RJ 10214 (A0108-001) August 1, 2001
Computer Science

IBM Research Report

Multidimensional Rational Approximations with an Application to Image Compression

Jennifer Q. Trelewicz
IBM Research Division
Almaden Research Center
650 Harry Road
San Jose, CA 95120

Multidimensional Rational Approximations with an Application to Image Compression

J. Q. Trelewicz
IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120 USA
trelewic@us.ibm.com

July 30, 2001

Abstract

I present two methods for the calculation of simultaneous rational approximations with specific characteristics related to computation. The methods are intended for computationally efficient calculation of linear transforms for image compression; e.g., the Discrete Cosine Transform (DCT) or certain wavelet transforms. The rational approximations must have controlled precision (i.e., bits required for representation), and the numerators must use a minimum number of powers of two over the coefficient vector (i.e., the representation must have minimum "length"), to facilitate efficient multiplierless implementation.

The first method uses successive approximation of the coefficients in a suboptimal algorithm with low computational complexity to achieve low-length representations. The convergence of this method is analyzed, along with the closeness of the approximations. The suitability of this method for computational implementation, along with the quality of its representations, are evaluated in the context of Jacobi-Perron and Furtwängler type algorithms, which are shown to have shortcomings for this application. While the method is shown to provide low-length representations, it is not guaranteed to produce the lowest-length representation. However, the method is shown to be suitable for real-time computation environments.

The second method, which finds the lowest-length representation, requires significantly more computation, but is suitable for offline computation of representations. This method employs search tree pruning and, at each branch in the search tree, provides a lower bound for the representation length.

1 Introduction

Consider the d-tuple $\underline{x} \in [0,1]^d$. We are interested in simultaneous rational approximations for $\underline{x}$; i.e., rational representations of the form $\underline{p}/q \equiv (p_1/q, \ldots, p_d/q)$, with $p_1, \ldots, p_d, q \in \mathbf{N}$. Let $\epsilon > 0$ be given. Put $R_\epsilon = \{\underline{p}/q : \|\underline{x} - \underline{p}/q\| < \epsilon\}$, and $R_{\epsilon,m} \subset R_\epsilon$ such that $q < 2^m$. We are interested in finding the smallest such $m \in \mathbf{N}$ such that $R_{\epsilon,m} \neq \emptyset$, since m determines the number of bits required for implementing $\underline{p}$ in a microprocessor or hardware device.

These simultaneous rational representations have applications in numerical signal processing algorithms. For example, in image processing, the Discrete Cosine Transform (DCT) may be implemented in a conventional embedded microprocessor or in hardware (FPGA or ASIC). Integer DCT (IDCT) takes less power or fewer cycles than fixed-point or floating-point DCT in a number of microprocessors. Additionally, an integer arithmetic logic unit often takes fewer gates in an FPGA or ASIC than a floating-point unit. The importance of IDCTs is discussed in [1] and [2]. Furthermore, in a number of popular, cost-effective embedded microprocessors (for example, the Intel 80186), the multiply operation takes significantly longer than addition, subtraction, and shift operations. In these processors, if real-time execution of the DCT is required, performance improvements can be realized by eliminating multiplies from the operation, and replacing these with addition/subtraction and shifts. Furthermore, hardware implementations in FPGAs or ASICs benefit from multiplierless implementation to save hardware real estate. A method for generating multiplierless implementations of linear transforms is discussed in [3], while performance improvements utilizing multiplierless operations in software are discussed in

[4]. In these types of implementation q is absorbed into the quantization operation, and $\underline{p}$ becomes the integer transform coefficients that lend themselves to multiplierless implementation.

Multiplierless implementation furthermore benefits from minimizing $L(\underline{p})$, where L is defined as follows. Put

$$\mathcal{P} = \left\{ \sum_{\mu=0}^{m-1} \eta_{j,\mu} 2^{\mu} : \eta_{j,\mu} \in \{\pm 1, 0\}, m \in \mathbf{N} \right\}.$$

An equivalence class $\langle p \rangle$ in $\mathcal{P}$ contains the set of all polynomials with integer value equal to that of p. Consider $j \in 1..d$, and $p_j \in \mathcal{P}$ (where the notation $n_1..n_2$ is used to represent $[n_1, n_2] \cap \mathbf{Z}$):

$$p_j = \sum_{\mu=0}^{m-1} \eta_{j,\mu} 2^{\mu}.$$

$L(\underline{p}) = \# \{\mu \in 0..m-1 | \exists j \in 1..d : \eta_{j,\mu} \neq 0\}$. L is a metric on the collection of $\underline{p}$, as shown in Lemma 7.2 in the Appendix.

Note that the polynomial for p_j is not unique; e.g., $7 = 2^3 - 2^0 = 2^2 + 2^1 + 2^0$. A "run" occurs in p_j from μ_1 to μ_2 when $\exists \mu_1, \mu_2 \in 0..m-2$ such that $\eta_{\mu} \neq 0 \; \forall \mu \in \mu_1..\mu_2$. Thus, $L(\underline{p})$ may be reduced by replacing runs in $\underline{p}$ with alternate polynomials.

