Lecture 19

Pure Nash Equilibria and $\mathcal{PLS}$-Completeness

The final two lectures study the limitations of learning dynamics and computationally efficient algorithms for converging to and computing equilibria, and develop analogs of $\mathcal{NP}$-completeness that are tailored to equilibrium computation problems. After setting the stage by reviewing our positive results and motivating the use of computational complexity theory (Section 19.1), this lecture develops the theory of $\mathcal{PLS}$-completeness (Section 19.2) and applies it to give evidence that computing a pure Nash equilibrium of a congestion game is an intractable problem (Section 19.3).

This lecture and the next assume basic familiarity with polynomial-time algorithms and $\mathcal{NP}$-completeness (see the Notes for references).

19.1 When Are Equilibrium Concepts Tractable?

19.1.1 Recap of Tractability Results

Lectures 16–18 prove four satisfying equilibrium tractability results, stating that simple and natural dynamics converge quickly to an approximate equilibrium. See also Figure 19.1. These results support the predictive power of these equilibrium concepts.

Four Tractability Results

1. (Corollary 17.7 and Proposition 17.9) In every game, the time-averaged history of joint play of no-regret dynamics converges quickly to an approximate coarse correlated equilibrium (CCE).

2. (Proposition 18.4 and Theorem 18.5) In every game, the time-averaged history of joint play of no-swap-

regret dynamics converges quickly to an approximate correlated equilibrium (CE).

3. (Corollary 17.7 and Theorem 18.7) In every two-player zero-sum game, the time-averaged history of joint play of no-regret dynamics converges quickly to an approximate mixed Nash equilibrium (MNE).

4. (Theorem 16.3) In every atomic routing game where all agents share the same origin and destination, many variants of ϵ-best-response dynamics converge quickly to an approximate pure Nash equilibrium (PNE).

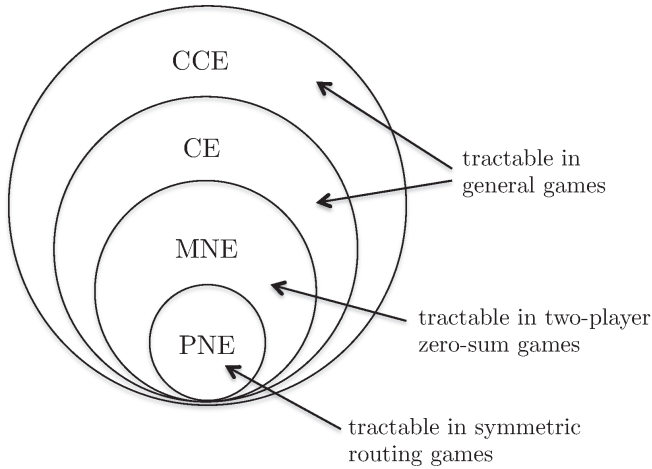

Figure 19.1: Tractability results for the hierarchy of equilibrium concepts. "Tractable" means that simple and natural dynamics converge quickly to an approximate equilibrium, and also that there are polynomial-time algorithms for computing an exact equilibrium.

Can we prove stronger tractability results? For example, can simple dynamics converge quickly to approximate MNE in general two-player games, or to approximate PNE in general atomic selfish routing games?

19.1.2 Dynamics vs. Algorithms

With an eye toward intractability results, we weaken our standing notion of tractability from

> Are there simple and natural dynamics that converge quickly to a given equilibrium concept in a given class of games?

to

> Is there an algorithm that computes quickly a given equilibrium concept in a given class of games?

Technically, by "quickly" we mean that the number of iterations required for convergence or the number of elementary operations required for computation is bounded by a polynomial function of the number of parameters needed to specify all of the agents' cost or payoff functions.[1] For instance, kn^k parameters are required to define all of the costs or payoffs of an arbitrary game with k agents with n strategies each (k payoffs per outcome). Special classes of games often have compact descriptions with much fewer than kn^k parameters. For example, in an atomic selfish routing game with k agents and m edges, mk parameters suffice to specify fully the agents' cost functions (the cost $c_e(i)$ of each edge e for each $i \in \{1, 2, \ldots, k\}$).

