CAEPR Working Paper
#2015-002

A Divide and Conquer Algorithm for Exploiting Policy Function Monotonicity

Grey Gordon
Indiana University

Shi Qiu
Indiana University

January 27, 2015

This paper can be downloaded without charge from the Social Science Research Network electronic library at <http://ssrn.com/abstract=2556345>

The Center for Applied Economics and Policy Research resides in the Department of Economics at Indiana University Bloomington. CAEPR can be found on the Internet at: <http://www.indiana.edu/~caepr>. CAEPR can be reached via email at caepr@indiana.edu or via phone at 812-855-4050.

**©2015 by Grey Gordon and Shi Qiu. All rights reserved.
Short sections of text, not to exceed two paragraphs,
may be quoted without explicit permission provided
that full credit, including © notice, is given to the source.**

A Divide and Conquer Algorithm for Exploiting Policy Function Monotonicity

Grey Gordon and Shi Qiu

January 27, 2015

Abstract

A divide and conquer algorithm for exploiting policy function monotonicity is proposed and analyzed. To compute a discrete problem with n states and n choices, the algorithm requires at most $5n + \log_2(n)n$ function evaluations and so is $O(n \log_2 n)$. In contrast, existing methods for non-concave problems require n^2 evaluations in the worst-case and so are $O(n^2)$. The algorithm holds great promise for discrete choice models where non-concavities naturally arise. In one such example, the sovereign default model of [Arellano \(2008\)](#), the algorithm is six times faster than the best existing method when $n = 100$ and 50 times faster when $n = 1000$. Moreover, if concavity is assumed, the algorithm combined with [Heer and Maußner \(2005\)](#)'s method requires fewer than $18n$ evaluations and so is $O(n)$.

1 Introduction

Monotone policy functions arise naturally in economic models.¹ In this paper, we provide a computational algorithm, which we refer to as “binary monotonicity,” that exploits policy function monotonicity using a divide and conquer algorithm. Our focus is on problems of the following form:

$$\Pi(i) = \max_{i' \in \{1, \dots, n'\}} \pi(i, i') \quad (1)$$

for $i \in \{1, \dots, n\}$ with an associated optimal policy g . While this is abstract, economic problems of this form abound. For example, consider the two period utility maximization problem $V(b, y) = \max_{b' \in \mathcal{B}} \log(-b' + b + y) + \log(b')$ where b and b' are some measure of savings constrained to lie in a grid $\mathcal{B}$, y is an endowment, and $\log(\cdot)$ is the period utility

¹For example, one of the most common state and choice variables is some measure of savings. It is then quite natural for the model to predict that wealthier agents save more in levels else equal, which results in a savings policy that is monotonically increasing in wealth. This is the case for the real business cycle model, the standard incomplete markets models of [Aiyagari \(1994\)](#) and [Krusell and Smith \(1998\)](#), the default models of [Arellano \(2008\)](#) and [Chatterjee, Corbae, Nakajima, and Ríos-Rull \(2007\)](#), and many others.

function. Letting $\mathcal{B}_j$ denote the j th element of the grid, one can create a separate problem for each y with $\Pi_y(i) = \max_{i' \in \{1, \dots, \#\mathcal{B}\}} \pi_y(i, i')$ and $\pi_y(i, i') = \log(-\mathcal{B}_{i'} + \mathcal{B}_i + y) + \log(\mathcal{B}_{i'})$. In this case, one has $V(\mathcal{B}_i, y) = \Pi_y(i)$.² Beyond this simple example, we show how the standard real business cycle model (with leisure) and [Arellano \(2008\)](#)’s model can be cast in this form. Letting an optimal policy associated with (1) be denoted g , we say g is monotonically increasing if $g(i) \leq g(j)$ whenever $i \leq j$. We restrict attention to monotonically increasing policies, but our results extend naturally to monotonically decreasing policies. Our method exploits monotonicity to solve (1) in no more than $3n' + 2n + \log_2(n)n'$ evaluations of π .

While concavity also arises naturally in economic models, for certain classes of models, it does not. Discrete choice models are the foremost example. In them, the value function is the upper envelope of a finite number of “subsidiary” value functions conditional on the discrete choice. Even if the subsidiary value functions are concave, the upper envelope is generally not. For instance, in the [Arellano \(2008\)](#) model we consider in this paper, the value function for an indebted sovereign (for whom it is refeasible to repay debt) is $V(b, y) = \max_{d \in \{0, 1\}} dV^d(y) + (1 - d)V^{nd}(b, y)$ where b is a sovereign’s bond position, y is output, d is the default decision, $V^d(y)$ is the value conditional on default, and $V^{nd}(b, y)$ is the value conditional on repaying. In this case, even if the subsidiary value functions V^d and V^{nd} are concave in b , V is not.³

Within the classes of models where monotonicity obtains but concavity does not, few computational options are available. Many of the fastest and most accurate methods, such as projection and [Carroll \(2006\)](#)’s endogenous gridpoints method (EGM), work with first order conditions (FOCs). Similarly, linearization and higher-order perturbations (which are less accurate but even faster) also work with FOCs. However, without concavity, FOCs are necessary but not sufficient, and so one cannot rely on FOC methods alone as there may be multiple local maxima. This also makes the use of any direct maximization routines over a continuous search space (perhaps the most common being the simplex method) dangerous. Because of these problems, grid search is often used for non-concave problems.⁴

For solving a non-concave problem via grid search, we are aware of only two existing methods. The first, which we refer to as “brute force,” evaluates π for every possible (i, i') combination. The second, which we refer to as “simple monotonicity,” uses monotonicity to restrict the search space for $g(i)$ to $\{g(i - 1), \dots, n'\}$ when $i > 1$. Our method dominates

²Of course, not every i' may be feasible, and whether a given i' is feasible depends on i (and y). In [Section 2.1](#) and [Appendix B](#), we show how the results generalize to a problem where the feasible choice set is not necessarily fixed. Note also that, since the choice set is not convex, the problem (1) may have more than one optimal policy. We discuss this issue more in [Section 2.1](#).

³Using the convention $-b$ is a sovereign’s debt, for a b^* such that the sovereign is indifferent between default and repayment, i.e. $V^d(y) = V^{nd}(b^*, y)$, one has $V(b^* - \varepsilon, y) = V(b^*, y) < V(b^* + \varepsilon, y)$ for any $\varepsilon > 0$. Hence, V is not concave at b^* .

⁴Even if one ultimately solves a continuous problem, a grid search is often done as a first step to avoid local minima.

these in terms of worst-case behavior where both brute force and simple monotonicity require nn' evaluations.⁵ While the worst case behavior for simple monotonicity could be misleading, we find practically that it is not: For both the RBC model and [Arellano \(2008\)](#) model, simple monotonicity uses roughly $.4nn'$ to $.5nn'$ evaluations for many different values of n and n' . In contrast, our method requires at most $3n' + 2n + \log_2(n)n'$ evaluations in the worst case. This makes binary monotonicity faster for small n and n' and exponentially faster as n and n' become large.

Our method, which we refer to as “binary monotonicity,” achieves this by using a divide and conquer algorithm. A brief description of it is as follows with a complete description given in Section 2.3. The algorithm initially computes $g(1)$ and $g(n)$, essentially by brute force, and defines $\underline{i} = 1$ and $\bar{i} = n$. The core of the algorithm assumes $g(\underline{i})$ and $g(\bar{i})$ are known and computes the optimum for all $i \in \{\underline{i}, \dots, \bar{i}\}$ recursively. Defining $m = \lfloor (\underline{i} + \bar{i})/2 \rfloor$, it solves for $g(m)$. Because of monotonicity, $g(m)$ —which in general might be any of $\{1, \dots, n'\}$ —must lie in $\{g(\underline{i}), \dots, g(\bar{i})\}$. With $g(m)$ computed, the algorithm then recursively proceeds to the core of the algorithm twice, once for $(\underline{i}, \bar{i})$ redefined to be $(\underline{i}, m)$ and once for $(\underline{i}, \bar{i})$ redefined to be $(m, \bar{i})$. The recursion stops when $\bar{i} \leq \underline{i} + 1$, since then g is known for all $i \in \{\underline{i}, \dots, \bar{i}\}$.

To assess our method for real-life cases, we consider two common economic environments. The first environment we consider is the sovereign default model of [Arellano \(2008\)](#). For this model, the sovereign’s savings policy is substantially non-linear. The second environment we consider is the real business cycle (RBC) model. For this model, it is well known that the capital policy is nearly linear in capital holdings. For both these models, we find simple monotonicity is roughly twice as fast as brute force. In contrast, we find that binary monotonicity is roughly six, 50, and 400 times faster than simple monotonicity for grid sizes of 100, 1000, and 10000 points, respectively.

For the RBC model, we can combine brute force, simple monotonicity, and binary monotonicity with techniques that exploit concavity.⁶ We are aware of two methods that exploit concavity to solve for $\max_{i' \in \{a, \dots, b\}} \pi(i, i')$ in possibly fewer than $b - a + 1$ evaluations. The first, which we refer to as “simple concavity,” iterates through the list $a, a + 1, \dots$ and stops whenever $\pi(i, i')$ falls. The second, which we refer to as “binary concavity,” is the method of [Heer and Maufner \(2005\)](#).⁷ This approach uses divide and conquer, evaluating $\pi(i, i')$ at $i' = m := \lfloor (a + b)/2 \rfloor$ and $m + 1$, and subsequently restricting the $\{a, \dots, b\}$ search space to either $\{a, \dots, m\}$ or $\{m + 1, \dots, b\}$.

⁵The worst case behavior for simple monotonicity obtains when the true policy is $g(\cdot) = 1$.

⁶Throughout the paper, we say the algorithms exploit concavity. However, the algorithms actually exploit something closer to quasi-concavity. See Appendix B, Section B.2, for details.

⁷Their algorithm in its original form can be found on p. 26 of [Heer and Maufner \(2005\)](#). Our implementation, given in Appendix A, differs slightly.

We prove that if the problem is concave, binary monotonicity can be combined with binary concavity to solve (1) in no more than $8n' + 10n$ evaluations of π in the worst case, a striking result. Surprisingly, however, for the RBC model with a linearly spaced capital grid we find that binary concavity with binary monotonicity is the second fastest approach. Simple concavity with simple monotonicity ends up faster because the optimal policy very nearly satisfies $g(i + 1) = g(i) + 1$ and, for a given i , only three π evaluations to compute the optimum in this case.⁸ We find binary monotonicity with binary concavity takes an average of 3.7 evaluations of π for each i , slightly more than the 3.0 for simple monotonicity with simple concavity. However, if the problem is concave, there are typically far superior methods available already such as projection or EGM, and so we do not think there is much added in this case.

To the best of our knowledge, our method is novel. However, the problem we consider is very common, not just in economics, but also in fields such as operations research. In a search of the literature, we failed to find anything close to a divide and conquer algorithm exploiting policy function monotonicity.⁹ While the general divide and conquer principle is of course well known, we have not seen it applied to monotone policy functions.

2 The Algorithm

This section gives a more general but equivalent formulation of (1). It then describes the two economic models we consider in Section 4 and how they map into the form given in (1). Lastly, it presents the algorithm in detail.

2.1 A More General, but Equivalent, Formulation

Often in economic models, not every choice is feasible and whether a choice is feasible depends on state variables. To handle this in a general way, we suppose the feasible choice set, $I'(i) \subset \{1, \dots, n'\}$, depends on the state i and may be empty, and for every i such that $I'(i)$ is nonempty we define

$$\tilde{\Pi}(i) = \max_{i' \in I'(i)} \tilde{\pi}(i, i'). \quad (2)$$

⁸The search space for computing $g(i + 1)$ is $\{g(i), \dots, n'\}$. If $g(i + 1) = g(i) + 1$, the method computes $\pi(i + 1, g(i))$, $\pi(i + 1, g(i) + 1)$, and $\pi(i + 1, g(i) + 2)$. It then stops since $\pi(i + 1, g(i) + 2) \leq \pi(i + 1, g(i) + 1)$.

⁹In particular, we searched Google Scholar for various combinations of the following phrases: Markov decision process, monotone policy, binary, divide and conquer, DSGE, grid search, and direct search.

In Appendix B, Section B.2, we prove that if $\tilde{\pi}$ is bounded below by some $\underline{\pi}$ and $I'(i)$ is monotonically increasing, then an equivalence between (1) and (2) exists. Namely, defining

$$\pi(i, i') = \begin{cases} \tilde{\pi}(i, i') & \text{if } I'(i) \neq \emptyset \text{ and } i' \in I'(i) \\ \underline{\pi} & \text{if } I'(i) \neq \emptyset \text{ and } i' \notin I'(i) \\ \mathbf{1}[i' = 1] & \text{if } I'(i) = \emptyset \end{cases},$$

the two problems have the same solutions whenever $I'(i)$ is nonempty. Moreover, we show every policy function associated with (1) is monotone when every policy function associated with (2) is monotone.¹⁰ Further, we show that if $I'(i) = \{1, \dots, n'(i)\}$ for some $n'(i)$ function and $\tilde{\pi}(i, i')$ is concave in i' (in the particular discrete sense defined in Appendix B) over $I'(i)$, then simple concavity and our formulation of binary concavity both deliver an optimal policy when applied to (1). While $\tilde{\pi}$ may not be bounded below theoretically, one can typically bound $\tilde{\pi}$ artificially without affecting the computational results in any material way (a point we revisit shortly in Section 2.2).

