**Published in Journal of Economic Theory
Volume 135, Issue 1, July 2007, Pages 514-532.**

FINDING ALL EQUILIBRIA IN GAMES OF STRATEGIC COMPLEMENTS

FEDERICO ECHENIQUE

Address:
Division of the Humanities and Social Sciences
California Institute of Technology,
Pasadena CA 91125,
USA
Fax: 1 - 626 793 8580
`email: fede@caltech.edu`
`http://www.hss.caltech.edu/~fede`

JEL Classification: C63, C72
Running Title: Finding All Equilibria.

I am grateful to an associate editor and two referees for their comments. I thank Gerard Cachon, Eddie Dekel, Juan Dubra, Matt Jackson, Ivana Komunjer, Andy McLennan, John Rust, Bill Sandholm, Ilya Segal, Chris Shannon, and Bernhard von Stengel for comments and suggestions. I also thank seminar audiences in a number of institutions for their comments.

Abstract. I present a simple and fast algorithm that finds all the pure-strategy Nash equilibria in games with strategic complementarities. This is the first non-trivial algorithm for finding all pure-strategy Nash equilibria.

Keywords: Computation of equilibrium, Supermodular Games, Algorithms for finding Nash equilibria.

1. Introduction

I present an algorithm that finds all the pure-strategy equilibria in n-player games with strategic complementarities (GSC). This is the first non-trivial algorithm for finding all equilibria in some general class of games.

GSC were first formalized by Topkis [20], and were introduced by Vives [22] to economics. GSC are important in many areas in economics. For example, both price- and quantity-competition oligopoly models can be modeled as GSC, arguably covering most static market models one may wish to consider. See Milgrom and Roberts [14], Milgrom and Shannon [15], Topkis [21], and Vives [23] for economic examples of GSC.

Many models in operations research have recently been analyzed as GSC. Examples are Lippman and McCardle [12], Bernstein and Federgruen [3], Bernstein and Federgruen [4], Netessine and Rudi [17], Netessine and Shumsky [18], Cachon and Lariviere [6], and Cachon [5]. See Cachon and Netessine [7] for a survey.

I wish to emphasize three features of the algorithm:

(1) It finds all pure-strategy equilibria, but no mixed-strategy equilibria. The omission is justified because mixed-strategy equilibria are not good predictions in GSC [10]. Further, pure-strategy equilibria always exist in GSC [20, 22].
(2) It is generally very fast. For example, it needs less than 10 seconds to find all equilibria in a game with 3.6×10^9 strategy profiles.
(3) It is simple. I use the algorithm "by hand" on some bimatrix games to show that the algorithm is very simple to apply.

There are many algorithms for finding *one* equilibrium, called a "sample" equilibrium (see the surveys by McKelvey and McLennan [13] and von Stengel [24]). But there is currently only one method for finding *all* pure equilibria: the "underlining" method one teaches undergraduates—fix one player, for each strategy-profile of the player's opponents, find her best-response, and then check if some opponent wants to deviate. The method is close to testing all the game's strategy-profiles to see if they are equilibria; I shall call this method the "trivial algorithm." Not surprisingly, the trivial algorithm is typically very slow, and, in practice, useless for large games.

Some algorithms find a sample equilibrium that survives an equilibrium refinement—typically perfection (a recent example is von Stengel, van den Elzen, and Talman [25]; see McKelvey and McLennan [13] and von Stengel [24] for other examples). This is some times adequate, but it is in general restrictive: there is normally no guarantee that only one equilibrium survives the refinement, and the refinements do not always have bite. (An exception is Judd, Yeltekin, and Conklin [11]; their algorithm finds all perfect-equilibrium payoffs in repeated games.)

The algorithm I present is based on Topkis's (1979) results that Robinson's (1951) method of "iterating best-responses" finds an equilibrium in GSC (see also

Vives [22]), so the algorithm uses different—and simpler—ideas than the more recent literature on finding equilibria. Topkis shows that GSC possess "extremal" (smallest and largest) pure-strategy equilibria, and that iterating best-responses results in monotone sequences that approach an extremal equilibrium. My algorithm is to, essentially, restart the iterations after the extremal equilibria have been found, and force new monotone sequences that find additional equilibria. The iterations restart by removing certain strategies from the game in a way that ensures that all equilibria will be found.

I shall not apply the algorithm to economic, or operations research, examples. The paper presents a method, and it argues that the method works well. So the chosen applications either illustrate how the algorithm works, or show that it is fast. Nevertheless, there are many applications in operations research and economics. I give two examples:

- Supply-chain analysis. Cachon [5] studies inventory competition in a supply chain with retailers that face stochastic demands. The resulting game is a GSC. Cachon compares numerically the system-optimal solution to the Nash equilibria of the game. He uses exhaustive search (after identifying the extremal equilibria—see Section 3) to find all Nash equilibria. The algorithm I introduce can be used instead of exhaustive search; it will be more efficient. [1]
- Oligopoly models. Under mild conditions, Bertrand oligopoly with differentiated products is a GSC [16]. In turn, Bertrand oligopoly with differentiated products is a very common market structure. The algorithm has then natural applications in the empirical analysis of markets.

  One important example is the evaluation of mergers by the US Department of Justice. The Department of Justice needs to predict the consequences of mergers between firms. They postulate a model of a market—they often use Bertrand models with differentiated products, see for example Werden, Froeb, and Tschantz [26] or Crooke, Froeb, Tschantz, and Werden [8]—-and compute a Nash equilibrium before and after the merger of some firms in the market. [2]

  But their conclusions might of course change if they could find all equilibria before and after the merger. For example, the merger could have no effect on price if one looks at some equilibria, but a large price increase if you compare most equilibria.

  Besides Bertrand oligopoly, the algorithm can also be applied to Cournot oligopoly. By exploiting Amir and Lambson's (2000) ideas for using GSC-techniques in Cournot oligopoly, one can easily adapt the algorithm to find

[1] Other applications to supply-chain analysis include discretized versions of Lippman and McCardle [12].

[2] The software they use is in `http://mba.vanderbildt.edu/luke.froeb/software/`

all symmetric equilibria in Cournot models as well. Arguably, the algorithm is applicable to most static models of a market one may wish to consider in applied work.

The paper is organized as follows. Section 2 presents some preliminary definitions and results. Section 3 shows informally how the algorithm works. Section 4 defines the algorithm and presents the main results of the paper. Section 5 develops two simple examples. Section 6 outlines the argument that the algorithm is typically fast, and explains the benchmark algorithm. Section 7 presents computational results for simulations of GSC. Section 8 discusses issues related to the speed of the algorithm. Section 9 discusses an algorithm for a special class of GSC.

2. Preliminary Definitions and Results

2.1. **Basic Definitions and Notation.** Let $X \subseteq \mathbf{R}^n$, and $x, y \in \mathbf{R}^n$. Denote the vector $(\max\{x_i, y_i\})$ by $x \vee y$, and the vector $(\min\{x_i, y_i\})$ by $x \wedge y$. Say that X is a *lattice* if, whenever $x, y \in X$, $x \wedge y, x \vee y \in X$.

If X is a lattice, a function $f : X \to \mathbf{R}$ is *quasi-supermodular* if for any $x, y \in X$, $f(x) \geq f(x \wedge y)$ implies $f(x \vee y) \geq f(y)$ and $f(x) > f(x \wedge y)$ implies $f(x \vee y) > f(y)$. Quasi-supermodularity is an ordinal notion of complementarities; it was introduced by Milgrom and Shannon [16]. Let $T \subseteq \mathbf{R}^m$. A function $f : X \times T \to \mathbf{R}$ satisfies the *single-crossing condition* in (x, t) if whenever $x < x'$ and $t < t'$, $f(x, t) \leq f(x', t)$ implies that $f(x, t') \leq f(x', t')$ and $f(x, t) < f(x', t)$ implies that $f(x, t') < f(x', t')$.

For two subsets A, B of X, say that A is smaller than B in the *strong set order* if $a \in A$, $b \in B$ implies $a \wedge b \in A, a \vee b \in B$. Let $\phi : X \twoheadrightarrow X$ be a correspondence. Say that ϕ is *increasing in the strong set order* if, whenever $x \leq y$, $\phi(x)$ is smaller in the strong set order than $\phi(y)$. A detailed discussion of these concepts is in Topkis [21].

An n-player normal-form game (a game, for short) is a collection $\Gamma = \{(S_i, u_i) : i = 1, \dots n\}$, where each player i is characterized by a set of possible strategies, S_i, and a payoff function $u_i : S \to \mathbf{R}$, where $S = \times_{j=1}^n S_j$. Say that players have *strict preferences* if, for all i and $s_{-i} \in S_{-i}$, the function $s_i \mapsto u_i(s_i, s_{-i})$ is one-to-one.

