

Enhancing an Evolutionary Algorithm with a Solution Archive to Reconstruct Cross Cut Shredded Text Documents

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Computational Intelligence

eingereicht von

Benjamin Biesinger

Matrikelnummer 0927842

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Günther Raidl
Mitwirkung: Univ.Ass. Dipl.-Ing. Christian Schauer
Univ.Ass. Dipl.-Ing. Dr.techn. Bin Hu

Wien, 07.05.2012

(Unterschrift Verfasser)

(Unterschrift Betreuung)

Erklärung zur Verfassung der Arbeit

Benjamin Biesinger
Stättermayergasse 8/21-22, 1150 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum)

(Unterschrift Verfasser)

Danksagung

An dieser Stelle möchte ich mich bei einigen Personen bedanken, die einen Beitrag zum Abschluss dieser Arbeit geleistet haben.

Einerseits möchte ich mich bei Prof. Günther Raidl bedanken, der es mir ermöglichte, diese Diplomarbeit über das spannende Thema der Rekonstruktion von Dokumenten zu schreiben. Andererseits bedanke ich mich hier auch für die exzellente Betreuung Christian Schauers, der mir immer zur Seite gestanden ist, und mich mit Rat und Tat bei allen möglichen Fragen und Unklarheiten unterstützt hat. Ein weiterer Dank geht an Bin Hu, der mit seinem großen Fachwissen über Lösungsarchive mir immer weiterhelfen konnte.

Außerdem möchte ich meinem Bruder, Christian Biesinger, für das Korrekturlesen und für einige sprachliche Verbesserungsvorschläge danken.

Zu guter Letzt danke ich meinen Eltern, die mich auf meinem Bildungs- und Lebensweg zu jeder Zeit unterstützt und mir damit diesen Abschluss ermöglicht haben.

Abstract

In this thesis a method for improving existing metaheuristics for the Reconstruction of Cross-Cut Shredded Text Documents (RCCSTD) problem is presented. For this purpose a memetic algorithm is enhanced by a solution archive, which is implemented in two different ways. Finally, the results of using the solution archive with different configurations of the memetic algorithm are compared to each other.

Cross-cut shredded text documents are documents that are cut in rectangular pieces using a shredding device. The aim is to fit the pieces in such a way next to each other so that the original document is reconstructed. Since this problem is NP-complete several heuristic approaches exist. Some of the best results are delivered by a memetic algorithm (MA), which is an extension of an evolutionary algorithm (EA), i.e., a population based metaheuristic. One of the main problems of this kind of algorithms is the loss of diversity in later generations because a lot of solutions are equal to each other.

To circumvent this problem, already generated solutions can be stored and looked up in a solution archive so that only new solutions are accepted by the EA. The insert and the search method for this datastructure have to be as efficient as possible because all solutions generated by the EA are inserted and looked up in the archive. Another requirement of the solution archive is to generate a new solution efficiently if a duplicate was found. A trie-based datastructure meets all the requirements since insertion and search run in time $O(h)$ where h is the height of the trie, which is bounded by the size of the input.

First an appropriate solution representation is developed—an array of shreds, which are represented by their integer IDs, containing the right and the bottom neighbor of each shred. With this representation the maximum solution size is drastically reduced compared to the currently used representation which stores the absolute positions of the shreds.

Two different strategies for generating new, yet unvisited, solutions are presented. In the first method a random permutation point is chosen. From this point on the decision which shred is chosen is entirely based on a list of available shreds, which is stored in each trie node. This list contains all shreds that can possibly be inserted at this level, which reveals also the difficulty of this approach—not all shreds can be chosen on every level and sometimes there is even only one shred left to choose. The second method is also based on a random permutation point. On that point the shred that has been inserted in the duplicate solution is swapped with an available shred. In this case the list of available shreds can be computed more easily.

In the end the archive is tested on several instances with different cutting patterns, thus different sizes. It was tested if the solution archive helps the memetic algorithm to find a better solution in the same amount of time. The results showed that in most cases the memetic algorithm in combination with the solution archive performed only as good as the memetic algorithm alone. This is also because of the vast memory consumption of the solution archive, which made testing very difficult.

Kurzfassung

In dieser Arbeit wird eine Methode vorgestellt, die existierende Metaheuristiken für das Reconstruction of Cross-Cut Shredded Text Documents (RCCSTD) Problem verbessert. Um dieses Ziel zu erreichen wurde ein memetischer Algorithmus durch ein Lösungsarchiv erweitert, welches auf zwei unterschiedliche Arten implementiert wurde. Zuletzt werden die Resultate verglichen, die durch das Verwenden des Lösungsarchiv mit verschiedenen Konfigurationen des memetischen Algorithmus entstanden sind.

Cross-Cut zerkleinerte Textdokumente sind Dokumente, die von einem Papierschredder in rechteckige Teile zerschnitten wurden. Das Ziel ist, diese Teile so zusammzusetzen, damit das Originaldokument wieder rekonstruiert wird. Da dieses Problem NP-vollständig ist, existieren diverse heuristische Lösungsansätze. Einige der besten Ergebnisse liefert ein memetischer Algorithmus (MA), der eine Erweiterung eines evolutionären Algorithmus (EA) darstellt, d.h. eine populationsbasierte Metaheuristik. Eines der größten Probleme solcher Algorithmen ist der Verlust von Diversität in späteren Generationen, da viele gleiche Lösungen generiert werden.

Um dieses Problem zu umgehen, können schon besuchte Lösungen in einem Lösungsarchiv gespeichert und nachgeschlagen werden, sodass nur neue Lösungen vom EA akzeptiert werden. Die Einfüge- und Suchmethode für die benötigte Datenstruktur muss so effizient wie möglich sein, da alle vom EA generierten Lösungen in dem Archiv gespeichert und nachgeschlagen werden. Eine weitere Anforderung ist, dass, wenn eine Duplikatlösung gefunden wurde, eine neue Lösung effizient generiert wird. Eine Trie-basierte Datenstruktur erfüllt alle Anforderungen, da die Einfüge- und Suchmethode in $O(h)$ läuft, wobei h die Höhe des Tries ist, die wiederum durch die Größe des Inputs beschränkt ist.

Zuerst wurde eine geeignete Lösungsrepräsentation entwickelt – die Integer-IDs der Schnipsel in einem Array, das den rechten und den unteren Nachbar von jedem Schnipsel enthält. Mit dieser Repräsentation wurde die maximale Größe einer Lösung im Vergleich zu der bisherigen drastisch reduziert, die die absoluten Positionen der Schnipsel speicherte.

Es wurden zwei verschiedene Strategien entwickelt, um neue, noch unbesuchte Lösungen zu generieren. In der ersten Methode wurde ein zufälliger Permutationspunkt gewählt. Von diesem Punkt aus wurde die Entscheidung, welches Schnipsel als nächstes gewählt wird, ausschließlich auf Basis einer Liste von verfügbaren Schnipseln, die in jedem Trie-Knoten gespeichert wird, getroffen. Diese Liste enthält alle Schnipsel, die auf dieser Ebene eingefügt werden können. Das verdeutlicht auch die Schwierigkeit dieser Methode – nicht alle Schnipsel können auf jeder Ebene eingefügt werden und manchmal kann sogar nur ein Schnipsel auf der Ebene gewählt werden. Die zweite Methode basiert auch auf einem zufällig gewählten Permutationspunkt. Auf diesem Punkt wird der Schnipsel, der in der Duplikatlösung auf diesem Level eingefügt wurde, mit einem verfügbaren Schnipsel getauscht. In diesem Fall kann die Liste der verfügbaren Schnipsel leichter berechnet werden.

Schlussendlich wurde das Archiv auf diversen Instanzen mit verschiedenen Schnittmustern (daher auch unterschiedlichen Größen) getestet. Es wurde getestet, ob das Lösungsarchiv mit gleichem Zeitaufwand dem memetischen Algorithmus hilft, eine bessere Lösung zu finden. Die Ergebnisse zeigten auf, dass in den meisten Fällen der memetische Algorithmus in Kombination mit dem Lösungsarchiv nur genauso gut wie der memetische Algorithmus alleine ist. Das kommt unter anderem daher, dass das Lösungsarchiv einen riesigen Speicherbedarf hat, was das Testen deutlich erschwerte.

Contents

1	Introduction	1
2	Problem definition	4
2.1	Complexity	6
3	Literature Survey	8
3.1	Document Reconstruction	8
3.2	Solution Archives	9
4	Heuristic Solution Techniques	11
4.1	Genetic Algorithms	11
4.1.1	Memetic Algorithms	12
4.2	Ant Colony Optimization	12
4.3	Local Search	13
4.3.1	Variable Neighborhood Descent	14
4.3.2	Variable Neighborhood Search	14
4.4	Solution Archives	16
5	Solution Archive	20
5.1	Solution Representation	20
5.2	Trie Design	22
5.2.1	Termination Conditions	23
5.3	Shred-based Permutation	24
5.3.1	Calculation of the available Shreds	24
5.3.2	Altering Neighbor Array	25
5.3.3	Invalid Swaps	26
5.3.4	Duplicate generated	27
5.4	Trie-based Permutation	27
5.4.1	Calculation of the available Shreds	27
5.4.2	Advantages	29
5.4.3	Problems	29

5.5	Memetic Algorithm using a Solution Archive	30
6	Implementation	31
6.1	Existing Framework	31
6.1.1	Construction Heuristics	32
6.1.2	Memetic Algorithm	32
6.1.3	Ant Colony Optimization	35
6.2	Implementation Issues for the Solution Archive	36
6.2.1	Memory Consumption	36
6.2.2	Trie nodes	36
6.2.3	Calculation of Shred Blocks	38
7	Tests	39
8	Conclusions	54
8.1	Future Work	55
A	Instances	56
B	Generations	65
	Bibliography	67

CHAPTER 1

Introduction

Document Shredding has a long history. The first document shredder was invented by the US citizen Abbot Augustus Low, who applied for a patent in the year 1909. According to [16] he named his device 'Waste-Paper Receptacle'. From then on many others follow his idea of shredding documents and two different kinds of shredding techniques evolved as standard:

- **Strip shredding**
The height of the shreds of strip shredded documents is equal to the height of the original documents and all shreds are rectangular and have equal width.
- **Cross-cut shredding**
All shreds of cross-cut shredded documents are rectangular and have the same size but the height of the shreds is smaller than the height of the original document, see Figure 1.1 for a cross-cut shredding device and a pile of cross-cut shreds.

Another method to destroy documents is to tear them apart. This is usually done manually—in contrast to the mechanical shredding process—and therefore the amount of shreds per document is often smaller. In this scenario, in contrast to the others, the shape and the edges of each shred is different, which can be exploited in a reconstruction procedure.

Although nowadays most information is available online and does not necessarily have to be printed out, it can often be useful to have a document in paper form—therefore document shredding can be a very useful, or even necessary, method for obfuscating



(a) A pile of shreds



(b) A typical cross-cut shredder, taken from [35]

Figure 1.1: A shredding device and its output

data. The main aim of document shredding is to make printed sensitive information such as passwords, signatures, confidential data, etc. unreadable to non-authorized people.

Being able to reconstruct shredded documents is highly useful in criminal investigation and can be used by forensic document examiners to improve their efficiency. Remnants of destroyed documents are even found in war zones, which is why DARPA¹ recently initiated a challenge to solve five different puzzles using document reconstruction techniques, see [4]. Furthermore, reconstructing shredded documents can be of historic interest like the project aiming at the reconstruction of torn STASI² documents [12].

Since all shreds have the same size and shape and due to the fact that there is no edge information available, the reconstruction of shredded documents can be formulated as a combinatorial optimization problem. Thus, a cost function that is based on the information printed on the shreds has to be defined, which is described in the next section. Another property of the shredded document is that blank shreds can be safely ignored because there is no information on them. Many different heuristic solving techniques have been used to solve the reconstruction problem and in this thesis one of them, namely a genetic algorithm, is improved by a complete solution archive, see 4.4.

In Chapter 2 the problem of reconstructing Cross-Cut shredded documents is defined and the complexity of this problem is presented. In Chapter 3 recent research about both reconstructing of documents and using solution archives together with heuristic meth-

¹Defensive Advanced Research Projects Agency

²Ministry for State Security, also known as Stasi—The official state security service of the DDR (East Germany)

ods is summarized. This chapter is followed by a description of several meta-heuristics illustrated with pseudo-code. In Chapter 5 a solution archive for the Reconstruction of Cross-Cut Shredded Text Documents problem is developed and in Chapter 6 implementation issues are discussed. Finally in Chapter 7 the results are presented and compared to other solving methods. In the last chapter a conclusion is drawn and it is shown what could be done in future work.

Problem definition

Suppose that the output of a shredding device is a set of shreds $S = \{s_0, \dots, s_{n-1}\}$. A shred is a fragment of the document that is not totally blank. Moreover, all shreds have the same width and height. Let the virtual shred s_n be a blank piece of paper of the same size as the other shreds. Here we assume that the orientation of each shred is known and the documents are printed only on one side.

As this thesis only deals with the Reconstruction of Cross-Cut Shredded Text Documents (RCCSTD) problem, only this problem is defined here (based on the definitions in [21] and [29]) and references are given to the definitions of the other two similar reconstruction problems. (i.e., strip shredding and manually torn documents)

A candidate solution to the RCCSTD problem consists of an injective mapping $\Pi = S \rightarrow \mathbb{D}^2$ where each shred is mapped to a position (x, y) in the Euclidean space, where $x, y \in \mathbb{D} = \{0, \dots, n-1\}$. The remaining positions are filled with the virtual shred.

Since the RCCSTD problem is an optimization problem we have to define a cost function, whose result should be minimized. First we define the following auxiliary function $sp = \mathbb{D} \cup \{-1, n\} \rightarrow \{0, \dots, n\}$, which returns the index i of s_i on position (x, y) :

$$sp(x, y) = \begin{cases} i & \text{if there is a shred } s_i \in S \text{ on position } (x, y) \\ n & \text{else} \end{cases} \quad (2.1)$$

Let the position of the shred s be (x, y) . Then we can define the functions $n_t(s_i), n_b(s_i)$,

$n_l(s_i)$ and $n_r(s_i)$, which return the indices of the neighbors of the shred s_i as follows:

	top neighbor $n_t(s_i) = sp(x, y - 1)$		
left neighbor $n_l(s_i) = sp(x - 1, y)$	s_i	right neighbor $n_r(s_i) = sp(x + 1, y)$	(2.2)
	$n_b(s_i) = sp(x, y + 1)$ bottom neighbor		

Finally we can define the cost function as follows:

$$c(\Pi) = \sum_{y=-1}^{n-1} \sum_{x=-1}^{n-1} c_r(sp(x, y), n_r(sp(x, y))) + c_b(sp(x, y), n_b(sp(x, y))) \quad (2.3)$$

The functions c_r and c_b indicate how well two shreds fit together, where $c_r(s_1, s_2)$ is the cost when shred s_2 is placed right of s_1 and $c_b(s_3, s_4)$ is the cost of shred s_4 placed below s_3 . These two functions are critical to the overall performance of any algorithm because they measure the error induced by placing shreds next to each other. The algorithm described in [30] which is based on the algorithm used for the RSSTD problem described in [22] gives one of the best practical results. Hence, it is used in this work.

This cost function is briefly described in the following paragraph. The error estimation function (EEF), i.e. cost function, is based on the edge information of the shreds while the inner part of the shreds is completely ignored. Since the shredded documents have to be scanned in order to reconstruct them automatically, it is assumed that the pictures of the shreds that the scanner generate all have the same resolution, i.e., the number of the pixels along the edges is the same for all shreds. Thus, a pixel-wise comparison along the edges is possible. As shown in [29] a greyscale color space is suitable for Cross-Cut shredded documents. To avoid side-effects, not only directly opposite pixels of each shred are compared to each other but two pixels above and below are also taken into account, see Figure 2.1 for an illustration.

This leads to a formula that computes a weighted average of the five pixels and compares it with the weighted average of the corresponding shred which is then summed up along the x-axis (or y-axis, respectively) of the shreds. The result of this formula is the error induced by placing the two shreds next to each other.

The formal definition of the formula c_r is the following (taken from [29]):

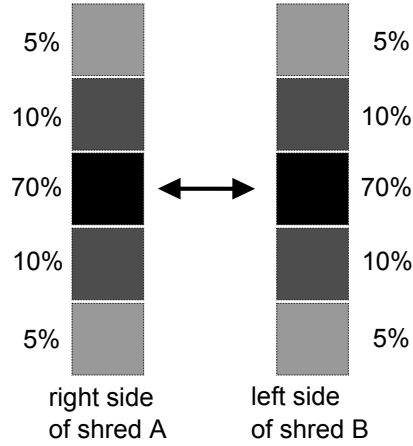


Figure 2.1: Weighted average of two opposite pixels including their neighbor pixels

$$\begin{aligned}
c_r(i, j) &= \sum_{y=3}^{h-2} e_h(i, j, y) \\
e_h(i, j, y) &= \begin{cases} 1 & \text{if } e'_h(i, j, y) \geq \tau \\ 0 & \text{else} \end{cases} \\
e'_h(i, j, y) &= |0.7 \cdot (v_r(i, y) - v_l(j, y)) \\
&\quad + 0.1 \cdot (v_r(i, y + 1) - v_l(j, y + 1)) \\
&\quad + 0.1 \cdot (v_r(i, y - 1) - v_l(j, y - 1)) \\
&\quad + 0.05 \cdot (v_r(i, y + 2) - v_l(j, y + 2)) \\
&\quad + 0.05 \cdot (v_r(i, y - 2) - v_l(j, y - 2))| \quad (2.4)
\end{aligned}$$

where h is the height of each shred (in pixels) and τ is a threshold value, which is determined by preliminary tests. The functions $v_r(i, y)$ and $v_l(i, x)$ give the greyscale value of pixel y (respectively x) of shred i where y is a pixel of the right (left) edge. These tests showed that the best results are achieved when setting $\tau = 25$. The function c_b is defined analogously.

The other two problems, the Reconstruction of Strip Shredded Text Documents (RSSTD) and the Reconstruction of Manually Torn Paper Documents (RMTDP), are defined in [21].

2.1 Complexity

All three problems are NP-Complete. Those kind of problems are very difficult to solve, i.e., assuming that $P \neq NP$ there is no algorithm that solves every instance of this problem in polynomial time. Thus, approximative algorithms have to be developed that

give good solutions in a reasonable amount of time. Assuming that there is an algorithm that solves an NP-Complete problem in polynomial time would lead to the conclusion that every NP-Complete problem can be solved in polynomial time because of the fact that every NP-Complete problem can be reduced to any other NP-Complete problem in polynomial time. Hence, this would be a proof that $P = NP$.

In [21] a proof for the NP-completeness of the RMTPD problem is given, which basic idea is that the Subset Sum problem is reduced to the RMTPD problem. In [21] a polynomial-time reduction from the decision variant of the (symmetric) travelling salesman problem to the decision variant of the RSSTD problem is shown, which, together with the reduction the other way round in [21], proves the NP-completeness of the RSSTD problem. Since the RCCSTD problem is a generalization of the RSSTD problem, it is at least as hard as the RSSTD problem and it is obviously in NP, since one could easily think of a guess and check algorithm.

Literature Survey

In this chapter an overview about current literature and research results is given. In the first part of this chapter the three types of document reconstruction are discussed, while the second part deals with solution archives. Since latter is a not yet deeply explored topic there is not very much literature available. For the methodology of solution archives see section 4.4.

3.1 Document Reconstruction

Manually torn documents can be reconstructed by exploiting edge information and often feature extracting methods are used. See [10, 11, 28] for some work in this area. In [20] feature extraction is also used and then the document is reconstructed by an LCS¹ algorithm based on the use of dynamic programming. Rane and Bhirud used mosaicing techniques for reconstructing torn documents in [19]. They extracted the text on the documents and tried to reconstruct the documents using this text information.

Most of the techniques used for reconstructing manually torn documents cannot be used for solving the RSSTD or the RCCSTD problem because, as described in Chapter 1, there is no edge information available, since the documents are cut using a shredding device.

A straight-forward method for solving the reconstruction of strip shredded documents is based on the reformulation to a TSP² as mentioned in Section 2.1. Prandtstetter discussed this approach in [22] using the well-known TSP heuristic *chained Lin-Kernighan*

¹Longest Common Subsequence

²Travelling Salesman Problem

from [1]. Additionally, a VNS³ implementation and a hybrid approach making use of human interaction are also presented in [22], which outperformed previous approaches like Ukovich et al in [33], who used MPEG-7 standard descriptors for content description. Lin and Fan-Chiang used image-based techniques for reconstructing strip shredded documents in [15]. In the first step they defined a measure of similarity between any pair of shreds using pattern recognition techniques. Then they used a shortest path algorithm on a constructed graph to reconstruct the shreds. The first genetic algorithm for solving the RSSTD problem was introduced by Skeoch in her dissertation [31]. Moreover, she also discussed different methods for fitness evaluation and she concluded that the pixel comparison method is the most effective technique for measuring the similarity between a pair of shreds. This method is also the base for the error estimation function described in Chapter 2.

This thesis is an extension to the memetic algorithm presented in [30] to solve the RCCSTD problem, which will be described in more detail in Section 6.1.2. In [23] Prandtstetter introduced a VNS and an ACO⁴ for the RCCSTD problem. A total different approach to solving this problem was made by Sleit et al. in [32]. They proposed a clustering approach, which is usually done as a preprocessing step for assigning shreds to a document if the shreds came from more than one document. In that work the clustering algorithm also generates the solution using a merge function, which can merge two clusters if there is no collision on the positions of the shreds.

3.2 Solution Archives

It is not obvious that eliminating duplicate solutions improves the performance of a genetic algorithm. Duplicate removal prevents elitism, whose advantages could outweigh the disadvantages of the loss of diversity in the population. However, Mauldin disproved this thought in [17], where he demonstrated that maintaining the diversity in each population significantly improves the performance of genetic search. Based on this result, Ronald argued in [26] that duplicate removal is not at odds with the basic mechanisms of genetic algorithms. In the same paper he introduced hash tagging for the solutions to prevent duplicates in the current population. In [36] Yuen and Chow introduced a non-revisiting GA⁵ containing a complete solution archive implemented as a binary tree. They discovered that the pruning of subtrees, where all solutions have already been visited, is isomorphic to a parameter-less self adaption mutation operator.