2.2 Examples

Economic problems of the type given in (1) and (2) abound. An example that we will use later in Section 4 is the maximization step associated with the real business cycle model when capital k must be chosen from a discrete (monotonically increasing) grid $\mathcal{K}$ and productivity z follows a Markov chain $\pi_{zz'}$:

$$\begin{aligned} V(k, z) = \max_{c \geq 0, l \in [0, 1], k' \in \mathcal{K}} & u(c, l) + \beta \sum_{z'} \pi_{zz'} V(k', z') \text{ s.t.} \\ & c + k' = zF(k, l) + (1 - \delta)k \end{aligned} \quad (3)$$

(where l is labor). Letting $\mathcal{K}_j$ denote the j th element of $\mathcal{K}$, this can be written as $V(k, z) = \Pi_z(i)$ where

$$\Pi_z(i) := \max_{i' \in \{1, \dots, n'_z(i)\}} \pi_z(i, i') \quad (4)$$

and π_z is given by

$$\begin{aligned} \pi_z(i, i') = \max_{c, l} & u(c, l) + \beta \sum_{z'} \pi_{zz'} V(k', z') \\ & c + k' = zF(k, l) + (1 - \delta)k \\ & c \geq 0, l \in [0, 1] \end{aligned} \quad (5)$$

¹⁰Note that in general there may be more than one optimal policy. This is true even if π is strictly concave in the sense that $\pi(i, i') = f(i')$ for some twice-differentiable, strictly concave function $f : \mathbb{R} \rightarrow \mathbb{R}$. For instance, if $f = -(x - 1.5)^2$, then $\arg \max \pi(i, i') = \{1, 2\}$. Our algorithm only finds an optimal policy, and it implicitly assumes that *every* optimal policy is monotone.

where $k = \mathcal{K}_i, k' = \mathcal{K}_{i'}$ and $n'_z(i) = \min\{i' | \mathcal{K}_{i'} \leq zF(k, 1) + (1 - \delta)k\}$.¹¹ The problem (4) fits the form given in (2). Moreover, because $n'_z(i)$ is increasing, the problem can be mapped into (1) as long as $u(0, 1)$ is defined.¹² Additionally, if $u(0, 1)$ is not defined, as is the case for log-log preferences, the results will not typically be affected by replacing $u(c, l)$ with $\tilde{u}(c, l) = u(c + \varepsilon, l - \varepsilon)$ for $\varepsilon > 0$ a small constant.¹³

Since we will also use the Arellano (2008) model as a test case, we describe a discretized version now and show how it also fits this form.¹⁴ In that model, a sovereign has output y that follows a Markov chain $\pi_{yy'}$, chooses discount bond holdings b from a finite set $\mathcal{B}$ with price $q(b, y)$, and has a default decision d . If the sovereign defaults, output falls to $y^{def} := h(y) \leq y$. The bond price satisfies $q(b', y) = (1 + r)^{-1} \sum_{y'} \pi_{yy'}(1 - d(b', y'))$. The sovereign's problem reduces to

$$V(b, y) = \max_{d \in \{0, 1\}} dV^d(y) + (1 - d)V^{nd}(b, y) \quad (6)$$

where the value of repaying is

$$\begin{aligned} V^{nd}(b, y) &= \max_{c \geq 0, b' \in \mathcal{B}} u(c) + \beta \sum_{y'} \pi_{yy'} V(b', y') \\ \text{s.t. } c + q(b', y)b' &= b + y \end{aligned} \quad (7)$$

and the value of defaulting is

$$V^d(y) = u(y^{def}) + \beta \sum_{y'} \pi_{yy'} (\theta V^d(y') + (1 - \theta)V^{nd}(0, y')). \quad (8)$$

The only computational difficulty is in solving (7). To map it into the form given in (2), create a separate problem for each y and solve

$$\Pi_y(i) = \max_{i' \in I'_y(i)} \pi_y(i, i') \quad (9)$$

¹¹Technically, to ensure there is a feasible choice for every z and every $k \in \mathcal{K}$, one needs $zF(\underline{k}, 1) \geq \delta \underline{k}$ for every z where $\underline{k} = \min \mathcal{K}$.

¹²In this case, one has π_z bounded below by $u(0, 1)/(1 - \beta)$ assuming $V(k', z') \geq u(0, 1)/(1 - \beta)$ for all k', z' , as the fixed point must satisfy.

¹³Because of the limits on machine precision, $c + \varepsilon = c$ whenever c is large relative to ε . Hence, the results will only be affected for very small values of c and similarly for l , which are not likely to occur for commonly used calibrations.

¹⁴The theoretical model of Arellano (2008) has continuous bond holdings and shocks, but it is solved using grid search with discretized versions of both.

where $\pi_y(i, i')$ is defined by

$$\pi_y(i, i') = u(b + y - q(b', y)b') + \beta \sum_{y'} \pi_{yy'} V(b', y') \quad (10)$$

and $I'_y(i) := \{i' | b + y - q(b', y)b' \geq 0\}$ with $b = \mathcal{B}_i$ and $b' = \mathcal{B}_{i'}$. Since $I'_y(i)$ is monotonically increasing, (9) can be mapped into the equivalent formulation (1) provided u is bounded below.¹⁵

2.3 The Algorithm

Our algorithm computes an optimal policy g and the optimal value Π using divide-and-conquer. The algorithm is as follows:

1. **Initialization:** Compute $g(1)$ and $\Pi(1)$ by searching over $\{1, \dots, n'\}$. If $n = 1$, STOP. Compute $g(n)$ and $\Pi(n)$ by searching over $\{g(1), \dots, n'\}$. Let $\underline{i} = 1$ and $\bar{i} = n$. If $n = 2$, STOP.
2. At this step, $(g(\underline{i}), \Pi(\underline{i}))$ and $(g(\bar{i}), \Pi(\bar{i}))$ are known. Find an optimal policy and value for all $i \in \{\underline{i}, \dots, \bar{i}\}$ as follows:
 - (a) If $\bar{i} = \underline{i} + 1$, STOP: For all $i \in \{\underline{i}, \dots, \bar{i}\} = \{\underline{i}, \bar{i}\}$, $g(i)$ and $\Pi(i)$ are known.
 - (b) Take the midpoint $m = \lfloor \frac{\underline{i} + \bar{i}}{2} \rfloor$ and compute $g(m)$ and $\Pi(m)$ by searching over $\{g(\underline{i}), \dots, g(\bar{i})\}$.
 - (c) Divide and conquer: Go to (2) twice, first computing the optimum for $i \in \{\underline{i}, \dots, m\}$ and second computing the optimum for $i \in \{m, \dots, \bar{i}\}$. In the first case, redefine $\bar{i} := m$; in the second case, redefine $\underline{i} := m$.

While the algorithm is most easily stated and analyzed in recursive form, using recursion can sometimes carry a computational cost. To avoid this cost, Appendix A provides a non-recursive implementation of this algorithm.

Figure 1 provides a graphical illustration of binary monotonicity and, for comparison, simple monotonicity. The blue dots represent the optimal policy ($\lfloor n' \sqrt{i/n'} \rfloor$ in this case) and the black empty circles represent, for a given i , the search space when solving for $g(i)$, i.e., the range of i' that might possibly be optimal. With simple monotonicity, the search for $i > 1$ is restricted to $\{g(i-1), \dots, n'\}$. For binary monotonicity, the search is restricted to $\{g(\underline{i}), \dots, g(\bar{i})\}$ where $\underline{i}$ and $\bar{i}$ are far apart for the first iterations but become close together rapidly (with the distance roughly halved at each iteration). For this example, the average

¹⁵Again, if u is not bounded below, one can bound it artificially. Also, in the Arellano (2008) model, if there is no feasible choice conditional on repayment (i.e., $I'_y(i) = 0$), it is assumed the sovereign must default.

search space for simple monotonicity, i.e. the average size of $\{g(i-1), \dots, n'\}$, is 8.7 when $n = n' = 20$ and grows to 35.3 when $n = n' = 100$. Since not exploiting monotonicity would have the search space be 20 and 100, respectively, this represents roughly a 60% improvement on brute force. In contrast, the average size of binary monotonicity's search space is 6.2 when $n = n' = 20$ (a 29% improvement on simple monotonicity) and 8.9 when $n = n' = 100$ (a 75% improvement).

Figure 1: A Graphical Illustration of Simple and Binary Monotonicity

The remaining sections show that this example is typical. Specifically, in Section 3 we show that binary monotonicity always—i.e., for any monotone g —exhibits a very slow increase in search space as n increases. In contrast, simple monotonicity may not exhibit any reduction in search space (which is what happens when $g(\cdot) = 1$). Then, in Section 4, we show that, for both the RBC and [Arellano \(2008\)](#) models, simple monotonicity is about 60% faster than brute force but binary monotonicity is faster than simple monotonicity for small grids and becomes increasingly faster as the grid size increases.

3 Theoretical Analysis

In the preceding algorithm, we assume there is some method for solving

$$\max_{i' \in \{a, \dots, a+\gamma-1\}} \pi(i, i') \quad (11)$$

for arbitrary $a \geq 1$ and $\gamma \geq 1$ (satisfying $a + \gamma - 1 \leq n'$). One possibility is brute force (checking every possible value), which requires γ evaluations of $\pi(i, \cdot)$. However, if $\pi(i, i')$ is known to be concave in i' , then Heer and Maußner (2005)'s binary concavity can find the solution in no more than $2\lceil \log_2(\gamma) \rceil - 1$ for $\gamma \geq 3$ and γ for $\gamma \leq 2$.¹⁶ We allow for many such methods by characterizing the algorithm's properties conditional on a $\sigma : \mathbb{Z}^{++} \rightarrow \mathbb{Z}^+$, restricting attention to monotonically increasing σ .

Because of the recursive nature of the algorithm, the π evaluation bound for general σ is also naturally recursive. The following definition and Proposition 1 provide such a bound.

Definition. For any $\sigma : \mathbb{Z}^{++} \rightarrow \mathbb{Z}^+$, define $M_\sigma : \{2, 3, \dots\} \times \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$ recursively by $M_\sigma(z, \gamma) = 0$ if $z = 2$ or $\gamma = 0$ and

$$M_\sigma(z, \gamma) = \sigma(\gamma) + \max_{\gamma' \in \{1, \dots, \gamma\}} \left\{ M_\sigma(\lfloor \frac{z}{2} \rfloor + 1, \gamma') + M_\sigma(\lfloor \frac{z}{2} \rfloor + 1, \gamma - \gamma' + 1) \right\}$$

for $z > 2$ and $\gamma > 0$.

Proposition 1. Let $\sigma : \mathbb{Z}^{++} \rightarrow \mathbb{Z}^+$ be an upper bound on the number of π evaluations required to solve (11). Then, the algorithm requires at most $2\sigma(n') + M_\sigma(n, n')$ evaluations where n and n' are from the problem stated in (1) for $n \geq 2$ and $n' \geq 1$.

While not as tight as possible, Proposition 1 gives a fairly tight bound. However, it is also unwieldy because of the discrete nature of the problem. By bounding σ whose domain is $\mathbb{Z}^{++}$ with a $\bar{\sigma}$ whose domain is $[1, \infty)$, a more convenient bound can be found. This bound is given in Lemma 1.

Lemma 1. Suppose $\bar{\sigma} : [1, \infty) \rightarrow \mathbb{R}^+$ is either the identity map ($\bar{\sigma}(\gamma) = \gamma$) or is a strictly increasing, strictly concave, and differentiable function. If $\bar{\sigma}(\gamma) \geq \sigma(\gamma)$ for all $\gamma \in \mathbb{Z}^{++}$, then an upper bound on function evaluations is

$$3\bar{\sigma}(n') + \sum_{j=1}^{I-2} 2^j \bar{\sigma}(2^{-j}(n' - 1) + 1)$$

if $I > 2$ where $I = \lceil \log_2(n - 1) \rceil + 1$. An upper bound for $I \leq 2$ is $3\bar{\sigma}(n')$.

¹⁶We prove this in Lemma 2 in Appendix B.

The main theoretical result of the paper is given in Proposition 2. It applies Lemma 1 with $\bar{\sigma}$ bounds corresponding to brute force concavity and binary concavity.

Proposition 2. *Suppose $n \geq 4$ and $n' \geq 3$. If brute force grid search is used, then no more than $\log_2(n-1)(n'-1) + 3n' + 2n - 4$ evaluations are required. Consequently, fixing $n = n'$ the algorithm’s worst case behavior is $O(n \log n)$ with a hidden constant of one.*

If binary concavity is used, then no more than $10n + 8n' - 4 \log_2(n' - 1) - 32$ evaluations are required. Consequently, fixing $n = n'$ the algorithm’s worst case behavior is $O(n)$ with a hidden constant of 18.

These worst-case bounds show binary monotonicity is very powerful, even without an assumption about concavity. The remainder of the paper assesses binary monotonicity’s performance in two economic models commonly employed in practice. As will be seen, binary monotonicity also does very well in the context of these models.

4 Comparison with Existing Grid Search Techniques

This section compares our method with existing grid search techniques, or what we now call “speedups.” Table 1 lists all speedups known to us, along with brief descriptions of each. We break the analysis into two parts. First, we compare our method with existing speedups that do not assume concavity. For this, we use the Arellano (2008) model. Second, we compare our method with existing speedups that do assume concavity. For this, we use the RBC model since the Arellano (2008) model has non-concave value functions. Since the accuracy of all the grid search methods is identical (because they all produce exactly the same policy), we do not report any accuracy measures. Calibrations are presented in Appendix C.

4.1 Not Assuming Concavity

Figure 2 reports the time required to obtain convergence in Arellano (2008) model for brute force, simple monotonicity, and binary monotonicity (the only speedups that apply to this non-concave problem). The run times for brute force and simple monotonicity both grow exponentially. In contrast, binary monotonicity grows almost linearly. For a grid size of 100, all the methods solve the model much less than a minute. This changes quickly with n . For a 1000 point grid, brute force takes about 8 minutes, simple monotonicity takes about half that, but binary monotonicity takes far less than a minute. In general, it seems that simple monotonicity is about 50% faster than brute force.

Table 2, which gives run times and π evaluation counts for a wide range of n values, confirms and extends these findings. Simple monotonicity requires roughly 60% fewer eval-

Grid Search Speedup	Description
Simple Monotonicity	Exploits policy function monotonicity by only searching $i' \in \{g(i-1), \dots, n'\}$ when $i > 1$.
Binary Monotonicity	The approach in this paper. Uses divide-and-conquer to exploit policy function monotonicity. Restricts search range to $i' \in \{g(\underline{i}), \dots, g(\bar{i})\}$.
Simple Concavity	Exploits concavity of $\pi(i, i')$ in i' . To find the maximum in $i' \in \{\underline{i}, \dots, \bar{i}\}$, iterates through $i' = \underline{i}, \underline{i}+1, \dots$ and stops once $\pi(i, i')$ decreases or $\bar{i}$ is reached.
Binary Concavity	The approach in Heer and Maußner (2005) . Uses divide-and-conquer to exploit concavity of $\pi(i, i')$ in i' . In finding the maximum of $i' \in \{\underline{i}, \dots, \bar{i}\}$, checks $i' = \lfloor \frac{\underline{i} + \bar{i}}{2} \rfloor$ and $i' = \lfloor \frac{\underline{i} + \bar{i}}{2} \rfloor + 1$, then restricts search to only $\{\underline{i}, \lfloor \frac{\underline{i} + \bar{i}}{2} \rfloor\}$ or $\{\lfloor \frac{\underline{i} + \bar{i}}{2} \rfloor + 1, \bar{i}\}$.
X Monotonicity and Y Concavity	Uses X monotonicity to restrict the search space for i' to some interval $\{\underline{i}, \dots, \bar{i}\}$ and then Y concavity to find the maximum within that range.

Table 1: Description of Grid Search Speedups

Figure 2: Convergence Time For Methods Not Assuming Concavity

uations than brute force and so exhibits the same exponential growth in run times. The running times and evaluation counts for binary monotonicity are less initially and grow much more slowly in n . As a consequence, the ratio of the simple monotonicity running time to the binary time begins at around 6 for a 100 point grid, grows to 50 for a 1000 point grid, 400 for a 1000 point grid, and continues to grow close to linearly in n . Hence, even for small grid sizes, one can expect a large speedup relative to simple monotonicity. Binary monotonicity is so fast that it takes less than a minute for a 10,000 point grid and less than ten minutes for a 100,000 point grid. In contrast, simple monotonicity’s run time for a 10,000 point grid is 5 hours and (estimated) run time for a 100,000 point grid is roughly three weeks.