One particular type of algorithm for computing an approximate equilibrium is to simulate a choice of dynamics until it (approximately) converges. Provided each iteration of the dynamics can be simulated in polynomial time and that the dynamics requires only a polynomial number of iterations to converge, the induced algorithm runs in polynomial time. This is the case for the four tractability results reviewed in Section 19.1.1, provided no-regret and no-swap-regret dynamics are implemented using a computationally efficient subroutine like the multiplicative weights algorithm. We conclude that the second goal is weaker than the first, and hence impossibility results for it are only stronger.

In all four of the settings mentioned in Section 19.1.1, there are also polynomial-time algorithms for computing an exact equilibrium that are not based on any natural dynamics (Problems 18.3, 18.4,

[1] To be fully rigorous, we should also keep track of the number of bits required to describe these costs or payoffs. We omit further discussion of this issue.

and 19.1). These exact algorithms seem far removed from any reasonable model of how agents learn in strategic environments.

19.1.3 Toward Intractability Results

There is no simple learning procedure that is known to converge quickly to approximate MNE in general two-player games or to approximate PNE in general atomic selfish routing games. There are not even any known polynomial-time algorithms for computing such equilibria. Do we merely need a new and clever idea, or are such results impossible? How might we prove limitations on equilibrium tractability?

These questions are in the wheelhouse of computational complexity theory. Why is it so easy to come up with polynomial-time algorithms for the minimum-spanning tree problem and so difficult to come up with one for the traveling salesman problem? Could it be that no efficient algorithm for the latter problem exists? If so, how can we prove it? If we can't prove it, how can we nevertheless amass evidence of computational intractability? These questions are addressed by the theory of $\mathcal{NP}$-completeness. This lecture and the next assume basic knowledge of this theory and describe analogs of $\mathcal{NP}$-completeness for equilibrium computation problems.

19.2 Local Search Problems

This section is a detour into a branch of complexity theory designed to reason about local search problems. The resulting theory is perfectly suited to provide evidence of the inherent intractability of computing a PNE of an atomic selfish routing game. Briefly, the connection is that computing a PNE of such a game is equivalent to computing a local minimum of the potential function defined in (13.7).

19.2.1 Canonical Example: The Maximum Cut Problem

A canonical problem through which to study local search is the *maximum cut* problem. The input is an undirected graph $G = (V, E)$ with a nonnegative weight $w_e \geq 0$ for each edge $e \in E$. Feasible solutions correspond to *cuts* $(X, \overline{X})$, where $(X, \overline{X})$ is a partition of V into two sets. The objective is to maximize the total weight of the cut edges,

meaning the edges with one endpoint in each of X and $\overline{X}$.[2] The maximum cut problem is $\mathcal{NP}$-hard, so assuming that $\mathcal{P} \neq \mathcal{NP}$, there is no polynomial-time algorithm that solves it.

Local search is a natural heuristic that is useful for many $\mathcal{NP}$-hard problems, including the maximum cut problem. The algorithm is very simple.

Local Search for Maximum Cut

initialize with an arbitrary cut $(X, \overline{X})$
while there is an improving local move **do**
    take an arbitrary such move

By a *local move*, we mean moving a single vertex v from one side of the cut to the other. For example, when moving a vertex v from X to $\overline{X}$, the increase in objective function value is

$$\underbrace{\sum_{u \in X\,:\,(u,v) \in E} w_{uv}}_{\text{newly cut}} - \underbrace{\sum_{u \in \overline{X}\,:\,(u,v) \in E} w_{uv}}_{\text{newly uncut}}. \tag{19.1}$$

If the difference in (19.1) is positive, then this is an *improving* local move. Local search stops at a solution with no improving local move, a *local optimum*. A local optimum need not be a global optimum (Figure 19.2).

We can visualize local search as a walk in a directed graph H (Figure 19.3). For a maximum cut instance with input graph G, vertices of H correspond to cuts of G. Each directed edge of H represents an improving local move from one cut to another. There can be no cycle of such moves, so H is a directed acyclic graph. Vertices with no outgoing edges—*sink vertices* of the graph H—correspond to the local optima. Local search repeatedly follows outgoing edges of H until it reaches a sink vertex.