Lemma 1.1 *Consider a run $p_j = \sum_{\mu=\mu_1}^{\mu_2} 2^{\mu}$. Two polynomials which can replace this run to potentially reduce $L(\underline{p})$ are $\sum_{\mu=\mu_1}^{\mu_2} 2^{\mu}$ (called form F_0) and $2^{\mu_2+1} - 2^{\mu_1}$ (called form F_1). Furthermore, no other polynomial can reduce $L(\underline{p})$ more than forms F_0 and F_1.*

Proof: Since $\sum_{\mu=\mu_1}^{\mu_2} 2^{\mu} = 2^{\mu_2+1} - 2^{\mu_1}$, it suffices to show that no other equivalent polynomial can reduce $L(\underline{p})$. Consider such a polynomial $p'_j = \sum_{\mu=\mu_1}^{\mu_2} \eta_{\mu} 2^{\mu}$. By Lemma 7.1, all $p'_j \in \langle p_j \rangle \setminus \{p_j\}$ are in the form

$$p'_j = 2^{\mu_4} - \sum_{\mu=\mu_2+1}^{\mu_4-1} 2^{\mu} - 2^{\mu_3} + \sum_{\mu=\mu_1}^{\mu_3-1} 2^{\mu}.$$

Since $\eta_{\mu_2+1} \neq 0$ regardless of μ_4, and $\eta_{\mu_1} \neq 0$ regardless of μ_3, μ_4 is taken as μ_2+1 and μ_3 is taken as μ_1 to minimize the terms in p'_j. The only other equivalent polynomial is the original form p_j, the only form in which 2^{μ_2+1} has zero coefficient. $\square$

2 Known algorithms

Algorithms for multidimensional continued fractions are well known. However, many of these algorithms display exponential growth in the denominator; e.g., the Furtwängler algorithm [5], and the Ordered Jacobi-Perron algorithm (OJPA) [6]. Exponential growth of the denominator is unsuitable for the multiplierless transform implementations described in this manuscript, where computational complexity and/or integer precision must be controlled. Furthermore, these algorithms do not directly address the structure of the simultaneous representations for the numerators, as are needed for computational purposes. Specifically, these algorithms do not create simultaneous representations with minimum length $L(\underline{p})$. The result of the OJPA algorithm is compared to the other algorithms in Table 2, Section 5.

Alternatively, exhaustive search trees can be employed, where for each value of q within the desired precision, all possible representations for each $\underline{p}$ within the ϵ bound may be explored, and the resulting $L(\underline{p})$ tracked. However, the inefficiency and computational complexity of this approach is clear. For example, on a Pentium III computer, the brute force algorithm took days to find 10-bit solutions, and required unreasonable amounts of memory for construction of its search trees.

3 Single-pass Fast Approximation

It is known that given a continued fraction with the terms $[\chi_1; \chi_2, \chi_3, ..., \chi_n]$, the numerator p_i and denominator q_i of the ith convergent are defined for all $i \geq 1$ by the recursive definition

$$\begin{aligned} p_i &= \chi_i p_{i-1} + p_{i-2} \\ q_i &= \chi_i q_{i-1} + q_{i-2} \end{aligned}$$

where we define $p_{-1} = 0$, $p_0 = 1$, and $q_{-1} = q_0 = 1$. The algorithm described in this section uses as its basis a modification of this form in creating a fast algorithm that provides reasonable sub-optimal simultaneous rational representations for a vector $\underline{x} \in [0,1]^d$.

The algorithm is designed with the following restrictions:

- Since a simultaneous rational representation is desired, q_i must be the same for each element in the ith approximation to $\underline{x}$;
- To keep $L(\underline{p})$ small, the term added to q_i to obtain q_{i+1} must be chosen so that the resulting $p_{i+1} - p_i$ has few or no runs.

To minimize the number of runs, χ_i is taken to be a power of 2 at each iteration. Thus, if p_i is found with few or no runs, p_{i+1} will retain that property. The ith estimate (convergent) of element x_j is $p_{j,i}/q_i$, and the remainder is defined as $r_{j,i} = x_j - p_{j,i}/q_i$. Where $[\cdot]$ is the nearest integer operation,

$$a_{j,i} = \begin{cases} 0, & |r_{j,i}| < \delta \\ [1/|r_{j,i}|], & |r_{j,i}| \geq \delta \end{cases}.$$

In a practical implementation, δ may be taken, for example, as 10^{-5}, since the data are not rescaled at each pass. This is appropriate for an image processing application, where the processor on which the algorithm will be executed has a finite number of bits available in a register. If $\max_j |r_{j,i}| \leq \delta$, then no more iterations are used.

