

by Daniel E. Cooke, Richard Duran, Ann Gates,
and Vladik Kreinovich¹

*Computer Science Department
University of Texas at El Paso, El Paso, TX 79968*

Abstract. Environmentally safe manufacturing includes recycling. In particular, it includes reuse of the objects (whose shapes are no more needed) by cutting and pasting them into the necessary shapes. From the geometric viewpoint, the problem of how to cut and paste can be expressed in terms of equidecomposability of polygons and polyhedra. In (Kosheleva et al 1994) it has been shown that algorithms that solve all particular cases of this problem require (in the worst case) exponential computation time, and are thus not always feasible. It is therefore necessary to find particular cases of this problem that allow feasible algorithms.

Part of the complexity of the problem may be due to the fact that we have allowed all possible cut-and-paste transformations. In this paper, we only allow transformations from a fixed finite set. Can we use these transformations to transform one given shape into another? In general, the problem is still exponential-time, but we are now able to formulate a case for which a feasible algorithm exists. Our proofs use developed for *linear logic*, an extension of the standard logic proposed by J.-Y. Girard in 1987 to describe concurrent algorithms.

¹ This work was partially sponsored by the Air Force Office of Scientific Research (AFSC), under contracts F49620-89-C-0074 and F49620-93-1-0152, by NSF grant No. CDA-9015006, NASA Research Grants No. 9-482 and No. NAG 2-670 Supplement No. 2, and a Grant No. PF90-018 from the General Services Administration (GSA), administered by the Materials Research Institute and the Institute for Manufacturing and Materials Management. The authors are thankful to M. A. Taitlin for valuable discussions.

Geometric recycling problem: what is known? Environmentally safe manufacturing means, in particular, that if we have a material in shapes that are no longer needed, then instead of throwing these objects away or completely remolding them, we should try to reuse this material by cutting and pasting these objects into desirable shapes. The problem is: “how to cut and paste”? In geometry, if one shape can be “cut and pasted” into another one, we talk about equidecomposability. In (Kosheleva et al 1994) it has been shown that even for polygons (and polyhedra), algorithms that find the proper “cut and paste” procedure for all pairs of shapes require (in the worst case) exponential computation time, and are thus not always feasible. It is therefore necessary to find particular cases of this problem that allow feasible algorithms.

We are trying to formulate a tractable class of recyclable problems. One possible reason why in general, the problem of finding a proper “cut-and-paste” procedure is intractable can be as follows: In formulating this problem, we have allowed arbitrary cuts and pastes. So, maybe we can simplify the problem by restricting possible cuts and pastes?

To answer this question, let’s go back to real manufacturing. For a layperson, all cuts are equal. In real manufacturing, to implement a cut, one must first transform its geometric description into a precise manufacturing procedure (ideally, an automated one), and then perform this procedure. Therefore, if we have a choice, it is clearly preferable to implement a cut-and-paste procedure that has already been translated into a manufacturing language.

In view of this remark, it is reasonable to consider the following problem: We have a list of shape-to-shape transformations, for which the corresponding manufacturing procedures have already been spelled out. Now, we know the current shape, and we know the desired shape. Our first problem is: *Can we transform the current shape into the desired shape using only the given transformations?*

Suppose that the answer to that question is “yes, we can.” Then the next question is “how?” In what order should we apply

follows: first, apply whatever transformation is applicable to the initial shape; then, apply whatever procedure is applicable to the resulting shape, etc. Our second problem is as follows: *When will this simple procedure work, and when, if we start with a wrong procedure, we will end up in a dead end?*

In this paper, we will analyze these two problems.

2. MATHEMATICAL FORMULATION OF THE PROBLEMS

Assume that an integer k is fixed. This k will be called the *dimension*. Assume also that a finite set D of different k -dimensional polytopes is given. This set will be called a *domain*. By a *shape*, we mean a polytope from D . By a *collection of shapes* (or *collection*, for short), we mean an expression of the type $n_1 \cdot A_1 + \dots + n_k \cdot A_k$, where n_i are positive integers, and A_1, \dots, A_k are different shapes (i.e., different polytopes from D).

