On Computing with Factored Rational Expressions

W. S. Brown
Bell Laboratories
Murray Hill, New Jersey 07974

This paper discusses the extension of an efficient polynomial manipulation system to deal with rational expressions. The proposed system is intended for a wide range of practical computations, including problems involving sparse expressions in many indeterminates.

Following a general discussion of the key role of simplification in symbolic computation, we conclude that rational expressions should ordinarily be represented as formal quotients of relatively prime, possibly factored polynomials.

We then show that a factored representation can often lead to substantial savings in space and time, and can reduce or even eliminate the need for costly computations of greatest common divisors. However, we cannot afford to maintain all numerators and denominators in fully factored form or even to seek factors via extra computations of polynomial greatest common divisors. Hence we conclude that a system for symbolic computation should preserve all available factors while avoiding extra work to discover more.

Finally, we show that this strategy can readily yield an abundance of shared factors, and we present specific algorithms for the basic operations.

On Computing with Factored Rational Expressions

W. S. Brown
Bell Laboratories
Murray Hill, New Jersey 07974

1. Introduction

This paper discusses the extension of an efficient polynomial manipulation system to deal with rational expressions. The proposed system is intended for a wide range of practical computations, including problems involving sparse expressions (defined below) in many (perhaps a dozen or even several dozen) indeterminates.

Since we are especially concerned with large computations where the computer is more than a mere convenience, it follows that the issues of data representation and algorithm design are of paramount importance. This paper establishes an overall approach to the subject, while a companion paper [1] describes the implementation of that approach in ALTRAN [2, 3], the options which permit users to modify it in unusual situations, and some recent experience with practical problems.

We begin (see Section 2) with a general discussion of simplification, in which we argue that simplification is the main theme in the field of symbolic computation, and not just an interesting sidelight. Applying this insight to rational expressions (see Section 3), we conclude that a system for symbolic computation should ordinarily be expected to maintain rational expressions with their numerators and denominators relatively prime.

Since factoring information is often extremely valuable (see Section 4), the system should also represent the numerators and denominators in factored form whenever factors are known. However, the system should not routinely seek to discover factors, either by complete factoring or by computing otherwise unnecessary greatest common divisors (gcd's), since any such effort would seriously degrade its performance in many situations including the very important special case of purely polynomial problems. Nevertheless, ordinary computation tends to yield factored results (see Section 5), and the system can profit handsomely by not expanding them.

In order to produce factored rational expressions with their numerators and denominators relatively prime, the system must be able to remove the gcd from a pair of factored polynomials, as discussed in Section 6. Given this facility, Section 7 presents the basic algorithms for canonical computation with factored rational expressions. Finally, some concluding observations are presented in Section 8.

2. Simplification

In performing a random sequence of operations starting with inputs which are rational expressions with random coefficients, it is easy to see that the size of the intermediate expressions can increase very rapidly with the number of steps, and simplification of any kind is almost never possible. Fortunately, in meaningful computations, opportunities for simplification are not uncommon, and the physical or mathematical structure of the problem often imposes substantial simplicity on the final result. However, if the opportunities for simplification are not exploited, the growth of intermediate expressions will be the same as in the random case, and the final unsimplified result, if it can be obtained at all, will be of little or no value.

Although it is obvious that an oversized, unsimplified rational expression would convey little meaning to the human eye, some might argue that the numerical evaluation, by computer, of such a result would allow it to serve its intended purpose. This argument is fallacious in the extreme. Frequently the purpose of computing symbolically is to avoid numbers in favor of the extra insight that comes with a comprehensible analytical result. Otherwise, when the desired results are numerical, it is highly probable that the symbolic phase was introduced so that analytical simplification could remove severe numerical difficulties. In either case the whole exercise would be pointless without simplification.

On the basis of these observations and more than a decade of practical experience,

we claim that simplification is the main theme in the field of symbolic computation, and not just an interesting sidelight. But what is simplification? Clearly, to simplify an expression is to transform it into an equivalent expression with some desirable properties. Although two distinct simplified expressions may be equivalent, it is important to require that the only simplified expression equivalent to zero is zero itself. Without this uniqueness a system for symbolic computation could generate arbitrarily bulky versions of zero with which to obscure the truth. With this uniqueness one cannot guarantee that the simplified equivalent of a nonzero expression will be usefully compact, but one can at least test two expressions for equivalence by simplifying their difference.