Since there are only more local optima than global optima, they are only easier to find. For example, consider the special case of maximum cut instances in which every edge has weight 1. Computing

[2] Graph cuts are usually defined with the additional restriction that both sides are nonempty. Permitting empty cuts as feasible solutions does not change the maximum cut problem.

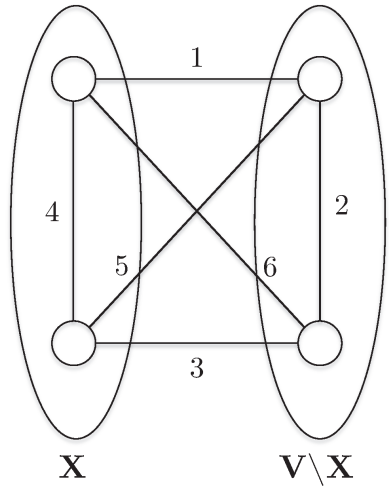

Figure 19.2: A local maximum of a maximum cut instance that is not a global maximum. The cut $(X, V \setminus X)$ has objective function value 15, and every local move results in a cut with smaller objective function value. The maximum cut value is 17.

a global maximum remains an $\mathcal{NP}$-hard problem, but computing a local maximum is easy. Because the objective function in this case can only take on values in the set $\{0, 1, 2, \ldots, |E|\}$, local search stops (at a local maximum) within at most $|E|$ iterations.

There is no known polynomial-time algorithm, based on local search or otherwise, for computing a local optimum of a maximum cut instance with arbitrary nonnegative edge weights. How might we amass evidence that no such algorithm exists?

The strongest negative result would be an "unconditional" one, meaning a proof with no unproven assumptions that there is no polynomial-time algorithm for the problem. No one knows how to prove unconditional results like this, and such a result would separate $\mathcal{P}$ from $\mathcal{NP}$. The natural next goal is to prove that the problem is $\mathcal{NP}$-hard, and therefore admits a polynomial-time algorithm only if $\mathcal{P} = \mathcal{NP}$. Lecture 20 explains why this is also too strong a negative result to shoot for. Instead, we develop an analog of $\mathcal{NP}$-completeness tailored to local search problems. As a by-product, we also obtain strong unconditional lower bounds on the worst-case number of iterations required by local search to reach a local optimum.

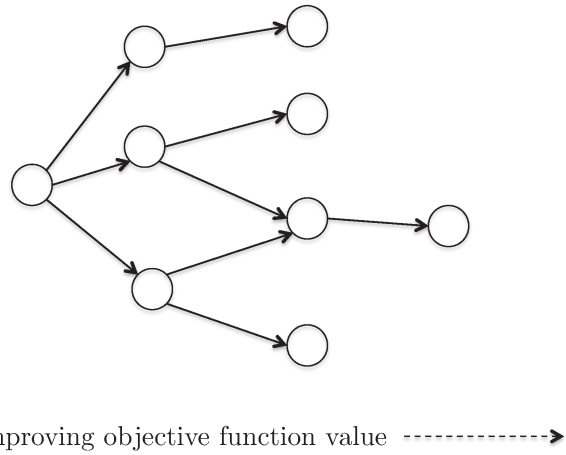

Figure 19.3: Local search can be visualized as a walk in a directed acyclic graph. Vertices correspond to feasible solutions, edges to improving local moves, and sink vertices to local minima.

19.2.2 $\mathcal{PLS}$: Abstract Local Search Problems

This section and the next make precise the idea that the problem of computing a local optimum of a maximum cut instance is *as hard as any other local search problem*. This statement is in the spirit of an $\mathcal{NP}$-completeness result, which establishes that a problem is as hard as any problem with efficiently verifiable solutions. For such "hardest" local search problems, we don't expect any clever, problem-dependent algorithms that always improve significantly over local search. This parallels the idea that for $\mathcal{NP}$-complete problems, we don't expect any algorithms that always improve significantly over brute-force search.

What could we mean by "any other local search problem?" For an analogy, recall that an $\mathcal{NP}$ problem is defined by a polynomial-time verifier of alleged solutions to a given instance, like truth assignments to the variables of a logical formula or potential Hamiltonian cycles of a graph. In some sense, an efficient verifier of purported solutions is the minimal ingredient necessary to execute brute-force search through all possible solutions, and $\mathcal{NP}$ is the class of problems that admit such a brute-force search procedure. So what are the minimal ingredients necessary to run local search?