Raidl and Hu introduced a trie-based complete solution archive for genetic algorithms

³Variable Neighborhood Search - see section 4.3.2

⁴Ant Colony Optimization - see section 4.2

⁵Genetic Algorithm

in [25]. Together with Šramko, who wrote his master thesis (see [34]) about trie-based solution archives, they compared the impact on using a complete solution archive with the results of algorithms of famous NP-complete problems, i.e. Royal Road function, NK landscape problem, and the MAX-SAT problem. It turned out that in most cases, the quality of the solutions increased when using the archive. In [27] Ruthmair and Raidl used a trie-based solution archive in combination with a memetic algorithm for the Rooted Delay-Constrained Minimum Spanning Tree Problem, which is a more complicated problem than those mentioned above. They compared the results of the memetic algorithm with and without the solution archive with solution hashing, whose overhead is negligible. The result was that although the solution archive could improve the quality of the solution if the amount of revisits is very high, in most cases the time overhead was too big so that the results were actually worse than using no duplicate detection or hashing. Another application of a trie-based solution archive on a more complex problem—the Generalized Minimum Spanning Tree problem—can be found in [9]. In their work Hu and Raidl used two tries in the solution archive in order to exploit benefits of two different solution representations. Their implementation of the solution archive in combination with an evolutionary algorithm produced comparable results to existing state of the art metaheuristic approaches.

Heuristic Solution Techniques

In this chapter a brief overview about some of the most used metaheuristics will be given. Further the theoretic basics of the algorithms that the author used in his work will be explained.

4.1 Genetic Algorithms

Genetic Algorithms (GAs), which are first mentioned by Holland in [8], are population-based metaheuristics based on an observation of the nature—evolution. Genetic algorithms imitate survival of the fittest (selection), sexual reproduction (recombination) and random changes to the genotype (mutation).

Algorithm 1: Genetic Algorithm Scheme

```
1  $t \leftarrow 0$ ;  
2 initialize( $P(t)$ );  
3 while stopping criterion not met do  
4    $t \leftarrow t + 1$ ;  
5   select( $P(t)$ );  
6   recombine( $P(t)$ );  
7   mutate( $P(t)$ );  
8   evaluate( $P(t)$ );  
9 end  
10 return best solution in  $P(t)$ ;
```

In Algorithm 1 pseudocode for a generic GA is listed. The implementations of genetic algorithms differ not only in how the individuals are selected from the population $P(t)$ (line 5), how they are recombined (line 6) and how they are mutated (line 7) but also in the mutation rate, the number of generations and the number of individuals in each population. The population can either be constant or variable. Possible stopping criteria could be a time limit or a given number of created populations.

4.1.1 Memetic Algorithms

Memetic Algorithms (MAs) are genetic algorithms extended with a local search procedure. Moscato and Norman first mentioned memetic algorithms in [18]. A local search algorithm finds the local optimum within one (Local Search) or more (VND, VNS—see below) given neighborhoods. It is a design decision when the local search is applied and which solutions within the generation are selected for the procedure, but usually this is not done every generation for every solution. Often at the end of the GA a local search procedure is done with a larger neighborhood or with more neighborhoods when using a VND / VNS approach. A common scheme of a MA is listed in Algorithm 2. Since

Algorithm 2: Memetic Algorithm Scheme

```

1  $t \leftarrow 0$ ;
2 initialize( $P(t)$ );
3 while stopping criterion not met do
4    $t \leftarrow t + 1$ ;
5   select( $P(t)$ );
6   recombine( $P(t)$ );
7   mutate( $P(t)$ );
8   if local search criterion is met then
9     improveSolutions( $P(t)$ );
10  end
11  evaluate( $P(t)$ );
12 end
13 improveFinalSolutions( $P(t)$ );
14 return best solution in  $P(t)$ ;
```

2009 the *Springer Verlag* publishes a journal dedicated to memetic algorithms [14].

4.2 Ant Colony Optimization

An Ant Colony Optimization (ACO) is based on swarm intelligence. It was first introduced by Colorni et al. in [3] and mimicks the foraging behavior of ants. In an ant

colony the ants do not explicitly know what the other ants are doing, but nevertheless the ants of the colony behave in a very structured way. Every moving ant leaves pheromone on the ground, which can be detected by other ants and can influence the direction of their movement. While ants initially are moving in random directions for the search of food, over the time more and more ants are following the pheromone trails that were laid by other ants increasing the pheromone density for this trail. One can exploit this behavior to form a heuristic search algorithm in which the solution candidates are constructed step-by-step and the decisions are based on former good solutions, i.e., where the most pheromones are.

Algorithm 3: Ant Colony Optimization

```

1 initialize pheromone trail;
2 while stopping criterion not met do
3   | construct ant solutions based on pheromone information;
4   | if local search criterion is met then
5   |   | improveSolutions;
6   | end
7   | update pheromone trail;
8 end
9 return best solution found;

```

In Algorithm 3 the basic structure of an ACO is shown. A good overview about Ant Colony Optimizations and its different algorithms can be found in [5].

4.3 Local Search

Local Search (LS) aims to find the local optimum within a given neighborhood. A neighborhood structure of a solution is a function that maps a set of solutions to a solution, i.e., let $s \in S$ be an arbitrary solution, where S is the whole solution space. Then a neighborhood structure N is a function $S \rightarrow S^2$ — $N(s)$ is the set of neighbors of s (cf. [21]). While searching the neighborhood for other solutions three different functions (step functions) can be defined when a solution is accepted:

1. Random neighbor
Choose a random solution from the neighborhood although it could be worse than the original solution.
2. Next improvement
Search the neighborhood and accept the first solution that is better than the current solution.

3. Best improvement

Examine every solution in the neighborhood and take the solution with the best solution value (if there is one).

A general scheme for a local search procedure is given in Algorithm 4.

Algorithm 4: Local Search

```
1 define a start solution  $s$ ;  
2 define a neighborhood structure  $N(s)$ ;  
3 set step function = {random neighbor, next improvement, best improvement};  
4 while stopping criterion not met do  
5   | choose  $s' \in N(s)$  according to step function;  
6   | if  $s'$  is better than  $s$  then  
7   |   |  $s \leftarrow s'$   
8   | end  
9 end  
10 return  $s$ ;
```

The following two subsections describe metaheuristics that are similar to local search, but more complex.

4.3.1 Variable Neighborhood Descent

In Variable Neighborhood Descent (VND) multiple neighborhoods are explored systematically. In Algorithm 5 the structure of a VND is shown. Note that only *next improvement* and *best improvement* are valid step functions.

The neighborhood structures as well as their ordering is critical for the performance of a VND. Usually the neighborhood structures are sorted in order of increasing size, s.t. 'nearer' solutions are found earlier. A VND is based on the fact that a global optimum is also a local optimum, so that exploring different local optima can lead to the global one. Therefore, the result of VND is always an optimum with respect to all the defined neighborhoods.

4.3.2 Variable Neighborhood Search

In contrast to LS and the deterministic VND procedure, Variable Neighborhood Search (VNS) chooses a random solution of the current neighborhood, which is called *shaking*, and improves this solution using local search. To avoid the weaknesses of local search, it is often replaced by a VND because the VND is able to escape local optima. In

Algorithm 5: Variable Neighborhood Descent

```
1 define a start solution  $s$ ;  
2 define  $K$  different neighborhood structures  $N_i(s)_{1 \leq i \leq K}$ ;  
3 set step function = {next improvement, best improvement};  
4  $k \leftarrow 1$ ;  
5 while  $k \leq K$  do  
6   | choose  $s' \in N_k(s)$  according to step function;  
7   | if  $s'$  is better than  $s$  then  
8   |   |  $s \leftarrow s'$ ;  
9   |   |  $k \leftarrow 1$ ;  
10  | else  
11  |   |  $k \leftarrow k + 1$ ;  
12  | end  
13 end  
14 return  $s'$ ;
```

Algorithm 6 pseudocode of a VNS is shown. Successful applications of a VNS can be

Algorithm 6: (General) Variable Neighborhood Search

```
1 define a start solution  $s$ ;  
2 define  $K$  different neighborhood structures  $N_i(s)_{1 \leq i \leq K}$ ;  
3 while stopping criterion not met do  
4   |  $k \leftarrow 1$ ;  
5   | while  $k \leq K$  do  
6   |   | choose  $s'$  randomly from  $N_k(s)$ ;  
7   |   |  $s'' = \text{localSearch}(s') / \text{VND}(s')$ ;  
8   |   | if  $s''$  is better than  $s$  then  
9   |   |   |  $s \leftarrow s''$ ;  
10  |   |   |  $k \leftarrow 1$ ;  
11  |   | else  
12  |   |   |  $k \leftarrow k + 1$ ;  
13  |   | end  
14  | end  
15 end  
16 return  $s$ ;
```

found in [2, 7, 13].

4.4 Solution Archives

A common property of population based metaheuristics like a GA or MA is that they are revisiting algorithms, i.e., the same solution is generated more than once. This implies multiple evaluations of the fitness for the same solution while no additional information is added to the population. If we keep duplicate solutions in the population the GA could suffer from a loss of diversity, which could even lead to a premature convergence of the GA. Especially the calculation of the fitness value is usually a time consuming task. Thus, unnecessary re-evaluations should be avoided. A reasonable way to archive this is to add a solution archive to the metaheuristic. A solution archive stores all solutions visited so far and should have the properties that the insertion and search methods are efficient. In Algorithm 7 a memetic algorithm that uses a solution archive is listed.

Algorithm 7: A Memetic Algorithm with a Solution Archive

```
1  $t \leftarrow 0$ ;  
2 initialize( $P(t)$ );  
3 while stopping criterion not met do  
4    $t \leftarrow t + 1$ ;  
5   select( $P(t)$ );  
6   recombine( $P(t)$ );  
7   mutate( $P(t)$ );  
8   if local search criterion is met then  
9     improveSolutions( $P(t)$ );  
10  end  
11  foreach solution s in P(t) do  
12    remove  $s$  from  $P(t)$ ;  
13    if s is already in the archive then  
14       $s' = \text{generateNewSolution}(s)$ ;  
15    else  
16       $s' = s$ ;  
17    end  
18    insertIntoSolutionArchive( $s'$ );  
19    insert  $s'$  into  $P(t)$ ;  
20  end  
21  evaluate( $P(t)$ );  
22 end  
23 improveFinalSolutions( $P(t)$ );  
24 return best solution in  $P(t)$ ;
```

The archive has to fulfill the following tasks:

- insert a solution,
- check if a solution is already in the archive,
- generate a new solution if a duplicate was found.

Therefore, a data structure is needed that implements these tasks efficiently. A straightforward data structure for storing already visited solutions is a hash table or a binary tree. However, both of them cannot efficiently implement the third task. A trie-based solution archive [25,34] turned out to give the best results in terms of time and memory. In [34] Šramko compared the memory and time consumption for each of the tasks of the different data structures.

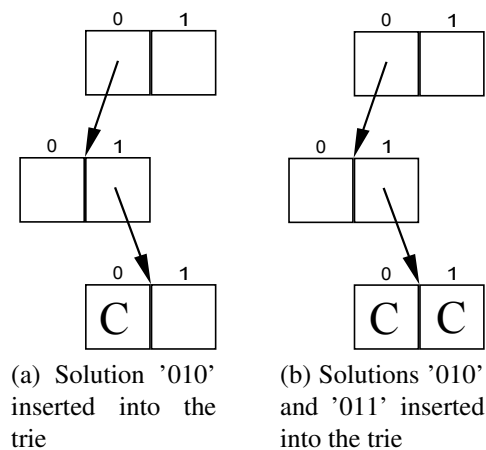


Figure 4.1: A schematic view of a Trie

In Figure 4.1 a solution archive implemented with a trie for a problem with solutions that can be represented as binary strings (i.e., a binary trie) is shown. It can be easily modified for other solution representations (e.g., integer arrays). The insertion method for a solution which is represented as integer array is illustrated in Algorithm 8. In case the array contains only 1s and 0s, the insertion method is also suitable for binary tries.

The search function is basically the same as the insert function but does not actually insert the solution. When a duplicate is found the search function returns `true`, else `false`. How the new solution is generated in case of a duplicate insertion differs from problem to problem. While the generation of a new solution is quite easy if a permutation of a solution could not generate invalid solutions, it can be more complex if this is not the case. In Section 5.3 it is described in great detail how the author of this thesis solved this task for the RCCSTD problem.

Algorithm 8: Insert

Input: Solution *sol* to be inserted as integer array

```
1 TrieNode current = root;
2 for int i ← 0 to sol.length-1 do
3   if current.get(sol[i]) ≠ null then
4     if current.get(sol[i]) is completed then
5       duplicate found;
6       return;
7     end
8     current=current.get(sol[i]);
9   else
10    create new trie node newNode;
11    current.set(sol[i], newNode);
12    current=newNode;
13  end
14 end
15 set current node to complete;
```

Another huge advantage of a trie over the other data structures is that a trie can be pruned. If all solutions of a subtree are already visited, the whole subtree can be pruned, which saves both search time and memory. An approach for pruning is given in Algorithm 9. After every insertion this pruning method is called in order to keep the number of nodes in the trie low. In Figure 4.2 a pruned trie is shown.

Algorithm 9: Subtrie Pruning

```
1 while current ≠ root do
2   if all children of current node are completed then
3     set current node to complete;
4   else
5     return;
6   end
7   current=current.parent;
8 end
```

There are several possibilities to enhance the performance of the trie, e.g., pruning subtrees whose solutions cannot have a better objective value than the best solution found so far—this could be implemented as computation of bounds during solution insertion. Another modification of the trie that could improve its performance is the randomization of the trie by a permutation of the insertion order of the solution parts.

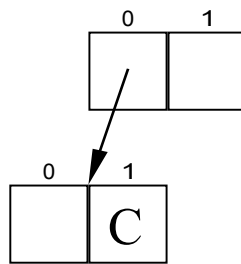


Figure 4.2: Trie from figure4.1b after pruning

If we deal with non-binary solution representations we have to deal with high memory consumption caused by the huge amount of null pointers stored in the trie. We will take a deeper look into the problem of high memory use and how to minimize it in Section 6.2.1.

Solution Archive

Based on an existing framework, which will be described in more detail in Section 6.1, a solution archive for the Reconstruction of Cross-Cut Shredded Text Documents (RCCSTD) problem was designed. For this problem a VNS, an ACO and an MA were developed and used in this framework, see [21] for the VNS and ACO and [29] for the MA.

In this chapter the details of the data structure and its algorithms will be discussed. It will also be shown how to integrate the solution archive into the already existing MA.

Two different types of solution archives were developed, which differ in their handling of duplicates. While the *trie-based permutation* (TBP) tries to iteratively build a solution by calculation of all shreds that could possibly be inserted at the specific position, the *shred-based permutation* (SBP) is based on a random shred exchange of two shreds.

5.1 Solution Representation

The existing algorithms in this framework store for each candidate solution for each position in the Euclidean space as described in Chapter 2 the corresponding shred.

While this solution representation is suitable for many genetic operators, it is not adequate for a solution archive. Let the number of shreds of the document be given by n . Then we have n^2 possible positions of each shred, so the height of the trie would be n^2 in the worst case.

So another solution representation is used, which is based on the neighbors of each shred. For each shred the right and the bottom neighbor is stored. Since the solution quality is based solely on the relative positions of the shreds to each other and not on the

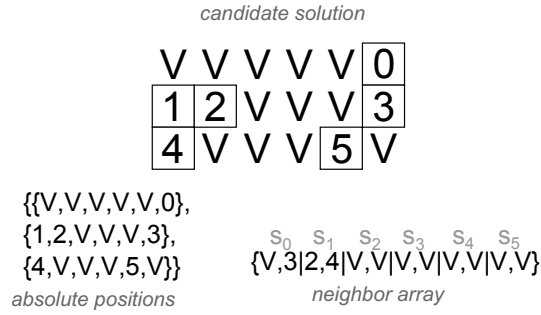


Figure 5.1: Comparison of the two solution representations

absolute positions in the Euclidean space, this is a valid solution representation. From now on we will refer to this representation as *neighbor array*. In this array the neighbors of each shred can be determined in the following way: For every shred with the ID i with $0 \leq i < n$ the position in the array of its right neighbor is $2i$ and the position of the bottom neighbor is $2i + 1$. In Table 5.1 an overview of the array positions and their corresponding entries is given.

Table 5.1: New solution representation

Array position	Shred
0	$n_r(s_0)$
1	$n_b(s_0)$
2	$n_r(s_1)$
3	$n_b(s_1)$
\vdots	\vdots
$2n - 2$	$n_r(s_{n-1})$
$2n - 1$	$n_b(s_{n-1})$

A comparison of the two types of the solution representations is given in Figure 5.1. Note that the virtual shred V is a constant and its value is always n (assuming that there are n shreds in the instance).

This method for storing solutions has two advantages:

- Drastically reducing the space needed for storing the solutions from n^2 in the worst case to $2n$.
- Identifying more duplicates since the absolute positions of the shreds are ignored.

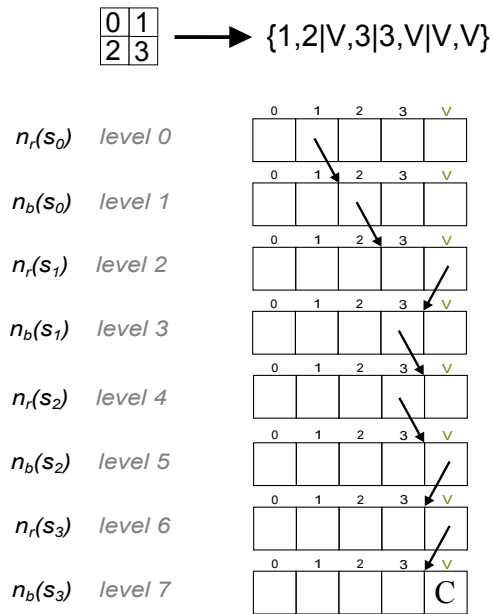


Figure 5.2: Insert a solution to the RCCSTD problem into the trie

The solution size could even be more reduced if we take into account that the relative position of each shred once fixed will not change anymore. So we can stop inserting the solution at the time when all shreds are fixed. We will discuss this termination condition of the insert function in Section 5.2.1.

5.2 Trie Design

In this section the methods that both the TBP and the SBP have in common are discussed. The insertion and the search function are designed according to the scheme described in Algorithm 8 with an additional calculation of available shreds per node. These available shreds are important for generating new yet unvisited solutions, which is described in the Sections 5.3 and 5.4. The calculation of the available shreds is also described in separated sections because it is dependent on which type of trie is used. Note that with the above solution representation we need two trie nodes for each shred (one for the right neighbor and one for the bottom neighbor). A sample trie after inserting one solution is shown in Figure 5.2. The number of children of each node equals the number of shreds plus one for the virtual shred, which of course can also be a neighbor of a shred.

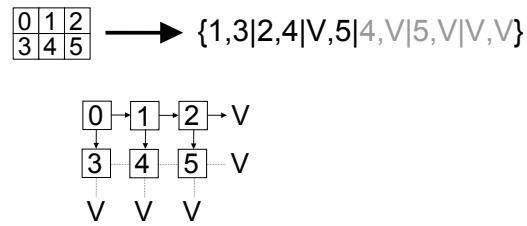


Figure 5.3: The last part of the solution is redundant (marked in grey)

5.2.1 Termination Conditions

Two termination conditions of the insert method were developed. The insert function terminates either if

- the whole solution is inserted or
- all shreds are connected in one shred block.

The latter is based on the observation that at the time all shreds are connected to each other they are all fixed—see Figure 5.3. Thus, inserting the remaining shred neighbors into the trie would not add more information.

The handling of duplicate solutions of the TBP approach is different from the SBP approach. Nevertheless, they have to fulfill the same task: Find a solution in a finite amount of time that is not already in the trie. One cannot just randomly guess a solution because of the restriction that the time needed for generating a new solution is bounded. Therefore, more sophisticated methods have to be applied. A general scheme for generating new solutions is described below:

1. Choose a node u , which has been visited while inserting the duplicate solution.
2. In this node, choose a child v that is different from the one chosen before.
3. Insert the remaining solution starting at v .

Usually the node u is either the last visited node or chosen randomly. For the second and third step it must be assured that no invalid solutions are generated. Invalid solution creation could easily happen since we have no restriction on the choice so far. This is also the reason why a set of available shreds is calculated and stored in each trie node. In the next two sections the trie dependant methods will be discussed.

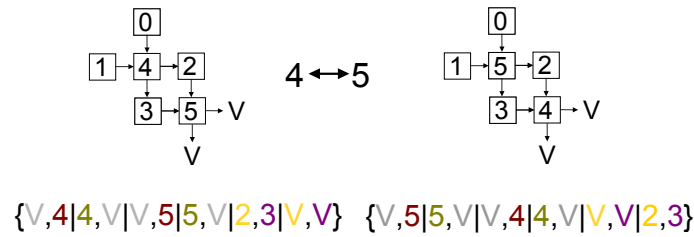


Figure 5.4: Only a constant amount of changes in the solution is needed

5.3 Shred-based Permutation

As mentioned before the shred-based permutation is based on a swap of two shreds. First a level is chosen randomly. The first shred to swap is the one that has previously been inserted on this level. The second shred is chosen randomly from a list of available shreds of the chosen node. After these shreds have been determined, the neighbor array is adjusted accordingly. Then, three different cases could occur:

1. A valid and new solution is generated.
2. An invalid solution is generated.
3. A valid solution is generated but this solution is already in the trie, i.e., a duplicate.

It is obvious that the first output is preferred and should be eventually reached, see Figure 5.5 for a duplicate detection and generation of a new and valid solution. In the following it is assumed that the first case happened. The other two cases are discussed in Section 5.3.3 and 5.3.4. Since a constant amount of shreds are swapped only a constant amount of changes in the neighbor array is needed, see Figure 5.4 for an illustration. That is the great advantage of this method because not only the generation of a new solution can be efficiently done but also the solution value can be updated in a constant amount of time. See Section 5.3.2 for a more specific explanation.

The question remaining is, how to determine the list of available shreds?