A *manufacturing procedure* (or, a *procedure*, for short) is an expression of the type $I \rightarrow O$, where I and O are collections of shapes, and O is equidecomposable with I . By a *procedure base* B , we mean a finite set of procedures P_1, \dots, P_r (i.e., expressions $I_1 \rightarrow O_1, \dots, I_r \rightarrow O_r$).

Assume that a collection C is given. We say that a procedure $I \rightarrow O$ is *applicable* to a collection C , if I is a subcollection of C (i.e., if I can be obtained from C by deleting some of C 's shapes). By a *result* of applying a procedure $I \rightarrow O$ to the collection C , we mean a collection $(C - I) \cup O$ (i.e., we delete all elements of I from C , and replace them with O ; this can be easily spelled out in terms of n_i).

Assume that a collection C , and a procedure base B are given. By a *manufacturing process* (or simply a *process*, for short), we mean a finite sequence or pairs $(P^{(i)}, C^{(i)})$, $i = 0, 1, \dots, N$, where, first, $C^{(0)} = C$ and, second, for every i , $P^{(i)} \in B$, and the result of applying $P^{(i)}$ to $C^{(i)}$ is $C^{(i+1)}$. The final collection $C^{(N)}$ is called a *result* (or, a *possible result*) of applying B to C .

Comment. In principle (and it is very natural in mass manufacturing), one and the same procedure can be applied several times.

tion of shapes C_{in} , and a collection C_{out} , that is a possible result of applying B to S_{in} . We say that *no scheduling* is required to transform C_{in} into C_{out} if every process that starts with C_{in} either contains C_{out} as one of its intermediate collections, or can be extended to lead to C_{out} .

Now, we can formulate two problems:

- *Problem 1: Can we do it?*

Given: A procedure base B , and collections C_{in} and C_{out} .

Check: Is C_{out} a possible result of applying B to C_{in} ?

- *Problem 2: Is scheduling required?*

Given: A procedure base B , and collections C_{in} and C_{out} .

Check: Is scheduling required?

3. OUR MAIN RESULTS

Our main result is that these two problems are NP-hard. Before we formulate these theorems and start proving them, let's recall what NP-hard means.

What is NP-hard: a brief informal explanation. In theory of computations, it is well known that not all algorithms are feasible (see, e.g., (Garey et al 1979), (Lewis et al 1981), (Martin 1991)): it depends on how many computational steps they need. If for input of length n , the algorithm requires computational time 2^n , then for an input of a reasonable length $n \approx 300$, we would need more computational steps than there can be during the lifetime of our Universe. So, such algorithms (called *exponential-time*) are usually considered not feasible. If, however, the running time grows only as a polynomial of n (i.e., an algorithm is *polynomial-time*), then the algorithm is considered feasible.

The fact that an algorithm is not feasible, does not mean that it can never be applied: it simply means that there are cases when its running time will be too large for this algorithm to be practical.

We want to prove that our problems are NP-hard. This notion (see, e.g., Garey et al, 1979) means that if there exists an algorithm that solves our problems in polynomial time (i.e., whose running

the polynomial-time algorithm would exist for practically all discrete problems such as propositional satisfiability problem, discrete optimization problems, etc. It is, however, a common belief that for at least some of these discrete problems, no polynomial-time algorithm is possible (this belief is formally described as $P \neq NP$). So, the fact that the problem is NP-hard means that no matter what algorithm we use, there will always be some cases for which the running time grows faster than any polynomial, and therefore, for these cases the problem is intractable.

Main results.

