Speech (POS) Tagging and Sentence detectors [34, 29].

- WordNet - Searching for synonyms in the the synonym network graph [15].
- Learning Based Java Entity Tagger - To identify all the named entities in the documents [24].

2.4. RESEARCH QUESTIONS

The Affect Propagation algorithm uses a set of real valued parameters for which the best possible values are unknown. Table 2.1 lists the different real valued parameters in use.

For different parameter list vectors given manually, the algorithm generates different Affect-Entity graphs. This leads us to the following four research questions:

1. What is the best possible parameter set for the Affect Propagation algorithm?
2. What fitness criterion determines the best parameter set solution vector?
3. Which optimization algorithm produces the best results?
4. Is the result of the affect propagation algorithm employing the optimal parameter set of sufficient quality to be useful?

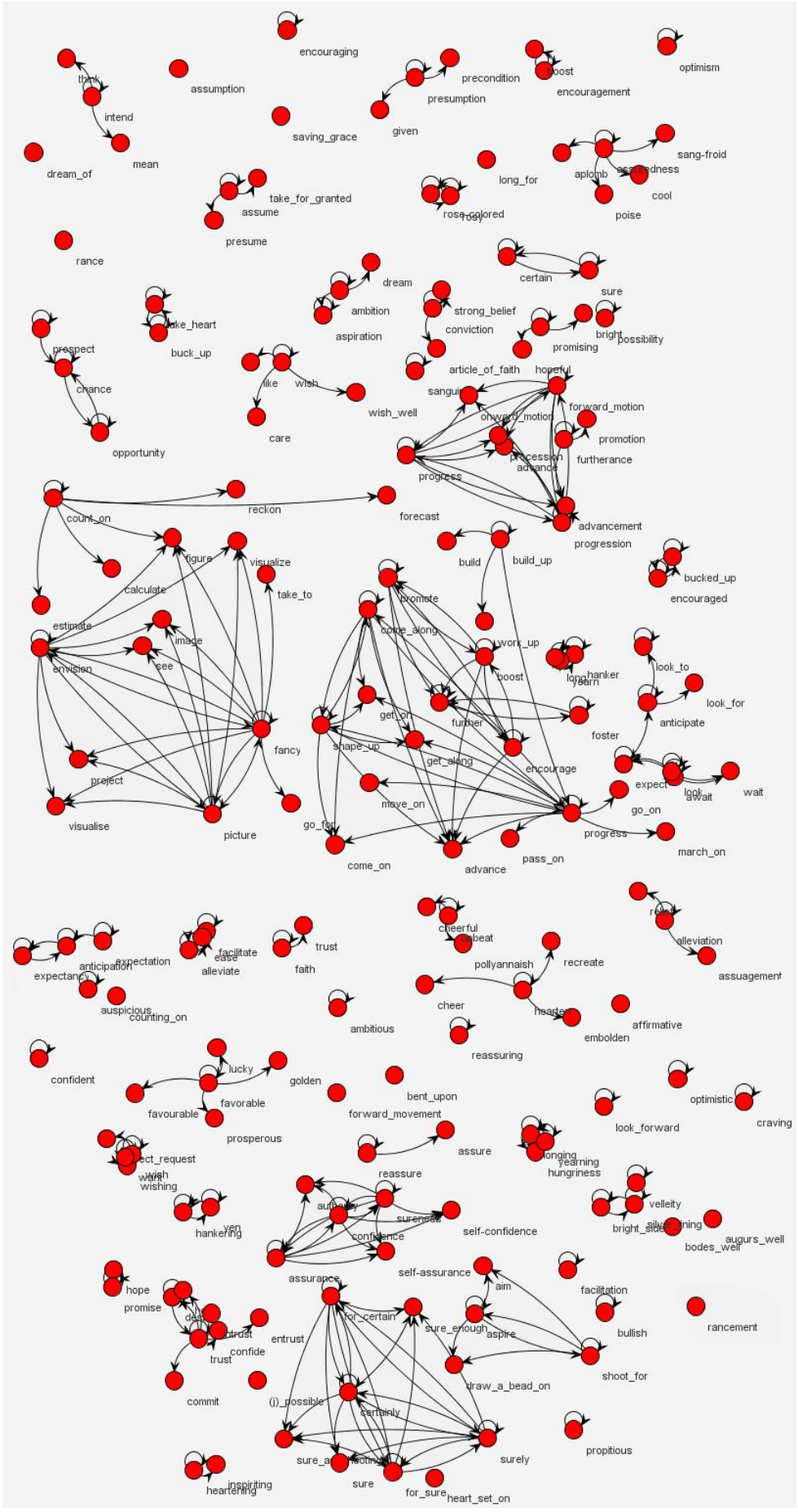


Figure 2.2: Affect network graph for the affect category ‘Hope, Hopeful’

Table 2.1: Optimization parameter set

No.	Parameter Name	Algorithm Step	Range	Dependency
1.	affectWeightage	Two	[0,1]	Importance of the affect word considered in the document
2.	entityWeightage	Two	[0,1]	Importance of the entity considered in the document
3.	sentenceEndWeightage	Two	[0,1]	Sentence endings
4.	commaWeightage	Two	[0,1]	Comma in the sentence considered
5.	quoteWeightage	Two	[0,1]	Quotes in the sentence considered
6.	RWR Algorithm Restart probability (alpha)	Three	(0,1]	Aggregation criteria of initial document graph
7.	Author threshold value (tAuthor)	Three	(0,1)	Amount of weightage given to author relationships
8.	Binary threshold value (tBinary)	Three	(0,1)	Amount of binary affect entity intensity to be considered
9.	Unary threshold value (tUnary)	Three	(0,1)	Amount of unary affect entity intensity to be considered
10.	rootTransformationConstant	Three	(0,1)	Amount of normalization required for benchmark tagger levels
11.	ghostAuthorRank	Three	(0,1)	Weightage of author when author is not mentioned explicitly in the document

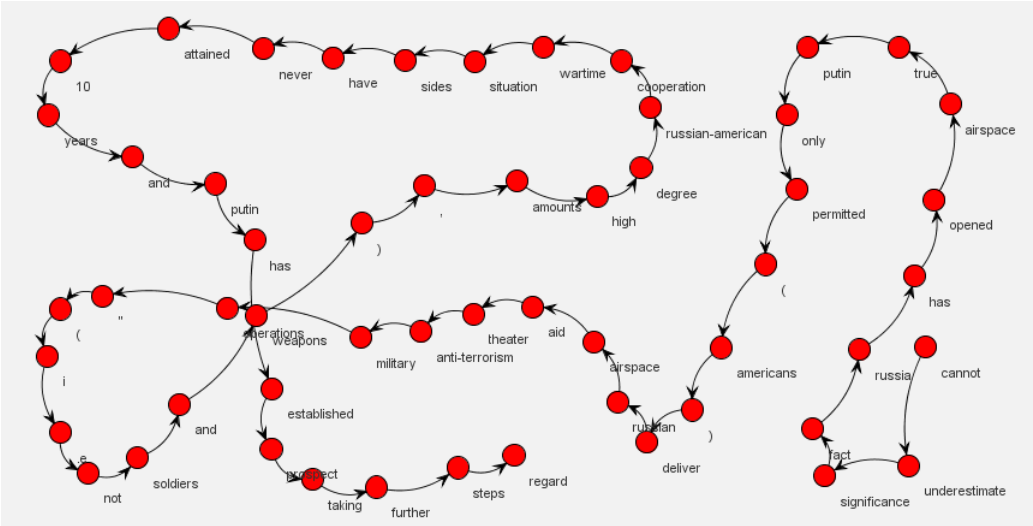


Figure 2.3: Document graph of second paragraph inside the example document

3. AFFECT PROPAGATION ALGORITHM OPTIMIZATION

This section explains in detail the optimization problem along with the benchmarking system. Section 3.1 elaborates the features of the benchmarking system. Section 3.2 and Section 3.3 define the objective function and fitness function respectively. Section 3.4 explains the so-called zero vector problem. Section 3.5 explains the improved fitness function and Section 3.6 accounts for the best solution with respect to the new fitness criteria.

3.1. BENCHMARKING

For finding the best parameter set for the Affect Propagation algorithm, there is a need for a benchmark set of document affect entity annotations to which the algorithm annotations can be compared.

3.1.1. MPQA Benchmark Corpus. The MPQA Opinion Corpus developed at the University of Pittsburgh, is a collection of news articles from a wide variety of news sources gathered for annotating opinions and other private states (i.e., beliefs, emotions, sentiments, speculations etc.) [33, 5]. Articles from the MPQA corpus were selected based on affect word occurrence. Articles with multiple affect words and emotional statements were selected over documents without affective relationships. The current benchmark document set consists of 50 documents. The documents had to be annotated manually to determine all possible entity relationships. Manually marking entities and affects on paper was a tedious task. There was a need to come up with an easier way of generating the document affect entity relationships.

3.1.2. Affect Entity Tagger. The Affect Entity Tagger is the software tool developed for manually annotating the benchmark set of documents.

The software tool facilitates:

- Marking affect words in the benchmark documents
- Marking entities and their synonymic labels in the benchmark document
- Annotating binary relationships between entity pairs as affect category rankings
- Annotating unary relationships of entities as affect category rankings

- Saving document annotations as xml files for future use
- Modifying saved document annotations

The Affect Entity Tagger was programmed in Java. A Model View Controller design pattern was used for the application design. The affects, entities and entity relationships form different models in the application that are visualized by the GUI panels which form the application view. The JUNG package [18] was used to generate the entity relationship graphs.

3.1.3. Entity Relationship Vectors. The entity relationships are represented as 22 element Entity Relationship (ER) vectors on the Affect Entity Tagger. Each element represents an affect category from the OCC Emotion Model. Each element has a value in the range -5 to 5, which represents the intensity of the affective relationship in that affect category.

The following are some observations from the benchmark ER vectors:

- A negative affect intensity does not necessarily mean that the affective relationship is of the opposite affect category. For example, consider the sentence:

  "John does not hate Mary."

  The resulting affective relationship would have a negative value for the affect hate from John towards Mary. But, that does not mean that there exists an opposite affective relation i.e., love from John to Mary.

- In a majority of cases, most of the elements of an ER vector are zeroes. Generally, there does not exist documents which deduce an ER vector between two entities with more than four non-zero affect elements.

- There exist entities which do not have any affective relationship with other entities in a document. This happens in the absence of affectively charged statements pertaining to the respective entity.

Affect categories are either binary (affect between two entities) or unary (affect on a single entity). They could be either positive or negative. The binary/unary and positive/negative distinctions are also embedded into the ER vectors.

ER vectors are generated by the affect propagation algorithm. The algorithm ER vectors have to be compared with the benchmark ER vectors for determining the effectiveness of the algorithm parameters. A fitness criteria is required to evaluate the quality of algorithm parameters.

3.2. OBJECTIVE FUNCTION

The quality of a solution generated by the affect propagation algorithm is inversely proportional to its error. The amount of error generated by a particular parameter set is decided based on a distance measure between the algorithm ER vectors and the benchmark ER vectors. We use a modified Euclidean distance as the distance measure between the ER vectors. The raw affect vector values undergo a sign-preserving square-root transformation to emphasize distance between low-valued scores, and minimize distance between high-valued scores. For example, the distance between 0 and +1 should exceed the distance between +4 and +5, as the former represents a difference between relationship and non-relationship, whereas the latter signifies only a slight difference between two very positive scores.

Let,

a be the 22 element ER vector obtained from the base algorithm,

b be the respective 22 element ER vector obtained from the benchmark

Modified Euclidean Distance,

$$d(a,b) = \sqrt{(\sqrt{a_1} - \sqrt{b_1})^2 + (\sqrt{a_2} - \sqrt{b_2})^2 + ... + (\sqrt{a_{22}} - \sqrt{b_{22}})^2} = \sqrt{\sum_{i=1}^{22} (\sqrt{a_i} - \sqrt{b_i})^2} \tag{1}$$

Let,

ER be the total number of possible entity relations,

E be the total number of possible entities.

Objective function,

$$O = \sum_{i=1}^{ER} \frac{d_i(a,b)}{E^2} \tag{2}$$

Algorithms employed to optimize the parameters must minimize the objective function. In the ideal case (no difference between algorithm ER vectors and benchmark ER vectors), the objective function returns zero.

3.3. FITNESS FUNCTION

Per definition, the fitness of a particular solution increases when a better solution is found. So, a fitness function of the optimization problem would be the negation of the previously discussed objective function.

Fitness function,

$$F = -O = -\sum_{i=1}^{ER} \frac{d_i(a,b)}{E^2} \tag{3}$$

3.4. ZERO VECTOR PROBLEM

On careful algorithmic analysis it was found that a majority of good fitness values obtained were generated when all the algorithm ER vectors were zero vectors. This would mean that there does not exist any affective relationships between the different document entities.

The reason for finding the zero vector solution as the best solution is that the number of affective relationships the benchmark has found is very few compared to the total number of possible affective relationships. Also, the number of affects pertaining to a single entity relationship is less than 5 compared to the total possible 22 affects.

For example, consider an example document having 30 different named entities. The number of possible entity relations is $30^2 = 900$. From the manually tagged affect entity graph, it can be observed that the document has only 40 entity relations detected. So, there would be $900 - 40 = 860$ benchmark ER vectors which are zero vectors. Also, the document does not have more than 3 affect categories detected per entity relationship. Therefore, zero vector ER solutions would give a higher fitness score compared to most other ER vectors considered.

This problem is very similar to the famous evolutionary programming experiment described in [9]. In this experiment a finite state machine was evolved to predict

if the next input in a sequence of integers is a prime or not. Since the majority of integers are non-prime, the most simple of finite state machines which simply always predicts that the next input is non-prime will produce reasonably good results. Therefore, because it is very difficult to find better performing more complex finite state machines, the simple one-state finite state machine which always predicts non-prime will have a strong evolutionary advantage which was confirmed experimentally.