Grid size (n)	Run Time (Minutes)			Evaluations/ n			Simple to Binary Time
	None	Simple	Binary	None	Simple	Binary	
100	0.08	0.03	0.01	100	42	5.5	6.1
250	0.48	0.20	0.01	250	102	6.1	14.4
500	1.90	0.78	0.03	500	204	6.5	26.7
1000	7.71	3.16	0.06	1000	406	7.0	51.4
2500	47.7	19.3	0.17	2500	1013	7.6	116
5000	192	78.6	0.36	5000	2025	8.0	220
10000	769*	309	0.75	10000*	4050	8.5	409
25000	4814*	1914*	2.21	25000*	10122*	9.1	867
50000	19264*	7634*	4.60	50000*	20243*	9.5	1659
100000	77075*	30491*	8.91	100000*	40484*	10.0	3423

Note: * means the value is estimated via a fitted quadratic polynomial.

Table 2: Run Times and Evaluations for Methods Not Assume Concavity

4.2 Assuming Concavity

The previous section showed that binary monotonicity outperforms existing grid search methods that do not assume concavity. However, binary monotonicity can be combined with various concavity speedups to deliver potentially much faster performance. Table 3 examines the running times for all nine combinations of the monotonicity and concavity speedups. Since the [Arellano \(2008\)](#) model has non-concave value functions, the RBC model is used.

Surprisingly, the fastest combination is simple monotonicity with simple concavity. This pair has the smallest run times for both values of n and the time increases linearly. Somewhat amazingly, to solve for the optimal policy for a given state requires, on average, only three evaluations of π . The second fastest, binary monotonicity with binary concavity, exhibits a similar linear time increase but fares slightly worse in absolute terms. In particular, it

Speedup		$n = 250$		$n = 500$		Time
Monotonicity	Concavity	Eval./ n	Min.	Eval./ n	Min.	Increase
None	None	250.0	1.15	500.0	4.59	4.0
Simple	None	127.0	0.40	252.5	1.58	4.0
Binary	None	10.7	0.06	11.7	0.14	2.2
None	Simple	126.0	0.77	250.5	3.04	4.0
Simple	Simple	3.0	0.02	3.0	0.04	2.0
Binary	Simple	6.8	0.05	7.3	0.10	2.1
None	Binary	13.9	0.09	15.9	0.20	2.3
Simple	Binary	12.6	0.07	14.6	0.16	2.3
Binary	Binary	3.7	0.03	3.7	0.05	2.0

Table 3: Run Times and Evaluations for All Monotonicity and Concavity Speedups

requires on average 3.7 evaluations of π per state, a very small number, but not quite as small as the 3.0 required by simple concavity and simple monotonicity. All the other combinations are slower and exhibit superlinear time increases.

Simple monotonicity paired with simple concavity is very effective because the policy function is very nearly linear when the grid is uniform (as it is here). In particular, the capital policy very nearly satisfies either $g(i) = g(i-1)$ or $g(i) = g(i-1) + 1$. In the first case, simple monotonicity and simple concavity requires only two π evaluations to compute $g(i)$ when $g(i-1)$ is known: The first evaluates $\pi(i, g(i-1))$ and the second evaluates $\pi(i, g(i-1) + 1)$; finding $\pi(i, g(i-1)) \geq \pi(i, g(i-1) + 1)$, the algorithm terminates. In the second case, a third evaluation, $\pi(i, g(i-1) + 2)$ is needed, but finding $\pi(i, g(i-1) + 1) \geq \pi(i, g(i-1) + 2)$, the algorithm stops.

Whereas simple monotonicity with simple concavity searches locally about $g(i-1)$ in computing $g(i)$, the second fastest combination of binary monotonicity with binary concavity performs something more akin to a global search. It finds $g(i)$ not using information about $g(i-1)$, but rather about $g(a)$ and $g(b)$ where $i = \lfloor (a+b)/2 \rfloor$. The search space is then restricted $\{g(a), \dots, g(b)\}$. If $g(a+1) = g(a) + 1$ for all a , then the true solution is, loosely speaking, $g(i) = g(a) + (a+b)/2$. However, binary concavity search does not use this information, but rather begins divide-and-conquer of the entire search domain $\{g(a), \dots, g(b)\}$.¹⁷

¹⁷This suggests a heuristic of searching a small interval about $g(a) + (a+b)/2$ first before expanding the search, if necessary, to all of $\{g(a), \dots, g(b)\}$. However, simple monotonicity with simple concavity would seemingly be preferable to this.

Figure 3: Empirical $O(n)$ Behavior

5 Conclusion

We have shown binary monotonicity is a powerful grid search technique. Without any assumptions of concavity, the algorithm is $O(n \log_2 n)$ for any monotone policy. Discrete choice models where monotonicity obtains but concavity does not should substantially benefit from our approach. Our approach may enable Bayesian estimation of this class of models, employing the particle filter of [Fernández-Villaverde and Rubio-Ramírez \(2007\)](#). When paired with binary concavity, binary monotonicity is $O(n)$ enabling grid search to be used for very large problems.

References

- S. R. Aiyagari. Uninsured idiosyncratic risk and aggregate savings. *Quarterly Journal of Economics*, 109(3):659–684, 1994.
- C. Arellano. Default risk and income fluctuations in emerging economies. *American Economic Review*, 98(3):690–712, 2008.
- C. D. Carroll. The method of endogenous gridpoints for solving dynamic stochastic optimization problems. *Economic Letters*, 91(3):312–320, 2006.

- S. Chatterjee, D. Corbae, M. Nakajima, and J.-V. Ríos-Rull. A quantitative theory of unsecured consumer credit with risk of default. *Econometrica*, 75(6):1525–1589, 2007.
- J. Fernández-Villaverde and J. F. Rubio-Ramírez. Estimating macroeconomic models: A likelihood approach. *The Review of Economic Studies*, 74(4):1059–1087, 2007.
- B. Heer and A. Maußner. *Dynamic General Equilibrium Modeling: Computational Methods and Applications*. Springer, Berlin, Germany, 2005.
- P. Krusell and A. A. Smith, Jr. Income and wealth heterogeneity in the macroeconomy. *Journal of Political Economy*, 106(5):867–896, 1998.
- G. Tauchen. Finite state Markov-chain approximations to univariate and vector autoregressions. *Economics Letters*, 20(2):177–181, 1986.

A Algorithms

This appendix contains a non-recursive version of binary monotonicity, as well as our implementation of binary concavity.

A.1 A Non-Recursive Formulation of Binary Monotonicity

Rather than directly implementing binary monotonicity using recursion, it is computationally more efficient to eliminate the recursion. The following algorithm does this:

1. **Initialization:** Compute $g(1)$ maximizing over $\{1, \dots, n'\}$ and compute $g(n)$ maximizing over $\{g(1), \dots, n'\}$. Allocate an array of size $(\lceil \log_2(n-1) \rceil + 1) \times 2$ that will hold the list below. Fill the first row of the array with (l_1, u_1) where $l_1 := 1$, $u_1 := n$. Define $k := 1$.
2. Expand the list from k rows to $\tilde{k}$ rows as follows:

$$\begin{bmatrix} l_1 & u_1 \\ \vdots & \vdots \\ l_k & u_k \\ l_{k+1} & u_{k+1} \\ l_{k+2} & u_{k+2} \\ \vdots & \vdots \\ l_{\tilde{k}} & u_{\tilde{k}} \end{bmatrix} := \begin{bmatrix} l_1 & u_1 \\ \vdots & \vdots \\ l_k & u_k \\ l_k & \lfloor \frac{l_k + u_k}{2} \rfloor \\ l_k & \lfloor \frac{l_k + u_{k+1}}{2} \rfloor \\ \vdots & \vdots \\ l_k & \lfloor \frac{l_k + u_{\tilde{k}-1}}{2} \rfloor \end{bmatrix}$$

stopping when $u_{\tilde{k}} \leq l_{\tilde{k}} + 1$ (corresponding to step 2(a) in the algorithm). At this step, $k \geq 1$ and $g(l_j)$ and $g(u_j)$ are known for all $j \leq k$. For $j = k+1, \dots, \tilde{k}$, compute $g(u_j)$ by maximizing over the interval $g(l_{j-1}), \dots, g(u_{j-1})$. Taking each row as specifying an interval and subdividing it into two intervals, the following row gives the leftmost subinterval.

At this point, g is known for every l and u in the list. Moreover, the interval corresponding to $\tilde{k}$ has exactly two elements, $l_{\tilde{k}}$ and $u_{\tilde{k}}$, which implies the policy $g(l_{\tilde{k}}), \dots, g(u_{\tilde{k}})$ is known.

Define $k := \tilde{k}$. Go to step 3.

3. If $k = 1$, STOP. If $u_k = u_{k-1}$, then eliminate the last row of the list, set $k := k - 1$, and repeat this step. Otherwise, go to the next step.
4. If here, then $u_k < u_{k-1}$. Set $(l_k, u_k) := (u_k, u_{k-1})$. This step corresponds to moving to the right subinterval of the interval corresponding to $k - 1$. Go to step 2.

A.2 Binary Concavity

Below is our implementation of [Heer and Maußner \(2005\)](#)'s algorithm for solving

$$\max_{i' \in \{a, \dots, b\}} \pi(i, i').$$

Throughout, n refers to $b - a + 1$.

1. **Initialization:** If $n = 1$, compute the maximum, $\pi(i, a)$, and STOP. Otherwise, set the flags $\mathbf{1}_a = 0$ and $\mathbf{1}_b = 0$. These flags indicate whether the value of $\pi(i, a)$ and $\pi(i, b)$ are known, respectively.
2. If $n > 2$, go to 3. Otherwise, $n = 2$. Compute $\pi(i, a)$ if $\mathbf{1}_a = 0$ and compute $\pi(i, b)$ if $\mathbf{1}_b = 0$. The optimum is the best of a, b .
3. If $n > 3$, go to 4. Otherwise, $n = 3$. If $\max\{\mathbf{1}_a, \mathbf{1}_b\} = 0$, compute $\pi(i, a)$ and set $\mathbf{1}_a = 1$. Define $m = \frac{a+b}{2}$, and compute $\pi(i, m)$.
 - (a) If $\mathbf{1}_a = 1$, check whether $\pi(i, a) > \pi(i, m)$. If so, the maximum is a . Otherwise, the maximum is either m or b ; redefine $a = m$, set $\mathbf{1}_a = 1$, and go to 2.
 - (b) If $\mathbf{1}_b = 1$, check whether $\pi(i, b) > \pi(i, m)$. If so, the maximum is b . Otherwise, the maximum is either a or m ; redefine $b = m$, set $\mathbf{1}_b = 1$, and go to 2.

4. Here, $n \geq 4$. Define $m = \lfloor \frac{a+b}{2} \rfloor$ and compute $\pi(i, m)$ and $\pi(i, m+1)$. If $\pi(i, m) < \pi(i, m+1)$, a maximum is in $\{m+1, \dots, b\}$; redefine $a = m+1$, set $\mathbf{1}_a = 1$, and go to 2. Otherwise, a maximum is in $\{a, \dots, m\}$; redefine $a = m$, set $\mathbf{1}_b = 1$, and go to 2.¹⁸

B Omitted Proofs and Lemmas

This section contains omitted proofs and lemmas. They are broken into two sections. Section B.1 examines the properties of the binary monotonicity algorithm and to a lesser extent the binary concavity algorithm. Section B.2 state in what sense (1) and (2) are equivalent and proves their equivalence. As stated in the main text, σ is always assumed to be monotonically increasing.

B.1 Algorithm properties

Lemma 2. *Consider the problem $\max_{i' \in \{a, \dots, a+n-1\}} \pi(i, i')$ for any a and any i . For any $n \in \mathbb{Z}^{++}$, binary concavity requires no more $2\lceil \log_2(n) \rceil - 1$ evaluations if $n \geq 3$ and no more than n evaluations if $n \leq 2$.*

Proof. We will show $\sigma(n) = 2\lceil \log_2(n) \rceil - 1$ for $n \geq 3$ and $\sigma(n) = n$ for $n \leq 2$ is an upper bound on the number of evaluations the binary concavity requires. For $n = 1$, the algorithm computes $\pi(i, a)$ and stops, so one evaluation is required. This agrees with $\sigma(1) = 1$. For $n = 2$, two evaluations are required ($\pi(i, a)$ and $\pi(i, a+1)$). This agrees with $\sigma(2) = 2$. For $n = 3$, step 3 requires $\pi(i, m)$ to be computed and may require $\pi(i, a)$ to be computed. Then step 3(a) or step 3(b) either stop with no additional function evaluations or go to step 2 with $\max\{\mathbf{1}_a, \mathbf{1}_b\} = 1$ where, in that case, at most one additional function evaluation is required. Consequently, $n = 3$ requires at most three function evaluations, which agrees with $\sigma(3) = 2\lceil \log_2(3) \rceil - 1 = 3$. So, the statement of lemma holds for $1 \leq n \leq 3$.

Now consider each $n \in \{4, 5, 6, 7\}$ for any $\mathbf{1}_a, \mathbf{1}_b$ flags. Since $n \geq 4$ the algorithm is in (or goes to) step 4. Consequently, two evaluations are required. Since the new interval is either $\{a, \dots, m\}$ or $\{m+1, \dots, b\}$ and $\pi(i, m)$ and $\pi(i, m+1)$ are computed in step 4, the *next* step has $\max\{\mathbf{1}_a, \mathbf{1}_b\} = 1$. Now, if $n = 4$, the next step must be step 2, which requires at most one additional evaluation (since $\max\{\mathbf{1}_a, \mathbf{1}_b\} = 1$). Hence, the total evaluations are less than or equal to 3 (two for step 4 and one for step 2). If $n = 5$, then the next step is either step 2, requiring one evaluation, or step 3, requiring two evaluations. So, the total evaluations are not more than four. If $n = 6$, the next step is step 3, and so four evaluations are required. Lastly, for $n = 7$, the next step is either step 3, requiring two evaluations, or step 4 (with $n = 4$), requiring at most three evaluations. So, the evaluations are weakly less

¹⁸Note that in the case of indifference, $\pi(i, m) = \pi(i, m+1)$, the algorithm proceeds to $\{a, \dots, m\}$.

than $5 = 2 + \max\{2, 3\}$. Hence, for every $n = 4, 5, 6$, and 7 , the required evaluations are less than $3, 4, 4$, and 5 , respectively. One can then verify that the evaluations are less than $2\lceil\log_2(n)\rceil - 1$ for these values of n .

