Working Paper Series

Computing Moral-Hazard Problems Using the Dantzig-Wolfe Decomposition Algorithm

WP 98-06

Edward S. Prescott
Federal Reserve Bank of Richmond

THE FEDERAL RESERVE BANK OF RICHMOND
RICHMOND ■ BALTIMORE ■ CHARLOTTE

Computing Moral-Hazard Problems Using the Dantzig-Wolfe Decomposition Algorithm

Edward S. Prescott*
Research Department
Federal Reserve Bank of Richmond

Working Paper 98-6
June 1998

Abstract

Linear programming is an important method for computing solutions to private information problems. The method is applicable for arbitrary specifications of the preferences and technology. Unfortunately, as the cardinality of underlying sets increases the programs quickly become too large to compute. This paper demonstrates that moral-hazard problems have a structure that allows them to be computed using the Dantzig-Wolfe decomposition algorithm. This algorithm breaks the linear program into subproblems, greatly increasing the size of problems that may be practically computed. Connections to dynamic programming are discussed. Two examples are computed. Role of lotteries is discussed.

1 Introduction

This paper computes static moral-hazard problems using the Dantzig-Wolfe decomposition algorithm. The method is applicable to single agent problems and some multiple agent problems where a source of private information is an agent's action. The technique is demonstrated in the classic principal-agent moral-hazard problem and variants of it.

I use the revelation principle framework developed in Myerson (1982) and Prescott and Townsend (1984). In this framework, lotteries are included in the contract space. As a consequence, the choice variable in the constrained maximization program is a probability distribution over the underlying sets of the relevant variables rather than deterministic functions. When the cardinality of each underlying set is finite, the program is a linear program.

There are two computational advantages to the linear programming formulation: First, the theory on computing linear programs is well developed and there are numerous linear programming

*The author would like to thank Jeff Lacker for helpful comments. The views expressed here are solely those of the author and do not necessarily reflect the views of the Federal Reserve Bank of Richmond or the Federal Reserve System.

codes available. Second, problems with arbitrary specifications of preferences and the technology can be computed. Unlike the analytical literature, there is no need to make highly restrictive assumptions on preferences and technology. The disadvantage to using linear programs is that as the cardinality of the underlying sets increase, the linear program quickly grows in size, making computation infeasible, either because of memory or software limitations.

In this paper, I demonstrate that the moral-hazard problem can be effectively computed using the Dantzig-Wolfe decomposition algorithm. The moral-hazard problems contains blocks of constraints and variables, one per action, that are only connected by a few constraints. A linear program with this structure is called *block angular* and can be computed using the Dantzig-Wolfe decomposition algorithm. Using this algorithm, I compute a linear program with 315,000 variables and 7057 constraints.

The decomposition algorithm is closely related to dynamic programming. Each block of constraints is computed separately, like the second-stage of a two-stage dynamic program. The results of these computations are summarized by state variables that are then used in a master program that includes the connecting constraints. Unlike a dynamic program, however, the algorithm iterates between the master program and the subproblems. This iterative process avoids the need to calculate the entire value function.

Economic reasons for lotteries

The literature does not usually incorporate lotteries into private information problems. One possible reason for their absence is the literature's emphasis on analytical results; lotteries complicate the analysis. Another possible reason is that they are not generally an explicit feature of contracts.

While explicit randomization is not necessarily frequently observed in contracts, neither are the complicated deterministic contracts that arise in models without lotteries. If a contractual device is potentially useful it should be included in the contract space unless there is an economic reason for excluding it. Furthermore, degenerate lotteries are a property of optimal contracts from some specifications of preferences and technology. For these specifications, any lotteries in the optimal solution are a result of the discrete grid. For a sufficiently fine grid, the solution will be a valuable approximation of the continuum case.

Some types of lotteries, however, may represent economy-wide effects rather than specific contractual devices. Their absence from contracts is less of an objection. For example, consider an economy with a continuum of *ex ante* identical agents who face an economy-wide resource

constraint. Each agent's effort is private information. Mathematically, this problem is identical to a principal-agent problem where the principal is risk-neutral and his reservation level of utility is zero.[1] The difference is in interpretation. Lotteries in this economy represent equilibrium fractions of the population. If in equilibrium, the solution to the moral hazard problem is a lottery over actions, then the lottery would represent a fraction of the population working each action. The implementation of the action lotteries need not be by an explicit contractual provision but could result from an unmodeled market feature.

Macroeconomic papers by Hansen (1985) and Rogerson (1988) use lotteries in a model with a continuum of agents in order to model the discrete nature of work. Prescott and Townsend (1996) and Lehnert (1997) interpret private-information models in similar frameworks. Lehnert (1997) finds that in a Solow growth model with moral hazard, leaving out lotteries substantially changes the growth and distribution effects in the economy. A final, practical, advantage of lotteries is that important sets are convex, allowing for some private information problems to be decentralized as competitive equilibria. See, for example, Prescott and Townsend (1984).

Why compute?

The literature has emphasized analytical properties of private information models and as a consequence, it has restricted itself to narrow assumptions on technology. In particular, the literature usually assumes that preferences are separable in consumption and effort and that the technology satisfies the monotone likelihood ratio property and the cumulative distribution function condition, assumptions sufficient to use the first-order approach. These assumptions are very restrictive and do not necessarily describe data. For economies without these assumptions, computing can help ascertain the qualitative properties of contracts. Prescott and Townsend (1997) contains several such examples.

Computing, however, is probably most important for using private information models to answer quantitative questions. Boyd and Smith (1994) is an excellent example of this. They examined the value of stochastic versus deterministic auditing, in the context of the costly state verification model. This model dates back to Townsend (1979) and has been extensively used in the auditing, corporate finance, banking, and macro literatures. It has received a lot of attention because when verification is restricted to deterministic strategies, debt contracts are optimal. This appealing prediction has led many, particularly in macroeconomics, to use this model. Un-

[1] I will present an example with these features later in the paper.

fortunately, a well-known analytical result is that stochastic verification dominates deterministic verification.[2]

To answer the question of how much better stochastic verification is than deterministic verification, Boyd and Smith (1994) calibrated a costly verification model using bankruptcy data. They calculated that the gains from stochastic verification are only on the order of 0.0003-0.03% of a firm's value. Assuming that the cost of implementing stochastic auditing is non-trivial, as they do, then the commonly observed debt contracts would be optimal. The question they asked was a quantitative one, and they needed to compute solutions in order to answer it.

The Computational Experiment

I evaluate the algorithm by the size of the programs and the time needed to compute them. There is a question of whether lotteries over grids are a good approximation to lotteries over continuum. I do not address this issue because the answer to this question depends the economic application. Furthermore, discrete grids are sometimes the most natural economic specification. The methods developed in this paper, however, do allow problems with finer grids to be computed than previously possible. As a consequence, some problems that once were inadequately addressed due to grid limitations are now manageable.

The paper proceeds by developing notation for the moral hazard problem. This notation is used to write the moral hazard problem as a linear program. Often, I will refer to this program as the global problem. Next, the Dantzig-Wolfe decomposition algorithm is developed. In my description, I emphasize the connections to dynamic programming and discuss the role of action lotteries. Afterwards, two examples that do not satisfy the usual assumptions made in the literature are computed. Solutions and computational results are reported. Finally, extensions of the technique to more complicated models are discussed.

2 The Moral-Hazard Problem

There are several stages to the moral hazard problem, once the principal and the agent have agreed to a contract.

1. The principal recommends an action;

2. The agent takes a hidden action;

[2]See for example Townsend (1979, 1987) and Mookherjee and Png (1989).