Otherwise, the smallest $a_{j,i}$ is used in subsequent processing so that the update to q_i may be chosen to reduce the error on the convergent resulting in the largest remainder. The result of this approach is rapid reduction of approximation error with each iteration i. Furthermore, as shown in Lemma 7.3, the result is that $p_{i+1} - p_i$ contains runs aligned to increase $L(\underline{p})$ very little at each iteration. Take $J_i = \arg\min_j a_{j,i}$. Put $b_i = \lceil \log_2 a_{J_i,i} \rceil$, the number of bits required for representing $a_{J_i,i}$. We put $q_0 = a_{J_0,0}$ and $q_{i+1} = q_i 2^{b_i} + 2^{b_i - 1}$, since $[(q2^n + r)/2^n] = q$ iff $|r| < 2^{n-1}$. This condition would ensure that q_{i+1} rounds to q_i when the bits of representation are reduced, for consistency between iterations; however, the required term $2^{b_i-1} - 1$ would destroy the runs structure of p_i. For this reason, $2^{b_i - 1}$ is used instead. The adjusted remainder $r'_{j,i} = x_j - p_{j,i}2^{b_i}/q_{i+1}$ is used to avoid roundoff error. Put $\Delta_{j,i} = \psi(r'_{j,i} q_{i+1})$, where $\psi(y)$ is defined as follows:

- If $y = 0$, then $\psi(y) = 0$.
- If $y \neq 0$ and $\nexists h \in \mathbf{Z}$ such that $|y| = 2^h + 2^{h-1}$, $\psi(y) = \text{signum}(y) 2^{\arg\min_h ||y| - 2^h|}$.
- Otherwise, $|y| = 2^h + 2^{h-1}$, and $\psi(y) = \text{signum}(y) 2^{h'}$, where $h' \in \{h, h+1\}$ is chosen to minimize $L(\underline{p})$.

Put $p_{j,0} = \Delta_{j,0}$, and $p_{j,i+1} = p_{j,i} 2^{b_i} + \Delta_{j,i}$.

In practical implementations, there may be a limit on the amount of precision actually needed, and this algorithm may generate more precision in the simultaneous estimates than is needed. The precision can be backed off trivially. If the maximum estimate error was underrun, it is possible that the solution can be implemented with less precision within the same error bound. This check needs to be done starting with the smallest powers of 2 in the numerator representations.

Convergence in ℓ_1 is considered for $\underline{p}_i$ (where $\underline{p}_i$ is compactly supported in ℓ_1), because of the structure of the linear transforms in which such multiplierless implementations may be used. Specifically, with image samples $y_j \in -255..255$ and error e_j on each coefficient, the worst-case error on a DCT coefficient is $255 \sum_j |e_j|$.

Theorem 3.1 *If $\delta = 0$, $\underline{p}_i/q_i \to \underline{x}$ in ℓ_1.*

Proof: Consider $i \geq 1$. Note that

$$q_{i+1} = (2q_i + 1)2^{b_i - 1} \geq q_i 2^{b_i}.$$

Put $t = \log_2(3/2) \in (0,1)$. Then

$$\psi(y)\text{signum}(y) \in \left\{ 2^{\lfloor \log_2 |y| \rfloor}, 2^{\lceil \log_2 |y| \rceil} \right\},$$

so that

$$|\psi(y) - y| \leq \max\left\{ |y|(2^t - 1), |y|(2^{1-t} - 1) \right\} = |y|(2^t - 1).$$

Therefore,

$$\begin{aligned} \left| x_j - \frac{p_{j,i+1}}{q_{i+1}} \right| &= \frac{1}{q_{i+1}} \left| q_{i+1} x_j - \left(p_{j,i} 2^{b_i} + \psi(r'_{j,i} q_{i+1}) \right) \right| \\ &= \frac{1}{q_{i+1}} \left| \left(x_j q_{i+1} - p_{j,i} 2^{b_i} \right) - \psi\left(x_j q_{i+1} - p_{j,i} 2^{b_i} \right) \right| \\ &\leq \frac{2^t - 1}{q_{i+1}} \left| x_j q_{i+1} - p_{j,i} 2^{b_i} \right| \leq \frac{2^t - 1}{2^{b_i} q_i} \left(\left| x_j 2^{b_i} q_i - p_{j,i} 2^{b_i} \right| + |x_j| 2^{b_i - 1} \right) \\ &\leq \left(2^t - 1 \right) \left| x_j - \frac{p_{j,i}}{q_i} \right| + \frac{(2^t - 1)|x_j|}{2q_i}. \end{aligned}$$

Since q_i is an increasing sequence and $(2^t - 1) \in (0,1)$, for $\epsilon' > 0$ and large enough i,

$$\left| x_j - \frac{p_{j,i+1}}{q_{i+1}} \right| \leq \left(2^t - 1 \right)^{i+1} \left| x_j - \frac{p_{j,0}}{q_0} \right| + \frac{|x_j|}{2} \sum_{\mu=0}^{i-1} q_\mu^{-1} (2^t - 1)^{i-\mu} \leq \left(2^t - 1 \right)^{i+1} \left| x_j - \frac{p_{j,0}}{q_0} \right| + \epsilon'.$$