5.3.1 Calculation of the available Shreds

In each trie node a set is stored that contains all shreds available for swapping. This set is dynamically generated and updated during the insertion of solutions. Initially the set contains all shreds except the virtual shred. Then the following steps are performed:

$$\begin{array}{|c|c|} \hline 0 & 1 \\ \hline 2 & 3 \\ \hline \end{array} \longrightarrow \{1,2|V,3|3,V|V,V\} \quad \begin{array}{|c|c|} \hline 0 & 1 \\ \hline 3 & 2 \\ \hline \end{array} \longrightarrow \{1,3|V,2|2,V|V,V\}$$

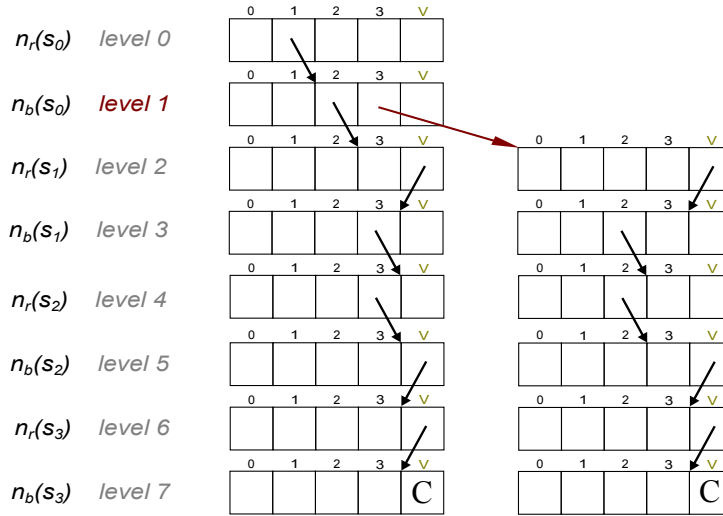


Figure 5.5: A new solution is generated using a shred based permutation

- Delete own shred, i.e. the shred with the ID $\lfloor \frac{\text{level}}{2} \rfloor$, and set an invalid flag at this child.
- Delete all shreds that have already been a right or a bottom neighbor earlier and set an invalid flag on the shreds that have already been a right/bottom neighbor on even/odd levels.
- Delete shreds with ID $< \lfloor \frac{\text{level}}{2} \rfloor$

The shreds in the set are basically those shreds that are free, i.e., do not have any neighbors until now and are not neighbors of any shred themselves. Once generated the set is only changed if a child node is set to complete. Then in the parent node this shred is deleted from the set. Note that the shreds that are deleted from the set are certainly invalid (or complete) shreds at the specific position but the shreds that remain in the set are not necessarily valid choices. The difficulty of calculating only valid choices is discussed in Section 5.4.1.

5.3.2 Altering Neighbor Array

The neighbor array has to be altered if a shred swap occurs. Fortunately this only needs a constant amount of eight changes, see Figure 5.4 for an illustration of which positions of the array have to be changed. There are two cases that can occur:

1. Two shreds are swapped

First the right neighbor of the first shred is swapped with the right neighbor of the second shred. The same is done for the bottom neighbors of the shreds (marked yellow and purple on Figure 5.4). Then, if the shreds have a top (left) neighbor that is not the virtual shred, then the bottom (right) neighbor of these neighbors are the shreds that have to be swapped. Therefore, the new bottom (right) neighbor of these shreds is the other shred of the swapped shreds (marked maroon and olive in Figure 5.4).

2. A shred is swapped with a virtual shred

This is a special case because the new neighbors of the shred are the former neighbors of the virtual shred. They cannot be determined by the neighbor array only because of the type of the solution representation in the trie. Therefore, an additional piece of information is needed—the level. With the help of the level the neighbors of this virtual shred can be specified by exploiting information about the absolute positions of the shreds.

5.3.3 Invalid Swaps

As mentioned before, an invalid shred swap can occur. Since invalid solutions are not allowed to be stored in the trie, a method has to be found to avoid such swaps. Furthermore, the algorithm needs to memorize when an invalid shred is found so that the same shred cannot be chosen again and again.

Claim 1. *Let l be the level that is chosen randomly and s_i be the chosen shred. Then a shred is invalid iff $n_t(s_i) < \lfloor \frac{l}{2} \rfloor$ or $n_l(s_i) < \lfloor \frac{l}{2} \rfloor$. (Note that the function $n_t(s)$ returns the position of the top neighbor of shred s and $n_l(s)$ returns the position of the left neighbor of shred s as defined in Chapter 2)*

Proof. The proof of the claim follows from the insertion order. First, the neighbors of the first shred are inserted, then the neighbors of the second shred and so on. This means that at level l all neighbors of shreds with an ID $< \lfloor \frac{l}{2} \rfloor$ are determined and cannot be changed. So, if the chosen shred has already been a neighbor of a shred with an ID smaller than its own ID, it cannot be chosen as a neighbor of another shred, since its relative position is already fixed. \square

To make sure that the algorithm steadily proceeds the previously determined invalid shred is deleted from the list of available shreds so that it cannot be chosen again. After an invalid shred was found the algorithm continues and tries another shred on the same node until the list of available shreds is empty. When the list of available shreds is empty

then this node can never be chosen again and for the current iteration another trie node is selected.

5.3.4 Duplicate generated

It is possible that the newly generated valid solution is already in the trie, i.e., a duplicate was generated. Only new solutions are accepted so the solution has to be transformed to a not yet generated one. Basically as many additional shred swaps as needed are made such that a new solution is found. The trie is traversed in a bottom up manner starting from the node where the duplicate was found. For each node all shreds from the set of available shreds are tried. If the root is reached during the traversal the enumeration is complete and all possible solutions have been visited and the algorithm terminates.

5.4 Trie-based Permutation

Unfortunately it turned out that the TBP approach was too complex, therefore inefficient, while it would not improve the results. Nevertheless, it was an interesting attempt, which is why it is described here.

In this permutation method the decision which shred is chosen at the selected level is entirely made in the trie based on the set of available shreds. This set contains all shreds that can be inserted at the specific position. In the next section it will be described how these sets can be computed.

5.4.1 Calculation of the available Shreds

The sets are calculated, like in the other method, during the insertion of the solutions. The sets are also initialized containing all shreds but in this case the virtual shred is included since it could be a valid neighbor of a shred. Then the following steps are performed:

1. Delete the own shred (as before).
2. Delete all shreds that have already been a right (on even levels) or a bottom neighbor (on odd levels).
3. If a shred is fixed at the current position then all other shreds are deleted. (see Section 5.4.1.1 for a definition of fixed shreds)
4. Delete all invalid shreds, see Section 5.4.1.2 for an explanation.

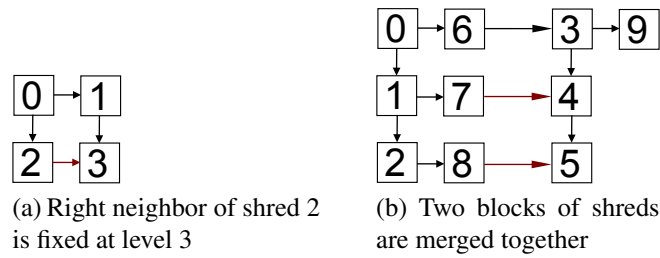


Figure 5.6: Example of fixed shreds

5. Delete all shreds that are connected in the same block as the own shred. This step is only done if there is no fixed shred at this level.

The remaining shreds represent the valid shreds at this position. Step 1 and 2 are basically the same as in the calculation of the available shreds of the shred based permutation method. Step 3 and 4 are explained in the next two sections. The last step is performed because of the observation that if two shreds are directly or indirectly (i.e., through intermediate shreds) connected then they cannot be direct neighbors again unless one shred is the fixed shred of the other.

5.4.1.1 Fixed Shreds

At each level there can be at most one fixed shred. A fixed shred is a shred that has to be placed at the specific position in any case. During the insertion at each level the fixed shreds for any future level have to be calculated and stored. A small and a more complex example of fixed shreds is given in Figure 5.6. The complex example points out what happens when two blocks of shreds are merged together. In Figure 5.6a the right neighbor of the shred with the ID 2 is fixed at the time when the bottom neighbor of the shred with the ID 1 is set (at level 3). The implicitly obtained right neighbor of shred 2, which is the shred with the ID 3, is indicated with a red arrow. In Figure 5.6b two neighbors are fixed at the time when the right neighbor of shred with the ID 6 is set to the shred with the ID 3. These two fixed shreds are also marked with red arrows in the figure.

The calculation of those fixed shreds is the big disadvantage of this permutation type. To be able to get the fixed shred at each level it is necessary to iteratively put the shreds together such that at each level the absolute shred positions are known. Unfortunately—to the author’s best knowledge—one cannot achieve this efficiently, i.e., without constantly copying array elements or recalculate positions, etc.

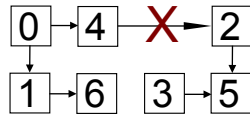


Figure 5.7: Example of an invalid shred due to a shred block merge

5.4.1.2 Invalid Shreds

Shreds that are invalid also have to be deleted from the set of available shreds. A shred is invalid if during a merge of two shred blocks a collision occurs, i.e., two shreds have the same absolute position, see Figure 5.7 for an illustration.

In the calculation of the invalid shreds lies the other difficulty of this permutation type. This is basically the same problem as the calculation of the fixed shreds with the addition that the calculation whether a shred is valid at the given position has to be made for each shred that is in the set of available shreds.

5.4.2 Advantages

The advantages of this method are that at each stage of the insertion all possible options for the next step are known. Actually this is only an advantage in the modelling of the algorithm, not in the implementation or in the efficiency. Another advantage is that under the premise that the set of available shreds can be efficiently computed, the algorithm is really easy and nearly all steps follow directly the general scheme given in Algorithm 8.

5.4.3 Problems

As mentioned before several problems occurred during this approach. A huge drawback is the lack of efficiency. A lot of computing power has to be invested in calculating the fixed and the invalid shreds without having an advantage in terms of solution quality. Although they could be calculated when the relative positions of the inserted shreds are saved and with the help of an algorithm for a shred block merge position update it would be too costly to compute. Another problem with this method is that the solution could look completely different from the originally inserted one, which may not be intended and raises the problem of the re-transformation of the solution to the original solution representation. Therefore, the inverse of the solution transformation function is needed, which also unnecessarily needs more computing power. For this reason it turned out that this solution permutation is not suitable for this problem, so it is completely omitted in the test cases.

5.5 Memetic Algorithm using a Solution Archive

The MA with the solution archive was designed according to the scheme of Algorithm 7. However, some adjustments were made. To keep some elitism, at every generation the best 10% of the current population are adopted by the next generation without inserting it into the solution archive (which would change each solution because it was already added to the archive before). The remaining 90% are inserted into the archive just as seen in the scheme mentioned above.

Implementation

The solution archive was implemented in Java and extended an already existing framework which contains several reconstruction approaches. This framework is also able to visually illustrate a candidate solution with its solution value. The solutions that are going to be inserted into the solution archive, which are represented as two dimensional `ArrayLists`, are transformed as described in Chapter 5.1 into an integer array.

In this chapter the existing algorithms are presented, which are later enhanced by the implemented solution archive. First, a brief summary of the construction heuristics that are used for the MA and the ACO are given. Then the MA operators, namely the select, recombination and mutate operators are described. The local search procedure at the end of the GA, which is a VNS, is described next. After that, a short explanation of the ACO is given and it is described, why the use of a solution archive would not improve the results of the ACO in this case. At the end of this chapter some implementation issues are discussed with a focus on how to handle the high amount of memory which is needed for storing the solutions.

6.1 Existing Framework

The framework was implemented in Java 1.6 and it uses SWT¹ for visually displaying candidate solutions. Either the commandline or the GUI² can be used to call the algorithms. Parameters can be used to control the program settings (e.g., input instance, reconstruction algorithm, etc.). The commandline interface is especially useful for larger testing purposes.

¹Standard Widget Toolkit

²Graphical User Interface

6.1.1 Construction Heuristics

Construction heuristics are used to generate an initial population for the MA. In [23] two different heuristics were introduced:

- *Row Building Heuristic*
For this method it is assumed that each row of a solution begins with a shred with a white left border and ends with a shred with a white right border. So for each row a shred with a white left border is chosen, then the shreds are added using a best fit heuristic until a shred with a white right border is added. Then a new row begins and the procedure is repeated.
- *Prim-Based Heuristic*
This construction heuristic is based on the algorithm of Prim for finding minimum spanning trees, see [24]. The algorithm starts with an arbitrarily chosen shred, which is placed in the top left corner, i.e., at position $(0, 0)$. Then one shred is added at a time, which currently induces the minimal error over all shreds with the additional restriction that the shred can only be inserted on positions next to already assigned shreds.

Half of the individuals needed for the initial population are generated using the Row Building Heuristic, the other half is generated using the Prim-Based Heuristic.

6.1.2 Memetic Algorithm

In this section a short summary of the MA will be given, which was introduced by Schauer in [30].

6.1.2.1 Selection

The selection of individuals is done by choosing shreds randomly using an equally distributed function. In his tests this performed better than other selection methods, especially the classical fitness-proportional selection. In addition the best 10% of the current population is copied unchanged to the next generation to guarantee the selection pressure.

6.1.2.2 Recombination

In the following several recombination methods are presented. Although Schauer developed more methods than are described here, the author of this thesis only describes those which will later be used in combination with the solution archive.

Horizontal Block Crossover Horizontal Block Crossover (HBX) is a type of a 1-point Crossover. As this problem can be seen as 2-dimensional, a horizontal splitting line instead of a splitting point is chosen. This line is chosen randomly using a Gaussian distribution function and is applied to the shorter of the two parents. With this method invalid solutions could be generated because shreds that occur in the upper part of one parent could also appear on the lower part of the other parent. Therefore, a check is needed to assure that only yet unused shreds are inserted. Of course it could happen that some of the shreds were not inserted at all. The upper part of both parents are inherited unchanged and the unassigned shreds are inserted at the end of the method using a best fit heuristic, i.e., they are inserted at the position which induces a minimal additional error over all possible positions (without replacing another shred).

Vertical Block Crossover Vertical Block Crossover (VBX) is basically the same as HBX, except that the splitting line is chosen vertically. The left part of both parents is inherited unchanged to the descendants. The handling of the unassigned shreds is done the same way as in HBX, i.e., they are inserted using a best fit heuristic.

Biased Uniform Crossover Best Uniform Crossover (BUX) is based on the observation that if two shreds fit together well in one parent they should stay together in the descendants. This leads to the following method: Decide for each position whether to take the shred from the first parent or from the other. The decision which shred fits better is based on the error induced in the offspring if the next shred is taken from parent one or two. Since the shape of the two parents might be different, the first child inherits the shape of the first parent and the second child inherits the shape of the other parent.

6.1.2.3 Mutation

In this section the mutation operators are described. Again, the focus lies on those operators that will be used with the solution archive.

Horizontal Flop Mutation HFM is the mutation analogon to the HBX recombination method. The individual is split along a horizontal line, then the two parts are swapped. This results in changes of the top/bottom neighbor relation only of the shreds along the splitting line, while all left right relations remain the same.

Vertical Flop Mutation VFM complements HFM with the difference that a vertical splitting line is chosen. In addition the empty positions of the left side of the mutated solution are filled with the virtual shred so that the top/bottom relation of all shreds are preserved.

Swap Two Mutation S2M is the simplest one: it just swaps two randomly chosen shreds. This method is repeated up to ten times during one mutation and the exact number of repeats is chosen randomly.

Break Line Mutation Due to the nature of the recombination operators it was observed that the lines of the individuals become longer over the time. The Break Line mutation (BLM) operator was designed to solve the problem of long lines. It finds the longest line of the individual, breaks it apart at a random position and inserts the right half as a new line into the end of the solution. The other half of the line stays unchanged at its position.

6.1.2.4 VNS

For the improvement of the final solution pool a VNS was applied. See Section 4.3.2 for a general description of a VNS. The VNS is taken from [23], where Prandtstetter introduced seven neighborhoods. First two moves are defined:

- *Swap Move*
Two shreds are swapped.
- *Shift Move*
A rectangular region of shreds is shifted within the solution.

Based on these two moves seven neighborhood structures were defined:

1. *Simple Swap*: One swap move is applied.
2. *Simple Shift*: One shred is shifted either horizontally or vertically.
3. *Simple Block Shift*: A single row or column of defined length is shifted.
4. *Rectangle Block Shift*: A block of shreds is shifted, where its width and height is chosen arbitrarily.
5. *Simple Double Shift*: One shred is shifted horizontally, then vertically.
6. *Simple Double Block Shift*: A simple row or column is shifted first horizontally then vertically.
7. *Rectangular Double Block Shift*: A rectangle of shreds of arbitrary width and height is shifted first horizontally then vertically.

For the shaking in the i -th neighborhood structure i^2 randomly chosen shift moves of single shreds are performed.

6.1.3 Ant Colony Optimization

For the ACO presented in [23] two pheromone matrices were introduced. One of them corresponds to the pheromone laid for placing one shred to the right of another shred and the other to the pheromone laid for placing one shred on top of another shred. The pheromone matrices are initialized in a two-stage procedure. First, five candidate solutions are generated using five different construction heuristics. Two of them are described above in Section 6.1.1 and the other three are described in Prandtstetter's work in [23]. Then all entries of both matrices are set to τ^0 where

$$\tau^0 = \frac{m}{\min_{1, \dots, 5} c(\Pi_i)} \quad (6.1)$$

and m denotes the number of ants. In the second step a pheromone update is performed, see Section 6.1.3.2.

6.1.3.1 Construct Ant Solutions

In [23] three different types of solution construction heuristics are presented. The type that performed best was the *Randomized Row Building Heuristic*. It is based on the row building construction heuristics and reconstructs a set of rows using a probability distribution. The shred that is chosen is not only based on the cost value but also on the pheromones produced by the ants. For each shred, the probability value

$$p_{ij} = \frac{\tau_{ij}^\alpha \cdot \left(\frac{1}{c(i,j)}\right)^\beta}{\sum_{k \in S'} \tau_{ik}^\alpha \cdot \left(\frac{1}{c(i,k)}\right)^\beta}, \quad \forall i \in S \setminus S', j \in S', \quad (6.2)$$

where S' is defined as the set of shreds not currently used in the intermediate solution. The solutions that are generated using this method are enhanced by a VND using the first three neighborhoods which are described in Section 6.1.2.4.

6.1.3.2 Pheromone Update

The pheromone update is done using the following formulas, where k is the solution obtained by ant k during the last iteration.

$$\begin{aligned}
\tau_{ij} &= (1 - \rho) \cdot \tau_{ij} + \sum_{k=1}^m \Delta_{ij}^k + \Delta_{ij}^0, & \forall i, j \in S, i \neq j \\
\Delta_{ij}^k &= \begin{cases} \frac{1}{c(\Pi_k)} & \text{if } j \text{ is placed right next to } i \text{ in the } k\text{-th solution} \\ 0 & \text{otherwise} \end{cases} & \forall i, j \in S, \forall k = 0, \dots, m
\end{aligned} \tag{6.3}$$

The other pheromone array is updated analogously.

6.1.3.3 The ACO and the Solution Archive

In preliminary tests it turned out that the ACO described above does not produce a significant amount of duplicate solutions. Therefore, using a solution archive would be a complete overhead in terms of time and memory consumption, since a solution archive could only improve the results of a heuristic algorithm if it produces duplicate solutions. This is why the ACO is completely omitted in the tests.

6.2 Implementation Issues for the Solution Archive

In this section implementation issues for the solution archive itself are discussed. First it is described how the high memory consumption is handled, especially how the trie nodes are implemented and which data structures are used. Then the calculation of the shred blocks is presented which should be as time-efficient as possible.

6.2.1 Memory Consumption

Since a high memory consumption lies in the nature of solution archives, efficient data structures have to be used to keep the usage as low as possible. Suppose that an instance of the RCCSTD problem consists of 225 different shreds, which corresponds to a 15×15 cutting pattern. Then the size of each node is 226 (225 shreds plus 1 for the virtual shred). Each solution needs, in the worst case, $225 \cdot 2 = 450$ nodes. Assume that a pointer to a child needs 8 bytes of memory. Then each solution would approximately consume (in the worst case) 800kB of memory!

6.2.2 Trie nodes

The example above should make clear that the choice of a datastructure for the trie nodes have a high impact of the overall memory consumption of the implementation. Several different types of datastructures were tested and compared.

- Array

The standard method for storing children of a trie node needs a lot of memory since at the time of the initialization of the array all of the memory that is possibly needed is allocated. A huge advantage of this method is the time efficiency: Looking up an element in an array takes $O(1)$ time.
- Map

When using a map as datastructure only those children are stored that are actually in the trie, i.e., there is no overhead because of empty children.

 - Hashmap

It turned out that a Java Hashmap actually consumes memory for unassigned entries (probably to enforce the amortized $O(1)$ entry lookup time). Due to the additional overhead of the hashmap this datastructure is not suited for this problem.
 - Treemap

In contrast to the *Hashmap* a *Treemap* uses a (Red-Black) tree to store its entries³. Since a Red-Black tree is a balanced tree a $\log(n)$ lookup time is guaranteed, which is acceptable because most trie nodes are densely populated. Unfortunately the overhead for the map entries is quite big (an entry needs approximately 40 bytes of memory), so the possible memory savings are nullified.
- List

The use of lists for storing the children of the trie nodes proved to be the best method in practice. With the improvement described in Section 6.2.2.1 only 'real' nodes (i.e., omitting complete and invalid nodes) are stored. When using a *vector* as datastructure retrieving a child only needs a constant amount of time since its underlying datastructure uses element indexing like an array. The only drawback of using a vector (with custom initial capacity and increment steps) is that if the capacity is too small to store the new child, all the elements have to be copied. Preliminary tests showed that all nodes except the nodes on the first few levels had a load factor of 2 to 3%. Therefore, the initial capacity was set to 5% of the total number of shreds and the capacity increment was set to 1 which means that if the size is currently too small the size is increased by 1.

For the set of available shreds a Java `BitSet` is suitable, since the delete operation is efficient for this datastructure.

