View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by University of Liverpool Repository

The computational complexity of ecological and evolutionary spatial dynamics

Rasmus Ibsen-Jensen, IST Austria Krishnendu Chatterjee, IST Austria,
Martin A. Nowak, Harvard University

March 7, 2016

Abstract

There are deep, yet largely unexplored connections between computer science and biology. Both disciplines examine how information proliferates in time and space. Central results in computer science describe the complexity of algorithms that solve certain classes of problems. An algorithm is deemed efficient if it can solve a problem in polynomial time, which means the running time of the algorithm is a polynomial function of the length of the input. There are classes of harder problems for which the fastest possible algorithm requires exponential time. Another criterion is the space requirement of the algorithm. There is a crucial distinction between algorithms that can find a solution, verify a solution, or list several distinct solutions in given time and space. The complexity hierarchy that is generated in this way is the foundation of theoretical computer science. Precise complexity results can be notoriously difficult. The famous P=NP question is one of the hardest open problems in computer science and all of mathematics. Here we consider simple processes of ecological and evolutionary spatial dynamics. The basic question is: what is the probability that a new invader (or a new mutant) takes over a resident population? We derive precise complexity results for a variety of scenarios. We therefore show that some fundamental questions in this area cannot be answered by simple equations.

1 Significance

There is a deep connection between computer science and biology, as both fields study how information proliferates in time and space. In computer science, the space and time requirements of algorithms to solve certain problems generate complexity classes, which represent the foundation of theoretical computer science. The theory of evolution in structured population has provided an impressive range of results, but an understanding of the computational complexity of even simple questions is still missing. In this work we prove – unexpectedly – that some fundamental problems in ecological and evolutionary spatial dynamics can be precisely characterized by well-established complexity classes of the theory of computation. Since we show computational hardness for several basic problems, our results imply that the corresponding questions cannot be answered by simple equations. For example, there cannot be a simple formula for the fixation probability of a new mutant given frequency dependent selection in a structured population. We also show that some problems, such as calculating the molecular clock of neutral evolution in structured populations, admit efficient algorithmic solutions.

Evolutionary games on graphs — Fixation probability — Complexity classes

2 Introduction

Evolution occurs in populations of reproducing individuals. Mutation generates distinct types. Selection favors some types over others. The mathematical formalism of evolution describes how populations change in their genetic (or phenotypic) composition over time. Many papers study evolutionary dynamics in structured

Table 1: Complexity results for various models and computational questions

	Qualitative	Quantitative
Ecological Scenario	NP-complete	#P-complete
Linear fitness	PSPACE-complete	PSPACE-complete
Exponential fitness	P	PSPACE-complete

populations [1, 2, 3, 4, 5, 6, 7, 8]. Spatial structure can affect the rate of neutral evolution [9]. There are results that describe which spatial structures do or do not affect the outcome of constant selection [10, 11, 12]. Constant selection refers to a situation where the competing types have constant reproductive rates independent of the composition of the population. Some population structures can be amplifiers or suppressors of constant selection [13, 6, 14] meaning that they modify the intensity of selective differences. A large literature deals with evolutionary games [15, 16, 17, 18, 19] in structured populations [1, 20, 21, 22, 23, 24, 25, 26, 27, 28]. In evolutionary games the reproductive success of an individual depends on interactions with others. Many population structures and update rules can affect the outcome of evolutionary games. For example, spatial structure can favor evolution of cooperation [1, 29].

In this paper we are interested in stochastic evolutionary dynamics in populations of finite size. A typical setting is provided by evolutionary graph theory [6, 30, 31, 32, 33, 34]. The individuals of a population occupy the vertices of a graph. The links determine who interacts with whom for receiving payoff and for reproduction. There can be a single graph for game dynamical interaction and evolutionary replacement, or the interaction and replacement graphs can be distinct [35]. Often the graph is held constant during evolutionary updating, but it is also possible to study dynamically changing graphs [36, 37, 38, 39, 40, 41, 42, 43, 44].

The study of spatial dynamics also has a long tradition in ecology [45, 46, 47, 48, 49]. Here the typical setting is that different species compete for ecological niches. Many evolutionary models are formally equivalent to ecological ones - especially if we consider only selection and not mutation. Then we can interpret the different types as individuals of different species.

This paper is structured as follows. First we give an intuitive account of the foundation of theoretical computer science. We describe classes of problems that can be solved by algorithms in certain time and space constraints. Subsequently we present two simple problems of evolutionary dynamics in spatial settings. The first problem is motivated by a very simple ecological dynamic; the second problem is the general setting of evolutionary games on graphs. In both cases, the basic question is to calculate the take over probability (or fixation probability) of a new type. That is we introduce a new type in a random position in the population and we ask what is the complexity of an algorithm that can characterize the probability that the new type takes over the population (becomes fixed). Unexpectedly we are able to prove exact complexity results (see Table 1).