An abstract local search problem can be a maximization or a minimization problem. One is specified by three algorithms, each running in time polynomial in the input size.

Ingredients of an Abstract Local Search Problem

1. The first polynomial-time algorithm takes as input an instance and outputs an arbitrary feasible solution.
2. The second polynomial-time algorithm takes as input an instance and a feasible solution, and returns the objective function value of the solution.
3. The third polynomial-time algorithm takes as input an instance and a feasible solution and either reports "locally optimal" or produces a solution with better objective function value.[3]

For example, in the maximum cut problem, the first algorithm can just output an arbitrary cut. The second algorithm computes the total weight of the edges crossing the given cut. The third algorithm checks all $|V|$ local moves. If none are improving, it outputs "locally optimal"; otherwise, it takes some improving local move and outputs the resulting cut.

Every abstract local search problem admits a local search procedure that uses the given three algorithms as subroutines in the obvious way. Given an instance, the generic local search procedure uses the first algorithm to obtain an initial solution, and iteratively applies the third algorithm until a local optima solution is reached.[4]

[3] We're glossing over some details. For example, all algorithms should check if the given input is a legitimate encoding of an instance. There is also some canonical interpretation when an algorithm misbehaves, by running too long or outputting something invalid. For example, we can interpret the output of the third algorithm as "locally optimal" unless it outputs a feasible solution better than the previous one, as verified by the second algorithm, within a specified polynomial number of steps. These details guarantee that a generic local search procedure, which uses these three algorithms only as "black boxes," eventually stops with a local optimum.

[4] The purpose of the second algorithm is to keep the third algorithm honest, and ensure that each solution produced does indeed have better objective function value than the previous one. If the third algorithm fails to produce an improved solution, the generic procedure can interpret its output as "locally optimal."

Since the objective function values of the candidate solutions strictly improve until a local optima solution is found, and since there is only a finite number of feasible solutions, this procedure eventually stops.[5]

As in the maximum cut problem, this local search procedure can be visualized as a walk in a directed acyclic graph (Figure 19.3)—the first algorithm identifies the starting vertex, and the third algorithm the sequence of outgoing edges. Because the number of feasible solutions can be exponential in the input size, this local search procedure could require more than a polynomial number of iterations to complete.

The goal in an abstract local search problem is to compute a local optimum, or equivalently to find a sink vertex of the corresponding directed acyclic graph. This can be done by running the generic local search procedure, but any correct algorithm for computing a local optimum is also allowed. The complexity class $\mathcal{PLS}$ is, by definition, the set of all such abstract local search problems.[6] Most if not all of the local search problems that you've ever seen can be cast as problems in $\mathcal{PLS}$.

19.2.3 $\mathcal{PLS}$-Completeness

Our goal is to prove that the problem of computing a local optimum of a maximum cut instance is as hard as any other local search problem. Having formalized "any other search problem," we now formalize the phrase "as hard as." This is done using polynomial-time reductions, as in the theory of $\mathcal{NP}$-completeness.

Formally, a *reduction* from a problem $L_1 \in \mathcal{PLS}$ to a problem $L_2 \in \mathcal{PLS}$ consists of two polynomial-time algorithms with the following properties.

A $\mathcal{PLS}$ Reduction

1. Algorithm $\mathcal{A}_1$ maps every instance $x \in L_1$ to an instance $\mathcal{A}_1(x) \in L_2$.
2. Algorithm $\mathcal{A}_2$ maps every local optimum of $\mathcal{A}_1(x)$ to a local optimum of x.

[5] The three algorithms run in polynomial time, which implicitly forces feasible solutions to have polynomial description length. Hence, there are at most exponentially many feasible solutions.

[6] The letters in $\mathcal{PLS}$ stand for "polynomial local search."

The definition of a reduction ensures that if we can solve the problem L_2 in polynomial time then, by combining the solution with algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$, we can also solve the problem L_1 in polynomial time (Figure 19.4).

