Reasoning about Additional Winning Strategies in Two-Player Games

Vadim Malvone and Aniello Murano

Università degli Studi di Napoli Federico II

Abstract. In game theory, deciding whether a designed player wins a game corresponds to check whether he has a winning strategy. There are situations in which it is important to know whether some extra winning strategy also exists. In this paper we investigate this question over two-player turn-based games under safety and fairness objectives. We provide an automata-based technique that allows to decide in polynomial-time whether the game admits more than one winning strategy.

1 Introduction

Game theory is a powerful framework, usefully applied in computer science to reason about *reactive* systems [15]. In recent years, it has been used efficiently to deal with the strategy reasoning in multi-agent systems [1, 17, 22, 27, 34].

In the basic setting, we consider two-player turn-based games. The configurations (states) of the game are partitioned between the two players, $Player_0$ and $Player_1$, and a player moves in a state whenever he owns it. Solving a two-player game amounts to checking whether $Player_0$ has a *winning strategy*, that is a complete plan of choices, one for each decision point of the player (*i.e.* a *strategy*), that allows him to satisfy the game objective, no matter how his opponent acts.

In several game settings it is mandatory to have a more precise (quantitative) information about *how many* winning strategies a player has at his disposal. For example, in Nash Equilibrium, such an information amounts to solving the question of checking whether the equilibrium is unique [2, 3, 13, 23, 24, 30]. This problem impacts on the predictive power of Nash Equilibrium since, in case there are multiple equilibria, the outcome of the game cannot be uniquely pinned down [10, 31, 35].

A recent line of research aiming at addressing uniqueness in Nash Equilibria (as well as other solution concepts), with goals expressed in LTL, concerns extending Strategy Logic (a powerful logic able to express Nash Equilibria [27]) with graded modalities [2, 3]. This approach however turns out to be less effective in practice as it requires double exponential-time. Conversely, the problem of checking the existence of a Nash equilibrium in games with LTL goals is in PSPACE [14]. This has spurred us to look for other and more efficient directions. In [25, 26], we have investigated the existence of additional winning strategies in two-player finite games under the reachability condition in which the players have perfect or imperfect information about the moves performed by the opponent. In this paper we go further and consider as objectives *safety* and *fairness*. Precisely, we consider games in which the states are partitioned between *good* and *bad* states. Under the safety condition, $Player_0$ wins the game if he can induce a play that never visits a *bad* state. Under the fairness objective, instead, $Player_0$ wins the game whenever he can induce a play along which a *good* state is visited infinitely often.

We solve the problem of checking the existence of additional winning strategy under safety and fairness objectives by using an automata-theoretic approach. Precisely, we build an automaton that accepts only trees that are witnesses of more than one winning strategy for the designed player over the game arena. Hence, we reduce the addressed quantitative question to the emptiness of this automaton. This leads to a polynomial-time solution, thus not harder than the one required for the existence of a winning strategy in safety and fair games [9,16]. As a important consequence of this result we get that checking the uniqueness of a Nash Equilbrium under safety or fairness objectives can be done in polynomial-time. This motivates our work.

Related works. Counting strategies has been deeply exploited in the formal verification of *reactive* systems by means of specification logics extended with *graded modalities*, interpreted over games of infinite duration [2,3,5,7,11,19,23, 24,26]. It is worth recalling that the solution algorithms present in the literature for graded modalities have been conceived to address complicated scenarios and, consequently, they usually perform much worse than our algorithm on the restricted setting we consider.

Finally, we remark that the automata-theoretic solution we provide takes inspiration from those ones introduced in [4,6,11,12,20,25,26,32].

2 Preliminaries

In this section we introduce some preliminary concepts needed to properly define the game setting under exam as well as to describe the adopted solution approach. In particular, we introduce trees useful to represent strategies and automata to collect winning strategies.