³<http://docs.oracle.com/javase/1.5.0/docs/api/java/util/TreeMap.html>

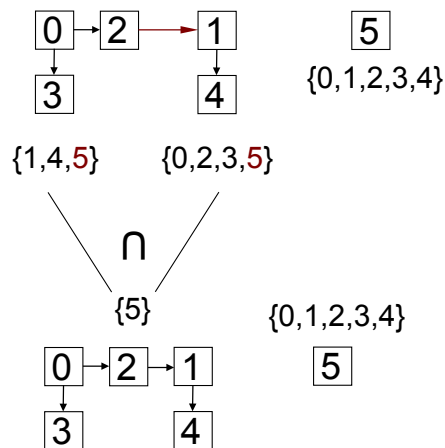


Figure 6.1: Two blocks of shreds are merged

6.2.2.1 Completed Nodes

To further reduce memory consumption, the completed and invalid nodes are not actually stored. Each node stores a set containing the indices of the child nodes that are already completed or invalid.

6.2.3 Calculation of Shred Blocks

For each shred block (set of connecting shreds, might just contain 1 element) there exists a set of integers (i.e., the indices of a Java `BitSet`), which corresponds to the shred IDs. Those integers are the shreds that are currently not connected to the shred block. Whenever a shred block merge happens the two sets of shreds can be merged by a logical AND operation (set intersection). See Figure 6.1 for an illustration of a shred block merge operation of two blocks of shreds.

CHAPTER 7

Tests

For testing the solution archive ten different documents are used, see Appendix A. This is the same set of documents which was defined by Prandtstetter in [21]. They are cut in 9 different predefined patterns which range from 9×9 to 15×15 . Most of the documents are text documents in A4 format but some also contain images and have a different size.

To make a fair comparison of the MA with and without the solution archive not the number of generations is taken into account but the running time. First the MA with the solution archive runs a specific number of generations. This number is determined and limited by available amount of memory. Then the result is compared to the MA without the archive, which runs the same time as the MA that uses the solution archive. This means that the MA without the archive runs a lot more generations than the MA with the archive. Thus, it is rather unlikely that the MA with the solution archive have already converged to a solution value. Therefore, it is assumed that the algorithm has even more potential but it cannot easily be exploited because of the memory restriction.

Two configurations of the MA were compared with and without the solution archive. They are taken from [29] where Schauer introduced some test configurations.

- HVREA

The HVREA uses the HBX and the VBX recombinations operators, see Section 6.1.2.2 for a description of them. Only the best offspring (out of two) of each operator is used for the next generation. The mutation rate is 25% and the operators that are used with the given probability can be seen in Table 7.1.

HFM	VFM	BLM	S2M
5%	5%	10%	5%

Table 7.1: Mutation rates of the HVREA

- BNREA

BNREA uses BUX recombination operator and both descendants are used for the next generation. The mutation rate is as well 25% and the probabilities of the mutation operators are given in Table 7.2

HFM	VFM	S2M
5%	15%	5%

Table 7.2: Mutation rates of the BNREA

For each of the 90 different test cases 30 runs were performed and executed on a single core of an Intel Xeon Quadcore CPU with 2.53GHz and 23GB of RAM.

Since the sizes of the instances and cutting patterns are very different, not all instances are run with the same number of generations. The population size is set to 50 on all instances but the number of generations is set to 3000, 5000 or 10000 and depends on the size of the instance, i.e., the number of non-blank shreds. See Table B.1 in Appendix B for information which instances runs with which number of generations.

A VNS was used to improve the solutions in the following way: after 3000 generations a VNS with neighborhoods N_1 to N_3 (see Section 4.3.2) was performed. In the end a more exhausting VNS was done which uses all seven neighborhoods N_1 to N_7 described in Section 4.3.2.

In Figure 7.2 a perfectly reconstructed document is shown (instance p01) while on some instances the output is not as good which is illustrated in Figure 7.1, which is clearly a not perfectly reconstructed document.

An algorithm can only benefit from a solution archive if the algorithm creates a significant number of duplicate solutions. The chart in Figure 7.3 shows the number of duplicates of some instances.

From Figure 7.3 it can be concluded, that the larger the instance the less duplicates will be generated by the MA. The instance *p01* is one of the smaller instances and for the 9×9 cutting pattern every fourth solution is a duplicate. Even in the largest instance of

whi improvement. The time (in seco rior instances averaged maximum run t^{ers}, which was also used as time l the executions of CBTC and RTC. fice paired Wilcoxon signed r med.

7.1 suitable for this type of instances deviations are eta-heuristics bounds and (nes. Ins with random edge costs. The of f_{max} is 10^{-1} average maximum running tim FC are listed.

	without VND			C _s [s]
	RTC	Cd ^d	J ^d	
175.7382 (4.23)				237.1403 (6.28)
163.1926 (4.31)				224.3123 (5.72)
149.9852 (5.14)				210.9872 (7.63)
139.9790 (4.32)				197.1772 (7.99)
128.1830 (4.90)				183.0157 (8.03)
119.5551 (4.46)	46.4919 (3.88)	68.3241 (0.72)	68.3226 (0.1)	172.8291 (10.59)
110.6725 (4.39)	80.8636 (2.40)	47.4045 (4.85)	47.1732 (2)	241.3032 (5.09)
	41.1201 (0.68)	33.5460 (0.67)	33.3408 (6)	222.1441 (4.50)
	35.7590 (0.47)	32.2571 (0.48)	31.956 (9)	204.6141 (6.00)

It can also be seen tha to t the runtime of the clustering he t^{ers} iteratively with the number of level tics clustering. When *D* is odd the set sta dominates the runtime. Thus, v beering heuristics even get faster s paral center edges to be considered, heated preprocessing step, decreases ple **1: Average tree instance** in a few sec **10 nodes for val, the standard** leading metahe **addition, the aster heuristics were** used as a **OR-Library** **used as time** **used as time** **used as time** **used as time**

Table 1 summarizes the results obta **CBTC and R**

	CBTC	RTC	Cd ^d	J ^d	t _{max} (C) [s]	t _{max} (C) [s]
329.0261 (6.02) 1)	65.2061 (0.55)	65.1598 (0.56)	65.86 (0.70)	2.54 (0.09)	15 in-	15 in-
306.2655 (9.02)	41.4577 (0.36)	41.3127 (0.50)	42.12 (4.01)	4.55 (0.49)	15 in-	15 in-
288.3842 (7.52)	35.0511 (0.35)	34.2171 (0.29)	34.74 (1.34)	5.92 (0.42)	15 in-	15 in-
266.3665 (9.01)	32.1181 (0.31)	30.9704 (0.24)	31.08 (0.66)	6.79 (0.42)	15 in-	15 in-
250.0016 (8.01)	30.2897 (0.29)	29.1796 (0.26)	28.63 (0.44)	7.11 (0.33)	15 in-	15 in-
249.0940 (0.28)	233.3644 (0.30)	31.3790 (0.37)	31.0176 (0.33)	7.00 (0.64)	15 in-	15 in-
28.2433 (0.28)	232.1965 (0.24)	30.7937 (0.33)	30.4287 (0.29)	7.20 (0.72)	15 in-	15 in-
27.9008 (0.27)	231.5826 (0.24)	30.5182 (0.29)	30.1348 (0.27)	7.32 (0.81)	15 in-	15 in-
27.1091 (0.26)	231.2682 (0.22)	30.3116 (0.31)	30.0384 (0.28)	7.57 (0.76)	15 in-	15 in-
26.6984 (0.28)	231.0864 (0.22)	30.2344 (0.30)	30.0739 (0.28)	8.56 (0.98)	15 in-	15 in-

We also performed test where a stro hood descend (VND) as proposed in to the best solutions of the various co As expected, it flattens the differences derived from clustering heuristic soluti

Figure 7.1: A not perfectly reconstructed document (p10)

the tested set of instances, the *p07* 9×9 instance, 10% of the solutions are duplicates. The number of duplicates of the other instances lies somewhere in between.

The number of shred swaps needed to generate a new solution was also analyzed. We distinguish two values: the number of swaps needed for a new solution without counting invalid swaps, i.e., only counting swaps that generated a feasible solution again, and the total number of swaps needed, i.e., including the invalid swaps. The first number should be low and ideally somewhere around 1 to avoid excessive duplicate generation. The second number is expected to be much higher because the invalid swaps are also counted. Although the check if a swap is invalid is quite fast, see Section 5.3.3, it is interesting how many such swaps are made. In Figure 7.4 the number of shred swaps needed without counting the invalid swaps is demonstrated on the sample instance *p01*.

It turned out that the number is very low even for small instances and on larger instances the number is even lower than 1.1.

In Figure 7.5 the total number of needed shred swaps is summarized for the sample instance *p03*. In this figure it can also be seen that the number of shred swaps needed slightly decreases with the number of generations. This is because whenever an invalid swap is made a branch of the tree is cut, so that this swap cannot be made again.

The tables below show all results using the configurations described above. The percentage value in the cells is the difference in percent of the given configuration in comparison to the optimal solution value. Note that negative values can appear because the

Vorwort

Die rasante Entwicklung im Bereich der Informations- und Kommunikationstechnologie hat in den letzten Jahren zu einem enormen Bedarf an universitär gut ausgebildeten Arbeitskräften in verschiedensten Bereichen der Wirtschaft, speziell im Raum der EU, geführt. Die Studienkommission Informatik am Standort Wien reagierte im Jahre 2001 auf den akuten Bedarf insbesondere an universitär ausgebildeten Informatikfachkräften durch eine Umstellung des Diplomstudiums Informatik auf dreijährige Bachelor- und darauf aufbauende zweijährige Masterstudien. Diese Gliederung entspricht auch dem Geist der Bologna Erklärung, in welcher der Wille zu einer derartigen EU weiten Entwicklung der Studienpläne bekundet wurde. Kürzere Normstudienzeiten erhöhen die Attraktivität der Studien für MaturantInnen und senken die Rate der StudienabbrecherInnen.

Das Ausbildungsniveau der AbsolventInnen der Masterstudien entspricht jener der AbsolventInnen des bis 2001 geführten Diplomstudiums Informatik. Die nun noch besseren Möglichkeiten zur Spezialisierung sowie die vielfältigen Kombinationsmöglichkeiten der Bachelorstudien (auch anderer Studienrichtungen) mit den angebotenen Masterstudien der Informatik orientieren sich an den stetig wachsenden Anforderungen der Wirtschaft an flexiblen, interdisziplinär ausgebildeten AkademikerInnen.

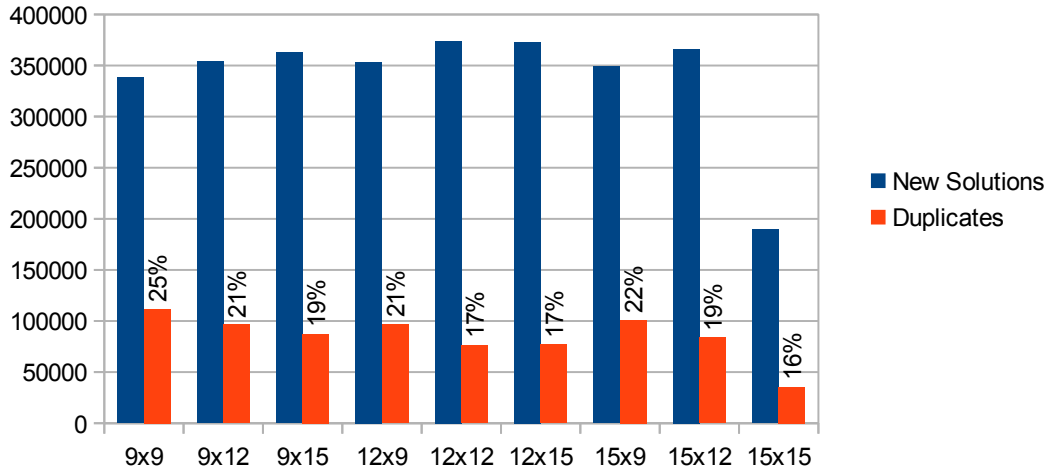
Die Bachelorstudien *Data Engineering & Statistics*, *Medieninformatik*, *Medizinische Informatik*, *Software & Information Engineering* sowie *Technische Informatik* vermitteln eine fundierte Grundlagenausbildung mit Schwerpunktsetzungen, die sowohl den klassischen Bereichen der Informatik (Software & Information Engineering, Technische Informatik) als auch aktuellen Trends (Data Engineering & Statistics, Medieninformatik, Medizinische Informatik) Rechnung tragen.

Die Masterstudien *Computational Intelligence*, *Computergraphik & Digitale Bildverarbeitung*, *Information & Knowledge Management*, *Medieninformatik*, *Medizinische Informatik*, *Software Engineering & Internet Computing*, *Technische Informatik* sowie *Wirt-*

Figure 7.2: A perfectly reconstructed document (p01)

solution that was found is even better than the original objective value regarding the objective function which shows that this function has some weaknesses. In Table 7.3 the results of the HVREA configuration without a VNS in the end is compared to the same configurations of the HVREA using the solution archive. Note that in this table also the results of the HVREA with an intermediate VNS after 3000 generations is shown. The intermediate VNS uses only the first three neighborhood structures described in Section 4.3.2.

p01 - HVREA



p07 - HVREA

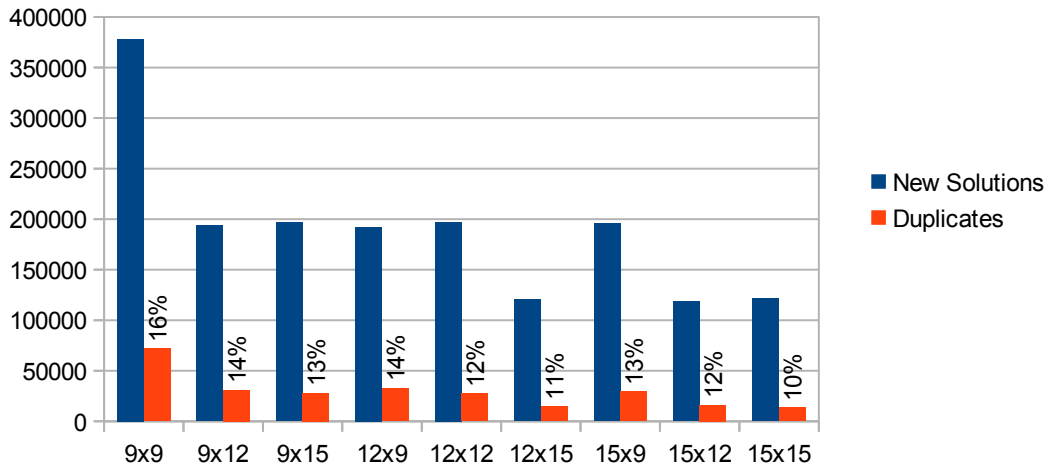


Figure 7.3: Number of duplicates in instance p01 and p07

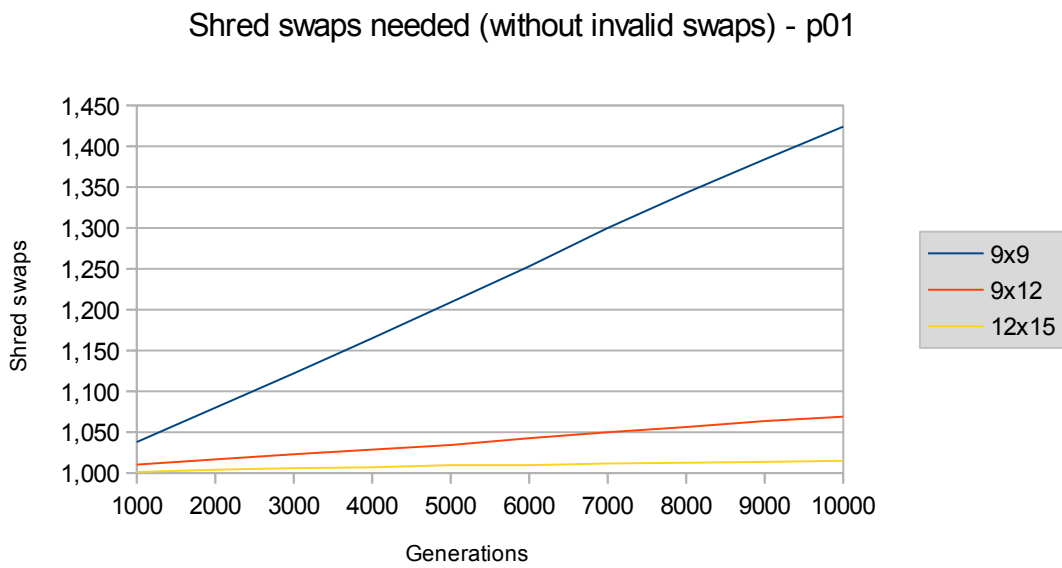


Figure 7.4: Number of shred swaps needed for a new solution (excluding invalid swaps)

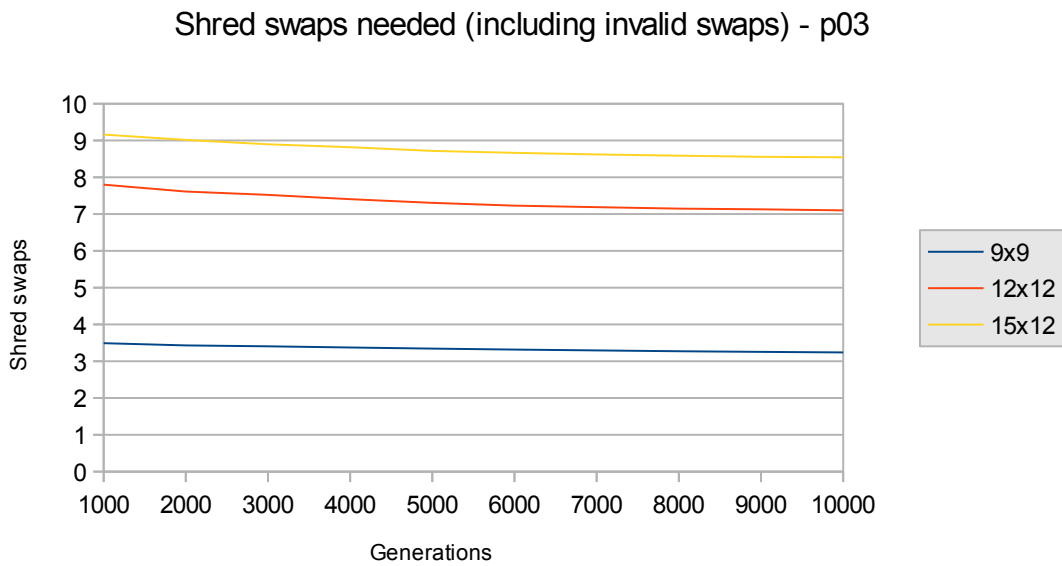


Figure 7.5: Number of shred swaps per new solution (including invalid swaps)

Table 7.3: The mean percentage gaps of 30 runs with the given configurations and **without** the VNS at the end. The column description indicates the use of the HVREA configuration with archive (+A) or without and with intertwined VNS (+V) or without. The entries in column p indicate whether the result with the solution archive or the result without the archive is better ($>$) or equal (\approx) according to a Student's t-test with an error level of 5%

	x	y	orig	HVR+A		p	HVR		HVR+A+V		p	HVR+V	
				gap	dev		gap	dev	gap	dev		gap	dev
Instance p01	9	9	2094	7,1%	12,9	>	0,0%	0,0	1,3%	4,1	\approx	0,0%	0,0
	9	12	3142	35,4%	8,3	>	28,5%	11,0	16,9%	3,9	\approx	15,2%	5,3
	9	15	3223	41,8%	8,6	>	35,8%	6,2	18,1%	6,6	\approx	15,5%	7,5
	12	9	2907	34,2%	5,3	>	24,7%	13,5	14,2%	8,2	\approx	13,9%	7,7
	12	12	3695	36,4%	4,7	>	32,6%	4,8	15,4%	3,0	\approx	14,8%	2,4
	12	15	3825	44,0%	5,0	>	40,0%	7,6	18,0%	2,5	\approx	17,9%	2,8
	15	9	2931	22,8%	18,0	\approx	19,7%	20,0	18,8%	10,9	>	11,2%	12,6
	15	12	3732	39,8%	6,1	>	34,8%	6,5	17,9%	2,9	\approx	16,5%	4,6
	15	15	3870	53,2%	5,5	>	49,1%	6,8	20,8%	2,8	\approx	20,7%	2,8
Instance p02	9	9	1434	-13,4%	12,7	>	-20,8%	8,7	-25,5%	2,9	\approx	-26,4%	3,8
	9	12	1060	27,0%	13,8	>	11,7%	11,9	9,3%	5,1	>	4,9%	4,1
	9	15	1978	13,3%	5,5	>	4,1%	4,9	-4,7%	2,7	\approx	-5,5%	3,2
	12	9	1396	-0,9%	8,0	>	-9,1%	9,3	-17,6%	5,5	>	-20,6%	5,7
	12	12	1083	27,4%	17,8	>	14,5%	12,8	10,5%	5,4	>	4,2%	5,5
	12	15	1904	20,6%	9,0	>	12,1%	9,2	-2,1%	3,3	\approx	-2,8%	2,8
	15	9	1658	7,9%	11,3	>	-4,8%	8,8	-9,3%	4,9	>	-11,6%	4,1
	15	12	1503	23,8%	12,4	>	12,3%	12,7	6,9%	7,0	>	2,5%	4,3
	15	15	2283	21,6%	7,3	>	8,8%	8,6	1,4%	3,3	>	-0,2%	3,1
Instance p03	9	9	2486	14,7%	11,3	>	6,6%	13,1	2,7%	6,9	\approx	1,3%	8,2
	9	12	2651	28,7%	13,3	>	20,9%	13,9	13,8%	6,2	\approx	10,1%	7,9
	9	15	2551	11,5%	12,9	>	4,0%	6,9	4,5%	4,9	\approx	2,4%	3,6
	12	9	3075	21,7%	7,4	>	13,6%	6,7	10,2%	3,7	\approx	9,7%	4,2
	12	12	3377	33,1%	7,1	>	28,1%	7,3	12,5%	3,9	\approx	10,8%	4,6
	12	15	3313	22,5%	10,2	>	10,7%	11,5	4,7%	5,0	>	1,4%	4,1
	15	9	3213	22,7%	5,5	\approx	20,0%	7,4	8,5%	4,5	\approx	7,3%	4,1
	15	12	3278	39,2%	9,4	>	32,7%	11,8	22,5%	4,3	>	18,5%	5,8
	15	15	3308	34,3%	15,4	>	15,3%	14,6	10,1%	5,1	\approx	7,2%	6,4
Instance p04	9	9	1104	14,7%	16,6	>	-0,4%	15,0	-10,5%	12,7	\approx	-10,4%	13,8
	9	12	1463	11,1%	11,5	>	1,9%	11,4	0,0%	6,0	>	-3,8%	8,0
	9	15	1589	-0,2%	8,8	>	-10,1%	7,4	-14,2%	4,5	\approx	-15,9%	4,7
	12	9	1515	34,2%	13,0	>	22,9%	15,7	8,9%	10,0	>	3,1%	7,7
	12	12	2051	22,4%	4,7	>	17,4%	6,7	5,5%	3,1	>	3,4%	4,6
	12	15	2146	4,1%	7,3	>	-4,4%	6,5	-9,6%	3,4	>	-11,7%	3,1
	15	9	1567	35,8%	12,2	>	24,4%	14,6	6,2%	8,1	\approx	6,0%	7,8
	15	12	1752	38,9%	11,8	>	26,0%	9,1	18,4%	6,3	>	14,0%	7,4
	15	15	2026	8,6%	8,8	>	0,7%	6,6	-2,9%	4,3	\approx	-4,6%	4,6
Instance p05	9	9	690	2,8%	9,2	\approx	0,0%	0,1	0,4%	2,1	\approx	0,0%	0,0
	9	12	888	81,1%	29,2	>	47,8%	33,4	34,4%	18,3	>	21,6%	16,2
	9	15	1623	50,8%	11,8	>	34,5%	16,0	19,6%	5,1	\approx	18,4%	6,1
	12	9	1016	24,1%	18,0	>	9,6%	15,0	3,7%	6,5	\approx	2,9%	6,1
	12	12	1325	47,4%	18,0	>	36,5%	20,6	13,3%	8,7	>	8,1%	10,5
	12	15	1986	55,5%	9,7	>	44,8%	20,7	22,8%	4,7	>	19,7%	5,9
	15	9	1010	-8,6%	14,0	>	-16,0%	9,8	-16,3%	6,7	\approx	-18,4%	3,0
	15	12	1156	55,2%	31,3	>	23,3%	23,0	16,1%	12,8	\approx	14,4%	14,8
	15	15	1900	66,8%	12,2	>	56,9%	20,0	15,9%	5,3	\approx	14,1%	4,6

