J. Symbolic Computation (1995) **20**, 207–214

Linear Problems and Linear Algorithms

B. CURTIS EAVES[†] AND URIEL G. ROTHBLUM[‡]

[†] *Department of Operations Research, Stanford University, Stanford, CA 94305-4022, USA*
[‡] *Faculty of Industrial Engineering and Management, Technion—Israel Institute of Technology, Haifa 32000, Israel and RUTCOR—Rutgers Center for Operations Research, Rutgers University, P.O.B. 5062, New Brunswick, NJ 08904, USA*

(Received February 1994; revised July 1995)

Using predicate logic, the concept of a linear problem is formalized. The class of linear problems is huge, diverse, complex, and important. *Linear and randomized linear algorithms* are formalized. For each linear problem, a linear algorithm is constructed that solves the problem and a randomized linear algorithm is constructed that *completely solves* it, that is, for any data of the problem, the output set of the randomized linear algorithm is identical to the solution set of the problem. We obtain a single machine, referred to as the Universal (Randomized) Linear Machine, which (completely) solves *every* instance of *every* linear problem. Conversely, for each randomized linear algorithm, a linear problem is constructed that the algorithm completely solves. These constructions establish a one-to-one and onto correspondence from equivalence classes of linear problems to equivalence classes of randomized linear algorithms.

Our construction of (randomized) linear algorithms to (completely) solve linear problems as well as the algorithms themselves are based on Fourier Elimination and have superexponential complexity. However, there is no evidence that the inefficiency of our methods is unavoidable relative to the difficulty of the problem.

1. Introduction

Recent results on linear problems and solvability by linear and randomized linear algorithms are synthesized. A full development and proofs can be found in Eaves and Rothblum (1989, 1992, 1993, 1994).

A *(predicate) problem* is a formula in the predicate language of ordered fields whose free variables have been partitioned into *data* and *solution variables.* Given an ordered field and an assignment of the data variables to elements of the ordered field, the task is then to find an assignment of solution variables to elements of the field so that the given formula is satisfied. A linear problem is a problem whose formula is linear in the solution and quantified variables. The class of the linear problems is huge, diverse, complex and important. The class contains, for example, finding the rank of matrices, optimizing linear programming problems, optimizing bounded variable integer programs, solving linear complementary problems, and determining satisfiability in sentential logic.

Using a binary rooted tree, with arithmetic operations and comparisons at its nodes, a *linear algorithm* is formalized. A linear algorithm is defined to *solve* a problem if for every problem data taken as input, the algorithm either emits an output which is a solution for the problem instance, or does not emit an output, indicating the problem has no solution. Given any linear problem, a linear algorithm is constructed which solves the linear problem. Given any linear algorithm, a linear problem is constructed which the algorithm solves. A single machine, referred to as the *Universal Linear Machine*, is developed which solves each and every linear problem, that is, given the problem and data, the Universal Linear Machine solves the problem instance.

Randomized linear algorithms are formed by extending linear algorithms to include random selections from intervals. A randomized algorithm is defined to *completely solve* a problem if for every problem data taken as input, the algorithm output set equals the problem solution set. Given a linear problem, a randomized linear algorithm is constructed which completely solves the linear problem. Given any randomized linear algorithm, a linear problem is constructed which the algorithm solves completely. A single machine, referred to as the *Universal Randomized Linear Machine*, is developed which will completely solve each and every linear problem, that is, given the problem and data, the Universal Randomized Linear Machine will completely solve the problem instance.

Using the above construction of equivalence classes of linear problems are shown to be in one-to-one and onto correspondence with equivalence classes of randomized linear algorithms. For this purpose two problems are *equivalent* if they share the same (data, solution) pairs and two algorithms are *equivalent* if they share the same (input, output) pairs. A problem and an algorithm correspond if the set of all (data, solution) pairs of the problem coincide with the set of all (input, output) pairs of the algorithm, or equivalently, if the algorithm completely solves the problem.

Our construction of (randomized) linear algorithms to (completely) solve linear problems as well as the algorithms themselves are based on Fourier Elimination and have superexponential complexity; further, practical experience with Fourier Elimination suggests that the applicability of these algorithms (even in purely existential problems) is extremely limited. However, there is no evidence that the inefficiency of our methods is unavoidable relative to the difficulty of the problem.