The parameter set solutions which could escape the zero vector space could be considered the best solutions. However, this approach would bias the solutions to a fewer discovered affective relationships. In the same example, suppose the algorithm finds correctly one of the 40 right entity relations. This increases its fitness and it is higher than the zero vector solution. However, remaining 39 entity relations were not found. Now consider a solution which detects 30 correct entity relations, along with a set of 40 incorrect entity relations. Although 30 correct relations were found, the fitness score is lower in this case because of the occurence of the wrong ones. Thus, the fitness evaluation biases optimized solutions toward as few discovered affective relationships as possible. There is a need to change the fitness criteria in this aspect.

3.5. IMPROVED FITNESS FUNCTION

To address the zero vector problem, an improved fitness function was developed. The improved fitness score is the ratio of expected error of a random guesser to the actual error generated by the affect propagation algorithm. The fitness ratio has a semantic interpretation, in that fitness reflects the extent to which the algorithm can improve upon an appropriate random guesser.

Let,

R represent that the affective relation is present,

NR represent that the affective relation is not present,

p be the probability of guessing that a relationship exists between any two entities

q be the actual number of non-zero affective relationships in a document relative to the number of potential relationships (the latter is equal to E^2 where E is the number of entities in a document)

The random guesser (RG) is modeled as a sequence of independent Bernoulli

trials for each potential affective relationship where the choice objects are in the set R, NR. The value of RG on a single Bernouli trial obeys the following expression:

$$RG = \begin{cases} R & p \\ NR & 1-p \end{cases} \tag{4}$$

An appropriate random guesser is one that matches the algorithm's propensity to find a relation between an arbitrary entity pair. If r is the total number of non-zero relations found by an algorithm, then the average probability of finding a relationship between the entities, $\hat{p} = \frac{r}{E^2}$.

Table 3.1: Expected probabilitites arising from the random guess model

		Ground Truth(baseline)	
		R	**NR**
Random Guess	**R**	pq	$p(1-q)$
	NR	$q(1-p)$	$(1-p)(1-q)$

The probabilities off the main diagonal in Table 3.1 represent error outcomes. We have an exact value for the average error z, for the baseline corpus when the model always predicts a non-relationship ($z = 0.0262477$). These are the zero-vector solutions. A small proportion (q) of possible relations will actually be present, but most (*1-q*) will be non-relations. In the benchmark corpus $q = 0.10114$. We can break out the components of z and write $z = \epsilon \times q + 0 \times (1-q)$ where ϵ is the expected random guess error when a relationship is falsely predicted (false positive) or an actual relationship is missed by the algorithm (false negative). Therefore $\epsilon = \frac{z}{q}$. Further, let $0 < d < 1$ equal the expected proportional error experienced when a relationship is correctly predicted but is still quantitatively off the mark. The expected random guess error for a single relationship is:

$$
\begin{aligned}
E\left(RG\right) &= \left[0+\hat{p}\left(1-q\right)+q\left(1-\hat{p}\right)+d\hat{p}q\right]\times\epsilon \\
&= \frac{\left(\left[0+\hat{p}\left(1-q\right)+q\left(1-\hat{p}\right)+d\hat{p}q\right]\times z\right)}{q} \\
&= \frac{\left(\left[\hat{p}+q+\left(d-2\right)q\hat{p}\right]\times z\right)}{q} \\
&= \left\{1+\left[\frac{1+\left(d-2\right)q}{q}\right]\times\hat{p}\right\}\times z \\
&= \left\{1+\left[\frac{1+\left(d-2\right)q}{q}\right]\times\left(\frac{r}{\sum E^2}\right)\right\}\times z \qquad (5)
\end{aligned}
$$

Let $E\left(Alg\right)$ be the objective function score (modified average Euclidean error) of the affect propagation algorithm. The new fitness is given by:

$$
F' = \frac{1+E\left(RG\right)}{1+E\left(Alg\right)} \qquad (6)
$$

where, $E\left(RG\right)\geq 0$ and $E\left(Alg\right)\geq 0$, so $F'\geq 0$.

When $F'>1$, the algorithm predicts more accurately than a similar random guessor. The improved fitness function eliminates the bias against finding relations. Instead, it attempts to optimize the balance of false positives and false negatives, leading to an optimum number of found relations.

In order to compute the fitness ratio, we require an estimate of the parameter d. From an analysis of the available random solutions, the error for a correctly found relationship to have quantitatively bad values was evaluated. The value thus obtained is $d=0.44$.

3.6. FITNESS VALUE UPPERBOUND

Let us determine the fitness of the best possible solution for the benchmark set of documents. The total number of entity relations found is 487.

If there is an algorithm which finds exactly the same solution, then $r=487$.

From Section 3.5, we know:

- $d=0.44$

- $q = 0.10114226375908619$
- $r = 487$
- $\sum E^2 = 4815$
- $z = 0.2624773328544865$

Applying these values to Equation 5 we get:

$$E\left(RG\right) \approx \{1 + 8.327 \times 0.10114\} \times 0.2625 \approx 0.4834 \tag{7}$$

In the ideal case, we would find the exact same benchmark solution and the modified average Euclidean error $E\left(Alg\right)$ is 0.

Therefore,

$$F'_{best} = \frac{1 + 0.4834}{1 + 0} = 1.4834 \tag{8}$$

Although it is highly unlikely to find a solution which achieves the fitness F'_{best}, the purpose of different optimization algorithms employed should be to attain a fitness value closest to F'_{best}.

4. GRID SEARCH HEURISTICS

This section elaborates various grid search heuristics applied to analyse the optimization search space. Section 4.1 gives details in an analysis of the search space. Section 4.2 lists the different grid search approaches. Section 4.3 explains the results and a corresponding graphical analysis is given in Section 4.4. The final observations of the grid search experiments are given in Section 4.5.

4.1. SEARCH SPACE ANALYSIS

As discussed in Section 1.4, a true brute-force search over all possible points on the search space is infeasible for the optimization problem under consideration.

However, a biased grid search approach could be applied to analyse possible parameter values. Each parameter lies in the range <0,1>. The set of possible values taken for a single parameter are (0.001, 0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.99, 0.999) along with the current best solution obtained from random search and simulated annealing runs. Experimental trial runs indicated a higher probability for high quality values to be present near the boundaries, so a finer search resolution was employed near the boundaries.

4.2. GRID SEARCH APPROACHES

Six different approaches are considered for performing the grid search analysis. Due to time constraints, all of the grid search analyses are performed using four carefully selected benchmark documents representative of the entire set which have a medium range of entities and entity relations. The four documents used were selected by an expert analyst at ORNL.

4.2.1. Current Best Grid Search Analysis. An input parameter list CPL1 is taken which is the current best parameter list obtained from all previously done fitness evaluations. Every parameter is changed from the 13 possible values keeping the other parameters unchanged. For each parameter, the set of 13 parameter lists is evaluated and the parameter value with best fitness is saved. A parameter list CPL2 of these best values for the parameters is thus generated. A total of 13 x 11 = 143 evaluations are performed for the analysis.

4.2.2. Permuted Current Best Grid Search Analysis. From CPL1 and CPL2, all possible parameter lists are evaluated. Each parameter in the best parameter list is assumed to have values either from CPL1 or CPL2. So, a total of 2^11 = 2048 parameter lists are evaluated and a possible range of best parameter lists is deduced.

4.2.3. Forward Greedy Grid Search Analysis. An input parameter list FGPL1 is taken which is the current best parameter list obtained from all previously done fitness evaluations. Every parameter is changed from the 13 possible values. For each parameter, the set of 13 parameter lists is evaluated and the parameter value with best fitness is taken as next FGPL1. Thus, a total of 13 x 11 = 143 evaluations are performed. The final best fit parameter list is taken as FGPL2. This approach differs from Section 4.2.1 approach in that the intermediate parameter list obtained is retained for the next iteration.

4.2.4. Permuted Forward Greedy Grid Search Analysis. From FGPL1 and FGPL2, all possible parameter lists are evaluated. Each parameter in the best parameter list is assumed to have values either from FGPL1 or FGPL2. So, a total of 2^11 = 2048 parameter lists are evaluated and a possible range of best parameter lists is deduced.

4.2.5. Reverse Greedy Grid Search Analysis. The greedy grid search analysis is done in the reverse order of occurrence of the parameters. An input parameter list RGPL1 is taken which is the current best parameter list obtained from all previously done fitness evaluations. Every parameter is changed from the 13 possible values. For each parameter, the set of 13 parameter lists is evaluated and the parameter value with best fitness is taken as next RGPL1. The order of performing the greedy evaluations is in the reverse order of parameters (with respect to the order considered in approach discussed in Section 4.2.3). Thus, a total of 13 x 11 = 143 evaluations are performed. The final best fit parameter list is taken as RGPL2.

4.2.6. Permuted Reverse Greedy Grid Search Analysis. From RGPL1 and RGPL2, all possible parameter lists are evaluated. Each parameter in the best parameter list is assumed to have values either from RGPL1 or RGPL2. So, a total of 2^11 = 2048 parameter lists are evaluated.

4.3. GRID SEARCH RESULTS

Table 4.1 displays the approximate best parameter values for the first, third and fifth approaches.

Table 4.2, Table 4.3 and Table 4.4 represent best parameter lists for the permuted approaches; second, fourth and sixth approaches respectively.

The rows in bold represent parameter best values which are in similar range for all three approaches.

4.4. GRAPHICAL SEARCH SPACE ANALYSIS

From the previous section results, the scatter plots shown in Figure 4.1 and Figure 4.2 can be generated for individual parameters. The darker regions represent the part of the search space where the likelihood of finding the best parameter solutions is higher.

4.5. GRID SEARCH OBSERVATIONS

It is clear from the grid search experiments that most of the parameter best values are dependent on values assigned to the other parameters at that time. However, very few parameters showed better results most of the time when they are included in a particular range of values.

From the experiment results, we can say that when the parameters alpha, tBinary and ghostAuthorRank are very low, the fitness values are good.

Approximately, the best values for the parameters alpha and tBinary appear to lie in the range <0, 0.02>. The parameter ghostAuthorRank has a higher probability to lie in the range <0, 0.01>.

Table 4.1: Grid search analysis results of first, third, and fifth approaches

Parameter Name	**First Approach**			**Third Approach**			**Fifth Approach**		
	Parameter Value	**Fitness**	**No. of Relations**	**Parameter Value**	**Fitness**	**No. of Relations**	**Parameter Value**	**Fitness**	**No. of Relations**
affectWeightage	0.001	1.2016	81	0.001	1.2016	81	0.7	1.2173	73
entityWeightage	0.4	1.1904	93	0.9	1.2112	81	0.999	1.2187	73
sentenceEndWeightage	0.3	1.1957	94	0.99	1.2089	82	0.9	1.2171	72
commaWeightage	0.3	1.1898	98	0.6	1.2112	81	0.9	1.2171	72
quoteWeightage	**0.001**	**1.1716**	**98**	**0.001**	**1.2112**	**81**	**0.001**	**1.2168**	**71**
alpha	**0.001**	**1.2046**	**82**	**0.01**	**1.2044**	**89**	**0.01**	**1.2137**	**70**
tBinary	0.001	1.1716	98	0.001	1.2044	89	0.01	1.1964	63
tUnary	0.1	1.171	99	0.001	1.2044	89	0.5	1.1964	63
tAuthor	0.4	1.1753	98	0.1	1.2044	89	0.2	1.1955	64
rootTransformationConstant	**0.4**	**1.2007**	**86**	**0.4**	**1.1996**	**76**	**0.5**	**1.1955**	**64**
ghostAuthorRank	**0.001**	**1.157**	**99**	**0.001**	**1.1994**	**78**	**0.001**	**1.1561**	**99**

Table 4.2: Grid search analysis results of second approach

Parameter Name	**Parameter List1**	**Parameter List2**	**Parameter List3**
affectWeightage	0.001	0.001	0.001
entityWeightage	0.9644988	0.9644988	0.9644988
sentenceEndWeightage	0.9070164	0.9070164	0.9070164
commaWeightage	0.7128705	0.7128705	0.7128705
quoteWeightage	**0.6329861**	**0.6329861**	**0.001**
alpha	**0.0056172**	**0.0056172**	**0.0056172**
tBinary	0.001	0.001	0.001
tUnary	0.1	0.1	0.1
tAuthor	0.3225383	0.4	0.4
rootTransformationConstant	**0.3543188**	**0.3543188**	**0.3543188**
ghostAuthorRank	**3.67E-04**	**3.67E-04**	**0.001**
No. of Relations Found	**86**	**86**	**86**
Fitness	**1.21245**	**1.21245**	**1.21245**

Table 4.3: Grid search analysis results of fourth approach

Parameter Name	**Parameter List1**	**Parameter List2**	**Parameter List3**	**Parameter List4**
affectWeightage	0.001	0.001	0.001	0.001
entityWeightage	0.9	0.9	0.9	0.9
sentenceEndWeightage	0.9070164	0.9070164	0.99	0.99
commaWeightage	0.7128705	0.6	0.6	0.6
quoteWeightage	**0.001**	**0.001**	**0.001**	**0.001**
alpha	**0.0056172**	**0.0056172**	**0.0056172**	**0.0056172**
tBinary	0.001	0.001	0.001	0.001
tUnary	0.118542	0.118542	0.118542	0.118542
tAuthor	0.3225383	0.1	0.3225383	0.1
rootTransformation-Constant	**0.3543188**	**0.3543188**	**0.3543188**	**0.3543188**
ghostAuthorRank	**3.67E-04**	**3.67E-04**	**3.67E-04**	**3.67E-04**
No. of Relations Found	**83**	**83**	**83**	**83**
Fitness	**1.21447**	**1.21447**	**1.21447**	**1.21447**