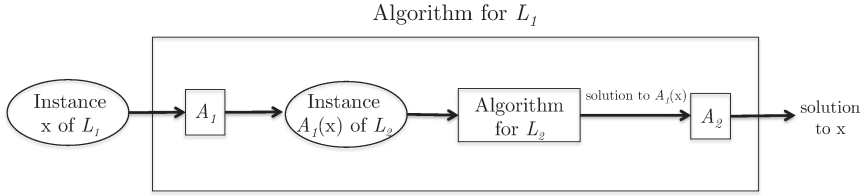

Figure 19.4: A reduction from L_1 to L_2 transfers solvability from $\mathcal{PLS}$ problem L_2 to $\mathcal{PLS}$ problem L_1.

Definition 19.1 ($\mathcal{PLS}$-Complete Problem) A problem L is *$\mathcal{PLS}$-complete* if $L \in \mathcal{PLS}$ and every problem in $\mathcal{PLS}$ reduces to it.

By definition, there is a polynomial-time algorithm for solving a $\mathcal{PLS}$-complete problem if and only if every $\mathcal{PLS}$ problem can be solved in polynomial time.[7]

A $\mathcal{PLS}$-complete problem is a *single* local search problem that simultaneously encodes *every* local search problem. If we didn't already have the remarkable theory of $\mathcal{NP}$-completeness to guide us, we might not believe that a $\mathcal{PLS}$-complete problem could exist. But just like $\mathcal{NP}$-complete problems, $\mathcal{PLS}$-complete problems *do* exist. Even more remarkably, many natural and practically relevant problems are $\mathcal{PLS}$-complete, including the maximum cut problem.

Theorem 19.2 (Maximum Cut is $\mathcal{PLS}$-Complete) *Computing a local maximum of a maximum cut instance with general nonnegative edge weights is a $\mathcal{PLS}$-complete problem.*

The proof of Theorem 19.2 is difficult and outside the scope of this book (see the Notes).

[7] Most researchers believe that $\mathcal{PLS}$-complete problems cannot be solved in polynomial time, though confidence is not quite as strong as for the $\mathcal{P} \neq \mathcal{NP}$ conjecture.

We already mentioned the conditional result that, unless every $\mathcal{PLS}$ problem can be solved in polynomial time, there is no polynomial-time algorithm, based on local search or otherwise, for any $\mathcal{PLS}$-complete problem. Independent of whether or not all $\mathcal{PLS}$ problems can be solved in polynomial time, the proof of Theorem 19.2 implies that the specific algorithm of local search requires exponential time in the worst case.

Theorem 19.3 (Lower Bounds for Local Search) *Computing a local maximum of a maximum cut instance with general nonnegative edge weights using local search can require an exponential (in $|V|$) number of iterations, no matter how an improving local move is chosen in each iteration.*

19.3 Computing a PNE of a Congestion Game

19.3.1 Computing a PNE as a $\mathcal{PLS}$ Problem

Section 13.2.3 introduces *congestion games* as a natural generalization of atomic selfish routing games in which strategies are arbitrary subsets of a ground set, rather than paths in a graph. Thus a congestion game is described by a set E of resources (previously, the edges), an explicitly described strategy set $S_i \subseteq 2^E$ for each agent $i = 1, 2, \ldots, k$ (previously, the o_i-d_i paths), and the possible costs $c_e(1), \ldots, c_e(k)$ for each resource $e \in E$. The cost $C_i(\mathbf{s})$ of an agent in an outcome $\mathbf{s}$ remains the sum $\sum_{e \in s_i} c_e(n_e(\mathbf{s}))$ of the costs of the resources she uses, where $n_e(\mathbf{s})$ denotes the number of agents in the outcome $\mathbf{s}$ that use a strategy that includes the resource e.

All of our major results for atomic selfish routing games (Theorems 12.3, 13.6 and 16.3) hold more generally, with exactly the same proofs, for the analogous classes of congestion games. In particular, every congestion game is a potential game (Section 13.3) with the potential function

$$\Phi(\mathbf{s}) = \sum_{e \in E} \sum_{i=1}^{n_e(\mathbf{s})} c_e(i) \tag{19.2}$$

satisfying

$$\Phi(s_i', \mathbf{s}_{-i}) - \Phi(\mathbf{s}) = C_i(s_i', \mathbf{s}_{-i}) - C_i(\mathbf{s}) \tag{19.3}$$

for every outcome $\mathbf{s}$, agent i, and unilateral deviation s_i' by i.