For each player i, let $\beta_{i,\Gamma}$ denote i's *best-response correspondence* in Γ—the correspondence defined by

$$\beta_{i,\Gamma}(s) = \mathrm{argmax}_{\tilde{s}_i \in S_i} u_i(\tilde{s}_i, s_{-i}).$$

And let $\beta_\Gamma(s) = \times_{i=1}^n \beta_{i,\Gamma}(s)$ denote the game's best-response correspondence. When Γ is understood I shall write β_i for $\beta_{i,\Gamma}$ and β for β_Γ.

A point $s \in S$ is a *Nash equilibrium* if $s \in \beta(s)$. Let $\mathcal{E}(\Gamma)$ be the set of all Nash equilibria of Γ. When Γ is understood, I shall write $\mathcal{E}$ for $\mathcal{E}(\Gamma)$.

2.2. **The Model.** Say that a game $\Gamma = \{(S_i, u_i) : i = 1, \dots n\}$ is a *finite game of strategic complementarities* (GSC) if, for each i,

- $S_i \subseteq \mathbf{R}^{d_i}$ is a finite lattice,
- $s_i \mapsto u_i(s_i, s_{-i})$ is quasi-supermodular for all s_{-i},
- and $(s_i, s_{-i}) \mapsto u_i(s_i, s_{-i})$ satisfies the single-crossing property.

The positive integer d_i is the number of dimensions of player i's strategies. I shall assume, in addition, that

- $S_i = \{1, 2, \dots K_i\}^{d_i}$.

The assumption that $S_i = \{1, 2, \dots K_i\}^{d_i}$ simplifies notation, but I should stress that all my results hold for arbitrary finite GSC.

The set of Nash equilibria of a GSC is a complete lattice [27].

Remark 1. One can think of the model as a discretized version of a game with continuous strategy spaces, where each S_i is an interval in some Euclidean space of dimension d_i. For an example, see Section 7.

2.3. **Auxiliary results.** First, GSC have monotone best-response correspondences:

Lemma 2. *[**16**] For all i, β_i is increasing in the strong set order, and* $\inf \beta_i(s), \sup \beta_i(s) \in \beta_i(s)$.

See Milgrom and Shannon [16] for a proof.

Second, I need some results and notation for games where we restrict the strategies that players can choose: For each $s_i \in S_i$, let $S_i^r(s_i) = \{\tilde{s}_i \in S_i : s_i \leq \tilde{s}_i\}$ be the strategy space obtained by letting i choose any strategy in S_i, as long as it is larger than s_i. Note that I use "larger than" short for "larger than or equal to." For each strategy profile $s = (s_1, \dots s_n) \in S$, let $S^r(s) = \times_{i=1}^n S_i^r(s_i)$. Denote by $\Gamma^r(s)$ the game where each player i is constrained to choosing a strategy larger than s_i. Then,

$$\Gamma^r(s_1, \dots s_n) = \left\{(S_i^r(s_i), u_i|_{S_i^r(s_i)}) : i = 1, \dots n\right\}.$$

The following lemmas are trivial.

Lemma 3. *If Γ is a GSC, then so is $\Gamma^r(s)$, for any strategy profile $s \in S$.*

Lemma 4. *If s is a Nash equilibrium of Γ, and $z \leq s$, then s is a Nash equilibrium of $\Gamma^r(z)$.*

Lemma 3 and Lemma 4 follow immediately from the definitions of GSC and of Nash equilibrium.

Third, I shall exploit some previous results on finding equilibria in GSC. The method of iterating β until an equilibrium is found is normally attributed to Robinson [19]. Topkis [20] proved that the method works in GSC. I call this method the "Robinson-Topkis algorithm," or RT algorithm.

Algorithm 1. *The following are three variants of the RT algorithm.*

- $\underline{T}(s)$*: Start with* $s^0 = s$*. Given* $s^k \in S$*, let* $s^{k+1} = \inf \beta_\Gamma(s^k)$*. Stop when* $s^k = s^{k+1}$*.*
- $\overline{T}(s)$*: Start with* $s^0 = s$*. Given* $s^k \in S$*, let* $s^{k+1} = \sup \beta_\Gamma(s^k)$*. Stop when* $s^k = s^{k+1}$*.*
- $T^r(s)$*: Do algorithm* $\underline{T}(s)$ *in* $\Gamma^r(s)$*.*

Lemma 5. $\underline{T}(\inf S)$ *stops at the smallest Nash equilibrium of* Γ*, and* $\overline{T}(\sup S)$ *stops at the largest Nash equilibrium of* Γ*.*

Topkis [20] proved Lemma 5 for supermodular games. For the extension to GSC, see Milgrom and Shannon [16] or Topkis [21].

Remark 6. Note that $\underline{T}(\inf S)$ is faster than "iterating $\inf \beta_\Gamma(s^k)$" suggests. When the algorithm has to find $\inf \beta_\Gamma(s^k)$, it knows that searching in the interval $[s_k, \sup S]$ is enough. The sequence $\{s_k\}$ is monotone increasing, so each iteration of $\underline{T}(\inf S)$ is faster than the previous iteration. A similar thing happens to $\overline{T}(s)$ and $T^r(s)$.

A "round-robin" version of RT—where players take turns in best-responding instead of jointly best-responding in each iteration—is faster than the version above (see Topkis [20]). All results in the paper hold if "round-robin" RT is used. The results reported in Section 7 use round-robin RT.

3. How it works

"In the authors' experience, an important idea in organizing the analysis of a game by hand is to find one equilibrium, then ask how other equilibria might differ from this one; there is currently no substantiation of this wisdom in theory or computational experience." [13, p. 28]

I use an example to explain how the algorithm works; the algorithm is one possible substantiation of McKelvey and McLennan's wisdom.

Consider a two-player GSC, Γ. Suppose that player 1 has strategy set $S_1 = \{1, 2, \ldots 15\}$, and player 2 has $S_2 = \{1, 2, \ldots 11\}$. The players' joint strategy space, $S_1 \times S_2$, is in Figure 1. Suppose that we have calculated the players' best-response functions—assume best-responses are everywhere unique—β_1 and β_2. The game's best-response function is $\beta(s_1, s_2) = (\beta_1(s_2), \beta_2(s_1))$. Because Γ is a GSC, β_1, β_2 and β are monotone increasing functions (Lemma 2).

First we need to understand how the RT algorithm works. RT starts at the smallest strategy profile, $(1, 1)$, and iterates the game's best-response function until two iterations are the same. Since $(1, 1)$ is smaller than $\beta(1, 1)$, and β is monotone, we have that $\beta(1, 1)$ is smaller than $\beta(\beta(1, 1)) = \beta^2(1, 1)$; $\beta^2(1, 1)$ is smaller than $\beta^3(1, 1)$, and so on—iterating β we get a monotone increasing sequence in S.

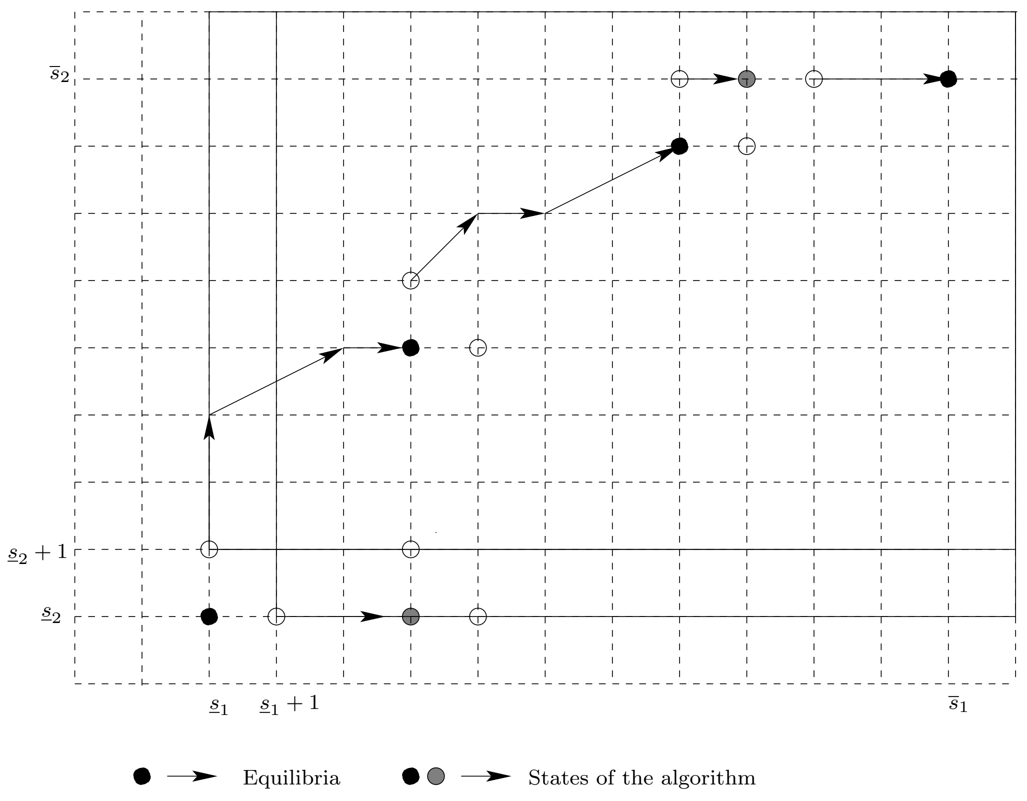

FIGURE 1. The algorithm in a two-player game.