Table 4.4: Grid search analysis results of sixth approach

Parameter Name	**Parameter List1**	**Parameter List2**	**Parameter List3**	**Parameter List4**	**Parameter List5**
affectWeightage	0.667086	0.667086	0.667086	0.667086	0.667086
entityWeightage	0.999	0.999	0.999	0.999	0.999
sentenceEndWeightage	0.9070164	0.9070164	0.9070164	0.9	0.9
commaWeightage	0.9	0.9	0.9	0.9	0.9
quoteWeightage	**0.6329861**	**0.6329861**	**0.6329861**	**0.6329861**	**0.6329861**
alpha	**0.01**	**0.01**	**0.01**	**0.01**	**0.01**
tBinary	0.013663	0.01	0.01	0.013663	0.01
tUnary	0.5	0.5	0.5	0.5	0.5
tAuthor	0.3225383	0.3225383	0.2	0.3225383	0.2
rootTransformation-Constant	**0.5**	**0.5**	**0.5**	**0.5**	**0.5**
ghostAuthorRank	**0.001**	**0.001**	**0.001**	**0.001**	**0.001**
No. of Relations Found	**74**	**74**	**74**	**74**	**74**
Fitness	**1.21898**	**1.21898**	**1.21898**	**1.21898**	**1.21898**

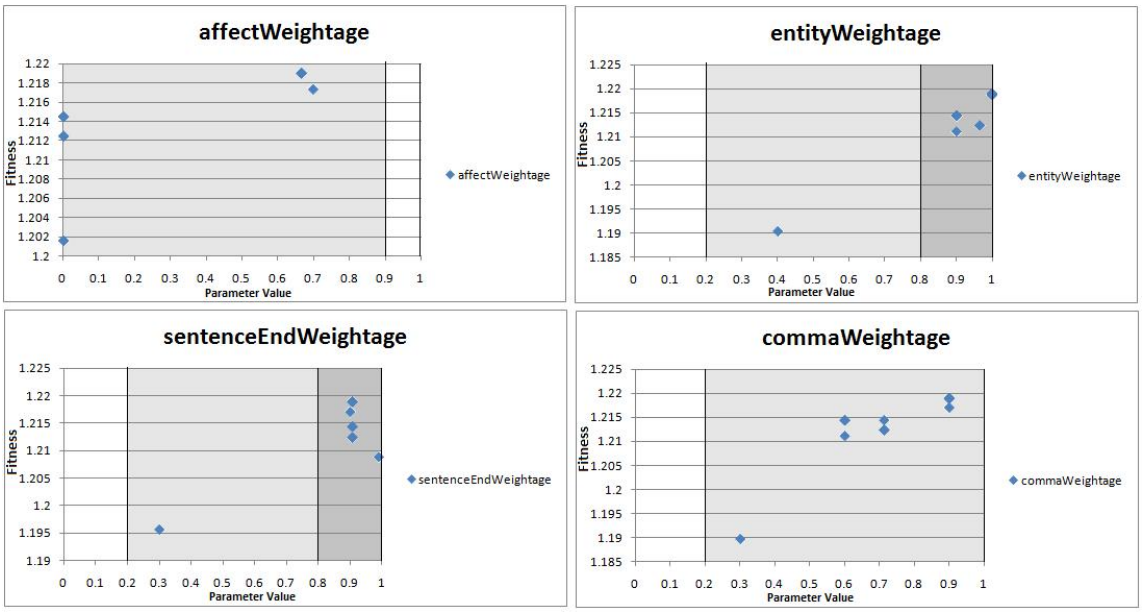


Figure 4.1: Graphical grid search result 1

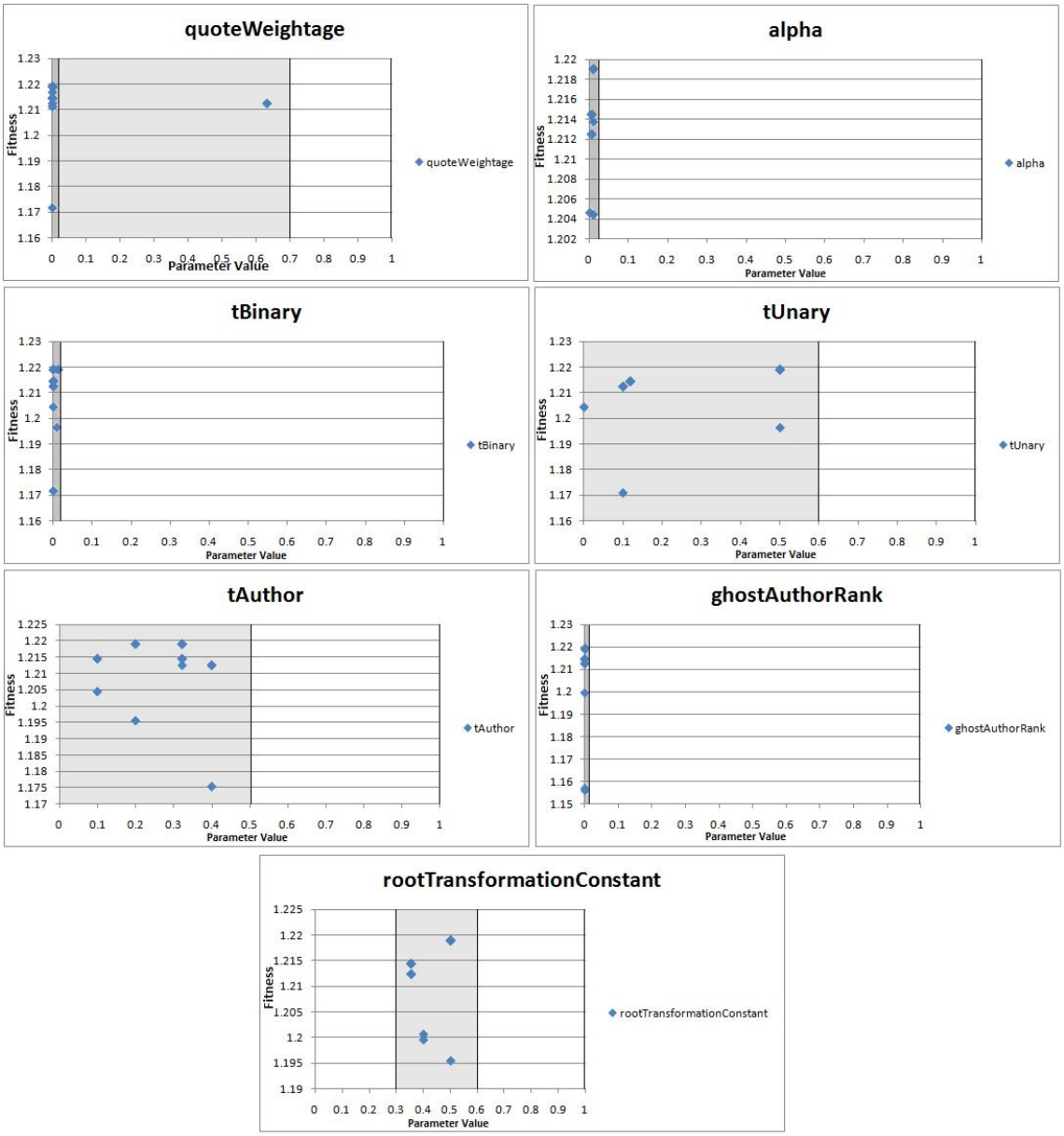


Figure 4.2: Graphical grid search result 2

5. OPTIMIZATION ALGORITHMS

This section describes different optimization algorithms investigated to solve the optimization problem. Section 5.1, Section 5.2, Section 5.3 and Section 5.4 explain random search, hill-climbing, simulated annealing and evolutionary algorithms respectively.

5.1. RANDOM SEARCH OPTIMIZER

The fitness evaluation is performed for a set of parameter lists randomly selected from the optimization search space. The higher the fitness values, the better the parameter list selected.

Algorithm 1 represents the pseudocode for the random search optimizer.

Algorithm 1 Random Search Optimizer Algorithm

```
RANDOM SEARCH OPTIMIZER (executeRandomSearchOptimization())
1   size(documentSet) is the no. of documents in the benchmark document set
2   for i = 1 to NO_OF_EVALUATIONS do
3      parameterList = generateRandomParameterList()
4      cumulativeError = evaluateAEPropAlgorithm(parameterList, documentSet)
5      fitness = -1 × cumulativeError / (size(documentSet))
6      log(fitness)
7   endfor
8
9   evaluateAEPropAlgorithm(parameterList, documentSet)
10  for each document in documentSet do
11     cumulativeRelationError = 0
12     algorithmERList = getAlgorithmERList(document)
13     benchmarkERList = getBenchmarkERList(document)
14     for each relation in algorithmERList do
15        cumulativeRelationError = cumulativeRelationError
16         + calculateEuclideanDistance(algorithmERList(relation), benchmarkERList(relation))
17     endfor
18  endfor
19  return cumulativeRelationError
```

Algorithm 2 Hill Climbing Algorithm

```
HILL CLIMBING OPTIMIZER (executeHillClimbingOptimization())
1   for i = 1 to NO_OF_EVALUATIONS do
2     newParameterList = generateRandomParameterList()
3     currentFitness = -1 × calculateFitness(newParameterList, documentSet)
4     repeat
5       oldParameterList = newParameterList
6       previousFitness = currentFitness
7       newParameterList = stochasticFirstChoice(oldParameterList, currentFitness, ALPHA)
8       currentFitness = -1 × calculateFitness(newParameterList, documentSet)
9     until currentFitness > previousFitness
10  endfor
11
12  stochasticFirstChoice(oldParameterList, previousFitness, alpha)
13  resultList = generateNextParameterList(oldParameterList, alpha)
14  currentFitness = -1 × calculateFitness(resultList, documentSet)
15  if currentFitness > previousFitness then
16    return resultList
17  else
18    return stochasticFirstChoice(oldParameterList, previousFitness, alpha))
19  endif
20  return previousFitness
21
22  generateNextParameterList(parameterList, alpha)
23  for i = 1 to NO_OF_PARAMETERS do
24    V[i] = generateRandomNumberFromBetaDistribution()
25    resultList.setParameter(i, V[i])
26  endfor
27  return resultList
28
29  calculateFitness(parameterList, documentSet)
30  cumulativeError = evaluateAEPropAlgorithm(parameterList, documentSet)
31  fitness = cumulativeError / (size(documentSet))
32  return fitness
33
```

From a sample of 50 different runs of the random search optimizer, the average fitness obtained is 1.0002 and the best fitness obtained is 1.0497. This is far less than the best solution and a desirable range of fitness.

Table 5.1: Parameter lists reached by 5 different hill-climbing starting points

Parameter Name	Parameter List1	Parameter List2	Parameter List3	Parameter List4	Parameter List5
affectWeightage	0.55723	0.23546	0.34678	0.21789	0.99178
entityWeightage	0.9087	0.39874	0.98344	0.11238	0.8345
sentenceEndWeightage	0.78654	0.18673	0.76584	0.25643	0.7256
commaWeightage	0.6744	0.37402	0.0873	0.1765	0.4673
quoteWeightage	0.63298	0.4521	0.11332	0.29811	0.8621
alpha	0.0011	0.1154	0.2231	0.9721	0.00021
tBinary	0.5329	0.4325	0.41471	0.01613	0.18431
tUnary	0.6513	0.9076	0.1165	0.07691	0.78567
tAuthor	0.31083	0.3379	0.21654	0.00322	0.1222
rootTransformation-Constant	0.7215	0.54801	0.1145	0.7892	0.6285
ghostAuthorRank	0.0001	0.12841	0.98701	0.314531	0.2891

5.2. HILL CLIMBING OPTIMIZER

The hill climbing search algorithm is shown in Algorithm 2. It is simply a loop that continually moves in the direction of increasing value - that is, uphill. It terminates when it reaches the highest peak where no neighbour has a higher value.

Hill-climbing algorithms have been modified with a number of variations. Stochas-

tic hill climbing chooses at random from among the uphill moves; the probability of selection can vary with the steepness of the uphill move. First-choice hill climbing implements stochastic hill climbing by generating successors randomly until one is generated that is better than the current state. This is a good strategy when a state has many of successors. Random-restart hill climbing conducts a series of hill-climbing searches from randomly generated initial states, stopping when a goal is found [25].

For the optimization problem under consideration, a first-choice stochastic hill climbing approach is employed. As can be seen from the pseudo-code, random searches on the search space are performed until a parameter list with a better fitness value is obtained. This is done for a given maximum number of evaluations.

The hill-climbing algorithm reaches the local maximum fast and gets stuck there. It almost never got out of the local maximum for the optimization problem of this thesis. The best fitness obtained from the algorithm was 1.2214. Although the best fitness achieved was better than the random search, it is not the best solution that could be reached.

Furthermore, an experiment was conducted to see whether hill climbing done from random starting points reach a single point on the search space. Table 5.1 shows parameter lists which were reached after hill climbing from five random starting points. As can be seen, all of them reached different search points at the end. This proves that the problem is a multi-modal optimization problem with the best solutions present at different peaks of the search space.

5.3. SIMULATED ANNEALING OPTIMIZER

5.3.1. Base Simulated Annealing Optimizer. A hill-climbing algorithm that never makes 'downhill' moves towards states with lower value is incomplete, because it can get stuck on a local maximum. In contrast, a purely random walk - that is, moving to a successor chosen uniformly at random from the set of successors - is complete, but extremely inefficient. Therefore, it seems reasonable to try to combine with a random walk in some way that yields both efficiency and completeness. Simulated annealing is such an algorithm [25].