3. The publicly observed output is realized; and

4. Consumption is distributed to the agent.

Let C, Q, and A be the sets of consumptions, outputs, and actions, respectively. Denote elements of these sets as $c \in C$, $q \in Q$, and $a \in A$. The set of feasible points is $P \subset C \times Q \times A$. (Often, $P = C \times Q \times A$.) The agent's preferences are defined over P by the utility function $u(c, a)$. The principal consumes the surplus, receiving utility $w(q - c)$. The action taken by the agent affects the probability distribution of output according to the function $p(q|a)$.

Program without lotteries

In the standard formulation of the principal-agent problem, lotteries are not allowed. The principal deterministically recommends an action and the compensation schedule $c(q)$ is also deterministic. The problem is written, under the assumption that Q contains a finite number of elements, as

$$\max_{a, c(q)} \sum_q p(q|a) w(q - c)$$

$$\text{s.t.} \quad \sum_q p(q|a) u(c(q), a) \geq U \quad \text{and,}$$
$$a \text{ solves } \max_{\tilde{a}} \sum_q p(q|\tilde{a}) u(c(q), \tilde{a}).$$

The first constraint guarantees the agent U utils. It is called a participation constraint or sometimes a reservation utility constraint. The second constraint is the incentive compatibility constraint. It can also be written $\sum_q p(q|a) u(c(q), a) \geq \sum_q p(q|\hat{a}) u(c(q), \hat{a})$, $\forall \hat{a} \in A$. When A is a continuum there is an uncountable number of these constraints. As a consequence, there is a substantial literature concerned with conditions under which it is valid to replace the incentive compatibility constraints with the much simpler first-order condition to the agent's maximization problem. See, for example, Rogerson (1985).

Program with lotteries[3]

Before introducing lotteries, I assume that the sets C, Q, and A each contain a finite number of elements. This assumption is only necessary for computation, not to use lotteries. Furthermore, I assume that $p(q|a) > 0$, $\forall a, q$, that is, each output has non-empty support. This assumption is not necessary for the techniques used in this paper but it simplifies the notation.

[3]See Myerson (1982), Prescott and Townsend (1984), and Townsend (1993) for other derivations of the linear program.

When lotteries are allowed into the contract space, the principal chooses probability distributions over recommended actions and compensation schedules. The principal's probability distribution over the set of recommended actions A is described by $\pi(a)$. Because it is possible for different recommended actions to be recommended, the compensation schedule is now conditioned by the recommended action as well as the output. The compensation schedule is the conditional probability density function $\pi(c|q,a)$. It should be noted that deterministic contracts are still feasible in this contract space. They are simply degenerate probability distributions.

The two sources of randomization provide different benefits. Randomization in compensation may improve welfare if utility functions have convex portions. It may also weaken incentive constraints if preferences are non-separable. For example, if the action affects risk-aversion then randomization in compensation may be an effective way to implement actions. Randomization in the recommended action may also improve welfare, though not in the same way. Later, when the dynamic program is introduced, I will demonstrate how action lotteries improve welfare.

Placing $\pi(a)$ and $\pi(c|q,a)$ directly into the previous program does not create a linear program. With some algebraic manipulations, however, it can be turned into a linear program where the joint distribution $\pi(c,q,a)$ is the choice variable. The joint distribution is related to the principal's choice variables by the identity

$$\pi(c,q,a) = \pi(c|q,a)p(q|a)\pi(a). \tag{1}$$

Technology constraints

The principal chooses $\pi(a)$ and $\pi(c|q,a)$, not $\pi(c,q,a)$. With the addition of several constraints, however, the choice of $\pi(c,q,a)$ is equivalent to only choosing the marginal and conditional density functions. These constraints are

$$\forall \bar{q}, \bar{a}, \ \sum_{c} \pi(c,\bar{q},\bar{a}) = p(\bar{q}|\bar{a}) \sum_{c,q} \pi(c,q,\bar{a}). \tag{2}$$

If a $\pi(c,q,a)$ that satisfies (2) is chosen, the principal has implicitly chosen $\pi(a)$ and $\pi(c|q,a)$ but not the exogenous $p(q|a)$.

Incentive constraints

By the revelation principle, the recommended action a needs to be a solution to the agent's decision problem. For any action recommended with positive probability, $\pi(a) > 0$, a must satisfy

$$\sum_{c,q} \pi(c|q,a)p(q|a)u(c,a) \geq \sum_{c,q} \pi(c|q,a)p(q|\hat{a})u(c,\hat{a}), \ \forall \hat{a} \in A. \tag{3}$$

On the right-hand side of (3), the deviating action $\hat{a}$ enters into utility and the technology. It does not enter into the compensation schedule because the recommended action, not the action actually chosen by the agent, is the input into the compensation schedule.

To make these constraints linear in $\pi(c,q,a)$, first substitute $\pi(c,q|a) = \pi(c|q,a)p(q|a)$ into the incentive constraints to obtain

$$\sum_{c,q} \pi(c,q|a)u(c,a) \geq \sum_{c,q} \pi(c,q|a)\frac{p(q|\hat{a})}{p(q|a)}u(c,\hat{a}),\ \forall \hat{a} \in A.$$

The term $\pi(c,q|a)\frac{p(q|\hat{a})}{p(q|a)}$ is the joint probability that the agent receives the pair (c,q) given that a was recommended but the agent instead takes action $\hat{a}$. The second step in making these constraints linear is to multiply both sides of the equation by $\pi(a)$. The incentive constraints are then

$$\sum_{c,q} \pi(c,q,a)u(c,a) \geq \sum_{c,q} \pi(c,q,a)\frac{p(q|\hat{a})}{p(q|a)}u(c,\hat{a}),\ \ \forall a, (\hat{a} \neq a) \in A \times A. \tag{4}$$

Because $\pi(a) = 0$ for actions that are never recommended, these constraints are trivially satisfied by actions that are recommended with zero probability. Finally, because the cardinality of A is finite, there is a finite number of incentive constraints.

Probability constraints

The following set of constraints ensures that $\pi(c,q,a)$ is a probability measure.

$$\sum_{c,q,a} \pi(c,q,a) = 1, \text{ and } \forall c,q,a,\ \ \pi(c,q,a) \geq 0. \tag{5}$$

The principal-agent problem, with lotteries, is

Program 1: Participation Constraint Problem

$$\max_{\pi} \sum_{c,q,a} \pi(c,q,a)w(q-c)$$

$$\text{s.t.} \sum_{c,q,a} \pi(c,q,a)u(c,a) \geq U, \tag{6}$$

$$(2),\ (4), \text{ and } (5).$$

Equation (6) is the participation constraint. Furthermore, this program is a linear program. There is a finite number of linear constraints, and there is a linear objective function.

Program 1 can be used to compute the Pareto frontier by solving the program for each feasible U. In a linear program, the set of feasible solutions is convex and the objective function is weakly concave. These two features imply that the space of feasible utilities is also convex. Consequently, the Pareto frontier can also be computed using the Planner's problem.[4] The Planner's problem makes the objective function a weighted sum of utilities and removes the participation constraint.

This alternative formulation is important for the algorithms developed in this paper. Furthermore, the Planner's problem is easier to compute. Consequently, it is sometimes a more advantageous formulation. For these reasons, I also write out the Planner's program.

Let $\lambda \in [0,1]$ be the agent's weight. The Planner's program is

Program 2: Planner's Problem

$$\max_{\pi} \sum_{c,q,a} \pi(c,q,a) \left[\lambda u(c,a) + (1-\lambda) w(q-c)\right]$$

$$\text{s.t. } (2), (4), (5).$$

This program is also a linear program. By solving it for each $\lambda \in [0,1]$ the Pareto frontier can be calculated.

