Enhancing a Genetic Algorithm with a Solution Archive to Reconstruct Cross Cut Shredded Text Documents*

Benjamin Biesinger, Christian Schauer, Bin Hu, and Günther Raidl

Institute of Computer Graphics and Algorithms
Vienna University of Technology
Favoritenstraße 9–11/1861, 1040 Vienna, Austria
{biesinger|schauer|hu|raidl}@ads.tuwien.ac.at

Abstract. In this work the concept of a trie-based complete solution archive in combination with a genetic algorithm is applied to the Reconstruction of Cross-Cut Shredded Text Documents (RCCSTD) problem. This archive is able to detect and subsequently convert duplicates into new yet unvisited solutions. Cross-cut shredded documents are documents that are cut into rectangular pieces of equal size and shape. The reconstruction of documents can be of high interest in forensic science. Two types of tries are compared as underlying data structure, an *indexed trie* and a *linked trie*. Experiments indicate that the latter needs considerably less memory without affecting the run-time. While the archive-enhanced genetic algorithm yields better results for runs with a fixed number of iterations, advantages diminish due to the additional overhead when considering run-time.

Keywords: genetic algorithm, solution archive, reconstruction

1 Introduction

A common weakness of most genetic algorithms is the generation of duplicate solutions, i.e., solutions that have already been visited. As Mauldin showed in [3] maintaining diversity in the population significantly improves the performance of genetic algorithms. Although duplicate detection in stochastic search methods has been studied thoroughly [8, 11], complete solution archives are a rather new approach to fulfill this task. In addition to storing the solution, solution archives are able to efficiently transform an already visited solution into a guaranteed new one. This avoids duplicate solutions, hence unnecessary solution evaluations, as well as premature convergence due to loss of diversity. Ronald has shown in [8] that diversity loss due to duplicate solutions is a weakness of genetic algorithms. In the same paper he introduced hash tagging for duplicate detection. Yuen and Chow used a binary tree in [11] to store all visited solutions. Solution archives have already been successfully applied to several benchmark problems by Raidl

* This work is supported by the Austrian Science Fund (FWF) under grant P24660.

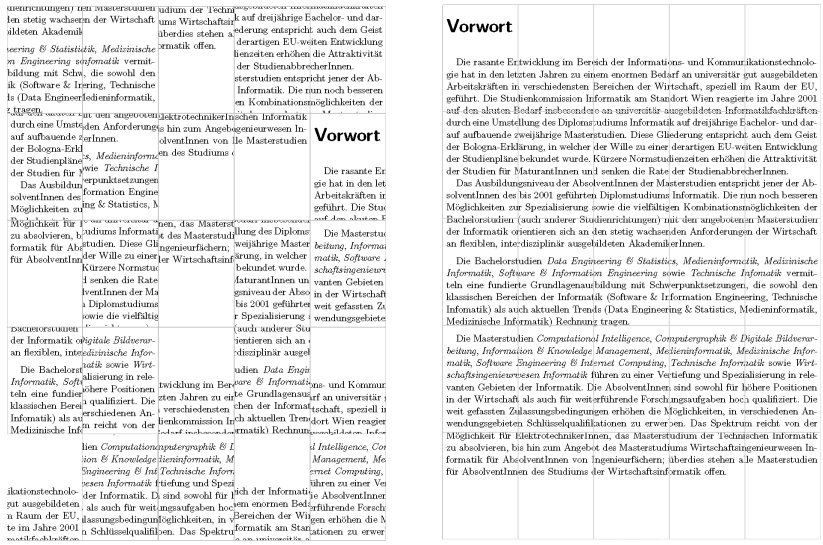

Fig. 1. A shredded document (left) and the fully reconstructed document (right).

and Hu in [7] and to the generalized minimum spanning tree problem in [2]. In this work we will investigate this concept in combination with a genetic algorithm for a more advanced and practical problem, the Reconstruction of Cross-Cut Shredded Text Documents (RCCSTD) problem, which Prandtstetter proved to be $\mathcal{NP}$-complete in [5].

Document shredding is a frequently used method to obfuscate data printed on paper, like personal or sensitive information (e.g., passwords, signatures, PINs). The reconstruction of such data is of high interest especially in the fields of forensic science. Figure 1 shows a sample cross-cut shredded document and its correct reconstruction.