It follows that $\ell_1(\underline{x} - \underline{p}_i/q_{i+1}) \to 0$. $\square$

4 Optimal Approximation

The optimal approximation is defined as that which gives the smallest possible $L(\underline{p})$ within the error bound ϵ. To avoid the computational difficulties of exhaustive search, the problem is mapped into efficient searching of a set of decision trees, $\{T(p_j) : j \in 1..d\}$, for each $\underline{p}/q$, where q takes each value from 1 to $2^n - 1$, the maximum desired denominator, determined by the application precision. Since each value of q must be searched, this method is best-suited for offline calculation, rather than real-time calculation. However, with the search tree pruning methods described in this section, the algorithm can find optimal representations in tractable periods of time; e.g., the 10-bit approximation mentioned in Section 2 takes only ten minutes on the same computer.

Given $j \in 1..d$ and $\underline{p}$, the construction of $T(p_j)$ proceeds as follows: the canonical representation $p_j = \sum_{\mu=0}^{m-1} \eta_{j,\mu} 2^\mu$ with $\eta_{j,\mu} \in \{0,1\}$ is mapped into $\{\alpha, A, \beta, B, \gamma, \Gamma\}^m$ according to mapping $S(p_j) = \{s_{j,\mu}\}$, where $S(p_j)$ is described in Table 1, defining $s_{j,-1} = \alpha$. A secondary run is defined as a run that exists only if a run immediately to its right is converted to form F_1.

Finally, $s_{j,\mu} = \beta$ is changed to β' and $s_{j,\mu} = \gamma$ is changed to γ' if $s_{j,\mu+1} = \Gamma$.

Put $\mathcal{A} = \{\mu \in 0..m-1 | \exists j \in 1..d : s_{j,\mu} = A\}$. Each search tree $T(p_j)$ is pruned by operation K described here, with adjacent symbols updated for consistency with $S(p_j)$:

- By Lemma 7.4, if $\exists \mu_1, \mu_2$ such that $\mu_2 \in \mathcal{A}$, $s_{j,\mu_2} \in \{\beta, \beta'\}$, and $s_{j,\mu} = B\ \forall \mu \in \mu_1..\mu_2 - 1$, then the tree $T(p_j)$ is pruned by converting the run to form F_1.

- By Lemma 7.5, if $\exists \mu_1, \mu_2$ such that $s_{j,\mu_2} = \beta$ and $s_{j,\mu} = B\ \forall \mu \in \mu_1..\mu_2 - 1 \subset \mathcal{A}$, then the tree $T(p_j)$ is pruned by converting the run to form F_0. This is called a "masked run" at $s_{j,\mu}$. If the run is not masked, it is called "unmasked".

Table 1: $S(p_j)$.

$s_{j,\mu-1}$	Interpretation of symbol	$s_{j,\mu}$ when $\eta_{j,\mu}=0$	$s_{j,\mu}$ when $\eta_{j,\mu}=1$
α	0 regardless of runs	α	A
A	±1 regardless of runs	α	B
β	0 "left" of run	α	Γ
B	±1 inside run	β	B
γ	0 "left" of secondary run	α	Γ
Γ	±1 inside secondary run	γ	B

- By Lemma 7.6, if $\exists\mu \in \mathcal{A}$ with $s_{j,\mu} = \Gamma$ and $s_{j,\mu+1} = \gamma$, then the tree $T(p_j)$ is pruned by setting $s_{j,\mu} = A$.

Put

$$\mathcal{B} = \{\mu \in 1..m-1 \setminus \mathcal{A} | \exists j \in 1..d : (s_{j,\mu} = \beta, s_{j,\mu-1} = B) \vee (s_{j,\mu} = \gamma, s_{j,\mu-1} = \Gamma)\}.$$

A lower bound estimate for $L(\underline{p})$ given the current tree or tree branch may be obtained from $\#\mathcal{A} \cup \mathcal{B}$, so that $\{T(p_j)\}$ or the current tree branch is not searched if $L(\underline{p})$ cannot be improved by this representation.

Since a decision branch in $T(p_j)$ at μ_1 can only affects symbols $s_{j,\mu}$ for $\mu > \mu_1$, $T(p_j)$ is searched for increasing μ. Suppose $\exists\mu$ such that $s_{j,\mu} \in \{\beta, \beta'\}$. Consider the smallest such μ. Then the trees $T(p_j)$ are pruned by converting all unmasked runs at $s_{j,\mu}$ to form F_1, by Lemma 7.7. Each remaining symbol β' (which has masked run, because of K) thus marks a decision point in the tree, since the run may be converted to form F_0 or form F_1. By Lemma 1.1, each β' symbol marks a twofold split in the tree, between the two run representations. At the decision point, the run is replaced with $\{\alpha, A\}$ according to the decision, subsequent symbols are updated according to $S(p_j)$, and the procedure is recursed. By Lemma 7.8, this search procedure results in a path through the tree containing only $\{\alpha, A\}$ symbols, which may be converted back to $\{\pm1, 0\}$ through Lemma 7.9.