Finally, it should be noted that there is yet a third method for calculating the Pareto frontier. The remaining method is to switch the principal's and agent's utility in the Participation constraint program and then solve the problem for the feasible range of principal's utility W. Later in the paper, I will present an example where this formulation is most appropriate for the economic problem.

Computation

A linear program in standard form is written

$$\max_{\mathbf{x} \geq 0} \mathbf{ex}$$
$$\text{s.t. } \mathbf{Ax} = \mathbf{b},$$

where $\mathbf{e}$ is a $(1 \times n)$ vector, $\mathbf{x}$ the choice variable is an $(n \times 1)$ vector, $\mathbf{b}$ is an $(m \times 1)$ vector, and $\mathbf{A}$ is an $(m \times n)$ matrix, often called the coefficient matrix and *not* related to the set of actions

[4] The literature rarely uses the Planner's formulation. There are two reasons. First, it is not necessarily the most natural economic specification. Second, without lotteries the incentive constraints do not necessarily define a convex set. In this latter case, the Planner's problem will not trace out the entire Pareto frontier.

A. Problems with inequality constraints can be converted into standard form by the use of slack variables.

The first step to computing a solution is to create the coefficients and the second step is to solve the resulting linear program. The latter step requires the use of linear programming code. Many codes, including the one I use in this paper, are based on the simplex algorithm.[5] Because one version of the simplex algorithm, the refined simplex algorithm, underlies the Dantzig-Wolfe decomposition it is necessary to briefly describe this algorithm.[6]

Refined simplex algorithm

Consider the system of linear equations $\mathbf{Ax} = \mathbf{b}$, where there are m equations, and n variables. For simplicity, assume that $m < n$ and that the rank of $\mathbf{A}$ is m. A basis of this system is m linearly independent columns. The *basic* variables are the m variables that correspond to these columns; the remaining variables are called *non-basic*. The constraint matrix can be partitioned into coefficients on the basis, $\mathbf{B}$, and coefficients on the rest of the columns $\mathbf{D}$. Because $\mathbf{B}$ is non-singular, there is a solution to the equation $\mathbf{x} = \mathbf{B^{-1}b}$. This solution is feasible if $\mathbf{x} \geq \mathbf{0}$. A property of this solution is that all non-basic variables are necessarily zero while the basic variables may or may not be zero. Geometrically, basic solutions are the extreme points of the set of solutions to $\mathbf{Ax} = \mathbf{b}$.

If an optimal solution to a linear program exists, then there is a basic feasible solution that is optimal. Simplex-based algorithms search among the basic solutions for an optimal feasible one. Starting with a basic feasible solution, the algorithm determines if entering a non-basic variable into the basis increases the objective function. If such a variable is found, the algorithm removes a basic variable from the basis and enters the new variable into the basis. If at some point, no such non-basic variable is found then the basic feasible solution is optimal. Geometrically, the algorithm moves through adjacent feasible extreme points until it reaches an optimal one.

Given a feasible basic solution, let $\mathbf{e_B}$ and $\mathbf{e_D}$ be the portions of the objective function corresponding to the basic variables and non-basic variables, respectively. The formula that the revised simplex algorithm uses to determine if the objective function can be increased is $\mathbf{e_D} - \mathbf{e_B B^{-1} D}$,

[5] Recently, there have been important developments in interior point algorithms. Discussion in the literature implies that for smaller problems simplex methods are faster in practice but for larger problems interior points method are comparable if not superior.

[6] See any linear programming textbook for a description of simplex algorithms. Luenberger (1973) is a good source.

or equivalently,

$$\mathbf{e_D} - \mu\mathbf{D}, \tag{7}$$

where $\mu = \mathbf{e_B}\mathbf{B^{-1}}$ are the dual variables (or simplex multipliers on the constraints). If any element of this vector is positive, then introducing the corresponding non-basic variable into the basis increases the value of the objective function. The algorithm then determines which basic variable should be replaced in the basis by the non-basic variable. The algorithm repeats this process until $\mathbf{e_D} - \mu\mathbf{D} \leq 0$. The basic variables that satisfy this condition are an optimal solution.

Computation limitations

There are two limitations for computation. The first is the storage required for the problem and the second is the speed of the code. Needless to say, increasing the size of the problem increases the storage required and generally increases the time it takes code to solve the problem.

One strategy is to calculate the matrices $\mathbf{e}$, $\mathbf{A}$, and $\mathbf{b}$, and then directly enter them into the linear programming code. While for some problems this is fine, memory limitations quickly matter. As the cardinality of P increases, the size of the linear program increases. If n_x denotes the cardinality of set X, then the number of variables in the program is $n = n_c n_q n_a$ variables plus $n_a(n_a - 1)$ slack variables. The number of constraints is $1 + n_a(n_a - 1) + n_a n_q$. For example, if there are 100 consumptions, 50 outputs, and 100 actions, then there are 500,000 variables, 9900 slack variables, and 14,901 constraints. Some commercial codes may be able to handle this size of problem, but it would still require an enormous amount of memory and computation, far exceeding the capabilities of my software and hardware.

In terms of memory requirements, the action set A is the most costly to increase in terms of the size of the program. Increasing the number of elements in this set increases the number of incentive constraints and technology constraints, as well as the number of variables. Increasing the number of outputs is also costly but unlike actions does not affect the number of incentive constraints. Increasing the number of consumptions is the least costly because the size of the consumption set only affects the number of variables.

3 Dantzig-Wolfe Decomposition

As described in the previous section, the cardinality of the grids is the primary limitation to computing solutions. In this section, I show that the moral hazard problem has a special structure

that can be computed using a variant of the Dantzig-Wolfe decomposition algorithm. This algorithm breaks the global program into subproblems. The subproblems are computed separately, greatly lowering the problems's memory requirements.

In the course of developing the algorithm, similarities to dynamic programming will be discussed. This connection is emphasized for two expositional reasons: First, the dynamic programming interpretation will make clear the value of action lotteries. Second, many readers are familiar with dynamic programming. Despite the similarities to dynamic programming, however, there is an important difference that lies at the heart of the algorithm's efficiency. The algorithm avoids computing the entire value function. Exactly how the algorithm does this will be discussed in some detail later, but there is an interesting interpretation in terms of searching over the space of Planner's weights.

$$\begin{bmatrix}
\mathbf{u_1} & \mathbf{u_2} & \mathbf{u_3} & . & . & . & \mathbf{u_n} \\
\mathbf{1} & \mathbf{1} & \mathbf{1} & . & . & . & \mathbf{1} \\
\mathbf{A_1} & & & & & & \\
 & \mathbf{A_2} & & & & & \\
 & & \mathbf{A_3} & & & & \\
 & & & . & & & \\
 & & & & . & & \\
 & & & & & . & \\
 & & & & & & \mathbf{A_n}
\end{bmatrix}$$

Figure 1: Structure of Program 1 constraint matrix.

Figure 1 describes the structure of the constraint matrix of Program 1, that is, the moral-hazard problem where the principal's utility is maximized subject to the agent receiving his reservation utility level. Each $\mathbf{A_i}$ is the portion of the coefficient matrix on the constraints and variables that are relevant only to action a_i. In terms of Program 1, the constraints corresponding to block $\mathbf{A_i}$ are the incentive constraints for an agent recommended action a_i and the technology constraints corresponding to action a_i. Coefficients in $\mathbf{A_i}$ are those on variables directly involving action a_i, that is, $\pi(c, q, a_i)$. All other coefficients in this row are zero because the values of variables $\pi(c, q, a_j)$, $j \neq i$, do not affect the feasibility of these constraints.