Now, suppose $n \geq 4$. We shall prove that the required number of evaluations is less than $2\lceil\log_2(n)\rceil - 1$ (i.e., is less than $\sigma(n)$). The proof is by induction. We have already verified the hypothesis holds for $n \in \{4, 5, 6, 7\}$, so consider some $n \geq 8$ and suppose the hypothesis holds for all $m \in \{4, \dots, n-1\}$. Let i be such that $n \in [2^i + 1, 2^{i+1}]$. Then note that two things are true, $\lceil\log_2(n)\rceil = i+1$ and $\lceil\log_2(\lfloor \frac{n+1}{2} \rfloor)\rceil = i$.¹⁹ Since $n \geq 4$, the algorithm is in (or proceeds to) step 4, which requires two evaluations, and then proceeds with a new interval to step 4 (again). If n is even, the new interval has size $n/2$. If n is odd, the new interval either has a size of $(n+1)/2$ or $(n-1)/2$. So, if n is even, no more than $2 + \sigma(n/2)$ evaluations are required; if n is odd, no more than $2 + \max\{\sigma((n+1)/2), \sigma((n-1)/2)\} = 2 + \sigma((n+1)/2)$ evaluations are required. The even and odd case can then be handled simultaneously with the bound $2 + \sigma(\lfloor \frac{n+1}{2} \rfloor)$. Manipulating this expression,

$$\begin{aligned} 2 + \sigma(\lfloor \frac{n+1}{2} \rfloor) &= 2 + 2\lceil\log_2 \lfloor \frac{n+1}{2} \rfloor\rceil - 1 \\ &= 2 + 2i - 1 \\ &= 2(i+1) - 1 \\ &= 2\lceil\log_2(n)\rceil - 1. \end{aligned}$$

Hence, the proof by induction is complete. □

Lemma 3. *For any σ , $M_\sigma(z, \gamma)$ is weakly increasing in z and γ .*

Proof. Fix a σ and suppress dependence on it. First, we will show $M(z, \gamma)$ is weakly increasing in γ for every z . The proof is by induction. For $z = 2$, $M(2, \cdot) = 0$. For $z = 3$, $M(3, \gamma) = \sigma(\gamma)$ which is weakly increasing in γ . Now consider some $z > 3$ and suppose $M(y, \cdot)$ is weakly increasing for all $y \leq z-1$. For $\gamma_2 > \gamma_1$,

$$\begin{aligned} M(z, \gamma_1) &= \sigma(\gamma_1) + \max_{\gamma' \in \{1, \dots, \gamma_1\}} \left\{ M(\lfloor \frac{z}{2} \rfloor + 1, \gamma') + M(\lfloor \frac{z}{2} \rfloor + 1, \gamma_1 - \gamma' + 1) \right\} \\ &\leq \sigma(\gamma_2) + \max_{\gamma' \in \{1, \dots, \gamma_2\}} \left\{ M(\lfloor \frac{z}{2} \rfloor + 1, \gamma') + M(\lfloor \frac{z}{2} \rfloor + 1, \gamma_1 - \gamma' + 1) \right\} \\ &\leq \sigma(\gamma_2) + \max_{\gamma' \in \{1, \dots, \gamma_2\}} \left\{ M(\lfloor \frac{z}{2} \rfloor + 1, \gamma') + M(\lfloor \frac{z}{2} \rfloor + 1, \gamma_2 - \gamma' + 1) \right\} \\ &= M(z, \gamma_2) \end{aligned}$$

¹⁹Note that both $\lceil\log_2(\cdot)\rceil$ and $\lceil\log_2(\lfloor \cdot \rfloor)\rceil$ are weakly increasing functions. So $n \in [2^i + 1, 2^{i+1}]$ implies $\lceil\log_2(n)\rceil \in [\lceil\log_2(2^i + 1)\rceil, \lceil\log_2(2^{i+1})\rceil] = [i+1, i+1]$. Likewise, $n \in [2^i + 1, 2^{i+1}]$ implies $\lceil\log_2(\lfloor \frac{n+1}{2} \rfloor)\rceil \in [\lceil\log_2(\lfloor \frac{2^i+1+1}{2} \rfloor)\rceil, \lceil\log_2(\lfloor \frac{2^{i+1}+1}{2} \rfloor)\rceil] = [i+1, i+1]$.

where the second inequality is justified by the induction hypothesis giving $M(\lfloor \frac{z}{2} \rfloor + 1, \cdot)$ as an increasing function (note $\lfloor \frac{z}{2} \rfloor + 1 \leq z - 1$ for all $z > 3$).

Now we will show $M(z, \gamma)$ is increasing in z for every γ . The proof is by induction. First, note that $M(2, \gamma) = 0 \leq \sigma(\gamma) = M(3, \gamma)$ for all $\gamma > 0$ and $M(2, \gamma) = 0 = M(3, \gamma)$ for $\gamma = 0$. Now, consider some $k > 3$ and suppose that for any $z_1, z_2 \leq k - 1$ with $z_1 \leq z_2$ that $M(z_1, \gamma) \leq M(z_2, \gamma)$ for all γ . The goal is to show that for any $z_1, z_2 \leq k$ with $z_1 \leq z_2$ that $M(z_1, \gamma) \leq M(z_2, \gamma)$ for all γ . So, consider such $z_1, z_2 \leq k$ with $z_1 \leq z_2$. If $\gamma = 0$, then $M(z_1, \gamma) = 0 = M(z_2, \gamma)$, so take $\gamma > 0$. Then

$$\begin{aligned} M(z_1, \gamma) &= \sigma(\gamma) + \max_{\gamma' \in \{1, \dots, \gamma\}} \left\{ M(\lfloor \frac{z_1}{2} \rfloor + 1, \gamma') + M(\lfloor \frac{z_1}{2} \rfloor + 1, \gamma - \gamma' + 1) \right\} \\ &\leq \sigma(\gamma) + \max_{\gamma' \in \{1, \dots, \gamma\}} \left\{ M(\lfloor \frac{z_2}{2} \rfloor + 1, \gamma') + M(\lfloor \frac{z_2}{2} \rfloor + 1, \gamma - \gamma' + 1) \right\} \\ &= M(z_2, \gamma). \end{aligned}$$

The inequality obtains since $\lfloor \frac{z_i}{2} \rfloor + 1 \leq k - 1$ for all i (which is true since even if $z_i = k$, one has $\lfloor k/2 \rfloor + 1 \leq k - 1$ by virtue of $k > 3$) and so the induction hypothesis gives $M(\lfloor \frac{z_1}{2} \rfloor + 1, \cdot) \leq M(\lfloor \frac{z_2}{2} \rfloor + 1, \cdot)$. Hence, the proof by induction is complete. $\square$

Proof of Proposition 1.

Proof. Since g is the policy function associated with (1), $g : \{1, \dots, n\} \rightarrow \{1, \dots, n'\}$. By monotonicity, g is weakly increasing. Define $N : \{1, \dots, n\}^2 \rightarrow \mathbb{Z}^+$ by

$$N(a, b) = M(b - a + 1, g(b) - g(a) + 1)$$

noting that this is well-defined (based on the definition of M) whenever $b > a$. Additionally, define a sequence of sets $\mathcal{I}_k$ for $k = 1, \dots, n - 1$ by

$$\mathcal{I}_k := \{(\underline{i}, \bar{i}) \mid \bar{i} = \underline{i} + k \text{ and } \underline{i}, \bar{i} \in \{1, \dots, n\}\}.$$

Note that for any $k \in \{1, \dots, n - 1\}$, $\mathcal{I}_k$ is nonempty and $N(a, b)$ is well-defined for any $(a, b) \in \mathcal{I}_k$.

We shall now prove that for any $k \in \{1, \dots, n - 1\}$, $(\underline{i}, \bar{i}) \in \mathcal{I}_k$ implies $N(\underline{i}, \bar{i})$ is an upper bound on the number of evaluations of π required by the algorithm in order to compute the optimal policy for all $i \in \{\underline{i}, \dots, \bar{i}\}$ when $g(\underline{i})$ and $g(\bar{i})$ are known. If true, then beginning at step 2 in the algorithm (which assumes $g(\underline{i})$ and $g(\bar{i})$ are known) with $(\underline{i}, \bar{i}) \in \mathcal{I}_k$, $N(\underline{i}, \bar{i})$ is an upper bound on the number of π evaluations.

The argument is by induction. First, consider $k = 1$. For any $(a, b) \in \mathcal{I}_1$, the algorithm

terminates at step 2(a). Consequently, the number of required π evaluations is zero, which is the same as $N(a, b) = M(b - a + 1, g(b) - g(a) + 1) = M(2, g(b) - g(a) + 1) = 0$ (recall $M(2, \cdot) = 0$).

Now, consider some $k \in \{2, \dots, n-1\}$ and suppose the induction hypothesis holds for all j in $1, \dots, k-1$. That is, assume for all j in $1, \dots, k-1$ that $(\underline{i}, \bar{i}) \in \mathcal{I}_j$ implies $N(\underline{i}, \bar{i})$ is an upper bound on the number of required π evaluations when $g(\underline{i})$ and $g(\bar{i})$ are known. We shall show it holds for k as well.

Consider any $(\underline{i}, \bar{i}) \in \mathcal{I}_k$ with $g(\underline{i})$ and $g(\bar{i})$ are known. Since $\bar{i} > \underline{i} + 1$, the algorithm does not terminate at step 2(a). In step 2(b), to compute $g(m)$ (where $m := \lfloor \frac{\underline{i} + \bar{i}}{2} \rfloor$), one must find the maximum within the range $g(\underline{i}), \dots, g(\bar{i})$, which requires at most $\sigma(g(\bar{i}) - g(\underline{i}) + 1)$ evaluations of π . In step 2(c), the space is then divided into $\{\underline{i}, \dots, m\}$ and $\{m, \dots, \bar{i}\}$.

If k is even, then m equals $\frac{\underline{i} + \bar{i}}{2}$. Since $(\underline{i}, m) \in \mathcal{I}_{k/2}$ and $g(\underline{i})$ and $g(m)$ are known, the induction hypothesis gives $N(\underline{i}, m)$ as an upper bound on the number of π evaluations needed to compute $g(\underline{i}), \dots, g(m)$. Similarly, since $(m, \bar{i}) \in \mathcal{I}_{k/2}$ and $g(m)$ and $g(\bar{i})$ are known, $N(m, \bar{i})$ provides an upper bound on the number of π evaluations needed to compute $g(m), \dots, g(\bar{i})$. Therefore, to compute $g(\underline{i}), \dots, g(\bar{i})$, at most $\sigma(g(\bar{i}) - g(\underline{i}) + 1) + N(\underline{i}, m) + N(m, \bar{i})$ evaluations are required. Defining $\gamma = g(\bar{i}) - g(\underline{i}) + 1$ and $\gamma' = g(m) - g(\underline{i}) + 1$ and using the definition of m and N , we have that the number of required evaluations is less than

$$\begin{aligned} & \sigma(\gamma) + M(m - \underline{i} + 1, g(m) - g(\underline{i}) + 1) + M(\bar{i} - m + 1, g(\bar{i}) - g(m) + 1) \\ &= \sigma(\gamma) + M\left(\frac{\bar{i} + \underline{i}}{2} - \underline{i} + 1, \gamma'\right) + M\left(\bar{i} - \frac{\bar{i} + \underline{i}}{2} + 1, g(\bar{i}) - g(m) + 1\right) \\ &= \sigma(\gamma) + M\left(\frac{\bar{i} - \underline{i}}{2} + 1, \gamma'\right) + M\left(\frac{\bar{i} - \underline{i}}{2} + 1, g(\bar{i}) - g(m) + \gamma' - \gamma' + 1\right) \\ &= \sigma(\gamma) + M\left(\frac{\bar{i} - \underline{i}}{2} + 1, \gamma'\right) + M\left(\frac{\bar{i} - \underline{i}}{2} + 1, g(\bar{i}) - g(m) + g(m) - g(\underline{i}) + 1 - \gamma' + 1\right) \\ &= \sigma(\gamma) + M\left(\frac{\bar{i} - \underline{i}}{2} + 1, \gamma'\right) + M\left(\frac{\bar{i} - \underline{i}}{2} + 1, \gamma - \gamma' + 1\right). \end{aligned}$$

By virtue of monotonicity, $g(m) \in \{g(\underline{i}), \dots, g(\bar{i})\}$ and so $g(m) - g(\underline{i}) + 1 \in \{1, \dots, g(\bar{i}) - g(\underline{i}) + 1\}$ or equivalently $\gamma' \in \{1, \dots, \gamma\}$. Consequently, the number of function evaluations is not greater than

$$\sigma(\gamma) + \max_{\gamma' \in \{1, \dots, \gamma\}} \left\{ M\left(\frac{\bar{i} - \underline{i}}{2} + 1, \gamma'\right) + M\left(\frac{\bar{i} - \underline{i}}{2} + 1, \gamma - \gamma' + 1\right) \right\}.$$

The case of k odd is very similar, but the divide-and-conquer algorithm splits the space unequally. If k is odd, then m equals $\frac{\underline{i} + \bar{i} - 1}{2}$. In this case $(\underline{i}, m) \in \mathcal{I}_{(k-1)/2}$ and $(m, \bar{i}) \in$

$\mathcal{I}_{(k-1)/2+1}$.²⁰ Consequently, computing the policy for $\underline{i}, \dots, \bar{i}$ takes no more than $\sigma(g(\bar{i}) - g(\underline{i}) - 1) + N(\underline{i}, m) + N(m, \bar{i})$ maximization steps. Defining γ and γ' the same as before and using the definition of m and N , we have the the required maximization steps is less than

$$\begin{aligned} & \sigma(\gamma) + M(m - \underline{i} + 1, \gamma') + M(\bar{i} - m + 1, \gamma - \gamma' + 1) \\ &= \sigma(\gamma) + M\left(\frac{\underline{i} + \bar{i} - 1}{2} - \underline{i} + 1, \gamma'\right) + M\left(\bar{i} - \frac{\underline{i} + \bar{i} - 1}{2} + 1, \gamma - \gamma' + 1\right) \\ &= \sigma(\gamma) + M\left(\frac{\bar{i} - \underline{i} + 1}{2}, \gamma'\right) + M\left(\frac{\bar{i} - \underline{i} + 1}{2} + 1, \gamma - \gamma' + 1\right) \end{aligned}$$

where γ and γ' are defined the same as before. Because M is increasing in the first argument, this is less than

$$\sigma(\gamma) + \max_{\gamma' \in \{1, \dots, \gamma\}} \left\{ M\left(\frac{\bar{i} - \underline{i} + 1}{2} + 1, \gamma'\right) + M\left(\frac{\bar{i} - \underline{i} + 1}{2} + 1, \gamma - \gamma' + 1\right) \right\}.$$

Combining the bounds for k even and odd, the required number of π evaluations is less than

$$\sigma(\gamma) + \max_{\gamma' \in \{1, \dots, \gamma\}} \left\{ M(\lfloor \frac{\bar{i} - \underline{i} + 1}{2} \rfloor + 1, \gamma') + M(\lfloor \frac{\bar{i} - \underline{i} + 1}{2} \rfloor + 1, \gamma - \gamma' + 1) \right\} \quad (*)$$

because if k is even, then $\lfloor \frac{\bar{i} - \underline{i} + 1}{2} \rfloor = \frac{\bar{i} - \underline{i}}{2}$. Consequently, $(*)$ gives an upper bound for any $(\underline{i}, \bar{i}) \in \mathcal{I}_k$ for $k \geq 1$ when $g(\underline{i})$ and $g(\bar{i})$ are known. If $N(\underline{i}, \bar{i})$ is less than this, then the proof by induction is complete.