	x	y	orig	HVR+A			HVR		HVR+A+V			HVR+V	
				gap	dev	p	gap	dev	gap	dev	p	gap	dev
Instance p06	9	9	2184	20,2%	6,5	≈	17,3%	7,2	-0,8%	2,6	>	-2,2%	2,8
	9	12	2915	24,2%	7,4	≈	20,0%	10,3	-2,2%	2,6	≈	-3,0%	3,3
	9	15	2265	72,3%	13,5	>	58,4%	16,9	15,7%	5,0	≈	16,6%	3,3
	12	9	2162	37,1%	7,1	>	32,6%	9,7	7,1%	3,3	≈	7,3%	3,6
	12	12	3031	37,5%	7,2	>	33,0%	4,6	6,6%	2,2	≈	6,6%	2,7
	12	15	2401	76,9%	18,0	>	64,5%	17,5	23,3%	4,7	≈	23,0%	4,7
	15	9	2719	34,0%	7,7	>	27,7%	8,0	1,3%	3,0	≈	0,9%	2,7
	15	12	3452	32,9%	5,5	>	25,9%	9,3	1,0%	3,2	≈	0,5%	2,7
15	15	2928	85,1%	12,1	>	75,7%	13,4	85,5%	11,2	>	15,4%	4,0	
Instance p07	9	9	6461	-11,4%	4,4	≈	-13,3%	3,2	-23,7%	2,6	≈	-24,1%	2,4
	9	12	6856	18,7%	6,9	>	11,6%	6,5	-14,6%	7,2	≈	-16,5%	5,5
	9	15	6952	34,1%	7,4	≈	33,8%	9,5	-10,6%	9,5	≈	-10,7%	9,4
	12	9	6758	-12,1%	3,6	≈	-13,7%	4,4	-30,5%	2,0	≈	-31,2%	1,7
	12	12	7090	20,9%	6,8	>	16,3%	5,3	-23,7%	2,3	>	-25,0%	2,3
	12	15	7325	40,1%	8,7	>	35,0%	9,4	39,7%	8,1	>	-23,0%	2,1
	15	9	6979	2,2%	3,5	>	-2,2%	4,8	-19,3%	2,5	≈	-20,2%	2,0
	15	12	7358	39,8%	8,4	≈	38,2%	10,7	41,4%	7,4	>	-10,2%	12,3
15	15	7551	51,5%	7,8	≈	49,5%	8,9	53,4%	9,6	>	-16,7%	4,7	
Instance p08	9	9	3467	23,1%	6,4	>	19,1%	7,1	1,0%	2,2	≈	0,2%	1,7
	9	12	3978	37,6%	6,7	≈	37,6%	7,7	2,2%	1,9	≈	2,0%	1,6
	9	15	3726	70,3%	7,1	≈	69,6%	8,7	12,1%	2,7	≈	10,8%	2,3
	12	9	3901	38,1%	5,3	≈	36,1%	5,9	8,3%	1,8	≈	7,4%	1,5
	12	12	4305	50,2%	5,8	>	47,0%	5,8	7,6%	1,5	≈	7,4%	2,2
	12	15	4225	81,1%	6,9	≈	79,4%	8,0	81,9%	6,6	>	12,8%	2,4
	15	9	4656	30,3%	4,8	>	27,5%	6,1	-0,3%	1,4	≈	-0,4%	1,8
	15	12	5042	46,1%	5,7	≈	43,7%	5,9	45,5%	6,0	>	0,3%	1,2
15	15	4909	72,8%	5,1	>	67,6%	6,5	73,9%	6,9	>	7,6%	1,8	
Instance p09	9	9	3319	44,2%	5,7	>	40,2%	7,4	23,0%	3,3	≈	22,9%	4,5
	9	12	3522	43,5%	7,6	≈	40,7%	7,8	15,0%	3,2	≈	13,8%	3,1
	9	15	4906	37,5%	5,8	≈	36,4%	5,0	6,6%	1,6	≈	6,4%	2,3
	12	9	3506	37,0%	6,6	>	27,7%	9,1	19,8%	5,0	≈	19,0%	5,5
	12	12	3706	44,4%	6,0	>	39,1%	11,2	13,0%	2,7	≈	11,8%	3,5
	12	15	4922	37,8%	4,3	≈	36,6%	5,4	8,1%	2,5	≈	7,9%	1,8
	15	9	4460	45,2%	6,7	>	39,9%	7,2	22,8%	2,8	>	20,4%	3,0
	15	12	4690	46,9%	4,7	>	42,5%	3,9	14,9%	2,6	≈	14,0%	2,9
15	15	6171	39,8%	3,3	>	36,4%	3,9	38,2%	3,6	>	5,9%	1,4	
Instance p10	9	9	3979	38,3%	5,7	>	32,5%	5,6	16,0%	3,2	≈	15,0%	2,8
	9	12	6496	12,1%	2,8	≈	12,0%	2,8	-0,3%	1,0	≈	-0,6%	1,1
	9	15	7821	20,9%	2,5	>	19,4%	2,7	4,5%	1,4	≈	4,0%	1,2
	12	9	3535	51,5%	7,6	≈	46,7%	10,5	20,1%	3,6	≈	19,7%	3,7
	12	12	5708	24,9%	3,5	>	21,1%	2,9	5,7%	1,5	≈	5,1%	1,8
	12	15	7138	26,5%	2,3	≈	25,1%	4,2	7,0%	1,4	≈	6,9%	1,3
	15	9	5190	38,9%	5,3	>	32,1%	5,7	10,5%	2,2	≈	9,3%	2,8
	15	12	7183	19,3%	3,0	>	16,1%	2,5	2,0%	1,7	≈	1,5%	1,0
15	15	8356	26,2%	2,5	≈	25,1%	2,5	6,4%	1,1	>	5,6%	1,3	

On some instances the solution archive seemed to improve the solution value, e.g., instance *p04* 9×9 , *p06* 9×15 and *p06* 12×9 (using the intertwined VNS in each case) of Table 7.3, even without the VNS at the end, but the Student's t-test revealed that they

are actually equal.

Table 7.4: The mean percentage gaps of 30 runs with the given configurations and with the VNS at the end. The column description indicates the use of the HVREA configuration with archive (+A) or without and with intertwined VNS (+V) or without. The entries in column p indicate whether the result with the solution archive or the result without the archive is better ($>$) or equal (\approx) according to a Student's t-test with an error level of 5%

	x	y	orig	HVR+A			HVR		HVR+A+V			HVR+V	
				gap	dev	p	gap	dev	gap	dev	p	gap	dev
Instance p01	9	9	2094	5,1%	10,6	$>$	0,0%	0,0	1,3%	4,1	\approx	0,0%	0,0
	9	12	3142	25,3%	5,4	\approx	22,6%	7,9	16,9%	3,9	\approx	14,8%	5,1
	9	15	3223	29,1%	6,0	\approx	27,4%	5,4	17,2%	6,9	\approx	14,8%	7,5
	12	9	2907	24,7%	5,7	$>$	18,5%	11,8	13,8%	8,4	\approx	12,9%	8,2
	12	12	3695	24,3%	5,6	\approx	22,1%	4,4	14,4%	2,9	\approx	14,3%	2,3
	12	15	3825	27,7%	4,8	$>$	24,9%	4,6	17,3%	2,8	\approx	17,6%	2,8
	15	9	2931	18,2%	13,7	\approx	16,9%	17,1	18,4%	10,6	$>$	11,1%	12,5
	15	12	3732	25,9%	5,4	\approx	24,6%	6,1	17,2%	3,0	\approx	15,8%	4,7
	15	15	3870	31,6%	4,4	\approx	30,3%	4,7	19,8%	3,1	\approx	20,3%	2,8
Instance p02	9	9	1434	-16,7%	9,4	$>$	-21,5%	8,0	-25,8%	2,9	\approx	-26,4%	3,8
	9	12	1060	17,5%	8,5	$>$	10,1%	11,0	8,7%	5,2	$>$	4,6%	3,8
	9	15	1978	5,0%	3,8	$>$	1,3%	4,1	-5,0%	2,6	\approx	-5,7%	3,1
	12	9	1396	-6,6%	7,8	$>$	-11,1%	8,4	-18,0%	5,3	\approx	-20,7%	5,4
	12	12	1083	17,9%	13,1	\approx	12,1%	11,5	9,8%	5,1	$>$	4,1%	5,4
	12	15	1904	5,6%	5,3	\approx	5,0%	7,6	-2,6%	3,2	\approx	-3,2%	2,8
	15	9	1658	-0,4%	6,1	$>$	-7,4%	6,5	-9,7%	5,0	\approx	-11,7%	4,0
	15	12	1503	17,1%	10,0	$>$	10,2%	10,4	6,3%	6,5	$>$	2,4%	4,3
	15	15	2283	10,0%	4,4	$>$	4,4%	6,0	1,1%	3,3	$>$	-0,6%	3,0
Instance p03	9	9	2486	10,5%	10,4	$>$	4,5%	11,6	2,1%	6,9	\approx	1,2%	8,1
	9	12	2651	23,0%	10,2	\approx	18,5%	12,0	13,6%	6,0	$>$	10,0%	7,8
	9	15	2551	7,4%	9,3	$>$	3,3%	5,5	4,2%	4,5	\approx	2,2%	3,3
	12	9	3075	16,2%	6,4	$>$	12,2%	5,9	9,7%	3,7	\approx	9,3%	4,2
	12	12	3377	22,3%	5,3	$>$	19,1%	6,2	12,1%	4,0	\approx	10,4%	4,7
	12	15	3313	12,3%	8,7	$>$	6,4%	8,0	3,8%	4,8	$>$	1,2%	4,0
	15	9	3213	16,5%	4,9	\approx	16,1%	6,8	7,7%	4,4	\approx	6,8%	3,9
	15	12	3278	29,1%	6,7	\approx	26,1%	9,5	21,9%	4,3	$>$	18,3%	5,8
	15	15	3308	18,2%	8,6	$>$	9,1%	8,1	9,6%	4,9	$>$	6,6%	6,1
Instance p04	9	9	1104	11,4%	14,7	$>$	-0,9%	13,9	-11,2%	13,0	\approx	-11,3%	14,1
	9	12	1463	6,4%	10,2	\approx	1,4%	11,0	-0,6%	6,3	\approx	-3,8%	8,0
	9	15	1589	-5,8%	6,5	$>$	-11,2%	7,3	-14,9%	4,4	\approx	-16,0%	4,7
	12	9	1515	24,8%	11,2	\approx	19,9%	14,1	8,4%	10,1	$>$	3,1%	7,7
	12	12	2051	16,4%	3,7	$>$	12,7%	6,3	5,1%	3,1	$>$	3,0%	4,7
	12	15	2146	-2,3%	5,1	$>$	-6,0%	5,8	-9,8%	3,3	$>$	-11,9%	3,0
	15	9	1567	25,0%	11,2	\approx	21,3%	13,9	5,9%	8,3	\approx	5,8%	7,6
	15	12	1752	31,8%	9,4	$>$	23,8%	7,8	17,6%	6,7	$>$	13,6%	7,3
	15	15	2026	2,7%	5,6	$>$	-0,4%	5,8	-3,5%	4,0	\approx	-4,8%	4,5
Instance p05	9	9	690	1,7%	5,1	\approx	0,0%	0,1	0,4%	2,1	\approx	0,0%	0,0
	9	12	888	66,0%	27,2	$>$	45,3%	30,0	33,7%	18,8	$>$	21,3%	15,9
	9	15	1623	37,3%	10,8	$>$	26,5%	12,0	19,4%	5,2	\approx	18,0%	5,9
	12	9	1016	18,8%	14,6	$>$	9,1%	14,4	3,5%	6,4	\approx	2,9%	6,1
	12	12	1325	35,4%	15,2	\approx	28,5%	16,9	12,6%	8,4	$>$	6,7%	10,2
	12	15	1986	38,5%	8,5	$>$	30,0%	12,1	21,7%	4,9	\approx	19,1%	5,8
	15	9	1010	-10,3%	12,4	$>$	-16,5%	7,6	-17,0%	4,1	\approx	-18,4%	2,9
	15	12	1156	40,5%	26,3	$>$	17,6%	20,7	14,8%	12,6	\approx	13,5%	15,0
	15	15	1900	34,5%	7,5	$>$	26,6%	9,2	14,9%	5,2	\approx	12,9%	5,0

	x	y	orig	HVR+A			HVR			HVR+A+V			HVR+V	
				gap	dev	p	gap	dev	gap	dev	p	gap	dev	
Instance p06	9	9	2184	7,7%	4,9	≈	7,7%	4,9	-1,2%	2,3	≈	-2,5%	2,7	
	9	12	2915	5,1%	4,5	≈	3,1%	5,2	-2,8%	2,9	≈	-3,2%	3,3	
	9	15	2265	30,8%	6,8	≈	25,9%	8,1	15,0%	4,9	≈	16,4%	3,4	
	12	9	2162	19,8%	5,9	>	16,4%	5,1	6,3%	3,3	≈	6,9%	3,7	
	12	12	3031	14,8%	3,4	≈	13,3%	3,8	6,1%	2,2	≈	6,1%	2,8	
	12	15	2401	37,6%	8,7	>	32,4%	7,7	22,7%	4,6	≈	22,4%	4,8	
	15	9	2719	8,3%	3,3	≈	6,9%	4,9	0,8%	2,9	≈	0,3%	2,8	
	15	15	3452	7,7%	3,6	>	4,6%	4,5	0,6%	3,1	≈	0,1%	2,8	
			2928	27,3%	4,8	>	24,4%	6,2	16,6%	4,7	≈	15,0%	4,0	
Instance p07	9	9	6461	-19,5%	3,8	≈	-19,6%	3,0	-23,8%	2,6	≈	-24,1%	2,4	
	9	12	6856	-9,3%	5,0	>	-12,1%	5,2	-15,2%	7,0	≈	-17,0%	5,4	
	9	15	6952	-1,9%	7,8	≈	-1,7%	9,3	-11,2%	9,9	≈	-11,3%	9,5	
	12	9	6758	-26,9%	3,1	≈	-27,6%	2,9	-30,6%	2,0	≈	-31,4%	1,7	
	12	12	7090	-17,0%	3,5	≈	-17,4%	4,5	-24,4%	2,4	>	-25,9%	2,4	
	12	15	7325	-15,4%	3,4	≈	-17,0%	3,7	-23,1%	2,7	≈	-23,4%	2,3	
	15	9	6979	-15,9%	2,4	≈	-16,9%	2,4	-19,6%	2,4	≈	-20,5%	1,9	
	15	15	7358	-7,6%	7,3	≈	-6,1%	7,5	-14,1%	6,6	≈	-11,0%	12,3	
			7551	-7,6%	7,6	≈	-8,6%	5,5	-15,0%	4,8	≈	-17,2%	4,7	
Instance p08	9	9	3467	7,8%	4,4	≈	7,1%	4,1	0,6%	2,2	≈	-0,2%	1,8	
	9	12	3978	9,2%	3,9	≈	9,3%	2,4	1,9%	1,8	≈	1,5%	1,6	
	9	15	3726	21,7%	5,4	≈	20,4%	3,9	11,7%	2,7	≈	10,5%	2,2	
	12	9	3901	14,2%	3,2	≈	14,4%	2,8	7,8%	1,7	≈	7,2%	1,6	
	12	12	4305	15,4%	3,2	≈	14,1%	2,2	7,3%	1,5	≈	7,0%	2,2	
	12	15	4225	22,3%	3,9	≈	22,0%	3,4	13,0%	2,2	≈	12,3%	2,4	
	15	9	4656	4,3%	2,6	≈	4,5%	2,4	-0,9%	1,5	≈	-0,9%	1,8	
	15	15	5042	6,5%	3,2	≈	5,2%	3,1	0,3%	1,3	≈	0,0%	1,3	
			4909	15,6%	3,9	≈	14,3%	3,5	7,5%	1,6	≈	7,3%	1,9	
Instance p09	9	9	3319	34,3%	5,3	≈	32,2%	6,0	22,7%	3,1	≈	22,6%	4,5	
	9	12	3522	24,7%	6,1	≈	22,5%	5,5	14,2%	3,4	≈	13,2%	3,1	
	9	15	4906	13,6%	3,8	≈	13,7%	3,5	6,4%	1,6	≈	6,0%	2,4	
	12	9	3506	28,8%	6,2	>	22,1%	6,9	19,5%	5,0	≈	18,9%	5,6	
	12	12	3706	21,3%	4,4	≈	19,6%	6,4	12,6%	2,6	≈	11,2%	3,6	
	12	15	4922	14,2%	3,1	≈	14,0%	2,6	7,8%	2,5	≈	7,7%	1,8	
	15	9	4460	28,9%	3,8	≈	27,1%	5,0	22,1%	2,7	>	20,1%	2,9	
	15	15	4690	23,5%	3,5	≈	22,0%	3,4	14,4%	2,5	≈	13,4%	2,8	
			6171	13,2%	2,8	≈	12,1%	2,7	6,0%	2,0	≈	5,7%	1,4	
Instance p10	9	9	3979	22,1%	4,6	≈	22,0%	4,2	15,6%	3,2	≈	14,1%	2,6	
	9	12	6496	4,0%	1,9	≈	4,2%	1,6	-0,7%	1,1	≈	-0,9%	1,1	
	9	15	7821	8,5%	1,5	>	7,3%	1,7	4,2%	1,5	≈	3,9%	1,2	
	12	9	3535	31,0%	7,0	≈	29,8%	7,2	19,3%	3,9	≈	18,3%	4,3	
	12	12	5708	11,9%	2,7	>	10,5%	1,8	5,3%	1,6	≈	4,8%	1,8	
	12	15	7138	11,8%	1,7	≈	11,3%	2,0	6,8%	1,3	≈	6,5%	1,3	
	15	9	5190	16,8%	4,8	≈	16,3%	4,9	9,8%	2,5	≈	8,6%	3,0	
	15	15	7183	6,5%	2,3	≈	6,3%	2,1	1,7%	1,8	≈	1,1%	1,1	
			8356	9,9%	1,8	≈	9,6%	1,6	6,2%	1,2	>	5,4%	1,4	

In Table 7.4 the same configurations of the HVREA is tested but in the end a VNS with all of the described neighborhoods is made. Note that on some instances the VNS improves the solution value of the HVREA using the solution archive a lot more than it

does when no archive is used, e.g., the solution value before the VNS is performed of instance *p06* 15×15 is 85.5% compared to 15.4% (using the intertwined VNS) when the archive is not used, so the HVREA without the archive is clearly better. After the VNS the solution value drastically decreased from 85.5% to 16.6% in contrast to the HVREA without the archive, which result only dropped to 15%. The Student's t-test showed that the results after the VNS performed are even equal.

In Table 7.5 and 7.6 the same configurations were tested but instead of the HVREA the BNREA was used. Again, the results of the BNREA with the archive is compared to the BNREA without the archive.

