Distortion Minimization with Fast Local Search for Fractal Image Compression

Raouf Hamzaoui, Dietmar Saupe, and Michael Hiller

Institut für Informatik, Universität Leipzig, Augustusplatz 10–11, 04109 Leipzig, Germany

E-mail: hamzaoui,saupe@informatik.uni-leipzig.de

Optimal fractal image coding is an NP-hard combinatorial optimization problem, which consists of finding in a finite set of contractive affine mappings one whose unique fixed point is closest to the original image. Current fractal image schemes are based on a greedy suboptimal algorithm known as collage coding. In a previous paper, Hamzaoui, Hartenstein, and Saupe proposed a local search algorithm that iteratively improves an initial solution found by collage coding. For a standard fractal scheme based on quadtree image partitions peak-signal-to-noise ratio (PSNR) gains are up to 0.8 dB. However, the algorithm is time-consuming because it involves many iteration steps, each of which requires the computation of the fixed point of an affine mapping. In this paper, we provide techniques that drastically reduce the complexity of the algorithm. Moreover, we show that the algorithm is also successful with a state-of-the-art fractal scheme based on more general image partitions.

Key Words: fractal image compression, combinatorial optimization, NP-hard problems, local search, graph algorithms

1. INTRODUCTION

Fractal image compression was introduced by Barnsley [2] and Jacquin [15]. Since then, many researchers improved the original approach in various ways [22]. Unfortunately, the rate-distortion results of the best fractal coders are still inferior to those of the state-of-the-art in image compression [22]. However, the potential of fractal image compression has not been fully exploited because current fractal schemes do not find optimal codes. Optimal fractal coding can be seen as a combinatorial optimization (minimization) problem where the domain of feasible solutions is a large finite set of contractive affine mappings, and the cost function is the distortion between the original image and the fixed point of one such mapping. Current fractal coders are based on a greedy algorithm known as collage coding which finds only a suboptimal solution. Even though the determination of an optimal solution is computationally infeasible [18, 12], better solutions than those obtained by collage coding have been reported [4, 5, 3, 14, 7, 16, 21, 11]. The most successful approach [11] uses local search [17, 1]. The algorithm starts from a code computed by collage coding and keeps on improving this code with a heuristic introduced in [3, 16] until convergence to a locally optimal solution. For quadtree-based image partitions [9], typical PSNR improvements over collage coding range from 0.2 dB to 0.8 dB for 8 bits per pixel 512×512 real-world images. Unfortunately, the algorithm is time expensive because many iteration steps are needed for convergence. Moreover, the evaluation of the cost function, which is needed at each iteration step, requires the computation of the fixed point of an affine mapping of a high-dimensional space.

This paper has two purposes. First, we show that the local search algorithm is also successful with the state-of-the-art fractal scheme of [19, 13]. This is an important result because this scheme, which uses highly adaptive image partitions, provides better objective and subjective reconstruction fidelity than the schemes based on quadtree partitions.

Second, we provide several techniques that improve the computational efficiency of the local search algorithm. Of these measures the most important one exploits the dependence graph [6] of the code for a fast computation of the cost function.

The rest of the paper is organized as follows. In Section 2, we give a short self-contained description of fractal image compression and present previous work. In Section 3, we give numerical results of our local search method for the fractal scheme of [19, 13]. In Section 4, which is the main part of the paper, we provide the techniques that accelerate the local search algorithm. Section 5 contains numerical results of an implementation based on these techniques together with an analysis of memory requirements. In the last section, we summarize and discuss the results.

2. BACKGROUND

In this section, we introduce terminology, provide a generic fractal coding scheme, and present previous work.

2.1. Terminology

In fractal image compression, the code for an original image is given by a contractive mapping of a complete metric space of digital images $(\mathcal{F}, d)$ [15, 2, 8]. The goal of the encoder is to find a contractive mapping T of which the fixed point f_T is a good approximation to the original image. The decoder computes f_T as the limit point of the sequence of iterates $\{f_k\}_{k\geq 0}$ where $f_{k+1} = T(f_k)$, and f_0 is an arbitrary starting image. Given a target image f^*, a large finite set of contractive mappings $\mathcal{T}$, and a number of bits r, an optimal encoding is a solution to the constrained minimization problem

$$\min_{T\in\mathcal{T}} \Delta(f^*, f_T) \text{ subject to } \mathrm{len}(c(T)) \leq r.$$

Here $\Delta(f^*, f_T) \geq 0$ is the reconstruction error, and $\mathrm{len}(c(T))$ is the length of the code $c(T)$ of the mapping T.

Let $\mathcal{F}$ be the vector space of digital images $f : I = \{0, 1, \ldots, N-1\} \times \{0, 1, \ldots, N-1\} \to \mathbb{R}$. Then $\mathcal{F}$ is a complete metric space for the l_∞ metric

$$d_\infty(f, g) = \|f - g\|_\infty = \max_{i,j} |(f(i, j) - g(i, j)|.$$

Let $B = \{i, i+1, \ldots, i+n-1\} \times \{j, j+1, \ldots, j+n-1\} \subset I$ be an $n \times n$ square block. We denote by $\mathbf{x}_{f|B}$ the column vector formed by stacking the pixel intensities of B row by row, left to right, and top to bottom, that is, $\mathbf{x}_{f|B} = (f(i,j), f(i,j+1), \ldots, f(i,j+n-1), f(i+1,j), f(i+1,j+1), \ldots, f(i+n-1,j), f(i+n-1,j+1), \ldots, f(i+n-1,j+n-1))^T$. Let $\mathcal{R} = \{R_1, \ldots, R_{n_R}\}$ be a partition of I into pairwise disjoint $2^n \times 2^n$ square blocks called *range blocks*. Let $\mathcal{D} = \{D_1, \ldots, D_{n_D}\}$ be a set of $2^{n+1} \times 2^{n+1}$ square blocks $D_i \subset I$ called *domain blocks*. Let $\mathcal{S} = \{s_1, \ldots, s_{n_s}\} \subset [-s_{\max}, s_{\max}]$, $s_{\max} < 1$, be a set of real numbers called *scaling factors*. Let $\mathcal{O} = \{o_1, \ldots, o_{n_o}\}$ be a set of real numbers called *offsets*. Let $\mathcal{P} = \{P_1, \ldots, P_{n_P}\}$ be a set of permutation matrices of order 2^{2n}. To each tuple

$$((D(1), s(1), o(1), P(1)), \ldots, (D(i), s(i), o(i), P(i)), \ldots, (D(n_R), s(n_R), o(n_R), P(n_R)))$$

where $(D(i), s(i), o(i), P(i)) \in \mathcal{D} \times \mathcal{S} \times \mathcal{O} \times \mathcal{P}$, we associate a transformation $T : \mathcal{F} \to \mathcal{F}$ such that for $f \in \mathcal{F}$ and all $i \in \{1, \ldots, n_R\}$

$$\mathbf{x}_{T(f)|R_i} = s(i)P(i)A\mathbf{x}_{f|D(i)} + o(i)\mathbf{1}.$$

Here $\mathbf{1} = (1, \ldots, 1)^T \in \mathbb{R}^{2^{2n}}$, and A is the $2^{2n} \times 2^{2(n+1)}$ *downsampling matrix*

$$A = \frac{1}{4} \begin{pmatrix} Q & Q & 0 & 0 & \ldots & 0 & 0 \\ 0 & 0 & Q & Q & 0 & \ldots & 0 \\ \vdots & & & & & & \vdots \\ 0 & \ldots & 0 & \ldots & 0 & Q & Q \end{pmatrix},$$

where Q is the $2^n \times 2^{n+1}$ submatrix

$$Q = \begin{pmatrix} 1 & 1 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & \dots & 0 & 0 & 0 \\ \vdots & & & & & & & & \vdots \\ 0 & \dots & 0 & 0 & 0 & \dots & 0 & 1 & 1 \end{pmatrix}.$$