Since $N(\underline{i}, \bar{i})$ is defined as $M(\bar{i} - \underline{i} + 1, g(\bar{i}) - g(\underline{i}) + 1)$, using the definitions of N and M shows

$$\begin{aligned} N(\underline{i}, \bar{i}) &= M(\bar{i} - \underline{i} + 1, g(\bar{i}) - g(\underline{i}) + 1) \\ &= M(\bar{i} - \underline{i} + 1, \gamma) \\ &= \sigma(\gamma) + \max_{\gamma' \in \{1, \dots, \gamma\}} \left\{ M(\lfloor \frac{\bar{i} - \underline{i} + 1}{2} \rfloor + 1, \gamma') + M(\lfloor \frac{\bar{i} - \underline{i} + 1}{2} \rfloor + 1, \gamma - \gamma' + 1) \right\}. \end{aligned}$$

Consequently, $N(\underline{i}, \bar{i})$ exactly equals the value in $(*)$, and the proof by induction is complete.

Step 1 of the algorithm requires at most $2\sigma(n')$ evaluations to compute $g(1)$ and $g(n)$. If $n = 2$, step 2 is never reached. Since $M(n, n') = 0$ in this case, $2\sigma(n') + M(n, n')$ provides an upper bound. If $n > 2$, then since $(1, n) \in \mathcal{I}_{n-1}$ and $g(1)$ and $g(n)$ known, only $N(1, n)$ additional evaluations are required. Therefore, to compute for each $i \in \{1, \dots, n\}$, no more

²⁰To see this, note that $(\underline{i}, \bar{i}) \in \mathcal{I}_k$ implies $k = \bar{i} - \underline{i}$. To have, $(\underline{i}, m) \in \mathcal{I}_{(k-1)/2}$, it must be that $m = \underline{i} + \frac{k-1}{2}$. This holds: $\underline{i} + \frac{k-1}{2} = \underline{i} + \frac{\bar{i} - \underline{i} - 1}{2} = \underline{i} + \frac{\bar{i} + \underline{i} - 1 - \underline{i} - \underline{i}}{2} = \underline{i} + m + \frac{-2\underline{i}}{2} = m$. Similarly, to have $(m, \bar{i}) \in \mathcal{I}_{(k-1)/2+1}$, one must have $\bar{i} = m + (k-1)/2 + 1$. This also obtains: $m + \frac{(k-1)}{2} + 1 = \frac{\underline{i} + \bar{i} - 1}{2} + \frac{\bar{i} - \underline{i} - 1}{2} + 1 = \frac{2\bar{i} - 2}{2} + 1 = \bar{i}$.

than $2\sigma(n') + N(1, n) = 2\sigma(n') + M(n, g(n) - g(1) + 1)$ function evaluations are needed. Lemma 3) then gives that this is less than $2\sigma(n') + M(n, n')$ since $g(n) - g(1) + 1 \leq n' - 1 + 1 = n'$.

□

Lemma 4. Define a sequence $\{m_i\}_{i=1}^{\infty}$ by $m_1 = 2$ and $m_i = 2m_{i-1} - 1$ for $i \geq 2$. Then $m_i = 2^{i-1} + 1$ and $\log_2(m_i - 1) = i - 1$ for all $i \geq 1$.

Proof. The proof of $m_i = 2^{i-1} + 1$ for all $i \geq 1$ is by induction. For $i = 1$, m_1 is defined as 2, which equals $2^{1-1} + 1$. For $i > 1$, suppose it holds for $i - 1$. Then

$$\begin{aligned} m_i &= 2m_{i-1} - 1 \\ &= 2[2^{i-2} + 1] - 1 \\ &= 2^{i-1} + 1. \end{aligned}$$

□

Lemma 5. Consider any $z \geq 2$. Then there exists a unique sequence $\{n_i\}_{i=1}^I$ such that $n_1 = 2$, $n_I = z$, $\lfloor \frac{n_i}{2} \rfloor + 1 = n_{i-1}$, and $n_i > 2$ for all $i > 1$. Moreover, $I = \lceil \log_2(z - 1) \rceil + 1$.

Proof. The proof that a unique sequence exists is by construction. Let $z \geq 2$ be fixed. Define an infinite sequence $\{z_i\}_{i=1}^{\infty}$ recursively as follows: Define $z_i = T_i(z)$ for all $i \geq 1$ with $T_1(z) := z$ and $T_{i+1}(z) = \lfloor \frac{T_i(z)}{2} \rfloor + 1$. We now establish all of the following: $T_i(z) \geq 2$, $T_i(z) \geq T_{i+1}(z)$, and $T_i(z) > T_{i+1}(z)$ whenever $T_i(z) > 2$. As an immediate consequence, for any $z \geq 2$, there exists a unique $I(z) \geq 1$ such that $T_{I(z)} = 2$ and, for all $i < I(z)$, $T_i(z) > 2$. We also show for later use that $T_i(z)$ is weakly increasing in z for every i .

To show $T_i(z) \geq 2$, the proof is by induction. We have $T_1(z) = z$ and $z \geq 2$. Now, consider some $i > 1$ and suppose it holds for $i - 1$. Then $T_i(z) = \lfloor \frac{T_{i-1}(z)}{2} \rfloor + 1 \geq \lfloor \frac{2}{2} \rfloor + 1 = 2$.

To show $T_i(z) > T_{i+1}(z)$ whenever $T_i(z) > 2$, consider two cases. First, consider $T_i(z)$ even. Then $T_{i+1}(z) = \lfloor \frac{T_i(z)}{2} \rfloor + 1 = \frac{T_i(z)}{2} + 1$ and so $T_{i+1}(z) < T_i(z)$ as long as $T_i(z) > 2$. Second, consider $T_i(z)$ odd. Then $T_{i+1}(z) = \lfloor \frac{T_i(z)}{2} \rfloor + 1 = \frac{T_i(z) - 1}{2} + 1$ and so $T_{i+1}(z) < T_i(z)$ as long as $T_i(z) > 1$.

To show that $T_i(z) \geq T_{i+1}(z)$, all we need to show now is that $T_{i+1}(z) = 2$ when $T_i(z) = 2$ (since the inequality is strict if $T_i(z) > 2$ and $T_i(z) \geq 2$ for all i). If $T_i(z) = 2$, then $T_{i+1}(z) = \lfloor \frac{2}{2} \rfloor + 1 = 2$.

To establish that $T_i(z)$ is weakly increasing in z for every i , the proof is by induction. For $a \leq b$, $T_1(a) = a \leq b = T_1(b)$. Now consider some $i > 1$ and suppose the induction hypothesis holds for $i - 1$. Then $T_i(a) = \lfloor T_{i-1}(a)/2 \rfloor + 1 \leq \lfloor T_{i-1}(b)/2 \rfloor + 1 = T_i(b)$.

The sequence $\{n_j\}_{j=1}^{I(z)}$ defined by $n_j = T_{I(z)-j+1}(z)$ —i.e., an inverted version of the sequence $\{T_i(z)\}_{i=1}^I$ —satisfies $n_{I(z)} = T_1(z) = z$, $n_1(z) = T_{I(z)} = 2$, and $n_{i-1} = T_{I(z)-(i-1)+1} = \lfloor \frac{T_{I(z)-(i-1)}}{2} \rfloor + 1 = \lfloor \frac{n_i}{2} \rfloor + 1$. Also, by the definition of $I(z)$, $T_i(z) > 2$ for any $i > I(z)$. So, if we can show that $I(z) = \lceil \log_2(z-1) \rceil + 1$, the proof is complete.

The proof of $I(z) = \lceil \log_2(z-1) \rceil + 1$ is as follows. Note that for $z = 2$, the sequence $\{z_i\}$ is simply $z_i = 2$ for all i which implies $I(2) = 1$. Since $\lceil \log_2(2-1) \rceil + 1 = 1$, the relationship holds for $z = 2$. So, now consider $z > 2$. The proof proceeds in the following steps. First, for the special $\{m_i\}$ sequence defined in Lemma 4, we show $T_j(m_i) = m_{i+1-j}$ for any $i \geq 1$ and any $j \leq i$. Second, we use this to show that $I(m_i) = i$ for all $i \geq 1$. Third, we show that $z \in (m_{i-1}, m_i]$ implies $I(z) = i$ by showing $I(m_i - 1) < I(z) \leq I(m_i)$. Fourth, we show that the i such that $z \in (m_{i-1}, m_i]$ is given by $\lceil \log_2(z-1) \rceil + 1$. This then gives $I(z) = \lceil \log_2(z-1) \rceil + 1$ since $I(z) = I(m_i) = i = \lceil \log_2(z-1) \rceil + 1$.

First, we show $T_j(m_i) = m_{i+1-j}$ for any $i \geq 1$ and any $j \leq i$. Fix some $i \geq 1$. The proof is by induction. For $j = 1$, $T_1(m_i) = m_i = m_{i+1-1}$. Now consider some j having $2 \leq j \leq i$ and suppose the induction hypothesis holds for $j-1$. Then

$$\begin{aligned} T_j(m_i) &= \lfloor \frac{T_{j-1}(m_i)}{2} \rfloor + 1 \\ &= \lfloor \frac{m_{i+1-(j-1)}}{2} \rfloor + 1 \\ &= \lfloor \frac{m_{i+2-j}}{2} \rfloor + 1 \\ &= \lfloor \frac{2m_{i+1-j} - 1}{2} \rfloor + 1 \\ &= m_{i+1-j} + \lfloor -\frac{1}{2} \rfloor + 1 \\ &= m_{i+1-j} - 1 + 1 \\ &= m_{i+1-j}, \end{aligned}$$

which proves $T_j(m_i) = m_{i+1-j}$ for $j \leq i$.

Second, we show $I(m_i) = i$ for all $i \geq 1$. Consider any $i \geq 1$. Then $T_j(m_i) = m_{i+1-j}$ for all $j \leq i$. Note then that $m_1 = 2$ and $m_j \geq 3$ for all $j \geq 2$. Since $I(z)$ is defined as the unique value such that $T_{I(z)}(z) = 2$ and $j < I(z)$ implies $T_j(z) > 2$, we must have $I(m_i) = i$ because $T_i(m_i) = 2$ and $j < i$ implies $T_j(m_i) = m_{i+1-j} \geq m_{i+1-(i-1)} = m_2 > 2$.

Third, we show that $z \in (m_{i-1}, m_i]$ implies $I(z) = i$ by showing $I(m_i - 1) < I(z) \leq I(m_i)$. Note that, since $z > 2$ (having taken care of the $z = 2$ case already), there is some $i \geq 2$ such that $z \in (m_{i-1}, m_i]$ (since $m_1 = 2$). To see $I(z) \leq I(m_i)$, suppose not, that $I(z) > I(m_i)$. But then $2 = T_{I(z)}(z) < T_{I(m_i)}(z) \leq T_{I(m_i)}(m_i) = 2$, which is a contradiction.²¹ Therefore,

²¹The second inequality uses that $T_i(z)$ is weakly increasing in z for every i , as established above.

$$I(z) \leq I(m_i).$$

To see $I(m_{i-1}) < I(z)$, we begin by showing $T_j(m_{i-1}) < T_j(m_{i-1} + \varepsilon)$ for any $\varepsilon > 0$ and any $j \leq i - 1$. It is equivalent to show $m_{i-j} < T_j(m_{i-1} + \varepsilon)$, which we show by induction. Clearly, for $j = 1$, we have $m_{i-1} < m_{i-1} + \varepsilon = T_1(m_{i-1} + \varepsilon)$. Now consider $j > 1$ and suppose it is true for $j - 1$. Then

$$\begin{aligned} T_j(m_{i-1} + \varepsilon) &= \left\lfloor \frac{T_{j-1}(m_{i-1} + \varepsilon)}{2} \right\rfloor + 1 \\ &= \left\lfloor \frac{T_{j-1}(m_{i-1} + \varepsilon) - m_{i-j+1} + m_{i-j+1}}{2} \right\rfloor + 1 \\ &= \left\lfloor \frac{T_{j-1}(m_{i-1} + \varepsilon) - m_{i-j+1} + 2m_{i-j} - 1}{2} \right\rfloor + 1 \\ &= \left\lfloor \frac{T_{j-1}(m_{i-1} + \varepsilon) - m_{i-j+1} - 1}{2} \right\rfloor + m_{i-j} + 1 \end{aligned}$$

Now, since the induction hypothesis of $T_{j-1}(m_{i-1} + \varepsilon) > m_{i-j+1}$ gives $T_{j-1}(m_{i-1} + \varepsilon) - m_{i-j+1} - 1 \geq 0$, one has

$$\begin{aligned} T_j(m_{i-1} + \varepsilon) &\geq \left\lfloor \frac{0}{2} \right\rfloor + m_{i-j} + 1 \\ &= m_{i-j} + 1 \\ &> m_{i-j}. \end{aligned}$$

Hence the proof by induction is complete.

Now, having established $T_j(m_{i-1}) < T_j(m_{i-1} + \varepsilon)$ for any $\varepsilon > 0$ and any $j \leq i - 1$, we show $I(m_{i-1}) < I(z)$. Suppose not, that $I(m_{i-1}) \geq I(z)$. Then since $z > m_{i-1}$, taking $\varepsilon = z - m_{i-1}$ we have $2 = T_{I(m_{i-1})}(m_{i-1}) < T_{I(m_{i-1})}(m_{i-1} + \varepsilon) = T_{I(m_{i-1})}(z) \leq T_{I(z)}(z) = 2$, which is a contradiction.

Lastly, we now show that the i such that $z \in (m_{i-1}, m_i]$ is given by $\lceil \log_2(z - 1) \rceil + 1$. That this holds can be seen as follows. Note that $z \in (m_{i-1}, m_i]$ implies $\log_2(z - 1) + 1 \in (\log_2(m_{i-1} - 1) + 1, \log_2(m_i - 1) + 1]$. Then, since $\log_2(m_j - 1) + 1 = j$ (from Lemma 4), we have $\log_2(z - 1) + 1 \in (i - 1, i]$. Then, by the definition of $\lceil \cdot \rceil$, one has $\lceil \log_2(z - 1) + 1 \rceil = i$, which of course is equivalent to $\lceil \log_2(z - 1) \rceil + 1 = i$. So, $I(z) = \lceil \log_2(z - 1) \rceil + 1$. $\square$

Proof of Lemma 1.

Proof. Fix some $z \geq 2$ (the z corresponds to n above, and we will characterize the upper bound for arbitrary, but fixed, z and any γ , which corresponds to n'). By Lemma 5, there is a strictly monotone increasing sequence $\{z_i\}_{i=1}^I$ with $z_I = z$, $z_i = \left\lfloor \frac{z_i + 1}{2} \right\rfloor + 1$ for $i < I$, and with $I = \lceil \log_2(z - 1) \rceil + 1$ (and having $z_1 = 2$).

For $i > 1$, define

$$W(z_i, \gamma) := \max_{\gamma' \in \{1, \dots, \gamma\}} M(z_{i-1}, \gamma') + M(z_{i-1}, \gamma - \gamma' + 1)$$

and for $i = 1$ define $W(z_i, \gamma) = 0$. The definition of M gives $M(z_i, \gamma) = \sigma(\gamma) + W(z_i, \gamma)$ for any $i > 1$ with $M(z_1, \cdot) = 0$. Note that $W(z_2, \gamma) = W(z_1, \gamma) = 0$.