Trees. Let Υ be a set. An *Υ-tree* is a prefix closed subset $T \subseteq \Upsilon^*$. The elements of T are called *nodes* and the empty word ε is the *root* of T. For $v \in T$, the set of *children* of v (in T) is $child(T, v) = \{v \cdot x \in T \mid x \in \Upsilon\}$. Given a node $v = y \cdot x$, with $y \in \Upsilon^*$ and $x \in \Upsilon$, we define $prf(v)$ to be y and $last(v)$ to be x. We also say that v *corresponds* to x. The complete Υ-tree is the tree Υ^*. For $v \in T$, a (full) path π of T from v is a *minimal* set $\pi \subseteq T$ such that $v \in \pi$ and for each $v' \in \pi$ such that $child(T, v') \neq \emptyset$, there is exactly one node in $child(T, v')$ belonging to π. Note that every word $w \in \Upsilon^*$ can be thought as a path in the tree Υ^*, namely the path containing all the prefixes of w. For an alphabet Σ, a Σ-labeled Υ-tree is a pair $< T, V >$ where T is an $\Upsilon-$tree and $V : T \to \Sigma$ maps each node of T to a symbol in Σ.

Automata Theory. An *alternating tree automaton* (*ATA*, for short) is a tuple $A = < \Sigma, D, Q, q_0, \delta, F >$, where Σ is the alphabet, D is a finite set of directions, Q is the set of states, $q_0 \in Q$ is the initial state, $\delta : Q \times \Sigma \rightarrow \mathcal{B}^+(D \times Q)$ is the transition function, where $\mathcal{B}^+(D \times Q)$ is the set of all positive Boolean combinations of pairs (d, q) with d direction and q state, and $F \subseteq Q$ is the set of the accepting states. An *ATA* A recognizes (finite) trees by means of runs. For a Σ-labeled tree $< T, V >$, with $T = D^*$, a run is a $(D^* \times Q)$-labeled N-tree $< T_r, r >$ such that the root is labeled with (ε, q_0) and the labels of each node and its successors satisfy the transition relation. A run is *accepting* if all its leaves are labeled with accepting states. An input tree is accepted if there exists a corresponding accepting run. By $L(A)$ we denote the set of trees accepted by A. We say that A is not empty if $L(A) \neq \emptyset$.

As a special case of alternating tree automata, we consider *nondeterministic tree automata* (*NTA*, for short), where the concurrency feature is not allowed. That is, whenever the automaton visits a node x of the input tree, it sends to each successor (direction) of x at most one copy of itself. More formally, an *NTA* is an *ATA* in which δ is in disjunctive normal form, and in each conjunctive clause every direction appears at most once.

Finally, *Alternating Büchi tree automata* (*BATA*, for short) are *ATA* accepting infinite trees. Precisely, a run is *accepting* if all its branches visit *infinitely often* at least one state belonging to F. As before, we also consider *nondeterministic Büchi tree automata* (*BNTA*, for short). We refer to [21] for a formal definition of *BATA*.

3 The Game Model

In this section, we introduce two-player turn-based games. Precisely, we consider games consisting of an arena coupled with an objective. The arena describes the configurations of the game through a set of states, being partitioned between the two players. In each state, only the player that owns it can take a move. The formal definition of the considered game model follows.

Definition 1. *A* two-player turn-based game *(2TG, for short), played between Player$_0$ and Player$_1$, is a tuple* $G \triangleq < \mathrm{St}, s_I, \mathrm{Ac}, \mathsf{tr}, \mathrm{W}, \mathsf{O} >$, *where* $\mathrm{St} \triangleq \mathrm{St}_0 \cup \mathrm{St}_1$ *is a finite non-empty set of* states, *with* St_i *being the set of states of Player$_i$,* $s_I \in \mathrm{St}$ *is a designated* initial state, $\mathrm{Ac} \triangleq \mathrm{Ac}_0 \cup \mathrm{Ac}_1$ *is the set of actions,* W *is a set of target states,* O *is the objective of Player$_0$, and* $\mathsf{tr} : \mathrm{St}_i \times \mathrm{Ac}_i \rightarrow \mathrm{St}_{1-i}$, *for* $i \in \{0, 1\}$ *is a* transition function *mapping a state of a player and its action to a state belonging to the other player.*

In a 2*TG* we only define the objective for *Player*$_0$ since, the objective for *Player*$_1$ is the opposite. In the following we only consider as objectives for *Player*$_0$ *safety* and *fairness*. Regarding the former, *Player*$_0$ wins the game if he has a strategy that prevents him from reaching all the states in $\mathrm{St} \setminus \mathrm{W}$. For the latter, *Player*$_0$ wins the game if he can induce plays along which he visits at least a target state infinitely often. These concepts will be formalized in the sequel.

To properly give the semantics of $2TG$s, we now introduce some basic concepts such as path, track, strategy, and play.