The matrix A reduces the size of vector $\mathbf{x}_{f|D(i)}$ to 2^{2n} by averaging the intensities of pairwise disjoint groups of four neighboring pixels of $D(i)$.

Because $|s(i)| < 1$ for all $i = 1, \dots, n_R$, T is a contraction for the l_∞ metric. Indeed, let f and g be two images. Then

$$\begin{aligned} \|T(f) - T(g)\|_\infty &= \max_{1 \le i \le n_R} \|\mathbf{x}_{T(f)|R_i} - \mathbf{x}_{T(g)|R_i}\|_\infty \\ &= \max_{1 \le i \le n_R} \|s(i)P(i)A(\mathbf{x}_{f|D(i)} - \mathbf{x}_{g|D(i)})\|_\infty \\ &\le (\max_{1 \le i \le n_R} |s(i)|) \max_{1 \le i \le n_R} \|A\|_\infty \|P(i)\|_\infty \|\mathbf{x}_{f|D(i)} - \mathbf{x}_{g|D(i)}\|_\infty \\ &= (\max_{1 \le i \le n_R} |s(i)|) \max_{1 \le i \le n_R} \|\mathbf{x}_{f|D(i)} - \mathbf{x}_{g|D(i)}\|_\infty \\ &\le (\max_{1 \le i \le n_R} |s(i)|) \|f - g\|_\infty. \end{aligned}$$

Thus, the decoding

$$f_0 \to T(f_0) \to T(T(f_0)) \to \cdots$$

is convergent to the fixed point f_T independently of the initial image f_0.

The set of all such transformations T is denoted by $\mathcal{T}$. Moreover, we call $\mathcal{D}$ a *domain pool*. Let $T \in \mathcal{T}$ be given by $D(i), s(i), o(i), P(i)$, $i = 1, \dots, n_R$. Then we call $(D(i), s(i), o(i), P(i))$ the *fractal parameters* of range block R_i. We also say that R_i is *encoded* by its fractal parameters. The code for T consists of bits that specify the fractal parameters of the range blocks.

For clarity of presentation, our generic fractal scheme was restricted to uniform partitions into square blocks. However, fractal coding provides better rate-distortion and subjective quality results when the partition is adapted to the contents of the image [22]. This includes quadtree partitions, rectangular partitions,

and more sophisticated partitions based on region growing [19, 13]. Because the range blocks may have various sizes and shapes, several domain pools have to be used. With nonuniform image partitions, the code for T must include additional bits to describe the image partition.

2.2. Collage coding

Let us assume that the codewords of the transformations T have the same length. Then an optimal transformation T_{opt} is one that minimizes the reconstruction error

$$E(T) = \Delta(f^*, f_T) = \|f^* - f_T\|_2^2 = \sum_{i,j} (f^*(i,j) - f_T(i,j))^2$$

over all feasible solutions

$$((D(1), s(1), o(1), P(1)), \ldots, (D(n_R), s(n_R), o(n_R), P(n_R))).$$

There are $(n_D n_s n_o n_P)^{n_R}$ such feasible solutions. Thus, finding T_{opt} by enumeration is impractical for large n_R. Usually, a suboptimal solution is found by a greedy algorithm known as collage coding [15]. The idea consists of minimizing the *collage error* $\Delta(f^*, T(f^*))$ instead of the reconstruction error $\Delta(f^*, f_T)$. The motivation for collage coding is the inequality

$$\|f^* - f_T\|_2 \leq \frac{1}{1 - s(T)} \|f^* - T(f^*)\|_2,$$

where it is assumed that T is a contraction for the euclidean metric, and where $s(T)$ is the contraction factor of T. Collage coding simplifies the optimization problem because

$$\begin{aligned} \Delta(f^*, T(f^*)) &= \sum_{i=1}^{n_R} \|\mathbf{x}_{f^*|R_i} - \mathbf{x}_{T(f^*)|R_i}\|_2^2 \\ &= \sum_{i=1}^{n_R} \|\mathbf{x}_{f^*|R_i} - (s(i)P(i)A\mathbf{x}_{f^*|D(i)} + o(i)\mathbf{1})\|_2^2. \end{aligned}$$

Hence optimal fractal parameters in collage coding are solutions of the n_R independent minimization problems

$$\min_{(D(i),s(i),o(i),P(i))\in\mathcal{D}\times\mathcal{S}\times\mathcal{O}\times\mathcal{P}} \|\mathbf{x}_{f^*|R_i} - (s(i)P(i)A\mathbf{x}_{f^*|D(i)} + o(i)\mathbf{1})\|_2^2,\ i = 1,\ldots,n_R.$$

Each of these minimization problems is usually treated as follows. For a given $(D(i), P(i)) \in \mathcal{D}\times\mathcal{P}$, let s and o denote the solutions of the least squares problem

$$\min_{s,o\in\mathbb{R}} \|\mathbf{x}_{f^*|R_i} - (sP(i)A\mathbf{x}_{f^*|D(i)} + o\mathbf{1})\|_2^2.$$

If we denote the vector $\mathbf{x}_{f^*|R_i}$ by $\mathbf{r}$ and the vector $P(i)A\mathbf{x}_{f^*|D(i)}$ by $\mathbf{c}$, and if we assume that $\mathbf{x}_{f^*|D(i)}$ is not in the linear span of $\mathbf{1}$, then the least squares solution is given by

$$s = \frac{2^{2n}(\mathbf{c}^T\mathbf{r}) - \mathbf{c}^T\mathbf{1}\mathbf{r}^T\mathbf{1}}{2^{2n}\mathbf{c}^T\mathbf{c} - (\mathbf{c}^T\mathbf{1})^2} \tag{1}$$

and

$$o = \frac{1}{2^{2n}}(\mathbf{r}^T\mathbf{1} - s\mathbf{c}^T\mathbf{1}). \tag{2}$$

Next, we quantize s and o in $\mathcal{S}$ and $\mathcal{O}$ respectively, yielding a scaling factor s^* and an offset o^*. Finally, we find a pair $(D(i), P(i))$ in $\mathcal{D}\times\mathcal{P}$ that minimizes the error

$$\|\mathbf{x}_{f^*|R_i} - (s^*P(i)A\mathbf{x}_{f^*|D(i)} + o^*\mathbf{1})\|_2^2,$$

which is equal to

$$\mathbf{c}^T\mathbf{c}(s^*)^2 + 2\mathbf{c}^T\mathbf{1}s^*o^* + 2^{2n}(o^*)^2 - 2\mathbf{r}^T\mathbf{c}s^* - 2\mathbf{r}^T\mathbf{1} + \mathbf{r}^T\mathbf{r}. \tag{3}$$

The minimum such error is called the local collage error for range block R_i.