Algorithm 3 Basic Simulated Annealing Algorithm

```
1  T = schedule[t]; schedule input determines the value of T as a function of time
2  oldParameterList = generateRandomParameterList()
3  currentFitness = calculateFitness(oldParameterList, documentSet)
4  previousFitness = currentFitness
5  while true do
6     if T = 0 then
7        return oldParameterList
8     endif
9     newParameterList = generateNextParameterList (oldParameterList, Alpha)
10    currentFitness = -1 × calculateFitness(newParameterList, documentSet)
11    deltaE = currentFitness - previousFitness
12    if deltaE > 0 then
13       oldParameterList = newParameterList
14    else
15       oldParameterList = newParameterList; only with probability e^deltaE/T
16    endif
17 endwhile
```

The innermost loop of the basic simulated annealing algorithm (Algorithm 3) is quite similar to hill climbing. Instead of picking the best move, however, it picks a random move. If the move improves the situation, it is always accepted. Otherwise the algorithm accepts the move with some probability less than 1. The probability decreases with the 'badness' of the move. The probability also decreases as the 'temperature' T goes down: 'bad' moves are more likely to be allowed at the start when temperature is high, and they become more unlikely as T decreases.

5.3.2. Adaptive Simulated Annealing Optimizer.

The basic simulated annealing algorithm is modified with better scaling and stopping criteria as discussed in [32]. The newer algorithm is based on the generalized method of Bohachevsky et al. It automatically adjusts the step sizes to reflect the local slopes and function values, and controls the random directions to point favorably toward potential improvements.

When the fitness score is relatively flat we permit a wide range of search parameters to prevail in order to stimulate a new and productive direction of search. Conversely, when great improvements in fitness are observed, the scope of parameter search is narrowed under the assumption that the algorithm is 'on the right track'.

Accordingly, we utilize a symmetric Beta distribution to model the variable step size for the modification of parameter values.

The initial parameter set used is the parameter set with the best fitness value from all the random fitness evaluations performed earlier.

Let,

α and β represent the shape parameters of the Beta distribution,

γ represent a multiplicative constant for adaptive step size selection,

pf be the previous fitness on a particular iteration,

cf be the current fitness on a particular iteration,

Then,

$$\alpha = \beta = \begin{cases} \gamma \times \left(\frac{pf}{cf}\right)^{\theta} & \forall pf \geq cf \\ \gamma \times \left(\frac{cf}{pf}\right)^{\theta} & \forall pf < cf \end{cases} \tag{9}$$

A greater difference between the current and previous fitness scores will generate a lesser variable and more peaked Beta distribution. Every parameter has a 0.5 probability of modification during each iteration.

The adaptive simulated algorithm is shown in Algorithm 4

5.4. EVOLUTIONARY ALGORITHMS

5.4.1. Background. Evolutionary Algorithms (EAs) are population based optimization algorithms inspired by the Darwinian theory of biological evolution. According to Darwin's theory of natural selection, a population of individuals within some environment that has limited resources would compete for the resources causing natural selection (survival of the fittest). The survival of an individual is determined by a fitness criteria with respect to the resources available. The individual's survival is also influenced by cross-over and mutation happening inside the genetic pool.

Similarly, for an optimization problem, a population of possible solutions of the problem is evaluated based on a fitness function. In general, an EA consists of the following steps:

Algorithm 4 Adaptive Simulated Annealing Algorithm

```
ADAPTIVE SIMULATED ANNEALING OPTIMIZER (executeASAOptimization())
1   for i = 1 to NO_OF_EVALUATIONS do
2     oldParameterList = generateRandomParameterList()
3     currentFitness = calculateFitness(oldParameterList, documentSet)
4     T = INITIAL_TEMPERATURE // initial temperature
5     previousFitness = currentFitness
6     iter = 1
7     Converge = FALSE
8     totUnder = 0
9     repeat
10      alpha = (maximum(previousFitness/currentFitness, currentFitness/previousFitness)^Theta
11      newParameterList = generateNextParameterList(oldParameterList, alpha)
12      currentFitness = calculateFitness(newParameterList, documentSet)
13      deltaE = currentFitness - previousFitness
14      randomNum = generateRandomRealNumberInRange(0, 1)
15      if deltaE < 0 OR randomNum <=e^(-deltaE/(T×previousFitness)) then
16        log(currentFitness)
17        previousFitness = currentFitness
18        oldParameterList = newParameterList
19      endif
20      if absoluteValueOf(deltaE)<= K then
21        totUnder++
22      else
23        totUnder = 0
24      endif
25      if iter > Jmin AND totUnder >= Jc then
26        Converge = TRUE
27      else
28        Converge = FALSE
29      endif
30      T = Decay × T
31      iter++
32    until iter <= Jmax AND Converge = FALSE
33  endfor
34
35  generateNextParameterList(parameterList, alpha)
36  for i = 1 to NO_OF_PARAMETERS do
37    V[i] = generateRandomNumberFromBetaDistribution()
38    if V[i] <= Qprob then
39      Brand = generateRandomNumberFromBetaDistribution()
40    endif
41    if Brand <= 0.5 then
42      resultList.setParameter(i, 2×Brand* V[i])
43    else
44      resultList.setParameter(i, V[i] + (2×Brand-1)×(1- V[i])
45    endif
46  endfor
47  return resultList
48
```

- Encoding of solutions to the problem as individuals of a population.
- Initialization of an initial population.
- Evaluation of individual quality using a fitness function.
- Selection operations - Parent selection done for selecting parents before recombination/mating. Survival selection for selecting individuals from the offspring pool for the next generation.
- Reproduction operations - using cross-over and mutation operators.

For the affect propagation algorithm optimization, the solution encoding is done as real valued vectors of the eleven different parameters. The encoding strategy and fitness function used are elaborated in Section 3.1.3.

5.4.2. Base Evolutionary Algorithm. A basic evolutionary algorithm with minimal EA features was devised. The pseudocode is shown in Algorithm 5.

The population is initialized with random real valued parameter set vectors (see Algorithm 6).

Parent selection is performed using a tournament selection of ten individuals (see Algorithm 7). I.e., ten indivuals from the current population are randomly selected and they compete with each other for getting selected as the parent. The individual with the best fitness is selected as the winner.

Recombination of selected two parents is performed using uniform cross-over (see Algorithm 8). Each parameter in the offspring is randomly selected from one of the two parent parameter lists.

Mutation of the offsprings is carried out by selecting a random parameter set from the beta distribution (see Algorithm 9). The beta distribution is chosen because the values returned lie in the range <0,1> which is the same range needed for each of the parameters.

Survivor selection is done using tournament selection of ten offsprings at a time.

The best fitness achieved from the algorithm was 1.2235 (from 10 different runs) which was far less than the best fitness we had till now from Adaptive Simulated Annealing Optimizer (which was 1.279). So there is a need to change the strategies in the evolutionary algorithm to make it perform better.

5.4.3. Evolutionary Algorithm with Fitness Sharing. The base EA was enhanced with measures to monitor and control diversity in the population. The enhanced algorithm is represented in Algorithm 10. The output results of the algorithm is discussed in Section 6.

- Diversity Metric

  A genotype diversity measure was used to measure the population diversity. This would represent the extent to which the individuals in the population are different. We have used the genotype diversity measure explained in [16]. The pseudocode for the diversity calculation is shown in Algorithm 14.

  Let,

  P represent population size,

  N represent no. of parameters,

  x_{ij} represent individual parameter value

  Centroid of ith parameter,

$$c_i = \frac{\sum_{j=1}^{j=P} x_{ij}}{P} \tag{10}$$

  Moment of inertia along the centroid,

$$I = \sum_{i=1}^{i=N} \sum_{j=1}^{j=P} (x_{ij} - c_i)^2 \tag{11}$$

- Fitness Sharing

  Fitness sharing is one of the explicit schemes to maintain population diversity. The fitnesses of individuals are adjusted prior to selection (see Algorithm 11) in order to allocate individuals to niches in proportion to the niche fitness [10].

  This scheme works by considering each possible pairing of individuals i and j within the population (including i with itself) and calculating a distance *d(i,j)* between them (see Algorithm 12). The distance measure used here is

the Euclidean distance. The fitness F of each individual i is then adjusted according to the number of individuals falling within some prespecified distance σ_{share} using a power-law distribution (see Algorithm 13):

$$F'(i) = \frac{F(i)}{\sum_j sh(d(i,j))}, \tag{12}$$

where the sharing function *sh(d)* is a function of the distance d given by

$$sh(d) = \begin{cases} 1 - \left(\frac{d}{\sigma_{share}}\right)^{\alpha} & if d \leq \sigma_{share}, \\ 0 & otherwise \end{cases} \tag{13}$$

- EA Parameter Tuning

  There were various algorithm specific parameters for which the best values had to be determined. From different runs of the algorithm, the parameter values were finalised based on how well the diversity was maintained all throughout the fitness evaluations.

  The constant value α determines the shape of the sharing function: For $\alpha = 1$ the function is linear, whereas for values greater than this the effect of similar individuals in reducing a solution's fitness falls off more rapidly with distance. From experimental observations, it was observed that using a linear function gives the best result for this optimization problem.

  The parameter σ_{share} decides both how many niches can be maintained and the granularity with which different niches can be discriminated. A default value in the range 5-10 is recommended [8]. It was experimentally determined that a value of 5 for σ_{share} gave the best results for this research.

Algorithm 5 Base Evolutionary Algorithm

```
1  population = initializePopulation()
2  log(population)
3  globalPopulation = population
4  initialBestIndividual = findBestParameterList(population)
5  log(initialBestIndividual)
6  for i = 1 to NO_OF_GENERATIONS do
7     offsprings.clear()
8     for j = 1 to CHILDREN_POPULATION_SIZE do
9        parentsForSelection = population
10       parent1 = selectIndividualFromTournamentSelection(parentsForSelection)
11       parentsForSelection.remove(parent1)
12       parent2 = selectIndividualFromTournamentSelection(parentsForSelection)
13       child = performUniformCrossover(parent1, parent2)
14       randomNum = getRandomNumber(0,1)
15       if randomNum < MUTATION_RATE  then
16          child = performBetaMutation(child)
17       endif
18       fitness = calculateFitness(child, documentSet)
19       child.setFitness(fitness)
20       log(child)
21       offsprings.add(child)
22       parentsForSelection.clear()
23    endfor
24    populationForSurvivorSelection.clear()
25    populationForSurvivorSelection.addAll(population)
26    populationForSurvivorSelection.addAll(offsprings)
27    population.clear()
28    for j = 1 to POPULATION_SIZE do
29       survivor = selectIndividualFromTournamentSelection(populationForSurvivorSelection)
30       population.add(survivor)
31       populationForSurvivorSelection.remove(survivor)
32    endfor
33    localBestIndividual = findBestParameterList(population)
34    log(localBestIndividual)
35    globalPopulation.addAll(population)
36    globalBestIndividual = findBestParameterList(globalPopulation)
37    log(globalBestIndividual)
38 endfor
```

Algorithm 6 Population Initialization

```
1  initializePopulation()
2  for i = 1 to POPULATION_SIZE do
3     parameterList = generateRandomParameterList()
4     fitness = calculateFitness(parameterList, documentSet)
5     parameterList.setFitness(fitness)
6     population.add(parameterList)
7  endfor
8  return population
```

Algorithm 7 Tournament Selection

```
1  selectIndividualFromTournamentSelection(population)
2  for i = 1 to TOURNAMENT_SIZE do
3     individual = selectRandom(population)
4     tournamentPopulation.add(individual)
5     population.remove(individual)
6  endfor
7  selectedIndividual = findBestParameterList(tournamentPopulation)
8  return selectedIndividual
```

Algorithm 8 Performing Uniform Crossover

```
1  performUniformCrossover(parent1, parent2)
2  for i = 1 to NO_OF_PARAMETERS do
3     parent = selectRandom(parent1, parent2)
4     child.setParameter(i) =parent.getparameter(i)
5  endfor
6  return child
7
```

Algorithm 9 Performing Beta Mutation

```
1  performBetaMutation(child)
2  for i = 1 to NO_OF_PARAMETERS do
3     mean = child.getParameter(i)
4     alpha = mean × [(mean(1-mean)/IDEAL_VARIANCE) - 1]
5     beta = (1 - mean) × [(mean(1-mean)/IDEAL_VARIANCE) - 1]
6     distribution = betaDistribution(alpha, beta)
7     randomNum = distribution.random()
8     child.setParameter(i, randomNum)
9  endfor
10 return child
```

Algorithm 10 Evolutionary Algorithm with Fitness Sharing

```
1  population = initializePopulation()
2  log(population)
3  globalPopulation = population
4  initialBestIndividual = findBestParameterList(population)
5  log(initialBestIndividual)
6  for i = 1 to NO_OF_GENERATIONS do
7     offsprings.clear()
8     for j = 1 to CHILDREN_POPULATION_SIZE do
9        parentsForSelection = population
10       parent1 = selectIndividualFromTournamentSelection(parentsForSelection)
11       parentsForSelection.remove(parent1)
12       parent2 = selectIndividualFromTournamentSelection(parentsForSelection)
13       child = performUniformCrossover(parent1, parent2)
14       randomNum = getRandomNumber(0,1)
15       if randomNum < MUTATION_RATE  then
16          child = performBetaMutation(child)
17       endif
18       fitness = calculateFitness(child, documentSet)
19       child.setFitness(fitness)
20       log(child)
21       offsprings.add(child)
22       parentsForSelection.clear()
23    endfor
24    populationForSurvivorSelection.clear()
25    populationForSurvivorSelection.addAll(population)
26    populationForSurvivorSelection.addAll(offsprings)
27    population.clear()
28    for j = 1 to POPULATION_SIZE do
29       survivor = selectIndividualFromTournamentSelection(populationForSurvivorSelection)
30       population.add(survivor)
31       populationForSurvivorSelection.remove(survivor)
32    endfor
33    population = shareFitness(population)
34    genotypeDiversity = calculateGenotypeDiversity(population)
35    log(genotypeDiversity)
36    localBestIndividual = findBestParameterList(population)
37    log(localBestIndividual)
38    globalPopulation.addAll(population)
39    globalBestIndividual = findBestParameterList(globalPopulation)
40    log(globalBestIndividual)
41 endfor
```