THEOREM 1. *Problem 1 (checking if we can transform the two given collections of shapes into each other) is NP-hard.*

THEOREM 2. *Problem 2 (checking if scheduling is required) is NP-hard.*

Comment. We prove these result using as analogies proofs from a special theory called *linear logic*. This theory is a generalization of the traditional logic; it has been originally proposed in (Girard, 1987) (for recent results, see, e.g., (Girard et al 1992), (Kanovich, 1991), (Kanovich, 1992), (Andreaoli et al, 1993), (Archangelsky, Dekhtyar, et al, 1993), (Archangelsky, Taitlin, 1993)). Namely, we use the results about a special fragment of linear logic called *multiplicative*, or *Horn* fragment (these results are given in (Archangelsky, Dekhtyar, et al, 1993), (Archangelsky, Taitlin, 1993)).

Remark. In this short paper, we cannot describe linear logic; to justify this omission, we simply mention that the original description in (Girard, 1987) took more than 100 pages.

From the practical viewpoint, Theorems 1 and 2 are negative results (something cannot be done). However, the same technique enable us to produce positive results as well. Namely, if we fix the total number r of procedures in a procedure base B , then *there exists an algorithm that checks whether two collections can be transformed into each other in $\leq n^{\log_2 n}$ computational steps*, where $n = |B| + |C_{in}| + |C_{out}|$ is the total length of the input ($|B|$ denotes the length of the procedure base B , i.e., the number of

the total number of polytopes in the input and output collections). This algorithm is, in effect, described in (Archangelsky, Dekhtyar, et al, 1993) and (Archangelsky, Taitlin, 1993).

4. PROOFS

Comment. These proofs will be similar to the ones presented in (Archangelsky, Dekhtyar, et al, 1993), (Archangelsky, Taitlin, 1993), where a problem of linear logic is also expressed in terms of the expressions of the type $I \rightarrow O$. There are two differences:

- We allow every procedure to be applied as many times as necessary (could be 0, could be 1, could be many), while in linear logic, each procedure has to be applied, and a procedure can be applied only once.
- In linear logic, there is no geometric interpretation, and so arbitrary procedures are possible; in particular, the ones like $A + A \rightarrow A$ for which there is no equidecomposability.

Proof of Theorem 1. To prove that the corresponding problem is NP-hard, we will prove that if it were possible to solve the problem in polynomial time, then it would be possible to solve in polynomial time a problem that is already known to be NP-hard: the so-called 3-partition problem (problem SP15 from (Garey et al, 1979); a similar reduction is used in (Archangelski, Dekhtiar, et al, 1993), (Archangelski, Taitlin, 1993)). This problem is as follows (we use slightly different notations than (Garey et al, 1979)): suppose that two positive integers m and a are fixed, and $3m$ positive integers a_1, \dots, a_{3m} are given such that $a_1 + \dots + a_{3m} = ma$. Can we divide these $3m$ integers into m triples in such a way that the sum of each triple is equal to a ?

Remark. One can easily see that it is sufficient to consider only the case when $a_i < a$ for all i : Indeed, if $a_i \geq a$ for some i , then for every triple (a_i, a_j, a_k) that contains this a_i , we have $a_i + a_j + a_k > a_i \geq a$ and hence $a_i + a_j + a_k \neq a$. So, in this case, the desired partition is impossible.

To prove Theorem 1, we will use the following reduction. Let us start with a particular case of the 3-partition problem with $a_i < a$ for all i . Let us consider the 2-D case ($k = 2$), and take the

that we will now proceed to design (higher dimensional cases can be obtained, if we consider direct products of these polygons with $[0, 1]^{k-2}$). First, we will describe the areas of these polygons, and then the polygons themselves.

