11-4-2005

Master's thesis proposal: computation reuse in stacking and unstacking

Anne Loomis
Dartmouth College

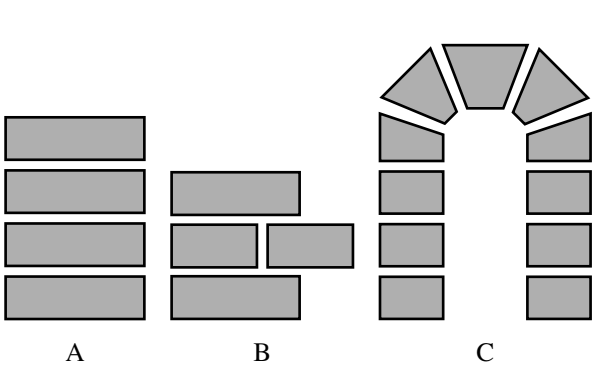
Submitted thesis proposal: November 4, 2005.

Master's thesis proposal: computation reuse in stacking and unstacking

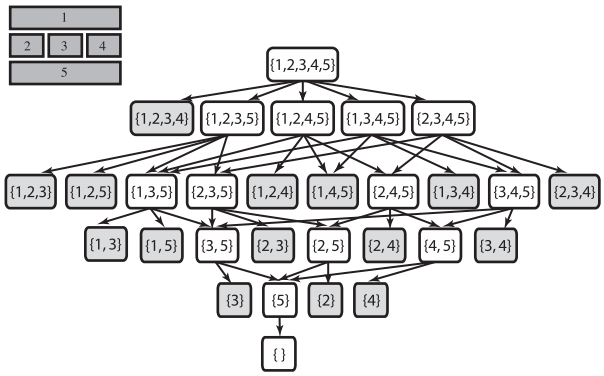
Anne Loomis
Advisor: Devin Balkcom

TR2005-563

Contents



(a) Three examples of stable piles.



(b) A simple pile and its disassembly graph.

1 Introduction

Efficient algorithms for dynamic simulation and control are fundamental to many areas of computer science, including computer games and movies, medical simulation, and mechanical design. In this proposal, we are concerned with using system dynamics to find a *stable unstacking sequence*, which we define as an order in which we can remove every object from a structure without causing the structure to collapse under gravity at any step. Consider the following simple examples. The column (A) in figure 1(a) can be unstacked by repeatedly removing the topmost block, but there is no way to take apart the Roman arch (C) by removing a single block at a time. Taking the topmost block from column (B) would cause the assembly to collapse, but the structure can be disassembled by removing the cantilevered block first. We show that one way to compute an unstacking sequence efficiently is through *computation reuse*, where we use the solution to one problem to solve a subsequent similar problem quickly.

Stable stacking and unstacking are applicable to problems of assembly planning and part nesting in manufacturing, and safely disassembling collapsed structures in search-and-rescue. We believe our findings have a broader impact as well. Although the proposed work focuses on reusing computation in a particular algorithm from physically-based modeling and considers only the discrete case, in which objects may be removed from an assembly instantaneously, we hope that the principles may be applied to other algorithms in dynamic simulation.

In the proposed work, we will extend algorithms for determining grasp and stability of a single structure to the design of computationally efficient planning and analysis tools for the stacking and unstacking of rigid-body systems. Given the number of objects to consider, this problem is more complicated than traditional grasping problems. Yet in some ways it is simpler, since the incremental nature of stacking leads us to consider very similar structures at each step.

A basic algorithm for unstacking planning is: for a structure $\mathcal{P}$, generate a *disassembly graph* like the one shown in figure 1(b), where each node holds a subset of the objects in $\mathcal{P}$, and directed edges connect nodes that differ by a single object. Run a search from the root node, which contains all objects in $\mathcal{P}$, and test the stability of the subset contained at each node that is reached. If some node contains an unstable subset, do not continue searching from that node. If a path is found to the goal, *i.e.* the node containing the empty set, stop searching and return this path. A naive implementation of this sort is computationally expensive: since there are 2^n nodes in this graph, if we test the stability of each, it can cost up to $2^n\times$ (cost of the stability test).

We can improve the efficiency of the basic algorithm by finding a way to reuse computation from one stability test to the next, and by reducing the number of nodes we search. The primary focus of the present work is in the former approach. In the remainder of this section, we discuss work related to stability calculations and computation reuse, and we give an overview of the problem space and some of the progress we have already made within it. We have developed an algorithm that uses a complementarity formulation to incrementally calculate the stability of a structure by first computing the magnitudes of contact forces necessary for the stability of a one-block subassembly, then for a two-block subassembly, and so on. In section 2 of this proposal, we argue that computation can be reused in this

stability test for any structure, and show that this algorithm outperforms the same approach without reuse, as well as a simpler test for stability. This success suggests that similar techniques may be used to optimize similar types of algorithms.

In section 3 we describe in detail what we intend to accomplish. Specifically, we propose to:

- Develop an efficient algorithm for unstacking three-dimensional structures with a Coulomb model of friction that reuses computation as a means for optimizing the test for stability, and includes methods for reducing the number of nodes we must visit in our graph search.
- Analyze the complexity of the test for stability, and give an upper bound on the number of stability tests that must be conducted for a given assembly.
- Classify structures according to their unstacking characteristics, and develop efficient tests for inclusion in different classes. An open question is whether a stable unstacking sequence always exists.
- Develop software tools for efficient unstacking planning, and demonstrate their effectiveness by applying them to structures in the physical world.

Finally, in section 4, we describe the timeline in which we plan to accomplish this work.

1.1 Related work

Work related to the unstacking problem falls into five categories: grasping, calculating stability with friction, dynamic simulation, approaches to computation reuse in various algorithms, and applications of unstacking.

1.1.1 Grasping

Whereas the goal of the unstacking problem is to determine whether all the objects in a given assembly are immobile based on the way they contact each other, the goal of traditional robot grasping problems is to immobilize a single object using some number of external contacts. Mason [20] discusses geometric methods for finding stable configurations, and presents a more complete survey than is possible here. Early work by Mishra *et al.* gave sufficient conditions and algorithms for placing frictionless fingers to ensure a stable grasp [21, 22] with respect to a force or set of forces; Nguyen's [23] and Erdmann's [13] work on force closure and friction grasps is also seminal. Recent work by Cheong *et al.* considers the problem of grasping a collection of polygons connected by hinges [8], which is a bilateral constraint problem in contrast to our unilateral constraint problem.

1.1.2 Stability

Grasping problems for rigid bodies with a Coulomb friction model are complicated by the fact that, for particular configurations, the dynamics equations may have multiple solutions (indeterminacy) or no solutions at all (inconsistency), (Painlevé, 1895 [24]). The problem of finding sufficient conditions for stability with friction has been explored by Pang and Trinkle [25, 29], who define *weak* and *strong* stability. A system of rigid bodies is *weakly stable* if there exist feasible contact forces such that there exists a solution to the equilibrium equations consistent with immobility. A system is *strongly stable* if there do not exist feasible contact forces consistent with motion. Balkcom and Trinkle [3] provide a complete, albeit exponential, solution for computing a set of external wrenches with regard to which for a single planar block is strongly stable.

Certain special cases of assemblies have been completely analyzed, including the popular unstacking game of Jenga, for which Zwick [32] gives simple sufficient conditions for stability in the case where there are three blocks in a row. Baraff *et al.* show how to find a stable orientation for a frictionless assembly by solving a single linear program [4].

Although we focus on gravitational stability as the primary constraint during disassembly, kinematic constraints may also prohibit objects from being removed from a pile. Snoeyink has discovered a class of assemblies that "cannot be taken apart with two hands," that is, as one hand holds the structure, the other cannot move any proper subset of objects in it without violating a non-penetration constraint [28].

1.1.3 Dynamics

Determining the stability of an assembly is more complex than the traditional grasping problem, since every subset of objects may be considered to be in the grasp of some or all of the remaining objects. For the general case, a common approach is to compute the forces between bodies to determine the possible motions of the system. If the constraints on the system are equality constraints, *e.g.* joints between objects, dynamics formulations may partition the system's generalized coordinates into dependent and independent coordinates, thus determining the degrees of freedom of the system. Alternatively, a Lagrange multiplier approach may be taken, in which unknown forces are introduced into the system to maintain known constraints. A standard reference for these techniques is Shabana [26]. Linear-time methods for loop-free systems are known both for computing generalized coordinate accelerations of rigid bodies [14], and for computing Lagrange multipliers of rigid and deformable bodies [6]. In [6], Baraff also gives a $O(kn)$ Lagrange multiplier method for systems of n loop-free equality constraints and k additional one-dimensional constraints which induce loops.