Theorem 4.1 *The algorithm described in this section delivers the optimum solution within R_ϵ.*

Proof: It is simple to verify that S is invertible on the canonical representation. Lemmas 7.4-7.6 show that K only prunes branches from the decision tree that cannot provide lower $L(\underline{p})$. Lemmas 7.7,7.8, and 7.9 show that further pruning is only performed on the tree when those branches cannot lower $L(\underline{p})$. $\square$

5 Computational Results

The exhaustive search algorithm without search tree pruning was unable to exceed 6 bits of q in some tested cases because of the memory requirements for storing the search trees and the intractable computational complexity associated with the computation. Therefore, the timing for exhaustive search is not discussed further in this section.

Table 2 shows the values returned by the algorithms of Section 4 ("Opt") and Section 3 ("Fast"), for the specified number of bits of q for the odd coefficients of the DCT:

$$\underline{x} = \{C_1, C_3, C_5, C_7\} = \{0.980785, 0.831470, 0.555571, 0.195091\}.$$

(The DCT notation is the same as that used in [7].) Error in the table is given as a fraction of $\|\underline{x}\|$ (ℓ_1 norm). All representations in the table were obtained with $\epsilon = 0.01\|\underline{x}\|$. Where runtime shows as 0 seconds in the table, the algorithm ran in finer resolution than the "clock()" subroutine could measure.

Table 2 also shows the simultaneous representation for the same $\underline{x}$, using the OJPA. It should be noted that the OJPA only returns binary values, which are shown in the table converted to the lowest possible $L(\underline{p})$ using the optimal algorithm from Section 4.

It is illustrative to note that Opt for $n = 6$ requires 4 bits, since the 3-bit solution of the form similar to Opt for $n = 8$ has an error larger than ϵ. It should also be noted that Opt for $n = 4$ has an error larger than ϵ.

Table 2: $\underline{p}/q$ for OJPA, Opt, Fast algorithm

	n	$\underline{p}/q$	$L(\underline{p})$	error	sec
OJPA		$(2^3+2^1+2^0, 2^3+2^0, 2^3-2^1, 2^1)/11$	3	2.18×10^{-2}	
Opt	4	$(2^3+2^1+2^0, 2^3+2^0, 2^3-2^1, 2^1)/11$	3	2.18×10^{-2}	0
Fast		$(2^3+2^2+2^0, 2^3+2^1+2^0, 2^3-2^1+2^0, 2^1)/13$	4	9.54×10^{-3}	0.01
OJPA		$(2^5+2^4-2^1, 2^5+2^3-2^0, 2^4+2^3+2^1, 2^3+2^0)/47$	5	3.79×10^{-3}	
Opt	6	$(2^5+2^3, 2^5+2^1, 2^5-2^3-2^0, 2^3)/41$	4	4.99×10^{-3}	0
Fast		$(2^5+2^4+2^2, 2^5+2^3+2^2, 2^5-2^3+2^2, 2^3)/52$	4	9.54×10^{-3}	0.01
OJPA		$(2^7+2^5+2^3+2^1+2^0, 2^7+2^4+2^0, 2^7-2^5+2^0, 2^5+2^1)/174$	6	2.36×10^{-3}	
Opt	8	$(2^7+2^5, 2^7+2^3, 2^7+2^5-2^3, 2^5)/163$	3	8.04×10^{-3}	0.661
Fast		$(2^7+2^6+2^4, 2^7+2^5+2^4, 2^7-2^5+2^4, 2^5)/208$	5	9.54×10^{-3}	0.01
OJPA		$(2^9+2^5+2^3+2^2+2^1, 2^9-2^5-2^3+2^0,$ $2^8+2^6-2^2, 2^6+2^5+2^3+2^2+2^1+2^0)/569$	8	2.06×10^{-4}	
Opt	10	$(2^9+2^7, 2^9+2^5, 2^9+2^7-2^5, 2^7)/653$	3	7.69×10^{-3}	600
Fast		$(2^7+2^6+2^4, 2^7+2^5+2^4, 2^7-2^5+2^4, 2^5)/208$	5	9.54×10^{-3}	0.01

It is also illustrative that Fast can only increase $L(\underline{p})$ as n increases because of its algorithm structure, but its resulting error terms tend to be lower as a result – note the first estimate of Fast.

The complexity of Opt does increase rapidly with an increased number of bits of precision, although the increase is not exponential. Specifically, the example given in Table 2, after n exceeds 9 or 10, slows its rate of increase because so much of the search trees may be pruned prior to decision branching.

6 Conclusions

Two algorithms have been described for finding simultaneous rational representations with small $L(\underline{p})$. The representations resulting from these algorithms are appropriate for multiplierless implementations of linear transforms, such as the DCT. One has been shown to be implementable in real-time, while the other runs in tractable computational complexity for offline computation of optimal solutions. Because the fast algorithm tends to generate higher $L(\underline{p})$ and lower error as a result, it is better suited for applications where required n is smaller (such as the application in [4]) or where more terms can be tolerated for lower error and controlled precision. In contrast to known algorithms for finding simultaneous rational representations, these algorithms provide for specific control of the precision of the solution, as well as the polynomial representations of the numerators.