The remaining two constraints connect or link the blocks of constraints. The top one in the matrix is the participation constraint, the $\mathbf{u_i}$ are vectors of coefficients determining utility the

agent receives if action a_i is taken. The second constraint is the probability measure constraint. The coefficients in this constraint are one, as indicated by the vector $\mathbf{1}$. Finally, it should be noted that the constraint matrix for the planner's formulation is obtained by removing the participation constraint.

A constraint matrix that can be divided into blocks with only a few connecting constraints is called *block angular*. Dantzig and Wolfe (1960) first developed an algorithm that takes advantage of this structure. Description of the method can be found in many linear programming textbooks, such as Bertsimas and Tsitsiklas (1997).

The block-angular structure of the constraint matrix allows the problem to be separated into subproblems, one for each block, that are only connected by the connecting constraints. Much like a dynamic program, the subproblems are analyzed separately to calculate utilities that the principal and agent may receive if action a_i is recommended. Then, the maximal combination of these utilities is chosen in a way that satisfies the connecting constraints.

Each subproblem i is a small problem consisting only of variables related to action a_i, the $\mathbf{A_i}$ block of constraints, plus an additional constraint that the variables sum to one. This latter constraint normalizes the subproblem's allocation into *ex post* utility terms. Because the values of the right-hand side variables $\mathbf{b}$ are zero, this normalization does not affect the relative values of utility.[7] The subproblem's constraints rearranged are

$$\begin{aligned} \forall \hat{a} \in A, \quad \sum_{c,q} \pi(c,q,a_i)(u(c,a) - \frac{p(q|\hat{a})}{p(q|a)} u(c,\hat{a})) &\geq 0, \\ \forall \bar{q}, \quad \sum_{c} \pi(c,\bar{q},a_i)(1 - p(\bar{q}|a_i)) - \sum_{c,q \neq \bar{q}} \pi(c,q,a_i) p(\bar{q}|a_i) &= 0, \\ \sum_{c,q} \pi(c,q,a_i) &= 1, \end{aligned} \tag{8}$$

and a non-negativity constraint.

The set of allocations $\pi(c,q,a_i)$ that satisfies these constraints is a convex set with a finite number of extreme points. Since this set is bounded by the last constraint, the set is a convex hull of these extreme points. Call this set of extreme points P_i.

[7] Adding this constraint to the subproblem is a departure from the standard development of the Dantzig-Wolfe decomposition algorithm. The standard form would calculate not only the subproblem's contributions to utilities, but also its contribution to the probability measure constraint. There would also be a difference in the master program. The reason I depart from the standard presentation is that my variant makes clear the connections to dynamic programming, a technique most economists are familiar with and it demonstrates the role of action lotteries.

Feasible principal-agent utility pairs for this subproblem are calculated by evaluating the objective function and participation constraint over the convex hull of P_i. Because these utility functions are linear, however, this set can be represented as the convex hull of their utility functions evaluated at each point in P_i. Let X_i be the set of utilities evaluated at P_i, and call an element in this set x_i. Each point x_i is a two-dimensional vector listing the principal's and agent's utilities. I use the notation x_{w_i} and x_{u_i} to describe the principal's and agent's utility, respectively.

Once the utility points are enumerated for each subproblem, the master program can be solved. Let $\pi(x_i)$ be the probability of choosing x_i. The master program is

Master Program

$$\max_{\pi \geq 0} \sum_{x_i, i} \pi(x_i) x_{w_i}$$

$$\text{s.t.} \quad \sum_{x_i, i} \pi(x_i) x_{u_i} \geq U, \tag{9}$$

$$\sum_{x_i, i} \pi(x_i) = 1, \quad x_i \in X_i, \tag{10}$$

where the two constraints correspond to the connecting constraints in Figure 1. This program is a linear program, just as was the original formulation. The difference is that the variables $\pi(c, q, a_i)$ have been replaced by (x_{u_i}, x_{w_i}) and the $\mathbf{A_i}$ constraints have been replaced by the restriction that $x_i \in X_i$. Also, notice that the scaling done in the subproblem is rescaled here so that the $\pi(c, q, a_i)$ embedded in x_i is the same value as would result from Program 1. Finally, the optimality condition is

$$\forall i, \quad \mathbf{x_{u_i}} - \mu_w \mathbf{x_{w_i}} \leq \mu_p, \tag{11}$$

where μ_w is the dual variable for the participation constraint and μ_p is the dual variable for the probability measure constraint.

A simple example illustrates the basic idea behind the program. Consider an economy with two actions, $A = \{a_1, a_2\}$. Figure 2 is a graph in utility space that describes Y_i, the sets of feasible utilities for each possible action that may be implemented. The agent's utility is on the x-axis and the principal's utility is on the y-axis. The set Y_1, bounded by the solid line and the axes, is the set of feasible utilities obtainable if action a_1 is recommended. Points in the set X_1 are scattered

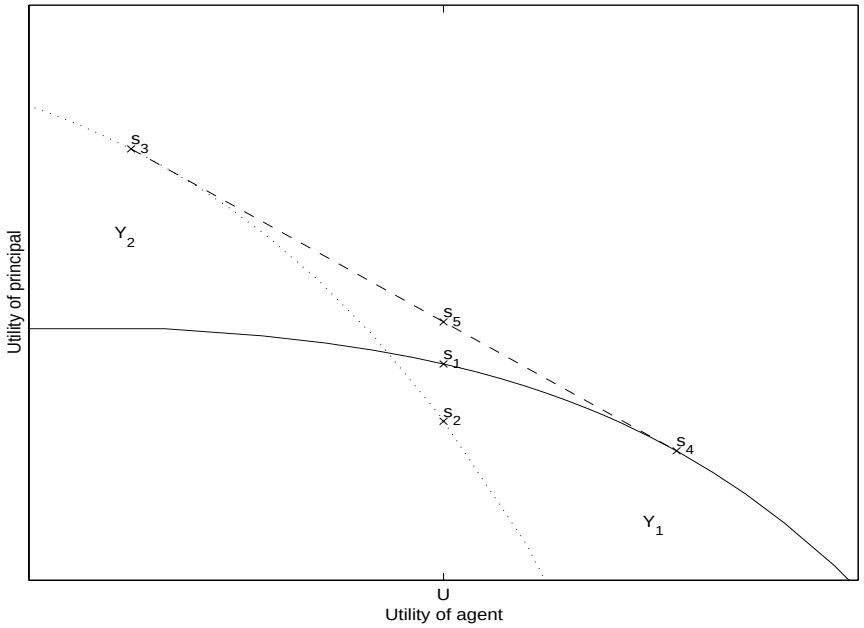

Figure 2: Feasible utility pairs for each implementable action.

along the boundary of Y_1. The set Y_2, bounded by the dotted line and the axes, is similarly defined for action a_2. These sets are independent of each other because of the block-angular structure of the constraints. While these sets are convex, their union need not be, as in the example.

Figure 2 describes the problem faced by the program, after the subproblem utilities have been calculated. The master program calculates the optimal combination of points from X_1 and X_2 that satisfies the two constraints. The set of feasible allocations in the master program is the convex hull of $X_1 \bigcup X_2$ (which is equal to $Y_1 \bigcup Y_2$). In Figure 2, the hyphenated line indicates the boundary of utility points that are feasible but are not in X_1 or X_2.