The RCCSTD problem is defined as follows: Let $S = \{1, ..., n\}$ be the set of shreds that belong to one document and let $X \times Y$ be its grid-shaped cutting pattern. A shred $s \in S$ is a non-blank piece of a document. All blank shreds are substituted by a single virtual shred V since there is no exploitable information available on these shreds. A solution to the problem is an injective mapping $\Pi = S \to \mathbb{D}^2$. In this mapping each shred $s \in S$ is assigned to exactly one position (x, y) in the Euclidean space, where $x \in \{0, \ldots, X-1\}$ and $y \in \{0, \ldots, Y-1\}$. The remaining positions are filled with the virtual blank shred. See Figure 2 for a schematic view of a candidate solution, where the white rectangles stand for the virtual shred, which is allowed to be used more than once. Further, we assume here that the document is only printed on one side and the orientation of the shreds is already known.

As a cost function to determine if two shreds should be placed side by side we take the metric from Schauer *et al.* [9] for approximately indicating this likelihood, which is based on the gray value of the pixels along the edges of the

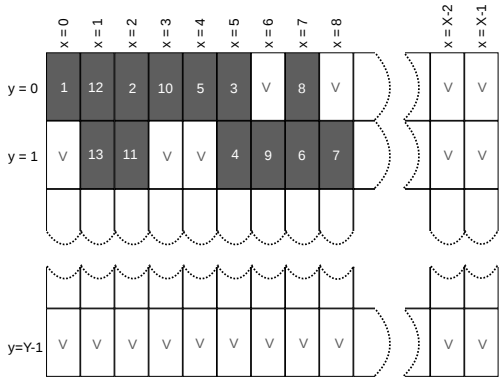

Fig. 2. Schematic view of a solution to the RCCSTD problem.

shreds. If two opposite pixels of the two shreds put side by side differ in their gray values above a certain threshold, a mismatch on this position is detected. Since we focus in this work on the combinatorial aspect of the problem we use this simple metric, which could be replaced by more advanced pattern recognition techniques, e.g., [4]. The objective is to find an injective mapping such that the weighted sum of gray-value mismatches is minimal.

Prandtstetter and Raidl developed an ant colony optimization and a variable neighborhood search for RCCSTD in [6]. Schauer *et al.* described a memetic algorithm to solve the RCCSTD problem in [9], which is modified and extended with a solution archive in this work. A different approach for RCCSTD, which is based on iteratively building clusters of shreds, is described by Sleit *et al.* in [10]. In their approach they merge two clusters that fit well together, while possibly occurring conflicts are repaired.

The rest of the paper is organized as follows. In Section 2 the genetic algorithm and its operators are described. A detailed presentation of the solution archive is given in Section 3. Finally, the computational results are presented in Section 4 and conclusions are drawn in Section 5.

2 Genetic Algorithm

This section gives a short overview of the genetic algorithm (GA). For a detailed description of the operators used within the GA we refer to [9].

Solution Representation: A solution with n shreds is represented as an $X \times Y$ matrix storing for each position the assigned shred. Note that all empty positions are filled with the virtual shred V. This representation is used for all operators of the GA.

Initial Population: To create an initial population the *Row Building Heuristic* and the *Prim-Based Heuristic* are used, which were introduced by Prandtstetter *et al.* in [6], but slightly changed to restrict the solutions to the given format.

Recombination: To exploit the two-dimensional structure of the problem standard crossover operators were adapted for RCCSTD. Preliminary tests showed that modified versions of the 1-point crossover perform best. Instead of choosing a splitting point in the one-dimensional case, for RCCSTD a random splitting line is computed that cuts the solution apart. This line is drawn either horizontally or vertically. In the *Horizontal Block Crossover* a top and a bottom block from two different individuals are recombined to create the offspring, while the *Vertical Block Crossover* recombines a left and a right block. Note that when using these operators invalid solutions could be created. To ensure that each shred is used exactly once, shreds that are already in the solution are replaced by the virtual shred and in the end all missing shreds are inserted using a greedy heuristic.

Mutation: The solution splitting of the block crossover is also performed in the mutation operators. In this case a single solution is either cut apart horizontally (*Horizontal Flop Mutation*) or vertically (*Vertical Flop Mutation*) and then the top/bottom or the left/right parts are exchanged. The *Swap Two Mutation* swaps two shreds randomly.

Selection and Replacement Strategy: We pass the best 10% of individuals with regard to the objective value directly to the next population. The other 90% are generated by applying the genetic operators to uniformly selected individuals of the current population. For a detailed discussion about the selection and generation replacement strategy we refer to [9].

3 A Solution Archive for the RCCSTD

Our primary concern in this work is the following solution archive for duplicate detection. The underlying data structure for our solution archive is a trie, a tree data structure commonly used for storing strings, e.g., for dictionaries in highly compact ways as explained in [1]. Raidl *et al.* argue in [7] that a trie is a very well suited data structure for solution archives in their applications because the operations *search*, *insertion* and *conversion* can be performed in $\mathcal{O}(l)$ time, where l is the length of the solution representation, and thus independent of the number of already stored solutions. Therefore, tries are also used in this work.

