Discrete Mathematics 106/107 (1992) 11–18
North-Holland

# Local search in coding theory

Emile H.L. Aarts

*Philips Research Laboratories, P.O. Box 80.000, 5600 JA Eindhoven, Netherlands*
*Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, Netherlands*

Peter J.M. van Laarhoven

*McKinsey & Company, Amstel 344, 1017 AS Amsterdam, Netherlands*

Received 17 December 1991
Revised 29 January 1992

*Abstract*

Aarts, E.H.L. and P.J.M. van Laarhoven, Local search in coding theory, Discrete Mathematics 106/107 (1992) 11–18.

We briefly review the application of local search to a special class of coding problems: covering and packing. To use local search techniques, covering and packing problems are viewed as combinatorial optimization problems. The advantage of local search is that it can be applied without the use of deep combinatorial arguments. However, the required computation times can be quite large.

## 1. Introduction

We consider the special class of coding problems known as covering and packing. In these problems we are given a set of items $V$ and the question is to find a set $W^* \subset V$, such that

• $W^*$ is of minimum cardinality and for each item in $V$ there is at least one item in $W^*$ such that their 'distance' is at most a given value (covering), or

• $W^*$ is of maximum cardinality and the 'distance' between any two items in $W^*$ is at least a given value (packing).

Clearly, in some sense these problems can be viewed as each other's dual. In coding theory, items are often words of a code and the distance is usually given by the Hamming distance between two words.

Traditionally, covering and packing problems have been solved by using combinatorial arguments; for an overview see [6]. Here, we present results that have been obtained by formulating these problems as combinatorial optimization

*Correspondence to:* E.H.L. Aarts, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, Netherlands.

problems and handling them by optimization techniques such as simulated annealing and genetic algorithms.

Before we discuss the problems in more detail, we introduce the following notations. $V_k^n$ is the set of all $n$-tuples $x = (x_1, \ldots, x_n)$ with $x_i \in \mathbb{Z}_k = \{0, 1, \ldots, k-1\}$. The Hamming distance $d_H(x, y)$ between two $n$-tuples $x, y \in V_k^n$ is defined as the number of entries in which the two $n$-tuples differ. The Hamming weight $w_H(x)$ of an $n$-tuple $x \in V_k^n$ is defined as the number of entries of $x$ different from 0, i.e., $w_H(x) = d_H(x, 0)$, where $0 \in V_k^n$ is the $n$-tuple with all entries equal to 0.

A $k$-ary code $W$ of length $n$ is a subset of $V_k^n$; the elements of $W$ are usually called code words. For $x \in V_k^n$, we define the sphere of radius $R$ around $x$ by $B_R(x) = \{y \in V_k^n \mid d_H(x, y) \leq R\}$. If $R = 1$, the sphere is also called a rook domain and it is denoted by $B(x)$.

We now formulate the following problems.

(a) Let $W$ be a covering by rook domains of $V_k^n$, i.e., $W \subset V_k^n$ and $V_k^n = \bigcup_{x \in W} B(x)$. Find $\sigma(n, k)$, i.e., the cardinality of the smallest covering by rook domains of $V_k^n$. Note that this implies that for any $y \in V_k^n$ there is at least one $x \in W$, such that $d_H(x, y) \leq 1$.

For $k = 3$, this covering problem is known as the football pool problem for the following reasons. With an entry of a football pool one forecasts the results of $n$ football matches. If one wishes the guarantee of winning the first prize, one obviously has to submit $3^n$ entries. To have the guarantee of winning the second prize, one has to submit a set of entries such that for any set of results there is at least one entry that differs for at most one match in the full set of results. Note that this amounts to a set of entries that is a covering by rook domains of the set of all possible results given by $V_3^n$. Thus $\sigma(n, 3)$ corresponds to the smallest set of entries that guarantees winning the second prize in a football pool of $n$ matches.

(b) Let $W$ be a $k$-ary $(n, d)$-code, i.e., $W \subset V_k^n$ and $d_H(x, y) \geq d$ for all $x, y \in W$, $x \neq y$. Find $A_k(n, d)$, i.e., the cardinality of a largest $k$-ary $(n, d)$-code.

This packing problem is known as the problem of constructing an error-correcting or channel code, due to the fact that up to $d/2$ transmission errors can be corrected if an $(n, d)$-code is used in a communication channel.

(c) Let $W$ be a $k$-ary $(n, d, w)$-code, i.e., a $k$-ary $(n, d)$-code in which all code words have Hamming weight $w$. Find $A_k(n, d, w)$, i.e., the cardinality of a largest $k$-ary $(n, d, w)$-code.

This packing problem is known as the problem of constructing a constant weight error-correcting code.