Table 7.5: The mean percentage gaps of 30 runs with the given configurations and **without** the VNS at the end. The column description indicates the use of the BNREA configuration with archive (+A) or without and with intertwined VNS (+V) or without. The entries in column *p* indicate whether the result with the solution archive or the result without the archive is better (>) or equal(\approx) according to a Student's t-test with an error level of 5%

			BNR+A			BNR			BNR+A+V			BNR+V	
	x	y	orig	gap	dev	p	gap	dev	gap	dev	p	gap	dev
Instance p01	9	9	2094	17,2%	19,7	\approx	14,2%	18,2	5,3%	10,1	\approx	4,5%	10,0
	9	12	3142	60,1%	9,4	>	54,0%	12,5	20,6%	4,0	\approx	19,6%	4,2
	9	15	3223	76,1%	12,2	>	63,0%	15,8	21,7%	4,9	\approx	20,5%	5,5
	12	9	2907	55,9%	9,9	\approx	51,6%	8,2	23,8%	6,1	\approx	21,7%	5,1
	12	12	3695	66,2%	7,0	>	57,0%	6,8	18,1%	3,8	\approx	17,1%	2,4
	12	15	3825	88,7%	6,1	>	75,4%	7,7	19,6%	2,5	\approx	19,9%	2,7
	15	9	2931	58,9%	9,1	>	51,2%	9,8	27,2%	5,7	\approx	27,3%	6,0
	15	12	3732	74,0%	7,6	>	62,5%	9,4	22,5%	3,3	\approx	21,7%	3,2
	15	15	3870	86,6%	7,0	>	72,3%	11,2	24,7%	3,1	\approx	25,6%	6,3
Instance p02	9	9	1434	29,2%	18,9	\approx	23,4%	21,5	-21,0%	3,8	\approx	-21,0%	4,4
	9	12	1060	109,1%	37,3	\approx	109,2%	31,5	13,1%	4,2	\approx	12,7%	3,3
	9	15	1978	59,2%	14,0	\approx	56,9%	15,8	-1,9%	2,3	\approx	-1,9%	2,3
	12	9	1396	20,6%	15,6	\approx	24,5%	15,2	-13,6%	3,6	\approx	-14,1%	3,4
	12	12	1083	77,7%	24,7	\approx	79,7%	28,7	14,1%	3,6	\approx	12,8%	4,4
	12	15	1904	54,3%	18,0	>	42,2%	14,3	-0,3%	3,1	\approx	5,9%	22,9
	15	9	1658	37,2%	10,2	>	22,4%	7,7	-2,9%	3,9	\approx	-2,6%	4,3
	15	12	1503	67,9%	10,4	>	43,6%	11,8	13,0%	4,5	\approx	11,4%	5,8
	15	15	2283	40,3%	6,3	>	22,9%	8,8	3,1%	2,6	\approx	2,4%	2,2
Instance p03	9	9	2486	25,5%	14,4	\approx	21,8%	10,7	4,7%	7,2	\approx	2,5%	4,8
	9	12	2651	59,1%	13,2	>	46,7%	11,3	18,3%	3,8	\approx	18,5%	5,0
	9	15	2551	45,5%	13,3	>	37,5%	11,6	10,5%	4,0	\approx	11,0%	4,0
	12	9	3075	34,3%	7,7	\approx	30,6%	8,3	11,7%	3,4	\approx	11,1%	3,0
	12	12	3377	51,5%	9,6	>	44,6%	11,4	14,9%	3,0	\approx	14,7%	2,5
	12	15	3313	61,4%	10,5	>	46,3%	8,4	9,7%	2,8	\approx	9,3%	3,3
	15	9	3213	49,9%	8,8	>	38,6%	7,3	11,2%	4,2	\approx	9,2%	5,2
	15	12	3278	84,1%	8,0	>	69,6%	11,9	21,3%	5,0	\approx	22,6%	4,3
	15	15	3308	74,8%	6,1	>	59,0%	11,1	13,4%	4,1	\approx	13,8%	3,3
Instance p04	9	9	1104	37,0%	20,8	\approx	33,5%	11,5	-9,9%	11,7	\approx	-9,3%	12,3
	9	12	1463	39,8%	12,1	>	28,3%	13,6	1,5%	5,5	\approx	1,6%	5,8
	9	15	1589	16,4%	12,1	>	8,1%	10,4	-10,2%	3,7	>	-12,4%	3,3
	12	9	1515	64,0%	12,2	>	50,8%	10,3	13,4%	7,6	\approx	10,1%	7,8
	12	12	2051	39,6%	13,2	>	29,8%	6,3	14,0%	3,0	>	12,2%	3,0
	12	15	2146	21,1%	7,0	>	11,4%	7,9	-3,0%	2,7	\approx	-4,2%	3,7

	x	y	orig	BNR+A			BNR		BNR+A+V			BNR+V	
				gap	dev	p	gap	dev	gap	dev	p	gap	dev
	15	9	1567	55,6%	16,3	>	47,5%	13,4	14,0%	6,5	≈	11,4%	5,4
	15	12	1752	55,5%	12,4	>	46,5%	10,2	22,4%	3,7	>	19,2%	4,8
	15	15	2026	28,6%	10,1	>	19,6%	7,9	-0,8%	3,0	>	-2,6%	3,6
Instance p05	9	9	690	21,7%	30,7	≈	12,5%	26,7	1,1%	2,7	≈	0,2%	0,8
	9	12	888	128,4%	20,9	≈	117,9%	26,7	46,9%	14,5	≈	46,0%	14,5
	9	15	1623	91,9%	23,4	>	77,7%	16,6	23,2%	6,4	≈	22,8%	5,4
	12	9	1016	56,0%	18,7	>	46,5%	15,8	18,1%	9,9	≈	15,7%	10,2
	12	12	1325	90,2%	14,2	>	74,9%	14,9	24,6%	7,0	≈	24,3%	7,5
	12	15	1986	132,2%	20,9	>	98,4%	24,0	28,2%	2,9	>	26,5%	3,0
	15	9	1010	24,8%	21,5	>	5,5%	23,8	-10,9%	6,4	>	-14,9%	6,8
	15	12	1156	116,8%	21,3	≈	108,4%	20,7	32,7%	8,2	≈	33,1%	8,3
	15	15	1900	145,3%	15,5	>	98,0%	15,3	20,6%	4,7	≈	20,2%	3,9
Instance p06	9	9	2184	49,8%	10,5	>	40,0%	16,6	2,3%	3,4	≈	1,7%	3,2
	9	12	2915	62,0%	13,8	>	44,7%	16,2	-1,0%	2,0	≈	-1,3%	1,7
	9	15	2265	135,0%	15,6	>	108,4%	16,4	17,8%	3,9	≈	17,7%	4,0
	12	9	2162	90,6%	16,2	>	61,8%	14,8	11,8%	4,0	≈	12,3%	3,5
	12	12	3031	101,0%	14,9	>	72,4%	18,8	7,4%	2,5	≈	8,3%	2,4
	12	15	2401	154,5%	15,7	>	127,2%	18,6	29,4%	4,2	≈	28,8%	4,2
	15	9	2719	80,7%	9,4	>	54,4%	12,9	6,7%	2,3	≈	7,1%	2,3
	15	12	3452	108,1%	5,2	>	94,6%	19,1	107,7%	7,7	>	4,2%	1,6
	15	15	2928	161,9%	7,3	>	126,7%	11,1	158,3%	10,3	>	21,9%	3,7
Instance p07	9	9	6461	0,0%	5,1	>	-4,1%	6,3	-23,2%	2,6	≈	-23,7%	2,4
	9	12	6856	30,6%	11,3	>	11,0%	8,5	-16,7%	2,5	≈	-17,0%	2,9
	9	15	6952	45,5%	11,5	>	30,2%	9,0	-16,9%	3,8	≈	-17,7%	3,1
	12	9	6758	10,2%	4,8	>	-3,0%	6,7	-29,4%	2,2	≈	-28,5%	2,4
	12	12	7090	41,1%	7,5	>	25,6%	10,1	-22,2%	2,8	≈	-22,3%	2,6
	12	15	7325	65,9%	9,2	>	47,1%	12,0	66,1%	9,6	>	-19,5%	2,4
	15	9	6979	21,2%	4,3	>	11,3%	5,6	-18,1%	1,9	≈	-18,6%	2,1
	15	12	7358	56,9%	9,1	>	35,0%	8,3	57,7%	9,3	>	-15,9%	3,8
	15	15	7551	79,5%	8,4	>	57,2%	9,2	78,3%	7,8	≈	76,0%	8,3
Instance p08	9	9	3467	47,0%	9,2	≈	42,6%	14,1	1,7%	2,3	≈	1,2%	2,4
	9	12	3978	64,5%	10,3	>	48,2%	12,8	2,8%	1,5	≈	2,8%	1,5
	9	15	3726	113,7%	11,7	>	91,6%	13,2	12,3%	2,0	≈	12,0%	2,1
	12	9	3901	78,6%	7,2	>	58,3%	9,4	11,8%	2,1	>	10,6%	2,3
	12	12	4305	88,1%	6,6	>	70,0%	11,5	8,2%	1,8	≈	8,3%	1,4
	12	15	4225	133,3%	7,5	>	113,2%	9,9	133,9%	8,5	>	14,2%	2,1
	15	9	4656	63,0%	5,8	>	49,0%	11,8	0,5%	1,4	>	-0,1%	1,2
	15	12	5042	83,7%	6,1	>	66,1%	7,7	85,7%	5,5	>	2,3%	1,8
	15	15	4909	127,8%	7,5	>	109,2%	10,1	129,8%	6,7	≈	129,2%	7,9
Instance p09	9	9	3319	83,9%	11,2	>	75,4%	10,7	30,5%	3,3	≈	30,7%	3,2
	9	12	3522	99,3%	10,4	>	82,2%	10,8	18,9%	4,7	≈	19,0%	4,0
	9	15	4906	67,8%	6,6	>	52,2%	10,7	8,8%	2,2	≈	8,4%	2,3
	12	9	3506	65,7%	10,4	>	50,9%	5,8	21,5%	3,3	≈	19,9%	3,9
	12	12	3706	89,2%	9,4	>	64,4%	8,8	12,3%	3,1	≈	12,3%	3,4
	12	15	4922	77,6%	7,0	>	59,8%	8,2	6,6%	2,4	≈	6,5%	2,6
	15	9	4460	81,8%	7,8	>	70,7%	9,7	27,3%	2,3	≈	27,7%	2,3
	15	12	4690	83,7%	4,9	>	76,1%	6,7	21,2%	2,7	≈	20,2%	2,3
	15	15	6171	74,9%	4,5	>	64,3%	6,6	73,9%	4,6	>	7,1%	1,4
Instance p10	9	9	3979	74,3%	9,7	>	65,4%	9,3	18,2%	2,7	≈	17,5%	2,4
	9	12	6496	34,6%	5,4	>	24,0%	5,1	0,7%	0,8	≈	0,4%	1,2
	9	15	7821	47,0%	2,6	>	40,9%	3,2	7,0%	1,4	≈	6,4%	1,7
	12	9	3535	94,3%	9,8	>	79,7%	11,7	23,4%	3,9	≈	23,7%	3,4

x	y	orig	BNR+A			BNR			BNR+A+V			BNR+V	
			gap	dev	p	gap	dev	gap	dev	p	gap	dev	
12	12	5708	46,1%	4,9	>	34,6%	4,9	6,4%	1,6	≈	6,4%	1,3	
12	15	7138	56,7%	4,3	>	48,0%	6,3	9,0%	1,4	≈	8,5%	1,3	
15	9	5190	76,4%	7,0	>	64,3%	9,2	14,6%	2,3	≈	14,6%	2,7	
15	12	7183	46,0%	4,5	>	36,3%	5,0	5,4%	1,1	≈	5,0%	1,5	
15	15	8356	57,4%	2,9	>	51,0%	4,1	57,4%	2,6	>	7,9%	1,3	

Table 7.6: The mean percentage gaps of 30 runs with the given configurations and **with** the VNS at the end. The column description indicates the use of the BNREA configuration with archive (+A) or without and with intertwined VNS (+V) or without. The entries in column p indicate whether the result with the solution archive or the result without the archive is better (>) or equal(≈) according to a Student's t-test with an error level of 5%

	x	y	orig	BNR+A			BNR			BNR+A+V			BNR+V	
				gap	dev	p	gap	dev	gap	dev	p	gap	dev	
Instance p01	9	9	2094	8,7%	13,5	≈	8,6%	13,1	5,3%	10,1	≈	4,5%	10,0	
	9	12	3142	30,9%	6,2	≈	30,4%	7,3	20,2%	3,7	≈	18,5%	4,2	
	9	15	3223	37,6%	6,7	≈	34,5%	5,2	21,4%	4,9	≈	20,0%	5,4	
	12	9	2907	36,0%	8,3	≈	37,7%	7,7	23,2%	6,1	≈	21,5%	5,1	
	12	12	3695	27,3%	5,2	≈	27,1%	5,0	17,4%	4,0	≈	16,3%	2,6	
	12	15	3825	29,9%	6,0	≈	30,9%	4,7	19,2%	2,4	≈	19,1%	3,0	
	15	9	2931	37,3%	7,7	≈	36,0%	6,9	26,1%	5,7	≈	26,2%	5,6	
	15	12	3732	31,0%	5,5	≈	29,5%	5,9	21,5%	3,7	≈	20,1%	3,3	
	15	15	3870	35,1%	5,4	≈	36,0%	5,5	24,3%	3,3	≈	23,9%	3,4	
Instance p02	9	9	1434	-3,6%	9,0	≈	-4,2%	11,1	-21,2%	3,7	≈	-21,2%	4,5	
	9	12	1060	29,7%	11,9	≈	34,1%	16,1	12,2%	4,2	≈	11,8%	3,5	
	9	15	1978	6,8%	4,8	≈	6,1%	5,0	-2,7%	2,4	≈	-2,2%	2,4	
	12	9	1396	0,0%	7,6	≈	-3,3%	9,3	-14,7%	3,4	≈	-14,5%	3,4	
	12	12	1083	32,1%	10,2	≈	30,2%	9,4	13,7%	3,6	>	11,2%	4,4	
	12	15	1904	7,7%	5,5	≈	9,3%	5,1	-1,0%	3,3	≈	-1,3%	3,1	
	15	9	1658	15,5%	9,1	>	10,7%	5,1	-3,3%	4,0	≈	-3,3%	4,3	
	15	12	1503	31,4%	9,1	>	25,3%	8,5	11,4%	4,1	≈	10,4%	5,7	
	15	15	2283	13,4%	4,9	≈	11,6%	4,3	2,6%	2,6	≈	1,6%	1,9	
Instance p03	9	9	2486	15,5%	8,6	≈	14,5%	8,0	4,1%	7,1	≈	1,8%	5,0	
	9	12	2651	31,5%	6,1	≈	30,3%	8,4	17,6%	4,2	≈	18,0%	4,9	
	9	15	2551	22,3%	10,1	≈	21,8%	7,7	10,1%	4,0	≈	10,4%	4,0	
	12	9	3075	19,0%	5,1	≈	17,6%	5,2	11,0%	3,1	≈	10,5%	3,3	
	12	12	3377	25,7%	5,5	≈	26,0%	5,1	14,2%	3,0	≈	14,0%	2,9	
	12	15	3313	21,8%	5,4	≈	21,3%	6,9	8,4%	3,2	≈	8,3%	3,6	
	15	9	3213	25,0%	5,9	≈	24,6%	7,5	10,9%	4,2	>	8,3%	5,0	
	15	12	3278	33,1%	7,1	≈	33,9%	5,4	20,4%	5,1	≈	22,0%	4,5	
	15	15	3308	24,5%	6,8	≈	23,7%	8,6	12,5%	4,2	≈	12,3%	3,2	
Instance p04	9	9	1104	22,3%	14,6	≈	24,5%	13,0	-10,2%	11,7	≈	-10,9%	12,2	
	9	12	1463	26,4%	7,4	>	17,0%	9,4	0,3%	4,8	≈	1,0%	6,4	
	9	15	1589	3,8%	7,2	>	-0,3%	6,4	-10,8%	3,9	>	-12,9%	3,6	
	12	9	1515	34,2%	10,2	≈	33,2%	12,1	12,1%	8,0	≈	8,8%	7,9	
	12	12	2051	23,3%	6,0	>	20,0%	5,6	13,5%	3,2	>	11,2%	3,2	
	12	15	2146	6,0%	4,9	>	2,3%	4,8	-3,7%	3,1	≈	-4,8%	3,6	
	15	9	1567	28,6%	11,8	≈	30,0%	11,8	12,8%	6,4	≈	10,2%	5,8	
	15	12	1752	38,0%	8,9	>	33,3%	7,7	21,2%	3,9	>	18,7%	4,7	
	15	15	2026	10,1%	5,3	≈	7,9%	5,2	-1,1%	3,0	>	-3,0%	3,2	

	x	y	orig	BNR+A			BNR			BNR+A+V			BNR+V	
				gap	dev	p	gap	dev	gap	dev	p	gap	dev	
Instance p05	9	9	690	13,2%	20,7	≈	8,2%	17,5	1,0%	2,7	≈	0,2%	0,6	
	9	12	888	85,2%	16,3	≈	79,4%	17,1	46,4%	14,4	≈	43,6%	13,9	
	9	15	1623	45,4%	10,5	>	40,2%	9,3	22,7%	6,7	≈	21,8%	5,1	
	12	9	1016	35,5%	14,3	≈	34,3%	14,7	17,5%	9,7	≈	15,1%	10,2	
	12	12	1325	42,8%	9,1	≈	40,9%	11,7	23,4%	7,1	≈	23,2%	6,6	
	12	15	1986	41,4%	6,7	≈	38,7%	8,4	27,4%	2,7	≈	26,0%	3,2	
	15	9	1010	4,7%	13,9	≈	-3,0%	17,1	-11,2%	6,4	>	-15,6%	5,9	
	15	12	1156	56,6%	10,5	≈	55,2%	14,1	30,1%	10,2	≈	31,1%	8,7	
15	15	1900	38,0%	9,6	≈	35,8%	5,7	20,0%	5,0	≈	19,4%	3,8		
Instance p06	9	9	2184	13,5%	6,1	≈	13,9%	6,0	1,7%	3,4	≈	1,2%	3,3	
	9	12	2915	9,4%	5,2	≈	9,4%	3,6	-1,3%	2,3	≈	-1,7%	2,0	
	9	15	2265	33,4%	6,2	≈	34,8%	7,1	17,1%	3,9	≈	16,7%	4,2	
	12	9	2162	26,5%	5,7	≈	23,7%	7,3	11,2%	4,0	≈	11,8%	3,7	
	12	12	3031	18,1%	3,8	≈	17,5%	4,1	7,0%	2,6	≈	8,1%	2,5	
	12	15	2401	44,7%	6,6	≈	43,1%	7,6	28,9%	4,4	≈	28,1%	4,1	
	15	9	2719	16,2%	4,5	≈	15,5%	5,6	5,9%	2,7	≈	6,3%	2,4	
	15	12	3452	13,9%	4,2	≈	12,7%	4,5	4,1%	1,6	≈	3,9%	1,6	
15	15	2928	36,2%	5,0	≈	34,4%	5,7	20,5%	4,0	≈	21,3%	3,8		
Instance p07	9	9	6461	-18,0%	3,0	≈	-17,4%	3,2	-23,4%	2,6	≈	-23,9%	2,5	
	9	12	6856	-8,3%	5,2	≈	-10,4%	4,8	-17,2%	2,5	≈	-17,9%	2,8	
	9	15	6952	-8,6%	4,8	≈	-8,3%	5,7	-17,5%	3,9	≈	-18,0%	3,0	
	12	9	6758	-24,0%	3,5	≈	-23,3%	3,8	-29,6%	2,3	≈	-28,8%	2,5	
	12	12	7090	-13,8%	6,3	≈	-15,1%	4,8	-23,0%	2,9	≈	-23,4%	2,9	
	12	15	7325	-10,2%	5,1	≈	-12,3%	5,6	-21,2%	3,6	≈	-20,2%	2,5	
	15	9	6979	-14,0%	2,5	≈	-14,6%	3,0	-18,3%	1,9	≈	-18,9%	2,1	
	15	12	7358	-8,1%	3,7	>	-11,3%	4,4	-17,0%	2,6	≈	-16,7%	3,9	
15	15	7551	-7,5%	4,4	≈	-8,3%	4,5	-6,0%	4,4	>	-14,9%	2,9		
Instance p08	9	9	3467	10,0%	4,7	≈	9,8%	4,0	0,8%	2,8	≈	0,3%	2,1	
	9	12	3978	10,1%	4,0	≈	9,7%	3,2	2,1%	1,9	≈	1,9%	1,8	
	9	15	3726	20,8%	5,0	≈	23,1%	5,2	11,9%	1,8	≈	11,6%	2,0	
	12	9	3901	19,2%	4,8	≈	19,3%	4,3	11,2%	2,2	≈	10,1%	2,4	
	12	12	4305	16,9%	4,4	≈	16,4%	2,7	7,8%	2,0	≈	7,7%	1,5	
	12	15	4225	24,3%	5,7	≈	23,9%	4,9	13,9%	2,3	≈	13,7%	2,3	
	15	9	4656	6,6%	2,9	≈	6,3%	2,9	0,0%	1,5	>	-0,8%	1,4	
	15	12	5042	9,2%	3,9	≈	8,8%	3,4	1,4%	1,5	≈	1,8%	1,9	
15	15	4909	15,6%	5,5	≈	17,1%	3,8	17,2%	3,6	>	8,5%	1,9		
Instance p09	9	9	3319	44,6%	6,4	>	39,8%	6,4	29,9%	3,6	≈	30,0%	3,4	
	9	12	3522	33,0%	7,0	≈	33,6%	7,3	18,0%	4,5	≈	18,4%	3,9	
	9	15	4906	20,1%	5,6	>	16,7%	4,3	8,5%	2,3	≈	8,2%	2,3	
	12	9	3506	33,4%	6,7	≈	31,3%	5,9	20,7%	3,3	≈	19,2%	4,1	
	12	12	3706	26,5%	7,1	≈	24,2%	5,9	11,7%	3,1	≈	11,5%	4,1	
	12	15	4922	16,6%	3,5	≈	16,6%	4,4	6,3%	2,3	≈	5,9%	2,5	
	15	9	4460	36,4%	4,0	≈	35,8%	5,6	26,9%	2,4	≈	27,2%	2,3	
	15	12	4690	29,8%	5,0	≈	29,0%	4,4	20,8%	2,6	≈	19,8%	2,3	
15	15	6171	14,4%	3,5	≈	15,3%	3,4	6,6%	1,6	≈	6,7%	1,4		

			BNR+A			BNR			BNR+A+V			BNR+V		
	x	y	orig	gap	dev	p	gap	dev	gap	dev	p	gap	dev	
Instance p10	9	9	3979	29,4%	5,0	>	25,9%	5,9	17,5%	3,0	≈	16,8%	2,7	
	9	12	6496	5,6%	3,3	≈	5,8%	2,3	0,3%	0,8	≈	0,1%	1,2	
	9	15	7821	12,2%	2,6	≈	12,0%	2,6	6,8%	1,5	≈	6,1%	1,6	
	12	9	3535	37,8%	8,0	≈	34,7%	6,6	22,0%	4,1	≈	23,1%	3,6	
	12	12	5708	12,7%	3,2	≈	12,5%	2,2	6,1%	1,6	≈	6,1%	1,4	
	12	15	7138	14,3%	2,7	≈	14,2%	3,2	8,8%	1,5	≈	8,3%	1,2	
	15	9	5190	25,1%	5,9	≈	23,4%	5,1	14,1%	2,3	≈	13,7%	3,0	
	15	12	7183	11,4%	3,4	≈	10,4%	3,0	5,1%	1,1	≈	4,8%	1,4	
	15	15	8356	13,9%	2,8	≈	12,8%	2,1	7,3%	1,1	≈	7,8%	1,3	

The results of the BNREA show that when using the BNREA on more instances the results of the BNREA with the solution archive are equal to the results of the BNREA alone. This shows that the BNREA benefits more from the solution archive than does the HVREA. It can also be seen that on both the HVREA and the BNREA configurations the intertwined VNS improves the solution value on all instances significantly.