Now, S is finite, so there must be an iteration k such that $\beta^k(1,1) = \beta^{k-1}(1,1)$. But then of course $\beta^k(1,1) = \beta(\beta^{k-1}(1,1))$, so $\underline{s} = \beta^{k-1}(1,1)$ is a Nash equilibrium.

It turns out that $\underline{s}$ is the *smallest* Nash equilibrium in Γ: Let s^* be any other equilibrium, and note that $(1,1) \leq s^*$. Monotonicity of β implies that $\beta(1,1) \leq \beta(s^*) = s^*$. Then, iterating β we get

$$\underline{s} = \beta^{k-1}(1,1) \leq \beta^{k-1}(s^*) = s^*.$$

In a similar way, RT finds the game's largest Nash equilibrium $\overline{s}$ by iterating the game's best-response function starting from the largest strategy profile, $(15,11)$.

I now describe my algorithm informally. A general description and proof is in Section 4.

The algorithm consists of the following steps:

(1) Find the smallest ($\underline{s}$) and largest ($\overline{s}$) Nash equilibrium using RT—note $\underline{s}$ and $\overline{s}$ in Figure 1.
(2) Consider $\Gamma^r(\underline{s}_1, \underline{s}_2+1)$, the game where player 1 is restricted to choosing a strategy larger than $\underline{s}_1$, and player 2 is restricted to choosing a strategy

larger than $\underline{s}_2+1$. The strategy profile $(\underline{s}_1, \underline{s}_2+1)$ is indicated in the figure with a circle ◯ above $(\underline{s}_1, \underline{s}_2)$, and the strategy space in $\Gamma^r(\underline{s}_1, \underline{s}_2+1)$ is the interval $[(\underline{s}_1, \underline{s}_2+1), (15, 11)]$ shown with non-dotted lines in the figure. Now use RT to find s^1, the smallest Nash equilibrium in $\Gamma^r(\underline{s}_1, \underline{s}_2+1)$. Each iteration of β is shown with an arrow in the figure, and s^1 is the black disk reached after three iterations.

Similarly, consider $\Gamma^r(\underline{s}_1+1, \underline{s}_2)$, the game where player 1 is restricted to choosing a strategy larger than $\underline{s}_1+1$, and player 2 is restricted to choosing a strategy larger than $\underline{s}_2$. The strategy profile $(\underline{s}_1+1, \underline{s}_2)$ is indicated in the figure with a circle ◯ to the right of $(\underline{s}_1, \underline{s}_2)$. Use RT to find s^2, the smallest Nash equilibrium in $\Gamma^r(\underline{s}_1+1, \underline{s}_2)$; s^2 should be a black disk at this point, I explain in the next step why it is gray.

(3) Check if s^1 and s^2 are Nash equilibria of Γ. First consider s^1. Because s^1 is an equilibrium of $\Gamma^r(\underline{s}_1, \underline{s}_2+1)$, and β is monotone increasing, we *only* need to check that $\underline{s}_2$ is not a profitable deviation for player 2. Similarly, to check if s^2 is an equilibrium we only need to check that $\underline{s}_1$ is not a profitable deviation for player 1; right after step (5) I explain why these checks are sufficient. Let us assume that s^1 passes the check while s^2 fails, this is indicated in the figure by drawing s^2 as a gray circle.

(4) Do steps 2 and 3 for $\Gamma^r(s^1_1, s^1_2+1)$, $\Gamma^r(s^1_1+1, s^1_2)$, $\Gamma^r(s^2_1, s^2_2+1)$, and $\Gamma^r(s^2_1+1, s^2_2)$.

(5) Generally, continue repeating steps 2 and 3 for each potential Nash equilibrium s^k found, unless s^k is equal to $\overline{s}$. The picture shows what the algorithm does for a selection of s^ks; note that the algorithm starts at larger and larger ◯-circles, and that it approaches $\overline{s}$.

I phrased item 3—the "check"-phase—in terms of the first iteration of the algorithm. More generally, let s^k be a candidate equilibrium obtained as the smallest equilibrium in some $\Gamma^r(\hat{s}+(1,0))$. To check if s^k is an equilibrium we need to see if s^k_i equals $\beta_i(s^k)$. Now, $\beta_i(\hat{s}) \leq \beta_i(s^k)$, and we know that there is no better response to s^k than s^k_i, among strategies in game $\Gamma^r(\hat{s}+(1,0))$.

So we only need to check if there is some strategy in the interval $[\beta_i(\hat{s}), \hat{s}_i]$ that is better than s^k_i against s^k. In the example, $\hat{s}$ is an equilibrium, so $[\beta_i(\hat{s}), \hat{s}_i] = \{\hat{s}_i\}$, and we only need to check if $\hat{s}_i$ is a profitable deviation. More generally, though, the algorithm will need to retain the value of $\beta_i(\hat{s})$ in order to make subsequent checks.

I now explain why the algorithm finds all the Nash equilibria of Γ. Suppose that s is an equilibrium, so $\underline{s} \leq s \leq \overline{s}$. If $s = \underline{s}$ or $s = \overline{s}$, then the algorithm finds s in step 1. Suppose that $\underline{s} < s < \overline{s}$, then either $(\underline{s}_1, \underline{s}_2+1) \leq s$ or $(\underline{s}_1+1, \underline{s}_2) \leq s$ (or both). Suppose that $(\underline{s}_1, \underline{s}_2+1) \leq s$, so s is a strategy in $\Gamma^r(\underline{s}_1, \underline{s}_2+1)$. Note that s is also an equilibrium of $\Gamma^r(\underline{s}_1, \underline{s}_2+1)$: if a player i does not want to deviate from s when allowed to choose any strategy in S_i, she will not want to deviate when only allowed to choose the subset of strategies in $\Gamma^r(\underline{s}_1, \underline{s}_2+1)$. The algorithm finds $\tilde{s}^1$, the smallest equilibrium in $\Gamma^r(\underline{s}_1, \underline{s}_2+1)$—so $\tilde{s}^1 \leq s$. If $\tilde{s}^1 = s$ the algorithm has found s. If $\tilde{s}^1 < s$ then either $(\tilde{s}^1_1, \tilde{s}^1_2+1) \leq s$ or $(\tilde{s}^1_1+1, \tilde{s}^1_2) \leq s$ (or both). Suppose that $(\tilde{s}^1_1, \tilde{s}^1_2+1) \leq s$, then repeating the argument above we will arrive at a new $\tilde{s}^2 \leq s$. There will be other strategies found by the algorithm starting from other points, but we can always find one that is smaller than s. Generally, for each $\tilde{s}^{k-1}$ we have found, we find the smallest equilibrium $\tilde{s}^k$ of $\Gamma^r(\tilde{s}^{k-1}_1+1, \tilde{s}^{k-1}_2)$ or $\Gamma^r(\tilde{s}^{k-1}_1, \tilde{s}^{k-1}_2+1)$, whichever contains s, and it must satisfy $\tilde{s}^{k-1} < \tilde{s}^k \leq s$; it satisfies $\tilde{s}^k \leq s$ because $\tilde{s}^k$ is the smallest equilibrium in the game that contains s, and it satisfies $\tilde{s}^{k-1} < \tilde{s}^k$ by definition of the Γ^r games. The sequence of strictly increasing $\tilde{s}^k$s would eventually reach $\overline{s}$, so $s < \overline{s}$ implies that there must be a $\tilde{s}^k = s$. Since s is an equilibrium, $\tilde{s}^k = s$ passes the test in item 3; hence the algorithm finds s.