Algorithm 11 Fitness sharing strategy

```
1  shareFitness(population)
2  for i = 1 to POPULATION_SIZE do
3     distancePerParamList = 0
4     for j = 1 to POPULATION_SIZE do
5        euclideanDistance = calculateEuclideanDistance(population.get(i), population.get(j))
6        distancePerParamList += shForFitnessSharing(euclideanDistance)
7     endfor
8     newFitness = population.get(i).getFitness()/distancePerParameterList
9     population.get(i).setFitness(newFitness)
10 endfor
11 return population
```

Algorithm 12 Euclidean distance calculation

```
1  calculateEuclideanDistance(ParameterLista, ParameterListb)
2  euclideanDistance = 0.0
3  d = 0.0
4  for i = 1 to NO_OF_PARAMETERS do
5     d += (a.getParam(i) − b.getParam(i))^2
6  endfor
7  euclideanDistance = √d
8  return euclideanDistance
```

Algorithm 13 Calculating sh for fitness sharing

```
1  shForFitnessSharing(d)
2  if V[i] <= Qprob then
3     if d <= FITNESS_SHARE_SIGMA then
4        return (1-Math.pow((d/FITNESS_SHARE_SIGMA),FITNESS_SHARE_ALPHA))
5     else
6        return 0
7     endif
8  endif
```

Algorithm 14 Genotype diversity calculation

```
1   calculateGenotypeDiversity(population)
2   centroidList = calculateCentroid(population)
3   momentOfInertia = calculateMomentOfInertia(population, centroidList)
4   return momentOfInertia
5
6   calculateCentroid(population)
7   for i = 1 to NO_OF_PARAMETERS do
8      centroidList.add(0.0)
9   endfor
10  for i = 1 to NO_OF_PARAMETERS do
11     for each pList in population do
12        centroidList.set(i,centroidList.get(i)+pList.getParam(i))
13     endfor
14  endfor
15  for i = 1 to NO_OF_PARAMETERS do
16     centroidList.set(i, centroidList.get(i)/POPULATION_SIZE)
17  endfor
18  return centroidList
19
20  calculateMomentOfInertia(population, centroidList)
21  momentOfInertia = 0
22  for each pList in population do
23     for i = 1 to NO_OF_PARAMETERS do
24        momentOfInertia += (pList.getParam(i) − centroidList.get(i))^2
25     endfor
26  endfor
27  return momentOfInertia
```

6. RESULTS AND DISCUSSION

This section discusses the results obtained by applying the optimization algorithms. Section 6.1 lists the set of algorithmic parameters used. Section 6.2 gives details on the best fitnesses from different algorithms and Section 6.3 gives the best parameter set from all algorithms and corresponding example affect entity relation models.

6.1. EXPERIMENTAL SETUP

The different optimization algorithms investigated have a set of algorithm specific parameter values. These are listed in Table 6.1 to Table 6.5.

For all the experiments run, a linear aggregation of document words is performed in the base algorithm. This means that the random walk performed at the third step of the base algorithm is actually deterministic in the experiments described here. For each node in the document graph, there are only two connecting edges, one forward and one reverse.

Except for the grid search heuristics, all other approaches used a set of 50 documents from the MPQA document corpus. Due to time constraints, the whole set of 50 documents was not used for grid search heuristics. Instead, a set of 4 carefully selected documents having a median amount of entities and entity relationships was used. For purposes of experimental comparison in this section, the reduced set is assumed to be a sufficiently good approximation of the full set, but obviously this may not actually be the case and further investigation of the grid search heuristics employing the full set are needed to verify this.

Table 6.1: Random search algorithm parameters

Algorithm Parameter	**Parameter Value**
Number of Evaluations	50

Table 6.2: Hill climbing algorithm parameters

Algorithm Parameter	**Parameter Value**
Number of Evaluations	50
Alpha	15

Table 6.3: Adaptive simulated annealing algorithm parameters

Algorithm Parameter	**Parameter Value**
Initial Temperature	5
Theta (rescaling constant for adaptive step size selection)	5
Decay (temperature decay factor)	0.98
Qprob (probability of mutating a single dimension in the ParameterList)	0.5
Jmin (minimum number of iterations)	100
Jmax (maximum number of iterations)	500
Jc (sequence length for stopping criterion)	10
K (change threshold for stopping criterion)	0.001
Initialization	Random Best Individual
Termination Condition	5000 evals

6.2. BEST FITNESSES FROM DIFFERENT APPROACHES

The best fitnesses achieved from the different approaches are represented in Table 6.6. The fitness scores are calculated from a sample of 10-20 different runs of the individual algorithms. More samples were not taken due to time constraints. The best global best fitness of an algorithm is the best fitness achieved from all the runs of the algorithm. The average global best fitness is the average of the best fitnesses obtained from all the runs.

Table 6.6 shows that the highest best global best fitness is obtained by the fitness shared EA followed by the adaptive simulated annealing algorithm. Among all the algorithms used, the highest average global best fitness was achieved by the

evolutionary algorithm. Therefore, statistically there is a higher probability that the EA can achieve a better solution. Also, the standard deviation with regards to average best fitness remains lowest for the EA. Figure 6.1 represents the fitness ranges achieved by the different approaches as a box plot. From the statistical analysis and the box plot, it is evident that the EA with fitness sharing covered a smaller span of fitness values compared to other algorithms. Apparently, among all the algorithms experimented, the EA is the best approach for the optimization problem.

6.2.1. Fitness Improvement on the Best Run. Figure 6.2 represents the fitness improvement achieved by the initial best run of the EA with fitness sharing (without parameter tuning). The graph shows the result in comparison to that of the adaptive simulated annealing. As can be seen, the EA population diversity decreased drastically from around 1200 fitness evaluations. Parameter set 1 in Table 6.7 was used for this case.

After proper parameter tuning, the diversity of the population was maintained. With reference to [8], the value of sigma lies in the range 5-10. Keeping all other parameters constant, the best fitness is obtained when the sigma value was 5. The best achieved fitness was using the linear function. A population size greater than 100 and an offspring size greater than 30 achieved fitnesses well below the normal range of best fitnesses. Table 6.7 shows the effects of further tuning of the parameters on the fitness. For each set of EA parameters used, the EA was executed 12 times. The best fitness is achieved by the set of parameters shown in bold. The same set of parameters had the best average fitness over all its runs. The different parameters used do not show much standard deviation from the average best fitness.

Figure 6.3 represents the fitness improvement achieved by the best run after diversity maintenance. As in the previous graphs, there is an initial peak to the EA diversity which happens because the EA is continuously increasing its diversity till a reasonable range is achieved. Random initialization typically results in a number of genotypes being clustered together which is suboptimal for diversity and the initial rapid increase in diversity is due to the diversity measure breaking up these clusters. After the peak is reached, the population diversity is preserved at the higher level all throughout the remaining fitness evaluations. With proper diversity maintenance, its seen that the EA maximum fitness is higher compared to simulated annealing and it

is attained earlier with a lower number of fitness evaluations. The EA achieves better fitness around 1400 fitness evaluations and continues to be better than simulated annealing for all rest of the fitness evaluations. It is interesting to note that although the simulated annealing fitness evaluations began with a parameter list having better fitness, in the long run it could not beat fitness of the population based EA search.

A statistical analysis of the best fitnesses obtained from the two optimization algorithms is also performed. The best fitnesses obtained from 17 runs each of the evolutionary and simulated annealing algorithms are analysed. At each fitness evaluation, the average best fitness is evaluated and plotted as shown in Figure 6.4. The EA shows a higher average best fitness consistently after 900 fitness evaluations.

The average best fitnesses of all the runs substantiates that there is a higher probability for the EA to achieve a better fitness at each instance of fitness evaluation.

6.3. FINAL BEST PARAMETER SET

Table 6.8 displays the final list of parameters obtained.

It is interesting to note that all the grid search heuristics observations turned out to be true. The parameters alpha, tBinary and ghostAuthorRank are very small as expected. A Wilcoxon rank sum test [35] was conducted between the best fitnesses obtained from simulated annealing algorithm and evolutionary algorithm executions (see Appendix A). The results prove that the mean value of best fitnesses of both algorithms are dissimilar, so the EA which has the higher mean is statistically significantly better.

6.4. EXAMPLE DOCUMENTS AND AFFECT-ENTITY NETWORKS

Figure 6.5 - Figure 6.7 represent affective relationships from the optimized affect propagation algorithm along with the benchmarks. The algorithm uses the set of parameters in Table 6.8. These examples demonstrate the following:

- Almost all benchmarked affective relationships were discovered.
- Many affective relationships discovered were not documented in the benchmark diagram, and hence they are incorrect relationships. Consider the example shown in Figure 6.5. The affect entity diagram from the algorithm displays an

affective relationship of fear, hope and pity from the entity ‘Chechens’ to the entity ‘Putin’. However, this relationship is absent in the benchmark diagram.

- Symmetric affective relationships were discovered. For example, in Figure 6.7 the algorithm discovers symmetrical relationship of admiration and fear between the entities ‘Author’ and ‘Bush’.

- Over-diffusive spreading of affect through the documents. This means that the same affect category appears between more entities in the document than is actually the case. Figure 6.5 displays the spreading of the affect category hope. Figure 6.6 and Figure 6.7 show spreading of the affect category admiration.

Table 6.4: Base evolutionary algorithm parameters

Algorithm Parameter	**Parameter Value**
Population Size	50 or 100
Tournament Size	10
Offspring Population Size	20
Mutation Rate	0.2
Ideal Variance	0.003
Initialization	Random
Termination Condition	5000 evals

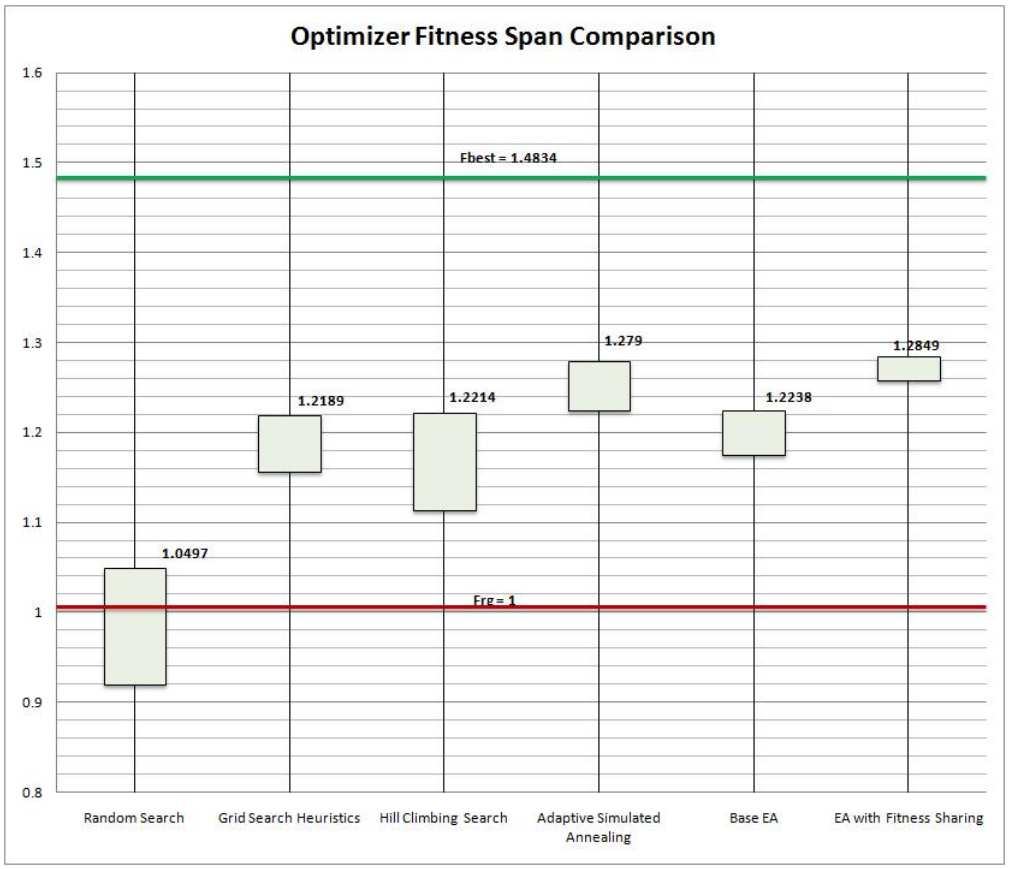


Figure 6.1: Span of best fitnesses from different approaches

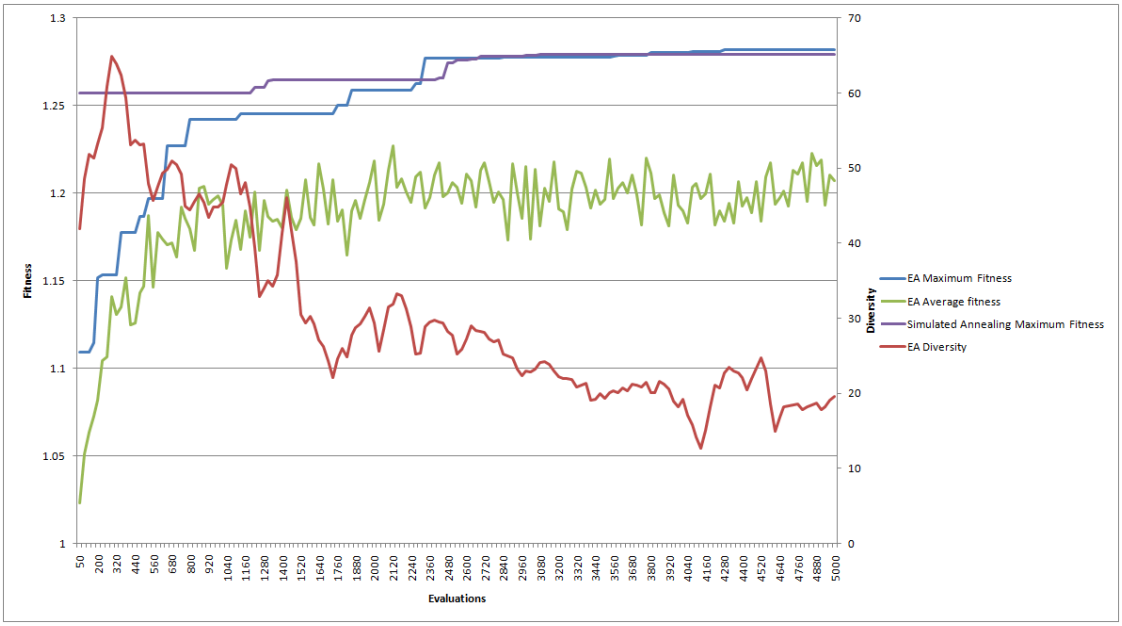


Figure 6.2: Best run of EA with fitness sharing before parameter tuning

Table 6.5: Fitness shared evolutionary algorithm parameters

Algorithm Parameter	**Parameter Value**
Population Size	50 or 100
Tournament Size	3 or 5 or 10
Offspring Population Size	30
Mutation Rate	0.2
Ideal Variance	0.003
Sigma for Fitness Share	4 or 5 or 6
Alpha for Fitness Share	0.5 or 1 or 2
Initialization	Random
Termination Condition	5000 evals