A Student's t-test with an error level of 5% was performed to compare the mean solution values with the different configurations. On some instances the MA that uses the archive performed better than the MA alone but in all these cases the Student's t-test unfortunately revealed that they are equal. The other way round, i.e., when the MA alone is compared with the MA which is enhanced by the solution archive, it also turned out that most results were equal according to the Student's t-test but on some instances the MA alone performed even statistically better.

Another result is, when both types of MA (with and without using the solution archive) are run the same number of iterations the MA with the solution archive performed better, but due to the longer running time of the archive these results are not directly comparable. Nevertheless, this encourages the assumption that the MA with the archive will converge in later generations as does the MA without the archive. It would be interesting to see the results when running the MA with the solution archive for a longer time, i.e., without the memory restrictions of the used test system.

Conclusions

Within this work an MA for the Reconstruction of Cross-Cut Shredded Text Documents (RCCSTD) problem was extended with a trie based solution archive. Its purpose is to store already visited solutions and thus avoids costs for reevaluating them and generate new solutions when a duplicate is found. Using this technique gives the population of the MA a higher diversity but some elitism is still conserved.

First the problem is formally defined, an error estimation function is described and the complexity of the problem is briefly discussed. In the next chapter an overview of current literature concerning both the RCCSTD problem and solution archives is given. Then, in Chapter 4 an overview of some heuristic solution techniques is provided, where among others memetic algorithms are presented. Chapter 5 describes the design and development of the solution archive. In this chapter two different approaches are compared to each other and it turned out that one of them, namely the Shred-Based Permutation (SBP) approach is superior over the Trie-Based Permutation (TBP) approach. Additionally, other problems, which have arisen, e.g., how to avoid invalid solutions, are discussed and solutions for these problems presented.

Chapter 6 deals with implementation issues, especially the memory problem. In this chapter also the recombination and the mutation operators of the underlying MA are described. Finally, in Chapter 7 the tests are performed and the results presented and compared. To be more precise, the results of different configurations of the MA without the archive are compared to the same settings of the MA that is enhanced by the implemented solution archive.


It turned out that although the MA using a solution archive is usually able to find better results than the MA without the archive within the same number of generations. But the

running time overhead of the archive is too high to really improve the performance of the MA. When both settings are given the same amount of time, the MA in combination with the archive cannot outperform the MA anymore. Instead, on some instances the MA alone performs even better without the archive.

8.1 Future Work

An interesting attempt to improve the solution archive would be to increase the running time bound of the MA with the archive. Since the solution archive for the RCCSTD problem needs a huge amount of memory, each run can only run between 1 and 5 minutes before the machine runs out of memory. Therefore, one could improve the memory efficiency of the trie with a branch and bound algorithm, as presented in [6]. Since the objective value of a solution cannot decrease when adding a shred to the solution many branches, especially the ones that are at the bottom half of the trie, can be cut. This probably would decrease the memory consumption of the trie significantly and this would maybe have an impact on the results, because the running time could then be increased.

A. Instances




ELSEVIER

Available online at www.sciencedirect.com

SCIENCE @ DIRECT®

BioSystems 72 (2003) 75–97



Bio Systems

www.elsevier.com/locate/biosystems

A memetic-aided approach to hierarchical clustering from distance matrices: application to gene expression clustering and phylogeny

Carlos Cotta^{a,*}, Pablo Moscato^b

^a *Dept. Lenguajes y Ciencias de la Computación, Universidad de Málaga ETSI Informática (3.2.49),
Campus de Teatinos, 29071 Malaga, Spain*

^b *Faculty of Engineering and Built Environment, School of Electrical Engineering and Computer Science,
The University of Newcastle, Callaghan, 2308 NSW, Australia*

Abstract

We propose a heuristic approach to hierarchical clustering from distance matrices based on the use of memetic algorithms (MAs). By using MAs to solve some variants of the Minimum Weight Hamiltonian Path Problem on the input matrix, a sequence of the individual elements to be clustered (referred to as patterns) is first obtained. While this problem is also NP-hard, a probably optimal sequence is easy to find with the current advances for this problem and helps to prune the space of possible solutions and/or to guide the search performed by an actual clustering algorithm. This technique has been successfully applied to both a Branch-and-Bound algorithm, and to evolutionary algorithms and MAs. Experimental results are given in the context of phylogenetic inference and in the hierarchical clustering of gene expression data.

© 2003 Elsevier Ireland Ltd. All rights reserved.

Keywords: Hierarchical clustering; Memetic algorithms; Phylogenetic inference; Gene expression; Data mining

Figure A.1: Instance p06

Vorwort

Die rasante Entwicklung im Bereich der Informations- und Kommunikationstechnologie hat in den letzten Jahren zu einem enormen Bedarf an universitär gut ausgebildeten Arbeitskräften in verschiedensten Bereichen der Wirtschaft, speziell im Raum der EU, geführt. Die Studienkommission Informatik am Standort Wien reagierte im Jahre 2001 auf den akuten Bedarf insbesondere an universitär ausgebildeten Informatikfachkräften durch eine Umstellung des Diplomstudiums Informatik auf dreijährige Bachelor- und darauf aufbauende zweijährige Masterstudien. Diese Gliederung entspricht auch dem Geist der Bologna-Erklärung, in welcher der Wille zu einer derartigen EU-weiten Entwicklung der Studienpläne bekundet wurde. Kürzere Normstudienzeiten erhöhen die Attraktivität der Studien für MaturantInnen und senken die Rate der StudienabbrecherInnen.

Das Ausbildungsniveau der AbsolventInnen der Masterstudien entspricht jener der AbsolventInnen des bis 2001 geführten Diplomstudiums Informatik. Die nun noch besseren Möglichkeiten zur Spezialisierung sowie die vielfältigen Kombinationsmöglichkeiten der Bachelorstudien (auch anderer Studienrichtungen) mit den angebotenen Masterstudien der Informatik orientieren sich an den stetig wachsenden Anforderungen der Wirtschaft an flexiblen, interdisziplinär ausgebildeten AkademikerInnen.

Die Bachelorstudien *Data Engineering & Statistics*, *Medieninformatik*, *Medizinische Informatik*, *Software & Information Engineering* sowie *Technische Informatik* vermitteln eine fundierte Grundlagenausbildung mit Schwerpunktsetzungen, die sowohl den klassischen Bereichen der Informatik (Software & Information Engineering, Technische Informatik) als auch aktuellen Trends (Data Engineering & Statistics, Medieninformatik, Medizinische Informatik) Rechnung tragen.

Die Masterstudien *Computational Intelligence*, *Computergraphik & Digitale Bildverarbeitung*, *Information & Knowledge Management*, *Medieninformatik*, *Medizinische Informatik*, *Software Engineering & Internet Computing*, *Technische Informatik* sowie *Wirtschaftsingenieurwesen Informatik* führen zu einer Vertiefung und Spezialisierung in relevanten Gebieten der Informatik. Die AbsolventInnen sind sowohl für höhere Positionen in der Wirtschaft als auch für weiterführende Forschungsaufgaben hoch qualifiziert. Die weit gefassten Zulassungsbedingungen erhöhen die Möglichkeiten, in verschiedenen Anwendungsgebieten Schlüsselqualifikationen zu erwerben. Das Spektrum reicht von der Möglichkeit für ElektrotechnikerInnen, das Masterstudium der Technischen Informatik zu absolvieren, bis hin zum Angebot des Masterstudiums Wirtschaftsingenieurwesen Informatik für AbsolventInnen von Ingenieurfächern; überdies stehen alle Masterstudien für AbsolventInnen des Studiums der Wirtschaftsinformatik offen.

5. Software & Information Engineering	32
5.1. Präambel	32
5.2. Qualifikationsprofil der Absolventinnen und Absolventen	32
5.3. Prüfungsfächer	33
5.4. Semesterempfehlung	35
6. Technische Informatik	37
6.1. Präambel	37
6.2. Qualifikationsprofil der Absolventinnen und Absolventen	38
6.3. Prüfungsfächer	38
6.4. Semesterempfehlung	40
II. Masterstudien	42
7. Allgemeine Regelungen	43
7.1. Studien und akademischer Grad	43
7.2. ECTS-Punkte und Semesterstunden	43
7.3. Prüfungsfächer	43
7.4. Freie Wahlfächer und Soft Skills	44
7.5. Masterarbeit (Diplomarbeit)	44
7.6. Voraussetzungen für die Absolvierung von Lehrveranstaltungen	45
7.7. Studienplanentsprechung von Lehrveranstaltungen	45
7.8. Lehrveranstaltungssprache	46
7.9. Erweiterung der Lehrveranstaltungskataloge	46
7.10. Prüfungsordnung	46
8. Computational Intelligence	48
8.1. Präambel	48
8.2. Qualifikationsprofil der AbsolventInnen	48
8.3. Studienvoraussetzungen	49
8.4. Prüfungsfächer und Diplomarbeit	49
8.5. Lehrveranstaltungskatalog	49
9. Computergraphik & Digitale Bildverarbeitung	53
9.1. Präambel	53
9.2. Qualifikationsprofil der AbsolventInnen	53
9.3. Studienvoraussetzungen	54
9.4. Prüfungsfächer und Diplomarbeit	54
9.5. Lehrveranstaltungskatalog	55
10. Information & Knowledge Management	58
10.1. Präambel	58
10.2. Qualifikationsprofil der AbsolventInnen	58
10.3. Studienvoraussetzungen	59

Figure A.3: Instance p02

3. Medieninformatik

3.1. Präambel

Das Studium *Medieninformatik* versteht sich als spezielle anwendungsorientierte Informatik, die die Bereiche Design, Computergraphik, Bildverarbeitung und Multimedia – kurz: die zunehmende Auseinandersetzung mit dem Begriff des *Visuellen* – in den Mittelpunkt stellt. Diese Bereiche entwickelten in den letzten Jahren in und außerhalb der Informatik eine starke Dynamik, die die Lehrinhalte beeinflusst und neue Berufsfelder erschließt. Ihre kompetente Bearbeitung verlangt nicht nur eine andere Gewichtung und informatikinterne Ausweitung der traditionellen Studieninhalte, sondern auch die Ergänzung um Themen aus dem Bereich Design.

Im Mittelpunkt der Medieninformatik steht der Umgang mit dem Visuellen, vornehmlich mit Bildern, bildhaften Darstellungen und graphischen Symbolen, der in allen Aspekten studiert wird, und zwar unter besonderer Berücksichtigung der Verwendung von Computern. Genau aus diesem Grund auch wird der Studiengang auf Initiative der Informatik vorangetrieben.

Multimedia und ihre Anwendungen gelten als ein wichtiger Zukunftsbereich in der Informatik. Aufgaben wie die Präsentation von Informationen mit unterschiedlichen Medien, die Gestaltung der interaktiven Schnittstellen und die Navigation durch virtuelle Welten stellen derart hohe Qualifikationsansprüche an zukünftige MedieninformatikerInnen, dass die Einrichtung eines eigenen Studiums dafür unbedingt notwendig ist.

Hierzu wird als Kern des Studienganges eine solide Grundausbildung in der Informatik angeboten, mit einer Spezialisierung auf visuelle Themen wie Design, Computergraphik, Bildverarbeitung und Mustererkennung. Hier sind Gebiete wie die technische Bildaufnahme, Bildvorverarbeitung, Bildauswertung und automatische Bildinterpretation vertreten, aber auch neben Bildwiedergabe und Bildkommunikation alle Aspekte der Bildsynthese, der virtuellen Realität und der wissenschaftlichen Visualisierung.

3.2. Qualifikationsprofil der Absolventinnen und Absolventen

Das Studium soll eine wissenschaftlich geprägte Ausbildung vermitteln, die Theorie, Fachwissen und praktische Kenntnisse von Medientechnik, Computergraphik, der digitalen Bildverarbeitung und Mustererkennung einschließt. Es soll die Studierenden in die Lage versetzen, Methoden und Werkzeuge aus den oben genannten Gebieten zu verstehen, anzuwenden sowie sich eigenständig an ihrer Erforschung und Weiterentwicklung zu beteiligen. Studienziel ist weiters die Vermittlung von Wissen um die kreative Gestaltung der Medien und deren Produktionsprozess. Dazu gehört die Befähigung der Auszu-

Informatik und Gesellschaft (12.0 Ects)

3.0/2.0 VU Daten- und Informatikrecht
3.0/2.0 VU Gesellschaftliche Spannungsfelder der Informatik
3.0/2.0 VU Gesellschaftswissenschaftliche Grundlagen der Informatik
3.0/2.0 SE Grundlagen methodischen Arbeitens

Grundlagen der Informatik (12.0 Ects)

6.0/4.0 VO Einführung in die Technische Informatik
6.0/4.0 VU Grundzüge der Informatik

Medizinische Informatik (24.0 Ects)

3.0/2.0 VO Biometrie und Epidemiologie
3.0/2.0 VO Einführung in die Medizinische Informatik
3.0/2.0 VU Einführung in wissensbasierte Systeme
3.0/2.0 VO Grundlagen der digitalen Bildverarbeitung
3.0/2.0 VO Grundlagen und Praxis der medizinischen Versorgung
3.0/2.0 VO Informationssysteme des Gesundheitswesens
6.0/4.0 SE Seminar (mit Bachelorarbeit)

Medizinische Grundlagen (27.0 Ects)

4.5/3.0 VD Anatomie und Histologie
3.0/2.0 VO Biochemie
3.0/2.0 VU Biosignalverarbeitung
1.5/1.0 VD Chemie-Propädeutikum
3.0/2.0 VU Grundlagen bioelektrischer Systeme
4.5/3.0 VU Grundlagen der Physik
3.0/2.0 PR Physikalisches Praktikum
4.5/3.0 VD Physiologie und Grundlagen der Pathologie

Programmierung und Datenmodellierung (24.0 Ects)

6.0/4.0 VL Algorithmen und Datenstrukturen 1
3.0/2.0 VO Algorithmen und Datenstrukturen 2
3.0/2.0 VL Datenmodellierung
6.0/4.0 VL Einführung in das Programmieren
3.0/2.0 VU Objektorientierte Modellierung
3.0/2.0 VL Objektorientierte Programmierung

Herkunftsstudium	Basismodule (à 18.0 Ects)			Vertiefungsmodule (à 6.0 Ects)			
	Ing.wiss.	Wirtschaft	Informatik	Ing.wiss.	Wirtschaft	Informatik	beliebig
Informatik	1	1	0	0 oder 1	1	2 oder 1	0
Wirtschaftsinformatik	1	0	0	1	1	2	2
Ingenieurwiss.	0	1	1	0 oder 1	1	2 oder 1	0
Wirtschafts-ing.wes. MB	0	0	1	1	1	2	2
Wirtschaft	1	0	1	0 oder 1	1	2 oder 1	0

Für Herkunftsstudien, die nicht von diesem Schema erfasst werden, kann das studienrechtliche Organ Basismodule festlegen.

Im Rahmen der Vertiefungsmodule sind Seminare im Umfang von 3.0 Ects bis 6.0 Ects zu absolvieren.

Freie Wahlfächer und Soft Skills (9.0 Ects)

Siehe Abschnitt 7.4.

Diplomarbeit (30.0 Ects)

Siehe Abschnitt 7.5.

15.5. Basis- und Vertiefungsmodule

Basismodul Informatik

Es sind Lehrveranstaltungen im Umfang von 18.0 Ects aus den folgenden Lehrveranstaltungen zu wählen. Die Kenntnisse der Lehrveranstaltungen dieses Moduls werden als Voraussetzung für das Verständnis der Vertiefungsmodule aus Informatik erwartet.

6.0/4.0 VL Algorithmen und Datenstrukturen 1

3.0/2.0 VL Datenmodellierung

6.0/4.0 VO Einführung in die Technische Informatik

3.0/2.0 VU Objektorientierte Modellierung

3.0/2.0 VL Objektorientierte Programmierung

3.0/2.0 VO Software Engineering und Projektmanagement

6.0/4.0 LU Software Engineering und Projektmanagement

6.0/4.0 VU Theoretische Informatik und Logik

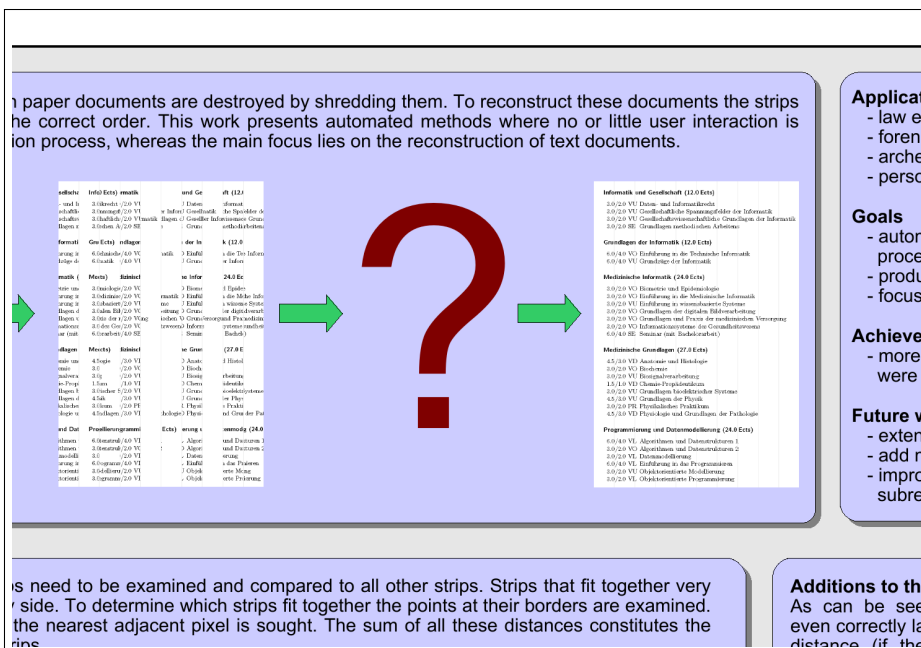


Figure A.7: Instance p07

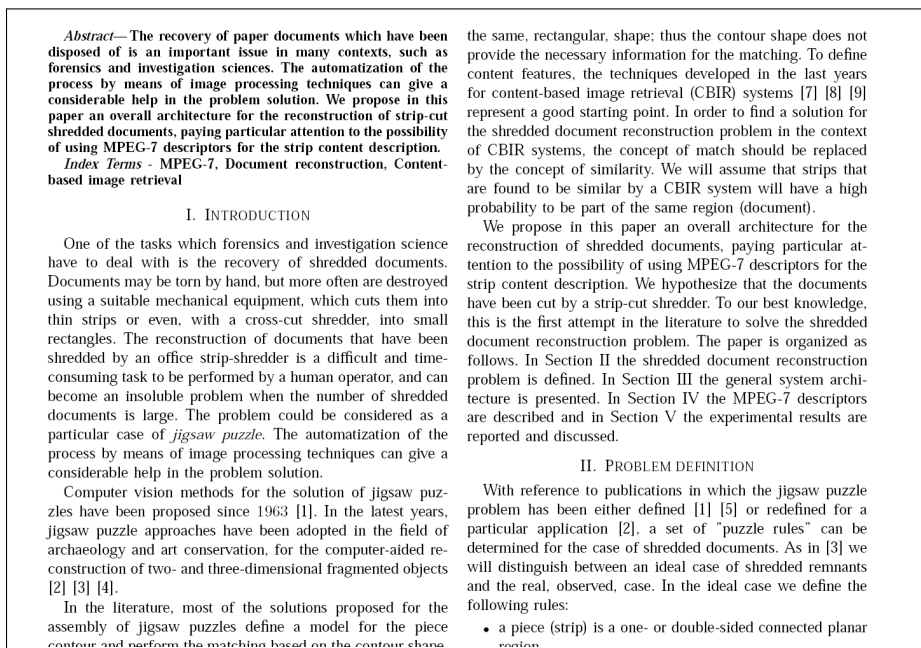


Figure A.8: Instance p08

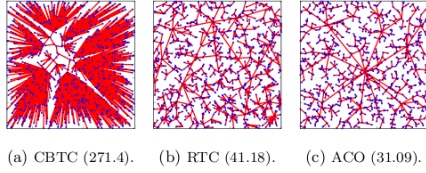


Figure 1: A diameter constrained tree with $D = 10$ constructed using (a) the CBTC heuristic, compared to (b) RTC (best solution from 1000 runs) and (c) a solution obtained by an ant colony optimization approach (complete, Euclidean graph with 1000 nodes distributed randomly in the unit square, the corresponding objective values are given in parentheses).

of relatively short edges and the majority of the nodes have to be connected to this backbone via rather long edges, see the example in Fig. 1(a). On the contrary, a reasonable solution for this instance, shown in Fig. 1(c), demonstrates that the backbone should consist of a few longer edges to span the whole area to allow the large number of remaining nodes to be connected as leaves by much cheaper edges. In a pure greedy construction heuristic this observation is difficult to realize. In the *randomized tree construction approach* (RTC, Fig. 1(b)) from [13] not the cheapest of all nodes is always added to the partial spanning tree but the next node is chosen at random and then connected by the cheapest feasible edge. Thus at least the possibility to include longer edges into the backbone at the beginning of the algorithm is increased. For Euclidean instances RTC has been so far the best choice to quickly create a first solution as basis for exact or metaheuristic approaches.