4. The Algorithm

Let $e^{d_i}_l$ be the l-th unit vector in $\mathbf{R}^{d_i}$, i.e. $e^{d_i}_l = (0, \ldots 1, 0 \ldots 0) \in \mathbf{R}^{d_i}$, where 1 is the l-th element of $e^{d_i}_l$.

Algorithm 2. *Find $\underline{s} = \inf \mathcal{E}$ using $\underline{T}(\inf S)$, and $\overline{s} = \sup \mathcal{E}$ using $\overline{T}(\sup S)$. Let $\hat{\mathcal{E}} = \{\underline{s}, \overline{s}\}$. The set of possible states of the algorithm is $2^{S \times S}$; start at state $\{(\underline{s}, \underline{s})\}$.*

Let the state of the algorithm be $\mathcal{M}$. While $\mathcal{M} \neq \{(\overline{s}, \inf \beta_\Gamma(\overline{s}))\}$, repeat the following sub-routine to obtain a new state $\mathcal{M}'$.

Subroutine *Let $\mathcal{M}' = \emptyset$. For each $(s, s^*) \in \mathcal{M}$, $i \in \{1, \ldots n\}$ and l with $1 \leq l \leq d_i$, if $(s_i + e^{d_i}_l, s_{-i}) \leq \overline{s}$, then do steps 1-3:*

1. *Run $T^r(s_i + e^{d_i}_l, s_{-i})$; let $\hat{s}$ be the strategy profile at which it stops.*
2. *Check that no player j wants to deviate from $\hat{s}_j$ to a strategy in the set*
$$\left\{ z \in S_j : s^*_j \leq z \text{ and } (s_i + e^{d_i}_l, s_{-i})_j \not\leq z \right\}.$$
If no player wants to deviate, add $\hat{s}$ to $\hat{\mathcal{E}}$.[3]
3. *Add $(\hat{s}, \inf \beta_\Gamma(\hat{s}))$ to $\mathcal{M}'$.*

Theorem 7. *The set $\hat{\mathcal{E}}$ produced by Algorithm 2 coincides with the set $\mathcal{E}$ of Nash equilibria of Γ.*

Proof. First I shall prove that the algorithm stops after a finite number of iterations, and that it stops when $\mathcal{M} = \{(\overline{s}, \inf \beta_\Gamma(\overline{s}))\}$, not before (step "well-behaved"). Then I shall prove that $\hat{\mathcal{E}} \subseteq \mathcal{E}$, and then that $\mathcal{E} \subseteq \hat{\mathcal{E}}$.

[3]This check, an improvement over a previous version of Algorithm 2, was suggested by a referee.

Step "well-behaved." Let $M \subseteq 2^{S\times S}$ be the collection of states visited by Algorithm 2. For $\mathcal{M}, \mathcal{M}' \in M$, say that $\mathcal{M} \nearrow \mathcal{M}'$ if $\mathcal{M}'$ was obtained from state $\mathcal{M}$ in an application of the subroutine. Let $C : M \to \mathbf{R}$ be the function

$$C(\mathcal{M}) = \min_{(s,s^*)\in\mathcal{M}} \sum_{i=1}^{n} \sum_{l=1}^{d_i} s_{il}.$$

First note that $\mathcal{M} \nearrow \mathcal{M}'$ implies that for each $(s, s^*) \in \mathcal{M}'$ there is $(\hat{s}, \hat{s}^*) \in \mathcal{M}$ with $(\hat{s}_i + e^{d_i}_l, \hat{s}_{-i}) \leq s$ for some i and l, namely the $\hat{s} \in \mathcal{M}$, i and l from which s was found in the subroutine. Hence $\mathcal{M} \nearrow \mathcal{M}'$ implies that $C(\mathcal{M}) < C(\mathcal{M}')$, so that $\nearrow$ is a transitive binary relation.

By the statement of Algorithm 2, for each $\mathcal{M} \in M$, $\mathcal{M} \neq \{(\bar{s}, \inf \beta_\Gamma(\bar{s}))\}$, there is a unique $\mathcal{M}'$ such that $\mathcal{M} \nearrow \mathcal{M}'$. Hence $\nearrow$ is a total order on M, and C is strictly monotonically increasing with respect to $\nearrow$.

Now, since C can take a finite number of values (as strategy sets are finite) M must be a finite set. And $\{(\bar{s}, \inf \beta_\Gamma(\bar{s}))\}$ must be the largest element in M according to $\nearrow$, as for each $\mathcal{M} \in M$, $\mathcal{M} \neq \{(\bar{s}, \inf \beta_\Gamma(\bar{s}))\}$, there is $\mathcal{M}'$ such that $\mathcal{M} \nearrow \mathcal{M}'$. Thus the algorithm stops after a finite number of steps, and only when it reaches state $\{(\bar{s}, \inf \beta_\Gamma(\bar{s}))\}$.

Step $\hat{\mathcal{E}} \subseteq \mathcal{E}$. Let $\hat{s} \in \hat{\mathcal{E}}$; we shall prove $\hat{s} \in \mathcal{E}$. Suppose $\hat{s} \neq \underline{s}$, or there is nothing to prove. Then $\hat{s}$ was obtained by $T^r(s_i + e^{d_i}_l, s_{-i})$, for some i, l, and $(s, s^*) \in \mathcal{M}$, for some state $\mathcal{M}$ of the algorithm.

Fix a player j. We have $s \leq \hat{s}$, so $\beta_{j,\Gamma}(s_{-j})$ is smaller in the strong set order than $\beta_{j,\Gamma}(\hat{s}_{-j})$ (Lemma 2); hence $s^*_i \leq z$, for all $z \in \beta_{j,\Gamma}(\hat{s}_{-j})$. Since $\hat{s}$ survives step (3) in the algorithm, and $\hat{s}$ is a Nash equilibrium of $\Gamma^r(s_i + e^{d_i}_l, s_{-i})$, $u_j(z, \hat{s}_{-j}) \leq u_j(\hat{s})$ for all $z \in S_j$ such that $s^*_j \leq z$. In particular, this is true for all $z \in \beta_{j,\Gamma}(\hat{s}_{-j})$. Then $\hat{s} \in \mathcal{E}$.

Step $\mathcal{E} \subseteq \hat{\mathcal{E}}$. Let $s \in \mathcal{E}$. Suppose, by way of contradiction, that $s \notin \hat{\mathcal{E}}$. For each $\mathcal{M}$, let $\mathcal{M}_1$ be the set of s such that $(s, s^*) \in \mathcal{M}$ for some s^*.

Claim: Let Algorithm 2 transit from state $\mathcal{M}$ to state $\mathcal{M}'$. If there is $z \in \mathcal{M}_1$ with $z < s$ then there is $z' \in \mathcal{M}'_1$ with $z' < s$.

Proof of the claim: Since $z < s$, there is i and l such that $z_{il} < s_{il}$. Then s is a strategy profile in $\Gamma^r(z_i + e^{d_i}_l, z_{-i})$. If $\hat{s}$ is the strategy profile found by $T^r(z_i + e^{d_i}_l, z_{-i})$, then Lemma 5 implies that $\hat{s} \leq s$, as s is a Nash equilibrium of $\Gamma^r(z_i + e^{d_i}_l, z_{-i})$. If $\hat{s} = s$ then s would pass the test of step 4 and be added to $\hat{\mathcal{E}}$, but we assumed $s \notin \hat{\mathcal{E}}$ so it must be that $\hat{s} < s$. Set $z' = \hat{s}$, then $z' \in \mathcal{M}'_1$ by step 3, and the proof of the claim is complete.

Now, $s \notin \hat{\mathcal{E}}$ implies that $s \neq \underline{s}$. Initially $\mathcal{M} = \{(\underline{s}, \inf \beta_\Gamma(\underline{s}))\}$ so there is z in $\mathcal{M}_1$ with $z_1 < s$. Using the Claim above inductively, it must be that all stages of Algorithm 2 contain a z with $z < s$. But the final state of the algorithm is $\mathcal{M} = \{(\bar{s}, \inf \beta_\Gamma(\bar{s}))\}$; a contradiction, since $s \leq \bar{s}$. $\square$

The following will make Algorithm 2 faster: Only do step 2 of the subroutine if there is no $s' \in \hat{\mathcal{E}}$ such that $\hat{s} \leq s'$, and $\hat{s} \in S(s_i + e_l^{d_i}, s_{-i})$, for the s, i and l at which s' was found. For, if there is such an s', then we know that $\hat{s} \notin \mathcal{E}$, as $\hat{s} \in \mathcal{E}$ would imply that $\hat{s}$ is an equilibrium of $\Gamma^r(s_i + e_l^{d_i}, s_{-i})$, which contradicts that s' is the smallest equilibrium of $\Gamma^r(s_i + e_l^{d_i}, s_{-i})$.