Lötstedt was the first to show that if the system constraints are inequality constraints, *e.g.* contact forces, then the system dynamics can be modeled as a quadratic program [19] or a linear complementarity problem [18]. Baraff gives a fast algorithm for computing contact forces between nonpenetrating rigid bodies in three-dimensional systems with static and dynamic friction in [5].

1.1.4 Statics and Computation Reuse

If we are only interested in whether the structure is in static equilibrium, then we can use a statics formulation. We can express the system's statics equations as the constraints of a linear program (LP)

$$\mathbf{J}^T\mathbf{x} = \mathbf{F}^{ext} \quad (1)$$

$$\mathbf{x} \geq \mathbf{0}, \quad (2)$$

where $\mathbf{J}^T$ is the transpose of the system Jacobian and describes the constraints imposed on each body by the contact forces. If this LP has a feasible solution, then the structure it represents is weakly stable. We can determine feasibility via standard algorithms for solving LPs. The simplex method is one such algorithm, and excellent discussions of it can be found in both Dantzig [11] and Cormen *et al.* [9]. A variety of methods are known for reusing computation in simplex implementations. Starting simplex with an optimal basis from a similar problem may reduce the number of iterations it takes to find an optimal solution [16]. Within the same problem, there are ways to use an old basis to solve systems of equations involving the current basis [12] or to update LU-factorizations between pivots [15], which are described by Vanderbei in [31].

1.1.5 Applications

A number of applications for stacking are in manufacturing, including packing, assembly planning, and part nesting. Ayyadevara *et al.* have developed a stacking planner and methods for generating interference-free part configurations by minimizing a cost function [2]. Another application is robotic excavation. Singh *et al.* [27] describe a voxel model of soil to analyze stability during planning and execution of digs. For self-reconfiguring robots, Kotay *et al.* have precomputed motion sequences to ensure dynamic stability at each step [17]. Unsal *et al.* propose verifying that the center of mass of a self-reconfiguring robot held together by link joints remains over its convex hull as part of a decision-making algorithm for the robot's motions [30].

1.2 Approach

In this section, we briefly map out the problem space, explain how the work we propose falls into this space, and discuss how this conceptual framework might be extended to related areas that are beyond the scope of this project.

The steps involved in developing an effective unstacking planner fall into three major categories: improvements to the stability test, improvements to the graph search, and improvements to the model our unstacking planner is capable of analyzing. We view these as the major axes that define the problem space, and believe that further improvements to the theory or implementation our unstacking algorithm will fall along one of these axes. For example, along the

Table 1: Map of the problem space

Stability Test	Statics: LP		Dynamics: LCP	
Optimizations	None	Comp Reuse	None	Comp Reuse
Model (SS/3D/Fr)				
000	■	◯	■	■
001	■	◯	⊕	⊕
010	×	×	⊕	⊕
011	×	×	⊕	⊕
100	■	◯	■	■
101	×	×	◯	◯
110	×	×	×	×
111	×	×	×	×
Graph Search				
BFS	■	◯	■	■
DFS	■	◯	■	■
Culling	⊕	◯	⊕	◯
Stable Union	⊕	◯	⊕	⊕

■ - Implemented ⊕ - Will Implement ◯ - Some Analysis × - Minimal Discussion

stability-test axis, we have identified two algorithms for testing the stability of an assembly, one that makes use of the statics equations and one that makes use of the dynamics. Further improvements to the stability test could be made by introducing another algorithm, or identifying additional optimizations to the two we already know, such as exploiting sparse matrices in the problem formulation.

Along the graph search axis, we have identified several strategies for searching the disassembly graph and heuristics for improving the search. Additional heuristics could be added along this axis, as well as alternate approaches to exploring the graph, such as randomized searches. One consideration as we add graph search heuristics is how they affect the performance of the stability test. For example, if an optimized test relies on reusing computation, it may assume that a solution has been found for a parent node. If a search strategy breaks this assumption, we must account for the trade-off in efficiency with the stability test when evaluating the performance of the overall unstacking algorithm. It is for this reason that we initially focus on the test for stability by itself.

Improvements to the model our unstacking planner uses to represent assemblies will allow the planner to better analyze structures in the physical world. Model attributes we have identified include friction, three dimensions, and strong stability. We indicate the features included in a model with a binary number in which bits set to one signify that an attribute is present. In preliminary work, we have analyzed frictionless planar structures and planar structures with Coulomb friction, and have primarily focused on identifying whether they are weakly stable. We would like to extend our stability analysis to three dimensions with friction.

Table 1 shows the progress we have made in preliminary work and the areas we expect to address in future work. We divide the latter into two categories: those we intend to implement in code, and those we intend to explore theoretically. The table also shows areas that are outside of the scope of this project.

We focus on a simple model of stable unstacking for assemblies with Coulomb friction, and strong assumptions: we assume perfect knowledge of the structure's geometry and mass properties, and that objects can be removed instantaneously without disturbing other blocks. We give no means for determining *how* stable a structure is, where a typical grasping algorithm might present a grasp metric. In spite of these limitations, we believe the proposed work is an important first step in understanding and developing efficient methods for solving grasping problems of this degree of complexity.

2 Preliminary results and current status

We begin with a basic unstacking sequence algorithm: consider the set of all objects in a structure. Collect all possible subsets into a *disassembly graph*. Search the graph, testing the stability of each node as it is visited. Any path of stable nodes from start to goal is a stable unstacking sequence.

This naive algorithm can be improved in two ways: by optimizing the test for stability, and by reducing the number of subassemblies we test by improving the graph search. Because this algorithm sequentially tests structures that differ only by a single object, if a set of force magnitudes exists that show a substructure to be stable, that solution is related to the set of forces that make the next structure stable. Thus the test for stability lends itself to *computation reuse*, whereby the solution from a parent node in the assembly graph is used to quickly find a solution for the child assembly.

In this section, we discuss a test for stability that is based on the system dynamics. For the frictionless case, the algorithm we use is based on Baraff's algorithm for computing contact forces from dynamics equations, described in [5], which is equivalent to Dantzig's algorithm for solving linear complementarity problems. After briefly introducing the algorithm, we explain how it can be modified for the stacking problem, so that computation is reused from one step to the next. We then present an example of the algorithm as applied to a simple assembly, with the primary goal of illustrating how computation is reused in this algorithm. A secondary goal of the example is to review some of the details of Baraff's algorithm. Readers should refer to Baraff, [5], for a thorough treatment. We give an argument for the correctness of our modifications, and finally, discuss the results of the implementation.

2.1 Model

We begin by presenting the test for stability for a frictionless model, and then explain how it can be extended to a model with static friction. For each contact normal $\mathbf{c}_i$ between two bodies, let a_i be the relative acceleration between the bodies in the direction of $\mathbf{c}_i$, and let f_i be the magnitude of the contact force between them. If $a_i < 0$ the bodies are accelerating toward each other; if $f_i < 0$, the force between them is attractive. Since our model adopts a non-penetration constraint as well as a unilateral force constraint, we require $a_i \geq 0$ and $f_i \geq 0$ at each contact point. If the bodies are separating at a contact point, then there is no force between them. Conversely, if $f_i > 0$ the bodies remain in contact, so their relative acceleration is zero. Thus, a third constraint is that one of a_i or f_i must always be zero, or $a_i f_i = 0$. We collect a_i and f_i into the vectors $\mathbf{a}$ and $\mathbf{f}$, respectively.

The Newton-Euler equations describe the dynamics of the system, and give a the linear relationship between $\mathbf{f}$ and $\mathbf{a}$,

$$\mathbf{a} = \mathbf{A}\mathbf{f} + \mathbf{b}, \tag{3}$$

where

$$\mathbf{A} = \mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T \qquad \text{and} \qquad \mathbf{b} = \mathbf{J}\mathbf{M}^{-1}\mathbf{F}^{ext}. \tag{4}$$

$\mathbf{J}$ is the system Jacobian and describes the constraints imposed on each body by the contact forces, $\mathbf{M}$ is a block-diagonal matrix that describes the mass characteristics of the system, and $\mathbf{F}^{ext}$ is a vector that represents the net external force acting on the system.

Equation (3), along with the conditions

$$a_i \geq 0, f_i \geq 0 \qquad \text{and} \qquad f_i a_i = 0, \tag{5}$$

form a linear complementarity problem (LCP), and thus one method for computing contact forces is to formulate and solve the dynamics equations as an LCP [18].