A *path* is a finite or infinite sequence of states $s_1, s_2, \dots$ such that $s_1 = s_I$ and for all i, if $s_i \in \mathrm{St_o}$ then there exists an action $a_0 \in \mathrm{Ac}_0$ such that $s_{i+1} = \mathsf{tr}(s_i, a_0)$, else there exists an action $a_1 \in \mathrm{Ac}_1$ such that $s_{i+1} = \mathsf{tr}(s_i, a_1)$.

A *track* $\rho \in \mathrm{St}^*$ is a finite path. For a track ρ, by $(\rho)_i$ we denote the i-st element of ρ, by $\rho_{\leq i}$ we denote the prefix track $(\rho)_0 \dots (\rho)_i$, and by $last(\rho)$ we denote the last element of ρ. By $\mathrm{Trk} \subseteq \mathrm{St}^*$, we denote the set of tracks over St. By Trk_i we denote the set of tracks ρ in which $last(\rho) \in \mathrm{St}_i$.

A *strategy* represents a scheme for a player containing a precise choice of actions along an interaction with the other player. It is given as a function over tracks. Formally, a *strategy* for $Player_i$ is a function $\sigma_i : \mathrm{Trk}_i \to \mathrm{Ac}_i$ that maps a track to an action.

The composition of strategies, one for each player in the game, induces a computation called *play*. Precisely, assume $Player_0$ and $Player_1$ take strategies σ_0 and σ_1, respectively. Their composition *induces* a play ρ such that $(\rho)_0 = s_I$ and for each $i \geq 0$ if $(\rho)_i \in \mathrm{St_o}$ then $(\rho)_{i+1} = \mathsf{tr}((\rho)_i, \sigma_0(\rho_{\leq i}))$, else $(\rho)_{i+1} = \mathsf{tr}((\rho)_i, \sigma_1(\rho_{\leq i}))$. A play ρ satisfies a safety objective, if and only if it contains only states in W. Conversely, let $inf(\rho)$ the set of states occurring infinitely often in ρ, the play satisfies the fairness objective if and only if $inf(\rho) \cap \mathrm{W} \neq \emptyset$.

A strategy is winning for a player if all the plays induced by composing such strategy with all the strategies of the adversarial player satisfies his objective. If such a winning strategy exists we say that the player wins the game. The formal definition of *winning condition* follows.

Definition 2. *Let G be a $2TG$. $Player_0$ wins the game G if he has a strategy such that for all strategies of $Player_1$ the resulting induced play satisfies* O.

4 Searching for Additional Winning Strategies

In this section, we show how to check whether $Player_0$ wins the game under safety and fairness objectives. To proper introduce our solutions procedure we first need to provide some auxiliary notation. Precisely, we introduce the concepts of *decision tree*, *strategy tree*, and *additional strategy tree.*

A decision tree simply collects all the tracks that come out from the interplays between the players. In other words, a decision tree can be seen as an unwinding of the game structure along with all possible combinations of player actions. The formal definition follows.

Definition 3. *Given a $2TG$ G, a* decision tree *is an* St*-labeled* Ac*-tree collecting all tracks over G.*

We now introduce strategy trees that allow to collect, for each fixed strategy for $Player_0$, all possible responding strategies for $Player_1$. Therefore, the strategy tree is a tree where each node labeled with $s \in \mathrm{St_o}$ has an unique successor determined by the strategy for $Player_0$ and each node labeled with $s \in \mathrm{St_1}$ has all possible successors determined by the actions of $Player_1$. Thus, a strategy tree is an opportune projection of the decision tree. The formal definition follows.

Definition 4. *Given a $2TG$ and a strategy σ for $Player_0$, a* strategy tree *for $Player_0$ is an* St*-labeled* Ac*-tree $< T, V >$, with $T \subset \mathrm{Ac}^*$ and V as follows: (i) $V(\varepsilon) = s_I$; (ii) for all $v \in T$, let $\rho = (\rho)_0 \ldots (\rho)_{|v|-1}$ be a track from s_I with $(\rho)_k = V(v_{\leq k})$ for each $0 \leq k \leq |v| - 1$, if $V(prf(v)) \in \mathrm{St}_o$ then $V(v) = \mathsf{tr}(V(prf(v)), \sigma(\rho))$, otherwise $V(v) = \mathsf{tr}(V(prf(v)), last(v))$.*

Following the above definition and Definition 2, given a $2TG$ G, $Player_0$ wins the game and $Player_1$ loses it by simply checking the existence of a strategy tree for $Player_0$, that is a tree such that each path satisfies the objective $\mathtt{O}$. Such a tree is called a *winning-strategy tree* for $Player_0$.