We claim that the problem of computing a PNE of a congestion game is a $\mathcal{PLS}$ problem. This follows from the correspondence between best-response dynamics (Section 16.1) in a congestion game and local search with respect to the potential function (19.2). Proving the claim formally involves describing the three polynomial-time algorithms that define a $\mathcal{PLS}$ problem. The first algorithm takes as input a congestion game, described via agents' strategy sets and the resource cost functions, and returns an arbitrary outcome, such as the one in which each agent chooses her first strategy. The second algorithm takes a congestion game and an outcome $\mathbf{s}$, and returns the value of the potential function (19.2). The third algorithm checks whether or not the given outcome is a PNE, by considering each unilateral deviation of each agent.[8] If so, it reports "locally optimal"; if not, it executes an iteration of best-response dynamics and returns the resulting outcome, which by (19.3) has a smaller potential function value.

19.3.2 Computing a PNE is a $\mathcal{PLS}$-Complete Problem

Computing a PNE of a congestion game is as hard as every other local search problem.[9]

Theorem 19.4 (Computing a PNE is $\mathcal{PLS}$-Complete) *The problem of computing a PNE of a congestion game is $\mathcal{PLS}$-complete.*

Proof: Since reductions are transitive, we only need to exhibit a reduction from some $\mathcal{PLS}$-complete problem to the problem of computing a PNE of a congestion game. We give a reduction from the problem of computing a local maximum of a maximum cut instance, which is $\mathcal{PLS}$-complete (Theorem 19.2).

The first polynomial-time algorithm $\mathcal{A}_1$ of the reduction is given as input a graph $G = (V, E)$ with nonnegative edge weights $\{w_e\}_{e \in E}$. The algorithm constructs the following congestion game.

[8] Because the strategy set of each agent is given explicitly as part of the input, this algorithm runs in time polynomial in the length of the game's description.

[9] Changing the goal to computing all PNE or a PNE that meets additional criteria can only result in a harder problem (e.g., Exercise 19.3). Intractability results are most compelling for the easier problem of computing an arbitrary PNE.

1. Agents correspond to the vertices V.
2. There are two resources for each edge $e \in E$, r_e and $\bar{r}_e$.
3. Agent v has two strategies, each comprising $|\delta(v)|$ resources, where $\delta(v)$ is the set of edges incident to v in G: $\{r_e\}_{e \in \delta(v)}$ and $\{\bar{r}_e\}_{e \in \delta(v)}$.
4. A resource r_e or $\bar{r}_e$ with $e = (u, v)$ can only be used by the agents corresponding to u and v. The cost of such a resource is 0 if used by only one agent, and w_e if used by two agents.

This construction can be carried out in polynomial time.

The key point is that the PNE of this congestion game are in one-to-one correspondence with the local optima of the given maximum cut problem. We prove this using a bijection between the $2^{|V|}$ outcomes of this congestion game and cuts of the graph G, where the cut $(X, \overline{X})$ corresponds to the outcome in which every agent corresponding to $v \in X$ (respectively, $v \in \overline{X}$) chooses her strategy that contains resources of the form r_e (respectively, $\bar{r}_e$).

This bijection maps cuts $(X, \overline{X})$ of G with weight $w(X, \overline{X})$ to outcomes with potential function value (19.2) equal to $W - w(X, \overline{X})$, where $W = \sum_{e \in E} w_e$ denotes the sum of the edges' weights. To see this, fix a cut $(X, \overline{X})$. For an edge e cut by $(X, \overline{X})$, each resource r_e and $\bar{r}_e$ is used by only one agent and hence contributes 0 to (19.2). For an edge e not cut by $(X, \overline{X})$, two agents use one of $r_e, \bar{r}_e$ and none use the other. These two resources contribute w_e and 0 to (19.2) in this case. We conclude that the potential function value of the corresponding outcome is the total weight of the edges not cut by $(X, \overline{X})$, or $W - w(X, \overline{X})$.

Cuts of G with larger weight thus correspond to outcomes with smaller potential function value, so locally maximum cuts of G are in one-to-one correspondence with the local minima of the potential function. By (19.3), the local minima of the potential function are in one-to-one correspondence with the PNE of the congestion game.