Theorem 7 says that Algorithm 2 works. In the rest of the paper I show that it is generally fast.

5. Example

The following example shows that the algorithm can be slow; I then argue that it will generally be very fast.

Consider the two-player game in Figure 2. Both players have identical strategy sets, $\{1, 2, 3, 4\}$. The game is a game with strategic complementarities.

	1	2	3	4
4	0, 0	0, 0	0, 0	0, 0
3	1, 3	1, 2	1, 1	0, 0
2	2, 3	2, 2	2, 1	0, 0
1	3, 3	3, 2	3, 1	0, 0

FIGURE 2. A slow example.

RT yields $(1, 1)$ as the smallest equilibrium, and $(4, 4)$ as the largest equilibrium. The initial state of the algorithm is thus $\{(1, 1)\}$. We start the subroutine at $(2, 1) = (1, 1) + (1, 0)$ and get back $(2, 1)$ as the smallest equilibrium of $\Gamma^r(2, 1)$. But player 1 prefers strategy 1 over strategy 2, so $(2, 1)$ does not survive step 2. We start the subroutine at $(1, 2) = (1, 1) + (0, 1)$ and get back $(1, 2)$ as the smallest equilibrium of $\Gamma^r(1, 2)$. But player 1 prefers strategy 1 over strategy 2, so $(1, 2)$ does not survive step 2.

If one completes all iterations (shown in Table I) it is clear that the algorithm stops at all strategy profiles, and discards all profiles but the largest and the smallest equilibria of the game.

The example presents a pathological situation; the algorithm is forced to check all strategy profiles of the game. The root of the problem is that, after each iteration, it is optimal for the players to choose their smallest allowed strategies. I argue in Section 6 that the situation will typically not occur in the games that one encounters in applications.

Table I. Iterations in Example 2

	$\mathcal{M}$	$\hat{\mathcal{E}}$
1	$\{(1,1)\}$	$\{(1,1),(4,4)\}$
2	$\{(2,1),(1,2)\}$	$\{(1,1),(4,4)\}$
3	$\{(3,1),(2,2),(1,3)\}$	$\{(1,1),(4,4)\}$
4	$\{(4,1),(3,2),(2,3),(1,4)\}$	$\{(1,1),(4,4)\}$
5	$\{(4,2),(3,3),(2,4)\}$	$\{(1,1),(4,4)\}$
6	$\{(4,3),(3,4)\}$	$\{(1,1),(4,4)\}$
7	$\{(4,4)\}$	$\{(1,1),(4,4)\}$

6. How fast is Algorithm 2?

6.1. **Outline.** The rest of the paper establishes that Algorithm 2 is generally very fast.[4] Here is an outline:

- Section 7. I simulate a large class of games, and show that Algorithm 2 finds all equilibria very quickly. The class of games was chosen trying to bias the test against Algorithm 2.
- Section 8. I show that Algorithm 2 is faster when best-responses in each iteration have large increases—I then argue that this will occur for many natural applications of the algorithm.
- Section 9. I present a version of the algorithm that applies to two-player games with strict preferences. This version is faster than Algorithm 2.

In the worst case, Algorithm 2 is slow (see Section 5). But the worst case is—not surprisingly—usually irrelevant for actual applications.

I identify the performance of Algorithm 2 with its running-time. In practice, though, the data the algorithm needs to store is also an important consideration, as the need to store large amounts of data may affect the algorithm's performance. One source of storage problems is the input data, i.e. the players' payoffs. But in economic applications these are usually given by a parametric payoff function, not by a payoff matrix. So this source of storage problems is probably not important for users of Algorithm 2. A more serious problem is the need to store large states, see Section 7.3 on how the implementation I wrote handles the problem.

6.2. **The Benchmark.** I now describe the trivial algorithm, the benchmark against which Algorithm 2 is compared in terms of speed.

Let $\Gamma = \{(S_i, u_i) : i = 1, \dots n\}$ be an n-player game. Fix a player, say i. First, for each s_{-i}, find the set of best-responses by i. Second, check if any player

[4] I mean speed in a loose sense, not in terms of polynomial vs. exponential time.

$j \neq i$ wants to deviate from her strategy in s_{-i} when i chooses one of her best-responses. This algorithm is essentially the "underlining" method for finding the Nash equilibria that one teaches first-year students.

Suppose that all players have K strategies, and that best-responses are everywhere unique. Let r be the time required to make a payoff-function evaluation. Note that r is independent of K. The trivial algorithm turns out to require $O(rK^n)$ time: The algorithm performs K payoff-function evaluations to find the best-responses, K^{n-1} times. Thus the algorithm performs K^n payoff-function evaluations. By the accounting procedure in Aho, Hopcroft, and Ullman [1], the running-time of the algorithm is $O(rK^n)$.

If best-responses are not unique, the algorithm needs to check if any player $j \neq i$ wants to deviate from her strategy in s_{-i}, for all of i's best-responses. In the worst case, the set of best responses grows at rate K, so the algorithm is $O(rK^{n+1})$.

Note that the $O(rK^n)$ calculation is not an unrealistic worst-case bound. It is the time the trivial algorithm *must* use, as long as best-responses are unique.

For n-player games with $n > 2$, Bernhard von Stengel (personal communication) has suggested a recursive procedure that improves on the trivial algorithm: for each $s_n \in S_n$, fix s_n as the strategy played by player n, and find all equilibria of the resulting $(n-1)$-player game. For each equilibrium found, check if player n wishes to deviate from s_n. In the worst-case calculation, this recursive procedure does not improve on the simple trivial algorithm—but in many games of interest it may speed up the trivial algorithm by saving on calculations of best-response by player n. If Γ is a GSC, for any $s_n \in S_n$, the resulting $(n-1)$-player game is also a GSC. So von Stengel's suggestion can be applied to Algorithm 2 as well. But I do not know if following his suggestion improves over the speed of Algorithm 2 or not.

7. Performance

I evaluate Algorithm 2 on a class of two-player games, where each player has the interval $[0, 1]$ as her strategy space. The algorithm is fast; I use Algorithm 2 with different discretizations of $[0, 1]$, and show that, even when the resulting grid is quite small, the algorithm is very fast. I compare Algorithm 2 to the trivial algorithm.

7.1. **Class of games.**

I use a class of games that tend to have a large number of equilibria—Algorithm 2 is faster the smaller is the number of equilibria, and I want to evaluate Algorithm 2 using games where it does not have an apriori advantage. The class of games is idiosyncratic, but that is unavoidable: the functional forms that economists normally use give few—or unique—equilibria. With those functional forms, Algorithm 2 is faster still.

TABLE II. Simulations.

One game

Strat.	Trivial	Alg. 2
20,000	2.8 min	2.96 sec.
40,000	12.1 min	10.0 sec.
60,000	26.1 min	10.8 sec.

2,000 games

Strat.	Trivial	Alg. 2
20,000	3.8 days	1.6 hours
40,000	16.8 days	5.5 hours
60,000	36.0 days	6.0 hours

The games have two-players, each player i has strategy set $S_i = [0,1]$, and payoff function

$$\begin{aligned} u_i(s_i, s_{-i}) &= -(\alpha_i/10)(s_i - s_{-i})^2 + 200\beta_i \sin(100 s_i) \\ &+ (1/100)\left[(1-\alpha_i)s_i(1+s_{-i}) - (1/2-\beta_i)s_i^2/100\right]. \end{aligned}$$

The parameters α_i and β_i are in $[0,1]$.

I arrived at the above functional form by trying to come up with games that have a fairly large number of equilibria. The first summand is a "pure-coordination term," its role is to produce multiple equilibria. The role of the second summand is to provoke multiple maxima (so that preferences are not strict, see Section 9); the second summand also helps in getting multiple equilibria. [5] The third and fourth summand are variants of polynomial terms that I found—by trial and error—often produce multiple equilibria.

Note that, for all α_i and β_i, $(\{1,2\}, \{S_1, S_2\}, \{u_1, u_2\})$ is a GSC.

I discretized the players' strategy spaces, so each player i chooses a strategy in $S_i = \{k/K : 0 \leq k \leq K\}$. Parameters α_i and β_i are chosen at pseudo-random from $[0,1]$ using a uniform distribution.

7.2. **Results.** The results are in Table II. I first compare the performance of Algorithm 2 and the trivial algorithm. Then I discuss Algorithm 2 in general.

I simulated a large number of games, and used the algorithms to find the equilibria of each game. In each individual game, the parameters α_i and β_i were generated at pseudo-random from a uniform distribution on $[0,1]$. The average results are in the table on the left.