The class PTIME (denoted as P) consists of problems whose solutions can be computed by an algorithm that uses polynomial time. This means the running time of the algorithm grows as a polynomial function of the size of the input. In computer science, PTIME represents the class of problems which can be solved efficiently.

The class NP (non-deterministic polynomial time) consists of problems, for which solutions exist that are of polynomial length, and given a candidate for a solution of polynomial length, whether the candidate is indeed a solution can be checked in polynomial time. Therefore, an NP algorithm can verify a solution in polynomial time.

In order to proceed further, we need the notion of 'reduction' between classes of problems. A reduction, from a given problem P_1 to a problem P_2, is a translation such that a solution for P_2 can provide a solution for P_1. More precisely, if there is a polynomial-time reduction from P_1 to P_2, then a polynomial-time algorithm for P_2 implies a polynomial-time algorithm for P_1.

A given problem is NP-hard if for every problem in NP there is a polynomial reduction to the given problem. A problem is NP-complete, if it is both NP-hard, as well as there is an NP algorithm for the problem.

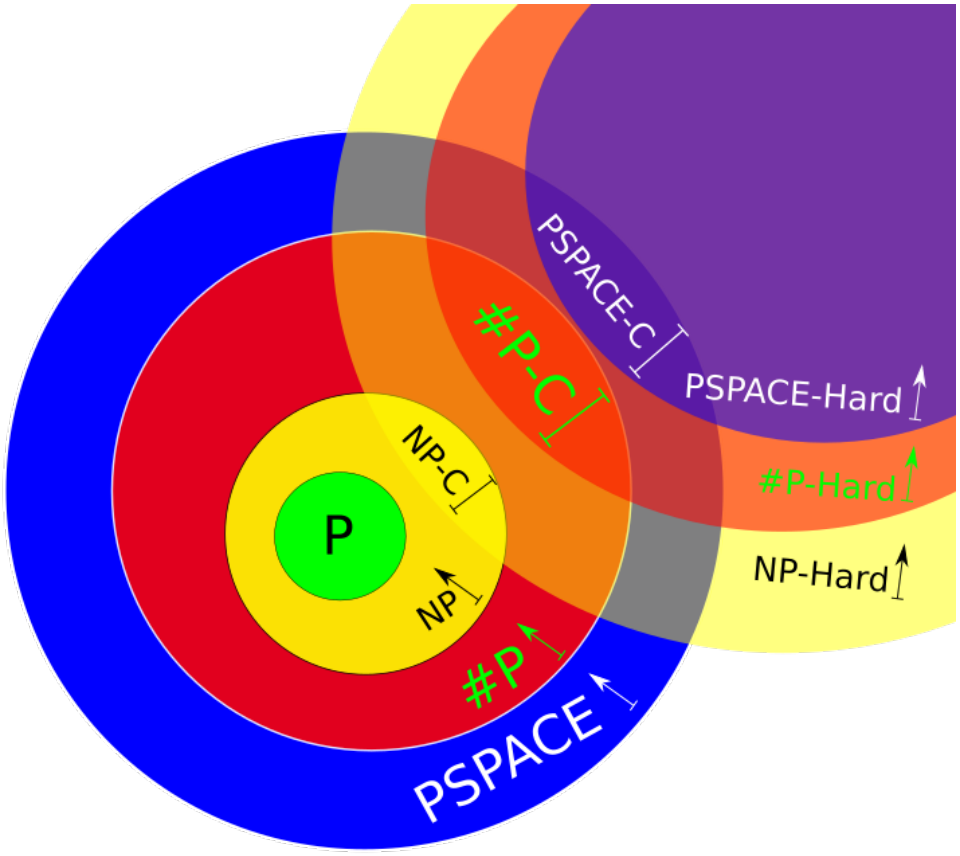

Figure 1: A pictorial illustration of the complexity classes P, NP, #P, and PSPACE. The complexity class P is contained in NP, NP is contained in #P, and #P is contained in PSPACE. The widely believed conjecture is that the complexity classes are different. A problem is NP-hard if it is at least as hard as each problem in NP; and similar for #P-hardness and PSPACE-hardness. The intersection of NP and NP-hard gives the NP-complete problems, and similarly for #P-complete and PSPACE-complete problems. Hence a polynomial-time solution for a NP-hard or NP-complete problem would imply P=NP.