- *Areas:* The area of Y is 1, the area of Z is $2a + 3$, the area of R_i is $(a - a_i)(2a + 2) - 2a$, and the area of S is $2a(2a + 3) - 8a = 2a(2a - 1)$.
- *Polygons:* We will design the polygons in the order in which they have been described in D . As Y , we take a unit square. Now, assume that we have already describe all polygons before the current one; let's denote this current one by X . We already know the area $A(X)$ of this polygon X . If this is the first polygon with this value of the area, make it a rectangle with sides 1 and $A(X)$. If we have already constructed p polygons with this same area, then make it a rectangle with sides $1/(p + 1)$ and $A(X) \cdot (p + 1)$.

Let's form a base consisting of the following $3m + 1$ procedures:

Procedure 1: $3a \cdot Y + R_1 \rightarrow (a - a_1) \cdot Z + a_1 \cdot Y$;

...

Procedure i : $3a \cdot Y + R_i \rightarrow (a - a_i) \cdot Z + a_i \cdot Y$;

...

Procedure $3m$: $3a \cdot Y + R_{3m} \rightarrow (a - a_{3m}) \cdot Z + a_{3m} \cdot Y$;

Procedure $3m + 1$: $a \cdot Y + 2a \cdot Z \rightarrow 9a \cdot Y + S$.

It is easy to check that the for each of these procedures, the total area of the input is the same as for the output and therefore, these collections are equidecomposable. As C_{in} , we take $C_{in} = 9a \cdot Y + R_1 + \dots + R_{3m}$, and as C_{out} , we take $C_{out} = 9a \cdot Y + m \cdot S$.

Let us show that the answer to the first problem ("is there a transformation from C_{in} to C_{out} ?") is "yes" \leftrightarrow the 3-partition problem is solvable. In other words, we will show that C_{out} is a possible result of applying B to C_{in} if and only if the required 3-partition exists.

- ← Assume that a 3-partition exists. Let us fix a 3-partition, and take the first triple (a_i, a_j, a_k) from this fixed partition. According to the definition of a 3-partition problem, for this

responding Procedures i , j , and k to C_{in} . As a result, we get rid of R_i , R_j , and R_k , and instead of $9a \cdot Y$, we have $(a_i + a_j + a_k) \cdot Y + ((a - a_i) + (a - a_j) + (a - a_k)) \cdot Z = a \cdot Y + 2a \cdot Z$. Now, we can apply Procedure $3m + 1$, and get $9a \cdot Y + S$. After these four procedures, we are almost back to the initial collection: the only difference is that the shapes R_i , R_j , and R_k are gone, and instead, we have S .

Then, we repeat this sequence of operations for another triple, etc. At the end, all polygons R_i will be gone, and we will end up with the desired output collection $C_{out} = 9a \cdot Y + m \cdot S$. So, if a 3-partition problem has a solution, then C_{out} is a possible result of applying B to C_{in} .

→ Assume now that C_{out} is a possible result of applying B to C_{in} . This means that there exists a process that leads from C_{in} to C_{out} . Let us show how to transform this process into the desired 3-partition.

Indeed, the initial collection C_{in} does not contain Z , so the first procedure that we apply to C_{in} cannot be Procedure $3m + 1$. It must therefore, be one of the Procedures i , $i \leq 3m$. After we apply this procedure, we have $a - a_i$ Z 's (hence, less than $2a$ of them).

Since we have some Z 's in the resulting collection, this means that we have not yet reached C_{out} . On the other hand, the number of Z 's in the resulting collection ($a - a_i$) is not sufficient for applying Procedure $3m + 1$, so we have to apply Procedure j with $j \leq 3m$. Since R_i is gone after we have applied Procedure i , we cannot apply the same Procedure i again. Indeed, to apply Procedure i , we need to have R_i (and, also, $3a$ Y 's) in the collection, and we do not have R_i any more. Therefore, $j \neq i$.