The optimal combination of points will be a lottery, possibly degenerate. If the optimal solution involves points from more than one set, then an action lottery is a property of the solution. As Figure 2 suggests, such a lottery may occur if the union of Y_1 and Y_2 is not convex. The discreteness of the action grid may cause such a non-convexity but, as demonstrated later in an example, more substantive aspects of the problem, such as the technology, may also cause such a non-convexity.

Figure 2 also illustrates the loss from disallowing action lotteries in the contract space. Without action lotteries, the corresponding master program would restrict itself to considering points that give at least U utils in sets Y_1 or Y_2. The point s_1 is the best such point in Y_1 and s_2 is the

best such point in Y_2 so the program would choose s_1 because it gives the principal higher utility. Because the action choice is deterministic, the participation constraint must hold, in a sense, pointwise. In contrast, the program with action lotteries would choose s_5, a convex combination of points s_3 and s_4. This point gives the agent U utils in expectation and is Pareto superior to s_1.

Computationally, the master program is a step backwards. Despite the reduction of the problem to two constraints, the number of variables has increased by an incredible amount, far too many to enumerate. The master program also demonstrates why computing a dynamic program is not a good strategy either. A dynamic program would solve for a value function, that is, the northeastern frontier of each subproblem in Figure 2, and then solve the master program. Solving for the value function, however, would require solving a huge number of subproblems, probably not much better than enumerating all of the vertices.

Dantzig-Wolfe decomposition

The Dantzig-Wolfe decomposition algorithm surmounts the problem posed by the large number of variables by only calculating values of x_i as they are needed. The idea is that most extreme points are not visited by the simplex routine. Consequently, most of the columns in the master program, that is the extreme points of the subproblems, do not ever need to be computed.

The algorithm starts with an initial feasible solution to the master problem. Let $\tilde{X}_i \subset X_i$ be a set consisting of the extreme points of this feasible solution. Next, the algorithm solves the master problem but with the restriction that $x_i \in \tilde{X}_i$. This is a very small linear program that has a basic feasible solution by assumption. The optimality condition (7) needs the dual variables μ and the coefficients of the non-basic variables $\mathbf{c_D}$ and $\mathbf{D}$. The dual variables are easily computed from the solution to the master problem because $\mu = \mathbf{B^{-1}b}$, but coefficients on the vast number of non-basic variables, however, are not known at this point.

It is at this step that the algorithm saves on storage requirements. Rather than calculating the coefficients on each of the non-basic variables, that is, calculating the huge number of extreme points of each subproblem, it calculates from each subproblem the extreme point that maximizes the objective function. These extreme points are computed by solving the following subproblem for each i

Subproblem Program

$$\max_{\pi(c,q,a_i)\geq 0} \sum_{c,q} \pi(c,q,a_i)(u(c,a_i) - \mu_w w(q-c))$$

subject to (8).

The solution to this problem is an $x_i^* \in X_i$, where

$$\forall x_i \in X_i, \;\; x_{u_i} - \mu_w x_{w_i} \leq x^*_{u_i} - \mu_w x^*_{w_i}.$$

This result is important because the optimality condition (11) for the master problem is $\forall i$, $\forall x_i \in X_i$, $x_{u_i} - \mu_w x_{w_i} \leq \mu_p$. Therefore, if for each i, the optimality condition is satisfied by x_i^* then it is satisfied by all $x_i \in X_i$ and the solution to the master problem is optimal.

If the optimality condition is not satisfied, then the solution to the master problem is not in $\tilde{X}_i$. The algorithm then proceeds by adding to each $\tilde{X}_i$ the previously calculated solutions to the subproblems. Then, the master problem is resolved and the algorithm continues until a solution is found.

The algorithm is powerful because it does not calculate all of the extreme points of each subproblem. During each iteration it calculates only the extreme points needed to find the solution to the subproblem. Furthermore, most of the computation is performed on the subproblems which are substantially smaller than the Program 1. If the algorithm does not require a large number of iterations it should be effective.

Connection to Planner's problem

There is a connection between the algorithm and the Planner's problem that provides intuition into the algorithm's workings. Before discussing this connection, it is necessary to develop some implications of the algorithm for the Planner's problem.

In the Planner's problem, Program 2, there is no participation constraint connecting the subproblems. Consequently, the optimization condition is

$$\lambda x_{u_i} + (1-\lambda) x_{w_i} \leq \mu_p.$$

Because there is one constraint the dual variable is trivial to calculate. In the master program for this problem, the basis consists one variable. Therefore, using the fundamental theorem of linear programming $\lambda x_{u_i} + (1-\lambda) x_{w_i} = \mu_p$, which is the optimality condition. Consequently,

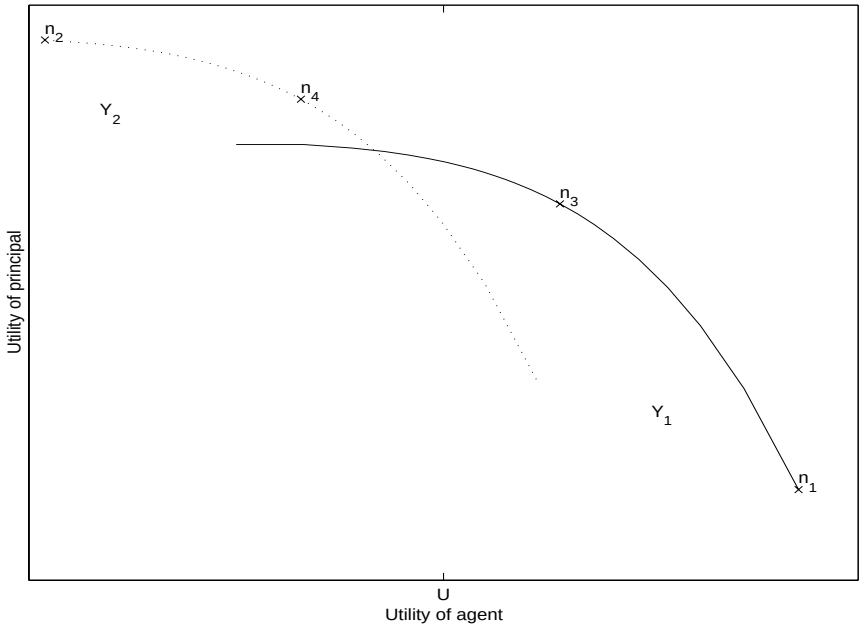

Figure 3: Points computed at successive iterations.

each subproblem only needs to be solved once. The results are then compared to determine which action or actions is optimal. An implication of this result, is that there exists a solution to the Planner's problem that is degenerate in action lotteries.

Returning to the Participation constraint program, the optimization condition is $x_{u_i} - \mu_w x_{w_i}$. This term can be interpreted in terms of a Planner's problem, where the Planner's weight on the agent is one and the Planner's weight on the principal is $-\mu_w$. The variable μ_w is called a simplex multiplier; at an optimum, it is the Lagrangian multiplier. The participation constraint is greater than or equal to in this problem so this multiplier is negative.

Consider Figure 3 which is similar to Figure 2. As before, Y_1 and Y_2 are the sets of feasible utility pairs for subproblems one and two, respectively. The algorithm starts with extreme points that can be used to form a feasible solution. On the first iteration, let $\tilde{X}_i = \{n_1, n_2\}$, where the feasible solution is a convex combination of n_1 and n_2. The algorithm proceeds by calculating the dual. In this problem, μ_w is the slope between the points. The agent's Planner's weight is 1 and the principal's is μ_w. The weights are not normalized to sum to one, in order to facilitate comparison with the optimality condition of the master program.