2.3. Previous work

Several researchers recognized the suboptimality of collage coding and proposed better, though also suboptimal, solutions. Most of these works [4, 14, 7, 21] start

from the solution found by collage coding, fix the domain blocks and the permutations, and optimize the scaling factor and the offset (considered as continuous variables) by, for example, gradient descent methods, which yield local minima of the reconstruction error. However, after quantization, the PSNR improvement over collage coding is negligible for practical encodings [21]. Moreover, the time complexity of the optimization is too high. In contrast, Barthel and Voyé [3] and Lu [16] suggested to update all fractal parameters by an iterative procedure, which starts from an original solution T_0 found by collage coding, then at step $n \geq 1$, modifies the fractal parameters of all range blocks R_i, $i = 1, 2, \ldots, n_R$ by solving the minimization problem

$$\min_{(D(i),s(i),o(i),P(i))\in\mathcal{D}\times\mathcal{S}\times\mathcal{O}\times\mathcal{P}} \|\mathbf{x}_{f^*|R_i} - (s(i)P(i)A\mathbf{x}_{f_{T_{n-1}}|D(i)} + o(i)\mathbf{1})\|_2^2. \quad (4)$$

In other words, one does collage coding based on the domain blocks with image intensities from the fixed point of step $n-1$. This method allows substantial PSNR improvements over collage coding. However, it has two drawbacks. First, there is no guarantee that the reconstruction error decreases after each step. Second, the procedure is time expensive because every step corresponds to a new encoding of the test image. To accelerate the procedure, Lu proposed to consider at each step only a portion of the range blocks, namely the range blocks R_i for which the ratio between the local collage error and the local reconstruction error $\|\mathbf{x}_{f^*|R_i} - \mathbf{x}_{f_{T_0}|R_i}\|_2^2$ is largest. Barthel and Voyé [3] saved time by fixing the range block - domain block association and updating only the scaling factor and the offset. Hamzaoui *et al.* [11] modified the method in a way to ensure that the reconstruction error is monotonically decreasing. This gives a local search [17] scheme, where the neighborhood of a feasible solution T_n is the set of n_R transformations obtained from T_n by modifying the fractal parameters of only a single range block according to rule (4). The resulting algorithm is as follows.

Local search algorithm

1. **Initialization**: Let M be a maximum number of trials. Set $n := 0$, $i := 0$, and $j := 0$. Find an initial feasible solution T_0 by collage coding. Let n_R be the number of range blocks in the partition.

2. Let $r := 1 + (i \bmod n_R)$. Determine for the range block R_r new fractal parameters by solving the minimization problem

$$\min_{(D(r),s(r),o(r),P(r))\in\mathcal{D}\times\mathcal{S}\times\mathcal{O}\times\mathcal{P}} \|\mathbf{x}_{f^*|R_r} - (s(r)P(r)A\mathbf{x}_{f_{T_n}|D(r)} + o(r)\mathbf{1})\|_2^2$$

with the least squares technique. Set $i := i + 1$.

3. Let T_c be the solution in the neighborhood of T_n obtained by changing the fractal parameters of range block R_r according to the result of Step 2. Compute the fixed point of T_c.

4. If $E(T_c) < E(T_n)$, set $T_{n+1} := T_c, n := n + 1, j := 0$. Otherwise set $j := j + 1$.

5. If ($i \leq M$ and $j < n_R$) go to Step 2. Otherwise stop.

When M is set to ∞, the algorithm stops at a local optimum.

The local search algorithm can be extended in a straightforward way to schemes with arbitrary partitions by searching in Step 2 for domain blocks in the appropriate domain pools. Note that our algorithm does not modify the image partition. Thus, a solution computed by the algorithm requires the same number of bits as the initial solution.

3. EXTENSIONS

We tried to determine a better local optimum by starting the algorithm with different initial solutions T_0. Unfortunately, the initial solution found by collage coding always provided the best local optimum.

In another attempt, we searched the neighborhood with a *steepest descent* technique where all solutions in the neighborhood of T_n were inspected, and the one providing the largest decrease in reconstruction error was selected. This approach

increased the complexity of the algorithm without providing better local optimal solutions.

In [11], we showed that the local search technique can improve the PSNR of the codes obtained with Fisher's quadtree scheme [9] by up to 0.8 dB. We now use the local search technique to improve codes computed by the efficient fractal scheme of [19, 13]. In this scheme, the image support is first partitioned into uniform *atomic* square range blocks. For each range block, the d sets of fractal parameters that yield the smallest collage error are determined. Then the two adjacent range blocks whose merging yields the least increase of the total collage error are iteratively merged until a partition into n_R range blocks is obtained. When two adjacent (parent) range blocks are merged, a set of candidate domain blocks is formed by appropriately extending the size of each of the $2d$ domain blocks corresponding to the parent range blocks and retaining the d ones that yield the smallest collage error. Table 1 gives the PSNR performance of the local search algorithm after $i = n_R$ trials and after convergence for three test images. In Step 2 of the algorithm, for each range block, the search for a new domain block was restricted to the corresponding final set of d domain blocks. We recall that the PSNR is defined for 8 bpp $n \times n$ images by

$$\text{PSNR} = 10 \log_{10} \left(\frac{255^2}{\frac{1}{n^2} \| f - \hat{f} \|_2^2} \right)$$

where f and $\hat{f}$ are the original image and the reconstructed image, respectively. In our experiments, we used the following settings. The atomic block size was equal to 8×8, the domain pool used for the initial code consisted of the set of 16×16 square blocks that uniformly partition the image support, d was equal to 50, and n_R to 1000. Figure 1 shows the original 512×512 Barbara image, together with the partition obtained with the code of [19]. The experiments show that the local search algorithm can improve the reconstruction fidelity of the coder [19] by up to 0.25 dB. Moreover, most of the gain was obtained after the first n_R trials. We

obtained similar PSNR gains for various values of d. Figure 2 shows the effects of varying the number of range blocks in the partition. Here two atomic block sizes were considered: 4×4 and 8×8.

TABLE 1

Reconstruction fidelity improvement for the fractal scheme of [19].