Table 6.6: Best fitnesses from different approaches

Approach	**Best Global Best Fitness**	**Average Global Best Fitness (Standard Deviation)**
Manual Parameters	1.0513	1.0513 (0)
Random Search	1.0497	0.9992 (0.0428)
Best Grid Search Heuristics	1.219	1.219 (0)
Hill Climbing Search	1.2214	1.1892 (0.0322)
Adaptive Simulated Annealing	1.279	1.2615 (0.0164)
Base Evolutionary Algorithm	1.2235	1.1912 (0.0142)
Evolutionary Algorithm with Fitness Sharing	1.2849	1.275 (0.0083)

Table 6.7: Parameter tuning on EA with fitness sharing

Parameter Set	**Population Size**	**Tournament Size**	**Offspring Population Size**	**Mutation Rate**	**Sigma for Fitness Sharing**	**Alpha for Fitness Sharing**	**Best Global Best Fitness Achieved**	**Average Global Best Fitness (Standard Deviation)**
1	50	5	30	0.2	5	1	1.281	1.2731 (0.0071)
2	100	5	30	0.2	5	1	1.2712	1.2675 (0.0078)
3	50	3	30	0.2	5	1	1.2847	1.2747 (0.0113)
4	**100**	**3**	**30**	**0.2**	**5**	**1**	**1.2849**	**1.275 (0.0083)**
5	50	3	30	0.2	4	1	1.2805	1.2741 (0.0073)
6	50	3	30	0.5	5	1	1.2767	1.2721 (0.013)
7	50	3	20	0.2	5	1	1.2732	1.2702 (0.0071)
8	50	3	30	0.2	5	0.5	1.2769	1.2711 (0.0101)
9	50	3	30	0.2	4	0.5	1.271	1.2657 (0.0083)

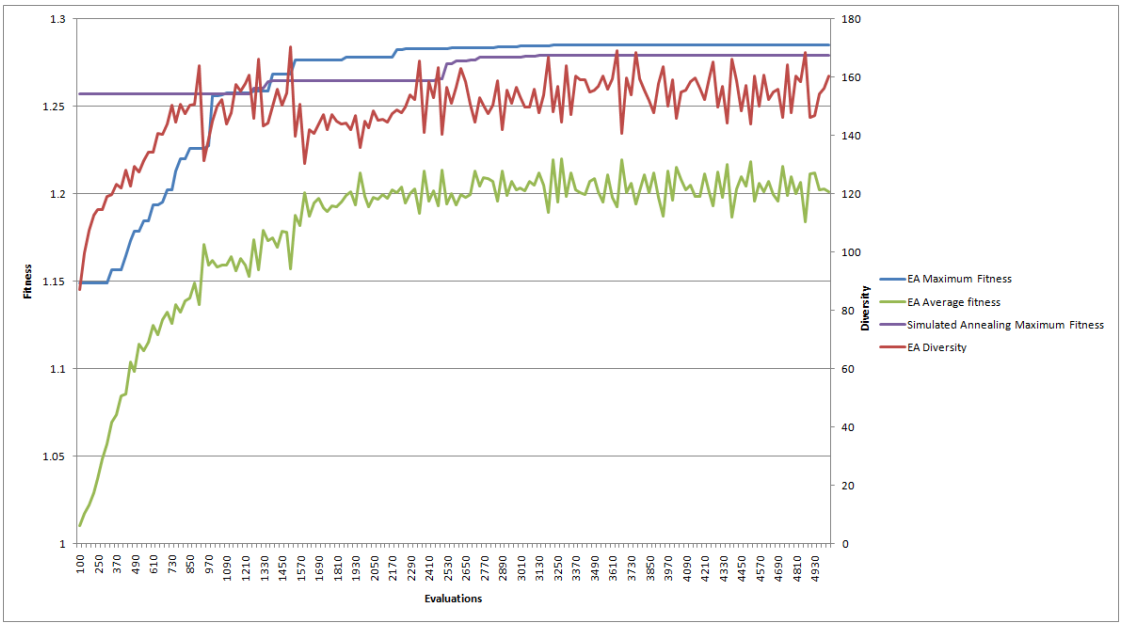


Figure 6.3: Best run of EA with fitness sharing after parameter tuning

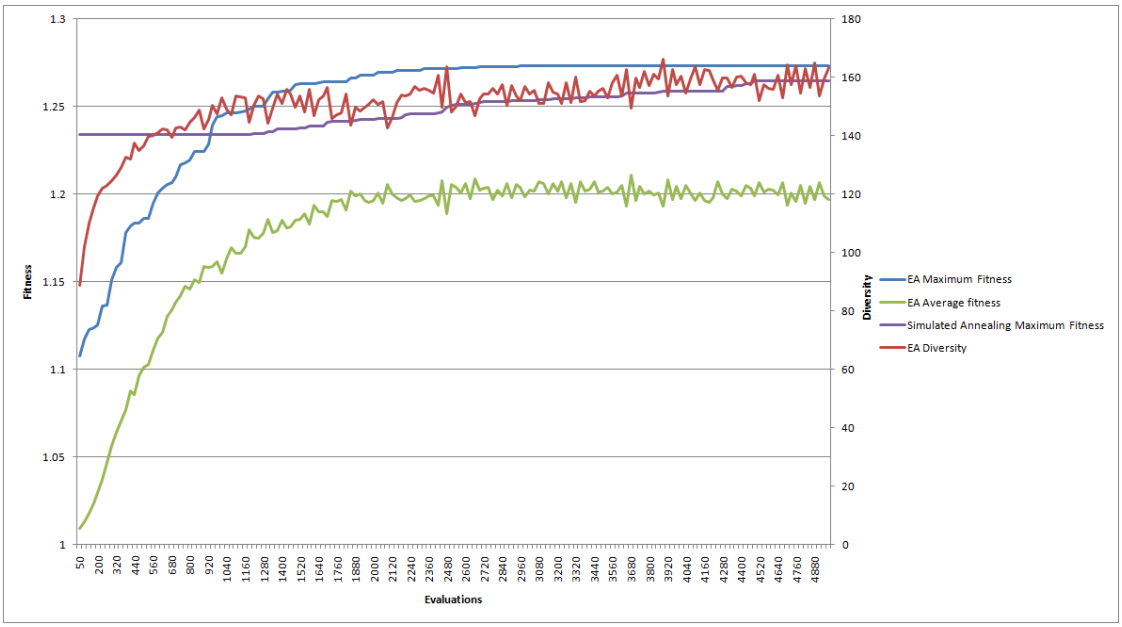
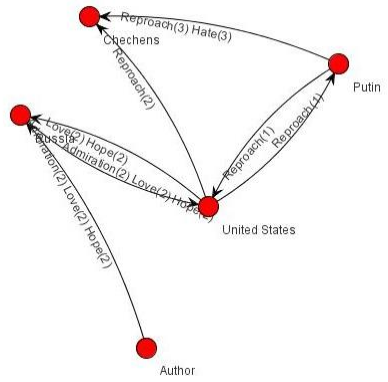


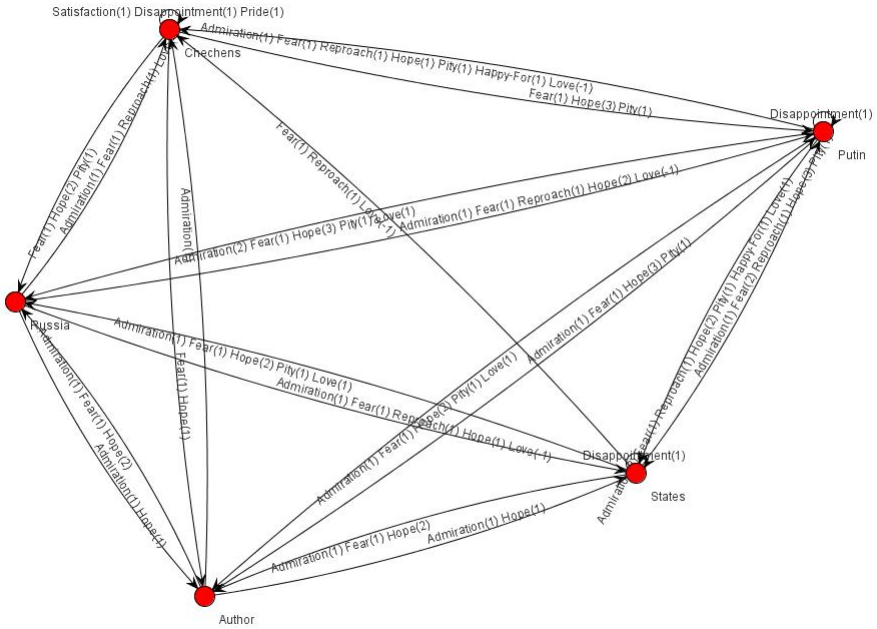
Figure 6.4: Comparison of average best fitness improvement achieved by simulated annealing and evolutionary algorithms

Table 6.8: Final best parameter list generated after EA parameter tuning

Parameter	**Parameter Value**
affectWeightage	0.999999882
entityWeightage	0.992598855
sentenceEndWeightage	0.997672876
commaWeightage	0.983690331
quoteWeightage	0.166119085
alpha	3.02E-05
tBinary	0.004666844
tUnary	2.86E-92
tAuthor	0.999997076
rootTransformationConstant	0.195870871
ghostAuthorRank	3.47E-08