Next, the subproblems are solved with the new weights. The solutions to these subproblems yield points $n_3 \in X_1$ and $n_4 \in X_2$. If the optimality condition is not yet satisfied, these points

are added to $\tilde{X}_1$ and $\tilde{X}_2$ and the algorithm continues. Each iteration, the subproblems are solved using updated Planner's weights. Under this interpretation, the algorithm is searching through the space of Planner's weights for the weights that deliver the optimal feasible solution. Furthermore, each of these calculations picks points along the Pareto frontiers of the subproblem.

In terms of dynamic programming, the Pareto frontiers of the subproblems are the state variables (the agent's utility and the action) and the value function evaluated at the state variables (the principal's utility). The algorithm is selectively evaluating points along the value function. Compared with dynamic programming, the effectiveness of the algorithm depends on the number of times the value functions are evaluated. For the examples I compute in the next section, the algorithm only evaluates the value function a few times before it finds the optimal solution.

4 Examples

This section computes solutions to two economies. The examples are designed to illustrate the capabilities of the dynamic programming methods developed in this paper. The first example uses the Dantzig-Wolfe decomposition algorithm to solve a participation constraint problem. The example demonstrates the role that action lotteries may play. The second example looks at a Planner's problem to evaluate the qualitative properties of the optimal contract in a problem where the agent controls two-dimensions of the probability distribution of returns.

Computation

All computational results reported in this section were done on a Risc6000 workstation with 256MB of RAM. The programs were computed using code written in Matlab v5.1 and c. The programs created the constraints in Matlab and then called the linear programming routine as a function. The routine I used is lp_solve version 1.2 which is written in c. Matlab is not as fast as compiled code but it is easier to program in. Most of the numerical work involves the linear programming routine. The routine lp_solve is a linear programming routine based on the simplex routine. It uses sparse matrix techniques to save on memory and computational costs.

4.1 Example 1: Action Lotteries

Consider an economy with a continuum of agents, of measure one, each of whom has the same preferences and his own production technology. Each agent may only supply effort on his own plot and his effort is hidden from everyone else. Output is publicly observed, however. The probability

distribution of a project's return is only a function of the agent's effort.

The economy is a closed economy so no resources may flow in or out. Because there is a continuum of agents and project returns are independent, there is no aggregate uncertainty. The resource constraint for the economy is

$$\sum_{c,q,a} \pi(c,q,a)(c-q) \leq 0. \tag{12}$$

Formally, this constraint is identical to the utility function of a risk-neutral principal.

My goal is to solve for allocations that treat the agents *ex ante* identical so I maximize the expected utility of the representative agent. The allocation problem in this economy is

$$\max_{\pi \geq 0} \sum_{c,q,a} \pi(c,q,a)u(c,a)$$

subject to a resource constraint (12), incentive constraints (2), technology constraints (2), and a probability measure constraint (5). The problem is formally equivalent to maximizing the utility of the agent subject to the principal receiving a reservation level of utility.

Grid

The grids are

$$\begin{aligned} C &= \{0.0,\ 0.01,\ 0.02,\ ...,\ 2.0\}, \\ Q &= \{0.5,\ 1.5\}, \\ A &= \{0.05,\ 0.075,\ 0.06,\ ...,\ 1.95\}. \end{aligned}$$

Preferences

Agents' preferences are $u(c,a) = c^{0.5} + 0.8(2-a)^{0.5}$. Utility is concave in consumption and convex in effort.

Technology

The distinctive feature of this example is the technology. It contains a range of actions with increasing returns to scale and another range with decreasing returns to scale. The probability density function of the high output, $q = 1.5$, given the action is

$$p(q = 1.5|a) = \begin{cases} \frac{1-(a-1)^{0.2}}{2} & a < 1, \\ \frac{1+(a-1)^{0.2}}{2} & a \geq 1. \end{cases}$$

The probability density function of the low output, $q = 0.5$, is simply $p(q = 0.5|a) = 1 - p(q = 1.5|a)$. The probability density function is best understood by examining Figure 4.

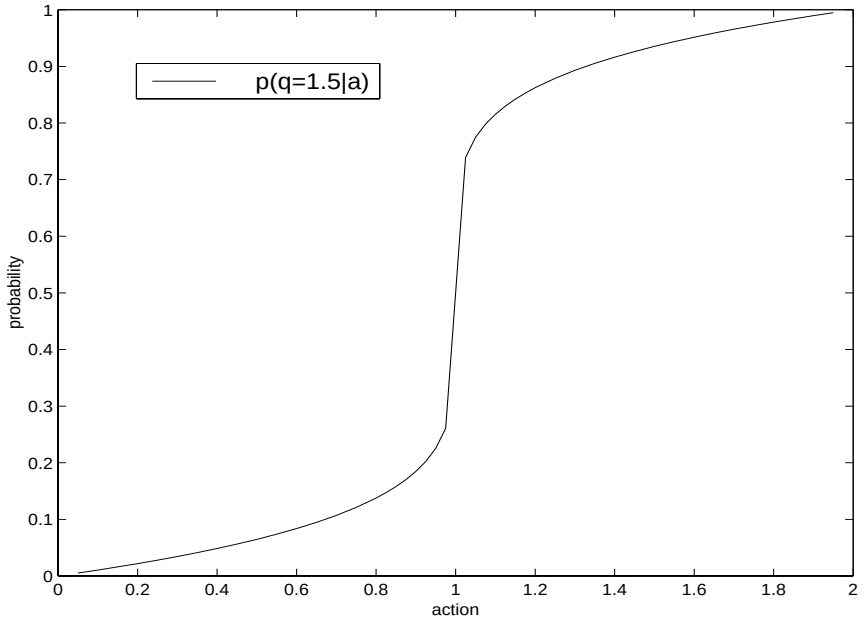

Figure 4: Probability of the high output given the action for Example 1.

Computation

There are 201 consumption grid points, 2 output grid points, and 77 action grid points. To solve this program directly, not using decomposition, would require solving a problem with 30,954 variables and 6008 constraints. While this is not a large program by many standards, it caused Matlab to run out of memory. By contrast, using the decomposition algorithm requires computing 77 linear programs with 402 variables and 79 constraints per iteration.

My code first created the constraint matrix for each subproblem and saved them to disk. Next, an initial feasible point was obtained by solving each subproblem using weights of (1,0) and (0,1). If a solution exists for the problem, these extreme pairs of Planner's weights produce a level of the resource constraint above and below zero. With these points in $\tilde{X}_i$, a solution to the master program will exist. On the first pass through the subproblems, only nine of the subproblems were found to have a feasible solution. (Not surprisingly, the infeasible ones were actions in the sharply increasing returns and the sharply diminishing returns range of A.) The algorithm kept track of feasible subproblems and only computed them on subsequent iterations. To use the decomposition algorithm, I set a stopping criterion of 10^{-8} on the optimum condition.

The program solved quickly. The total time it took to solve this program using the decomposition algorithm was 102.9 cpu seconds and it only took three iterations to converge. Almost half

of the time, 49 cpu seconds, was used by Matlab to create the constraint matrices. Each iteration was relatively quick, on the order of 3 to 4 cpu seconds. The reason for this is that only the nine subproblems with feasible solutions needed to be computed. On the first iteration, when all 77 were checked, the program took 39 cpu seconds.

Solution

The solution to the problem includes a lottery over the lowest action and a relatively productive action. The lottery is

$$\begin{aligned} \pi(a = 0.050) &= 0.0924, \\ \pi(a = 1.075) &= 0.9076. \end{aligned}$$

Slightly more than 9 percent of the population is assigned the low effort and nearly 91 percent are assigned the high effort.