The second algorithm $\mathcal{A}_2$ of the reduction simply translates a PNE of the congestion game constructed by $\mathcal{A}_1$ to the corresponding locally maximum cut of G. ■

The reduction in the proof of Theorem 19.4 establishes a one-to-one correspondence between improving moves in a maximum cut

instance and beneficial unilateral deviations in the constructed congestion game. Thus, the unconditional lower bound on the number of iterations required for local search to converge in the maximum cut problem (Theorem 19.3) translates to a lower bound on the number of iterations required by best-response dynamics to converge in a congestion game.

Corollary 19.5 (Lower Bound for Best-Response Dynamics) *Computing a PNE of a k-agent congestion game using best-response dynamics can require an exponential (in k) number of iterations, no matter how a beneficial unilateral deviation is chosen in each iteration.*

Lower bounds like Corollary 19.5 are often much easier to prove via reductions than from scratch.

19.3.3 Symmetric Congestion Games

We conclude this lecture with another reduction that extends Theorem 19.4 and Corollary 19.5 to the special case of *symmetric* congestion games, where every agent has the same set of strategies. Such games generalize atomic selfish routing games in which all agents have a common origin vertex and a common destination vertex.[10]

Theorem 19.6 ($\mathcal{PLS}$-Completeness in Symmetric Games) *The problem of computing a PNE of a symmetric congestion game is $\mathcal{PLS}$-complete.*

As with Corollary 19.5, the proof of Theorem 19.6 implies unconditional lower bounds on the number of iterations required for convergence in best-response dynamics (Exercise 19.4).

Corollary 19.7 (Lower Bound for Best-Response Dynamics) *Computing a PNE of a k-agent symmetric congestion game using best-response dynamics can require an exponential (in k) number of iterations, no matter how a beneficial unilateral deviation is chosen in each iteration.*

[10] The problem of computing a PNE of a symmetric atomic selfish routing game can be solved in polynomial time (Problem 19.1), so it is probably not $\mathcal{PLS}$-complete.

Why don't Theorem 19.6 and Corollary 19.7 contradict Theorem 16.3, which states that ϵ-best-response dynamics converges quickly in symmetric congestion games? The reason is that ϵ-best-response dynamics only converges to an approximate PNE, while Theorem 19.6 asserts the intractability of computing an exact PNE.[11] Thus, Theorem 19.6 and Corollary 19.7 provide an interesting separation between the tractability of exact and approximate PNE, and between the convergence properties of best-response and ϵ-best-response dynamics, in symmetric congestion games.

Proof of Theorem 19.6: We reduce the problem of computing a PNE of a general congestion game, which is $\mathcal{PLS}$-complete (Theorem 19.4), to that of computing a PNE of a symmetric congestion game. Given a general congestion game with resources E and k agents with arbitrary strategy sets $S_1, \ldots, S_k$, the first polynomial-time algorithm $\mathcal{A}_1$ of the reduction constructs a "symmetrized" version. The agent set remains the same. The new resource set is $E \cup \{r_1, \ldots, r_k\}$. Resources of E retain their cost functions. The cost function of each new resource r_i is defined to be zero if used by only one agent, and extremely large if used by two or more. Each strategy of S_i is supplemented by the resource r_i, and any agent can use any one of these augmented strategies. That is, the common strategy set of all agents is $\{s_i \cup \{r_i\} : i \in \{1, 2, \ldots, k\}, s_i \in S_i\}$. We can think of an agent choosing a strategy containing resource r_i as adopting the identity of agent i in the original game. The key insight is that at a PNE of the constructed symmetric game, each agent adopts the identity of exactly one agent of the original game. This is due to the large penalty incurred by two agents that choose strategies that share one of the new resources. The algorithm $\mathcal{A}_2$ can easily map such a PNE to a PNE of the original congestion game, completing the reduction. ■

The Upshot

☆ Simple and natural dynamics converge quickly to approximate CCE and approximate CE in arbitrary games, to approximate MNE in two-

[11] Our proof of Theorem 19.6 also violates the α-bounded jump assumption made in Theorem 16.3, but the proof can be modified to respect this condition.

player zero-sum games, and to approximate PNE in symmetric congestion games.