3.1 Solution Insertion / Conversion

For inserting a solution of the GA into the trie, the solution is transformed into a more compact one-dimensional array. This is done by adding a special character for line breaks. Blank shreds at the end of lines and empty lines at the bottom are skipped. At the top of Figure 3 two sample transformations are shown. In this example the numbers correspond to actual shreds, the V stands for the virtual shred and the $\hookleftarrow$-symbol is interpreted as a line break.

Then, this array is stored in the trie, where the i-th level of the trie corresponds to the i-th entry of the solution vector. During insertion, *invalid*-flags (denoted as I) are placed to avoid invalid (i.e., that are not in the given format)

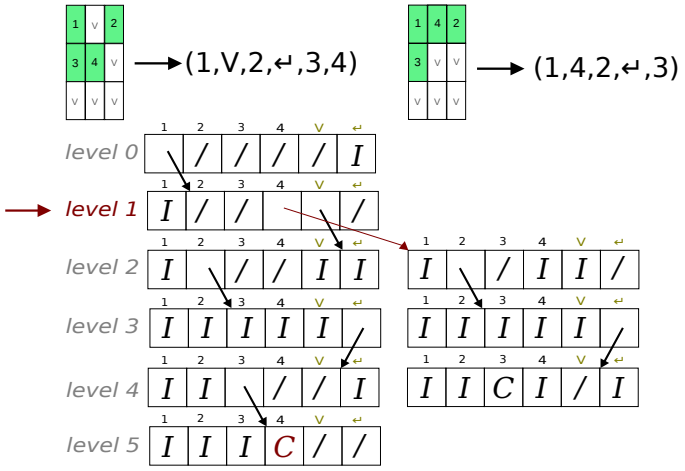

Fig. 3. Solution conversion after detecting a duplicate.

solutions to be stored in the trie. The /-symbols represent not yet explored subtries, i.e., areas of the solution space that have not been visited before. After the last element of the solution has been inserted, a *complete*-flag is placed at the last trie node (denoted as C) to indicate that this solution is in the trie.

Whenever a *complete*-flag is visited during solution insertion, a duplicate is detected and a conversion is performed. In Figure 3 a sample conversion is sketched. Suppose that the solution on the top left of the figure would be inserted twice. Eventually the *complete*-flag is reached and a level to deviate from the existing solution is chosen randomly; in this example level 1 is picked. Then, a different feasible element is chosen at this level and the remaining solution is inserted into the trie with as few changes as possible. The converted solution can be seen on the top right of the figure.

3.2 Adaption of the Trie Data Structure

In preliminary tests we noticed that the trie using standard arrays to realize trie-nodes, the so-called *indexed trie*, needs a huge amount of memory. Hence, in order to reduce the memory consumption we switched to a *linked trie* by replacing the arrays with linked lists. This obviously saves memory especially in the case of sparsely used nodes since the linked lists do not have to allocate space for all possible children at the time when the node is created. A disadvantage of the *linked trie* is that the worst case run-time for the *search* operation in a trie node increases to $\mathcal{O}(k)$ for a node containing k elements.

3.3 Integration of the Solution Archive into the GA

In each iteration the solutions generated by the GA are inserted into the solution archive after mutation. If the solution already exists then the conversion proce-

dure as described above is applied and this operation can also be considered as *intelligent mutation*. The resulting new solution replaces the original solution in the population.

4 Computational Results

We performed our tests on the benchmark instances from [9]. These instances are based on five different text documents cut into nine different patterns ranging from 9×9 to 15×15. We implemented our approach in Java 1.6 and performed all tests on a single core of an Intel Xeon Quadcore CPU with 2.53 GHz and 70 GB RAM.

By using the *indexed trie* as data structure, memory consumption increased quickly after a relatively small number of iterations. Our testing platform ran out of memory and stopped the GA before it converged, which led to worse results than the GA without the archive. By switching to the *linked trie* we are able to drastically reduce the memory usage by about 75% without affecting the run-time of the algorithm. Therefore, only the GA using the *linked trie* as underlying data structure for the solution archive is considered in the following.

The configuration of the GA is similar to the one used in [9]. It uses both recombination operators, see Section 2, with a probability of 50% but only the better of the two offspring individuals created each time is transferred into the next generation. The mutation rate is set to 15%, where each operator is applied with 5%.