Image	n_R	Fractal scheme of [19] PSNR (dB)	Local search (n_R) PSNR (dB)	Local search (convergence) PSNR (dB)
512×512 Barbara	1000	24.02	24.25	24.27
512×512 Lenna	1000	29.21	29.39	29.39
512×512 Boat	1000	27.70	27.91	27.94

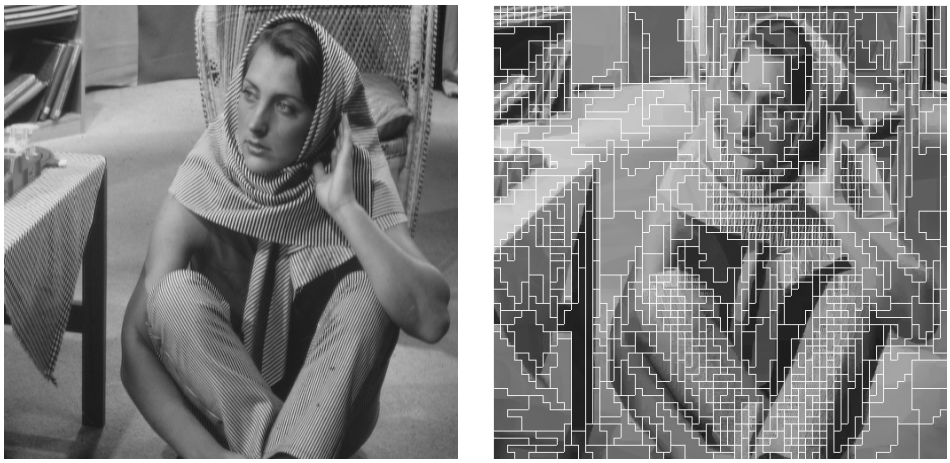


FIG. 1. Left: Original 512×512 Barbara image. Right: Partition into 1000 range blocks for an atomic block size of 8×8.

4. COMPLEXITY REDUCTION OF THE ALGORITHM

In this section, we present several techniques for accelerating the local search algorithm. The first two techniques were already used in [11].

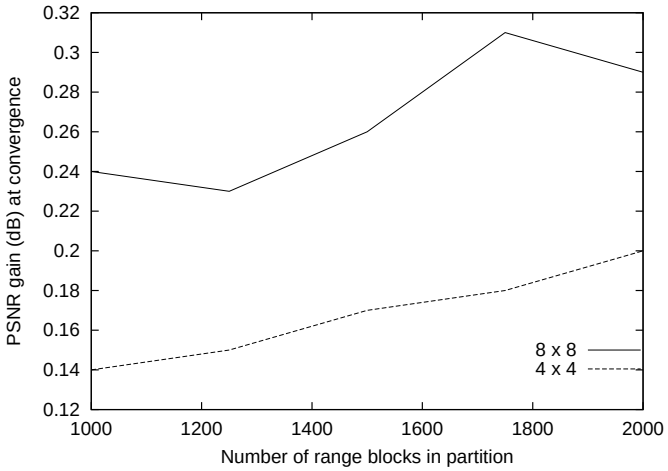


FIG. 2. PSNR gain versus number of range blocks in the partition for atomic block sizes 4×4 and 8×8 for the 512×512 Boat image.

4.1. Choice of starting point

The most time-consuming part of the local search algorithm is the computation in Step 3 of the fixed point $f_{T_c} = \lim_{k\to\infty} f_k$, where $\{f_k\}_{k\geq 0}$ is given by $f_{k+1} = T_c(f_k)$, and f_0 is an arbitrary initial image. By taking $f_0 = f_{T_n}$, we have fast convergence because f_{T_n} is close to f_{T_c}.

4.2. Pixel update

In Step 3, for the computation of the fixed point of T_c, we use the faster Gauss-Seidel like iteration [10] instead of the usual iteration. In this way, only one image array is stored, and the new pixel intensities are used as soon as they become available. For example, suppose that for $f_{k+1} = T_c(f_k)$, we have

$$f_{k+1}(i,j) = \frac{s}{4}(f_k(l,m) + f_k(l+1,m) + f_k(l,m+1) + f_k(l+1,m+1)) + o$$

where s is a scaling factor and o is an offset. Suppose also that the intensities of pixels $(l,m),(l+1,m),(l,m+1),(l+1,m+1)$ are computed before the pixel intensity of (i,j). Then with the iteration of [10], we get

$$f_{k+1}(i,j) = \frac{s}{4}(f_{k+1}(l,m) + f_{k+1}(l+1,m) + f_{k+1}(l,m+1) + f_{k+1}(l+1,m+1)) + o.$$

Note that the Gauss-Seidel like iteration converges to the fixed point of the usual iteration independently of the ordering of the pixels.

4.3. No computation of fixed point

In Step 2, if the new fractal parameters of range block R_r are equal to the current ones, then $T_c = T_n$. Thus, we just set $j := j + 1$ and go to Step 5.

4.4. Ordering of the range blocks

The local search algorithm is dependent on the ordering of the range blocks. In our previous publication [11], we considered the range blocks R_r, $r = 1, \ldots, n_R$ in the order in which they appear in the code. We have found out, however, that the reconstruction error decreases faster in the initial steps if the range blocks are ordered according to decreasing error $\|\mathbf{x}_{f^*|R_r} - \mathbf{x}_{f_{T_0}|R_r}\|_2^2$.

4.5. Decoding with dependence graph

The main idea for accelerating the local search algorithm exploits the fact that T_c and T_n differ only in the fractal parameters of range block R_r. Thus, in Step 3, if we start from the current fixed point f_{T_n} and apply the operator T_c once, then only the pixel intensities in R_r have to be updated, which avoids many unnecessary computations. If we now apply T_c to the first iterate $T_c(f_{T_n})$, then only the range blocks whose domain blocks overlap R_r have to be updated. Note that these range blocks may include R_r. This procedure is repeated until convergence to f_{T_c}. The decoding relation between the range blocks was studied by Domaszewicz and Vaishampayan [6] who introduced the notion of the *dependence graph* of the code, which is defined as follows. We say that a block $D \subset I$ *overlaps* a block $R \subset I$ if $D \cap R \neq \emptyset$. A range block R_j is called a *child* of a range block R_i if the domain block that encodes R_j overlaps R_i. The range block R_i is then called a *parent* of R_j. A *dependence graph* of the code of a transformation $T \in \mathcal{T}$ is a directed graph $\mathcal{G}(T) = (V, E)$ where each vertex $R_i \in V$ is a range block in the image partition $\mathcal{R}$,

and an ordered pair of vertices $(R_i, R_j) \in E$ if R_j is a child of R_i. Note that each range block has at least one parent, but not all range blocks have children.

EXAMPLE 4.1. Consider the image partition of Figure 3. Suppose that the domain pool consists of the following four domain blocks: $D_1 = R_1 \cup R_2 \cup R_3 \cup R_4$, $D_2 = R_5 \cup R_6 \cup R_7 \cup R_8$, $D_3 = R_9 \cup R_{10} \cup R_{11} \cup R_{12}$, $D_4 = R_{13} \cup R_{14} \cup R_{15} \cup R_{16}$. Suppose that the domain blocks encode the range blocks as in Table 2. Then the dependence graph of the code is given by Figure 4 (the figure was drawn with daVinci V2.1 [20]). Note that R_{13}, R_{14}, R_{15}, and R_{16} have no children because D_4 is not used in the encoding.