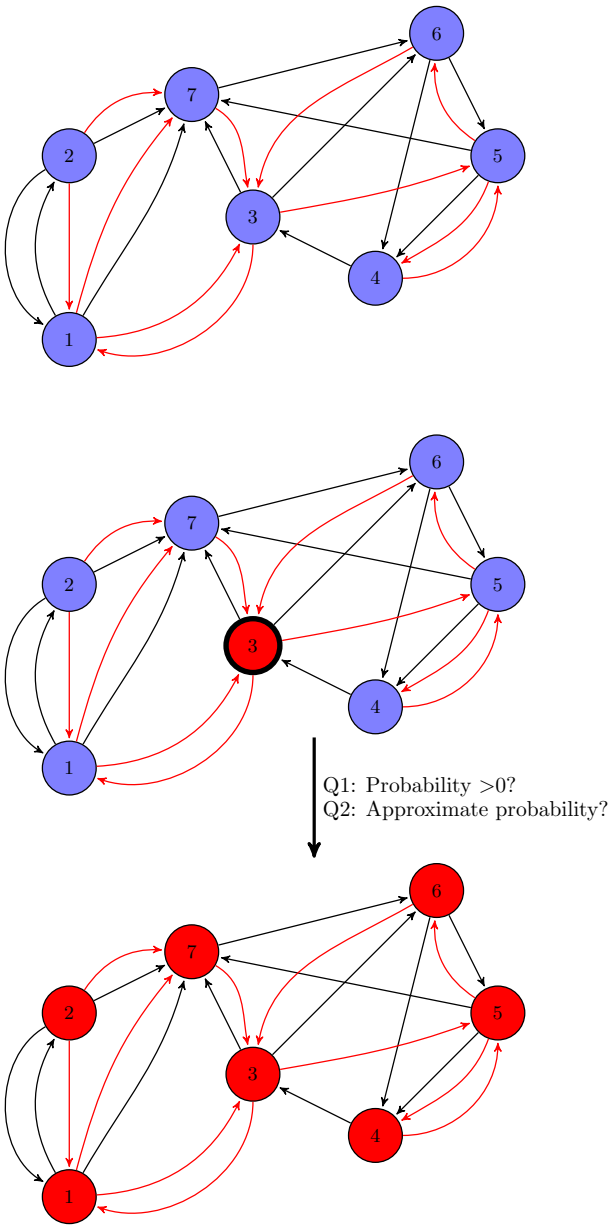

Figure 2: Illustration of mutant introduction. The residents (type A) are colored blue and the mutants (type B) are colored red. The black edges are the edges of the interaction graph and the red are the edges of the reproduction graph. The probability to introduce a mutant in a specific vertex is always one over the number of vertices. The computational questions of interest regarding the take over probability are as follows: whether the probability is positive (qualitative question), and compute an approximation of the probability (quantitative question).