2.2 The test for stability

This approach to computing contact forces in a structure is also a suitable test for the stability of an assembly – if a solution $(\mathbf{a}, \mathbf{f})$ is found such that 1) $\mathbf{a} = \mathbf{0}$ and 2) $\mathbf{f}$ satisfies the system's static equilibrium equations, $\mathbf{J}^T\mathbf{f} = \mathbf{F}^{ext}$, then the structure is weakly stable. (Figure 1 shows a structure for which the first condition, $\mathbf{a} = \mathbf{0}$, is satisfied but the

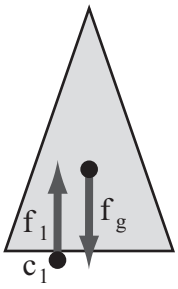


Figure 1: Example of a structure for which a necessary condition for stability, $\mathbf{Af} + \mathbf{b} = \mathbf{0}$, is satisfied, but the structure is not stable.

equilibrium equations are not.) If a solution is found such that some $a_i > 0$, then we know that the structure is not strongly stable.

The algorithm we use to solve this LCP is incremental. First, a value is found for f_1 that respects the constraints of the LCP and ignores all other elements of $\mathbf{f}$. Then, a value is found for f_2 that both satisfies the constraints on f_2, and allows the conditions on f_1 to be maintained. This may require modifying the value of f_1. The algorithm proceeds until all f_i in $\mathbf{f}$ have been found.

2.3 Computation reuse

We can reuse computation by exploiting the incremental nature of Dantzig's algorithm for solving LCPs. First, we solve the LCP for the smallest subassembly, thus finding the magnitudes of contact forces that makes this structure stable if one exists. Then, we add an object to the structure, and, *reusing the solution from the previous test*, we find values for the contact forces we have added to the structure, while maintaining the conditions on the previously considered contact forces. An illustration of this idea is shown in figure 2, and we will work through an example in section 2.4.

More formally, define a *phase* to be the test for stability of a single subassembly, and let S_i be the subassembly we test in phase i. Let σ be the set of objects and α be the set of contacts included in S_{k-1}. Let $\mathbf{a}_\alpha$ and $\mathbf{f}_\alpha$ be accelerations and forces at the contacts in α, and $\mathbf{A}_{\sigma\alpha}$ and $\mathbf{b}_\alpha$ the matrix and vector that relate them. Then

$$\mathbf{a}_\alpha = \mathbf{A}_{\sigma\alpha}\mathbf{f}_\alpha + \mathbf{b}_\alpha = \mathbf{0} \tag{6}$$

is a necessary condition for the stability of the subassembly. Let $\overline{\sigma}$ be the set of those objects in S_k that are not in S_{k-1}, with $\overline{\alpha}$ similarly defined for contacts. By storing the solution $\mathbf{f}_\alpha$ from phase $k-1$, instead of initializing $\mathbf{f}$ to zero at the beginning of phase k, we can populate all the values f_i for $i \in \alpha$ with those values from $\mathbf{f}_\alpha$, for which we know the normal force conditions hold. We then start the test with an equation of the form

$$\mathbf{A}\begin{bmatrix} \mathbf{f}_\alpha \\ \mathbf{0} \end{bmatrix} + \mathbf{b} = \begin{bmatrix} \mathbf{0} \\ \mathbf{a}_{\overline{\alpha}} \end{bmatrix}. \tag{7}$$

Thus, to determine the stability of S_k we need only calculate new values for contact forces in $\mathbf{f}_{\overline{\alpha}}$, while maintaining the conditions for forces in $\mathbf{f}_\alpha$.

2.4 Example

To explain the details of the algorithm with computation reuse, we show how it works on a specific example: a column of two blocks. The corresponding assembly is shown in the second frame of figure 2. The algorithm begins by considering the subassembly shown in the first frame. Since our model assumes point-edge contacts at polygonal vertices, there are two contact forces on the block. Contact normals $\mathbf{c}_1$ and $\mathbf{c}_2$ are drawn. For simplicity in the example, we take the mass matrix to be the identity matrix. Given the geometry of this structure, and gravity of 9.8 m/s in the negative y-direction, we compute the constraint equation to be

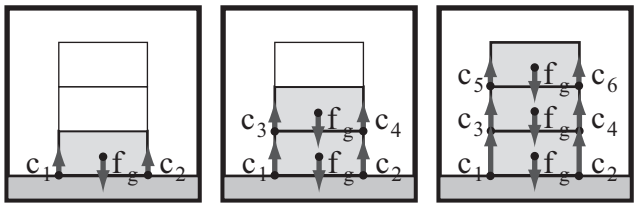


Figure 2: Sequential computation of contact forces for subassemblies. c_i indicates the ith contact force and f_g the gravitational force. The length of an arrow shows the relative magnitude of the force it represents.

$$\begin{bmatrix} 1.2601 & 0.7399 \\ 0.7399 & 1.2601 \end{bmatrix} \mathbf{f} + \begin{bmatrix} -9.8 \\ -9.8 \end{bmatrix} = \mathbf{a}. \tag{8}$$

We then calculate the $\mathbf{a}$ that results from setting $\mathbf{f} = 0$:

$$\mathbf{a} = \begin{bmatrix} -9.8 \\ -9.8 \end{bmatrix}. \tag{9}$$

The condition $\mathbf{a} \geq 0$ is not met, so we must alter the values in $\mathbf{f}$. We will find some f_d such that the corresponding $a_d < 0$ and increase it just enough that a_d will be pushed to zero. In Baraff's algorithm, this process is handled by the function *drive-to-zero*. In our example, since $a_1 < 0$, we consider a unit change to f_1 and determine that its direction must be positive in order to increase a_1. Then, we calculate $\Delta\mathbf{a} = \mathbf{A}\Delta\mathbf{f}$ and determine the size of the step necessary to push a_1 up to zero, by solving

$$s = -\frac{a_1}{\Delta a_1} = \frac{9.800}{1.2601} = 7.7772 \tag{10}$$

We would now evaluate how $\Delta\mathbf{f}$ and $\Delta\mathbf{a}$ affect values of f_i and a_i for which the LCP conditions have already been enforced, and limit s in order to maintain the normal force conditions for these contacts. There are not any prior values in this example, so we simply update $\mathbf{f}$ and $\mathbf{a}$ according to

$$\mathbf{f} = \mathbf{f} + s\Delta\mathbf{f} \qquad \text{and} \qquad \mathbf{a} = \mathbf{a} + s\Delta\mathbf{a}. \tag{11}$$

We obtain

$$\mathbf{f} = \begin{bmatrix} 7.7772 \\ 0.0000 \end{bmatrix} \qquad \text{and} \qquad \mathbf{a} = \begin{bmatrix} 0.0000 \\ -4.0457 \end{bmatrix} \tag{12}$$

Since $a_1 \geq 0$ is now satisfied, we are finished with f_1. We add its index to the index set C to signify that contact 1 is "clamped," that is, the solution is such that $f_1 > 0$, $a_1 = 0$. Throughout the algorithm, we maintain index sets C and NC to keep track of which of a_i or f_i is zero for each contact i that we have considered.

Returning to $\mathbf{a}$, we see that it still contains elements that do not satisfy $\mathbf{a} \geq 0$ (specifically $a_2 < 0$), so we repeat this process for f_2. The following explanation includes details from Baraff's algorithm regarding how we maintain the constraints on previously calculated values.

We use index sets C and NC to determine $\Delta\mathbf{f}$. In general terms, for a unit increase of f_d, we would like to maintain $f_i = 0$ for all $i \in NC$, and $a_i = 0$ for all $i \in C$. Accordingly, we set $\Delta f_d = 1$, $\Delta f_i = 0$ for all $i \in NC$, and Δf_i such that $\Delta a_i = 0$ for all $i \in C$. In addition, we must maintain the conditions $f_i \geq 0$ and $a_i \geq 0$ for all i we have considered so far. If taking the full unit step in $\Delta\mathbf{f}$ would cause any of f_i or a_i to become negative, then we can only take some fraction of the step. Which is to say, we must find the smallest scalar $s > 0$ such that increasing $\mathbf{f}$ by $s\Delta\mathbf{f}$ causes either a_d to reach zero, or some index i to move between C and NC. We then take this step, update the index sets, and begin again the process of driving a_d to zero.