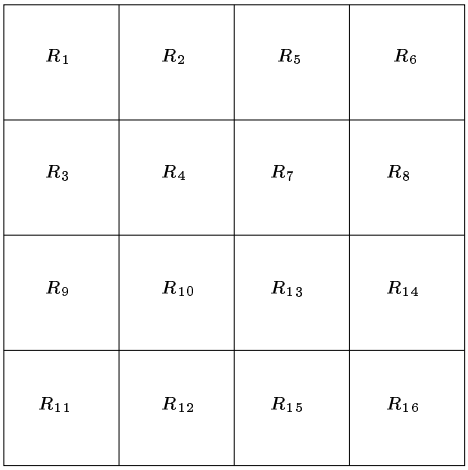


FIG. 3. Image partition into 16 square range blocks.

TABLE 2

Encoding of the range blocks by the domain blocks.

R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9	R_{10}	R_{11}	R_{12}	R_{13}	R_{14}	R_{15}	R_{16}
D_1	D_2	D_1	D_3	D_2	D_2	D_3	D_2	D_3	D_1	D_3	D_1	D_1	D_2	D_2	D_1

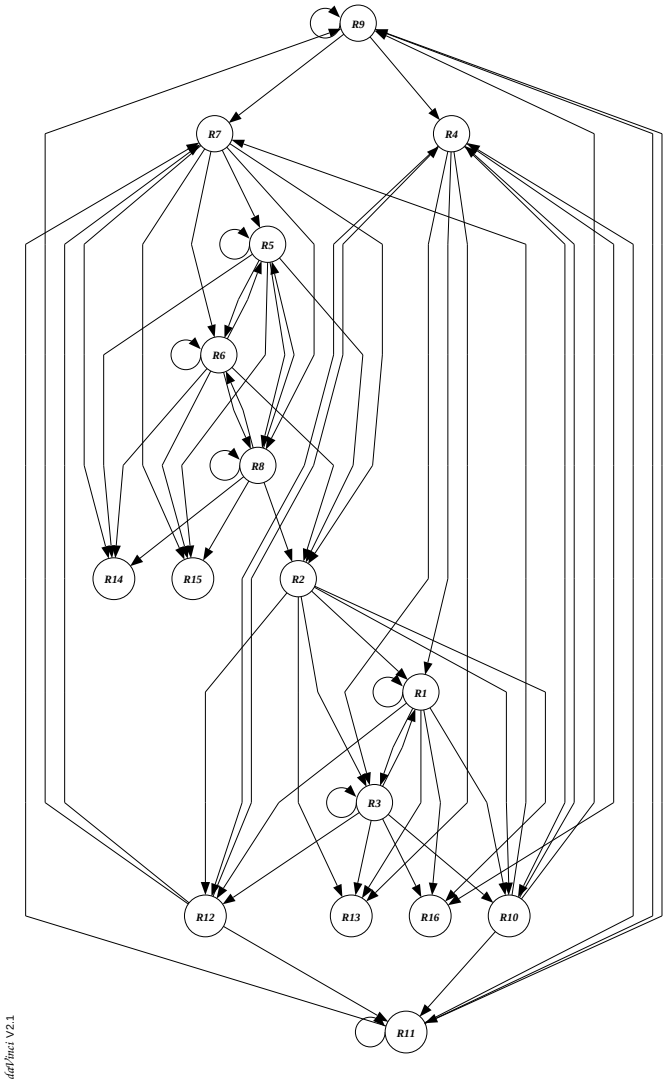


FIG. 4. Dependence graph of the code.

The iteration scheme $f_{T_n} \to T_c(f_{T_n}) \to T_c(T_c(f_{T_n})) \to \cdots \to f_{T_c}$ can be implemented as a breadth-first traversal of the dependence graph of the code of T_c, starting from vertex R_r. The first iteration $f_{T_n} \to T_c(f_{T_n})$ corresponds to visiting the root vertex R_r, and for $k \geq 2$, iteration k corresponds to visiting the children of all vertices visited at iteration $k-1$. Formally, if we denote by $\{L_k\}_{k\geq 1}$ the sequence of subsets of $\{R_1, \ldots, R_{n_R}\}$ given by $L_1 = \{R_r\}$ and

$L_{k+1} = \bigcup_{R_i \in L_k} \{R_j \in V : (R_i, R_j) \in E\}$, then at iteration k, we compute the pixel intensities of only the range blocks in L_k. Note that only vertices that are connected to R_r are visited. Note also that in contrast to typical breadth-first search a vertex may be visited many times. However, at a given iteration, a vertex needs to be visited only once.

In fact, when visiting a vertex, we use the newly computed pixel intensities as soon as they are available. In this situation, our breadth-first traversal of the dependence graph starting from vertex R_r corresponds to the iteration scheme

$$f_{k+1} = T_{c,i_k}(f_k), \; f_0 = f_{T_n}, \tag{5}$$

where T_{c,i_k} is a Gauss-Seidel like operator [10] where the range blocks in L_k are processed last. In the Appendix, we prove that iteration (5) converges to f_{T_c}.

The dependence graph is stored as an array of linked lists (adjacency lists), i.e., for each range block a linked list is kept, which holds all its children. Whenever the domain block that encodes range block R_r is changed, the dependence graph is updated as follows.

1. Determine the current parents of vertex R_r.
2. Go through the linked lists of these vertices and remove R_r from each list.
3. Determine the new parents of vertex R_r.
4. Insert R_r in the linked list of each new parent.

In example 4.1, suppose that $R_r = R_1$, and that R_r is now encoded by D_4. Then, the dependence graph is updated by removing R_1 from the children lists of R_1, R_2, R_3, and R_4 and inserting it in the lists of R_{13}, R_{14}, R_{15}, and R_{16}.

Note that if $E(T_c) > E(T_n)$, then the old dependence graph has to be restored. To find the parents of vertex R_r, one has to identify the range blocks that are overlapped by the domain block that encodes R_r. But since this operation may have to be done many times (to restore the old dependence graph or if the same

domain block is selected later), one can reduce computing time by saving in a preprocessing step for each domain block in the domain pool the list of range blocks that it overlaps.

4.6. Unchanged intensities in range and domain blocks

The least squares approach used in Step 2 of the algorithm requires the computation of the sum of the pixel intensities and the sum of the squared pixel intensities of the original image in a range block (see (1), (2), and (3)). Because the same range block may be considered many times, one can save computing time by storing these sums during the initialization (Step 1). Similarly, the same sums have to be computed for f_{T_n} in all downsampled domain blocks of the domain pool. But according to subsection 4.5, only a few range blocks have to be considered (Table 3). Thus, for $f_{T_{n+1}}$, we recompute these sums only for the domain blocks that overlap these range blocks. These domain blocks can be found efficiently by storing in a preprocessing step for each range block in the partition the list of the domain blocks that it overlaps. This can be done most conveniently during the determination of the range block lists (see the last paragraph of 4.5). Indeed, if domain block D overlaps range block R, then R overlaps D.