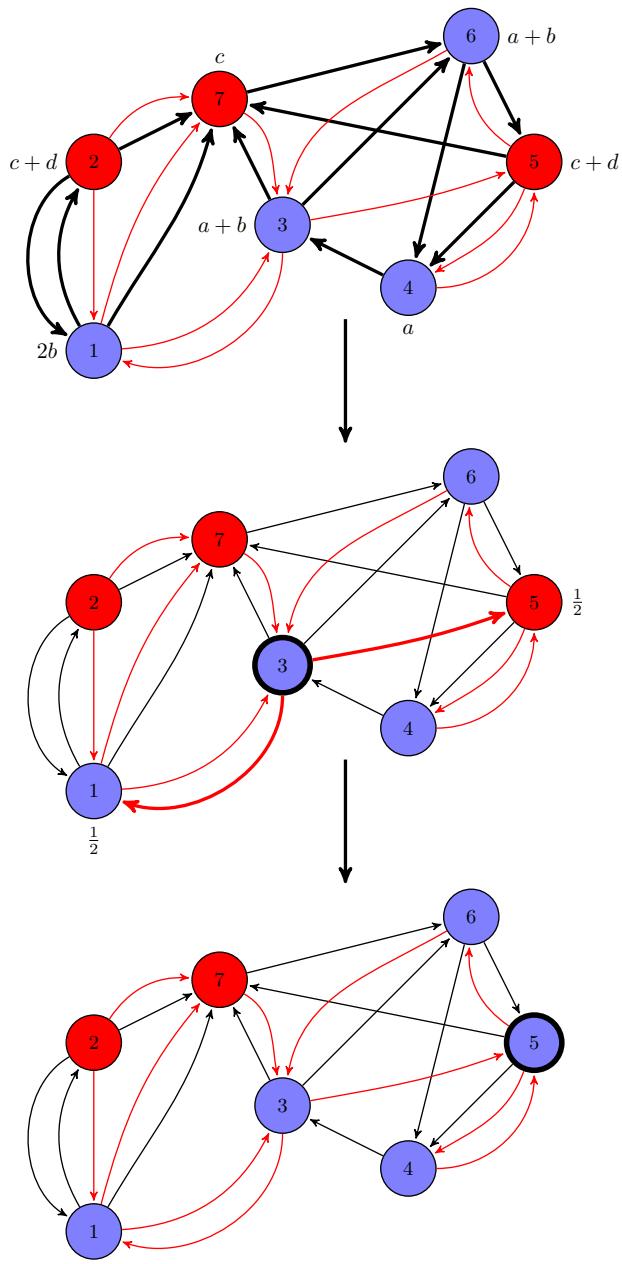

Figure 3: Illustration of reproduction with matrix $\begin{array}{cc} & \begin{array}{cc} A & B \end{array} \\ \begin{array}{c} A \\ B \end{array} & \begin{pmatrix} a & b \\ c & d \end{pmatrix} \end{array}$. The residents (type A) are colored blue and the mutants (type B) are colored red. The black edges are the edges of the interaction graph and the red are the edges of the reproduction graph. In the first figure beside each vertex the payoff of the vertex (which is the sum of the payoff of the interactions) is shown. Since the first figure shows the payoff computation, the interaction edges that are responsible for payoff calculation are boldfaced. In the second figure the vertex labeled 3 is selected for reproduction. The reproduction edges from vertex 3 are boldfaced, and each edge has probability 1/2. Finally, the successor 5 is chosen for replacement, i.e., vertex 3 reproduces to vertex 5.