In case we want to ensure that at least two winning strategies exist then, at a certain point along the tree, $Player_0$ must take two successors. Let $succ : \mathrm{St} \to 2^{\mathrm{St}}$ to be the function that for each state $s \in \mathrm{St}$ in G gives the set of its successors, the formal definition of *additional strategy tree* follows.

Definition 5. *Given a $2TG$ G, an* additional strategy tree *for $Player_0$ is an* St*-labeled* Ac*-tree $< T, V >$ that satisfies the following properties:*

1. *the root node is labeled with the initial state s_I of G;*
2. *for each $x \in T$ that is not a leaf and it is labeled with state s of $Player_0$, it holds that x has as children a non-empty subset of $succ(s)$;*
3. *for each $x \in T$ that is not a leaf and it is labeled with state s of $Player_1$, it holds that x has as children the set of $succ(s)$;*
4. *there exists at least one $x \in T$ that corresponds to a state of $Player_0$ in G and it has at least two children.*

Note that, the above definition, but item 4, is the classical characterization of strategy tree. As before, given a $2TG$ G, $Player_0$ has an additional strategy to win the game if there is an additional strategy tree for him, that is a tree such that each path satisfies the objective $\mathtt{O}$. Such a tree is called an *additional winning-strategy tree* for $Player_0$.

Now, we have all ingredients to solve $2TG$ in which the objective is safety or fairness. For the former we have the following result.

Theorem 1. *Given $2TG$ G with safety objective it is possible to decide in linear time whether $Player_0$ has more than one strategy to win the game.*

Proof. Consider a $2TG$ G with safety objective. We know that $Player_0$ wins G iff there exists a strategy for $Player_0$ that for all strategies for $Player_1$ the induced play does not reach any state in $\mathrm{St} \setminus \mathrm{W}$.

We build an NTA A that accepts all additional winning-strategy for $Player_0$ over G. The automaton A uses $\mathrm{Q} = \mathrm{St} \times \{ok, split, bad\}$ as set of states, where *ok* and *split* are flags and the latter is used to remember that along the tree $Player_0$ has to ensure the existence of two winning strategies by opportunely choosing a point where to "split", and *bad* is used when a state $s \in \mathrm{St} \setminus \mathrm{W}$ is occurred. We set as alphabet $\Sigma = \mathrm{St}$ and initial state $q_0 = (s_I, split)$. For the transitions, starting from a state $q = (s, flag)$ and reading the symbol a, we have that:

$$\delta(q,a) = \begin{cases} (s', bad) & \text{if } s \in \mathrm{St} \setminus \mathrm{W}; \\ (s', ok) & \text{if } s = a \wedge s \in \mathrm{St}_0 \wedge flag = ok; \\ ((s', ok) \wedge (s'', ok)) \vee (s', split) & \text{if } s = a \wedge s \in \mathrm{St}_0 \wedge flag = split; \\ succ(s) \times \{ok\} & \text{if } s = a \wedge s \in \mathrm{St}_1 \wedge flag = ok; \\ (s_1, f_1) \wedge \cdots \wedge (s_n, f_n) & \text{if } s = a \wedge s \in \mathrm{St}_1 \wedge flag = split; \\ \emptyset & \text{otherwise.} \end{cases}$$

where $s', s'' \in succ(s)$ with $s' \neq s''$, $\{s_1, \ldots, s_n\} = succ(s)$, and $f_1, \ldots, f_n$ are flags in which there exists $1 \leq i \leq n$ such that $f_i = split$ and for all $j \neq i$, we have $f_j = ok$. The set of accepting states is $\mathrm{W} \times \{ok\}$.

The automaton checks if each path of an input tree does not reach a bad state. It is not hard to see that the automaton only needs to check the tree up to a depth of $|\mathrm{St}| + 1$. Indeed, if a bad state does not occur within this bound, $Player_0$ can pump good states forever through some cycles in the game. To limit the automaton to check up to such a bound it is sufficient to use a binary counter along its states. For the sake of readability we omit this part. Finally, it is not hard to see that if A is not empty then $Player_0$ has at least two strategies to win G.