To formulate the above problems as combinatorial optimization problems, we recall that combinatorial optimization problems can be formalized as a pair $(\mathcal{S}, f)$, where $\mathcal{S}$ denotes a set of solutions and $f : \mathcal{S} \to \mathbb{R}$ a cost function that assigns a real-valued number to each solution in $\mathcal{S}$. Evidently, as $|\mathcal{S}|$ can be very large, the elements in $\mathcal{S}$ are not explicitly given but are specified by a compact representation from which they can be computed with some algorithm. The problem in combinatorial optimization is to find a solution $i^* \in \mathcal{S}$ for which $f(i^*)$ is optimal. It is now straightforward to formulate each of the aforementioned coding problems as combinatorial optimization problems. For instance, the problem of constructing an error-correcting code can be formulated as follows (see problem (b)).

Let $\mathcal{S} = \{W \mid W \text{ is a } k\text{-ary } (n, d)\text{-code}\}$, and $f(W) = |W|$ for all $W \in \mathcal{S}$. Find a $W^* \in \mathcal{S}$ that maximizes $f$.

To enable the use of local search techniques we, however, need a slightly different formulation. This and other local search related items are addressed in the following sections.

The organization of the remainder of the paper is as follows. Section 2 briefly discusses some aspects of local search in combinatorial optimization. Section 3 discusses the application of local search to the coding problems given above. The paper concludes with some final remarks given in Section 4.

## 2. Local search

Many combinatorial optimization problems have been proved to be NP-hard, which implies that it is unlikely that there exist algorithms that can solve each instance of such a problem to optimality within a computation time that is bounded by a polynomial in the size of the instance. Here the size of an instance refers to a number that is polynomially related to the size of the encoding used to store the compact representation of $\mathcal{S}$ mentioned in Section 1. As a consequence of the NP-hardness, one often resorts to approximation algorithms that find high-quality approximate solutions within acceptable computation times.

Local search constitutes a well-known class of approximation algorithms. The use of such techniques presupposes, in addition to the definition of the pair $(\mathcal{S}, f)$, a neighbourhood structure and a transition mechanism. A neighbourhood structure is a mapping $\mathcal{N}: \mathcal{S} \to 2^{\mathcal{S}}$, which assigns to each solution $i \in \mathcal{S}$ a neighbourhood $\mathcal{N}(i) \subset \mathcal{S}$, which is a set of solutions that are 'close' to $i$ in some sense. A transition mechanism is a set of rules that determines the iteration process among neighbouring solutions. Roughly speaking, one can view local search as a walk on a directed graph, whose nodes are given by the elements in $\mathcal{S}$ and whose arcs connect two neighbouring solutions; the transition mechanism then determines the walk that should eventually lead to good final solutions.

Iterative improvement is a simple example of a local search algorithm that can be described as follows. Given a start solution, generate a sequence of trials, where in each trial a current solution is replaced by a neighbouring solution with better cost. The algorithm stops if a solution is found whose cost is at least as good as the cost of all its neighbours. Such a solution is called a local optimum. The major advantage of iterative improvement is its general applicability; a start solution and a neighbourhood structure are usually straightforward to define. A drawback is the fact that in general no guarantees can be given with respect to the cost of the local optimum that is found by the algorithm and that the cost value can be far from optimal since the algorithm stops at the first local optimum that is found. To overcome this disadvantage, whilst maintaining the advantage of the generally applicable local search paradigm, a number of new transition mechanisms have been proposed in recent years, many of which have led to impressive results. Two of them, namely simulated annealing and genetic local search, have been shown to be successful in coding and are therefore discussed in more detail here.

### *2.1. Simulated annealing*

Simulated annealing is a generalization of iterative improvement in the sense that in addition to cost-improving, also cost-deteriorating transitions are made. More precisely, given two solutions $i, j \in \mathcal{S}$, with $j \in \mathcal{N}(i)$ and $f(j) \geq f(i)$, then, in the case of minimization, a transition from $i$ to $j$ can be made with nonzero probability. A frequently used expression for the probability is given by (again in the case of minimization)

$$\mathbb{P}\{i \to j \mid j \in \mathcal{N}(i)\} = \exp\left(-\frac{f(j)-f(i)}{c}\right),$$

where $c \in \mathbb{R}^+$ is some control parameter. Note that the probability decreases for increasing values of the cost difference $f(j) - f(i)$, as well as for decreasing values of $c$. The algorithms starts with a large value of $c$, which is gradually decreased in the course of the algorithm's execution to become zero in the end. So initially, cost-increasing transitions are frequently made, whereas in the end we have again iterative improvement. The basic idea of simulated annealing is gleaned from nature, more precisely, from the physical annealing process; see Kirkpatrick et al. [3].