To compare the results of the GA with (column "GA-SA") and without (column "GA-time") the archive we performed 30 runs for each configuration and test instance and used a run-time of 300 seconds. Additionally we performed tests where the GA with and without the solution archive run the same number of generation (column "GA-gen"). The results are shown in Table 1. The first two columns refer to the cutting pattern; the number of mismatches of the original document is given in column "orig"; in the columns "gap" and "sd" the gap values are given, which describe the relative difference of the objective values of the reconstructed and the original document and their associated standard deviations. Note that due to inexactness of the cost function, these gap values can sometimes be negative, i.e., a certain reconstruction can have a better fitness value than the original document. To verify the statistical significance of differences in the results of the GA with and without archive, Wilcoxon signed-rank tests with an error probability of 5% were performed as well. The outcome is given in column "p", where "<" means that the algorithm on the left-hand side performed significantly better, an ">" indicates significantly better results for the algorithm on the right-hand side. Whenever a cell is empty the statistical test was not able to make a statement about significant differences.

Unfortunately, even with the improvement of using a *linked trie* the solution archive could not improve the solution quality of the GA when using the same amount of time as stopping criterion. The reason is that the solution archive in this case is not able to fully compensate its overhead by saving the effort for

Table 1. The average percentage gaps and standard deviations of the GA with (GA-SA) and without the solution archive over 30 runs with a run-time of 300 seconds (GA-time) and with the same number of generations (GA-gen).

				GA-time		GA-SA			GA-gen					GA-time		GA-SA			GA-gen	
	x	y	orig	gap/ [%]	sd [%]	p	gap/ [%]	sd [%]	p	gap/ [%]	sd [%]		x	y	orig	gap/ [%]	sd [%]	p	gap/ [%]	sd [%]	p	gap/ [%]	sd [%]
instance p01	9	9	2094	24.8	22.8		33.5	19.1		37.8	19.1	instance p04	9	9	1104	-1.7	12.9		1.2	19.0		4.8	19.5
	9	12	3142	31.9	3.9		31.6	5.3	<	34.1	4.7		9	12	1463	3.0	7.1		3.3	7.4		4.6	9.3
	9	15	3223	34.0	3.8		33.4	6.1	<	36.9	6.3		9	15	1589	-2.8	7.8		-3.6	9.7		-1.8	9.0
	12	9	2907	24.0	10.3		26.3	6.3		29.4	6.4		12	9	1515	39.1	9.0	>	33.8	10.4		35.5	8.9
	12	12	3695	35.4	5.9		33.7	5.4	<	38.8	5.0		12	12	2051	16.4	5.5		18.3	4.8		19.0	6.4
	12	15	3825	36.8	5.1		35.5	4.5	<	40.5	5.1		12	15	2146	0.7	4.3		1.8	4.2		3.3	5.2
	15	9	2931	32.2	14.4	<	38.0	9.9		37.0	7.3		15	9	1567	41.5	11.7		43.7	15.5		47.1	12.9
	15	12	3732	37.8	5.5		37.9	3.1	<	42.0	4.4		15	12	1752	34.0	8.6		35.0	8.4		34.1	8.8
	15	15	3870	42.7	4.0	>	40.4	4.0	<	45.7	4.1		15	15	2026	8.3	5.7		9.5	5.7		9.5	5.5
instance p02	9	9	1434	-16.0	8.9	<	-8.5	12.6	>	-14.7	10.7	instance p05	9	9	690	2.2	3.6	>	0.0	0.0	<	0.8	1.2
	9	12	1060	21.9	13.1		23.3	10.5		22.2	12.7		9	12	888	58.1	31.4		50.1	28.7		55.4	25.1
	9	15	1978	7.7	4.1		7.3	5.2	<	11.9	4.7		9	15	1623	37.7	13.9		40.7	11.3		44.8	10.4
	12	9	1396	-10.0	8.6		-7.8	9.3		-7.6	8.9		12	9	1016	16.5	15.2	>	8.3	12.4	<	17.5	15.8
	12	12	1083	22.8	10.7	<	28.4	12.1		30.7	9.5		12	12	1325	30.9	16.5	<	43.1	14.8		43.5	17.8
	12	15	1904	7.6	5.9	<	10.4	5.3		11.1	7.1		12	15	1986	44.0	7.1		43.3	7.3	<	48.5	9.5
	15	9	1658	-1.6	8.1		1.8	7.9		5.9	11.9		15	9	1010	2.5	13.1		4.0	18.0		1.7	17.9
	15	12	1503	21.3	13.5		22.6	10.6		22.0	11.7		15	12	1156	65.1	15.5		59.9	12.7	<	68.4	12.7
	15	15	2283	15.6	7.6		17.1	5.3		20.1	7.2		15	15	1900	52.2	10.9		49.9	9.4	<	62.1	7.4
instance p03	9	9	2486	5.2	9.0	<	11.5	10.5		11.6	8.4												
	9	12	2651	31.3	6.8	<	36.5	8.9		33.1	9.3												
	9	15	2551	24.8	11.4	<	33.9	9.7	>	27.0	8.9												
	12	9	3075	20.2	4.6		20.3	5.6		19.0	5.3												
	12	12	3377	26.8	7.1		29.9	7.3		32.0	6.6												
	12	15	3313	31.8	6.0		30.0	6.4	<	35.0	9.1												
	15	9	3213	21.5	8.0		22.8	6.9		23.7	7.0												
	15	12	3278	44.0	6.9	>	41.4	5.2	<	47.6	5.6												
	15	15	3308	37.6	5.7	>	34.4	4.4	<	42.5	5.2												