For each of the two assigned actions, there is a compensation schedule. For the low action, the schedule is

$$\pi(c = 1.20|q = 0.5,\ a = 0.05) = 1,$$

$$\pi(c = 1.19|q = 1.5,\ a = 0.05) = 1.$$

No incentive constraint binds for lowest action in this problem, so analytically there should be full risk-sharing across the two outputs. As predicted by theory, risk-sharing is nearly perfect in this example. Any difference here is due to numerical approximation.

For the higher action, $a = 1.075$, there is an incentive problem. Consequently, consumption needs to depend on output. The compensation schedule for people recommended this action is

$$\begin{aligned} \pi(c = 0.54|q = 0.5,\ a = 1.075) &= 0.5311, \\ \pi(c = 0.55|q = 0.5,\ a = 1.075) &= 0.4689, \end{aligned}$$

$$\pi(c = 1.40|q = 1.5,\ a = 1.075) = 1.$$

The consumption lottery that occurs if the low output is realized is of no economic content. It is solely a result of the grid.

In terms of Figure 2, agents recommended the low action are receiving a utility point in set Y_1 and the other agents are receiving a utility point in set Y_2. Despite the convexity in effort disutility, the increasing returns in production are so strong in this economy that it is worth splitting the population into agents who work low and high amounts.

A surprising computational feature of the results was the low number of iterations between the master and subproblem programs. In general, problems may take more iterations to finish. I also computed the problem where the economy owes resources to the rest of the world, that is, $W = -0.5$. For this parameter specification, all agents are required to work a relatively high amount. The reason is that the program needs to extract resources to transfer to the rest of the world. Because agents work the high amount, the compensation schedule is output dependent and looks much like the schedule for those working high amounts in the previous experiment. Aside from the degeneracy in action lotteries, the main difference between this and the previous experiment is that this problem took 10 iterations to converge. The extra iterations added some time to the computing a solution, causing the program to solve in 125.0 cpu seconds.

4.2 Example 2: Two-dimensional action choice

In this example, I study a principal-agent problem where the agent chooses the mean and variance of a project's return. The agent is risk averse and receives disutility if he increases the mean or lowers the variance of the project's return. The principal is risk-neutral in consumption but he is extremely averse to low returns. A possible interpretation is that the principal loses control of the project if returns are below a certain amount, maybe because he has to make a debt payment. With these preferences the principal desires a high mean, low variance effort decision by the agent.

I formulate this problem as a Planner's program. My interest is the qualitative properties of the optimal contract and since the Planner's program is easier to compute, I follow this strategy. The Planner's weight was set to 0.5 in this example.

Grids

Let A_m and A_v roughly refer to the mean and standard deviation of the project, respectively. The grids are

$$
\begin{aligned}
C &= \{0.0,\ 0.02,\ 0.04,\ ...,\ 1.98\},\\
Q &= \{0.0,\ 0.04,\ 0.08,\ ...,\ 1.96\},\\
A_m &= \{0.70,\ 0.75,\ 0.80,\ ...,\ 1.10\},\\
A_v &= \{0.30,\ 0.35,\ 0.40,\ ...,\ 0.60\},\\
A &= A_m \times A_v.
\end{aligned}
$$

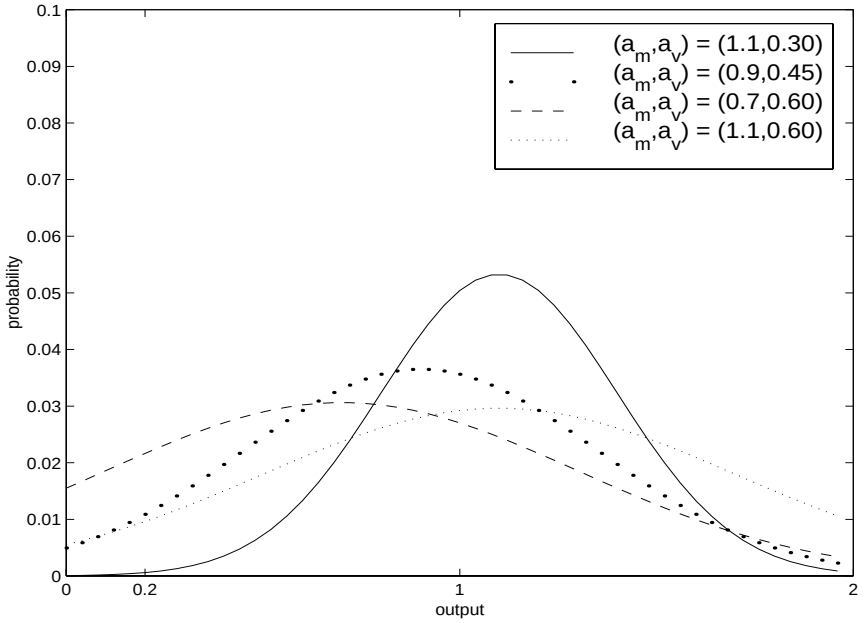

Figure 5: Probability density function for selected actions.

The grid of actions is a two-dimensional vector.

Preferences

Preferences are

$$u(c, a_m, a_v) = c^{0.5} + 0.25(2 - a_m)^{0.5} + a_v, \text{ and,}$$

$$w(c, q) = \begin{cases} q - c - 20 & q \leq 0.2 \\ q - c & q > 0.2. \end{cases}$$

The agent is risk-averse and dislikes high mean and low variance actions. The principal is risk-neutral but with an extreme aversion to low output.

Technology

The technology is a rough approximation to a normal distribution. For each (a_m, a_v), the probability density of output is obtained by evaluating the normal distribution for all Q and then normalizing so that $\sum_q p(q|a_m, a_v) = 1$. Figure 5 shows the distribution for a selection of the action pairs.

Solution

Because of the principal's aversion to low outputs, the principal wants an action that minimizes the probability of outputs less than 0.2. Accordingly, the solution is a degenerate lottery with

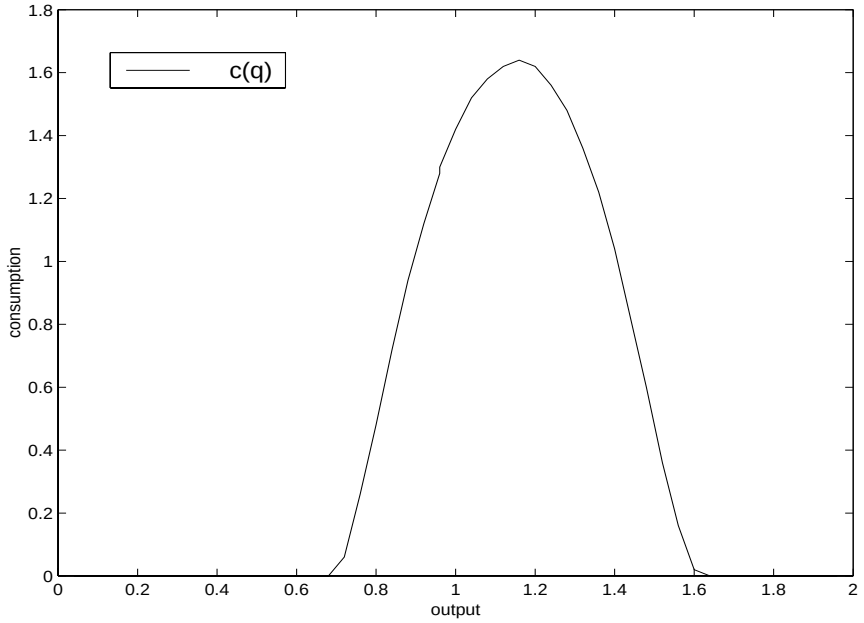

Figure 6: Consumption sharing rule for Example 2.

probability one on the action pair $(a_m, a_v) = (1.1, 0.3)$. The compensation schedule that implements this action pair needs to do two things. First, it has to ensure that the agent does not take a low mean action. Second, the schedule has to stop the agent from taking a high variance action. In this problem, these are potentially conflicting goals. For example, low consumption for high outputs rewards high variance actions and punishes low mean actions. The program weighs all of the incentive effects and implements the high mean, low variance action by punishing the agent for low and high outputs and rewarding him for intermediate outputs. Figure 6 shows the optimal compensation schedule which is non-monotonic.