After applying the second procedure, we have $(a - a_i) + (a - a_j)$ Z 's. This number is still $< 2a$ (because all the numbers a_i are positive). Therefore, we still cannot apply Procedure $3m + 1$. And we have not yet reached C_{out} . Therefore, the next procedure to apply in our process is a procedure k with $k \leq 3m$.

have $a_i + a_j + a_k \leq 3a$ and $(a - a_i) + (a - a_j) + (a - a_k) \geq 3a - s$. If we denote the sum $a_i + a_j + a_k$ by s , then this conclusion can be described in simpler terms: we have s Y 's and $3a - s$ Z 's in the resulting collection.

Since $a_i < a$ for all i , we have $s = a_i + a_j + a_k < 3a < 9a$. Therefore, we cannot apply any of the Procedures l , $l \leq 3m$. On the other hand, since we have Z 's in the collection, we have not yet reached C_{out} , so the process (that eventually will lead to C_{out} continues. Since we cannot apply any of the procedures l with $l \leq 3m$, the only procedure that can be applied is Procedure $3m + 1$. To be able to apply this procedure, we need to have at least a Y 's and at least $2a$ Z 's. So, the following two inequalities must be true:

- $s \geq a$ (for Y 's);
- $3a - s \geq 2a$ (for Z 's).

The second inequality leads to $s \leq a$, so, we can conclude that $s = a$. We have found our first triple.

The resulting collection thus contains exactly a Y 's and $2a$ Z 's. After applying Procedure $3m + 1$, we are thus back to no Z 's and $9a$ Y 's. The only difference is that the shapes R_i , R_j , and R_k , that correspond to the first triple, are gone, and there is a new shape S instead. There are still $3m - 3$ shapes R_l in the resulting collection. If we trace what procedures are performed to get rid of them, we will discover the second, the third triple, etc. As a result, we will have a solution to the original 3-partition problem.

This equivalence proves our Theorem. Indeed, suppose that there exists an algorithm that solves our Problem 1 ("can we do it?") in polynomial time (i.e., in time that does not exceed some polynomial of n , where $n = |B| + |C_{in}| + |C_{out}|$ is the total length of the input). Let us show that this algorithm will enable us to solve the 3-partition problem in polynomial time. Indeed, given the particular case of the 3-partition problem, we form a procedure base B , and two collections C_{in} and C_{out} (as above; it takes a polynomial time), and apply the hypothetical algorithm to check whether C_{out} is a possible result of applying B to C_{in} . If it is, this means that the original 3-partition problem has a solution; if it is

The running time of this algorithm (including the formation of B , C_{in} , and C_{out} , and checking whether C_{out} is a possible result of applying B to C_{in}) is polynomial in $3m + 2$ and thus polynomial in m .

So, if there exists an algorithm that solves our Problem 1 in polynomial time, then we can have a polynomial-time algorithm for the known NP-hard problem (3-partition). This means that the our problem 1 is itself NP-hard. Q.E.D.

Proof of Theorem 2. To prove this theorem, we consider a slightly different reduction: To the domain, we add one more shape P of area 1 (the corresponding polygon is designed using the same construction as in the proof of Theorem 1), and to the procedure base, we add the following procedures involving P :

Procedure $3m + 2$: $Z \rightarrow (2a + 3) \cdot P$;

Procedure $3m + 3$: $R_1 \rightarrow ((a - a_1)(2a + 2) - 2a) \cdot P$;

...

Procedure $3m + 2 + i$: $R_i \rightarrow ((a - a_i)(2a + 2) - 2a) \cdot P$;

...

Procedure $6m + 2$: $R_{3m} \rightarrow ((a - a_{3m})(2a + 2) - 2a) \cdot P$;

Procedure $6m + 3$: $Y + P \rightarrow 2 \cdot P$;

Procedure $6m + 4$: $S + P \rightarrow (2a(2a - 1) + 1) \cdot P$.

We take the same $C_{in} = 9a \cdot Y + R_1 + \dots + R_{3m}$ as in the proof of Theorem 1, and the new $C_{out} = (9a + 2a(2a - 1)m) \cdot P$.