Since a more thorough study of simplification in general would not be germane to the purposes of this paper, we refer the interested reader to the literature [4-6], and focus our attention now on the specialized world of rational expressions.

3. Canonical Form

By definition a rational expression is a formal quotient of two polynomials called its numerator and denominator, respectively. Of course, it is understood that the denominator must be nonzero, and therefore the expression is zero if and only if its numerator is zero. Since this is easy to recognize, there is no difficulty in adopting a unique representation for zero, in agreement with the principles set forth in Section 2.

It is customary in modern algebra to assert that the formal quotients A/B and C/D are equivalent whenever AD = BC. It follows that we may simplify a rational expression by removing common divisors from its numerator and denominator until the surviving numerator and denominator are relatively prime; that is, by reducing the fraction to lowest terms.

To assess the merits of such simplification, let us consider the transformation

$$\frac{x^5 - 1}{x - 1} \rightarrow x^4 + x^3 + x^2 + x + 1 .$$

One might argue that this transformation is invalid because the left side is undefined at x = 1, or that it is a step in the wrong direction because the right side is bulkier and less revealing than the left. Formally, we can dispose of the first objection by observing that the transformation conforms to the rules of the field Z(x) of rational expressions in x over the domain Z of integers. In that field the polynomial x-1 is a nonzero element and therefore a permissible denominator, and the indicated division is clearly valid. However, there is a pitfall here for the unwary. If the expression was originally obtained by solving the equation

$$(x-1)y = x^5 - 1$$

for y, then the last trace of the alternative solution

$$x = 1$$

would be lost. The way to avoid this pitfall is not to abandon simplification but to abandon unsound methods for solving problems. With regard to the second objection, the removal of a common factor from the numerator and denominator of a rational expression always reduces their degrees, almost always reduces their bulk, and is an essential prerequisite to the accurate numerical evaluation of the expression near any zero of the factor.

In view of these specific advantages and the vital general importance of simplification, we conclude that a system for symbolic computation should ordinarily be expected to maintain rational expressions in canonical form - that is, with numerators and denominators relatively prime. In the rare cases when canonicalization causes an increase in bulk, the increase is not likely to be harmful. In those very rare cases when there is an unacceptable increase in bulk, the system should so inform the user and let him exploit the evident special structure to improve his program or to restate his problem.

4. Factored Form

In order to produce canonical results, as defined in Section 3, the system must be able to compute the gcd of a given pair of polynomials. For unfactored polynomials one uses an appropriate version of Euclid's algorithm [7, 8], while for factored polynomials one applies that algorithm to every possible pair of factors. In general the computing time is very much less when the given polynomials are in factored form, even though a great many invocations of Euclid's algorithm may be required.

Fortunately, there are many practical situations where the use of a factored representation reduces or even eliminates the need for costly gcd computations. For example, suppose we want to compute the sum of A/B^m and C/B^n, where the operands are canonical and $m > n \geq 0$. Letting

$$N = A + B^{m-n}C,$$

it follows that the sum, N/B^m, is canonical. Otherwise, there would be some irreducible polynomial P which divides both B and N. But then P would also divide A, in contradiction to the hypothesis that A/B^m is canonical.

If the system uses a factored representation, then in this example the addition operator can easily discover the shared factor, and thereby obtain the desired result with no gcd effort. However, if an unfactored representation is used, then the relation between the two denominators in not apparent, and the system must perform one or more fruitless gcd computations in order to guarantee a canonical result.

Since ordinary computation tends to produce intermediate expressions with shared factors occurring to various powers (see Section 5), the phenomenon illustrated by this example occurs frequently, and is of central importance in the design of our algorithms (see Section 7).

We can already conclude that factoring information is extremely valuable, and should be preserved whenever it is available. However, there is much more to be said. First, factoring is a powerful aid to human comprehension, and the factors may be independently meaningful. Second, as we shall see, factoring is often helpful for subsequent symbolic computation, even apart from its tendency to reduce gcd effort. Third, factoring facilitates accurate numerical evaluation by minimizing and isolating the possibilities for destructive cancellation. Finally, whenever there is more than one indeterminate and/or the factors are sparse (a polynomial of degrees $d_1,\ldots,d_i$ in i indeterminates is called sparse if it has only a small fraction of the maximum possible number, $(d_1 + 1)\ldots(d_i + 1)$, of terms), factoring tends to decrease the bulk of the expression.