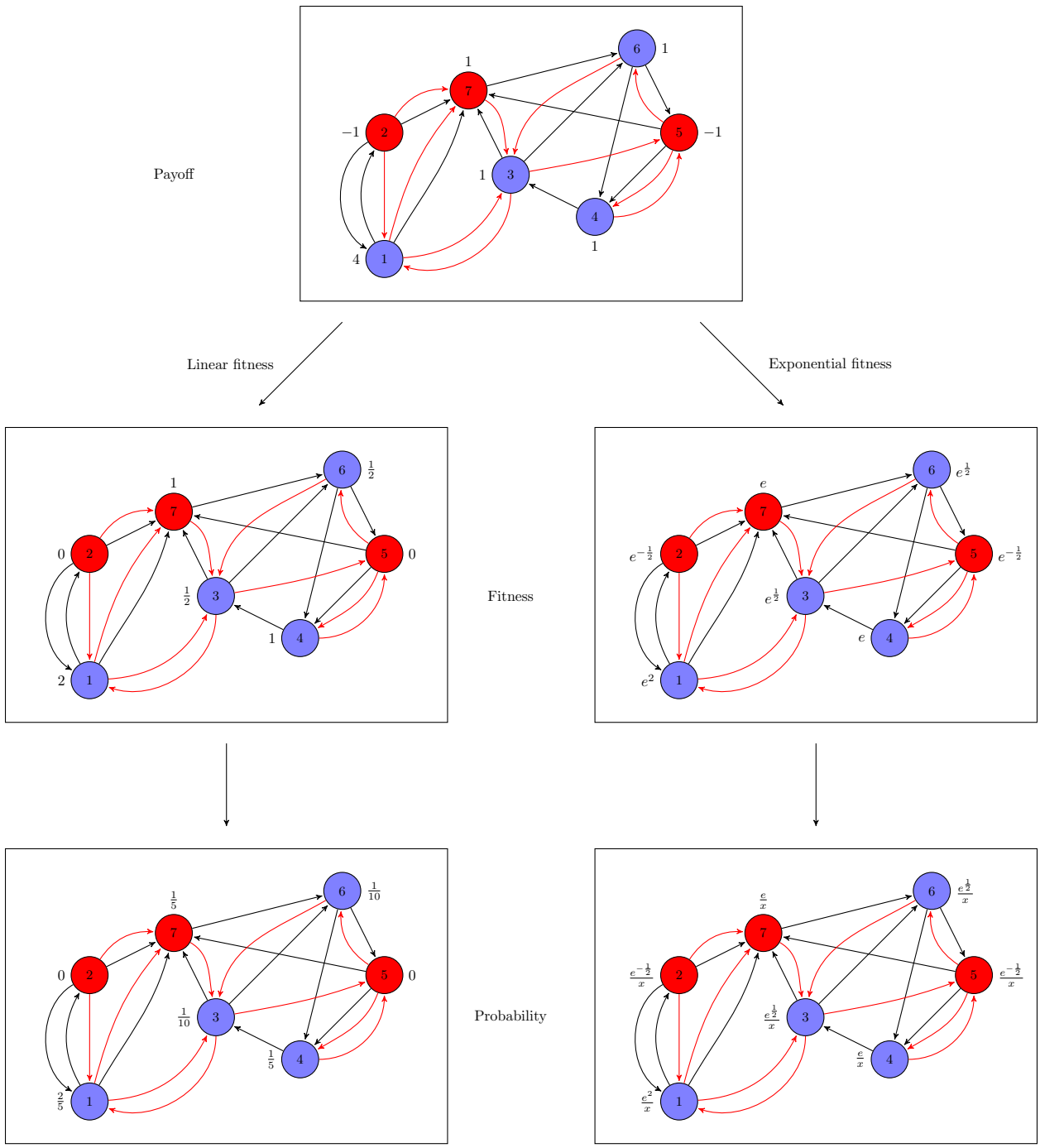

Figure 4: Illustration of different payoffs to fitness with $\begin{array}{c c} & \begin{array}{cc} A & B \end{array} \\ \begin{array}{c} A \\ B \end{array} & \begin{pmatrix} -1 & 2 \\ 1 & -2 \end{pmatrix} \end{array}$. The residents (type A) are blue and the mutants (type B) red. The black edges are the edges of the interaction graph and the red are the edges of the reproduction graph. In the figure of the first row we show the payoff for every vertex. In the next row we show the fitness which is either a linear function of the payoff but at least 0; or an exponential function of the payoff. Finally, in the third row, with each vertex we show the probability, which is the normalized fitness, that the vertex is selected for reproduction (in the last figure, the number x is the sum of the fitness, i.e., $x = e^2 + 2e + 2e^{\frac{1}{2}} + 2e^{-\frac{1}{2}}$).

For example, consider a Boolean formula over variables, and the question whether there exists an assignment to the variables such that the formula is true. A polynomial candidate solution is an assignment of truth values to variables, and given a candidate assignment the formula can be evaluated in polynomial time. This is the famous satisfiability, SAT, problem in computer science. The SAT problem is NP-complete.

The class P is contained in NP, and a major, long-standing open question in computer science is whether P=NP? A polynomial-time algorithm for an NP-complete (or an NP-hard) problem would imply that P=NP, resolving the long-standing open problem.

The class #P (sharpP) intuitively corresponds to counting the number of solutions. A problem is in #P if it counts the number of distinct solutions such that (i) every possible candidate for a solution is of polynomial length, and (ii) given a candidate for a solution, it can be checked in polynomial time whether the candidate is a solution. For example, given a Boolean formula, the problem whether there are at least k distinct satisfying assignments to the formula is a #P-problem. A given problem is #P-hard, if for every #P-problem there is a polynomial-time reduction to the given problem. A #P-complete problem is a problem that is both #P-hard, and there is a #P-solution. For example, counting the number of solutions in SAT is #P-complete.

The class NP is contained in #P because given the enumeration of solutions for #P, it is easy to check if there exists at least one solution. Intuitively, an NP problem asks whether there is at least one solution, whereas #P is the counting version which asks if there are least k distinct solutions (and the special case of $k = 1$ gives NP). Again a major open question is whether NP=#P? Note that a polynomial-time algorithm for a #P-complete problem would be an even bigger result as it would imply both P=NP and P=#P.

The class PSPACE consists of problems which can be solved with polynomial space. Note that a polynomial space algorithm can reuse space and can in general require exponential time. Every #P problem can be solved in PSPACE by simply enumerating each candidate for a solution and checking if it is a solution. Since we can reuse space to enumerate the candidates for solutions, the enumeration can be achieved in polynomial space. Moreover, every polynomial-time algorithm uses at most polynomial space. Hence it follows that #P is contained in PSPACE. The notion of PSPACE-hardness and PSPACE-completeness is similar to the notion of NP-hardness and NP-completeness, but with respect to the problems in PSPACE. Again a long-standing open question in computer science is whether #P=PSPACE, and a polynomial-time algorithm for a PSPACE-complete (or PSPACE-hard) problem would imply P=NP=#P=PSPACE.

We have mentioned that the major questions about the equality of the complexity classes are open problems, but the widely believed conjecture is that P is strictly contained in NP, NP is strictly contained in #P, and #P is strictly contained in PSPACE. In other words, it is widely believed that NP-complete problems cannot be solved in polynomial time, #P-complete problems are harder than NP-complete problems, and PSPACE-complete problems are harder than #P-complete problems. A pictorial illustration of the complexity classes is shown in Figure 1.

3 Results

The first problem is motivated by ecological dynamics. There is an ecosystem occupied by resident species. The spatial structure of the ecosystem is given by a graph. An invading species is introduced (see Figure 2 for an illustration). We assume the invading species has a competitive advantage in the sense that once a position is occupied by the invading species the resident cannot get it back. The invading species, however, has a density constraint: if the number of invaders around a focal invader is above a threshold, h, then the invader in the focal vertex can not colonize another vertex.

We are interested in the probability that the invader starting from a random initial position will take over the entire ecosystem (and therefore drive the resident to extinction). There are two types of questions. The 'qualitative question' is whether the take over probability is greater than zero. The 'quantitative question' is concerned with computing the take over probability subject to a small error. Figure 2 gives a pictorial illustration. We prove the following results. The qualitative question is NP-complete (See SI Appendix Theorem 4). The quantitive question is #P-complete (See SI Appendix Theorem 8).