To find the values of $\Delta\mathbf{f}$ such that $\Delta\mathbf{a} = 0$ for $i \in C$, we solve the equation $\mathbf{A}_{CC}\mathbf{x} = -\mathbf{A}_{Cd}$, where $\mathbf{A}_{CC}$ is the submatrix of $\mathbf{A}$ that contains only rows and columns i of $\mathbf{A}$ for $i \in C$ and $\mathbf{A}_{Cd}$ is the dth column of $\mathbf{A}$ that contains only rows i for $i \in C$. Continuing with our example, $C = \{1\}$ gives

$$\mathbf{A}_{CC} = \begin{bmatrix} 1.2601 \end{bmatrix} \qquad \text{and} \qquad \mathbf{A}_{Cd} = \begin{bmatrix} 0.7399 \end{bmatrix} \tag{13}$$

Solving, we find $\mathbf{x} = \begin{bmatrix} -0.5872 \end{bmatrix}$, so

$$\Delta\mathbf{f} = \begin{bmatrix} -0.5872 \\ 1.0000 \end{bmatrix} \qquad \text{and} \qquad \Delta\mathbf{a} = \begin{bmatrix} 0.0000 \\ 0.8256 \end{bmatrix} \tag{14}$$

This small example is well-behaved, and we can take the full step required to push a_2 to zero without moving any indices between C and NC. Finding $s = 4.9$ gives

$$\mathbf{f} = \begin{bmatrix} 4.9 \\ 4.9 \end{bmatrix} \qquad \text{and} \qquad \mathbf{a} = \begin{bmatrix} 0.0 \\ 0.0 \end{bmatrix} \tag{15}$$

We add index 2 to the set C, and all of the LCP conditions are satisfied, so the test is complete for the one-block assembly. Evaluating the solution with the static equilibrium equations confirms that the structure is weakly stable. Thus, we may continue by testing the stability of this structure with an added block.

Reformulating the LCP for the two-block column,

$$\mathbf{A} = \begin{bmatrix} 1.2601 & 0.7399 & -1.2601 & -0.7399 \\ 0.7399 & 1.2601 & -0.7399 & -1.2601 \\ -1.2601 & -0.7399 & 2.5202 & 1.4798 \\ -0.7399 & -1.2601 & 1.4798 & 2.5202 \end{bmatrix}, \tag{16}$$

$$\mathbf{b} = \begin{bmatrix} -9.8 & -9.8 & 0 & 0 \end{bmatrix}^T. \tag{17}$$

This time, instead of starting to solve $\mathbf{Af} + \mathbf{b} = \mathbf{a}$ with $\mathbf{f} = 0$, we can take the values of f_1 and f_2 from the previous test, which to know to satisfy the normal force conditions, and assign them to the corresponding values in $\mathbf{f}$. We then begin with

$$\mathbf{f} = \begin{bmatrix} 4.9 \\ 4.9 \\ 0.0 \\ 0.0 \end{bmatrix} \qquad \text{and} \qquad \mathbf{a} = \begin{bmatrix} 0.0 \\ 0.0 \\ -9.8 \\ -9.8 \end{bmatrix}. \tag{18}$$

From here, solving the LCP is simply a matter of repeating the steps we described for previous iterations.

We note that this small example is well-behaved, and extensive pivoting is not required here as it might be in some cases. As with most pivoting algorithms, it is possible to construct worst-case problems that cause exponential running times. In practice, however, the performance of this stability test appears to be linear in the number of contacts considered for a given structure. Implementation results are discussed in more depth in section 2.7.

2.4.1 What have we learned?

We have shown in this example that computation reuse is possible for a column structure, and that we can accomplish this reuse simply by starting the second phase with the values that we found in the previous phase filled in. What should be clear following this example is that each time we finish a phase, we have a solution for all the forces in the structure that we have considered so far. In the next phase, we can use this as a starting point for solving the newly formed LCP for the superstructure. Because of this, we will only need to call *drive-to-zero* once for each of the new contact forces we add when we add a single block, instead of solving the entire LCP from scratch each time.

The number of contact forces we add in a phase is typically small – in the physical world it is unlikely that more than three vertices of a polygon will contact the edges of adjacent blocks – so we can assume that a constant number of contacts is added in every phase. Thus only a constant number of forces must be solved for in each phase, and the time spent on each phase is directly proportional to the time it takes to calculate a single contact force, given the size of the structure for the phase. Intuitively, as the structures we analyze grow larger, we save a lot of time by considering only a few contacts in each phase, rather than recalculating all the forces from scratch.

2.5 Static friction

We add static friction to the model by augmenting the normal force conditions with several *static friction conditions*. The force due to friction at a contact point always acts in a direction tangent to the contact surface. Let a_{F_i} be the acceleration due to friction in the tangent plane at the ith contact, and let f_{F_i} be the magnitude of the friction force. At this point, we relabel a_i and f_i as a_{N_i} and f_{N_i}, to signify that they are in the direction of the contact normal, in contrast to a_{F_i} and f_{F_i}. Having introduced this notation, we can state the static friction conditions as follows. If there is no frictional acceleration at some contact i, then f_{F_i} must lie within the friction cone of the ith contact. Otherwise, in order to maintain the condition of static friction, *i.e.* that the structure has zero velocity at every contact point, we require that f_{F_i} does the maximum possible work to counter the frictional acceleration. Finally, we require that the friction force and acceleration are in opposite directions, that is, f_{F_i} and a_{F_i} must have opposite signs.

These conditions are summarized mathematically as

$$|f_{F_i}| \leq \mu f_{N_i}, \qquad a_{F_i}(\mu f_{N_i} - |f_{F_i}|) = 0, \qquad \text{and } a_{F_i} f_{F_i} \leq 0. \tag{19}$$

Baraff describes the necessary changes to the algorithm to maintain these additional conditions in [5]. For our purposes, it suffices to say that the algorithm is essentially the same, except for the shape of $\mathbf{Af} + \mathbf{b} = \mathbf{a}$. Each of $\mathbf{A}, \mathbf{f}, \mathbf{b}$ and $\mathbf{a}$ now contains frictional variables for every contact. We can arrange rows and columns so that the problem has the form

$$\mathbf{A} \begin{bmatrix} \mathbf{f}_{N_1} \\ \vdots \\ \mathbf{f}_{N_n} \\ \mathbf{f}_{F_1} \\ \vdots \\ \mathbf{f}_{F_n} \end{bmatrix} + \mathbf{b} = \begin{bmatrix} \mathbf{a}_{N_1} \\ \vdots \\ \mathbf{a}_{N_n} \\ \mathbf{a}_{F_1} \\ \vdots \\ \mathbf{a}_{F_n} \end{bmatrix}. \tag{20}$$

The necessary condition for stability is no longer $\mathbf{a} = \mathbf{0}$. Let $\mathbf{a}_N$ be the upper partition of $\mathbf{a}$ containing a_{N_i} for all i, and let $\mathbf{a}_F$ be the lower partition. Then for the assembly to be stable, we must have $\mathbf{a}_N = \mathbf{0}$ and $\mathbf{a}_F$ such that for all $a_{F_i} \neq 0$, $f_{F_i} = \mu f_{N_i}$.

We can reuse computation in the static friction model in the same way we described for the frictionless case. Given a solution $(\mathbf{f}_\alpha, \mathbf{a}_\alpha)$ from phase $k-1$, where all elements of $\mathbf{f}_\alpha$ and $\mathbf{a}_\alpha$ satisfy both the normal force and static friction conditions, we can test the stability of S_k by finding just those values that satisfy these conditions for the elements of $\mathbf{f}_{\overline{\alpha}}$ and $\mathbf{a}_{\overline{\alpha}}$.

2.5.1 What have we learned?

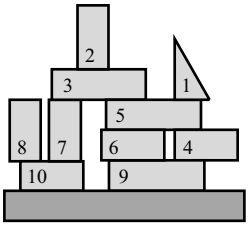
Rearranging $\mathbf{Af} + \mathbf{b} = \mathbf{a}$ into the above form shows that the problem consists of the one we know how to solve, namely $\mathbf{A}_N\mathbf{f}_N + \mathbf{b}_N = \mathbf{a}_N$, for which the necessary condition for stability is $\mathbf{A}_N\mathbf{f}_N + \mathbf{b}_N = \mathbf{0}$, and for which we have shown an example that computation reuse is possible.

The algorithm solves for the frictional forces and accelerations in essentially the same manner as it does for the normal forces and accelerations – by attempting a unit increase of the first force for which the friction conditions are not satisfied, determining whether any previously calculated forces limit the step size, pivoting as necessary, and continuing on to the next force that that does not satisfy the system constraints. Thus, frictional forces and accelerations can be dealt with and reused in essentially the same way as our normal forces and accelerations, by simply starting a phase with the solution from the prior phase filled in, and solving only for values of the forces that are added in the next phase.