Definitely, C_{out} is a possible result of applying this B to C_{in} : indeed, we transform all R_i into P 's using Procedures $3m + 3$ through $6m + 2$, and then transform all Y 's into P 's using Procedure $6m + 3$.

If we apply these procedures in arbitrary order, then, as soon as we have at least one P , Z , or R_i in the resulting collection, we can still transform everything into P 's. The only case when we may get stuck (and never be able to make it into all P 's) is when the only shapes in the resulting collection are Y 's and S 's.

Similarly to the proof of Theorem 1, we can prove that such a shape is possible iff the 3-partition problem has a solution. So, we arrive at the following conclusion:

- If the 3-partition problem does not have a solution, then scheduling is not required.

In other words, a 3-partition problem has a solution iff scheduling is not required, when we use B to transform C_{in} into C_{out} . Since checking whether a 3-partition problem is solvable or not is NP-hard, checking whether scheduling is required or not is also NP-hard. Q.E.D.

5. WHAT NEW RESULTS DID WE LEARN, AND WHERE DO WE GO FROM HERE?

This paper continues the research that has been started in (Kosheleva et al 1994). First, we have again proved that in general, the problem of transforming one shape into another by cutting and pasting is difficult to solve. But:

- In addition to the negative results, we also formulate the case for which this problem is feasible. This leads to the following open problem: *to find other reasonable situations in which there are reasonable algorithms for checking equidecomposability of polygons.*
- It turned out that the algorithmic problems of equidecomposability (= of environmentally safe engineering) are closely related with one of the most complicated techniques of modern theoretical computer science (namely, with linear logic; note the word “related”: the proofs themselves are not that complicated). We hope that this connection will help the above-formulate geombinatoric problems in getting more respect (and hence, more chances to be solved) by the computer science community.

REFERENCES

Andreaoli, J. M.; Castagnetti, T.; Pareschi, R. *Abstract interpretation of concurrent languages based on linear logic*, in: *Proceedings GULP '93*, Mediterranean Press, Giugno, Italy, 1993, pp. 15–18.

Archangelsky, D. A.; Dekhtyar, M. I.; Musikaev, I. Kh.; Taitslin, M. A. *Concurrency problem for Horn fragment of Girard's linear logic*, Tver State University, Tver, Russia, 1993.

- sources, Parts 1 and 2, Peter State University, IVER, Russia, 1993.
- Garey, M. R.; Johnson, D. S. *Computers and intractability: a guide to the theory of NP-completeness*, W. F. Freeman, San Francisco, 1979.
- Girard, J.-Y. *Linear logic*, Theoretical Computer Science, 1987, Vol. 50, pp. 1–102.
- Girard, J.-Y.; Scedrov, A.; Scott, P. G. *Bounded linear logic: a modular approach to polynomial-time computability*, Theoretical Computer Science, 1992, Vol. 97, pp. 1–66.
- Kanovich, M. I. *The multiplicative fragment of linear logic is NP-complete*, Technical Report No. X-91-13, Institute for Language, Logic, and Information, University of Amsterdam, 1991.
- Kanovich, M. I. *Horn programming in linear logic is NP-complete*, In: *Proceedings of the 7-th Annual IEEE Symposium on Logic in Computer Science*, IEEE Computer Society Press, Los Alamitos, CA, June 1992.
- Kosheleva, O.; Kreinovich, V. *Geoinformatics, computational complexity, and saving environment: let's start*, Geoinformatics, 1994, Vol. 3, No. 3 (January), pp. 90–99.
- Lewis, L. R.; Papadimitrou, C. H. *Elements of the theory of computation*, Prentice-Hall, Englewood Cliffs, NJ, 1981.
- Lincoln, P. *Linear logic*, SIGACT News, Spring 1992, p. 30–37.
- Martin, J. C. *Introduction to languages and the theory of computation*, McGraw-Hill, N.Y., 1991.