The second problem is concerned with evolutionary games in structured populations. There are two

types, A and B, whose reproductive rates depend on local interactions. We consider the setting of games on graphs. Each vertex is occupied by one individual, which is either A or B. Interactions occur pairwise with all neighbors. The payoff matrix is given by

$$\begin{array}{cc} & \begin{array}{cc} A & B \end{array} \\ \begin{array}{c} A \\ B \end{array} & \begin{pmatrix} a & b \\ c & d \end{pmatrix} \end{array} \tag{1}$$

The entries of the payoff matrix can be positive or negative (or zero). Each individual interacts with all of its neighbors on the graph to derive a payoff sum. The payoff sum is translated into reproductive success as follows. If the payoff sum is positive, then the fecundity equals the payoff sum. If the payoff sum is negative, then the fecundity is zero. We refer to this translation as linear fitness. In any one time step, a random individual is chosen for reproduction proportional to its fecundity. The offspring, which is of the same type as the parent, is placed into an adjacent position on the graph (see Figure 3 and Figure 4 for an illustration).

We are interested in the probability that a single A individual starting in a random position on the graph generates a lineage which will take over the entire population; this probability is generally called fixation probability. As before, there are two types of questions. The 'qualitative question' is whether the fixation probability is positive. The 'quantitative question' is concerned with computing the fixation probability subject to a small error. We prove the following results. The qualitative question is NP-hard and in PSPACE. The quantitive question is #P-hard and in PSPACE. The results follow from SI Appendix Theorem 4, Theorem 8, and Theorem 15.

Note that the first problem can also be obtained as a special case of the second problem. In the payoff matrix (1) we can set, for example, $a = -1$, $b = 1$, $c = d = 0$. This 'game' has the property that type B never reproduces and type A reproduces until half its neighbors are also of type A. This parameter choice leads to the same qualitative behavior and the same complexity bounds as described in the first problem.

A generalization of games on graphs is the setting where the interaction graph and the replacement graph are distinct [35]. Thus each individual interacts with all of its neighbors on the interaction graph to receive payoff. Subsequently an individual is chosen for reproduction proportional to its fecundity. The offspring is placed randomly among all neighbors of the focal individual on the replacement graph. In this case, both the qualitative and quantitative questions become PSPACE-complete (See SI Appendix Theorem 15)

We also consider a variation of the second problem. In particular we change the mapping from payoff to fecundity. We now assume that fecundity is an exponential function of payoff, and refer to it as exponential fitness (see Figure 4 for an illustration). Therefore the fecundity of an individual is always positive (even if its payoff sum is negative). In this setting the qualitative question can be decided in polynomial time. The reason is that the fixation probability is positive if the graph is connected. Thus, in order to answer the qualitative question the algorithm only needs to check whether the graph is connected; this problem is in P. However, the quantitative question has the same complexity as the previous problem (See SI Appendix Theorem 16 and Theorem 17).

A very special case of games on graphs is constant selection. Type A has constant fecundity a and type B has constant fecundity b independent of any interactions, i.e., fecundity is independent of the population structure. The qualitative question concerning the fixation probability of A is in P. The quantitative question is in PSPACE, but any non-trivial lower bound is an open question.

Finally, while we establish computational hardness for several problems, we also show that two classic problems can be solved in polynomial time (See SI Appendix Section 7). First, we consider the molecular clock, which is the rate at which neutral mutations accumulate over time. The molecular clock is affected by population structure [35]. We show that the molecular clock can be computed in polynomial time because the problem reduces to solving a set of linear equalities, which can be achieved in polynomial time using Gaussian elimination. Second, we consider evolutionary games in a well-mixed population structure, where the underlying structure is the complete graph [51]. We show that the exact fixation probability can be computed in polynomial time. In this case the problem can be reduced to computing absorption probabilities in Markov chains, where each state represent the number of mutants. Hence the Markov chain is linear in the number of vertices of the graphs, and since absorption probabilities in Markov chains can be

computed in polynomial time (by solving a set of linear equalities) we obtain the desired result.

4 Methods: Proof Ideas

We now present the key intuition and main ideas of our results. The most interesting and technically insightful results are the lower bounds (i.e., the hardness proofs), and we present the key ideas only for them.

1. *NP-hardness of the qualitative ecological problem.* One of the most classic NP-complete problems is the 3SAT-problem, which is the SAT problem where every clause has exactly three literals (a literal is a Boolean variable x or a negation of a variable $\overline{x}$). Given instances of the 3SAT problem we construct instances of the ecological problem, where we have a start vertex where the mutant arises, followed by a sequence of vertices (i.e., each vertex can reproduce a mutant to the next), one for each clause. By our construction, a vertex in this sequence can reproduce at most three times, one of which must be the next vertex of the sequence, and the others correspond to at most two literals of the clause (intuitively these two lierals represent the ones that are not set to true by a candidate satisfying assignment). The last vertex of the sequence reproduces to a new sequence of vertices that corresponds to an assignment of truth values to the variables. Each vertex in this new sequence can reproduce twice, one to the next vertex of the sequence, and other to a variable or its negation. The variables or the corresponding negation can then reproduce mutants to the corresponding literals of the clauses. After this sequence, all vertices that do not correspond to a literal in a clause become mutants. In essence our construction ensures if there is a satisfying assignment, then with positive probability all vertices can become mutants, and conversely, if there is no satisfying assignment, then the probability that all vertices become mutants is zero.