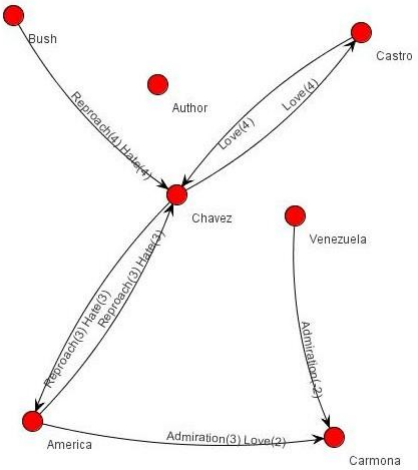


(a) Benchmark affect-entity diagram
for document 23.18.15-25073

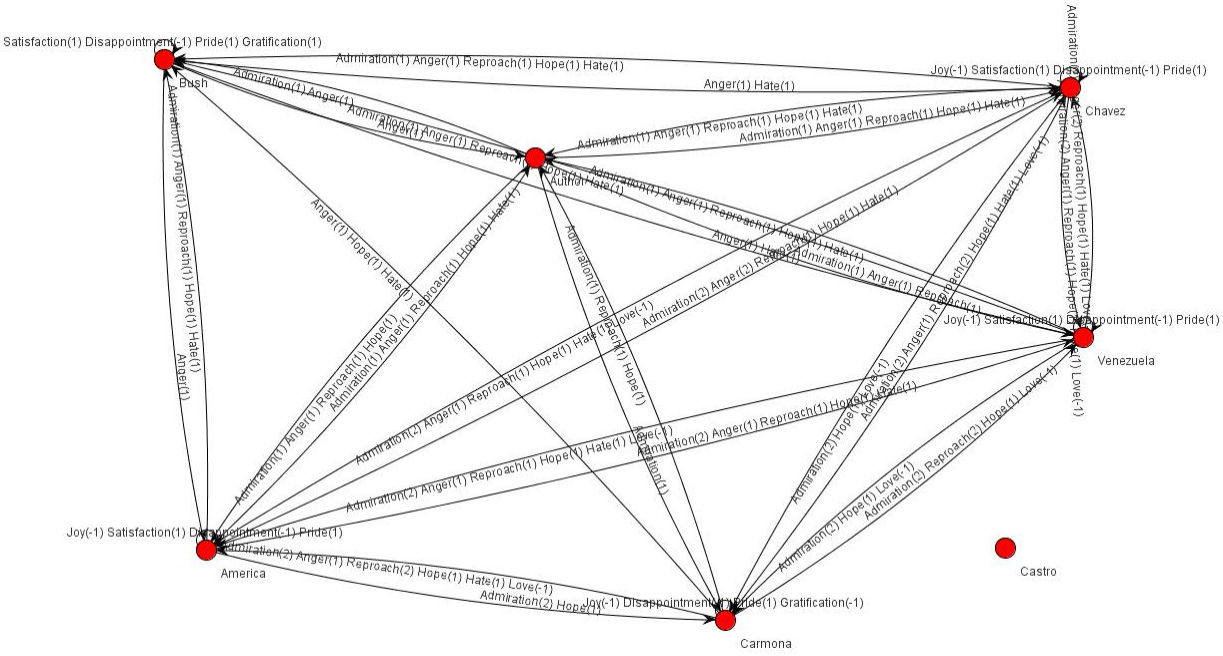


(b) Optimized affect propagation diagram for document 23.18.15-25073

Figure 6.5: Affect-entity diagram example 1

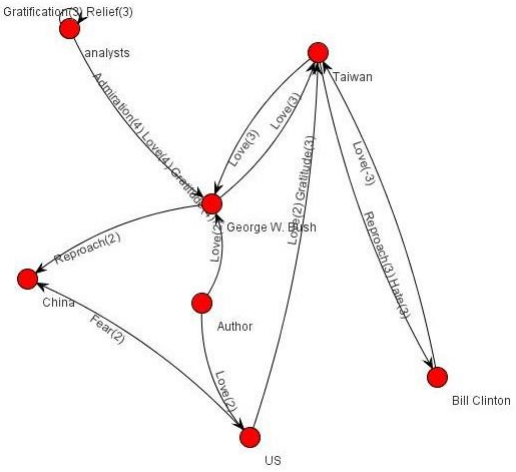


(a) Benchmark affect-entity diagram for document 21.50.57-15245

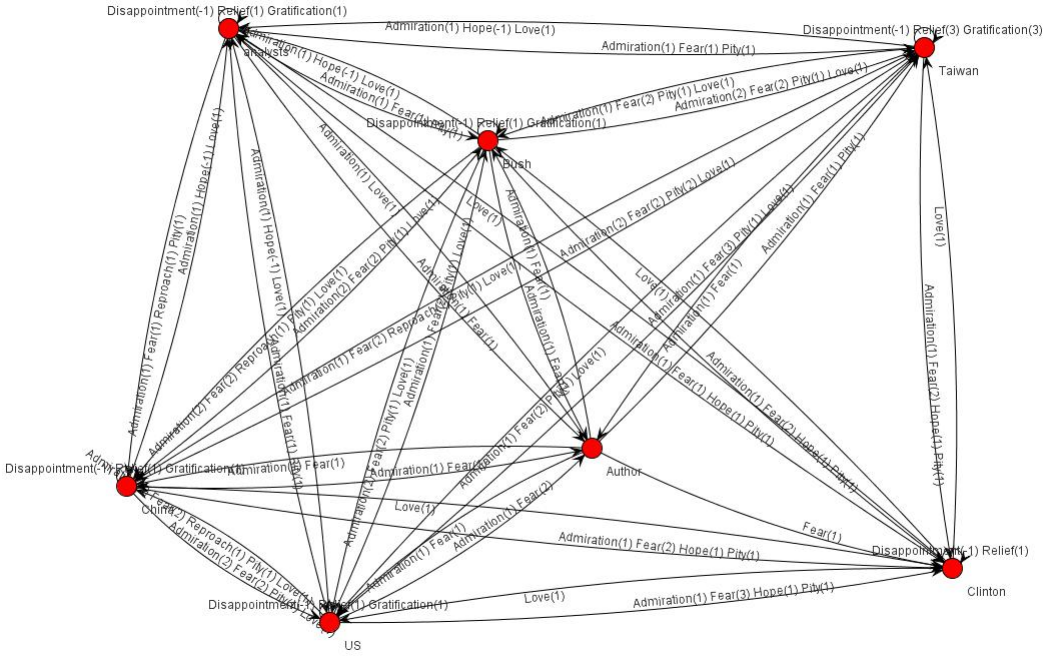


(b) Optimized affect propagation diagram for document 21.50.57-15245

Figure 6.6: Affect-entity diagram example 2



(a) Benchmark affect-entity diagram for document 21.07.24-24231



(b) Optimized affect propagation diagram for document 21.07.24-24231

Figure 6.7: Affect-entity diagram example 3

7. CONCLUSION AND FUTURE WORK

7.1. CONCLUSION

The focus of this thesis was to optimize the affect propagation algorithm. Various optimization algorithms have been attempted for the purpose. Among all the optimization techniques applied, the technique using evolutionary algorithm was the most successful one, giving a higher best fitness statistically. The evolutionary algorithm used an explicit diversity maintenance mechanism in which the fitnesses of individuals were adjusted prior to selection in an attempt to allocate individuals to niches in proportion to the niche fitness.

The four research questions posed in Section 2.4 are answered as follows:

1. What is the best possible parameter set for the Affect Propagation algorithm?

   After extensive investigation of different optimization algorithms, the best parameter set found is shown in Table 6.8.

2. What fitness criterion determines the best parameter set solution vector?

   An enhanced fitness measure as explained in Section 3.5 is used to find the best solution. The fitness function is the ratio of expected error of a random guesser to the actual error generated by the algorithm.

3. Which optimization algorithm produces the best results?

   A statistical analysis (see Section 6.2) of results from different optimization algorithms indicates that the fitness shared EA is the best approach tested.

4. Is the result of the affect propagation algorithm employing the optimal parameter set of sufficient quality to be useful?

   Although sophisticated optimization strategies were applied, the quality of the final affect entity relation models remains sub par compared to the benchmark ones. It is not certain if there exists a better solution for the problem, but it appears more likely that the problem lies with the affect propagation algorithm.

Significant changes are needed before it can approach the accuracy of human judgement of affective meaning in text.

7.2. FUTURE WORK

The following two paragraphs are adopted with minor modifications from an ORNL technical report [27] which was co-authored by the author of this thesis.

Many tasks and challenges remain before affective computing can begin to claim to be a mature research area. The performance of the affect extraction algorithms could be improved significantly by a more thorough accounting of context effects in documents. Different sentence constructions such as active versus passive voice ought to exert different influences on the mechanism of affect propagation. Word sense disambiguation can also be quite important.

An important future direction would be to extend the analysis from single to multiple documents. The evolution of affective relationships can be tracked by performing change detection on the structure of the affect-entity relation network over time-ordered multiple texts from a single source. Entities can also be clustered on the basis of affective similarity. A dynamic analysis of the affect-induced entity space properly displayed in an evolution panel would provide indications of formation and/or dissolution of clusters or groups with shared affect.

Instead of the linear approach, the second step of the base algorithm can utilize various node aggregation schemes based on affect words, entities or noun phrases for generating the document word graph.

The current strategy for all the optimization algorithms uses only a training set of input documents. In the future, a tagged set of test documents from the MPQA corpus needs to be used to analyse the effectiveness of the final optimized set of parameters on documents not yet trained during optimization. This would help to generalize the parameter set solution for any arbitrary document.

All the future changes to the affect propagation algorithm would also affect the optimization strategy by which the best set of parameters are selected. Any change to the base algorithm would require validation of the parameters selected. Hence, there can be changes to the optimization technique used.

For EA population diversity maintenance, currently we are using the fitness shared approach. Fitness sharing allocates individuals to peaks only in proportion to their fitness. So there is a higher likelihood for a best solution near to the lower fitness peak not to be uncovered. In the future the crowding technique can be investigated wherein the population is distributed evenly amongst the peaks. Furthermore, a combination of fitness sharing and crowding techniques can be tested. These changes would also require parameter tuning to find the best fit EA parameters.

Similar to the prime number predicting finite state machine experiment described in Section 3.4, the optimization problem considered in this thesis faced the problem that the optimal solutions were all sparse, so evolving parameter sets which caused solutions with no entity relationships to be produced appeared to be of reasonably good quality and therefore had an evolutionary advantage (the so-called "zero vector problem"), until a special fitness function was created which specifically addressed this through the use of the ratio of the expected error of a random guesser to the actual error generated by the base algorithm. This technique can be generalized for use in other problem domains which have in common that there are easy-to-evolve solutions with relatively high fitness.

APPENDIX A

WILCOXON RANK SUM TEST

Similar to a t test, the Wilcoxon Rank Sum Test is used to determine whether there is a significant difference between the means of two distributions. Specifically, we are trying to determine whether the two samples came from the same distributions or not. The Wilcoxon Rank Sum Test is used in place of the t test when either of the samples do not appear to be normally distributed and when the sample size is too small.

The Wilcoxon Rank Sum Test is a hypothesis test and has the following Null Hypothesis and Alternate Hypothesis:

H0 - Null Hypothesis states that the two samples came from population distributions having similar means.

H1 - Alternate Hypothesis states that the two samples come from population distributions with dissimilar means.

Table A.1 represents the list of best fitnesses obtained for both adaptive simulated annealing (ASA) and evolutionary algorithm (EA) executions.

- Step 1 : Count the samples in each column

  n1 = 16

  n2 = 17

- Step 2 : Rank all samples according to sample size. This is shown in Table A.2.

- Step 3 : Combine all samples and rank them. This is shown in Table A.3.

- Step 4 : Re-sort the rows according to algorithm. This is shown in Table A.4.

- Step 5 : Calculate the sum of the ranks (N1 and N2) for each algorithm. This is shown in Table A.5 and Table A.6.

- Step 6 : Determine whether R = N1 or N2. R is taken from the sample with the smaller size. n1 = 16 and n2 = 17 so sample group 1 is smaller. Therefore, R = N1 = 191.5

- Step 7 : Calculate Z Score and Z Critical to determine whether the sample groups come from different populations.

When n1 and n2 are both > 10, the normal distribution can approximate the distribution of R.

Z Score Calculation

Calculate μr and σr

$$\mu r = [n1 * (n1 + n2 + 1)]/2 \tag{A.1}$$

$$\sigma r = SQRT[(n1 * n2 * (n1 + n2 + 1))/12] \tag{A.2}$$

$$\mu r = [16 * (16 + 17 + 1)]/2 = 272 \tag{A.3}$$

$$\sigma r = SQRT[(16 * 17 * (16 + 17 + 1))/12] = 27.76 \tag{A.4}$$

Calculating the Z Score,

$$Z = (R - \mu r)/\sigma r = -2.89 \tag{A.5}$$

$$|Z| = 2.89 \tag{A.6}$$

Calculating Z Critical,

$$Zcrit = NORMSINV(1 - a/2) \tag{A.7}$$

For example, for a = 0.05 (for 95 percent certainty) and two-tailed test

$$Zcrit = NORMSINV(1 - 0.05/2) = NORMSINV(0.975) = 1.96 \tag{A.8}$$

Since $|Z|$ is greater than Zcrit, we reject the null hypothesis. Therefore, the means of the two best fitness populations are different and there does not exist a relationship between the best fitnesses obtained from simulated annealing and the evolutionary algorithm.

Table A.1: Best fitnesses from ASA and EA

Adaptive Simulated Annealing	**Evolutionary Algorithm**
1.2763	1.2765
1.279	1.2572
1.278	1.2849
1.2524	1.2731
1.2246	1.2672
1.2716	1.2721
1.2654	1.2614
1.256	1.2753
1.2713	1.2801
1.2492	1.2812
1.2771	1.2631
1.2678	1.2824
1.2556	1.2771
1.2763	1.281
1.2358	1.2846
1.248	1.2794
	1.2792

Table A.2: Best fitnesses ranked with respect to sample size

Fitness	**Algorithm**
1.2765	EA
1.2572	EA
1.2849	EA
1.2731	EA
1.2672	EA
1.2721	EA
1.2614	EA
1.2753	EA
1.2801	EA
1.2812	EA
1.2631	EA
1.2824	EA
1.2771	EA
1.281	EA
1.2846	EA
1.2794	EA
1.2792	EA
1.2763	ASA
1.279	ASA
1.278	ASA
1.2524	ASA
1.2246	ASA
1.2716	ASA
1.2654	ASA
1.256	ASA
1.2713	ASA
1.2492	ASA
1.2771	ASA
1.2678	ASA
1.2556	ASA
1.2763	ASA
1.2358	ASA
1.248	ASA

Table A.3: Ranked best fitnesses after combining

Rank	**Fitness**	**Algorithm**
1	1.2246	ASA
2	1.2492	ASA
3	1.2524	ASA
4	1.256	ASA
5	1.2572	EA
6	1.2614	EA
7	1.2631	EA
8	1.2654	ASA
9	1.2672	EA
10	1.2678	ASA
11	1.2713	ASA
12	1.2716	ASA
13	1.2721	EA
14	1.2731	EA
15	1.2753	EA
16	1.2763	ASA
17	1.2765	EA
18.5	1.2771	EA
18.5	1.2771	ASA
19	1.278	ASA
20	1.279	ASA
21	1.2801	EA
22	1.2812	EA
23	1.2824	EA
24	1.2849	EA

Table A.4: Best fitnesses re-sorted after combining

Rank	**Fitness**	**Algorithm**
1	1.2246	ASA
2	1.2358	ASA
3	1.248	ASA
4	1.2492	ASA
5	1.2524	ASA
6	1.2556	ASA
7	1.256	ASA
8	1.2572	EA
9	1.2614	EA
10	1.2631	EA
11	1.2654	ASA
12	1.2672	EA
13	1.2678	ASA
14	1.2713	ASA
15	1.2716	ASA
16	1.2721	EA
17	1.2731	EA
18	1.2753	EA
19.5	1.2763	ASA
19.5	1.2763	ASA
21	1.2765	EA
22.5	1.2771	EA
22.5	1.2771	ASA
24	1.278	ASA
25	1.279	ASA
26	1.2792	EA
27	1.2794	EA
28	1.2801	EA
29	1.281	EA
30	1.2812	EA
31	1.2824	EA
32	1.2846	EA
33	1.2849	EA

Table A.5: Best fitnesses summed for simulated annealing

Rank	**Fitness**	**Algorithm**
1	1.2246	ASA
2	1.2358	ASA
3	1.248	ASA
4	1.2492	ASA
5	1.2524	ASA
6	1.2556	ASA
7	1.256	ASA
11	1.2654	ASA
13	1.2678	ASA
14	1.2713	ASA
15	1.2716	ASA
19.5	1.2763	ASA
19.5	1.2763	ASA
22.5	1.2771	ASA
24	1.278	ASA
25	1.279	ASA
Sum, N1 = 191.5		