Since, the size of the automaton A is just linear in the size of the game and checking its emptiness can be performed in linear time (from [32]), the desired complexity result follows. $\square$

Theorem 2. *Given 2TG G with fairness objective it is possible to decide in quadratic time whether $Player_0$ has more than one strategy to win the game.*

Proof. Consider a $2TG$ G with fairness objective. We know that $Player_0$ wins G iff there exists a strategy for $Player_0$ that for all strategies for $Player_1$ the induced play reaches infinitely often a state in W. In particular, now to handle the winning condition we need a non-deterministic Büchi tree automaton.

We build a *BNTA* B that accepts all trees that are witnesses of more than a winning strategy for $Player_0$ over G. The automaton A uses $\mathrm{Q} = \mathrm{St} \times \{ok, split\}$ as set of states, $\Sigma = \mathrm{St}$ as alphabet, and $q_0 = (s_I, split)$ as initial state. For the transitions, starting from $q = (s, flag)$ and reading the symbol a, we have that:

$$\delta(q,a) = \begin{cases} (s', ok) & \text{if } s = a \wedge s \in \mathrm{St}_0 \wedge flag = ok; \\ ((s', ok) \wedge (s'', ok)) \vee (s', split) & \text{if } s = a \wedge s \in \mathrm{St}_0 \wedge flag = split; \\ succ(s) \times \{ok\} & \text{if } s = a \wedge s \in \mathrm{St}_1 \wedge flag = ok; \\ (s_1, f_1) \wedge \cdots \wedge (s_n, f_n) & \text{if } s = a \wedge s \in \mathrm{St}_1 \wedge flag = split; \\ \emptyset & \text{otherwise.} \end{cases}$$

where $s', s'' \in succ(s)$ with $s' \neq s''$, $\{s_1, \ldots, s_n\} = succ(s)$, and $f_1, \ldots, f_n$ are flags in which there exists $1 \leq i \leq n$ such that $f_i = split$ and for all $j \neq i$, we have $f_j = ok$. The set of accepting states is $\mathrm{W} \times \{ok\}$. Hence, if B is not empty then $Player_0$ has at least two strategies to win G.

Since, the size of the automaton B is just linear in the size of the game and checking its emptiness can be performed in quadratic time (from [33]), the desired complexity result follows. □

5 Conclusion and Future Work

In this paper we have introduced a simple but effective automata-based methodology to check whether a player has more than one winning strategy in a two-player game under safety and fairness objectives. We believe that the solution algorithm we have conceived in this paper can be used as core engine to count strategies efficiently in more involved game scenarios and in many solution concepts reasoning as we plan to investigate as a continuation of this paper.

This work opens to several interesting questions and extensions, which we plan to investigate. An interesting direction is to consider the counting of strategies in multi-agent concurrent games. This kind of games have several interesting applications in artificial intelligence [34]. As another direction of work, one can consider some kind of hybrid game, where one can opportunely combine teams of players working concurrently with some others playing in a turn-based manner as in [17, 18, 29]. These games arise for example in case the interaction among the players behaves in a recursive way [8, 28].

References

1. R. Alur, T. Henzinger, and O. Kupferman. Alternating-Time Temporal Logic. *Journal of the ACM*, 49(5):672–713, 2002.
2. B. Aminof, V. Malvone, A. Murano, and S. Rubin. Extended graded modalities in strategy logic. In *SR 2016*, pages 1–14, 2016.
3. B. Aminof, V. Malvone, A. Murano, and S. Rubin. Graded strategy logic: Reasoning about uniqueness of nash equilibria. In *AAMAS 2016*, pages 698–706, 2016.
4. B. Aminof, A. Murano, and S. Rubin. On ctl* with graded path modalities. In *LPAR-20*, pages 281–296, 2015.
5. F. Baader, S. Borgwardt, and M. Lippmann. Temporal conjunctive queries in expressive description logics with transitive roles. In *AI 2015*.
6. A. Bianco, F. Mogavero, and A. Murano. Graded Computation Tree Logic. *Transactions On Computational Logic*, 13(3):25:1–53, 2012.
7. P. Bonatti, C. Lutz, A. Murano, and M. Vardi. The Complexity of Enriched muCalculi. *Logical Methods in Computer Science*, 4(3):1–27, 2008.
8. L. Bozzelli, A. Murano, and A. Peron. Pushdown Module Checking. *Formal Methods in System Design*, 36(1):65–95, 2010.
9. K. Chatterjee and M. Henzinger. An $O(n^2)$ time algorithm for alternating büchi games. In *SODA 2012*, pages 1386–1399, 2012.