Acknowledgements: The author would like to thank Dr. Andrew J. Siefker of Murray State University and Dr. Timothy Trenary of IBM for helpful discussions related to this manuscript.

7 Appendix: Supporting Proofs

Lemma 7.1 *If $p = \sum_{\mu=\mu_1}^{\mu_2} 2^\mu$ and $\rho \in \langle p \rangle$ with $\rho \neq p$, then*

$$\rho = 2^{\mu_4} - \sum_{\mu=\mu_2+1}^{\mu_4-1} 2^\mu - 2^{\mu_3} + \sum_{\mu=\mu_1}^{\mu_3-1} 2^\mu, \tag{1}$$

for $\mu_4 \in \mathbf{N} + \mu_2$ and $\mu_3 \in \mu_1..\mu_2$, where $\sum_{\mu=\nu}^{\nu-1} 2^\mu$ is defined to have the value 0.

Proof: It may be verified quickly that $\rho = \sum_{\mu=\mu_1}^{\mu_2} 2^\mu$. Without loss of generality, it suffices to consider $\mu_1 = 0$, since multiplying each form by $2^{-\mu_1}$ must result in an integer.

First consider $p = 1$ so that $\mu_2 = \mu_1 = 0$ and (1) simplifies to $2^{\mu_4} - \sum_{\mu=0}^{\mu_4-1} 2^\mu$. Let $\rho \in \langle p \rangle \setminus \{p\}$ with $\rho = \sum_{\nu=\nu_1}^{\nu_2} \eta_\nu 2^\nu$ be given. It is clear that $\eta_{\nu_2} = 1$, since the sum is non-negative and $\sum_{\nu=\nu_1}^{\nu_2-1} 2^\nu < 2^{\nu_2}$. Thus, define $\mu_4 = \nu_2$. Also, $\nu_1 = 0$ with $\eta_0 \neq 0$, since the sum must be odd. Consider some $c \in \mathbf{N}$, which must be subtracted from 2^{ν_2} to get p:

$$2^{\nu_2} - c = p = 1 \Rightarrow c = 2^{\nu_2} - 1 = \sum_{\mu=0}^{\nu_2-1} 2^\mu.$$

Therefore, (1) holds for $p = 1$.

Now consider arbitrary run p and $\rho \in \langle p \rangle \setminus \{p\}$ with $\rho = \sum_{\nu=\nu_1}^{\nu_2} \eta_\nu 2^\nu$. As above, $\eta_{\nu_2} = 1$, so again define $\mu_4 = \nu_2$. Also, $\nu_1 = 0$ with $\eta_0 \neq 0$, since the sum must be odd. As above, consider some c, which must be subtracted from 2^{ν_2} to get p:

$$2^{\nu_2} - c = p = \sum_{\mu=0}^{\mu_2} 2^\mu < 2^{\mu_2+1} \Rightarrow c > 2^{\nu_2} - 2^{\mu_2+1} = \sum_{\mu=\mu_2+1}^{\nu_2-1} 2^\mu.$$

Since $p = \sum_{\mu=0}^{\mu_2} 2^\mu < 2^{\mu_2+1}$, ρ must also have the second term of (1). Finally,

$$\left(2^{\mu_4} - \sum_{\mu=\mu_2+1}^{\mu_4-1} 2^\mu\right) - p = 2^{\mu_2+1} - \sum_{\mu=0}^{\mu_2} 2^\mu = 1,$$

so the remainder of the terms of ρ must sum to -1. But from the discussion above, the remainder of the terms must have exactly the form of the last two terms in (1). $\square$

Lemma 7.2 *L is a metric.*

Proof: It is clear that $L(\underline{0}) = 0$ and that $L(\underline{p}) \geq 0\ \forall \underline{p}$. Consider $\underline{p}_1 = \{\eta_{1,0}2^0, 0, \ldots, 0\}$ and $\underline{p}_2 = \{\eta_{2,0}2^0, 0, \ldots, 0\}$. Put $\underline{p} = \underline{p}_1 + \underline{p}_2 = \{\eta_1 2^1 + \eta_0 2^0, 0, \ldots, 0\}$. The following list of possibilities is exhaustive:

- $\eta_0 = \eta_1 = 0$, which only occurs if $\eta_{1,0} = -\eta_{2,0} = \pm 1$, in which case $0 = L(\underline{p}) < L(\underline{p}_1) + L(\underline{p}_2) = 1 + 1$.
- $\eta_0 \in \{\pm 1\}$, which only occurs if $\eta_{1,0} \neq 0$ and $\eta_{2,0} = 0$ or vice versa, so that $1 = L(\underline{p}) = L(\underline{p}_1) + L(\underline{p}_2) = 1 + 0$. In this case, $\eta_1 = 0$.
- $\eta_1 \in \{\pm 1\}$, which only occurs if $\eta_{1,0} = \eta_{2,0} = \pm 1$, so that $1 = L(\underline{p}) < L(\underline{p}_1) + L(\underline{p}_2) = 1 + 1$ and $\eta_0 = 0$.