2. Numbers and formulae

The computation domain is taken as an ordered field and we refer to elements of ordered fields as *numbers*. Given an ordered field it is assumed that the *five ordered field operations* can be executed, that is, the *four arithmetic operations* of *addition, subtraction, multiplication* and *division by a nonzero number* and the *comparison operation* of *testing positivity*. Further, it is assumed that numbers can be received, stored, moved, and emitted. This model of computation is nearest 'reality' for the ordered field of rational numbers.

A *predicate formula over* an ordered field, referred to as a *formula*, is a finite string of symbols obtained by applying connectives like $\wedge$ (and), $\vee$ (or), $\neg$ (not), $\rightarrow$ (imply) and $\leftrightarrow$ (equivalent), and quantifiers $\forall$ (for all) and $\exists$ (for some) to polynomial inequalities and equations. For example $(\forall x((x^2 - y > 0) \wedge (\exists z(z^2 + yx - 5 = 0))))$ is a formula. The set of formulae is formally defined recursively.

A formula has two types of variables, namely those which are captured in the scope of a quantifier and those which are not. The former are called *quantified variables* and the latter are called *free variables*. For example, in the formula given above, the variables x and z are quantified and the variable y is free. Given an ordered field, a formula and an assignment of the free variables to numbers, the formula is either a *correct* (true) mathematical statement or an *incorrect* (false) one. A formula with assigned free variables may be correct over one ordered field but incorrect over another one.

A formula without quantifiers is called *quantifier-free*. A formula is defined to be *linear* in a particular subset of its variables if no variables of the set are multiplied together in any one of the polynomials of the problem. Testing for linearity requires the conversion of each polynomial of the formula to a sum of products.

3. Linear problems

A *predicate problem*, referred to as a *problem*, is a formula with the free variables partitioned into a pair of disjoint sets called *data variables* and the *solution variables*. Given an ordered field, *data* for a problem is an assignment of the data variables to numbers. A problem with data is referred to as a *problem instance*. Given an ordered field, a *solution to* a problem instance is an assignment of solution variables to numbers which yields the formula correct. Such an assignment may be a solution of a problem instance over one ordered field, but not over another. Two problems are defined to be *equivalent* if they have identical (data, solution) pairs over every ordered field.

A problem with an empty set of solution variables is called a *decision problem*. Given data, the problem instance is correct, that is, has a solution, if and only if the empty assignment is a solution, and the problem instance is incorrect, that is, has no solution, if and only if the empty assignment is not a solution.

A problem is called *linear* if its formula is linear in the collection of the quantified and solution variables. A problem is called *essentially* linear if it is equivalent to a linear problem. An example of a linear problem is linear programming, namely the task of maximizing a linear objective subject to linear inequality constraints or deciding that no such maximization is possible. Using matrix notation, linear programming is formulated as a linear problem with the formula $(Ax \leq b) \wedge (\forall z((Az \leq b) \rightarrow (cx \geq cz)))$. Here the components of (A, b, c) are the data variables and the components of x are the solution variables. This linear problem is equivalent to the linear problem with formula $(Ax \leq b) \wedge (\exists\lambda(\lambda^T A = c^T) \wedge (\lambda \geq 0) \wedge (cx = \lambda^T b))$.

A problem is called *quantifier-free* if its formula is quantifier-free. A solution of an instance of a quantifier-free problem over one ordered field remains a solution of the problem instance over all other ordered fields containing the data numbers. Every linear problem is equivalent to a quantifier-free problem. Hence, the solution of an instance of a linear problem over one ordered field remains a solution of the problem instance over all other ordered fields containing the data numbers. Also, quantifying solution variables of the formula of an essentially linear problem yields an essentially linear problem, and the conjunction of two essentially linear problems with common data and solution variables forms a linear problem. The above facts offer an avenue for proving that a problem is not essentially linear, that is, there is no equivalent linear problem. For example, the problem with formula $y = x_1 x_2$, solution variables x_1, x_2 and data variable y is not essentially linear; to see this observe that the problem with formula $\exists x_1 \exists x_2((y = x_1 x_2) \wedge (x_1 = x_2))$ and data $y = 2$ has a solution over the reals, but not over the rationals.

4. Linear algorithms

Linear algorithms execute ordered field operations on input numbers and emit output numbers.