Finally, in Step 4, the test $E(T_c) < E(T_n)$ reduces to

$$\sum_{R_i \in \bigcup_{k=1}^{m} L_k} \|\mathbf{x}_{f^*|Ri} - \mathbf{x}_{f_m|R_i}\|_2^2 - \|\mathbf{x}_{f^*|Ri} - \mathbf{x}_{f_{T_n}|R_i}\|_2^2 < 0, \tag{6}$$

where m is the iteration step at which (5) converged.

4.7. Fast local search

1. **Initialization**: Let M be a maximum number of trials. Set $n := 0$, $i := 0$, and $j := 0$.

(i) Find an initial feasible solution T_0 by collage coding. Let n_R be the number of range blocks in the partition.

(ii) Compute and store the sum of the pixel intensities and the sum of the squared pixel intensities in each range block of the original image.

(iii) Determine $\mathcal{G}(T_0)$, the dependence graph of the code of T_0. Set $\mathcal{G} = \mathcal{G}(T_0)$.

(iv) For each domain block D of the domain pool, store the list of range blocks that are overlapped by D. For each range block R in the partition, store the list of domain blocks that R overlaps.

(v) Sort $R_1, \ldots, R_{n_R}$ according to decreasing error $\|\mathbf{x}_{f^*|R_r} - \mathbf{x}_{f_{T_0}|R_r}\|_2^2$.

2. Let $r := 1 + (i \bmod n_R)$. Determine for the range block R_r new fractal parameters by solving the minimization problem

$$\min_{(D(r),s(r),o(r),P(r))\in\mathcal{D}\times\mathcal{S}\times\mathcal{O}\times\mathcal{P}} \|\mathbf{x}_{f^*_{|R_r}} - (s(r)P(r)A\mathbf{x}_{f_{T_n|D(r)}} + o(r)\mathbf{1})\|_2^2$$

with the least squares technique. Set $i := i + 1$.

3. If the new fractal parameters are different from the current ones, construct a candidate transformation T_c by changing only the fractal parameters of range block R_r according to the result of Step 2. Otherwise, set $j := j + 1$ and go to step 7.

4. If the new domain block is different from the current one, update $\mathcal{G}$, i.e., set $\mathcal{G} = \mathcal{G}(T_c)$.

5. Use $\mathcal{G}$ to compute the fixed point of T_c.

6. If (6) is true, set $T_{n+1} := T_c, n := n + 1, j := 0$. Otherwise set $j := j + 1$ and $\mathcal{G} = \mathcal{G}(T_n)$.

7. If ($i \leq M$ and $j < n_R$), go to Step 2. Otherwise stop.

5. EXPERIMENTAL RESULTS

This section illustrates the relevance of the proposed acceleration techniques. The test image was the 8 bpp 512×512 Peppers image. Collage coding was done with Fisher's quadtree scheme [9]. The smallest range block size was 4×4, and the largest range block size was 32×32. For each range block size, the domain

pool consisted of the square blocks of the image support whose linear size is twice that of the range blocks and whose upper-left pixels are situated on locations (i, j), where $i \equiv 0 \pmod 4$ and $j \equiv 0 \pmod 4$. Thus, there were $n_{D_1} = 12769$ domain blocks of size 64×64, $n_{D_2} = 14641$ domain blocks of size 32×32, $n_{D_3} = 15625$ domain blocks of size 16×16, and $n_{D_4} = 16129$ domain blocks of size 8×8. We slightly modified Fisher's coder in a way that only one permutation, namely the identity, was used (thus $n_P = 1$). For each block size, the parameters n_s and n_o were equal to 32 and 128, respectively. Thus, 5 bits were allocated to the scaling factor, 7 bits to the offset, and 14 bits to the domain position. If the scaling factor was equal to zero, then the bits for the domain position were not sent to the decoder because they are redundant. Note that our local search algorithm may assign a nonzero scaling factor to a range block whose collage coding was with a zero scaling factor. Thus, to ensure that we do not increase the bit rate of the code, we do not modify the fractal parameters of such range blocks. Finally, the root mean square (rms) threshold [9] was equal to 12. With these settings, the image was partitioned into $n_R = 2254$ range blocks. For the computation of the fixed point of T_c, we stopped the iteration (5) when the rms error between two consecutive image iterates was less than 0.1. Figure 5 (a) shows the PSNR yielded by the solution T_n as a function of the number of range blocks considered (counter i in the algorithm). Figure 5 (b) shows the PSNR as a function of time for the same experiment. The CPU time was measured on an SGI O2 running an MIPS R10000 195 MHz processor and having a main memory size of 192 megabytes. The curves labeled "with" correspond to an implementation of the local search algorithm which includes all acceleration techniques. The curves labeled "without" correspond to a brute-force implementation which does not include these techniques. The curve labeled "[11]" corresponds to the implementation of reference [11], which uses only the acceleration techniques of 4.1 and 4.2. Note that in Figure 5 (a) only the effect of the differing ordering of the range blocks can be seen (see 4.4). Although

both curves eventually reached about the same quality, the new ordering allowed a steeper increase of the PSNR in the beginning. This is advantageous when the algorithm is stopped after a few steps. The initialization (collage coding) took 429 seconds and yielded a PSNR of 29.79 dB at a compression ratio of 34.66:1. The fast local search algorithm was able to increase the PSNR by 0.3 dB in about 175 seconds. The PSNR gain reached 0.5 dB after 113 more seconds. Table 3 illustrates the computation savings due to the techniques of 4.3 and 4.5. The first column gives the current number of range blocks considered by the algorithm (counter i). The second column gives the number of blocks with unchanged fractal parameters. The last column gives the average number of range blocks that were visited in the breadth-first traversal of the dependence graph during the computation of f_{T_c}. For example, after $i = n_R$ steps, 20333 vertices were visited. Thus, the average number was $\frac{20333}{2254-489} = 11.52$.

We now study the memory requirements of the local search algorithm. Using C conventions, we denote a structure of type node by (struct node), a pointer to such a structure by (struct node*), and the size of a data structure "data" by sizeof(data). Finally, we assume that our computer allocates four bytes of memory to a pointer.

Compared to collage coding, our algorithm required two additional image arrays corresponding respectively to the fixed point of the current solution T_n and to the fixed point of the candidate solution T_c. Moreover, the data structures used by the complexity reduction techniques of 4.4, 4.5, and 4.6 needed further extra-memory as follows.

To process the range blocks with the order explained in 4.4, we used a linked list of size $n_R \times$ sizeof(struct sequence) $= 2254 \times (2 + 4 + 4) = 22,540$ bytes, where sequence is a structure containing the index of a range block R_r, the error $\|\mathbf{x}_{f^*|R_r} - \mathbf{x}_{f_{T_0}|R_r}\|_2^2$, and a pointer to the next sequence structure.