This last point is especially significant because it reinforces all the others, and because sparse expressions in many indeterminates are the rule rather than the exception. Typically, some indeterminates represent unknowns, while others represent known parameters. Not infrequently, an array of indeterminates is introduced to represent the coefficients in an expansion of some function. When more than a few indeterminates are involved, the expressions are invariably sparse, since otherwise neither people nor computers could cope with them.

To justify the assertion that factoring tends to decrease bulk, let us consider a polynomial in i indeterminates with f similar t-term factors. Clearly the factored representation involves a total of ft terms. In the sparse extreme no cancellations will occur when the factors are multiplied together, so the expanded form has t^f terms. In the dense extreme, suppose each factor has degree d in each of the indeterminates. Then $t = (d+1)^i$, and by the same reasoning the expanded form has

$$(fd+1)^i \approx f^i t$$

terms. Thus for dense polynomials in a single indeterminate the two representations are about the same size, but with increasing sparseness and/or an increasing number of indeterminates a highly factored representation becomes more and more favorable.

With all the advantages that factors offer, it is tempting to consider maintaining all numerators and denominators in fully factored form. With such a representation the removal of common factors would be trivial, and almost all expressions would be as compact as possible. Unfortunately, the addition of two unfactored polynomials, which is the easiest of all polynomial operations, would now require the invocation of the polynomial factoring algorithm [9, 10], which is certainly the hardest. Since one of the most basic design objectives of a system for computing with rational expressions is to retain the efficiency of the underlying polynomial manipulation system for purely polynomial problems, we cannot afford such a strategy.

An alternative and less costly way to pursue factoring would be to further factor each factored numerator or denominator, via repeated applications of Euclid's algorithm, until all its factors are pairwise relatively prime. Since an unfactored polynomial would be an acceptable expression, the straightforward addition of two unfactored polynomials yielding a third would not conflict with this strategy. However, the simple addition

$$AC + BC = (A+B)C,$$

which should involve C only to the extent that it must be recognized as a shared factor (note the advantage of the factored representation), would also require the computation of gcd(A + B, C) and still more gcd's if that one is nontrivial. Clearly we cannot afford this strategy either.

Thus we conclude that a system for symbolic computation should represent

rational expressions as formal quotients of relatively prime, possibly factored polynomials. Since factors are extremely valuable, they should always be preserved, but polynomial gcd's should be computed only to the extent necessary to maintain canonical form. For flexibility in unusual situations, the user should be permitted to modify this strategy (see [1]) to maintain numerators and/or denominators in unfactored form, and to perform more polynomial gcd computations when factoring is critical, or fewer when there is reason to believe that canonical form can be achieved or approximated by less costly means.

5. Factors Factors Everywhere With Opportunities to Share

In a purely polynomial problem the use of a factored form, as outlined above, may have little or no effect except to defer any requested multiplications until they are required for subsequent additions. However, for rational expressions with nontrivial denominators we shall demonstrate that every basic operations yields a factored result which shares one or more factors with the operand(s), while substitution of rational expressions into rational expressions, and operations on higher level structures such as matrices of rational expressions and truncated power series with rational-expression coefficients exhibit the same phenomenon even more dramatically.

The importance of sharing arises from the fact that the system need not maintain multiple copies of a shared factor, but instead can access a single copy through multiple pointers. It is therefore trivial to remember sharing information whenever it arises or is discovered, and as we shall see, such information is extremely useful.

Let us begin by considering the multiplication of two canonical rational expressions, denoted by A/B and C/D, where A, B, C, and D are (possibly factored) polynomials. If the polynomials AC and BD are relatively prime, then the (canonical) result of the operation is the formal quotient (AC)/(BD), which consists of all of the factors of both operands. Otherwise, the reduction of this quotient to canonical form will decrease the degrees of the numerator and the denominator, and may change the number of factors in either direction. Any new factor will, of course, be a proper divisor of some factor which originally occurred in one of the operands.

With this example in mind it is easy to see that the reduction of the result of an operation to canonical form either has no effect or yields an increase in the totality of factoring and sharing information. Since the message of this section does not depend on any benefits from canonicalization, we shall temporarily adopt the pessimistic assumption that there are none.