- ☆ Designing an algorithm that computes an (approximate) equilibrium quickly is a weaker goal than proving fast convergence of simple dynamics.
- ☆ $\mathcal{PLS}$ is the class of abstract local search problems, and it includes the problems of computing a locally maximum graph cut and of computing a PNE of a congestion game.
- ☆ A problem is $\mathcal{PLS}$-complete if every problem in $\mathcal{PLS}$ reduces to it.
- ☆ There is a polynomial-time algorithm for a $\mathcal{PLS}$-complete problem if and only if every $\mathcal{PLS}$ problem can be solved in polynomial time. Most experts believe that $\mathcal{PLS}$-complete problems cannot be solved in polynomial time.
- ☆ Computing a PNE of a congestion game is a $\mathcal{PLS}$-complete problem, even in the special case of symmetric congestion games.
- ☆ Best-response dynamics can require an exponential number of iterations to converge to a PNE in a congestion game, even in the special case of a symmetric congestion game.

Notes

Garey and Johnson (1979) is an accessible introduction to the theory of $\mathcal{NP}$-completeness; see also Roughgarden (2010b) for examples germane to algorithmic game theory. The definition of the complexity class $\mathcal{PLS}$ is due to Johnson et al. (1988), who also provided several examples of $\mathcal{PLS}$-complete problems and proved unconditional lower bounds on the worst-case number of iterations required by local search to compute a local optimum in these prob-

lems. Theorems 19.2 and 19.3 are proved in Schäffer and Yannakakis (1991). All of the results in Section 19.3, and also Problem 19.1(a), are from Fabrikant et al. (2004). Problem 19.1(b) is from Fotakis (2010). Fabrikant et al. (2004) also show that computing a PNE of an atomic selfish routing game with multiple origins and destinations is a $\mathcal{PLS}$-complete problem. Skopalik and Vöcking (2008) show that, in general atomic selfish routing games, Theorem 19.4 and Corollary 19.5 hold even for the problem of computing an ϵ-PNE and for ϵ-best-response dynamics, respectively. The problem of computing an exact or approximate correlated equilibrium in time polynomial in the number of agents (cf., Exercises 19.1–19.2) is addressed by Papadimitriou and Roughgarden (2008) and Jiang and Leyton-Brown (2015) for compactly represented games like congestion games, and by Hart and Nisan (2013) for general games.

Exercises

Exercise 19.1 Assume for this exercise that an optimal solution to a linear program, if one exists, can be computed in time polynomial in the size of the linear program's description. Use this fact and Problem 18.3 to give an algorithm for computing the correlated equilibrium of a general cost-minimization game with the minimum expected sum of agents' costs. Your algorithm should run in time polynomial in the description length of the game.

Exercise 19.2 *(H)* Does Exercise 19.1 imply that a correlated equilibrium of a congestion game can be computed in time polynomial in the game's description?

Exercise 19.3 *(H)* Prove that the following problem is $\mathcal{NP}$-complete: given a description of a general congestion game and a real-valued target τ, decide whether or not the game has a PNE with cost at most τ.

Exercise 19.4 Explain why the reduction in the proof of Theorem 19.6 implies Corollary 19.7.

Exercise 19.5 Given a general atomic selfish routing game with origins $o_1, \ldots, o_k$ and destinations $d_1, \ldots, d_k$, construct a symmetric such game by adding new origin and destination vertices o and d, and new directed edges $(o, o_1), \ldots, (o, o_k)$ and $(d_1, d), \ldots, (d_k, d)$, each with a cost function that is zero with one agent and extremely large with two or more agents.

Why doesn't this idea lead to a reduction, analogous to that in the proof of Theorem 19.6, from the problem of computing a PNE of a general atomic selfish routing game to that of computing a PNE of a symmetric such game?

Problems

Problem 19.1 This problem considers atomic selfish routing networks with a common origin vertex o and a common destination vertex d.

(a) *(H)* Prove that a PNE can be computed in time polynomial in the description length of the game. As usual, assume that each edge cost function is nondecreasing.

(b) *(H)* Suppose the network is just a collection of parallel edges from o to d, with no other vertices. Prove a converse to Theorem 13.6: every equilibrium flow minimizes the potential function (13.6).

(c) Show by example that (b) does not hold in general networks with a common origin vertex and a common destination vertex.