Define $\bar{W}$ —an upper bound and continuous version of W —as

$$\bar{W}(z_i, \gamma) := \bar{\sigma}^*(\gamma) + \max_{\gamma' \in [1, \gamma]} \bar{W}(z_{i-1}, \gamma') + \bar{W}(z_{i-1}, \gamma - \gamma' + 1)$$

for $i > 2$ with $\bar{\sigma}^*(\gamma)$ defined as $\max_{\gamma' \in [1, \gamma]} \bar{\sigma}(\gamma') + \bar{\sigma}(\gamma - \gamma' + 1)$. For $i = 1$ or 2 , define $\bar{W}(z_i, \gamma) = 0$. Then $W(z_i, \gamma) \leq \bar{W}(z_i, \gamma)$ for all $i \geq 1$ and all $\gamma \in \mathbb{Z}^{++}$. The proof is by induction. They are equal for $i = 1$ and $i = 2$. Now consider an $i > 2$ and suppose it holds for $i - 1$.

$$\begin{aligned} W(z_i, \gamma) &= \max_{\gamma' \in \{1, \dots, \gamma\}} \sigma(\gamma') + \sigma(\gamma - \gamma' + 1) + W(z_{i-1}, \gamma') + W(z_{i-1}, \gamma - \gamma' + 1) \\ &\leq \max_{\gamma' \in \{1, \dots, \gamma\}} \sigma(\gamma') + \sigma(\gamma - \gamma' + 1) + \max_{\gamma' \in \{1, \dots, \gamma\}} W(z_{i-1}, \gamma') + W(z_{i-1}, \gamma - \gamma' + 1) \\ &\leq \max_{\gamma' \in \{1, \dots, \gamma\}} \bar{\sigma}(\gamma') + \bar{\sigma}(\gamma - \gamma' + 1) + \max_{\gamma' \in \{1, \dots, \gamma\}} \bar{W}(z_{i-1}, \gamma') + \bar{W}(z_{i-1}, \gamma - \gamma' + 1) \\ &\leq \max_{\gamma' \in [1, \gamma]} \bar{\sigma}(\gamma') + \bar{\sigma}(\gamma - \gamma' + 1) + \max_{\gamma' \in [1, \gamma]} \bar{W}(z_{i-1}, \gamma') + \bar{W}(z_{i-1}, \gamma - \gamma' + 1) \\ &\leq \bar{\sigma}^*(\gamma) + \max_{\gamma' \in [1, \gamma]} \bar{W}(z_{i-1}, \gamma') + \bar{W}(z_{i-1}, \gamma - \gamma' + 1) \\ &= \bar{W}(z_i, \gamma) \end{aligned}$$

If $\bar{\sigma}(\gamma) = \gamma$, then $\bar{\sigma}(\gamma') + \bar{\sigma}(\gamma - \gamma' + 1) = \gamma + 1$ which does not depend on γ' . So, $\bar{\sigma}^*(\gamma) = \gamma + 1 = 2\bar{\sigma}(\frac{\gamma+1}{2})$. If the $\bar{\sigma}$ function is strictly increasing, strictly concave, and differentiable, then the first order condition of the $\bar{\sigma}^*(\gamma)$ problem yields $\bar{\sigma}'(\gamma') = \bar{\sigma}'(\gamma - \gamma' + 1)$. The derivative is invertible (by strict concavity) and so $\gamma' = \frac{\gamma+1}{2}$. So, $\bar{\sigma}^*(\gamma) = \bar{\sigma}(\frac{\gamma+1}{2}) + \bar{\sigma}(\gamma - \frac{\gamma+1}{2} + 1)$, which gives $\bar{\sigma}^*(\gamma) = 2\bar{\sigma}(\frac{\gamma+1}{2})$, the same condition as in the linear case.²² So, for $i > 2$,

$$\bar{W}(z_i, \gamma) = 2\bar{\sigma}(\frac{\gamma+1}{2}) + \max_{\gamma' \in [1, \gamma]} \bar{W}(z_{i-1}, \gamma') + \bar{W}(z_{i-1}, \gamma - \gamma' + 1).$$

We will now show for $i > 2$ that $\bar{W}(z_i, \gamma) = 2\bar{\sigma}(\frac{\gamma+1}{2}) + 2\bar{W}(z_{i-1}, \frac{\gamma+1}{2})$, which gives a simple recursive relationship for the upper bound (for $i = 1$ or 2 , $\bar{W}(z_i, \gamma) = 0$). To do this,

²²Since this is an interior solution and the problem is concave, the constraint $\gamma' \in [1, \gamma]$ is not binding.

we will prove by induction that for any $i \geq 2$ and any γ that

$$\max_{\gamma' \in [1, \gamma]} \bar{W}(z_i, \gamma') + \bar{W}(z_i, \gamma - \gamma' + 1) = 2\bar{W}(z_i, \frac{\gamma+1}{2}),$$

which, as a consequence, immediately gives for $i > 2$ that

$$\begin{aligned} \bar{W}(z_i, \gamma) &= 2\bar{\sigma}(\frac{\gamma+1}{2}) + \max_{\gamma' \in [1, \gamma]} \bar{W}(z_{i-1}, \gamma') + \bar{W}(z_{i-1}, \gamma - \gamma' + 1) \\ &= 2\bar{\sigma}(\frac{\gamma+1}{2}) + 2\bar{W}(z_{i-1}, \frac{\gamma+1}{2}). \end{aligned}$$

For $i = 2$, the induction hypothesis clearly holds. So consider some $i > 2$ and suppose it holds for $i - 1$. Now,

$$\begin{aligned} &\max_{\gamma' \in [1, \gamma]} \bar{W}(z_i, \gamma') + \bar{W}(z_i, \gamma - \gamma' + 1) \\ &= \max_{\gamma' \in [1, \gamma]} 2\bar{\sigma}(\frac{\gamma'+1}{2}) + 2\bar{W}(z_{i-1}, \frac{\gamma'+1}{2}) + 2\bar{\sigma}(\frac{(\gamma - \gamma' + 1) + 1}{2}) + 2\bar{W}(z_{i-1}, \frac{(\gamma - \gamma' + 1) + 1}{2}) \\ &= 2 \max_{\gamma' \in [1, \gamma]} \bar{\sigma}(\frac{\gamma'+1}{2}) + \bar{\sigma}(\frac{\gamma+1}{2} - \frac{\gamma'+1}{2} + 1) + \bar{W}(z_{i-1}, \frac{\gamma'+1}{2}) + \bar{W}(z_{i-1}, \frac{\gamma+1}{2} - \frac{\gamma'+1}{2} + 1) \\ &= 2 \max_{\tilde{\gamma}' \in [1, \frac{\gamma+1}{2}]} \bar{\sigma}(\tilde{\gamma}') + \bar{\sigma}(\frac{\gamma+1}{2} - \tilde{\gamma}' + 1) + \bar{W}(z_{i-1}, \tilde{\gamma}') + \bar{W}(z_{i-1}, \frac{\gamma+1}{2} - \tilde{\gamma}' + 1) \\ &\leq 2 \max_{\tilde{\gamma}' \in [1, \frac{\gamma+1}{2}]} \bar{\sigma}(\tilde{\gamma}') + \bar{\sigma}(\frac{\gamma+1}{2} - \tilde{\gamma}' + 1) + 2 \max_{\tilde{\gamma}' \in [1, \frac{\gamma+1}{2}]} \bar{W}(z_{i-1}, \tilde{\gamma}') + \bar{W}(z_{i-1}, \frac{\gamma+1}{2} - \tilde{\gamma}' + 1) \\ &= 2\bar{\sigma}^*(\frac{\gamma+1}{2}) + 2 \cdot 2\bar{W}(z_{i-1}, \frac{\frac{\gamma+1}{2} + 1}{2}) \\ &= 2 \cdot 2\bar{\sigma}(\frac{\frac{\gamma+1}{2} + 1}{2}) + 2 \cdot 2\bar{W}(z_{i-1}, \frac{\frac{\gamma+1}{2} + 1}{2}) \\ &= 2 \left(2\bar{\sigma}(\frac{\frac{\gamma+1}{2} + 1}{2}) + 2\bar{W}(z_{i-1}, \frac{\frac{\gamma+1}{2} + 1}{2}) \right) \\ &= 2\bar{W}(z_i, \frac{\gamma+1}{2}) \end{aligned}$$

This establishes $\max_{\gamma' \in [1, \gamma]} \bar{W}(z_i, \gamma') + \bar{W}(z_i, \gamma - \gamma' + 1) \leq 2\bar{W}(z_i, \frac{\gamma+1}{2})$. However, $2\bar{W}(z_i, \frac{\gamma+1}{2})$ is just $\bar{W}(z_i, \gamma') + \bar{W}(z_i, \gamma - \gamma' + 1)$ evaluated at $\frac{\gamma+1}{2}$, and so $\max_{\gamma' \in [1, \gamma]} \bar{W}(z_i, \gamma') + \bar{W}(z_i, \gamma - \gamma' + 1) \geq 2\bar{W}(z_i, \frac{\gamma+1}{2})$. Hence, they are equal, and the proof by induction is complete.

Now, fix any $\gamma \geq 1$ (corresponding to n' in the statement of the lemma) and define

$\gamma_I := \gamma$ and $\gamma_i = \frac{\gamma_{i+1}+1}{2}$. Then $\gamma_i = 2^{i-I}(\gamma_I - 1) + 1$.²³ Then, for $i > 2$,

$$\bar{W}(z_i, \gamma_i) = 2\bar{\sigma}(\gamma_{i-1}) + 2\bar{W}(z_{i-1}, \gamma_{i-1}).$$

So, if $I > 2$, repeatedly expand the expression to find a value for $\bar{W}(z_I, \gamma_I)$:

$$\begin{aligned} \bar{W}(z_I, \gamma_I) &= 2\bar{\sigma}(\gamma_{I-1}) + 2\bar{W}(z_{I-1}, \gamma_{I-1}) \\ &= 2\bar{\sigma}(\gamma_{I-1}) + 2^2\bar{\sigma}(\gamma_{I-2}) + 2^2\bar{W}(z_{I-2}, \gamma_{I-2}) \\ &= 2\bar{\sigma}(\gamma_{I-1}) + \dots + 2^{I-2}\bar{\sigma}(\gamma_2) + 2^{I-2}\bar{W}(z_2, \gamma_2) \\ &= 2\bar{\sigma}(\gamma_{I-1}) + \dots + 2^{I-2}\bar{\sigma}(\gamma_2) \\ &= \sum_{j=1}^{I-2} 2^j \bar{\sigma}(\gamma_{I-j}) \\ &= \sum_{j=1}^{I-2} 2^j \bar{\sigma}(2^{I-j-I}(\gamma_I - 1) + 1) \\ &= \sum_{j=1}^{I-2} 2^j \bar{\sigma}(2^{-j}(\gamma_I - 1) + 1) \end{aligned}$$

If $I \leq 2$, then $\bar{W}(z_I, \gamma_I) = 0$.

Proposition 1 shows the number of required evaluations is less than or equal to $2\sigma(\gamma_I) + M(z_I, \gamma_I)$ for $z_I \geq 2$ and $\gamma_I \geq 1$. Since $M(z_i, \gamma) = \sigma(\gamma) + W(z_i, \gamma)$ for any $i > 1$ (with $M(z_1, \gamma) = 0$) and $W(z_i, \gamma) \leq \bar{W}(z_i, \gamma)$, the required function evaluations are weakly less than $3\sigma(\gamma_I) + \bar{W}(z_I, \gamma_I)$ for any I (recalling $W(z_I, \gamma_I) = 0$ for $I \leq 2$). Hence, if $I > 2$, then an upper bound is

$$3\sigma(\gamma_I) + \sum_{j=1}^{I-2} 2^j \bar{\sigma}(2^{-j}(\gamma_I - 1) + 1)$$

and if $I \leq 2$ an upper bound is $3\sigma(\gamma_I)$.

□

Proof of Proposition 2.

Proof. Define $\bar{W}(n, n') = \sum_{j=1}^{I-2} 2^j(2^{-j}(n' - 1) + 1)$ where $I = \lceil \log_2(n - 1) \rceil + 1$. Since $n \geq 4$,

²³The proof of this is by induction. First note for $i = I$ since $\frac{\gamma_{I+1}}{2} = 2^{(I-1)-I}(\gamma_I - 1) + 1$. Now if let inductive hypothesis be $\gamma_{j+1} = 2^{j+1-I}(\gamma_I - 1) + 1$, then it follows immediately that $\gamma_j = \frac{\gamma_{j+1}+1}{2} = \frac{2^{j+1-I}(\gamma_I - 1) + 2}{2}$. And the induction is completed.

$I > 2$. In the case of brute force with $\bar{\sigma}(\gamma) = \gamma$,

$$\begin{aligned}
W(n, n') &= (I - 2)(n' - 1) + \sum_{j=1}^{I-2} 2^j \\
&= (I - 2)(n' - 1) + 2^{I-1} - 2 \\
&= (I - 2)(n' - 1) + 2^{\lceil \log_2(n-1) \rceil} - 2 \\
&\leq (I - 2)(n' - 1) + 2^{\log_2(n-1)+1} - 2 \\
&\leq (I - 2)(n' - 1) + 2(n - 1) - 2 \\
&\leq (I - 2)(n' - 1) + 2n - 4 \\
&\leq (\lceil \log_2(n - 1) \rceil - 1)(n' - 1) + 2n - 4 \\
&\leq \log_2(n - 1)(n' - 1) + 2n - 4
\end{aligned}$$

So, no more than $\log_2(n - 1)(n' - 1) + 3n' + 2n - 4$ evaluations are required.

In the case of binary concavity, Lemma 2 shows $\sigma(\gamma) = 2\lceil \log_2(\gamma) \rceil - 1$ for $\gamma \geq 3$ and $\sigma(\gamma) = \gamma$ for $\gamma \leq 3$ is an upperbound. Now consider $\bar{\sigma}(\gamma) = 2\log_2(\gamma) + 1$. It is a strictly increasing, strictly concave, and differentiable function. For $\gamma = 1$ or 2 , one can plug in values to find $\sigma(\gamma) \leq \bar{\sigma}(\gamma)$. Additionally, for $\gamma \geq 3$, $\sigma(\gamma) \leq 2\lceil \log_2(\gamma) \rceil - 1 \leq 2(1 + \log_2(\gamma)) - 1 = \bar{\sigma}(\gamma)$. So, it satisfies $\bar{\sigma}$ satisfies all the conditions of Lemma 1.