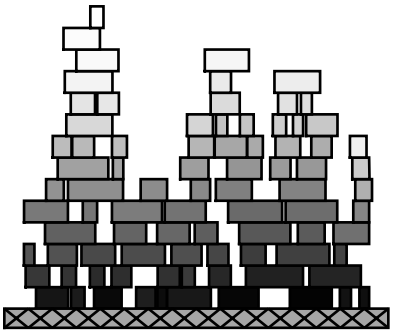
We show why that computation reuse is always possible in the following section.

2.6 Why we can reuse computation

In this section, we argue that computation reuse is always possible in Dantzig's algorithm for the LCP formulation of the dynamics equations. We do so first by showing that computation reuse is inherent in the algorithm for solving



(a) A simple structure, labeled with an unstacking order.



(b) A randomly generated assembly, where unstacking order is indicated by block color. Observe that blocks in the column on the far right must be removed before other blocks at higher levels in the structure.

Figure 3: Several assemblies and their unstacking sequences.

an LCP in general, since, at any stage, we begin with a solution we found previously by solving a smaller LCP. We then look at the special case where the LCP we are solving has the structure of our dynamics formulation. It turns out that we can partition the problem so that solving an LCP as formulated for a substructure will give us an intermediate solution.

2.6.1 General LCP

Let us assume that we have just completed a call to *drive-to-zero*. Then $\mathbf{Af} + \mathbf{b} = \mathbf{a}$ can be partitioned by

$$\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{0} \end{bmatrix} + \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{y} \\ \mathbf{z} \end{bmatrix} \tag{21}$$

where the top row contains the elements of $\mathbf{f}$ and $\mathbf{a}$ that we have considered so far, and the bottom row contains the elements that we have not yet considered. We note that the bottom partition of $\mathbf{f}$ is zero because these values remain from its initialization. Similarly, the value $\mathbf{z}$ in the bottom partition of $\mathbf{a}$ has been set to $\mathbf{A}_{21}\mathbf{x} + \mathbf{b}_2$ by the algorithm in the call to *drive-to-zero*. Since the algorithm has maintained the conditions $\mathbf{x} \geq 0$, $\mathbf{y} \geq 0$, $\mathbf{x} \cdot \mathbf{y} = 0$ throughout its operation, the solution $(\mathbf{x}, \mathbf{y})$ is a solution to the smaller LCP

$$\mathbf{A}_{11}\mathbf{x} + \mathbf{b}_1 = \mathbf{y}, \qquad \mathbf{x} \geq \mathbf{0}, \qquad \mathbf{y} \geq \mathbf{0}, \qquad \mathbf{x} \cdot \mathbf{y} = 0. \tag{22}$$

What is important here is to notice that we begin the next call to *drive-to-zero* with $\mathbf{f}$ and $\mathbf{a}$ filled-in with a solution to a smaller LCP.

2.6.2 LCP for a physical structure

Now let us look at the special case where $\mathbf{Af} + \mathbf{b} = \mathbf{a}$ is the LCP as formulated from the dynamic equations for a given structure; that is, when $\mathbf{A} = \mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T$ and $\mathbf{b} = \mathbf{J}\mathbf{M}^{-1}\mathbf{F}^{ext}$. Expanding $\mathbf{A}$ and $\mathbf{b}$, we can partition the problem in the same way as we do in equation (21):

$$\begin{bmatrix} \mathbf{J}_{11} & \mathbf{J}_{12} \\ \mathbf{J}_{21} & \mathbf{J}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{M}_{11}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{M}_{22}^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{J}_{11}^T & \mathbf{J}_{21}^T \\ \mathbf{J}_{12}^T & \mathbf{J}_{22}^T \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{0} \end{bmatrix} + \begin{bmatrix} \mathbf{J}_{11} & \mathbf{J}_{12} \\ \mathbf{J}_{21} & \mathbf{J}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{M}_{11}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{M}_{22}^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{F}_1^{ext} \\ \mathbf{F}_2^{ext} \end{bmatrix} = \begin{bmatrix} \mathbf{y} \\ \mathbf{z} \end{bmatrix}. \tag{23}$$

Multiplying out the matrices in equation (23) and looking only at the top row, we get a smaller LCP of the form

$$\mathbf{B}\mathbf{x} + \mathbf{c} = \mathbf{y}, \qquad \mathbf{x} \geq \mathbf{0}, \qquad \mathbf{y} \geq \mathbf{0}, \qquad \mathbf{x} \cdot \mathbf{y} = 0 \tag{24}$$

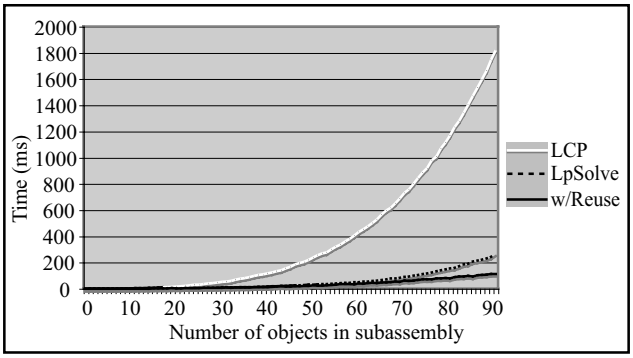


Figure 4: Time taken for a single stability test, by the size of the subassembly tested, by algorithm, for a 91-block pyramid.

where

$$\mathbf{B} = \mathbf{J}_{11}\mathbf{M}_{11}^{-1}\mathbf{J}_{11}^{T} + \mathbf{J}_{12}\mathbf{M}_{22}^{-1}\mathbf{J}_{12}^{T} \tag{25}$$

$$\mathbf{c} = \mathbf{J}_{11}\mathbf{M}_{11}^{-1}\mathbf{F}_{1} + \mathbf{J}_{12}\mathbf{M}_{22}^{-1}\mathbf{F}_{2}. \tag{26}$$

In order to solve this LCP, we must know the values for each of $\mathbf{J}_{11}$, $\mathbf{M}_{11}^{-1}$, $\mathbf{F}_{1}^{ext}$, $\mathbf{J}_{12}$, $\mathbf{M}_{22}^{-1}$, $\mathbf{F}_{2}^{ext}$. However, we can further simplify the problem by making the following observation. Recall that in the system Jacobian, rows correspond to contacts and columns to objects. Thus $\mathbf{J}_{11}$ represents the Jacobian of the substructure we have considered so far, since it contains both the contacts and the objects in this structure. $\mathbf{J}_{12}$, on the other hand, represents contacts from the substructure we have considered, and objects from the unconsidered substructure. But, since none of the contacts in the considered substructure are between objects in the substructure we have not yet considered, $\mathbf{J}_{12} = \mathbf{0}$. Thus $\mathbf{B}$ and $\mathbf{c}$ are simply

$$\mathbf{B} = \mathbf{J}_{11}\mathbf{M}_{11}^{-1}\mathbf{J}_{11}^{T} \tag{27}$$

$$\mathbf{c} = \mathbf{J}_{11}\mathbf{M}_{11}^{-1}\mathbf{F}_{1}. \tag{28}$$

The equation $\mathbf{Bx} + \mathbf{c} = \mathbf{y}$, then, is exactly the LCP matrix equation as formulated for a substructure of the complete structure we are considering. Since the order in which we consider indices while solving the LCP is arbitrary, we may partition the LCP so that any substructure is represented in the top row. Thus, we can solve the LCP for any substructure to obtain an intermediate solution to the LCP for the complete structure.

This argument holds for the problem with static friction as well, since in every call to *drive-to-zero*, we maintain the LCP conditions as well as the auxiliary static friction conditions.

2.7 Implementation and simulation results

Reusing computation in the LCP stability test makes the unstacking planning algorithm significantly faster. We have implemented the algorithm for the frictionless case, with and without computation reuse. Figure 4 shows the time difference between algorithms, for a single stability test of a given size.

Not only is computation reuse an improvement over the dynamically-based LCP stability test, it is faster even than solving the system statics as a linear program. We have implemented the unstacking algorithm with the LP stability test as well, employing `lp_solve` – a fast, open-source linear program solver written in C, that uses the revised simplex method [7] – to determine the feasibility of a structure's LP. Our implementation of the LCP algorithm with computation reuse, written in Java and without the development effort one might expect from an industrial-grade solver, consistently outperformed the `lp_solve` implementation.

We tested the different algorithms on a variety of examples, including columns, pyramids, and randomly generated structures[1]. To give a sense of how many stability test computations each structure required, using a depth-first search

[1]We describe the algorithm we use to randomly generate structures in section 3.5

Table 2: Average time by algorithm, measured over 100 executions.