10. J. B. D. Simchi-Levi, X. Chen. *The Logic of Logistics: Theory, Algorithms, and Applications for Logistics Management.* SBM. Springer, 2013.
11. A. Ferrante, A. Murano, and M. Parente. Enriched Mu-Calculi Module Checking. *Logical Methods in Computer Science*, 4(3):1–21, 2008.
12. A. Ferrante, M. Napoli, and M. Parente. Model Checking for Graded CTL. *Fundamenta Informaticae*, 96(3):323–339, 2009.
13. C. D. Fraser. The uniqueness of nash equilibrium in the private provision of public goods: an alternative proof. *Journal of Public Economics*, 49(3):389–390, 1992.
14. J. Gutierrez, G. Perelli, and M. Wooldridge. Iterated games with LDL goals over finite traces. In *AAMAS 2017*, pages 696–704, 2017.
15. D. Harel and A. Pnueli. *On the Development of Reactive Systems.* Springer, 1985.
16. N. Immerman. Number of Quantifiers is Better Than Number of Tape Cells. *Journal of Computer and System Science*, 22(3):384–406, 1981.
17. W. Jamroga and A. Murano. On Module Checking and Strategies. In *AAMAS 2014*, pages 701–708, 2014.
18. W. Jamroga and A. Murano. Module checking of strategic ability. In *AAMAS 2015*, pages 227–235, 2015.
19. O. Kupferman, U. Sattler, and M. Vardi. The Complexity of the Graded muCalculus. In *CADE 02*, LNCS 2392, pages 423–437. Springer, 2002.
20. O. Kupferman and M. Vardi. Module Checking Revisited. In *Computer Aided Verification'97*, LNCS 1254, pages 36–47. Springer, 1997.
21. O. Kupferman, M. Vardi, and P. Wolper. An Automata Theoretic Approach to Branching-Time Model Checking. *Journal of the ACM*, 47(2):312–360, 2000.
22. O. Kupferman, M. Vardi, and P. Wolper. Module Checking. *Information and Computation*, 164(2):322–344, 2001.
23. V. Malvone, F. Mogavero, A. Murano, and L. Sorrentino. Reasoning about graded strategy quantifiers. *Inf. Comput. (to appear).*
24. V. Malvone, F. Mogavero, A. Murano, and L. Sorrentino. On the counting of strategies. In *TIME 2015*, pages 170–179, 2015.
25. V. Malvone and A. Murano. Additional winning strategies in two-player games. In *ICTCS 2016*, pages 251–256, 2016.
26. V. Malvone, A. Murano, and L. Sorrentino. Games with additional winning strategies. In *CILC 2015*, pages 175–180, 2015.
27. F. Mogavero, A. Murano, G. Perelli, and M. Vardi. Reasoning About Strategies: On the Model-Checking Problem. *TOCL*, 15(4):34:1–42, 2014.
28. A. Murano and G. Perelli. Pushdown multi-agent system verification. In *IJCAI 2015*, pages 1090–1097, 2015.
29. A. Murano and L. Sorrentino. A game-based model for human-robots interaction. In *WOA'15*, pages 146–150, 2015.
30. G. Papavassilopoulos and J. B. Cruz. On the uniqueness of nash strategies for a class of analytic differential games. *JOTA*, 27(2):309–314, 1979.
31. L. Pavel. *Game Theory for Control of Optical Networks.* Science and Business Media. Springer, 2012.
32. W. Thomas. Infinite trees and automaton definable relations over omega-words. In *STACS'90*, pages 263–277, 1990.
33. M. Y. Vardi. Alternating automata and program verification. In *Computer Science Today*, pages 471–485. Springer, 1995.
34. M. Wooldridge. *An Introduction to Multi Agent Systems.* John Wiley & Sons, 2002.
35. Y. Zhang and M. Guizani. *Game Theory for Wireless Communications and Networking.* CRC Press, 2011.