2. *#P-hardness of the quantitative ecological problem.* A #P-complete problem is counting the number of *perfect matchings* in a bipartite graph (which also corresponds to computing permanent of Boolean matrices). A bipartite graph consists of two set of vertices, a set on the left side and a set on the right side, with edges from the left side to right side. A perfect matching is a one-to-one mapping of each vertex of the left side to a vertex on the right side such that there is an edge between them. First we argue that for the hardness proof it suffices to consider bipartite graphs in which each vertex in the left side has outdegree 2^k for some integer k. A key idea in our construction is that in a full binary tree, if the root becomes a mutant, and every vertex can reproduce exactly once, then the set of mutants will eventually consists of a path from the root to a leaf, chosen uniformly at random. Our construction is then as follows: We have a start vertex where the mutant arises, which reproduces to turn each of the vertices in the left side of the bipartite graph to mutants. Each of the vertices in the left side is the root of a full binary tree, where the leaves correspond to the right side of the bipartite graph. We show that the fixation process corresponds to a perfect matching (defined from the path in the full binary trees), and given an approximation of the fixation probability, the exact number of perfect matchings of the bipartite graph can be computed.

3. *PSPACE-hardness for the game on evolutionary graph problem.* Our PSPACE-hardness proof shows that the evolutionary process can solve the following *concurrent-if problem*, which we show is PSPACE-hard. The concurrent-if problem consists of a set of Boolean variables $x_1, x_2, \ldots, x_n$, with a given initial truth assignment to the variables, and a set of if-statements. Each if-statement s_i is of the follwing form: If a conjunctive clause C_i over the variables is true, then assign a truth value to a variable (e.g., if $(x_2 \wedge x_4 \wedge \overline{x}_5)$, then x_3 is assigned false). The problem is to decide whether the first variable (which is the accepting variable) eventually becomes true. We show that each variable can be represented as four vertices, and each if statement as a single vertex, in the evolutionary graph, and the evolutionary process can mimic the execution of the concurrent-if problem. Finally, if the accepting variable becomes true, then it corresponds to making a special vertex in the evolutionary graph as mutant. There exists a part of the evolutionary graph that can only become mutants after the special vertex has become a

mutant. Using this construction we show that for evolutionary games on graphs both the qualitative and quantitative problems are PSPACE-hard.

5 Discussion

In summary, we have established computational complexity results for some fundamental problems of ecological and evolutionary dynamics in structured populations. Our main results are summarized in Table 1. We now discuss the significance of our findings.

1. *Interdisciplinary connection.* While both computer science and biology examine the proliferation of information in time and space, the deep connection between them has been largely unexplored. Our work provides precise computational complexity results for several well-studied problems in biology and can be viewed as a step to establish a connection between the two disciplines.

2. *Well-studied open problem.* The problems we have considered are basic aspects of well-studied questions for ecological and evolutionary dynamics in structured populations [1, 5, 6, 14, 28, 29]. Several reviews have been written on this topic [7, 30, 33, 50]. We first discuss the significance of an algorithmic approach in evolutionary graph theory. An efficient algorithm, which considers all (even worst-case) graphs for evolutionary processes, is important for the following reasons: (i) It has been shown that some population structures (called amplifiers) can increase the effect of natural selection [6, 29]; but amplifiers are rare and constructing them is difficult [6, 29, 14, 52, 53, 33]. If there was an efficient algorithmic approach that worked for all graphs, then one could design candidates for amplifiers and efficiently check their fixation probabilities. Since there exists no algorithmic approach, research has to focus on special classes of graphs to identify simple formulas, such as calculating the fixation probabilities on star like graphs [52]. (ii) It is known that some population structures and evolutionary dynamics promote evolution of cooperation, but others do not [29]. An important open problem is to characterize the set of graphs that promote cooperation. An efficient algorithmic approach would be useful to check candidate structures. Since no efficient algorithm exists, one has to study special cases, for example by considering nearly regular graphs [29].

   Thus a general algorithmic approach is a very important problem for the well-studied question concerning the effect of population structures on evolutionary dynamics. An algorithmic approach has been studied for important special cases such as for complete graphs [54], and NP-hardness was stated for the quantitative problem [6]. The review [50] identifies the complexity of computing fixation probabilities on evolutionary graphs as important open question in the area. In [50], two open problems (2.1 and 2.2) are identified that ask for the complexity of computing the exact fixation probabilities for graphs and for games on graphs. Our results not only present answers to those crucial questions, but also show that both the approximation problem and the qualitative question are computationally hard. The most interesting aspects of our results are the lower bounds, which show that in most cases there exists no efficient algorithm, under the widely believed conjecture that P is different from NP. A simple equation based solution would give an efficient algorithm, and thus our result formally shows that for evaluating the fixation probability in spatial settings there does not exist a simple equation based solution in general. Our results are significant for the following reasons: (a) it establishes the computational complexity for fundamental problems of ecological and evolutionary dynamics in structured populations considered in for example [1, 5, 6, 14, 28, 29, 7, 30, 33, 50]; and (b) it significantly improves the complexity result of [6] and solves the computational complexity questions of the area as identified in [50].