Structure	Blocks	LpSolve	Baraff LCP	LCP + Reuse
Random	10	0.017 s	0.024 s	0.010 s
Column	50	0.483 s	2.987 s	0.469 s
Column	80	2.880 s	20.997 s	2.005 s
Random	84	9.396 s	72.099 s	5.532 s
Pyramid	91	5.260 s	38.825 s	3.074 s

strategy on the assembly graph, our algorithm found disassembly paths for the highly-structured column and pyramid assemblies after testing only n nodes. The same was not usually true for randomly generated structures, however. For the example shown in figure 3(b), 44 of the 129 total stability tests showed the tested subassembly to be unstable (this is because of the column stacked precariously on the far right of the assembly – these blocks must be removed well before other blocks of the same height). Table 2 presents the results of our tests, run on a 1.5 GHz PowerBook G4.

2.7.1 What have we learned?

The most important result from this comparison is that our implementation, which solves the system dynamics as a linear complementarity problem, runs faster than an implementation that solves the system statics as a linear program. This is surprising because LCPs are more complex than LPs, and dynamics equations are similarly more complex than statics equations. We expect the matrices in the dynamics formulation $\mathbf{Af} + \mathbf{b} = \mathbf{a}$ to be dense, since $\mathbf{A} = \mathbf{JM}^{-1}\mathbf{J}^T$ and the statics formulation $\mathbf{J}^T\mathbf{f} = \mathbf{F}^{ext}$ to be sparse. It also takes more time to calculate $\mathbf{A}$ and $\mathbf{b}$ in each phase, even though we are able to reuse some parts of these values from the previous phase.

In addition, this is a satisfying result because calculating stability for three-dimensional structures with friction is a problem that linear programs are not well-suited to deal with. One method for adapting linear programs to three-dimensional models with friction is by using linearized friction cones. This requires choosing a finite number of friction directions, and approximating the friction cone with some number of planes. Baraff's algorithm for computing contact forces with static friction, on the other hand, deals easily with the nonlinear constraint imposed by friction cones in three dimensions [5], and is thus better suited to the problem, as well as being a more efficient way of solving it.

3 Research Plan

The driving goals of this project are to:

- Develop an efficient algorithm for unstacking three-dimensional structures with a Coulomb model of friction that reuses computation as a means for optimizing the test for stability, and includes methods for reducing the number of nodes we must visit in our graph search.
- Analyze the complexity of the test for stability, and give an upper bound on the number of stability tests that must be conducted for a given assembly.
- Classify structures according to their unstacking characteristics, and develop efficient tests for inclusion in different classes.
- Develop software tools for efficient unstacking planning, and demonstrate their effectiveness by applying them to structures in the physical world.

3.1 Further improvements to the test for stability

The primary goal of this project is to find ways to improve upon the basic unstacking algorithm. We have identified two types of optimizations: computation reuse in the test for stability, and more clever traversal of the search graph. Of these two approaches, our foremost interest is in computation reuse. In the following sections, we identify two ways that we would like to improve upon the test for stability. First, we would like to modify Baraff's algorithm for computing contact forces in three dimensions so that it reuses computation, in much the same way that we have modified his planar algorithm. Second, we would like to find a way to reuse computation in an algorithm that solves the system statics.

3.1.1 Preliminary work

We have shown a method for reusing computation in the stability test for planar structures, and given an argument that computation reuse is always possible for systems with static friction; this is the focus of section 2. Yet, we are even more interested in results that apply to three-dimensional systems with friction. We believe it is possible to show that computation can always be reused in solving the complementarity problem for three dimensions. Does this add complexity beyond the increase in the size of the system due to the added dimension? In three dimensions, the Coulomb friction law does not completely specify the direction of friction at a contact point. Does a computation reuse approach give us different solutions than a less efficient approach? Is there an efficient way to guarantee that our algorithm finds an unstable solution if one exists?

In addition to considering system dynamics, we have tried to reuse computation in the statics formulation, but have to date been unsuccessful. Our approach has been similar to the method used in the LCP formulation. Consider the LP for a subassembly of one block. Solve for the contact force magnitudes using the simplex method. Add an object to the structure, and add the necessary rows and columns to the LP, applying to them the affine transformation that would have been applied had they been present in the LP from the beginning of the problem.

We have shown that this approach works for a simple example, but not the general case. An alternative approach is to convert the LP to an LCP and solve the LCP in a manner that reuses computation. The following is a discussion of how to convert an LP to an LCP, as presented by Cottle in [10]. Consider an LP in the primal-dual form. In the primal LP, the goal is to find $\mathbf{x}$ such that the objective function $\mathbf{z}_p$ is minimized:

$$\mathbf{A}\mathbf{x} \geq \mathbf{b}, \qquad \mathbf{x} \geq \mathbf{0}, \qquad \mathbf{z}_p = \mathbf{c}\mathbf{x}. \tag{29}$$

In the dual LP, the goal is to find $\mathbf{y}$ such that the objective function $\mathbf{z}_d$ is maximized:

$$\mathbf{y}\mathbf{A} \leq \mathbf{c}, \qquad \mathbf{y} \geq \mathbf{0}, \qquad \mathbf{z}_d = \mathbf{y}\mathbf{b}. \tag{30}$$

The duality theorem of linear programming states that $\min \mathbf{z}_p = \max \mathbf{z}_d$ when the primal and dual systems are both feasible. Thus, we must find a solution such that

$$\mathbf{y}\mathbf{b} = \mathbf{c}\mathbf{x}. \tag{31}$$

We can convert the LP inequalities to equalities by introducing slack variables $\mathbf{v}$ and $\mathbf{u}$, subject to nonnegativity constraints

$$\mathbf{A}\mathbf{x} - \mathbf{v} = \mathbf{b}, \qquad \mathbf{v} \geq \mathbf{0}, \qquad \mathbf{x} \geq \mathbf{0}, \tag{32}$$

$$\mathbf{A}^T\mathbf{y} + \mathbf{u} = \mathbf{c}, \qquad \mathbf{u} \geq \mathbf{0}, \qquad \mathbf{y} \geq \mathbf{0}. \tag{33}$$

Then, we can convert the LP to the LCP

$$\mathbf{B}\mathbf{z} + \mathbf{q} = \mathbf{w}, \qquad \mathbf{z} \geq \mathbf{0}, \qquad \mathbf{w} \geq \mathbf{0}, \qquad \mathbf{z} \cdot \mathbf{w} = 0 \tag{34}$$

where

$$\mathbf{B} = \begin{bmatrix} \mathbf{0} & -\mathbf{A}^T \\ \mathbf{A} & \mathbf{0} \end{bmatrix}, \qquad \mathbf{z} = \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix}, \qquad \mathbf{q} = \begin{bmatrix} \mathbf{c} \\ -\mathbf{b} \end{bmatrix}, \qquad \mathbf{w} = \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix}. \tag{35}$$

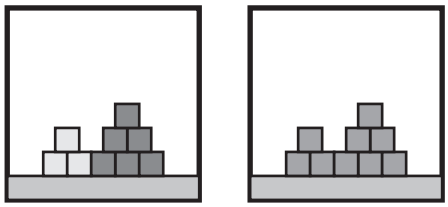


Figure 5: Non-intersecting stable subassemblies. Their union is weakly stable whether or not there are contact forces between them.

We believe that we can solve this LCP with an algorithm that reuses computation, in order to efficiently solve the LP. Dantzig's algorithm for LCPs, which we have shown to work on our dynamics formulation, is not likely to work on this formulation in the general case, as it relies on $\mathbf{B}$ being positive definite or positive semidefinite. Finding a solution to the general case is a goal of future work.

3.1.2 Remaining work

We will extend our argument that computation reuse is possible to three-dimensional systems. The first step is to work through the problem symbolically, identifying the structure of the complementarity formulation in three dimensions, and determining how different matrix partitions are affected by the addition of objects and contacts from one phase to the next. We also would like to show a second method for reusing computation for the planar case with static friction, by means of the LP conversion as described above.

3.2 Improvements to the graph search

In addition to finding ways to reuse computation in the test for stability, the basic unstacking algorithm can be improved by finding better paths through the disassembly graph.

3.2.1 Preliminary work

We have also made some preliminary steps toward improving the graph search in the stable unstacking sequence algorithm. The following are strategies we have identified that may allow us to test fewer nodes in the disassembly graph. We have not yet added any of these methods to our software implementation.

Culling rules. Since we are interested only in complete paths from root to leaf in the assembly graph, some culling of nodes may be possible during the search.

1. If a node is unstable, cut all of its edges.
2. If there is a stable path to a node from the root, cut all of the node's entrant edges.
3. If there is a stable path to a node from the leaf, cut all of the node's exit edges.