Plugging in, one finds

$$\begin{aligned}
\bar{W}(n, n') &= \sum_{j=1}^{I-2} 2^j [1 + 2\log_2(2^{-j}(n' - 1) + 1)] \\
&= (2^{I-1} - 2) + 2 \sum_{j=1}^{I-2} 2^j \log_2(2^{-j}(n' - 1) + 1)
\end{aligned}$$

To handle the $\log_2(2^{-j}(n' - 1) + 1)$ term, we break the summation into two parts, one with $2^{-j}(n' - 1) < 1$ and one with $2^{-j}(n' - 1) \geq 1$. We do this to exploit the following fact: For $x \geq 1$, $\log_2(x + 1) \leq \log_2(x) + 1$ since they are equal at $x = 1$ the right hand side grows more

quickly in x (i.e., the derivative of $\log_2(x+1)$ is less than the derivative of $\log_2(x)+1$).

$$\begin{aligned}
\bar{W}(n, n') &= (2^{I-1} - 2) + 2 \sum_{j=1}^{I-2} \mathbf{1}[2^{-j}(n' - 1) < 1] 2^j \log_2(2^{-j}(n' - 1) + 1) \\
&\quad + 2 \sum_{j=1}^{I-2} \mathbf{1}[2^{-j}(n' - 1) \geq 1] 2^j \log_2(2^{-j}(n' - 1) + 1) \\
&\leq (2^{I-1} - 2) + 2 \sum_{j=1}^{I-2} \mathbf{1}[2^{-j}(n' - 1) < 1] 2^j \\
&\quad + 2 \sum_{j=1}^{I-2} \mathbf{1}[2^{-j}(n' - 1) \geq 1] 2^j \log_2(2^{-j}(n' - 1) + 1) \\
&\leq (2^{I-1} - 2) + 2 \sum_{j=1}^{I-2} 2^j + 2 \sum_{j=1}^{I-2} \mathbf{1}[2^{-j}(n' - 1) \geq 1] 2^j \log_2(2^{-j}(n' - 1) + 1) \\
&= 3(2^{I-1} - 2) + 2 \sum_{j=1}^{I-2} \mathbf{1}[2^{-j}(n' - 1) \geq 1] 2^j \log_2(2^{-j}(n' - 1) + 1)
\end{aligned}$$

Now, exploiting $\log_2(x+1) \leq \log_2(x) + 1$ for $x \geq 1$,

$$\begin{aligned}
&\leq 3(2^{I-1} - 2) + 2 \sum_{j=1}^{I-2} \mathbf{1}[2^{-j}(n' - 1) \geq 1] 2^j [1 + \log_2(2^{-j}(n' - 1))] \\
&\leq 5(2^{I-1} - 2) + 2 \sum_{j=1}^{I-2} \mathbf{1}[2^{-j}(n' - 1) \geq 1] 2^j \log_2(2^{-j}(n' - 1)) \\
&= 5(2^{I-1} - 2) + 2 \sum_{j=1}^{I-2} \mathbf{1}[j \leq \log_2(n' - 1)] 2^j \log_2(2^{-j}(n' - 1)) \\
&= 5(2^{I-1} - 2) + 2 \sum_{j=1}^{\lceil \log_2(n' - 1) \rceil} 2^j \log_2(2^{-j}(n' - 1)) \\
&= 5(2^{I-1} - 2) + 2 \sum_{j=1}^{\lceil \log_2(n' - 1) \rceil} 2^j (\log_2(n' - 1) - j) \\
&= 5(2^{I-1} - 2) + 2 \log_2(n' - 1) \sum_{j=1}^{\lceil \log_2(n' - 1) \rceil} 2^j - 2 \sum_{j=1}^{\lceil \log_2(n' - 1) \rceil} 2^j j
\end{aligned}$$

Then defining $J = \lceil \log_2(n' - 1) \rceil$ and using that (for $r \neq 1$) $\sum_{j=a}^b jr^j = \frac{ar^a - br^{b+1}}{1-r} +$

$$\frac{r^{a+1}-r^{b+1}}{(1-r)^2},^{24}$$

$$\begin{aligned}
\bar{W}(n, n') &\leq 5(2^{I-1} - 2) + 2\log_2(n' - 1)[2^{J+1} - 2] - 2[-2 + J2^{J+1} + 2^2 - 2^{J+1}] \\
&= 5(2^{I-1} - 2) + \log_2(n' - 1)[2^{J+2} - 4] - 4 - (J - 1)2^{J+2} \\
&= 5 \cdot 2^{I-1} + \log_2(n' - 1)[2^{J+2} - 4] - (J - 1)2^{J+2} - 14 \\
&= 5 \cdot 2^{I-1} + \log_2(n' - 1)[2^{\lceil \log_2(n'-1) \rceil + 2} - 4] - (J - 1)2^{\lceil \log_2(n'-1) \rceil + 2} - 14 \\
&\leq 5 \cdot 2^{I-1} + \log_2(n' - 1)[(n' - 1)2^3 - 4] - (J - 1)(n' - 1)2^3 - 14 \\
&= 5 \cdot 2^{I-1} + \log_2(n' - 1)[8(n' - 1) - 4] - (\lceil \log_2(n' - 1) \rceil - 1)8(n' - 1) - 14 \\
&\leq 5 \cdot 2^{I-1} + \log_2(n' - 1)[8(n' - 1) - 4] - (\log_2(n' - 1) - 1)8(n' - 1) - 14 \\
&= 5 \cdot 2^{\lceil \log_2(n'-1) \rceil} - 4\log_2(n' - 1) + 8(n' - 1) - 14 \\
&\leq 10(n - 1) - 4\log_2(n' - 1) + 8n' - 22 \\
&= 10n + 8n' - 4\log_2(n' - 1) - 32
\end{aligned}$$

□

B.2 Equivalence between the two problems

This subsection establishes the equivalence between (1) and (2). Currently, the notation for the $\tilde{\pi}$ and π problems are the opposite of what is in the main text.

Suppose $I'(i) \subset \{1, \dots, n'\}$ may be empty, but is monotonically increasing. Let $I = \{i | i \in \{1, \dots, n\}, I'(i) \neq \emptyset\}$ so that $i \in I$ has a feasible solution. For all $i \in I$, let

$$\Pi(i) := \max_{i' \in I'(i)} \pi(i, i').$$

and let $G(i) := \arg \max_{i' \in I'(i)} \pi(i, i')$. Let $\mathcal{G}$ be the set of optimal policies, i.e., for every $g \in \mathcal{G}$ and every $i \in I$, $g(i) \in G(i)$. Further, suppose there is a $\underline{\pi}$ such that $\pi(i, i') > \underline{\pi}$ for all $i \in I$ and $i' \in I'(i)$ (i.e., π is bounded below, strictly, whenever defined).

Define

$$\tilde{\Pi}(i) = \max_{i' \in \{1, \dots, n'\}} \tilde{\pi}(i, i')$$

and

$$\tilde{\pi}(i, i') = \begin{cases} \pi(i, i') & \text{if } I'(i) \neq \emptyset \text{ and } i' \in I'(i) \\ \underline{\pi} & \text{if } I'(i) \neq \emptyset \text{ and } i' \notin I'(i) \\ \mathbf{1}[i' = 1] & \text{if } I'(i) = \emptyset \end{cases}$$

²⁴To find the value of the weighted geometric series, define $S = \sum_{j=a}^b jr^j$, cancel terms to find $(1-r)S = ar^a - br^{b+1} + \frac{r^{a+1}-r^{b+1}}{1-r}$, apply the usual formula for the (non-weighted) geometric series, and divide through by $1-r$.

Similarly, let $\tilde{G}(i) := \arg \max_{i' \in \{1, \dots, n'\}} \tilde{\pi}(i, i')$. Let $\tilde{\mathcal{G}}$ be the set of optimal policies, i.e., for every $\tilde{g} \in \tilde{\mathcal{G}}$ and every $i \in \{1, \dots, n\}$, $\tilde{g}(i) \in \tilde{G}(i)$.

Lemma 6. *All of the following are true:*

1. $\Pi(i) = \tilde{\Pi}(i)$ for all $i \in I$.
2. $G(i) = \tilde{G}(i)$ for all $i \in I$.
3. *If every $g \in \mathcal{G}$ is weakly increasing, then every $\tilde{g} \in \tilde{\mathcal{G}}$ is also weakly increasing. NOTE: This requires $I'(i)$ to be monotonically increasing, as is assumed, so that the infeasible region is $\{1, \dots, \underline{i} - 1\}$ and $G(i)$ in the infeasible region equals $\{1\}$.*

Proof. For the proof of the first claim, let $i \in I$. Then $I'(i) \neq \emptyset$. Therefore,

$$\tilde{\Pi}(i) = \max_{i' \in \{1, \dots, n'\}} \tilde{\pi}(i, i') = \max\{\underline{\pi}, \max_{i' \in I'(i)} \pi(i, i')\} = \max_{i' \in I'(i)} \pi(i, i') = \Pi(i)$$

where the third equality is justified by $\pi(i, i') > \tilde{\pi}$ for all feasible i' .

For the proof of the second claim, again let $i \in I$. Then $I'(i) \neq \emptyset$. As before, infeasible choices, i.e., $i' \in \{1, \dots, n'\} \setminus I'(i)$, are strictly suboptimal because any feasible choice $j' \in I'(i)$ delivers $\tilde{\pi}(i, j') > \pi = \tilde{\pi}(i, i')$. Hence,

$$\tilde{G}(i) = \arg \max_{i' \in \{1, \dots, n'\}} \tilde{\pi}(i, i') = \arg \max_{i' \in I'(i)} \tilde{\pi}(i, i') = \arg \max_{i' \in I'(i)} \pi(i, i') = G(i).$$

For the proof of the third claim, first note that if every i has a feasible choice, i.e. $I = \{1, \dots, n\}$, then $G(i) = \tilde{G}(i)$ for all $i \in \{1, \dots, n\}$ and hence $\mathcal{G} = \tilde{\mathcal{G}}$ giving the desired result. So, suppose there is a nonempty set of states J where $j \in J$ implies $I'(j) = \emptyset$. Then because $I'(i)$ is monotone increasing, $J = \{1, \dots, \underline{i}\}$ for some $\underline{i} \leq n'$.

Let $\tilde{g} \in \tilde{\mathcal{G}}$. For any $j \in J$, $\tilde{G}(j) = \arg \max_{i' \in \{1, \dots, n'\}} \mathbf{1}[i' = 1] = \{1\}$. Hence, $\tilde{g}(i) = 1$ for all $i \in \{1, \dots, \underline{i}\} = J$. So, since $\tilde{g}(i) \geq 1$ for all i , the desired result is had if $\tilde{g}$ is monotone increasing over the domain $\{\underline{i} + 1, \dots, n\} = I$. Because $\tilde{G}(i) = G(i)$ for all $i \in I$, $\tilde{g}$ restricted to the domain I is in $\mathcal{G}$. Since every $g \in \mathcal{G}$ is monotone increasing, $\tilde{g}$ restricted to the domain I must be increasing, and so the result obtains. □

Definition. *We say the problem is concave if for all $i \in I$, $I'(i) = \{1, \dots, n'(i)\}$ and for any $i' < n'(i)$, $\pi(i, \cdot)$ is either first strictly increasing and then weakly decreasing; or is always weakly decreasing; or is always strictly increasing.*

Note that this formulation allows for multiple maxima but ensure that the argmax is an integer interval. The strictly increasing is necessary because the algorithms, when indifferent

between two points, assume there is a maximum to the left of the two points. Note that if there is some strictly concave function $f(x)$ such that $\pi(i, i') = f(i')$ for all $i' \in I'(i)$, then π satisfies the conditions. In fact, if f is strictly quasi-concave, then it satisfies these conditions.

Lemma 7. *If the problem is concave and $\tilde{\pi}(i, j) \geq \tilde{\pi}(i, j+1)$ for some j , then $\tilde{\pi}(i, j) = \max_{i' \in \{j, \dots, n'\}} \tilde{\pi}(i, i')$. If $\tilde{\pi}(i, k-1) < \tilde{\pi}(i, k)$ for some k , then $\tilde{\pi}(i, k) = \max_{i' \in \{1, \dots, k\}} \tilde{\pi}(i, i') = \max_{i' \in \{1, \dots, k\}} \pi(i, i')$.*

Proof. First, we prove the case for $\tilde{\pi}(i, j) \geq \tilde{\pi}(i, j+1)$.

Consider two cases. First, suppose $i \notin I$. Then $I'(i) = \emptyset$ and $\tilde{\pi}(i, i') = \mathbf{1}[i' = 1]$. Consequently, for any j , $\tilde{\pi}(i, j) = \max_{i' \in \{j, \dots, n'\}} \tilde{\pi}(i, i')$.

Now, suppose $i \in I$. If $j > n'(i)$, then $\tilde{\pi}(i, j) = \underline{\pi} = \max_{i' \in \{j, \dots, n'\}} \tilde{\pi}(i, i')$. If $j = n'(i)$, then $\tilde{\pi}(i, j) > \underline{\pi} = \max_{i' \in \{j+1, \dots, n'\}} \tilde{\pi}(i, i')$ implying $\tilde{\pi}(i, j) = \max_{i' \in \{j+1, \dots, n'\}} \tilde{\pi}(i, i')$. If $j < n'(i)$, then

$$\begin{aligned} \max_{i' \in \{j, \dots, n'\}} \tilde{\pi}(i, i') &= \max\left\{ \max_{i' \in \{j, \dots, n'(i)\}} \tilde{\pi}(i, i'), \max_{i' \in \{n'(i)+1, \dots, n'\}} \tilde{\pi}(i, i') \right\} \\ &= \max\left\{ \max_{i' \in \{j, \dots, n'(i)\}} \tilde{\pi}(i, i'), \underline{\pi} \right\} \\ &= \max_{i' \in \{j, \dots, n'(i)\}} \tilde{\pi}(i, i') \\ &= \max_{i' \in \{j, \dots, n'(i)\}} \pi(i, i'). \end{aligned}$$

Since $\tilde{\pi}(i, j) = \pi(i, j)$ and $\tilde{\pi}(i, j+1) = \pi(i, j+1)$, $\pi(i, j) \geq \pi(i, j+1)$. Because the problem is concave and $\pi(i, \cdot)$ is either flat or decreasing from j to $j+1$, it must be decreasing from j to $n'(i)$. Hence, $\pi(i, j) = \max_{i' \in \{j, \dots, n'(i)\}} \pi(i, i')$. So, $\tilde{\pi}(i, j) = \pi(i, j) = \max_{i' \in \{j, \dots, n'(i)\}} \pi(i, i') = \max_{i' \in \{j, \dots, n'\}} \tilde{\pi}(i, i')$.

Now, we prove the case for $\tilde{\pi}(i, k-1) < \tilde{\pi}(i, k)$. In this case, $k-1$ and k must both be feasible, i.e. $k \leq n'(i)$, because (1) if they were both infeasible, then $\tilde{\pi}(i, k-1) = \underline{\pi} = \tilde{\pi}(i, k)$ and (2) if only k were infeasible, then $\tilde{\pi}(i, k-1) > \underline{\pi} = \tilde{\pi}(i, k)$. Given that $k-1$ and k are feasible, $\tilde{\pi}(i, k-1) = \pi(i, k-1)$ and $\tilde{\pi}(i, k) = \pi(i, k)$. Since $\pi(i, \cdot)$ is strictly increasing until it switches to weakly decreasing, $\pi(i, 1) < \dots < \pi(i, k-1) < \pi(i, k)$. Since all of $1, \dots, k$ are feasible, this also gives $\pi(i, 1) = \tilde{\pi}(i, 1) < \dots < \pi(i, k-1) = \tilde{\pi}(i, k-1) < \pi(i, k) = \tilde{\pi}(i, k)$. Hence, $\tilde{\pi}(i, k) = \max_{i' \in \{1, \dots, k\}} \tilde{\pi}(i, i') = \max_{i' \in \{1, \dots, k\}} \pi(i, i')$. □

Lemma 8. *Suppose it is known that $\tilde{G}(i) \cap \{a, \dots, b\}$ is nonempty. Then brute force applied to*

$$\max_{i' \in \{a, \dots, b\}} \tilde{\pi}(i, i')$$

delivers an optimal solution, i.e., letting $\hat{g}$ be the choice the algorithm delivers, $\hat{g} \in \tilde{G}(i)$. Additionally, if the problem is concave, then the simple concavity and binary concavity algorithms also deliver an optimal solution.