Table A.6: Best fitnesses summed for evolutionary algorithm

Rank	Fitness	Algorithm
8	1.2572	EA
9	1.2614	EA
10	1.2631	EA
12	1.2672	EA
16	1.2721	EA
17	1.2731	EA
18	1.2753	EA
21	1.2765	EA
22.5	1.2771	EA
26	1.2792	EA
27	1.2794	EA
28	1.2801	EA
29	1.281	EA
30	1.2812	EA
31	1.2824	EA
32	1.2846	EA
33	1.2849	EA
Sum, N2 = 369.5		

APPENDIX B

EXAMPLE TEXT DOCUMENT

This appendix shows the listing of the complete text document used to explain the affect propagation algorithm in Section 2.2.

```
Leaders of the United States probably withheld their breath Monday
    evening while watching on television what Russian President Vladimir
     Putin was about to say to his people. After Putin's statement they
    rubbed their palms at length. It amounted to much more than what one
     could have expected. The military great power that towers over
    Afghanistan has committed itself to overthrow the ruling Taliban
    system, and has pledged to support the US ``anti-terrorism military
    operation" now being prepared. Putin has exceeded the ``positive
    neutrality policy" his country (the Soviet Union at the time) has
    pursued during the Gulf War, and whose continued validity the
    military leaders also regarded as desirable. The situation in which
    Putin has agreed to open former Soviet military airfields to
    American armed forces in three (former Soviet) Central Asian
    countries could be regarded as dramatic. These countries are
    Tajikistan, Uzbekistan and Turkmenistan. Although these are
    sovereign states, and the final decision would have been in their
    hands, people familiar with the actual balance of power indicate
    that these states would have done nothing, had Moscow wanted a
    different thing to happen.


One cannot underestimate the significance of the fact that Russia has
    opened its airspace. True, Putin only permitted (Americans) to
    deliver via Russian airspace aid to the ``theater of the anti-
    terrorism military operations," (i.e. not soldiers and weapons).
    Nevertheless, this amounts to such high degree of Russian-American
    cooperation in a wartime situation, the two sides have never
    attained during the past 10 years. And Putin has established the
    prospect of taking further steps in this regard.


Accordingly, Russia has joined the anti-terrorist coalition of the
    United States virtually without reservation. Putin has written the
    name of Moscow on the registration page of the American side. He
    wanted to become an ally, and that's what he has become. But making
```

this complex decision was not easy for him. Reformers told Putin all along that he should take advantage of ‘‘the rare opportunity for (Russia) to become integrated with the civilized world.” But main stream Russian military policy has been to keep a distance from America, the country that has been historically responsible for ‘‘educating” the Taliban that had caused the demise of Soviet occupation forces in Afghanistan. In the end, a certain consideration that differed from the above two proved to be decisive. It had to do with Chechnya. Putin has no greater desire than to present to the West the Chechen independence movement as a chapter of ‘‘international terror.” He did not succeed with that so far. He was criticized more than recognized for his policy. At this time however, he taught a lesson to the Americans. After the Americans had promised everything good and nice, Putin remarked almost as an aside that the events in Chechnia ‘‘could be interpreted only in the context of the struggle against international terrorism.” In reality, Putin asked for a free hand regarding Chechnia, some understanding, or even that the Americans look the other way. Putin did not receive what he sought in a quick response from Washington, namely, that the State Department would continue to encourage political dialogue between Moscow and the Chechen insurgents. Nevertheless, one could not fail taking note of the fact that in the same response the United States called upon the Chechens to ‘‘unconditionally and without delay to terminate every contact with international terrorist groups,” and that they accept Putin’s offer to make peace. If we view this warning against the background that Putin’s government has always seen Usama Bin Ladin’s hands behind ‘‘Chechen terrorism,” we were witnessing a late recognition of Moscow’s views.

The Russian President has understood this American recognition. In Berlin, where he was the center of stormy celebration, he vindicated for himself the deeper truth that may be seen behind terror. He argued that had the West not continued to keep alive during the past several years the Cold War stereotype of a threat from the East, but would have concentrated instead on terrorism, the common enemy, the twin towers of New York may not have collapsed. ‘‘Accordingly, at this time it is your turn to think,”

```
Putin said. Feeling like a true ally, he was able to ask for a
place for Russia almost inconspicuously – in NATO. This would have
been a joke only yesterday. Today, at the threshold of expansion,
it is no longer a joke.
```

Listing 1: Complete listing of the example text document

APPENDIX C

EXAMPLE AFFECT CATEGORY SEEDLIST

This appendix shows a partial seedlist of affect words belonging to the 'Hope Hopeful' affect category.

Hope, Hopeful

1. (n) anticipation#1, (v) anticipate#5
2. (n) expectancy#1, (v) expect#1, (n) expectation#2
3. (j) optimistic#1#2, (n) optimism#1#2
4. (v) trust#4#5
5. (v) desire#2
6. (n) wish#1#2, (v) wish#1#2#3#4
7. (n) promise#2, (j) promising#1#2
8. (n) want#4
9. (n) velleity#1
10. (v) look#8
11. (v) await#1
12. (v) look forward#1
13. (v) look to#1
14. (j) positive#1
15. (j) bullish#1
16. (j) upbeat#1
17. (j) rose-colored#1
18. (j) rosy#1
19. (j) sanguine#1
20. (j) affirmative#2
21. (n) faith#2
22. (n) confidence#1#2#3, (j) confident#1#2
23. (n) assurance#1, (n) assuredness#1
24. (n) sureness#1, (j) sure#1#3, (r) surely#1, (r) for sure#1
25. (j) certain#2#4, (r) certainly#1, (r) for certain#1
26. (n) reassurance#1, (v) reassure#1#2, (j) reassuring#1
27. (j) encouraged#1, (n) encouragement#2#3, (j) encouraging#1, (v) encourage#1#2#3
28. (j) bucked up#1, (v) buck up#1
29. (n) conviction#1
30. (n) assumption#7, (v) assume#1
31. (n) presumption#1, (v) presume#4
32. (n) aspiration#1#2, (v) aspire#1
33. (v) draw a bead on#2

```
34. (v) shoot for#1
35. (n) ambition#1, (j) ambitious#1
36. (n) longing#1
37. (v) long for
38. (n) craving#1
39. (n) yearning#1, (v) yearn#1
40. (v) hunger for
41. (v) dream of
42. (v) heart set on
43. (v) bent upon
44. (v) fancy#1#2
45. (v) envision#1
46. (v) picture#1
47. (n) hankering#1, (v) hanker#1
48. (n) yen#1
49. (v) count on#1, (v) counting on
50. (n) chance#1#5
51. (j) lucky#3
52. (n) opportunity#1
```

Listing 2: Seedlist words of the affect category 'Hope Hopeful'

BIBLIOGRAPHY

[1] Abbasi, A., and Chen, H. Affect intensity analysis of dark web forums. In *ISI '07: Proceedings of the IEEE International Conference on Intelligence and Security Informatics* (Washington, DC, USA, 2007), IEEE Computer Society, pp. 282–288.

[2] Abbasi, A., Chen, H., Thoms, S., and Fu, T. Affect Analysis of Web Forums and Blogs Using Correlation Ensembles. *IEEE Trans. on Knowl. and Data Eng. 20*, 9 (2008), 1168–1180.

[3] Alm, C. O., Roth, D., and Sproat, R. Emotions from text: machine learning for text-based emotion prediction. In *HLT '05: Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing* (Morristown, NJ, USA, 2005), Association for Computational Linguistics, pp. 579–586.

[4] Aman, S., and Szpakowicz, S. Using Roget's Thesaurus for Fine-grained Emotion Recognition. In *Proceedings of the Third International Joint Conference on Natural Language Processing* (2008), Association for Computational Linguistics.

[5] Ann., W. T. *Fine-grained subjectivity and sentiment analysis: recognizing the intensity, polarity, and attitudes of private states.* PhD thesis, Pittsburgh, PA, USA, 2008.

[6] Brin, S., and Page, L. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems 30*, 1-7 (1998), 107 – 117. Proceedings of the Seventh International World Wide Web Conference.

[7] Damasio, A. R. *Descartes' error : emotion, reason, and the human brain.* G.P. Putnam, New York, 1994.

[8] Deb, K., and Kalyanmoy, D. *Multi-Objective Optimization Using Evolutionary Algorithms.* John Wiley & Sons, Inc., New York, NY, USA, 2001.

[9] Eiben, A. E., and Smith, J. E. *Introduction to Evolutionary Computing (Natural Computing Series).* Springer, 2007.

[10] Goldberg, D. E., and Richardson, J. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application* (Hillsdale, NJ, USA, 1987), L. Erlbaum Associates Inc., pp. 41–49.

[11] He S., D. B., and Gildea, D. Building and tagging with an affect lexicon. Technical Report 833, 2004.

[12] Hugo, L., Henry, L., and Ted, S. A model of textual affect sensing using real-world knowledge. In *IUI '03: Proceedings of the 8th international conference on Intelligent user interfaces* (New York, NY, USA, 2003), ACM, pp. 125–132.

[13] Lodge, M., and Taber, C. S. The Automaticity of Affect for Political Leaders, Groups, and Issues: An Experimental Test of the Hot Cognition Hypothesis. *Political Psychology 26(3)* (2005), 455–482.

[14] Lucas, P. Certainty-factor-like Structures in Bayesian Belief Networks. *Knowledge-Based Systems 14* (1999), 327–335.

[15] Miller, G. A. WordNet: A Lexical Database for English. *Communications of the ACM 38* (1995), 39–41.

[16] Morrison, R. W., and Jong, K. A. D. Measurement of population diversity. In *Selected Papers from the 5th European Conference on Artificial Evolution* (London, UK, 2002), Springer-Verlag, pp. 31–41.

[17] Mostafa Al Masum, S., Prendinger, H., and Ishizuka, M. Emotion Sensitive News Agent: An Approach Towards User Centric Emotion Sensing from the News. In *WI '07: Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence* (Washington, DC, USA, 2007), IEEE Computer Society, pp. 614–620.

[18] O'Madadhain, J., Fisher, D., White, S., and Boey, Y. The JUNG (Java Universal Network/Graph) Framework. Tech. rep., UCI-ICS, October 2003.

[19] O'Rorke, P., and Ortony, A. Explaining emotions. *Cognitive Science 18* (1994), 283–323.

[20] Ortony A., G. L. C., and Collins, A. *The cognitive structure of emotions.* Cambridge University Press, New York, 1988.

[21] Pang, B., and Lee, L. Opinion Mining and Sentiment Analysis. *Foundations and Trends in Information Retrieval 2*, 1-2 (Jan. 2008), 1–135.

[22] Pennebaker, J. W., and Francis, M. E. *Linguistic Inquiry and Word Count*, 1 ed. Lawrence Erlbaum, August 1999.

[23] Picard, R. W. *Affective Computing.* MIT Press, 1997.

[24] Ratinov, L., and Roth, D. Design challenges and misconceptions in named entity recognition. In *CoNLL '09: Proceedings of the Thirteenth Conference on Computational Natural Language Learning* (Morristown, NJ, USA, 2009), Association for Computational Linguistics, pp. 147–155.

[25] Russell, S. J., and Norvig, P. *Artificial Intelligence: A Modern Approach.* Pearson Education, 2003.

[26] Sang, E. F. T. K., and Meulder, F. D. Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition. In *Proceedings of CoNLL-2003* (2003), pp. 142–147.

[27] Schryver, J. C., Begoli, E., Jose, A. C., and Griffin, C. Inferring Group Processes from Computer-Mediated Affective Text Analysis. Tech. Rep. ORNL/TM-2010/277, Oak Ridge National Laboratory, Oak Ridge, TN, 2011.

[28] Simon, H. A. *Motivational and emotional controls of cognition (1967). In Models of Thought, pages 29-38.* Yale University Press, New Haven, 1979.

[29] Szarvas, G. Hedge Classification in Biomedical Texts with a Weakly Supervised Selection of Keywords. In *Association for Computational Linguistics 08: Human Language Technologies* (2008).

[30] Valitutti, R. WordNet-Affect: an Affective Extension of WordNet. In *Proceedings of the 4th International Conference on Language Resources and Evaluation* (2004), pp. 1083–1086.

[31] Valitutti, R., and Stock, O. Developing Affective Lexical Resources. *Psychology Journal* (2004), 61–83.

[32] Wang, P. P., and Chen, D.-S. Continuous optimization by a variant of simulated annealing. *Comput. Optim. Appl. 6*, 1 (1996), 59–71.

[33] Wiebe, J., and Cardie, C. Annotating expressions of opinions and emotions in language. Language Resources and Evaluation. In *Language Resources and Evaluation (formerly Computers and the Humanities* (2005), pp. 165–210.

[34] Wilcock, G. *Introduction to Linguistic Annotation and Text Analytics*, 1st ed. Morgan & Claypool Publishers, 2009.

[35] Wilcoxon, F. Individual Comparisons by Ranking Methods. *Biometrics Bulletin 1*, 6 (1945), 80–83.

VITA

Ajith Cherukad Jose was born on July 12, 1984 in Kasaragod, Kerala, India. He completed his higher secondary education from Kendriya Vidyalaya No.2, Kasaragod in May 2001 and was accepted as an undergraduate student at the LBS College of Engineering (under Kannur University, Kerala, India) in the fall of that year. He received a BTech in Information Technology in May 2005. Afterwards, he worked as a software engineer at Accenture, Bangalore, India till July 2008. He was enrolled in the Computer Science Graduate Program at the Missouri University of Science and Technology in fall of 2008. He received his Master's degree in Computer Science in May 2011 after completing a research internship at the Oak Ridge National Laboratory.