If a subgraph has no entrant edges or exit edges, the stability of the nodes in the subgraph need not be calculated. We have not determined whether efficient ordering of the search could be used to remove significant subgraphs from consideration.

Union of subassemblies. Intuitively, if two subassemblies are weakly or strongly stable and do not touch, their union is also weakly or strongly stable. In fact, even if the stable subassemblies touch, their union is guaranteed to be at least weakly stable.

Theorem 1 (Union of subassemblies) *The union of non-intersecting weakly-stable subassemblies is a weakly-stable assembly.*

Proof: Consider the contact graph for the assembly. Each subassembly is stable. Set the force magnitudes at contacts connecting subassemblies to zero. ■

Heuristics, search order, and existence. Since all successful stacking sequences are of length n, depth-first search is an obvious choice for searching the disassembly graph. Other strategies may be more effective for some types of structures. Search algorithms that give preference to stacking from the ground up also seem to be more likely to succeed quickly.

We expect artificial structures to have a stable unstacking sequence, but the set of structures that are stably unstackable may be small. Whether we can determine the existence of a stacking sequence for a structure more efficiently than we can construct the sequence is an open question.

We expect squat structures will be more likely to be stable than tall spires.

3.2.2 Remaining work

We will implement some of the heuristics we have presented here in code, and give some description of how they affect the performance of the unstacking algorithm. If time permits, we will identify additional heuristics for searching the graph.

3.3 Analysis of stability and planning algorithms

In the basic unstacking planning algorithm, both the stability test and the search path affect the complexity of finding an unstacking sequence. We have obtained empirical measures of the complexity of the stability test, but we would like to extend this analysis to include a more formal description of the average-case complexity. We would also like to find an upper bound on the number of substructures whose stability we must test.

3.3.1 Preliminary work

We have identified several ways to upper-bound the complexity of the graph search, although these methods are only faster than actually visiting each node if there are no loops in the structure.

Since a structure is not stable if it contains floating objects, the planning algorithm need not visit nodes whose contact graph is disconnected. Thus the number of connected, induced subgraphs of the contact that contain the ground node is an upper bound on the number of nodes tested in the disassembly graph. The following algorithm counts these subgraphs.[2]

SUBGRAPHS(G)

```
1  if G is a tree
2     then return TREESUBGRAPHS(G)
3     else Let u be a node adjacent to the ground node r
4          G' := the induced subgraph of G with u
                 removed, discarding any components not
                 connected to ground
5          G'' := G with r and u contracted
6          return SUBGRAPHS(G') + SUBGRAPHS(G'')
```

[2]The TREESUBGRAPHS problem was solved independently by the authors and Amit Chakrabarti; we use Chakrabarti's notation here for its elegance.

TREESUBGRAPHS(G)

1 Let r be the ground node, and root G at r
2 Compute `value(`x`)` for each node x as follows:
3 **if** x is a leaf
4 **then** `value(`x`)` := 1
5 **else** `value(`x`)` := $\prod_{y \text{ child of } x}(1 + \texttt{value}(y))$
6 **return** `value(`r`)`

If G is a tree, we can count these subgraphs in linear time. For a complete binary tree, this number can be characterized as a double exponential in the height of the tree [1]. If G contains cycles, we expect the exact counting algorithm to take exponential time.

In many cases, the number of stable nodes in the assembly graph is not an accurate indicator of the difficulty of the planning problem. For example, in the case of a set of n independently stable objects, all 2^n nodes in the assembly graph are stable, but since any sequence will work, since depth-first search returns a solution in linear time. Finding an efficient algorithm to estimate the complexity of disassembly planning for general structures is an open problem.

3.3.2 Remaining work

Describing the complexity of pivoting algorithms for the general case is difficult. For most pivoting algorithms, it is possible to construct problems that result in exponential running times. We do not know whether such problems correspond to physical structures our planner might encounter, although we believe this to be unlikely, as problems of this sort are considered to be not naturally-occurring.

We will explore how various structures affect the running time of the test for stability, in order to understand the complexity of the test for the general case, and to identify more efficient ways of reusing computation in the complementarity formulation. The first step is to look at theoretical results regarding worst-case analyses of pivoting algorithms and determine whether we can produce structures that have similar effects on the running time of our stability test. Whether or not this is possible, it should help us identify cases that cause our algorithm to pivot more than usual. The next step will then be to generate a suite of structures which affect our algorithm in different ways, and use this to develop an analysis of the stability test for the general case.

3.4 Classify structures by unstacking characteristics

In addition to identifying structures that affect the performance of our algorithm, we are interested in developing a more general classification of physical structures with regard to their unstacking characteristics. An open question is whether a stable unstacking sequence always exists. We expect that artificial structures belong to a class for which such sequences exist, since structures that have been assembled may be disassembled. Yet Snoeyink *et al.*have identified a class of objects that cannot be taken apart with two hands. This class might be a subset of a class in our taxonomy, if we determine that some structures cannot be unstacked due to problems with the kinematics of removal paths of objects in the structure.

The goal of this research area is to develop a coherent classification of structures, and find salient examples of structures in each class. Ideally, we would also like to find efficient means of identifying whether an assembly belongs to a particular class, as well.

Understanding the characteristics of our classes and ways to identify elements in them will allow us to match algorithms to the structures on which they perform well. For example, if we can quickly identify that a structure can be taken apart by removing blocks from the top, then there is no reason to analyze its stability using computationally expensive tests for intermediate subassemblies. If we know that a certain structure causes a pivoting algorithm to run inefficiently, we might use an alternate method. Developing such a classification may also lead us to greater understanding of deep questions about stability.

3.4.1 Preliminary work

We have roughly defined several classes we may consider for use in our classification: structures for which no unstacking sequence exists; structures that can be unstacked only by buttressing the structure at certain steps; structures that can be taken apart only by removing several objects at a time, like the Roman arch in figure 1(a); assemblies that are stable only with Coulomb friction; those that are stable in the frictionless case; structures that can be made stable only if their inequality constraints are made into equality constraints; and those that can be taken apart simply by sequentially removing blocks from the top, such as the column in figure 1(a).

Many of these classes have naive tests for inclusion implicit in their description. For example, a simple test for inclusion in the class of blocks that can be taken apart from the top-down might be: run the basic unstacking sequence algorithm, using breadth-first search on the disassembly graph. If any of the unstacking sequences found is a simple top-down sequence, accept. Otherwise, reject. For structures that can be taken apart by removing multiple blocks at a time, a (very inefficient) test for inclusion might be: run the unstacking algorithm, removing $i = 1$ block at a time. If no stable unstacking sequence is found, begin again, this time removing $i = 2$ blocks at a time. Repeat until $i = n - 1$. If some stable unstacking sequence is found for $i < n$, accept. Otherwise, reject.

3.4.2 Remaining work

Our work on developing a classification of structures based on their unstacking characteristics has just begun. We must expand our framework, identify the obvious tests for inclusion, and find ways of making these tests more efficient. We intend to generate structures that fall into the different classes, including three-dimensional models. If time allows, we will give unstacking sequence algorithms that run more efficiently on different classes of structures, as well.

Also of interest here would be an efficient algorithm for deciding where to place buttresses while disassembling structures that can only be taken apart with buttressing; however, this problem is outside the scope of this project.

3.5 Development of software tools

As we progress in identifying improvements to the unstacking sequence algorithm, it is important that we verify the feasibility and correctness of our optimizations by means of software implementation and simulation. To this end, we intend to develop an efficient unstacking planner for three-dimensional systems with static friction.

In preliminary work, we have developed an unstacking sequence planner for planar systems that may be configured to use one of several methods for its stability test. The planner is able to find unstacking sequences for structures with Coulomb friction, but computation reuse has been implemented only for the frictionless case. In addition to the unstacking platform, we have developed software for generating structures in different ways. The first parses an svg file that contains a structure drawn in Adobe Illustrator and converts it to a format our planner can use. The second generates a structure at random using the following algorithm. Starting with an empty world, pick a random offset on the first row and place a block there. Test the stability of the resulting structure. If it is unstable, remove the last block that was added and continue. Repeatedly pick rows and offsets at random, placing blocks and testing the stability of the assembly, until the structure is a prespecified number of rows high.

We will extend our current platform to deal with three-dimensional structures, and implement computation reuse for systems with static friction. We will also use the extended platform to explore the effect of heuristics on the efficiency of the search algorithm. Finally, our software is currently able to correctly identify unstacking sequences for structures of no more than ninety objects: it is unable to find a disassembly path for a column of one hundred blocks. We believe this is due to a bug in the graph search and not a limitation of the stability test. Identifying the source of this limitation and extending the planner's correctness for structures of over one hundred blocks is a goal of future work.