Note that $L(2^\mu \underline{p}) = L(\underline{p})$. Since any $\underline{p}$ may be written as a finite sum of vectors of this form with power-of-2 multipliers, it follows from induction that the triangle inequality holds, and thus L is a metric. $\square$

Lemma 7.3 *The resulting representations for $\underline{p}_i$ in the fast algorithm of Section 3 increase $L(\underline{p}_i)$ by the smallest possible amount at each iteration to minimize the error in $|\Delta_{j,i} - r'_{j,i} q_{i+1}|$.*

Proof: It suffices to consider those $\Delta_{j,i} \neq 0$, since these are the only terms that can increase $L(\underline{p}_i)$. Since $\Delta_{j,i} = 2^{h_{j,i}}$ for some $h_{j,i} \in \mathbf{N}$, each (j, i) contributes at most one to $L(\underline{p}_i)$. Because of the way in which $\Delta_{j,i}$ are chosen (see step 3 of the definition for $\psi()$ in Section 3), $L(\underline{p}_i)$ is increased by the minimum count required to minimize the error in $\Delta_{j,i}$. $\square$

Lemma 7.4 *Suppose $\exists \mu_1, \mu_2$ such that $\mu_2 \in \mathcal{A}$, $s_{j,\mu_2} \in \{\beta, \beta'\}$, and $s_{j,\mu} = B\ \forall \mu \in \mu_1..\mu_2 - 1$. Then the tree $T(p_j)$ may be pruned by setting $s_{j,\mu_2} = A$ and $s_{j,\mu} = \alpha\ \forall \mu \in \mu_1..\mu_2 - 1$; i.e., by converting the run to form F_1.*

Proof: The tree $T(p_j)$ may be pruned if the replacement does not increase $L(\underline{p})$ above its minimal value. Since $\mu_2 \in \mathcal{A}$, setting $s_{j,\mu_2} = A$ cannot increase $L(\underline{p})$. If s_{j,μ_2} had been β, the run has no secondary run, so converting to form F_1 affects no other symbols in the string. If s_{j,μ_2} had been β', converting to form F_1 allows the secondary run to be converted to form F_0 or F_1, depending on the resulting $L(\underline{p})$. $\square$

Lemma 7.5 *Suppose $\exists \mu_1, \mu_2$ such that $s_{j,\mu_2} = \beta$ and $s_{j,\mu} = B \; \forall \mu \in \mu_1..\mu_2 - 1 \subset \mathcal{A}$. Then the tree $T(p_j)$ may be pruned by setting $s_{j,\mu_2} = \alpha$ and $s_{j,\mu} = A \; \forall \mu \in \mu_1..\mu_2 - 1$; i.e., by converting the run to form F_0.*

Proof: Since $s_{j,\mu_2} = \beta$, the run has no secondary run, so conversion to form F_0 affects no other symbols in the string. Since $s_{j,\mu} = B \; \forall \mu \in \mu_1..\mu_2 - 1 \subset \mathcal{A}$, converting to F_0 cannot increase $L(\underline{p})$. $\square$

Lemma 7.6 *Suppose $\exists \mu \in \mathcal{A}$ with $s_{j,\mu} = \Gamma$ and $s_{j,\mu+1} = \gamma$. Then the tree $T(p_j)$ may be pruned by setting $s_{j,\mu} = A$.*

Proof: Since $s_{j,\mu+1} = \gamma$, the run has no secondary run, so changing $s_{j,\mu}$ affects no other symbols in the string. Since $\mu \in \mathcal{A}$, setting $s_{j,\mu} = A$ cannot increase $L(\underline{p})$. $\square$

Lemma 7.7 *Suppose $\exists \mu_1$ such that $s_{j,\mu_1} \in \{\beta, \beta'\}$. Consider the smallest such μ_1. Then the tree $T(p_j)$ may be pruned by converting all unmasked runs at $s_{\cdot,\mu_1}$ to form F_1.*

Proof: First suppose that $\exists j_1, j_2$ such that the run at s_{j_1,μ_1} is masked and the run at s_{j_2,μ_1} is not masked. Put μ_2 such that $s_{j_1,\mu} = B \; \forall \mu \in \mu_2..\mu_1 - 1$. Then $s_{j_2,\mu} = B \; \forall \mu \in \mu_2 - 1..\mu_1 - 1$ and $\mu_2 - 1 \notin \mathcal{A}$. By K, $\mu_1 \notin \mathcal{A}$. Thus, converting all unmasked runs at $s_{\cdot,\mu_1}$ to F_1 cannot increase $L(\underline{p})$ above the $L(\underline{p})$ resulting from converting these unmasked runs to form F_0. $\square$