On average, when each player has 20,000 strategies, Algorithm 2 needed 2.96 seconds to find all equilibria. The trivial algorithm needed 2.8 minutes to do the same work. The table reports the results for 40,000 and 60,000 strategies as well.

The table on the right reports how much each algorithm needs to find all the equilibria of 2,000 games—presumably a task similar to calibrating an economic

[5]Simulations without the second summand are slightly faster, and the games have fewer equilibria.

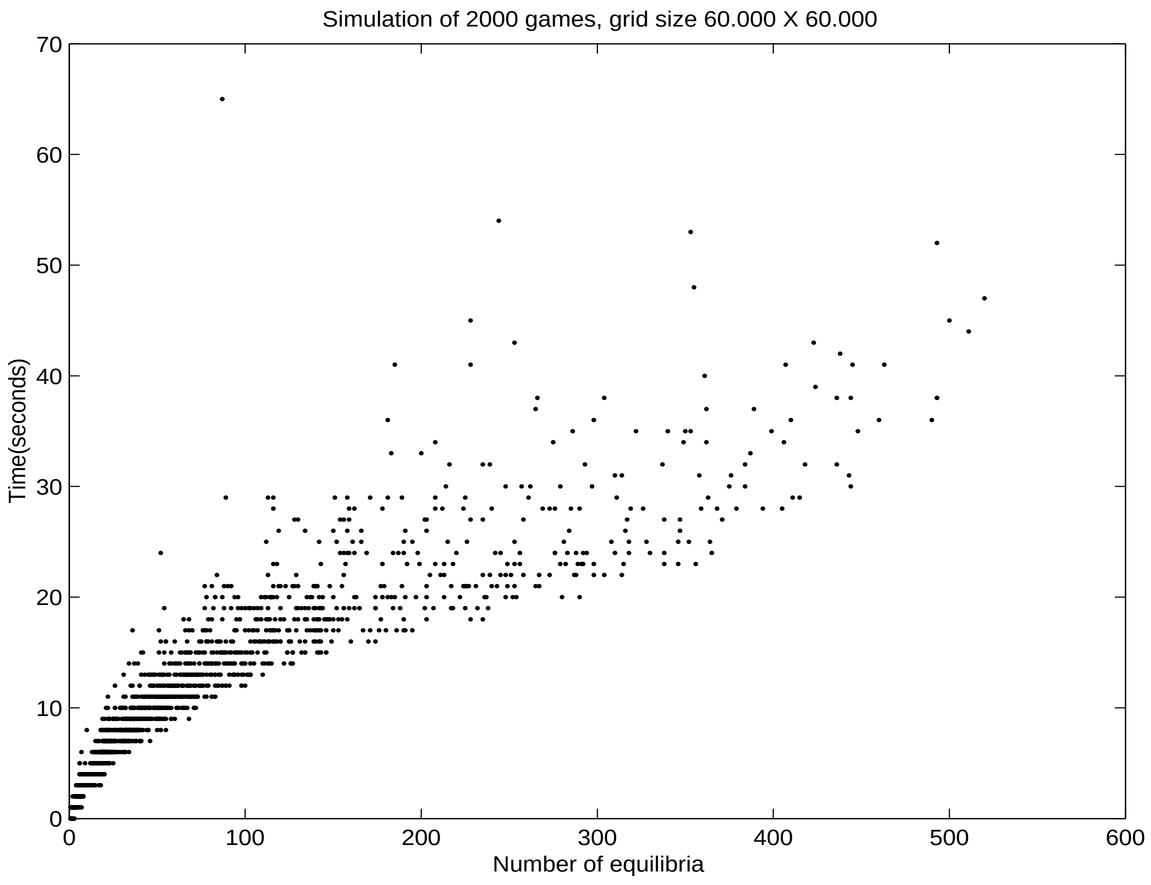

FIGURE 3. Relation between time and number of equilibria.

model. Already with 20,000 strategies the trivial algorithm needs 3.8 days. Algorithm 2, on the other hand, needs only 1.6 hours.

On the whole, Algorithm 2 is remarkably fast. It finds all equilibria of a game with 3.6×10^9 strategy-profiles in only 10 seconds, and it finds the equilibria of 2,000 such games in 6 hours. The graph in Figure 3 shows that nothing is hidden in the averages; the graph plots all 2,000 games. It shows how many equilibria each game has, and how long it takes for the algorithm to find them. Note that, with one exception, Algorithm 2 never needs more than one minute to find all equilibria.

Table III compares the performance of the two algorithms. With 20,000 strategies, Algorithm 2 is 56 times faster than the Trivial algorithm. The relative performance of Algorithm 2 improves with the number of strategies, because the performance of Algorithm 2 is sensitive to the number of equilibria more than to the number of strategies.

7.3. **Implementation.** The implementation is written in C. The code and output can be downloaded from `http://www.hss.caltech.edu/~fede`. The implementation does not take advantage of all the features of Algorithm 2. In particular, when it checks that a candidate strategy profile is an equilibrium, it searches for a profitable deviation over all the players' strategies. So the above results are an lower bound on the performance of a less daft implementation of Algorithm 2.

A difficulty in implementing Algorithm 2 is that the set of possible states of the algorithm is potentially large. Reserving space for the possible states may slow down the program. I found a rudimentary solution in my implementation by choosing, in each iteration, only the minimal elements of the state. The algorithm may then search more than is necessary, but the set of strategies in a space is kept to a minimum. There is also a choice to be made of data structures for representing states. I chose a sufficiently large array—a simple solution, but probably inefficient. Another possible choice is a linked list.

The results on the left in Table II are based on 100 simulations in the case of the trivial algorithm, and 2,000 simulations in the case of Algorithm 2. The trivial algorithm needs the same amount of time to compute the equilibria of each game. There is a very slight difference in the actual computation of some games, probably due to how the computer organizes the task, or because some payoff functions require slightly more time than others. But all 100 games with 20,000 strategies required essentially 2.8 minutes.

All the simulations were done on a Linux Dell Precision PC with a 1.8 GHz Xeon CPU and 512 MB Ram. The computer performs 252.2 million floating-point operations per second (based on output MFLOPS(1), from Al Aburto's flops.c program).

8. Calculating best-responses

8.1. **The source of slowness.** The example in Section 5 shows that Algorithm 2 can be slow. I shall argue that the example is in some sense pathological, and that the algorithm is likely to be very fast in most situations—in particular when the source of the game is the discretization of a game with continuous strategy-spaces.

The algorithm is slow when best-response calculations "advance slowly." For example, consider a game with two players, each with $K = 10$ strategies: the

Table III. Comparison of Trivial and Algorithm 2

Strat.	Trivial/Alg. 2
20,000	56
40,000	66
60,000	146

numbers $31, 32, \dots 40$. If player 1's best response to 2 playing 31 is 31, then computing 1's best-response to 2 playing 32 requires 10 computations; it requires computing the payoffs from *all* strategies, and finding the maximum. But if 1's best response to 2 playing 31 is 39, then computing 1's best response to 32 requires only 2 computations—best responses are monotone increasing, so it is enough to compare the payoffs from playing 39 and 40.

Hence, if best-responses advance slowly relative to K—as K grows—then Algorithm 2 is relatively slow. On the other hand, if best-responses advance at rate K or faster, the algorithm will be fast. In fact, I shall establish that the algorithm will be linear in the worst case.

When will best-responses advance quickly? Consider an n-player game, and suppose that player i has strategy-space $[0, 1]$. We can apply Algorithm 2 by first discretizing i's strategy-space to $\{k/K : k = 0, \dots K\}$.

Figure 4 illustrates the situation. On the top part of the figure is player i's payoff function $s_i \mapsto u_i(s_i, s_{-i})$, holding opponents' strategies fixed at s_{-i}. In the middle of the figure is i's best-response to s_{-i}, when i is restricted to choosing s_i or a larger strategy—her strategy-space in $\Gamma^r(s_i, s_{-i})$. We select the smallest best response when there is more than one. For strategies $s_i \in [0, s_i^1]$, s_i^1 is the best-response. For strategies $s_i \in (s_i^1, s_i^2)$, it is a best-response to choose s_i; the solution is at a corner. For $s_i \in [s_i^2, s_i^3]$, on the other hand, i's best response is to choose s_i^3. Finally, for $s_i \in (s_i^3, 1]$, it is again optimal for i to be at a corner, and s_i is the unique best-response in the restricted game. The bottom of Figure 4 shows $\beta_{i,\Gamma^r(s_i,s_{-i})}(s_i, s_{-i}) - s_i$.