Proof. First, suppose $i \notin I$ so that $\tilde{\pi}(i, i') = \mathbf{1}[i' = 1]$. Then it must be that $a = 1$ since $\tilde{G}(i) = \{1\}$. Brute force clearly finds the optimum since it checks every value of i' . Simple concavity will compare $i' = a = 1$ against $i' = a + 1 = 2$ and find $i' = 2$ is strictly worse. So, it stops and gives $\hat{g} = 1$, implying $\hat{g} \in \tilde{G}(i) = \{1\}$. Binary concavity first checks whether $b - a + 1 \leq 2$. If so, it is the same as brute force. If not, it checks whether $b - a + 1 \leq 3$. If so, then $b - a + 1 = 3$ and it does a comparison of either (1) a and $m = (b + a)/2$, in which case it correctly identifies the maximum as a or (2) m and b in which case it drops b from the search space and does a brute force comparison of a and $a + 1$ (when it goes to step 2). If $b - a + 1 > 3$, it will evaluate the midpoint $m = \lfloor (a + b)/2 \rfloor$ and $m + 1$ and find $\tilde{\pi}(i, m) = \tilde{\pi}(i, m + 1) = 0$. It will then proceed to step 2, searching for the optimum in $\{1, \dots, m\}$ with $a = 1$ and $b = m$ in the next iteration of the recursive algorithm. This proceeds until $b - a + 1 \leq 3$, where it then correctly identifies the maximum (as was just shown). Therefore, binary concavity finds a correct solution, $\hat{g} \in \tilde{G}(i)$.

Now, suppose $i \in I$. Because brute force will evaluate $\tilde{\pi}(i, \cdot)$ at every $i' \in \{a, \dots, b\}$, it finds $\hat{g} \in \tilde{G}(i)$.

Now, suppose the problem is concave.

The simple concavity algorithm evaluates $\tilde{\pi}(i, i')$ at $i' \in \{a, \dots, b\}$ sequentially until it reaches an $x \in \{a + 1, \dots, b\}$ that $\tilde{\pi}(i, x - 1) \geq \tilde{\pi}(i, x)$. If this stopping rule is not triggered, then simple concavity is identical to brute force and so finds an optimal solution. So, it suffices to consider otherwise. In this case, $x - 1$ satisfies the conditions for “ j ” in Lemma 7 and hence $\tilde{\pi}(i, x - 1) = \max_{i' \in \{x-1, \dots, n'\}} \tilde{\pi}(i, i')$. By virtue of not having stopped until $x - 1$, $\tilde{\pi}(i, x - 1) \geq \max_{i' \in \{a, \dots, x-1\}} \tilde{\pi}(i, i')$. Consequently, $\tilde{\pi}(i, x - 1) \geq \max_{i' \in \{a, \dots, x-1\} \cup \{x-1, \dots, n'\}} \tilde{\pi}(i, i') = \max_{i' \in \{a, \dots, n'\}} \tilde{\pi}(i, i')$. Since a maximum is known to be in $\{a, \dots, b\}$, i.e. $\tilde{G}(i) \cap \{a, \dots, b\}$,

$$\tilde{\Pi}(i) = \max_{i' \in \{a, \dots, b\}} \tilde{\pi}(i, i') \leq \max_{i' \in \{a, \dots, n'\}} \tilde{\pi}(i, i') \leq \tilde{\pi}(i, x - 1) \leq \tilde{\Pi}(i).$$

So, $\tilde{\pi}(i, x - 1) = \tilde{\Pi}(i)$ giving $x - 1 \in \tilde{G}(i)$.

Now consider the modified binary concavity algorithm. If $b \leq a + 1$, the algorithm is the same as brute force and so finds a maximum. If $b = a + 2$ (corresponding to step 3 in the algorithm), the algorithm goes to either step 3(a) or step 3(b). In step 3(a), it stops if $\tilde{\pi}(i, a) > \tilde{\pi}(i, m)$ (where $m = (a + b)/2$) taking the maximum as a and otherwise does the same as brute force. So, suppose the stopping condition is satisfied. A maximum is a as long as $\tilde{\pi}(i, a) = \max_{i' \in \{a, \dots, b\}} \tilde{\pi}(i, i')$, which it is since a satisfies the conditions for “ j ” in

Lemma 7. In step 3(b), it stops if $\tilde{\pi}(i, b) > \tilde{\pi}(i, m)$ taking the maximum as b and otherwise does the same as brute force. So, suppose the stopping condition is satisfied. A maximum is b as long as $\tilde{\pi}(i, b) = \max_{i' \in \{a, \dots, b\}} \tilde{\pi}(i, i')$, which is true since b satisfies all the conditions for “ k ” in Lemma 7.

If $b \geq a + 3$, binary concavity goes to step 4 of the algorithm. In this case, it evaluates at two points $m = \lfloor (a + b)/2 \rfloor$ and $m + 1$. If $\tilde{\pi}(i, m) \geq \tilde{\pi}(i, m + 1)$, it assumes a maximum is in $\{a, \dots, m\}$. Since m satisfies the conditions for “ j ” in Lemma 7, $\tilde{\pi}(i, m) \geq \max_{i' \in \{m, \dots, b\}} \tilde{\pi}(i, i')$ giving $\max_{i' \in \{a, \dots, b\}} \tilde{\pi}(i, i') = \max_{i' \in \{a, \dots, m\}} \tilde{\pi}(i, i')$ justifying this assumption. If $\tilde{\pi}(i, m) < \tilde{\pi}(i, m + 1)$, it instead assumes a maximum is in $\{m + 1, \dots, b\}$. This again is justified since $m + 1$ satisfies all the conditions for “ k ” in Lemma 7 and so $m + 1$ is better than any value of $i' < m + 1$. The algorithm repeatedly divides $\{a, \dots, b\}$ into either $\{a, \dots, m\}$ or $\{m + 1, \dots, b\}$ until the size of the interval is either two or three. Since we have already shown the algorithm correctly identifies a maximum when the interval is of size two or three (i.e., $b = a + 1$ or $b = a + 2$), the algorithm correctly finds the maximum for larger intervals as long as this subdivision stops in a finite number of iterations (since then induction can be applied). Lemma 2 shows the required number of function evaluations is finite, and so this holds. $\square$

Proposition 3. *Any of brute force, simple, or binary monotonicity combined with any of brute force, simple, or binary concavity delivers an optimal solution provided the policies are monotone and the problem is concave as required by the algorithm choices. That is, letting $\hat{g}$ be the solution the algorithm finds, $\hat{g} \in \tilde{\mathcal{G}}$.*

Proof. Each of the brute force, simple, and binary monotonicity algorithms can be thought of as iterating through states i (in some order that, in the case of binary monotonicity, depends on $\tilde{\pi}$) with a search space $\{a, \dots, b\}$. If every state is visited and optimal choice is found at each state, then an optimal solution is found. So, it suffices to show that each of the brute force, simple, and binary monotonicity algorithms explore every state $i \in \{1, \dots, n\}$ and at each state, the following conditions are met so that Lemma 8 can be applied: (1) $\{a, b\} \subset \{1, \dots, n'\}$; (2) $a \leq b$; and (3) $\tilde{G}(i) \cap \{a, \dots, b\} \neq \emptyset$. The application of Lemma 8 gives $\hat{g}(i) \in \tilde{G}(i)$ (provided an appropriate concavity algorithm is used).

Brute force monotonicity trivially explores all states $i \in \{1, \dots, n\}$ sequentially. At each i , $a = 1$ and $b = n'$. Consequently, $\tilde{G}(i) \cap \{a, \dots, b\} \neq \emptyset$ and Lemma 8 can be applied.

Now, we prove simple monotonicity and binary monotonicity deliver a correct solution when $\mathcal{G}$ consists of monotone policy functions (implying, by Lemma 6, $\tilde{\mathcal{G}}$ also has monotone policies).

The simple monotonicity algorithm explores all states $i \in \{1, \dots, n\}$ sequentially always

with $b = n'$ (and so $a \leq b$). For $i = 1$, $a = 1$ and so $\tilde{G}(1) \cap \{a, \dots, b\} \neq \emptyset$. Consequently, Lemma 8 gives that $\hat{g}(1) \in \tilde{G}(1)$. Now, consider some $i > 1$ and suppose that $\hat{g}(i-1) \in \tilde{G}(i-1)$. We shall show $\hat{g}(i) \in \tilde{G}(i-1)$. Because every $\tilde{g} \in \tilde{\mathcal{G}}$ is monotone, $\max \tilde{G}(i-1) \leq \min \tilde{G}(i)$. Since $a = \hat{g}(i-1)$ and $\hat{g}(i-1) \in \tilde{G}(i-1)$, we have $a \leq \min \tilde{G}(i)$. This implies $\tilde{G}(i) \cap \{a, \dots, n'\} = \tilde{G}(i) \neq \emptyset$. So Lemma 8 applies, and $\hat{g}(i) \in \tilde{G}(i)$ completing the induction argument.

Now consider the binary monotonicity algorithm. If $n = 1$ or $n = 2$, the algorithm is the same as simple monotonicity and so delivers a correct solution. If $n > 2$, then the algorithm first correctly identifies $\hat{g}(1)$ (by brute force) and $\hat{g}(n)$ (using the same argument as simple monotonicity). It then defines $\underline{i} = 1$ and $\bar{i} = n$ and maintains the assumption that $\hat{g}(\underline{i}) \in \tilde{G}(\underline{i})$ and $\hat{g}(\bar{i}) \in \tilde{G}(\bar{i})$.

The goal of step 2 is to find the optimal solution for all $i \in \{\underline{i}, \dots, \bar{i}\}$. The algorithm stops at step 2(a) if $\bar{i} \leq \underline{i} + 1$, in which case this objective is clearly met since $\{\underline{i}, \dots, \bar{i}\} = \{\underline{i}, \bar{i}\}$. If the algorithm does not stop, then it computes $\hat{g}(m)$ for $m = \lfloor (a+b)/2 \rfloor$ using the search space $\{\hat{g}(\underline{i}), \dots, \hat{g}(\bar{i})\}$. By Lemma 8, an optimum is found as long as $\tilde{G}(m) \cap \{\hat{g}(\underline{i}), \dots, \hat{g}(\bar{i})\} \neq \emptyset$. Because monotonicity implies that $\max \tilde{G}(i) \leq \min \tilde{G}(i+1)$ for all i , one has $\tilde{G}(m) \cap \{\hat{g}(\underline{i}), \dots, \hat{g}(\bar{i})\} \neq \emptyset$ if $\hat{g}(\underline{i}) \in \tilde{G}(\underline{i})$ and $\hat{g}(\bar{i}) \in \tilde{G}(\bar{i})$. So, an optimum at the midpoint is found as long as $\hat{g}(\underline{i}) \in \tilde{G}(\underline{i})$ and $\hat{g}(\bar{i}) \in \tilde{G}(\bar{i})$. Clearly, this is true at the first instance of step 2. Since in step 2(c) the algorithm then divides into $\{\underline{i}, \dots, m\}$ and $\{m, \dots, \bar{i}\}$, this is also true at the next iteration. Consequently, induction gives that a maximum is found for every midpoint. So, if every $i \in \{2, \dots, n-1\}$ is the midpoint of some $(\underline{i}, \bar{i})$ in some iteration, the proof is complete.

To show that every $i \in \{2, \dots, n-1\}$ is a midpoint of some interval reached in the recursion, fix an arbitrary such i and suppose not. Define $(\underline{i}_1, \bar{i}_1) = (1, n)$. When step 2 is first reached, $i \in \{\underline{i}_1 + 1, \dots, \bar{i}_1 - 1\}$. At every step 2, uniquely and recursively define $(\underline{i}_k, \bar{i}_k)$ to be the one of $(\underline{i}_{k-1}, m)$ and $(m, \bar{i}_{k-1})$ with $m = \lfloor (\underline{i}_{k-1} + \bar{i}_{k-1})/2 \rfloor$ such that $i \in \{\underline{i}_k + 1, \dots, \bar{i}_k - 1\}$ (because i is assumed to never be a midpoint, this is well-defined).

Now, consider the cardinality of $\{\underline{i}_k, \dots, \bar{i}_k\}$ defining it as $n_k = \bar{i}_k - \underline{i}_k + 1$. By construction, $i \in \{\underline{i}_k + 1, \dots, \bar{i}_k - 1\}$ for each k . So, a contradiction is reached if $\{\underline{i}_k + 1, \dots, \bar{i}_k - 1\} = \emptyset$ which is equivalent to $N_k \leq 2$. So, it must be that $N_k \geq 3$ for all N_k . If N_{k-1} is odd, then $N_k = (N_{k-1} + 1)/2$. If N_{k-1} is even, $N_k \leq N_{k-1}/2 + 1$. Defining M_k recursively by $M_1 = N_1$ and $M_k = M_{k-1}/2 + 1$, one can show by induction that $N_k \leq M_k$ for all k . Because $N_k \geq 3$ for all k , $M_k \geq 3$ for all k . Hence $M_k - M_{k-1} = 1 - M_{k-1}/2 \leq 1 - 3/2 = -1/2$. Hence $M_k \leq M_{k-1} - 1/2$. Therefore, M_k will be less than three in a finite number of iterations, which gives a contradiction.

□

C Calibration [Not For Publication]

For the RBC model, we use the following calibration. We assume leisure is not valued with $u(c, l) = c^{1-\sigma}/(1-\sigma)$ and $\sigma = 2$. Consequently, (5) reduces to $U(k, k', z) = -1/(-k' + zF(k', 1) + (1-\delta)k)$. We take $\beta = .99$, $\delta = .025$, and $F(k, l) = k^{.36}$. We discretize the TFP process $\log z = .95 \log z_{-1} + .007\varepsilon$ with $\varepsilon \sim N(0, 1)$ using Tauchen (1986)'s method with 21 points spaced evenly over ± 3 unconditional standard deviations. The capital grid is linearly spaced over $\pm 20\%$ of the steady state capital stock.

For the Arellano (2008) model, we adopt the same calibration as the original Arellano (2008) paper and refer the reader to that paper for details. For n bond grid points, 70% are linearly spaced from $-.35$ to 0 and the rest from 0 to $.15$. Again using Tauchen (1986)'s method, we discretize the exogenous output process $\log y = .945 \log y + .025\varepsilon$ with $\varepsilon \sim N(0, 1)$ using 21 points spaced evenly over ± 3 unconditional standard deviations.