A *linear pre-algorithm* is built on a *finite rooted binary tree* which is a finite directed graph with at most two successors to each node, and with exactly one directed path from the distinguished node, *the root*, to each other node. The root has no predecessor and every other node has exactly one predecessor. There are four distinct types of nodes, namely, the *root*, *forks* which have two successors, *joints* which have exactly one successor, and *leaves* which have no successors. Associated with the root is a finite set of *input variables*. Associated with each fork is a formula $(z_j > 0)$ which tests positivity of a previously calculated number $\bar{z}_j$. There are four kinds of joints: *addition*, *multiplication*, *subtraction* and *division*. Associated with each joint i is, respectively, a formula $(z_i = z_j + z_k)$, $(z_i = z_j z_k)$, $(z_i = z_j - z_k)$ or $(z_i = z_j / z_k)$, which calculates a new number $\bar{z}_i$ by executing an arithmetic operation on previously calculated numbers $\bar{z}_j$ and $\bar{z}_k$. There are two kinds of leaves: *output* and *no-output*. Associated with an output leaf is a finite set of *output variables*, possibly empty, which are used to emit calculated numbers. A description of a linear pre-algorithm is complete.

Inputs and *outputs* of a linear pre-algorithm are, respectively, assignments of input and output variables to numbers. A linear pre-algorithm emits outputs from inputs in the following way. At the root, an input is received. Control flows down a path of the tree through joints, where arithmetic operations on previously calculated numbers are executed, through forks where positivity of such values is tested and corresponding branching occurs, and terminates at a division joint with a zero divisor or terminates at a leaf. If an output leaf is reached, an output is emitted. If a no-output leaf is reached no output is emitted. Assuming unit time for input, an arithmetic operation, a comparison operation, and output, the worst case running time of a linear algorithm is the length of the longest path in its tree from the root to a leaf.

A pre-algorithm is defined to be *executable*, if for every input the path terminates at a leaf, that is, a division joint with a zero divisor is never encountered. Executability is decidable. A *linear algorithm* is defined to be a linear pre-algorithm which is executable.

A linear algorithm is defined to *solve* a problem *over* an ordered field, if for any data over the ordered field, using the data as input, either an output is emitted which is a solution to the problem instance over the ordered field, or no output is emitted and the problem instance has no solution over the ordered field. A linear algorithm is defined to *solve* a problem if it solves the problem over any ordered field.

For a *randomized linear algorithm* one additional operation is introduced beyond that of a linear algorithm, namely, the ability to select a random number from an interval in the ordered field with computed bounds. To accommodate random selections, *random selection joints* are introduced into the tree of a linear pre-algorithm. Associated with each random selection joint i is a formula $(z_j \bullet z_i)$, $(z_i \bullet z_j)$, $(z_j \bullet z_i \bullet z_k)$ or $(z_i = z_i)$, with each $\bullet$ standing for either $<$ or $\leq$; the formula selects a new number $\bar{z}_i$ subject to bounds determined by previously calculated numbers $\bar{z}_j$ and $\bar{z}_k$, as needed. There is also the requirement at each division joint that the divisor cannot have random heritage, and at each multiplication joint that at most one of the two multiplicands may have random heritage.

Randomized linear pre-algorithms, like linear pre-algorithms, emit outputs from inputs by following paths that begin at the root. Specific paths depend on the data and on random selections. A path can terminate at a division joint with a zero divisor, at a random selection joint with an empty interval, or at a leaf. If an output leaf is reached, an output is emitted. If a no-output leaf is reached no output is emitted. Assuming unit time for input, an arithmetic operation, a comparison operation, a random selection operation, and output, the worst case running time of a randomized linear pre-algorithm is the length of the longest path in its tree from the root to a leaf.

A randomized linear pre-algorithm is defined to be *executable* if for every input the path terminates at a leaf, that is, a division joint with a zero divisor or a random selection joint with empty interval is never encountered. An executable randomized linear pre-algorithm is defined to be *consistent for* an input, if all paths yielded by the input lead to the same type of leaf, namely, either all lead to output leaves or all lead to no-output leaves. An executable randomized linear pre-algorithm is defined to be *consistent*, if it is consistent for all inputs. (Executability and consistency are decidable.) A *randomized linear algorithm* is defined to be a randomized linear pre-algorithm which is executable and consistent. A linear algorithm is a special case of a randomized linear algorithm.

A randomized linear algorithm is defined to *completely solve* a problem *over* an ordered field, if for any data over the ordered field, using the data as input, the set of emitted outputs over all randomizations is the solution set of the problem instance over the ordered field. A randomized linear algorithm is defined to *completely solve* a problem if it completely solves the problem over every ordered field. Two randomized linear algorithms are defined to be equivalent, if they have the same (input, output) pairs over every ordered field.