The data structure for the dependence graph was an array of n_R linked lists, each of which contained the children of a range block. In $i = 2n_R$ iteration steps, the

number of children in each of the successive dependence graphs was always less than 11605 (the number of children changes every time we update the graph). Thus, the maximum memory space required was $n_R \times$ sizeof(struct childrennode*) $+ 11605 \times$ sizeof(struct childrennode) $= 2254 \times 4 + 11605 \times (2 + 4) = 78,646$ bytes, where childrennode is a structure containing the index of a range block and a pointer to the next childrennode.

Moreover, we needed $3n_R \times$ sizeof(char) $= 3 \times 2254 = 6,762$ bytes for three binary arrays of size n_R used in the breadth-first search. These arrays indicate at iteration k the range blocks in L_k, L_{k+1}, and $\bigcup_{1 \le i \le k} L_i$. Finally, for each domain block of the domain pool we stored an array that specifies the indices of the range blocks that overlap this domain block. The total number of these range blocks was equal to 729220. Thus, the memory space was $(n_{D_1} + n_{D_2} + n_{D_3} + n_{D_4}) \times$ sizeof(unsigned short int *) $+ (n_{D_1} + n_{D_2} + n_{D_3} + n_{D_4}) \times$ sizeof(char) $+ 729220 \times$ sizeof(unsigned short int) $= 1,754,260$ bytes. Here, for each domain block, we used a pointer to an array of range blocks and a character to indicate the length of the array.

In 4.6, the space used to store the sum of the pixel intensities and the sum of squared pixel intensities in the range blocks was equal to $2n_R \times$ sizeof(int) $= 18,032$ bytes. To identify the domain blocks whose pixel intensities have changed, we used an array of flags, which required $n_{D_1} + n_{D_2} + n_{D_3} + n_{D_4} = 59,164$ bytes. The space needed for the linked lists of domain blocks that overlap a range block was equal to $n_R \times$ sizeof(struct domnode*) $+ 729220 \times$ sizeof(struct domnode) $= 2254 \times 4 + 729220 \times (2 + 4) = 4,384,336$ bytes, where domnode is a structure giving the index of the domain block and a pointer to the next domnode.

The memory requirements are summarized in Table 4. Because our compiler allocated a multiple of four bytes to a structure, the actual extra space was equal to 8,334,186 bytes instead of 6,848,028 bytes. For example, sizeof(struct sequence) was equal to 12 and not 10 as in the calculation above.

A worst-case analysis of the space complexity of our fast implementation would be much more complicated and without practical interest. More important, for typical encodings, the memory space was less than 20 megabytes for high to medium compression ratios and less than 50 megabytes for very low compression ratios. An implementation of the fast local search algorithm for low memory computers should not store the lists of domain blocks (Step 1 (iv)). Also, the lists of range blocks (Step 1 (iv)) should not be stored. Instead, one determines the current and new parents of R_r in Step 4 and stores their indices in two temporary arrays, which are used in Step 6 to restore the current graph if (6) is false. If R_r is of size $2^n \times 2^n$, the maximum number of parents that R_r can have is equal to $(2^{n+1-m}+1)^2$, where $2^m \times 2^m$ is the size of the smallest range block in the partition. Thus, in our encoding, the two arrays required at most $2 \times (2^{5+1-2}+1)^2 \times$ sizeof(short int) $= 1,156$ bytes. At $i = 2n_R$, this low memory implementation used only $6,848,028 - 1,754,260 - 4,384,336 + 1,156 = 710,588$ bytes of memory space, but its running time was 36 seconds slower than that of the implementation using the lists.

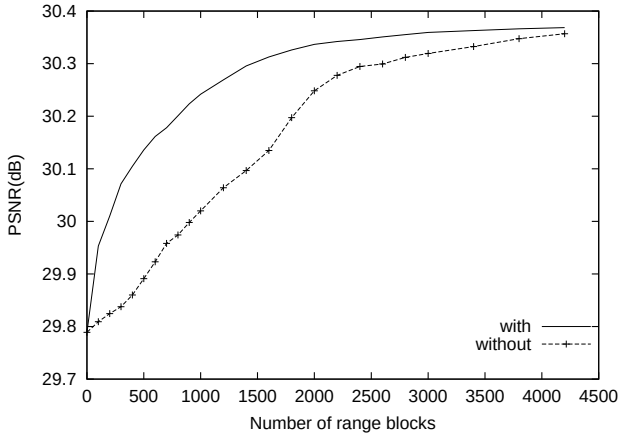
TABLE 3

Computation savings for the local search algorithm.

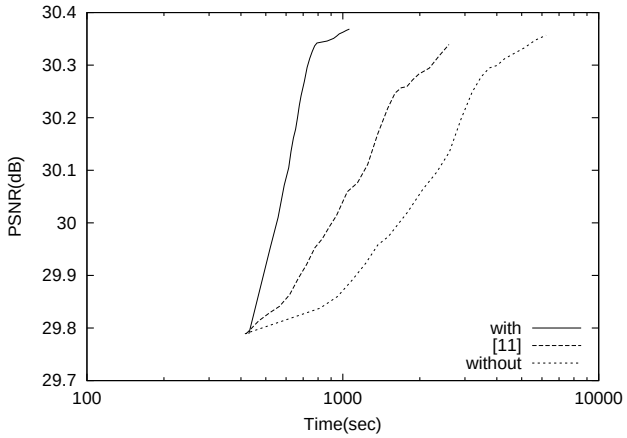
Number i of range blocks	Number of range blocks with unchanged parameters (4.3)	Average number of processed range blocks (4.5)
$n_R = 2254$	489 (21.69 %)	11.52
$2n_R = 4508$	1943 (43.10 %)	13.63

6. CONCLUSION

Finding an optimal fractal image code is an intractable problem [18]. Hamzaoui *et al.* introduced a local search algorithm that iteratively improves an initial solution [11]. The PSNR gain is typically between 0.2 and 0.8 dB when the initial code is



(a)



(b)

FIG. 5. (a) PSNR versus number of range blocks for the 512 × 512 Peppers image. (b) PSNR versus time (logarithmic scale) for the 512 × 512 Peppers image. The curves labeled "with" correspond to an implementation of the algorithm including the acceleration techniques. "[11]" corresponds to an implementation with only the first two accelerations techniques. The curves labeled "without" correspond to a brute-force implementation.

computed with a widely used quadtree coder [9]. Moreover, for some images, the visual quality is also better. Two important questions had to be answered. First, is local search successful with fractal schemes that are based on other partitions?

TABLE 4

Memory requirements for the 512 × 512 Peppers image.

Data structures	Bytes
two additional image arrays	524,288
ordering of range blocks (4.4)	22,540
dependence graph (4.5)	78,646
breadth-first search (4.5)	6,762
lists of range block (4.5)	1,754,260
lists of domain blocks (4.6)	4,384,336
intensity sums for range blocks (4.6)	18,032
unchanged domain blocks (4.6)	59,164
Total	6,848,028

Second, is it possible to reduce the complexity of the algorithm? This paper gives a positive answer to both questions.