A major advantage of simulated annealing is that under relatively mild conditions the algorithm can be proved to find asymptotically, i.e., for $c \downarrow 0$, optimal solutions with probability 1. In practice, the asymptoticity region cannot be reached, nevertheless the algorithm has been successfully applied to problems in a wide variety of application areas, yielding high-quality solutions, but often at the cost of substantial amounts of running times. For more details see Van Laarhoven and Aarts [4].

### *2.2. Genetic local search*

Another approach to improve upon iterative improvement is given by the class of genetic local search algorithms. Again the basic idea is gleaned from nature leading in this case to a combination of iterative improvement with recombination mechanisms and population genetics in biology; see Mühlenbein et al. [7].

Instead of iterating among single solutions, genetic local search algorithms continuously try to improve a population of solutions. To this end two neighbourhood structures are used. The improvement neighbourhood structure is used to generate a neighbouring solution of a single solution, as in iterative improvement. The recombination neighbourhood structure is used to generate a solution from a set of solutions, as in nature where two parents produce offspring. The general idea of genetic local search is best explained by the following scheme.

*Step* 1. *Initialize:* Construct an initial population of $n$ solutions.

*Step* 2. *Improve:* Use iterative improvement to replace the $n$ solutions in the population by $n$ local optima.

*Step* 3. *Recombine:* Augment the population by adding $m$ offspring solutions. The population size now equals $n + m$.

*Step* 4. *Improve:* Use iterative improvement to replace the $m$ offspring solutions by $m$ local optima.

*Step* 5. *Select:* Reduce the population to its original size by selecting the $n$ best solutions from the current population.

*Step* 6. *Evolute:* Repeat Steps 3–5 until a stop criterion is satisfied.

Evidently, an important step is the recombination, since there one must try to exploit the structure present in the available local optima.

As with simulated annealing, genetic local search has been applied with success to a large class of problems. In the next section we discuss some of the successes obtained in coding theory.

## 3. Local search for covering and packing

### 3.1. *Covering by simulated annealing*

Van Laarhoven et al. [5] describe an application of simulated annealing to the football pool problem, i.e., they compute upper bounds on the value of $\sigma(n, 3)$ for different values of $n$. They use the following problem formulation. Let

$$\mathcal{S} = \{W \mid W \in V_3^n \text{ and } |W| = a\} \quad \text{and} \quad f(W) = \left|\left\{V_3 \Big\backslash \bigcup_{x \in W} B(x)\right\}\right|.$$

Find a $W^* \in \mathcal{S}$ that minimizes $f$.

If the solution to this problem is a code $W$, for which $f(W) = 0$, then $W$ is a covering of $V_3^n$ by rook domains and $\sigma(n, 3) \leq a$. The above problem formulation allows the use of extremely simple neighbourhood structures. Given a code $W \in \mathcal{S}$, then its neighbourhood consists of all codes $W' \in \mathcal{S}$ that can be obtained from $W$ by deleting a code word $x \in W$ and adding a code word $y \in V_3^n \backslash W$. Van Laarhoven et al. [5] have used these neighbourhoods in a simulated annealing algorithm and were able to improve the previous best values of $\sigma(6, 3)$, $\sigma(7, 3)$, and $\sigma(8, 3)$, from 79, 216, and 567, respectively, to 73, 186, and 486, respectively. To obtain the latter improvement, also combinatorial arguments were needed in conjunction with the annealing approach. The computation times used to obtain these results were in the range of several minutes for the smaller instances up to a few hours for the large ones.

## *3.2. Packing by simulated annealing*

El Gamal et al. [2] describe an application of simulated annealing to the problem of finding good binary constant weight codes, i.e., they compute lower bounds on $A_2(n, d, w)$ for different values of $n$, $d$, and $w$. They use the following problem formulation, which is slightly different from the one given in Section 1. Let

$$\mathscr{S} = \{W \mid W \in V_2^n, |W| = a \text{ and for all } x \in W\colon w_H(x) = w\} \quad \text{and}$$

$$f(W) = \sum_{x \in W} \sum_{y \in W, y \neq x} [d_H(x, y)]^{-2} \quad \text{for all } W \in \mathscr{S}.$$

Find a $W^* \in \mathscr{S}$ that minimizes $f$.

In other words, the set of solutions is restricted to constant weight codes of length $n$ and cardinality $a$, and the cost function is chosen to favour codes with large distances between the code words. If a solution to this problem is a code $W^*$ for which $d_H(x, y) \geq d$, for all $x, y \in W^*$, $x \neq y$, then $W^*$ is a binary $(n, d, w)$-code and $A_2(n, d, w) \geq a$. One of the reasons to use this alternative formulation is that it allows the use of an extremely simple neighbourhood structure. Given a code $W \in \mathscr{S}$, then its neighbourhood consists of all codes $W' \in \mathscr{S}$ that can be obtained from $W$ by taking one of the code words $x \in W$ and swapping one component $x_i$ from 1 to 0, and another component $x_j$ from 0 to 1. Evidently, the resulting code $W'$ is again in $\mathscr{S}$.