Turning to addition we thus obtain the formula

$$\frac{A}{B} + \frac{C}{D} = \frac{AD+BC}{BD} .$$

In general the numerator of the result is in unfactored form, but the denominator consists of all of the factors of the denominators of both operands.

In the case of differentiation we have

$$\left(\frac{A}{B}\right)' = \frac{BA'-AB'}{B^2} .$$

Here the denominator of the result has the same factors as the denominator of the operand, but the exponent of each factor is doubled.

Next let us consider substituting the rational expression A/B for the indeterminate x in the form

$$F = \prod F_i^{e_i}$$

where e_i is a positive or negative integer and

$$F_i = f_{i,d_i} x^{d_i} + f_{i,d_i-1} x^{d_i-1} + \ldots + f_{i,0}$$

Clearly the result is the expression

$$\frac{\prod\left(f_{i,d_i} A^{d_i} + f_{i,d_i-1} A^{d_i-1} B + \ldots + f_{i,0} B^{d_i}\right)^{e_i}}{B^{\Sigma d_i e_i}}$$

which in general contains all of the factors of B together with a new factor corresponding to each of the F_i.

As an example of matrix computation, let us consider the determinant of a matrix of rational expressions. By putting each row over a common denominator, it is easy to show that

$$\det\left(\frac{A_{ij}}{B_{ij}}\right) = \frac{\det\left(C_{ij}\right)}{\prod_{i,j} B_{ij}}$$

where

$$C_{ij} = A_{ij} \prod_{k \neq j} B_{ik} \quad .$$

Here the denominator of the result consists of all of the factors of all of the denominators of the given matrix elements.

Finally, as an example of computation with truncated power series, let us examine the operation of reciprocation. Letting

$$F = \frac{A_0}{B_0} + \frac{A_1}{B_1} x + \frac{A_2}{B_2} x^2 + \frac{A_3}{B_3} x^3 + \frac{A_4}{B_4} x^4 + \ldots$$

where the coefficients are independent of x, we find that

$$\frac{1}{F} = \frac{B_0}{A_0} - \frac{{B_0}^2 A_1}{{A_0}^2 B_1} x - \frac{{B_0}^2 C_2}{{A_0}^3 {B_1}^2 B_2} x^2$$

$$- \frac{{B_0}^2 C_3}{{A_0}^4 {B_1}^3 B_2 B_3} x^3 - \frac{{B_0}^2 C_4}{{A_0}^5 {B_1}^4 {B_2}^2 B_3 B_4} x^4 + \ldots$$

where C_2, C_3, C_4, ... are sums of products of A's and B's. Not only do the coefficients of the resulting truncated power series contain all of the factors of A_0 and the B's, but furthermore each of these factors, except the factors of B_0, recurs to higher and higher powers as the order increases.

6. Seeking Common Divisors of Factored Polynomials

In writing algorithms for canonical computation on factored rational expressions one frequently invokes a subprocedure to determine the greatest common divisor of two factored polynomials. On other occasions it would be helpful to know about common divisors, but it is not essential to have the greatest common divisor and may not be worth the cost.

In this section we define three common-divisor functions

icd — identity common divisor

idcd — identity-divisibility common divisor

idgcd — identity-divisibility-greatest common divisor

corresponding to three specific levels of effort (see [1] for detailed algorithms), and we introduce the generic function

cd — common divisor

to refer to any one of the three. For later convenience we adopt the convention that each function removes the common divisor from its arguments. Thus the assignment

$$G = \mathrm{cd}(A, B)$$

replaces A and B by their cofactors A/G and B/G, respectively.

Now suppose we are given the polynomials

$$A = \prod {A_i}^{\alpha_i}$$

$$B = \prod {B_j}^{\beta_j}$$

where the A_i and B_j are in unfactored form. The icd function considers each pair (A_i, B_j) for identity (we assume that equal unfactored polynomials are represented identically), and returns the greatest common divisor that is attainable in that way. After computing

$$G = \mathrm{icd}(A, B)$$

the cofactors (which, by convention, have just replaced A and B) have no explicit factors in common.