I now show that best-responses advance quickly over sets of s_i such that $\beta_{i,\Gamma^r(s_i,s_{-i})}(s_i, s_{-i}) - s_i > 0$ (indicated as "Fast" regions in Figure 4), and slowly over sets of s_i such that $\beta_{i,\Gamma^r(s_i,s_{-i})}(s_i, s_{-i}) - s_i = 0$ (indicated as "Slow" regions in Figure 4).

Let $k^*(K)/K$ be i's smallest best-response to s_{-i} in the discretized version of the game—i.e. the game where i has strategy-space $\{k/K : k = 0, \dots K\}$. Assume that i's payoff function is continuous, then the maximum theorem ensures that

$$\inf \beta_{i,\Gamma^r(s_i,s_{-i})}(s_i, s_{-i}) \leq \liminf_{K\to\infty} k^*(K)/K,$$

as the set of maximizers parameterized by K is upper hemicontinuous.

First, let s_i be such that $\inf \beta_{i,\Gamma^r(s_i,s_{-i})}(s_i, s_{-i}) - s_i > 0$. Then $k^*(K) - s_i$ grows at rate at least K. So the number of strategies that we eliminate from future best-response calculations, $k^*(K)$, grows at rate at least K, and thus best-responses advance quickly. In fact (see 8.2) it will in the worst case advance linearly.

Second, let s_i be such that $\inf \beta_{i,\Gamma^r(s_i,s_{-i})}(s_i, s_{-i}) - s_i = 0$. As K grows, $k^*(K)/K$ will approach s_i, so the number of strategies we eliminate from future calculations will be, in the limit, zero. In practice, the algorithm will advance one-step-at-a-time, until an increase in some other player's strategy pulls the algorithm away from the region.

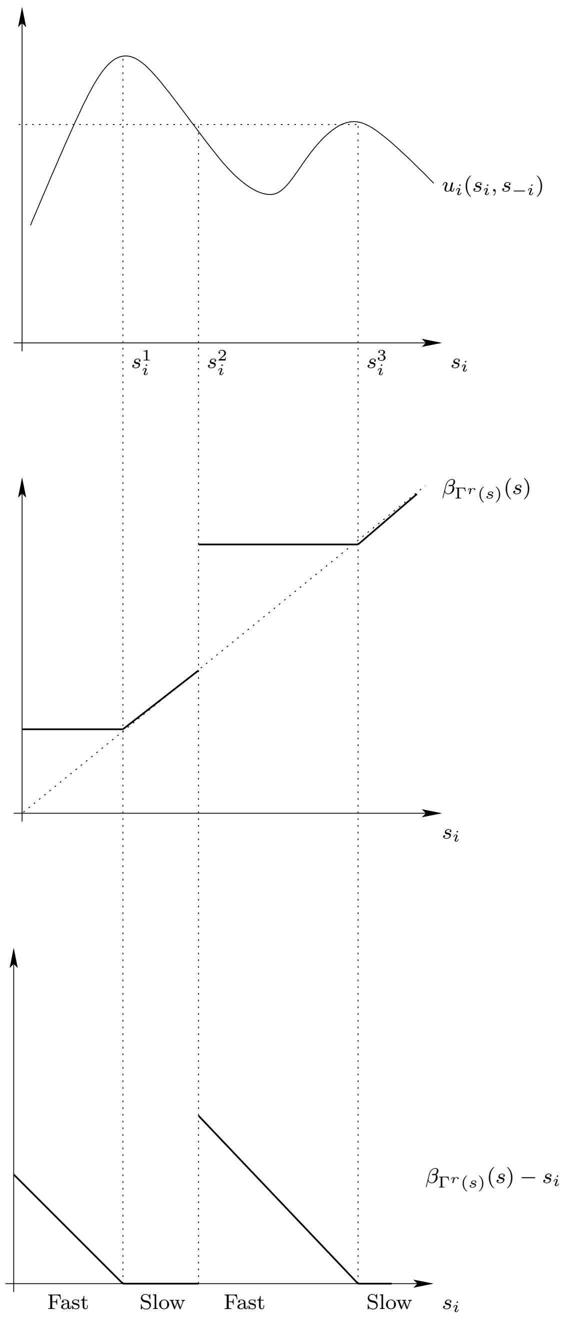

Figure 4. Slow and fast regions.

.

So Algorithm 2 will advance quickly over fast regions, and slowly in slow regions. The overall result depends on the size of the regions, on how much time it spends in each region, and on whether the algorithm will actually end up in these regions. For example, the largest equilibrium often implies that the algorithm does not need to search over the slow regions on the upper side of players' strategy spaces.

Normally, fast regions are important enough, and the algorithm is fast enough over these regions (faster than linear), that the algorithm finds all equilibria very quickly.

8.2. **Bounding how much best-responses change with K.**

The algorithm's running time depends on how slowly best responses advance. But it is too strong to require that best responses advance quickly globally—normally there are some strategy-profiles at which best responses advance slowly. One possible solution is to control the size of the regions in strategy space where best responses must advance slowly. I develop a different solution below. I consider a family of games, and assume that, at each s in strategy space, the average best-response over the family of games does not advance slowly. It follows that Algorithm 2 is on average linear.

Let $(\Omega, \mathcal{F}, P)$ be a probability space. Let $(\Gamma_\omega, \omega \in \Omega)$ be a family of GSC such that Γ_ω has n players, for all $\omega \in \Omega$, and each player i has strategy-space $\{1, 2, \ldots K\}$.

Assume that there is $\gamma \in (0,1)$ such that, for any strategy-profile s,

$$\gamma(K - s_i) \leq E \inf \beta_{i,\Gamma^r_\omega(s)}(s) - s_i.$$

Note that S_i is finite, so $E \inf \beta_{i,\Gamma^r_\omega(s)}(s)$ is well-defined.

Let $T(\omega)$ be the time required by Algorithm 2 to find all candidate equilibria. $T(\omega)$ requires counting the number of payoff-function evaluations performed by the algorithm [1, p. 35–38].Strategy spaces are fixed, so $T(\omega)$ has finite range, and thus $E(T)$ is well-defined.

Proposition 8. *$E(T)$ is $O(nK/\gamma^2)$.*

Proof. Let L be the number of iterations in Algorithm 2. Each iteration l, $l = 1, 2, \ldots L$, has I_l iterations in the RT-algorithm. Each iteration h, $h = 1, \ldots I_l$, implies a best-response calculation for each player i. Fix one player. Let x^h_l be the strategy that is a best response in iteration l of Algorithm 2 and iteration h of the corresponding RT algorithm. Let K_l be the number of strategies we need to consider for that player in iteration l; note that $K_l = K - x^{I_{l-1}}_{l-1}$. Since the player under consideration is fixed, there should be no confusion in using subindexes of K to denote iterations in this section. If the calculated best response in iteration $h-1$ was x^{h-1}_l, the best-response calculation is done in $K_l - x^{h-1}_l$ time. We thus need to bound $\sum_{l=1}^{L} \sum_{h=0}^{I_l} (K_l - x^h_l)$.

Consider the l-th iteration of the RT algorithm, where each player has K_l strategies. I shall prove that

$$E\sum_{h=0}^{I_l-1}(K_l - x_l^h) \leq EK_l/\gamma$$

by induction. First,

$$\begin{aligned} E\sum_{h=0}^{I_l-1}(K_l - x_l^h) &= E\left\{\sum_{h=0}^{I_l-2}(K_l - x_l^h) + E\left[(K_l - x_l^{I_l-1})|x_l^{I_l-2}\right]\right\} \\ &\leq E\left\{\sum_{h=0}^{I_l-2}(K_l - x_l^h) + (1-\gamma)(K_l - x_l^{I_l-2})\right\}, \end{aligned}$$

as the assumption on mean best responses implies that $E\left[(K_l - x_l^{I_l-1})|x_l^{I_l-2}\right] \geq x_l^{I_l-2} + \gamma\left(K_l - x_l^{I_l-2}\right)$, and so $E\left[(K_l - x_l^{I_l-1})|x^{I_l-2}\right] \leq (1-\gamma)(K_l - x_l^{I_l-2})$.