5. Linear problems to linear algorithms

Given a linear problem, a linear and a randomized linear algorithm are constructed which, respectively, solve and completely solve the problem.

Fourier Elimination, sometimes called Dines–Fourier–Motzkin Elimination, is a method based on ordered field operations for solving linear inequality systems. Given a linear inequality system, Fourier Elimination forms a new linear inequality system with one less variable, referred to as the *eliminated variable*. A solution to the original system yields a solution to the new system by deleting the eliminated variable. A solution to the new system augmented by choosing a value for the eliminated variable in a certain interval yields a solution to the original system, and further, every solution to the original system, if any, can be emitted in this manner. The end points of the interval, if any, are computed using the solution for the new system and coefficients of the original system. By repeating the elimination method until no variables remain, and then by repeatedly selecting values in the appropriate intervals any solutions can be calculated for the original system. When the last variable is eliminated, an empty interval indicates that the system has no solution.

Fourier Elimination can be extended to linear inequality systems where the coefficients are also unknown; however, here several cases occur, but for each case there is a new linear inequality system with one less variable and an interval just as before. Once the coefficients are known the case can be determined. This extended version of Fourier Elimination can be applied to a linear problem to eliminate an innermost existential quantifier and thereby generate an equivalent linear problem with one less quantified variable. Cases created are incorporated in the new problem formula. By introducing double complementation an innermost universal quantifier can be converted to an existential quantifier where again Fourier Elimination can be applied to eliminate the quantifier and generate an equivalent linear problem with one less quantifier. By repeatedly applying Fourier Elimination to a linear problem, an equivalent quantifier-free linear problem is obtained. The resulting problem can be written as a disjunction of conjunctions of inequalities where each is linear in the solution variables. Again, applying Fourier Elimination to each conjunction more cases are created and the stage is set to solve the system when the data is supplied. Receiving the data, checking the cases, and computing endpoints of intervals, and selecting values in intervals, and emitting the results can be organized on a rooted tree. Given input, the control moves down the tree, first determining the case, and then repeatedly computing endpoints and selecting solution variables in the appropriate intervals, and finally emitting the output, that is, the solution to the problem instance, if any. The cases avoid division by zero and selections from empty intervals. If values are selected in the intervals deterministically, a linear algorithm is obtained. If the values are selected in the intervals randomly, there is never a division by a number with random heritage, two numbers both with random heritage are never multiplied, and a randomized linear algorithm is obtained. Given data for the linear problem the linear algorithm will emit a solution, if any, whereas the randomized algorithm can emit every solution, if any.

Procedures have now been described which for every linear problem constructs a linear algorithm for solving it and a randomized linear algorithm for completely solving it. These procedures, referred to as *compilers*, can be automated on a Turing machine, independent of the ordered field in question. On the other hand, the procedures are not linear algorithms as looping is required; after all, problems with formulae of any length can be processed by the procedures. The assembly of these procedures for creating a linear and randomized linear algorithm with the execution of the algorithm on data are referred to as the *Universal Linear Machine* and *Universal Randomized Linear Machine*, respectively. The Universal Linear Machine receives any linear problem and data and emits a solution, if any, to the problem, whereas the Universal Randomized Linear Machine receives any linear problem and data and can emit every solution.

6. Linear problems from linear algorithms

Given a linear algorithm, a linear problem is constructed which the algorithm solves. Given a randomized linear algorithm, a linear problem is constructed which the randomized linear algorithm completely solves.

For a linear algorithm with input, control passes on a path from the root to a leaf. Moving down the path the numbers computed at joints are, inductively, rational functions of input numbers. These paths are characterized by positivity tests at forks on such rational functions, that is, after clearing denominators, by a conjunction of positivity tests on polynomials in the input numbers. If the leaf of a path is an output leaf, the output numbers are rational functions in the data numbers. In this case conjoin the conjunction of positivity tests with the output variables equated to the rational functions in the input variables. Clear denominators to make a formula for this leaf which is quantifier-free and linear in the output variables. If the leaf of the path is a no-output leaf, make a formula for the leaf by conjoining the conjunction of positivity tests with $1 = 0$. A grand formula is formed by taking the disjunction over the formulae made for each leaf. With this formula form a problem by taking the data and solution variables as the input and output variables of the algorithm, respectively. This problem is linear as its formula is quantifier-free and linear in the solution variables.