3.6 Physical demonstration of planner

We would like to demonstrate the effectiveness of our algorithms on structures in the physical world as well. To this end, we will design a set of "augmented reality" glasses for aiding users in identifying unstacking sequences for three-dimensional systems. A full prototype is beyond the scope of this project; however, we intend to purchase see-through

LCD glasses and implement a rudimentary platform for running our unstacking analysis software using them as a display.

Ideally, we would like to develop a means for obtaining input from the physical world by using a vision system to parse polygons and the points at which they contact from a digital photograph. If after some investigation it becomes apparent that this task is too time-consuming, since it does not significantly contribute to the primary goals of this project, we may resort to alternative means of input. This may include creating data files that represent physical structures by hand, by taking careful measurements of the assemblies as they are constructed.

4 Timeline

- **November**: Extend argument that computation reuse always works to the general case for three dimensions. Implement Baraff's fast contact force computation algorithm for the planar case with static friction. Modify the implementation to reuse computation.
- **December**: Measure the performance improvement of Baraff's algorithm for friction with computation reuse against the LpSolve implementation with friction, and against the Baraff algorithm without computation reuse. Develop a simple framework for delivering 3D structure data to planner. Evaluate and order see-through LCD glasses for unstacking planner hardware implementation.
- **January**: Implement Baraff's fast contact force computation algorithm for three dimensions with static friction. Modify the implementation to reuse computation, thus completing the implementation of the unstacking sequence planner. Measure the performance improvement of the algorithm with computation reuse against the basic algorithm.
- **February**: Build interface to allow unstacking planner to display unstacking sequences using LCD glasses as a monitor. Demonstrate proof of concept first for the planar case, and then for the 3D case. Begin work on structure classification by identifying major categories and example structures that fall into each of them. Perform a basic complexity analysis of the Baraff LCP stability test with computation reuse.
- **March**: Find a way to reuse computation in the LP formulation of the stability test, by means of a conversion to LCP, or alternate primal-dual approach. Complete work on structure classification by identifying formal tests for inclusion in each class, and at a minimum several algorithms that run more efficiently on structures from different classes.
- **Monday, April 3, 2006**: Begin writing thesis.
- **Monday, May 1, 2006**: Give thesis draft to advisor for feedback.
- **Monday, May 15, 2006**: Deliver thesis to committee.
- **Thursday , May 25, 2006**: Thesis defense.
- **Thursday, June 1, 2006**: Complete revisions suggested by committee.
- **Thursday, June 1, 2006**: Turn in official copy of completed thesis to Office of Graduate Studies, no later than 3PM.

5 Acknowledgments

The author would like to thank her advisor Devin Balkcom for all his input and support, as well as the committee, Amit Chakrabarti, Chris Bailey-Kellogg, and Peng Song, and the Dartmouth Robotics Laboratory, for their helpful feedback, and Amit Chakrabarti for his elegant independent solution to the subgraph counting algorithm and insight into the simplex method. This research program is a part of the Institute for Security Technology Studies, supported by Grant No. 2005-DD-BX-1091 awarded by the Bureau of Justice Assistance. The Bureau of Justice Assistance

is a component of the Office of Justice Programs, which also includes the Bureau of Justice Statistics, the National Institute of Justice, the Office of Juvenile Justice and Delinquency Prevention, and the Office for Victims of Crime. Points of view or opinions in this document are those of the author and do not represent the official position or policies of the United State Department of Justice.

References

[1] A. V. Aho and N. J. A. Sloane. Some doubly exponential sequences. *Fibonacci Quarterly*, 11:429–437, 1970.

[2] V. Ayyadevara, D. Bourne, K. Shimada, and R. Sturges. Interference-free polyhedral configurations for stacking. *IEEE Transactions on Robotics and Automation*, 18(2):147 – 165, April 2002.

[3] D. J. Balkcom and J. C. Trinkle. Computing wrench cones for planar contact tasks. *International Journal of Robotics Research*, pages 1053–1066, 2002.

[4] D. Baraff. Issues in computing contact forces for non-penetrating rigid bodies. *Algorithmica*, 10:292–352, 1993.

[5] D. Baraff. Fast contact force computation for nonpenetrating rigid bodies. In *Computer Graphics Proceedings*, pages 23–34, 1994.

[6] D. Baraff. Linear-time dynamics using Lagrange multipliers. In *SIGGRAPH*, pages 137–146, Aug. 1996.

[7] M. Berkelaar, K. Eikland, and P. Notebaert. lp_solve, open source (mixed-integer) linear programming system, May 2004. Version 5.1.0.0.

[8] J.-S. Cheong, K. Goldberg, M. H. Overmars, and A. F. van der Stappen. Immobilizing hinged polygons. In *IEEE International Conference on Robotics and Automation*, pages 876–881, 2002.

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press and McGraw-Hill, second edition, 2002.

[10] R. W. Cottle and G. B. Dantzig. Complementary pivot theory of mathematical programming. *Linear algebra and its applications*, 1:103–125, 1968.

[11] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, 1963.

[12] G. B. Dantzig and W. Orchard-Hayes. The product form for the inverse in the simplex method. *Mathematical Tables and Other Aids to Computation*, 8:183–195, 1954.

[13] M. A. Erdmann. On motion planning with uncertainty. Master's thesis, Massachusetts Institute of Technology, Aug. 1984.

[14] R. Featherstone. *Robot Dynamics Algorithms*. Kluwer Academic Publishers, 1987.

[15] J. Forrest and J. Tomlin. Updating triangular factors of the basis to maintain sparsity in the product form simplex method. *Mathematical Programming*, 2:263–278, 1972.

[16] R. Fourer. Linear programming frequently asked questions, 2000. http://www-unix.mcs.anl.gov/otc/Guide/faq/linear-programming-faq.html.

[17] K. Kotay, D. Rus, and M. Vona. Using modular self-reconfiguring robots for locomotion. In *Proceedings of Experimental Robotics IV*, 2000.

[18] P. Lötstedt. Coulomb friction in two-dimensional rigid-body systems. *Zeitschrift für Angewandte Mathematik und Mechanik*, 61:605–615, 1981.

[19] P. Lötstedt. Numerical simulation of time-dependent contact friction problems in rigid body mechanics. *SIAM Journal of Scientific Statistical Computing*, 5(2):370–393, 1984.

[20] M. T. Mason. *Mechanics of robotic manipulation*. MIT Press, 2001.

[21] B. Mishra, J. T. Schwartz, and M. Sharir. On the existence and synthesis of multifinger positive grips. *Algorithmica*, 2(4):541–558, 1987.

[22] B. Mishra and M. Teichmann. On immobility. In *Special Issue: Robot Kinematics, Laboratory Robotics and Automation: International Journal*, volume 4, pages 145–153. VCH Publisher, 1992.

[23] V.-D. Nguyen. Constructing force-closure grasps. *International Journal of Robotics Research*, 7(3), 1988.

[24] P. Painlevé. Sur les lois du frottemont de glissement. *C. R. Acad. Sci.*, 121:112–115, 1895.

[25] J. Pang and J. Trinkle. Stability characterizations of rigid body contact problems with Coulomb friction. *Zeitschrift für Angewandte Mathematik und Mechanik*, 80(10):643–663, 2000.

[26] A. A. Shabana. *Dynamics of Multibody Systems*. Cambridge University Press, third edition, 2005.

[27] S. Singh and S. Simmons. Task planning for robotic excavation. In *Proc. IEEE/RSJ International Conference on Intelligent Robot Systems*, July 1992.

[28] J. Snoeyink and J. Stolfi. Objects that cannot be taken apart with two hands. In *SCG '93: Proceedings of the ninth annual symposium on Computational geometry*, pages 247–256, New York, NY, USA, 1993. ACM Press.

[29] J. Trinkle, J. Tzitzouris, and J. Pang. Dynamic multi-rigid-body systems with concurrent distributed contacts: theory and examples. *Philosophical transactions on mathematical, physical, and engineering sciences*, 359(1789):2575–2593, 2001.

[30] C. Unsal and P. K. Khosla. Solutions for 3-d self-reconfiguration in a modular robotic system: Implementation and motion planning. In *Proceedings of SPIE, Sensor Fusion and Decentralized Control in Robotic Systems III*, November 2000.

[31] R. J. Vanderbei. *Linear Programming: Foundations and Extensions*. Kluwer Academic Publishers, 1996.

[32] U. Zwick. Jenga. In *SODA*, pages 243–246, 2002.