We showed that local search is also useful for a state-of-the-art fractal scheme that yields highly adaptive partitions obtained with region growing [19, 13]. However, generally, the PSNR improvement was not as important as for the quadtree scheme. This may indicate that the scheme of [19, 13] finds solutions whose fixed points are close to that of the optimal solution. Next, we provided an efficient implementation of the local search algorithm. This allowed us to significantly improve our initial solution after a small amount of time. The basic idea is to see that when the code parameters of only one range block are modified, then the new fixed point differs from the current one in only a few range blocks, which are given by the dependence graph of the code. The price for the speed-up of the algorithm was additional space needed in particular for various data structures associated to the dependence graph.

A slower alternative implementation reduced these extra memory requirements and was still much faster than the original implementation.

Unfortunately, our local search technique finds only a locally optimal solution, and we are not able to say how far this solution is from the global one. Thus, the fundamental question of the performance limit of fractal image compression remains an open question. On the other hand, our solution is locally optimal with respect to *our* predefined neighborhood function. One topic for future research could be the selection of a better neighborhood. Finally, it may be interesting to use more sophisticated local search strategies that try to escape local minima by accepting neighbors that increase the cost function. These algorithms, which include simulated annealing and tabu search, have been successfully applied to many NP-hard problems [1].

APPENDIX

Let $T_c \in \mathcal{T}$ be the candidate transformation, and let $\mathcal{T}_c = \{T_{c,1}, \ldots, T_{c,M}\}$, $M \leq N^2!$, be the set of all possible Gauss-Seidel like transformations derived from T_c by a reordering of the pixels. Let $i_k \in \{1, \ldots, M\}$. Then the sequence of images $\{f_k\}_{k \geq 0}$ given by $f_{k+1} = T_{c,i_k}(f_k)$ converges to f_{T_c}.

Proof. For all $i = 1, \ldots, M$ we know that $T_{c,i}$ is a contraction in the l_∞ metric with fixed point f_{T_c} [10]. Let t_i denote the contraction factor of $T_{c,i}$ in l_∞. Then

$$\begin{aligned} \|f_k - f_{T_c}\|_\infty &= \|T_{c,i_{k-1}}(f_{k-1}) - T_{c,i_{k-1}}(f_{T_c})\|_\infty \\ &\leq t_{i_k} \|f_{k-1} - f_{T_c}\|_\infty \\ &\leq t_{i_k} t_{i_{k-1}} \cdots t_{i_1} \|f_0 - f_{T_c}\|_\infty \\ &\leq (\max_{1 \leq i \leq M} t_i)^k \|f_0 - f_{T_c}\|_\infty . \end{aligned}$$

Thus, since $(\max_{1 \leq i \leq M} t_i) < 1$, we have $\lim_{k \to \infty} f_k = f_{T_c}$. ■

ACKNOWLEDGMENT

We thank Giang Ky Nguyen for implementing the algorithm that applies the local search algorithm to the coder of [19].

REFERENCES

1. Aarts, E. H. L., Lenstra, J. K. (eds.), *Local Search in Combinatorial Optimization,* Wiley, 1997.

2. Barnsley, M., Hurd, L., *Fractal Image Compression,* AK Peters, Wellesley, 1993.

3. Barthel, K. U., Voyé, T., *Adaptive fractal image coding in the frequency domain,* in: *Proc. Int. Workshop on Image Processing: Theory, Methodology, Systems and Applications,* Budapest, June 1994.

4. Domaszewicz, J., Vaishampayan, V. A., *Structural limitations of self-affine and partially self-affine fractal compression,* in: *Proc. SPIE Visual Communications and Image Processing,* Vol. 2094, pp. 1498–1504, 1993.

5. Domaszewicz, J., Vaishampayan, V. A., *Iterative collage coding for fractal compression,* in: *Proc. ICIP-94 IEEE Int. Conf. on Image Processing,* Austin, Texas, Nov. 1994.

6. Domaszewicz, J., Vaishampayan, V. A., *Graph-theoretical analysis of the fractal transform,* in: *Proc. ICASSP-1995 IEEE Int. Conf. on Acoustics, Speech and Signal Processing,* Vol. 4, Detroit, 1995.

7. Dudbridge, F., Fisher, Y., *Attractor optimization in fractal image encoding,* in: *Proc. of the Conference Fractals in Engineering,* Arcachon, June 1997.

8. Fisher, Y., *Fractal Image Compression — Theory and Application,* Springer-Verlag, New York, 1994.

9. Fisher, Y., *Fractal image compression with quadtrees,* in: *Fractal Image Compression — Theory and Application,* Y. Fisher (ed.), Springer-Verlag, New York, 1994.

10. Hamzaoui, R., *Fast iterative methods for fractal image compression,* Journal of Mathematical Imaging and Vision 11,2 (1999) 147–159.

11. Hamzaoui, R., Hartenstein, H., Saupe, D., *Local iterative improvement of fractal image codes,* Image and Vision Computing 18,6-7 (2000) 565–568.

12. Hartenstein, H., *Topics in Fractal Image Compression and Near-Lossless Image Coding,* Doctoral Dissertation, University of Freiburg, 1998.

13. Hartenstein, H., Ruhl, M., Saupe, D., *Region-based fractal image compression,* IEEE Transactions on Image Processing 9,7 (2000) 1171–1184.

14. Hürtgen, B., *Performance bounds for fractal coding,* in: *Proc. ICASSP-1995 IEEE Int. Conf. on Acoustics, Speech and Signal Processing,* Vol. 4, Detroit, 1995.

15. Jacquin, A. E., *Image coding based on a fractal theory of iterated contractive image transformations,* IEEE Trans. Image Processing 1 (1992) 18–30.

16. Lu, N., *Fractal Imaging,* Academic Press, 1997.

17. Papadimitriou, C. H., Steiglitz, K., *Combinatorial Optimization: Algorithms and Complexity,* Dover, 1998.

18. Ruhl, M., Hartenstein, H., *Optimal fractal coding is NP-hard,* in: *Proc. DCC'97 Data Compression Conference,* J. A. Storer and M. Cohn (eds.), IEEE Comp. Soc. Press, March 1997.

19. Ruhl, M., Hartenstein, H., Saupe, D., *Adaptive partitionings for fractal image compression,* in: *Proc. ICIP-97 IEEE Int. Conf. on Image Processing,* Vol. II, pp. 310–313, Santa Barbara, California, Oct. 1997.

20. daVinci V2.1, Computer Science Department, University of Bremen, Germany.

21. Vrscay, E. R., Saupe, D., *Can one break the "collage barrier" in fractal image coding,* in: *Fractals: Theory and Applications in Engineering,* M. Dekking, J. L. Vehel, E. Lutton and C. Tricot (eds.), pp. 307–323, Springer-Verlag, London, 1999.

22. Wohlberg, B. E., de Jager G., *A review of the fractal image coding literature,* IEEE Trans. Image Processing 8,12 (1999) 1716–1729.