re-evaluating duplicate solutions. However, as can be seen in Table 1, when using numbers of generations as stopping criterion, the GA with solution archive outperforms the GA without the solution archive and achieved statistically better results in 15 out of 45 instances.

5 Conclusions and Future Work

In this work we investigated a solution archive for a genetic algorithm (GA) for the reconstruction of cross-cut shredded text documents problem. We used a solution archive for duplicate detection and a solution conversion method was presented such that no invalid solutions are generated. The first approach of using an *indexed trie* for the archive consumed a huge amount of memory. Therefore, we proposed an alternative data structure, the *linked trie*, in which the trie nodes are stored as linked lists instead of arrays. This modification reduced memory consumption significantly without affecting the run-time in a negative way. The solution archive was able to improve the performance of the GA when run-time is of minor importance. The use of a *linked trie* was essential for this success. However, compared to state-of-the-art algorithms our approach could

not compete because we did not use any pattern recognition technique, which would likely improve results.

Possible future work for the RCCSTD problem should be the incorporation of a more elaborate cost function that uses pattern recognition techniques and to further reduce the memory overhead, e.g., by calculating bounds in order to cut off search areas that evidently do not contain optimal solutions. We will also investigate the concept of solution archives on other problems where we can take considerable advantage of this extension.

References

1. Gusfield, D.: Algorithms on strings, trees, and sequences: computer science and computational biology. Cambridge University Press, New York, NY, USA (1997)
2. Hu, B., Raidl, G.R.: An evolutionary algorithm with solution archives and bounding extension for the generalized minimum spanning tree problem. In: Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation (GECCO). pp. 393–400. ACM Press, Philadelphia, PA, USA (2012)
3. Mauldin, M.L.: Maintaining Diversity in Genetic Search. In: National Conference on Artificial Intelligence. vol. 19, pp. 247–250. AAAI, William Kaufmann (1984)
4. Perl, J., Diem, M., Kleber, F., Sablatnig, R.: Strip shredded document reconstruction using optical character recognition. In: 4th International Conference on Imaging for Crime Detection and Prevention 2011 (ICDP 2011). pp. 1–6 (2011)
5. Prandtstetter, M.: Hybrid Optimization Methods for Warehouse Logistics and the Reconstruction of Destroyed Paper Documents. Ph.D. thesis, Vienna University of Technology (2009)
6. Prandtstetter, M., Raidl, G.R.: Meta-heuristics for reconstructing cross cut shredded text documents. In: Proceedings of the 11th Annual conference on Genetic and evolutionary computation. pp. 349–356. GECCO 2009, ACM Press, New York (2009)
7. Raidl, G.R., Hu, B.: Enhancing genetic algorithms by a trie-based complete solution archive. In: Cowling, P., Merz, P. (eds.) Evolutionary Computation in Combinatorial Optimization. LNCS, vol. 6022, pp. 239–251. Springer Berlin Heidelberg (2010)
8. Ronald, S.: Duplicate genotypes in a genetic algorithm. In: Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence. pp. 793–798 (1998)
9. Schauer, C., Prandtstetter, M., Raidl, G.R.: A memetic algorithm for reconstructing cross-cut shredded text documents. In: Proceedings of the 7th international conference on Hybrid metaheuristics. LNCS, vol. 6373, pp. 103–117. Springer-Verlag, Berlin, Heidelberg (2010)
10. Sleit, A., Massad, Y., Musaddaq, M.: An alternative clustering approach for reconstructing cross cut shredded text documents. Telecommunication Systems pp. 1–11 (2011)
11. Yuen, S.Y., Chow, C.K.: A non-revisiting genetic algorithm. In: Evolutionary Computation, 2007. CEC 2007. IEEE Congress on. pp. 4583–4590 (2007)