To go farther, the idcd function first invokes the icd function, and then examines each pair (A_i, B_j) for divisibility, making appropriate adjustments if either divides the other. Clearly this function requires a polynomial division algorithm which returns either the exact quotient of its arguments or an indication that none exists, and it is fortunate that such an algorithm can be remarkably efficient [11]. After computing

$$G = \mathrm{idcd}(A, B)$$

it is guaranteed that no explicit factor of either cofactor divides any explicit factor of the other.

Finally, to obtain the true greatest common divisor, the idgcd function first invokes the idcd function, and then applies Euclid's algorithm to each pair (A_i, B_j), making appropriate adjustments whenever a new factor is discovered. After computing

$$G = \text{idgcd}(A, B)$$

one can be certain that the cofactors are relatively prime.

7. Basic Algorithms for Canonical Computation

We are now prepared to present algorithms for the basic operations on factored rational expressions. In each operation we assume that the operands are canonical, and we insist that the result be made canonical with a minimum of effort. For simplicity we ignore special cases involving zeros, units, and monomials, but the reader should be aware that the proper treatment of these cases is crucial in practice [1].

Let us begin by considering the multiplication of A/B and C/D, where A, B, C, and D are (possibly factored) polynomials. Clearly

$$\frac{A}{B} \cdot \frac{C}{D} = \frac{AC}{BD} \quad ,$$

but the right side is not necessarily canonical. Since the operands are canonical, we know that

$$\gcd(A, B) = 1$$
$$\gcd(C, D) = 1$$

so it suffices to seek common divisors between A and D and between B and C. However, if A and C have any common factors, they cannot enter into either gcd(A,D) or gcd(B,C), and similarly for any common factors of B and D. Of course, we can rarely if ever afford to seek such common factors with polynomial gcd computations, but it may well be worthwhile to seek them with polynomial divisions, and it seems clearly a good bet to look for common factors that are explicit in the representation. Although all such factors could be found by computing icd(A, C), any factor which occurs to different powers in A and C would not be fully removed from both. To remove all available common factors of A and C from A, the correct divisor is $\text{cd}(A,C^n)$ for any sufficiently large integer n, and we denote this, for convenience, by $\text{cd}(A, C^\infty)$. According to our convention, this invocation has the side effect of removing the common divisor from A; however, it has no effect on C, since C is not an argument. The multiplication algorithm now follows:

1. Set $T=A$, $G=\text{cd}(A,C^\infty)\text{cd}(T^\infty,C)$.
2. Set $T=B$, $H=\text{cd}(B,D^\infty)\text{cd}(T^\infty,D)$.
3. Invoke gcd(A,D), gcd(B,C).
4. Return $\frac{GAC}{HBD}$.

To divide A/B by C/D, it suffices to multiply A/B by D/C.

To raise A/B to the n-th power, one simply returns A^n/B^n, which is clearly canonical.

To compute the sum (or difference)

$$S = \frac{A}{B} \pm \frac{C}{D}$$

we first note that

$$S = \frac{AD \pm BC}{BD} \quad .$$

If a common divisor

$$G = \text{cd}(A,C)$$

is available, it can be used profitably to factor the numerator, but it is clearly relatively prime to the denominator. On the other hand, the result is canonical as it stands if and only if B and D are relatively prime, and therefore we are obliged to compute

$$H = \gcd(B,D).$$

Taking account of the side effects of these functions, we now have

$$S = \frac{GN}{HBD} \quad ,$$

where

$$N = AD \pm BC$$

and B and D relatively prime. We have

already observed that G is relatively prime to the entire denominator, and it is easy to see that N is relatively prime to both B and D. Furthermore, if H has any known factors in common with either B or D, they may be removed completely from H before computing the gcd of H and N. Thus we compute

$$K = cd(H, (BD)^{\infty})$$

to obtain

$$S = \frac{GN}{HKBD} ,$$

and then we invoke

$$gcd(H, N)$$

to canonicalize this result. In summary, the algorithm for the addition (subtraction) of A/B and C/D is:

1. Set G = cd(A, C).
2. Set H = gcd(B,D).
3. Set N = AD ± BC.
4. Set K = cd(H, $(BD)^{\infty}$).
5. Invoke gcd(H, N).
6. Return $\frac{GN}{HKBD}$.