Lemma 7.8 *For those paths not pruned because of error bounds or $L(\underline{p})$, the algorithm described in Section 4 gives a path through a search tree containing only α and A.*

Proof: $s_{j,\mu} \in \{\gamma, \gamma', \Gamma\}$ only if $s_{j,\mu-1} \in \{\beta, \beta'\}$. Thus, if μ_1 is the smallest μ such that $s_{j,\mu} \notin \{\alpha, A\}$, $s_{j,\mu_1} \in \{\beta, \beta', B\}$. The operation K, and subsequent tree searching for increasing μ, ensures that all $s_{j,\mu} = \beta$ are changed to α or A, and that the only symbols remaining for decisions are β' with masked runs. At the decision point μ_1, these runs (including corresponding B) are replaced with α or A, and $\forall \mu > \mu_1$, $s_{j,\mu} \in \{\gamma, \gamma', \Gamma\}$ are converted per $S(p_j)$. Thus, as μ increases, $s_{j,\mu} \in \{\alpha, A, \beta, \beta', B\}$ for $\mu \leq \mu_1$ decision points, and if $s_{j,\mu_2} = B$, $\exists \mu_1 > \mu_2$ such that $s_{j,\mu} = B \; \forall \mu \in \mu_2..\mu_1$ and $s_{j,\mu_1} \in \{\beta, \beta'\}$. $\square$

Lemma 7.9 *For $z \in 0..2^m - 1$, put $\mathcal{P}_z = \left\{ \sum_{\mu=0}^{m-1} \eta_\mu 2^\mu = z : \eta_\mu \in \{0, \pm 1\} \right\}$. Put $\sigma : \mathcal{P}_z \to \{\alpha, A\}^m$ such that with $y = \sigma \left(\sum_{\mu=0}^{m-1} \eta_\mu 2^\mu \right)$,*

$$y_\mu = \left\{ \begin{array}{ll} \alpha, & \eta_\mu = 0 \\ A, & \eta_\mu \in \{\pm 1\} \end{array} \right. .$$

Then σ is invertible.

Proof: Let $x = \sum_{\mu=0}^{m-1} \eta_\mu 2^\mu \in \mathcal{P}_z$ be given, and put $\zeta = \{\zeta_\mu\} = \sigma(x)$. It suffices to show that $\sigma^{-1}(\{\zeta\})$ contains at most one element, since the inverse image contains x by definition. Suppose that $\exists x_1, x_2 \in \sigma^{-1}(\{\zeta\})$. Put $x_1 = \sum_{\mu=0}^{m-1} \eta_{1,\mu} 2^\mu$ and $x_2 = \sum_{\mu=0}^{m-1} \eta_{2,\mu} 2^\mu$. By definition of σ,

$$\{\mu : (\eta_{1,\mu} = 0, \eta_{2,\mu} \neq 0) \vee (\eta_{1,\mu} \neq 0, \eta_{2,\mu} = 0)\} = \emptyset,$$

which implies that $\forall \mu$, $\eta_{1,\mu} - \eta_{2,\mu} \in \{0, \pm 2\}$. Also, $x_1 - x_2 = \sum_{\mu=0}^{m-1} \eta'_\mu 2^\mu = 0$. Without loss of generality, suppose that $\exists \mu$ such that $\eta_{1,\mu} - \eta_{2,\mu} = 2$. Since $\eta'_\mu = 0$, $\eta_{1,\mu+1} - \eta_{2,\mu+1} \in \{\pm 1\}$, a contradiction. Thus, $x_1 = x_2$. $\square$

References

[1] T.-C. J. Pang, C.-S. O. Choy, C.-F. Chan, and W.-K. Cham, "A self-timed ICT chip for image coding," *IEEE trans. circuits and systems for video technology*, vol. 9, no. 6, pp. 856–860, 1999.

[2] T. D. Tran, "A fast multiplierless block transform for image and video compression," in *ICIP*, (Kobe, Japan), pp. 822–826, 1999.

[3] J. Q. Trelewicz, M. T. Brady, and J. L. Mitchell, "Efficient integer implementations for faster linear transforms," Tech. Rep. RC 21877 (98443), IBM, Nov. 2000.

[4] M. T. Brady, J. Q. Trelewicz, and J. L. Mitchell, "Vector processing in scalar processors for signal processing algorithms," in *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, (Salt Lake City, UT), May 2001.

[5] K. M. Briggs, "On the Furtwängler algorithm for simultaneous rational approximation." Unpublished report, Complexity Res. Group, BT Res. Labs, May 2000.

[6] D. M. Hardcastle and K. Khanin, "On almost everywhere strong convergence of multidimensional continued fraction algorithms," Tech. Rep. HPL-BRIMS-2000-12, Hewlett Packard, May 2000.

[7] W. B. Pennebaker and J. L. Mitchell, *JPEG still image data compression standard*. New York: Van Nostrand Reinhold, 1993.