In the following we will introduce a new construction heuristic for the BDMST problem which is especially suited for very large Euclidean instances. It is based on a hierarchical clustering that guides the algorithm to find a good backbone. This approach is then refined by a local improvement method and extended towards a greedy randomized adaptive search procedure (GRASP) [17]. A preliminary version of this work can be found in [10].

3. THE CLUSTERING HEURISTIC

The clustering-based construction heuristic can be divided into three steps: Creating a hierarchical clustering (*dendrogram*) of all instance nodes based on the edge costs, deriving a height-restricted clustering (HRC) from this dendrogram, and finding for each cluster in the HRC a good root (center) node.

3.1 Hierarchical Clustering

For the purpose of creating a good backbone especially for an Euclidean instance of the BDMST problem agglomerative hierarchical clustering seems to provide a good guidance. To get spatially confined areas, two clusters A and B are merged when $\max\{c_{a,b} : a \in A, b \in B\}$ is minimal over all pairs of clusters (complete linkage clustering [12]).

The agglomeration starts with each node being an individual cluster, and stops when all nodes are merged within one

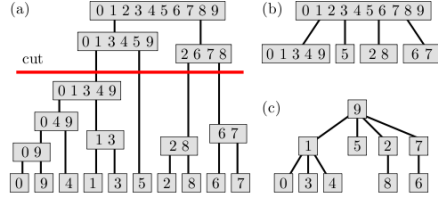


Figure 2: Hierarchical clustering (a), height-restricted clustering (b), and the resulting diameter constrained tree with $D = 4$ (c) after choosing a root for each cluster in (b).

single cluster. The resulting hierarchical clustering can be illustrated as a binary tree, also referred to as a *dendrogram*, with $|V|$ leaves and $|V| - 1$ inner nodes each representing one merging operation during clustering; see Fig. 2(a) for an example with $|V| = 10$. An inner node's distance from the leaves indicates when the two corresponding clusters – relative to each other – have been merged.

3.2 Height-Restricted Clustering

After performing the agglomerative hierarchical clustering, the resulting dendrogram is transformed into a height-restricted clustering (HRC) for the BDMST, i.e. into a representation of the clustering respecting the diameter and thus the height condition. The dendrogram itself cannot directly act as HRC since in general it will violate this constraint, see Fig. 2(a). Therefore, some of the nodes in the dendrogram have to be merged to finally get a tree of height $H - 1$, the HRC for the BDMST; see Fig. 2(b) for a diameter of $D = 4$.

For the quality of the resulting tree this merging of dendrogram nodes is a crucial step, worth significant effort. It can be described by $H - 1$ cuts through the dendrogram defining which nodes of it will also become part of the height-restricted clustering and which are merged with their parent clusters. As an example, starting at the root containing all instance nodes agglomerated within one single cluster the cut illustrated in Fig. 2(a) defines the dendrogram nodes $\{0, 1, 3, 4, 9\}$, $\{5\}$, $\{2, 8\}$, and $\{6, 7\}$ to become direct successors of the root cluster in the height-restricted clustering.

One fundamental question arising in this context is the way of defining the cutting positions through the dendrogram. After preliminary tests, the identification of the precise iteration at which two clusters have been merged in the agglomeration process turns out to be a good criterion. This *merge number* (or *merge#*), which allows a fine-grained control of the cutting positions, can be stored within each node of the dendrogram, with the leaves having a merge number of zero and the root $|V| - 1$.

Based on the merge numbers cutting positions ς are computed as

$$\varsigma_i = (|V| - 1) - 2^{i \cdot \frac{\log_2 x}{H-1}} \quad i = 1, \dots, H-1, \quad (1)$$

where x is a strategy parameter. This formula is motivated by a perfectly balanced tree, where parameter x can be interpreted as the number of nodes that shall form the backbone.

These cutting positions can now be used to build the height-restricted clustering for the BDMST, as depicted in

Figure A.9: Instance p09

Table 1: Averaged objective values over all 15 Euclidean Steiner tree instances of Beasley’s OR-Library with 1000 nodes for various diameter bounds and (meta-)heuristics, the standard deviations are given parentheses. In addition, the averaged maximum running times of the clustering heuristics that were used as time limit for CBTC and RTC are listed.

D	without VND					with VND				
	CBTC	RTC	Cd ^A	Cd ^B	t _{max} (C) [s]	RTC	Cd ^B	ACO	t _{max} (C) [s]	
4	329.3021 (6.02)	146.4919 (3.88)	68.3241 (0.72)	68.3226 (0.70)	2.54 (0.09)	65.2061 (0.55)	65.1598 (0.56)	65.8010 (0.48)	5.56 (1.01)	
6	306.2655 (9.02)	80.8636 (2.40)	47.4045 (4.85)	47.1702 (4.61)	4.55 (0.49)	41.4577 (0.36)	41.3127 (0.50)	42.1167 (0.26)	9.94 (1.52)	
8	288.3842 (7.52)	53.2535 (1.33)	37.0706 (1.35)	36.9408 (1.34)	5.92 (0.42)	35.0511 (0.35)	34.2171 (0.29)	34.7489 (0.23)	11.61 (1.61)	
10	266.3665 (9.01)	41.1201 (0.68)	33.5460 (0.67)	33.3408 (0.66)	6.79 (0.42)	32.1181 (0.31)	30.9704 (0.24)	31.0388 (0.20)	13.43 (2.16)	
12	250.0016 (8.01)	35.7590 (0.47)	32.2571 (0.48)	31.9561 (0.44)	7.11 (0.33)	30.2897 (0.29)	29.1796 (0.26)	28.6356 (0.23)	14.68 (2.49)	
14	237.1403 (6.28)	33.3644 (0.30)	31.3790 (0.37)	31.0176 (0.33)	7.00 (0.64)	29.0940 (0.28)	28.0093 (0.23)	26.6524 (0.32)	15.05 (3.00)	
16	224.3123 (5.72)	32.1965 (0.24)	30.7937 (0.33)	30.4287 (0.29)	7.20 (0.72)	28.2433 (0.28)	27.1363 (0.19)	25.5760 (0.19)	15.63 (2.89)	
18	210.9872 (7.63)	31.5826 (0.24)	30.5182 (0.29)	30.1348 (0.27)	7.32 (0.81)	27.6008 (0.27)	26.5601 (0.20)	24.8811 (0.16)	16.78 (3.61)	
20	197.1772 (7.99)	31.2682 (0.22)	30.3116 (0.31)	30.0384 (0.28)	7.57 (0.76)	27.1091 (0.26)	26.1079 (0.23)	24.3698 (0.15)	18.54 (3.89)	
22	183.0157 (8.03)	31.0864 (0.22)	30.2344 (0.30)	30.0739 (0.28)	8.56 (0.98)	26.6984 (0.28)	25.8048 (0.21)	24.0129 (0.17)	21.39 (5.19)	
24	172.8251 (10.59)	30.9921 (0.23)	30.0202 (0.23)	30.1603 (0.27)	8.28 (1.41)	26.3648 (0.27)	25.4523 (0.24)	23.7723 (0.20)	21.36 (6.42)	
5	241.3032 (5.09)	117.3238 (2.22)	62.2867 (0.76)	62.0646 (0.67)	24.59 (2.02)	58.9883 (0.53)	58.7930 (0.56)	59.5964 (0.49)	30.82 (3.28)	
7	222.1441 (4.50)	67.7577 (1.31)	46.7291 (3.92)	46.4112 (3.73)	27.94 (1.79)	39.4703 (0.34)	39.3817 (0.46)	39.9948 (0.25)	38.79 (4.03)	
9	204.6141 (6.00)	47.3168 (0.85)	37.0224 (1.25)	36.8904 (1.27)	18.27 (1.68)	33.9677 (0.30)	33.2142 (0.25)	33.5907 (0.23)	32.51 (4.88)	
11	189.7513 (4.62)	38.4754 (0.50)	33.4140 (0.70)	33.1749 (0.66)	13.97 (0.71)	31.3661 (0.29)	30.3683 (0.20)	30.2701 (0.19)	29.47 (4.70)	
13	175.7382 (4.23)	34.5154 (0.32)	32.1094 (0.43)	31.8041 (0.41)	12.79 (1.17)	29.7644 (0.28)	28.7554 (0.21)	28.1224 (0.20)	29.94 (6.28)	
15	163.1926 (4.31)	32.7069 (0.25)	31.2654 (0.35)	30.8941 (0.32)	11.03 (1.27)	28.6966 (0.26)	27.6899 (0.20)	26.3893 (0.25)	28.54 (6.29)	
17	149.9852 (5.14)	31.8467 (0.23)	30.7699 (0.33)	30.3664 (0.30)	8.93 (0.94)	27.9309 (0.27)	26.9097 (0.19)	25.3794 (0.23)	28.47 (6.19)	
19	139.9730 (4.32)	31.4048 (0.21)	30.5350 (0.29)	30.0837 (0.27)	7.91 (1.08)	27.3691 (0.26)	26.3784 (0.20)	24.7705 (0.18)	29.67 (7.37)	
21	128.1830 (4.90)	31.1697 (0.23)	30.3017 (0.30)	30.0384 (0.27)	7.60 (0.71)	26.9015 (0.26)	25.9415 (0.20)	24.3128 (0.18)	30.05 (6.74)	
23	119.5551 (4.46)	31.0421 (0.22)	30.0627 (0.24)	30.1166 (0.31)	6.96 (0.81)	26.5346 (0.27)	25.6021 (0.21)	23.9719 (0.21)	28.55 (7.05)	
25	110.6725 (4.39)	30.9772 (0.23)	29.9450 (0.21)	30.1393 (0.24)	6.68 (0.89)	26.2126 (0.26)	25.2289 (0.21)	23.7773 (0.25)	25.59 (6.02)	

which sub-cluster to choose can be based on the same criterion as in the agglomeration process the choice which clusters to merge, i.e. a root node is assigned to the sub-cluster where the maximum distance to a node of it is minimal.

7. COMPUTATIONAL RESULTS

The experiments have been performed on an AMD Opteron 2214 dual-core machine (2.2GHz) utilizing benchmark instances already used in the corresponding literature (e.g. [15, 11]) from Beasley’s OR-Library [3, 2] originally proposed for the Euclidean Steiner tree problem. These complete instances contain point coordinates in the unit square, and the Euclidean distances between each pair of points are taken as edge costs. As performance differences are more significant on larger instances, we restrict our attention here to the 15 largest instances with 1000 nodes.

Table 1 summarizes the results obtained for various heuristics. Given are the objective values averaged over all 15 instances, where for each instance 30 independent runs have been performed, together with the standard deviations in parentheses. Considered are the two previous construction heuristics CBTC and RTC as well as the clustering heuristic C where each cluster a good root node is assigned using one of the two dynamic programming approaches d^A (restricted search space) and d^B (approximating optimal cluster roots using a correction value κ). Since d^A and d^B derive no optimal trees for a given clustering local improvement is used to further enhance their solutions.

Binary search to identify a good value for x was performed within $\frac{|V|}{2}$ and $|V|$, except when $D < 6$. In this latter case the interval bounds have been set to $\frac{|V|}{20}$ and $\frac{|V|}{8}$. In GRASP a mean μ of 0 and, after preliminary tests, a variance σ^2 of 0.25 was used, and the procedure was aborted after $l_{\max} =$

100 iterations without improvement. The time (in seconds) listed is the over all instances averaged maximum running time of Cd^A and Cd^B , which was also used as time limit for the corresponding executions of CBTC and RTC. To verify statistical significance paired Wilcoxon signed rank tests have been performed.

Clearly, CBTC is not suitable for this type of instances, its strength lies in problems with random edge costs. The clustering heuristic outperforms RTC for every diameter bound, where the gap in solution quality is huge when D is small and becomes less with increasing diameter bound. In general, Cd^B outperforms all other heuristics significantly with an error probability of less than $2.2 \cdot 10^{-16}$. Only when the diameter bound gets noticeably loose the first dynamic programming approach Cd^A dominates Cd^B (error probability always less than $2.13 \cdot 10^{-9}$). It can also be seen that in the even-diameter case the runtime of the clustering heuristic only increases moderately with the number of levels in the height-restricted clustering. When D is odd the search for a good center edge dominates the runtime. Thus, with increasing D the clustering heuristics even get faster since the number of potential center edges to be considered, determined in the presented preprocessing step, decreases (less direct successors of the root cluster in the HRC).

We also performed test where a strong variable neighborhood descend (VND) as proposed in [11] has been applied to the best solutions of the various construction heuristics. As expected, it flattens the differences but still the BDMSTs derived from clustering heuristic solutions are in general of higher quality. On instances with diameter bounds less than approximately 10, these trees – computed in a few seconds – can also compete with results from the leading metaheuristic, the ACO from [11], which requires computation times of one hour and more.

Figure A.10: Instance p10

B. Generations

Table B.1: Number of generations per instance

Instance	Size	Generations	Instance	Size	Generations
p01	9x9	10000	p06	9x9	10000
	9x12	10000		9x12	5000
	9x15	10000		9x15	5000
	12x9	10000		12x9	10000
	12x12	10000		12x12	5000
	12x15	10000		12x15	5000
	15x9	10000		15x9	5000
	15x12	10000		15x12	5000
	15x15	5000		15x15	3000
p02	9x9	10000	p07	9x9	10000
	9x12	10000		9x12	5000
	9x15	10000		9x15	5000
	12x9	10000		12x9	5000
	12x12	10000		12x12	5000
	12x15	5000		12x15	3000
	15x9	10000		15x9	5000
	15x12	10000		15x12	3000
	15x15	5000		15x15	3000
p03	9x9	10000	p08	9x9	10000
	9x12	10000		9x12	5000
	9x15	10000		9x15	5000
	12x9	10000		12x9	5000
	12x12	10000		12x12	5000
	12x15	10000		12x15	3000
	15x9	10000		15x9	5000
	15x12	10000		15x12	3000
	15x15	5000		15x15	3000

Table B.1: Number of generations per instance

Instance	Size	Generations	Instance	Size	Generations
p04	9x9	10000	p09	9x9	10000
	9x12	10000		9x12	10000
	9x15	10000		9x15	5000
	12x9	10000		12x9	10000
	12x12	10000		12x12	5000
	12x15	10000		12x15	5000
	15x9	10000		15x9	5000
	15x12	10000		15x12	5000
	15x15	10000		15x15	3000
p05	9x9	10000	p10	9x9	10000
	9x12	10000		9x12	10000
	9x15	10000		9x15	5000
	12x9	10000		12x9	10000
	12x12	10000		12x12	5000
	12x15	10000		12x15	5000
	15x9	10000		15x9	5000
	15x12	10000		15x12	5000
	15x15	5000		15x15	5000

Bibliography

- [1] D. Applegate, W. J. Cook, and A. Rohe. Chained lin-kernighan for large traveling salesman problems. *INFORMS Journal on Computing*, pages 82–92, 2003.
- [2] A. M. Chwatal and S. Pirkwieser. Solving the Two-Dimensional Bin-Packing Problem with Variable Bin Sizes by Greedy Randomized Adaptive Search Procedures and Variable Neighborhood Search. In *Computer Aided Systems Theory - EUROCAST 2011: 13th International Conference*, pages 392–399, 2012.
- [3] A. Colomi, M. Dorigo, V. Maniezzo, and Others. Distributed optimization by ant colonies. In *Proceedings of the first European conference on artificial life*, volume 142, pages 134–142, 1991.
- [4] DARPA. DARPA Shredder Challenge. <http://archive.darpa.mil/shredderchallenge/Default.aspx>, 2011. Accessed: 20/02/2012.
- [5] M. Dorigo, M. Birattari, and T. Stützle. Ant Colony Optimization: Artificial Ants as a Computational Intelligence Technique. *IEEE Computational Intelligence Magazine*, 1(4):28–39, 2006.
- [6] C. Gruber. Ein Lösungsarchiv mit für das Generalized Minimum Spanning Tree Problem. Master’s thesis, Vienna University of Technology, 2011.
- [7] P. Hansen and N. Mladenovi. Variable neighborhood search : Principles and applications. *European Journal Of Operational Research*, 130(3):1097–1100, 2001.
- [8] J. H. Holland. *Adaptation in natural and artificial systems*. MIT Press, Cambridge, MA, USA, 1992.
- [9] B. Hu and G. R. Raidl. An Evolutionary Algorithm with Solution Archive for the Generalized Minimum Spanning Tree Problem. In *Extended Abstracts of EUROCAST 2011 13th International Conference on Computer Aided Systems Theory*, pages 256–259, 2011.
- [10] E. Justino, L. S. Oliveira, and C. Freitas. Reconstructing shredded documents through feature matching. *Forensic science international*, 160(2-3):140–7, July 2006.

- [11] F. Kleber, M. Diem, and R. Sablatnig. Document reconstruction by layout analysis of snippets. volume 7531. *Proceedings of SPIE - The International Society for Optical Engineering*, 2010. art no. 753107.
- [12] B. Koch. The stasi puzzle. *Fraunhofer magazine*, pages 32–33, 1 2008.
- [13] M. Leitner and R. Raidl. Variable Neighborhood and Greedy Randomized Adaptive Search for Capacitated Connected Facility Location. In *Proceedings of the 13th International Conference on Computer Aided Systems Theory: Part I*, pages 295–302, 2011.
- [14] M.-H. Lim. *Memetic Computing Journal*. <http://www.springer.com/engineering/computational+intelligence+and+complexity/journal/12293>, 2009. Accessed: 24/02/2012.
- [15] H.-y. Lin and W.-C. Fan-Chiang. Image-Based Techniques for Shredded Document Reconstruction. In *Proceedings of the 3rd Pacific Rim Symposium on Advances in Image and Video Technology*, pages 155–166. Springer-Verlag, 2008.
- [16] A. A. Low. Waste paper receptacle. Patent, 08 1909. US 929960.
- [17] M. L. Mauldin. Maintaining Diversity in Genetic Search. In *National Conference on Artificial Intelligence*, volume 19, pages 247–250. AAAI, William Kaufmann, 1984.
- [18] P. Moscato and M. G. Norman. A memetic approach for the traveling salesman problem implementation of a computational ecology for combinatorial optimization on message-passing systems. *Parallel Computing and Transputer Applications*, 28(1):177–186, 1992.
- [19] K. P. Rane and S. G. Bhirud. Text Reconstruction using Torn Document Mosaicing. *International Journal of Computer Applications*, 30(10):21–27, Sept. 2011.
- [20] A. Pimenta, E. Justino, L. S. Oliveira, and R. Sabourin. Document reconstruction using dynamic programming. In *Proceedings of the 2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 1393–1396. IEEE Computer Society, 2009.
- [21] M. Prandtstetter. *Hybrid Optimization Methods for Warehouse Logistics and the Reconstruction of Destroyed Paper Documents*. PhD thesis, Vienna University of Technology, 2009.
- [22] M. Prandtstetter and G. R. Raidl. Combining Forces to Reconstruct Strip Shredded Text Documents. *HM '08 Proceedings of the 5th International Workshop on Hybrid Metaheuristics*, 5296:175–189, 2008.

- [23] M. Prandtstetter and G. R. Raidl. Meta-Heuristics for Reconstructing Cross Cut Shredded Text Documents. *GECCO '09: Proceedings of the 11th annual conference on Genetic and evolutionary computation*, pages 349–356, 2009.
- [24] R. C. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36(6):1389–1401, 1957.
- [25] G. Raidl and B. Hu. Enhancing Genetic Algorithms by a Trie-Based Complete Solution Archive. *Evolutionary Computation in Combinatorial Optimisation - EvoCOP 2010*, 6022:239–251, 2010.
- [26] S. Ronald. Duplicate Genotypes in a Genetic Algorithm. In *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, pages 793–798, 1998.
- [27] M. Ruthmair. A Memetic Algorithm and a Solution Archive for the Rooted Delay-Constrained Minimum Spanning Tree Problem. In *Proceedings of the 13th International Conference on Computer Aided Systems Theory: Part I*, pages 351–358, 2012.
- [28] S. A. Santosh Kumar and B. K. Shreyamsha Kumar. Edge Envelope based Reconstruction of Torn Document. In *Proceedings of the Seventh Indian Conference on Computer Vision Graphics and Image Processing*, pages 391–397. ACM, 2010.
- [29] C. Schauer. Reconstructing Cross-Cut Shredded Documents by means of Evolutionary Algorithms. Master’s thesis, Vienna University of Technology, 2010.
- [30] C. Schauer, M. Prandtstetter, and G. R. Raidl. A Memetic Algorithm for Reconstructing Cross-Cut Shredded Text Documents. In *Hybrid Metaheuristics 7th Int Workshop HM 2010*, volume 6373 of *LNCIS*, pages 103–117. Springer, 2010.
- [31] A. Skeoch. *An Investigation into Automated Shredded Document Reconstruction using Heuristic Search Algorithms*. PhD thesis, University of Bath, 2006.
- [32] A. Sleit, Y. Massad, and M. Musaddaq. An alternative clustering approach for reconstructing cross cut shredded text documents. *Telecommunication Systems*, pages 1–11, Sept. 2011.
- [33] A. Ukovich, G. Ramponi, H. Doulaverakis, Y. Kompatsiaris, and M. Strintzis. Shredded document reconstruction using MPEG-7 standard descriptors. In *Signal Processing and Information Technology, 2004. Proceedings of the Fourth IEEE International Symposium on*, pages 334–337. IEEE, 2004.
- [34] A. Šramko. Enhancing a genetic algorithm by a complete solution archive based on a trie data structure. Master’s thesis, Vienna University of Technology, 2009.

- [35] Walmart. Cross-cut shredder. <http://www.walmart.com/ip/Fellowes-W-6C-6-Sheet-Cross-Cut-Shredder/15529739>, 2012. Accessed: 06/05/2012.
- [36] S. Y. Yuen and C. K. Chow. A non-revisiting genetic algorithm. *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 4583–4590, 2007.