Finally, to compute the derivative

$$D = \left(\frac{A}{B}\right)'$$

we first note that

$$D = \frac{BA'-AB'}{B^2} .$$

where A′ and B′ are the derivatives of A and B, respectively. If a common divisor

$$G = cd(A,A')$$

is available, it can be used profitably to factor the numerator, and it is clearly relatively prime to the denominator. On the other hand, the result is canonical as it stands if and only if B and B′ are relatively prime, and therefore we are obliged to compute

$$H = gcd(B, B') .$$

Taking account of the side effects of these functions, we now have

$$D = \frac{GN}{HB^2} ,$$

where

$$N = BA'-AB'$$

and B and B′ are relatively prime. We have already observed that G is relatively prime to the denominator, and it is easy to see that N is relatively prime to B. Thus it remains only to remove all common factors from H and N, and as we shall see, this is quite easy. Suppose the originally given B has the decomposition

$$B = b \prod B_j^{\beta_j}$$

where b is the content of B (i.e., the product of the factors with derivative zero), while each of the B_j is an irreducible polynomial with a nonzero derivative. (Fortunately, it suffices to know that this decomposition exists; we needn't actually find the factors.) Then it is eady to show that

$$H = b \prod B_j^{\beta_j-1} ,$$

and the computation of H has the side effect of setting

$$B = B/H = \prod B_j .$$

Since all of the factors of H, except b, are shared with this new value of B, they are relatively prime to N. Thus we simply compute

$$b = content(H)$$
$$H = H/b$$

(a decomposition which can be found as a byproduct of computing H) to obtain

$$D = \frac{GN}{bHB^2} ,$$

and then invoke

$$\gcd(b, N)$$

to achieve canonical form. (To see that this gcd can be nontrivial, the reader is invited to differentiate the expression

$$\frac{x}{b(x+b)}$$

with respect to x.) In summary, the algorithm for the differentiation of A/B is:

1. Differentiate A and B to obtain A′ and B′, respectively.
2. Set G = cd(A, A′).
3. Set H = gcd(B, B′).
4. Set b = content(H), H = H/b.
5. Set N = BA′-AB′.
6. Invoke gcd(b, N).
7. Return $\frac{GN}{bHB^2}$.

8. Conclusion

Our stated purpose was to discuss the extension of an efficient polynomial manipulation system to deal with rational expressions. After assessing the role of simplification, we concluded that rational expressions should be represented as formal quotients of relatively prime, possibly factored polynomials, and that all available factors should be preserved while polynomial gcd computations are minimized. Finally, we showed that this strategy can readily yield an abundance of shared factors, and we presented specific algorithms to implement the basic operations.

Having come this far, we are well prepared to study the substitution of rational expressions into rational expressions, and operations on higher level structures such as matrices of rational expressions and truncated power series with rational-expression coefficients. Although the study has barely begun, these topics have already proven to be rich and fascinating. Success in this effort promises a new era of usefulness for systems for systems for symbolic computation.

Acknowledgments

The author is deeply indebted to A. D. Hall, S. C. Johnson, and S. I. Feldman for their inspired and intensive collaboration over the past several years in the work from which this paper evolved.

REFERENCES

1. A. D. Hall, Factored Rational Expressions in ALTRAN, this issue.
2. A. D. Hall, The ALTRAN System for Rational Function Manipulation - A Survey, Comm. ACM 14, 517-521 (1971).
3. W. S. Brown, ALTRAN User's Manual, Third Edition, Bell Laboratories, 1973.
4. B. F. Caviness, On Canonical Forms and Simplification, Journ. ACM 17, 385-396 (1970).
5. Joel Moses, Algebraic Simplification: A Guide for the Perplexed, Comm. ACM 14, 527-537 (1971).
6. S. C. Johnson, On the Problem of Recognizing Zero, JACM 18, 559-565 (1971).
7. W. S. Brown, On Euclid's Algorithm and the Computation of Polynomial Greatest Common Divisors, JACM 18, 478-504 (1971).
8. Joel Moses and David Y. Y. Yun, The EZ GCD Algorithm, Proc. of the 1973 ACM National Conference, ACM, New York, 1973.
9. Paul S. Wang and L. Preiss Rothschild, Factoring Multivariate Polynomials over the Integers, SIGSAM Bulletin 28, 21-29, December 1973.
10. David R. Musser, Multivariate Polynomial Factorization, to appear.
11. S. C. Johnson, Sparse Polynomial Arithmetic, this issue.