Second, suppose as inductive hypothesis that

$$E\sum_{h=0}^{I_l-1}(K_l - x_l^h) \leq E\left\{\sum_{h=0}^{(I_l-1)-m}(K_l - x_l^h) + \sum_{h=0}^{m}(1-\gamma)^h(K_l - x_l^{(I_l-1)-m})\right\},$$

for $m = 1, \ldots I_l - 2$. Then

$$E\left\{\sum_{h=0}^{(I_l-1)-m}(K_l - x_l^h) + \sum_{h=0}^{m}(1-\gamma)^h(K_l - x_l^{(I_l-1)-m})\right\}$$

$$= E\left\{\sum_{h=0}^{(I_l-1)-m}(K_l - x_l^h) + \sum_{h=0}^{m}(1-\gamma)^h E\left[(K_l - x_l^{(I_l-1)-m})|x_l^{(I_l-1)-(m+1)}\right]\right\}$$

$$\leq E\left\{\sum_{h=0}^{(I_l-1)-(m+1)}(K_l - x_l^h) + \sum_{h=0}^{m+1}(1-\gamma)^h(K_l - x_l^{(I_l-1)-(m+1)})\right\}.$$

Where the inequality follows from the hypothesis on mean best responses, similarly to the case I considered first. Induction on m thus proves

$$E\sum_{h=0}^{I_l-1}(K_l - x_l^h) \leq EK_l\sum_{h=0}^{I_l-1}(1-\gamma)^h = EK_l\left[1-(1-\gamma)^{I_l-1}\right]/g \leq K_l/\gamma$$

Now, by a similar calculation

$$E\sum_{l=1}^{L}K_l/\gamma \leq E\sum_{l=1}^{L}(1-\gamma)^{\sum_{j=0}^{l-1}I_j}K/\gamma \leq E\sum_{l=1}^{L}(1-\gamma)^{l\bar{I}}K/\gamma,$$

where $\overline{I} = \max \{I_l : l = 1, \dots L\}$. Then

$$E \sum_{l=1}^{L} K_l/\gamma \leq E \frac{1-(1-\gamma)^{\overline{I}(L+1)}}{1-(1-\gamma)} K/\gamma \leq K/\gamma^2.$$

But there are n players, so the expected time used by the algorithm is bounded by

$$nK/\gamma^2.$$

□

9. Two-player games with strict preferences

Let Γ be a two-player game where players have strict preferences and $d_1 = d_2 = 1$. I present a simple version of Algorithm 2 that finds all the equilibria of Γ.

Algorithm 3. *Find $\underline{s} = \inf \mathcal{E}$ using $\underline{T}(\inf S)$, and $\overline{s} = \sup \mathcal{E}$ using $\overline{T}(\sup S)$. Let $\hat{\mathcal{E}} = \{\underline{s}, \overline{s}\}$. The set of possible states of the algorithm is $S \times S$, the algorithm starts at state $(\underline{s}, \underline{s})$.*

Let the state of the algorithm be $m = (s, s^*) \in S \times S$. While $s \neq \overline{s}$, repeat the following sub-routine to obtain a new state m'.

Subroutine *If $s + (1,1) \leq \overline{s}$, then do steps 1-3:*

1. *Run $T^r(s + (1,1))$; let $\hat{s}$ be the strategy profile at which it stops.*
2. *Check that no player j wants to deviate from $\hat{s}_j$ to a strategy in the interval $\left[s^*_j, (m + (1,1))_j\right]$. If no player wants to deviate, add $\hat{s}$ to $\hat{\mathcal{E}}$.*
3. *Let $m' = (\hat{s}, \inf \beta_\Gamma(\hat{s}))$.*

Say that Algorithm 3 makes an *iteration* each time it does steps 1-3. Let $\underline{K} = \min \{K_1, K_2\}$.

Proposition 9. *Algorithm 3 finds all Nash equilibria in at most $\underline{K}$ iterations.*

Proof. The proof that Algorithm 3 is well-behaved and finds all Nash equilibria is similar to the proof for Algorithm 2. I omit it; it can be found in the working-paper version of the paper [9].

Now I shall prove that the algorithm needs less than $\underline{K}$ iterations. First, each iteration of Algorithm 3 produces one and only one element of M, so there are no more iterations than there are elements in M. Second, $M \subseteq \{1, \dots K_1\} \times \{1, \dots K_2\}$, and for each $m, m' \in M$, $m \neq m'$ then either $m + (1,1) \leq m'$ or $m' + (1,1) \leq m$. Thus M cannot have more elements than either $\{1, \dots K_1\}$ or $\{1, \dots K_2\}$. Thus, M has not more than $\min \{K_1, K_2\} = \underline{K}$ elements. □

References

[1] Aho, A. V., J. E. Hopcroft, and J. D. Ullman (1974): *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, MA.

[2] Amir, R., and V. E. Lambson (2000): "On the Effects of Entry in Cournot Markets," *Review of Economic Studies*, 67(2), 235–254.

[3] Bernstein, F., and A. Federgruen (2004a): "Decentralized Supply Chains with Competing Retailers under Demand Uncertainty," Forthcoming in Management Science.

[4] ——— (2004b): "A General Equilibrium Model for Industries with Price and Service Competition," Forthcoming in Operations Research.

[5] Cachon, G. P. (2001): "Stock wars: inventory competition in a two echelon supply chain," *Operations Research*, 49(5), 658–674.

[6] Cachon, G. P., and M. A. Lariviere (1999): "Capacity choice and allocation: strategic behavior and supply chain performance," *Management Science*, 45(8), 1091–1108.

[7] Cachon, G. P., and S. Netessine (2003): "Game Theory in Supply Chain Analysis," Forthcoming in the collecction "Supply Chain Analysis is the eBusiness Era.".

[8] Crooke, P., L. M. Froeb, S. Tschantz, and G. J. Werden (1997): "Properties of Computed Post-Merger Equilibria," Mimeo, Vanderbildt University, Department of Mathematics.

[9] Echenique, F. (2002): "Finding All Equilibria," Caltech Social Science Working Paper 1153.

[10] Echenique, F., and A. Edlin (2004): "Mixed Strategy Equilibria in Games of Strategic Complements are unstable," *Journal of Economic Theory*, 118(1), 61–79.

[11] Judd, K. L., S. Yeltekin, and J. Conklin (2000): "Computing Supergame Equilibria," mimeo Hoover Institution.

[12] Lippman, S. A., and K. F. McCardle (1997): "The Competitive Newsboy," *Operations Research*, 45(1), 54–65.

[13] McKelvey, R. D., and A. McLennan (1996): "Computation of Equilibria in Finite Games," in *Handbook of Computational Economics*, ed. by H. M. Amman, D. A. Kendrick, and J. Rust, vol. 1. North Holland, Amsterdam.

[14] Milgrom, P., and J. Roberts (1990): "Rationalizability, Learning and Equilibrium in Games with Strategic Complementarities," *Econometrica*, 58(6), 1255–1277.

[15] Milgrom, P., and C. Shannon (1992): "Monotone Comparative Statics," Stanford Institute for Theoretical Economics Working Paper.

[16] ——— (1994): "Monotone Comparative Statics," *Econometrica*, 62(1), 157–180.

[17] Netessine, S., and N. Rudi (2003): "Supply Chain choice on the Internet," Mimeo, Upenn, http://www.netessine.com/.

[18] Netessine, S., and R. Shumsky (2003): "Revenue management games: horizontal and vertical competition," Mimeo, Upenn, http://www.netessine.com/.

[19] Robinson, J. (1951): "An Iterative Method of Solving a Game," *The Annals of Mathematics*, 54(2), 296–301.

[20] Topkis, D. M. (1979): "Equilibrium Points in Nonzero-Sum n-Person Submodular Games," *SIAM Journal of Control and Optimization*, 17(6), 773–787.

[21] ——— (1998): *Supermodularity and Complementarity.* Princeton University Press, Princeton, New Jersey.

[22] Vives, X. (1990): "Nash Equilibrium with Strategic Complementarities," *Journal of Mathematical Economics*, 19(3), 305–321.

[23] ——— (1999): *Oligopoly Pricing.* MIT Press, Cambridge, Massachusetts.

[24] von Stengel, B. (2002): "Computing Equilibria for Two-Person Games," in *Handbook of Game Theory*, ed. by R. J. Aumann, and S. Hart, vol. 3. North Holland, Amsterdam.

[25] von Stengel, B., A. van den Elzen, and D. Talman (2002): "Computing Normal-Form Perfect Equilibria for Extensive Two-Person Games," *Econometrica*, 70(2), 693–715.

[26] Werden, G. J., L. M. Froeb, and S. Tschantz (2001): "The Effects of Merger Synergies on Consumers of Differentiated Products," Mimeo, US Department of Justice, Antitrust Division.

[27] Zhou, L. (1994): "The Set of Nash Equilibria of a Supermodular Game Is a Complete Lattice," *Games and Economic Behavior*, 7(2), 295–300.

HSS 228-77, California Institute of Technology, Pasadena CA 91125.
E-mail address: fede@caltech.edu
URL: http://www.hss.caltech.edu/~fede/