Computational results

This program is a large linear program, with 315,000 variables and 7057 constraints. Solving it using the Dantzig-Wolfe decomposition algorithm requires solving 63 linear programs with 5000 variables and 113 constraints.[8] It took 181.6 cpu minutes to compute this problem. Slightly more than 10 cpu minutes was spent creating the constraints. The rest of the time was spent solving the subproblems.

The algorithm greatly increases the number of actions that can be included into the problem.

[8]For this problem, the limit on the number of actions I could compute globally was 8. At 9 actions, I ran into memory constraints.

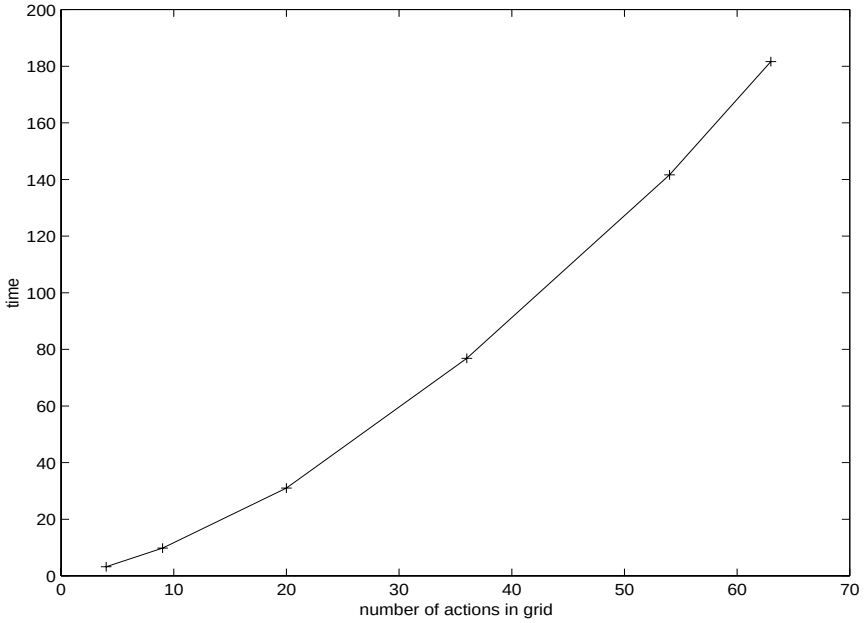

Figure 7: Computation time in cpu minutes as a function of the size of the action grid.

The nature of the subproblems suggest that the time to compute solution is roughly linear in the number of actions. As the number of actions increases, one more subproblem needs to be computed while each subproblem only increases by one constraint. Figure 7 shows the computation time in cpu minutes as a function of the size of the action grid. The x-axis lists the number of actions while the y-axis lists the cpu minutes required to compute each problem. In each experiment, a subset of A was used. Figure 7 demonstrates that for this example, the time required, while convex, is close to linear.

5 Extensions

Any linear program with a block-angular constraint matrix can be computed using the Dantzig-Wolfe decomposition algorithm. Several important extensions of the moral-hazard problems retain this structure, including some multiple-agent models and models with production inputs.

For a multiple-agent model to be block-angular, decisions made by one agent can only affect other agents through the connecting constraints. An example that satisfies this condition is one where agents' technologies are independent and there is a risk-neutral principal who has access to outside funds. Mathematically equivalent, is the problem where there is a finite number of

agent types, each of whom is a fraction of the population, that face an economy-wide resource constraint like in Example 1.

Another important class of models that the technique can be applied to is moral-hazard models where there is also a publicly observed input into the technology. In these models, there would be one subproblem per effort-input pair. The connecting constraints would depend on the specification of the problem. For example, if there was a closed economy that not only faced a consumption resource constraint, like (12) in Example 1, but also a resource constraint on the amount of the input available to the economy, then there would be an additional connecting constraint. The additional connecting constraint would create one more dual variable in the master program and add a term to the objective function, corresponding to the value of input resource constraint. If instead, the input is purchased by the principal at some price, then the master program would only have a resource constraint and probability measure constraint just like the master program in Example 1.

6 Conclusion

Linear programming is an important tool for computing private-information problems. The method can be used for problems with arbitrary specifications of preferences and technology. The Dantzig-Wolfe decomposition algorithm expands the capabilities of linear programming by greatly increasing the size of problems that may be computed. The algorithm was demonstrated by computing solutions to two examples. Both examples departed from the standard technology assumptions made in the literature.

References

[1] Bertsimas, Dimitris and John N. Tsitsiklis. *Introduction to Linear Optimization.* Belmont, Massachusetts: Athena Scientific, 1997.

[2] Boyd, John H. and Bruce D. Smith. "How Good are Standard Debt Contracts? Stochastic versus Nonstochastic Monitoring in a Costly State Verification Environment." *Journal of Business* 67 (1994): 539-561.

[3] Dantzig, G. B. and P. Wolfe. "The Decomposition Principle for Linear Programs." *Operations Research*, 8, (1960): 101-111.

[4] Hansen, Gary D. "Indivisible Labor and the Business Cycle." *Journal of Monetary Economics* 16 (1985): 309-27.

[5] Lehnert, Andreas. "Solow Growth Models and Moral Hazard." Manuscript, 1997.

[6] Luenberger, David G. *Introduction to Linear and Nonlinear Programming.* Reading, Massachusetts: Addison-Wesley Publishing Company, Inc, 1973.

[7] Mookherjee, Dilip and Ivan Png. "Optimal Auditing, Insurance, and Redistribution." *Quarterly Journal of Economics* 104 (May 1989): 399-415.

[8] Myerson, Roger B. "Optimal Coordination Mechanisms in Generalized Principal-Agent Problems." *Journal of Mathematical Economics* 10 (June 1982): 67-81.

[9] Prescott, Edward C. and Robert M. Townsend. "Pareto Optima and Competitive Equilibria with Adverse Selection and Moral Hazard." *Econometrica* 52 (January 1984a): 21-54.

[10] Prescott, Edward S. and Robert M. Townsend. "Theory of the Firm: Applied Mechanism Design." Manuscript 1997.

[11] Rogerson, Richard. "Indivisible Labor, Lotteries and Equilibrium," *Journal of Monetary Economics* 21 (January 1988): 3-16.

[12] Rogerson, William. "The First-Order Approach to Principal-Agent Problems." *Econometrica* 53 (1985): 1357-68.

[13] Townsend, Robert M. "Information Constrained Insurance: The Revelation Principle Extended." *Journal of Monetary Economics* 21 (1988): 411-450.

[14] Townsend, Robert M. *The Medieval Village Economy: A Study of the Pareto Mapping in General Equilibrium Models.* Princeton, N.J.: Princeton University Press, 1993.