El Gamal et al. [2] use these neighbourhoods in a simulated annealing algorithm and report that several good codes have been found with their approach. They found a $(23, 10, 7)$-code of size 18, a $(23, 10, 8)$-code of size 28, and a $(24, 10, 8)$-code of size 33, which all improve the previously known best codes. For some codes the obtained lower bounds are still far off from the corresponding theoretical upper bounds. For instance, 68 is an upper bound for $A_2(24, 10, 8)$. The computation times used to compute these codes are in the order of several minutes up to a few hours.

## *3.3. Packing by genetic local search*

Vaessens et al. [8] describe an application of genetic local search to the problem of constructing good ternary $(n, d)$-codes, i.e., they compute lower bounds on $A_3(n, d)$ for different values of $n$ and $d$. In this case, the improvement neighbourhood of a given ternary $(n, d)$-code $W$ is given by all ternary $(n, d)$-codes $W'$ that can be obtained from $W$ by adding one code word $x \in V_3^n \setminus W$ to $W$. Furthermore, $x$ should satisfy the condition $d_H(x, y) \geq d$ for all $y \in W$, in order for $W'$ to be an $(n, d)$-code. The recombination neighbourhood of two ternary $(n, d)$-codes $W_1$ and $W_2$ is given by all ternary $(n, d)$-codes $W'$ for which

$$W' = \{x \in W_1 \mid d_H(x, z) \leq D - d\} \cup \{x \in W_2 \mid d_H(x, z) \geq D\}$$

or

$$W' = \{x \in W_2 \mid d_H(x, z) \leq D - d\} \cup \{x \in W_1 \mid d_H(x, z) \geq D\}$$

with arbitrary code words $z \in V_3^n$ and integers $D$, $0 \leq D \leq n + d$.

In their paper, Vaessens et al. [8] give a table of $A_3(n, d)$ bounds for $n \leq 16$. Several of the best found lower bounds were obtained with a genetic local search algorithm operating along the lines sketched above. The computation times needed to find these results in some cases could mount up to a few days.

## 4. Concluding remarks

The results discussed in this paper illustrate the successful use of rather simple approaches such as local search to difficult problems as covering and packing. They demonstrate that these approaches, which can be viewed as clever approximate enumerations of the set of solutions, are viable alternatives to the more traditional combinatorial construction techniques that are used to handle these problems. The search for better lower and upper bounds remains a challenging subject in combinatorics, since the gap between the two is in many cases still quite large. We mention two possible directions to extend this work. On the one hand, one could try to use combinatorial arguments in order to construct more complicated neighbourhoods which then might further improve the present results. On the other hand, local search techniques could be used to construct special classes of codes which can then be further used to study the combinatorial structure of other more complicated codes, as in fact is done to obtain the rook domain covering of $V_3^8$ by Van Laarhoven et al. [5].

Finally, we must admit that there are also examples of problems in coding theory for which local search so far fails. For instance, Beenker et al. [1] report on an extensive numerical study in which they investigated the performance of several methods for the problem of finding binary sequences with maximal autocorrelation coefficients. One of their conclusions was that the local search techniques they applied could not improve upon the more traditional approaches.

## References

[1] G.F.M. Beenker, T.A.C.M. Claasen and P.W.C. Hermens, Binary sequences with maximally flat amplitude spectrum, Philips J. Res. 40 (1985) 289–304.

[2] A. El Gamal, L.A. Hemachandra, I. Shperling and V.K. Wei, Using simulated annealing to design good codes, IEEE Trans. Inform. Theory 33 (1987) 116–123.

[3] S. Kirkpatrick, C.D. Gelatt Jr and M.P. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671–680.

[4] P.J.M. van Laarhoven and E.H.L. Aarts, Simulated Annealing: Theory and Applications (Reidel, Dordrecht, 1987).

[5] P.J.M. van Laarhoven, E.H.L. Aarts, J.H. van Lint and L.T. Wille, New upper bounds for the football pool problem for 6, 7 and 8 matches, J. Combin. Theory Ser. A 52 (2) (1989) 304–312.
[6] J.H. van Lint, Introduction to Coding Theory (Springer, New York, 1982).
[7] H. Mühlenbein, M. Gorges-Schleuter and O. Krämer, Evolution algorithms in combinatorial optimization, Parallel Comput. 7 (1988) 65–85.
[8] R.J.M. Vaessens, E.H.L. Aarts and J.H. van Lint, Genetic algorithms in coding theory: a table for $A_3(n, d)$, Philips Research Laboratories, manuscript, M.S.16.373.