Consider a data for the constructed linear problem. The data identifies the leaf which is reached when the data is used as input for the randomized linear algorithm. If the leaf reached is an output leaf, an output is emitted and the formula of the output leaf is correct for the input and output. Hence, the output yields a solution to the problem instance. If the leaf reached is a no-output leaf, the formula is incorrect for any assignment as it contains $1 = 0$. The formulae made for other leaves are incorrect with the data because the algorithm with the data as input did not reach those leaves. Hence the problem instance has no solution. The linear algorithm solves the constructed linear problem.

For a randomized linear algorithm with input, just as for a linear algorithm, control passes on a path from the root to a leaf. Moving down the path the numbers computed at joints are, inductively, rational functions of input numbers and random selection numbers. As numbers with random heritage are never divisors and as two numbers with random heritage are never multiplied, these rational functions are linear in the random selection numbers. The paths are characterized by positivity tests at forks, and after clearing denominators, the paths are characterized by a conjunction of positivity tests on polynomials in the input numbers and random selection numbers. These polynomials are linear in the random selection numbers. If the leaf reached is an output leaf, conjoin the conjunction of positivity tests with the output variables equated to those rational functions. Clearing denominators and introducing an existential quantifier on the selection variables, a formula is made for the leaf which is linear in its quantified and output variables. If the leaf reached is a no-output leaf, make a formula for the leaf by conjoining the conjunction of positivity tests with $1 = 0$. A grand formula is formed by taking the disjunction over the conjunction constructed for each leaf. With this formula form a problem is formed by taking the data and solution variables as the input and output variables of the algorithm, respectively. This problem is linear as its formula is linear in the quantified and solution variables.

Consider a data for the constructed linear problem and use the data as input for the randomized linear algorithm. If an output leaf is reached the formula of that leaf is correct for the input and output. Hence, the output yields a solution to the problem instance. Conversely, a solution to the problem instance identifies a leaf whose formula is correct with the data. This leaf is an output leaf as the formula of a no-output leaf contains $1 = 0$ which is incorrect for any assignment. The formula of the identified leaf asserts that with the input, there are assignments to the quantified variables under which the path leads to this leaf with output equalling the solution. Hence, the solution is obtained as an output from the randomized linear algorithm with the input. Thus, the solution set of the problem with the given data coincides with the output set of the randomized linear algorithm with the data as input. The randomized linear algorithm completely solves the constructed linear problem.

7. Linear problems to and from linear algorithms

Equivalence of linear problems and equivalence of randomized linear algorithms have been defined. These equivalences yield a partition of linear problems and a partition of randomized linear algorithms into *equivalence classes*. An algorithm solves a problem if and only if every algorithm in the equivalence class of the algorithm solves every problem in the equivalence class of the problem.

Given a linear problem, a randomized linear algorithm that completely solves it has been constructed, and conversely, given a randomized linear algorithm, a linear problem that is completely solved by it has been constructed. These constructions extend to correspondences between equivalence classes of linear problems and equivalence classes of randomized linear algorithms. As such, they are one-to-one, onto and inverses of each other.

8. Solving parametric linear problems

Given a linear problem and a linear algorithm which solves it, consider the task of solving the linear problem for parametric data, namely, of finding a parametric solution for the linear problem with parametric data over a range of values of the parameter. If the input is a rational function of a single parameter, a parametric solution for the problem with parametric data can be computed for all small positive values of the parameter using the linear algorithm. To do so replace the parameter by a positive infinitesimal and view the parametric data as a single data in the ordered field obtained by augmenting the original field with the positive infinitesimal. The linear algorithm applied in the larger ordered field solves the new problem instance. Replacing the infinitesimal by the original parameter in the solution yields a parametric solution to the original problem with parametric data for all small positive values of the parameter. This procedure is referred to as 'lift, solve, and lower'.

References

Eaves, B. C., Rothblum, U. G. (1989). A theory on extending algorithms for parametric problems, *Mathematics of Operations Research* **14**, pp. 502–533.

Eaves, B. C., Rothblum, U. G. (1992). Dines–Fourier–Motzkin quantifier-Elimination and applications of corresponding transfer principles over ordered fields, *Mathematical Programming* **53**, pp. 307–321.

Eaves, B. C., Rothblum, U. G. (1993). Complete solvability of linear problems and randomized linear algorithms, unpublished manuscript, 54 pages.

Eaves, B. C., Rothblum, U. G. (1994). Formulation of linear problems and solution by a universal machine, *Mathematical Programming* **65**, pp. 263–310.