3. *Methodological insight.* Our proof ideas also reveal some important points. We show how evolutionary processes in structured populations can mimic aspects of computation. This insight could be useful for future research on understanding the computational complexities of other stochastic processes on population structures.

References

[1] Nowak MA, May RM (1992) Evolutionary games and spatial chaos. *Nature* 359:826.

[2] Killingback T, Doebeli M (1996) Spatial evolutionary game theory: Hawks and doves revisited. *Proc. R. Soc. B* 263(1374):1135–1144.

[3] Szabó G, Toke C (1998) Evolutionary prisoner's dilemma game on a square lattice. *Phys. Rev. E.* 58(1):69–73.

[4] Szabó G, Hauert C (2002) Phase transitions and volunteering in spatial public goods games. *Phys. Rev. Lett.* 89:118101.

[5] Hauert C, Doebeli M (2004) Spatial structure often inhibits the evolution of cooperation in the snowdrift game. *Nature* 428:643.

[6] Lieberman E, Hauert C, Nowak MA (2005) Evolutionary dynamics on graphs. *Nature* 433(7023):312–316.

[7] Nowak MA, Tarnita CE, Antal T (2009) Evolutionary dynamics in structured populations. *Phil Trans R Soc B* 365(1537):19–30.

[8] Allen B, Tarnita CE (2012) Measures of success in a class of evolutionary models with fixed population size and structure. *J. Math. Biol* pp. 1–35.

[9] Allen B et al. (2015) The molecular clock of neutral evolution can be accelerated or slowed by asymmetric spatial structure. *PLoS Comput Biol* 11(2):e1004108+.

[10] Adlam B, Nowak MA (2014) Universality of fixation probabilities in randomly structured populations. *Sci Rep* 4:6692.

[11] Maruyama T (1974) A Markov process of gene frequency change in a geographically structured population. *Genetics* 76(2):367–377.

[12] Barton NH (1993) The probability of fixation of a favoured allele in a subdivided population. *Genetics Research* 62:149–157.

[13] Nowak MA, Michor F, Iwasa Y (2003) The linear process of somatic evolution. *Proc Natl Acad Sci USA* 100(25):14966–14969.

[14] Nowak MA (2006) *Evolutionary Dynamics.* (Harvard University Press).

[15] Smith JM (1982) *Evolution and the Theory of Games.* (Cambridge University Press, Cambridge, UK).

[16] Hofbauer J, Sigmund K, eds. (1988) *The theory of evolution and dynamical systems: Mathematical aspects of selection.* (Cambridge University Press, Cambridge).

[17] Hofbauer J, Sigmund K (1998) *Evolutionary Games and Population Dynamics.* (Cambridge University Press, Cambridge).

[18] Cressman R (2003) *Evolutionary dynamics and extensive form games*, Economic learning and social evolution. (MIT Press, Cambridge (Mass.)).

[19] Broom M, Rychtář J (2013) *Game-Theoretical Models in Biology.* (Chapman and Hall/CRC Math. Comput. Biol. Ser.).

[20] Ellison G (1993) Learning, local interaction, and coordination. *Econometrica* 61(5):1047–1071.

[21] Herz AV (1994) Collective phenomena in spatially extended evolutionary games. *J. Theor. Biol.* 169:65 – 87.

[22] Nakamaru M, Nogami H, Iwasa Y (1998) Score-dependent fertility model for the evolution of cooperation in a lattice. *J. Theor. Biol.* 194:101 – 124.

[23] Szabó G, Antal T, Szabó P, Droz M (2000) Spatial evolutionary prisoner's dilemma game with three strategies and external constraints. *Phys. Rev. E* 62:1095+.

[24] Kerr B, Riley MA, Feldman MW, Bohannan BJM (2002) Local dispersal promotes biodiversity in a real-life game of rock-paper-scissors. *Nature* 418:171–174.

[25] Helbing D, Yu W (2008) Migration as a Mechanism to Promote Cooperation. *Advances in Complex Systems* 11:641–652.

[26] Tarnita CE, Ohtsuki H, Antal T, Fu F, Nowak MA (2009) Strategy selection in structured populations. *J. Theor. Biol.* 259:570 – 581.

[27] Perc M, Szolnoki A (2010) Coevolutionary gamesa mini review. *Biosystems* 99:109 – 125.

[28] van Veelen M, García J, Rand DG, Nowak MA (2012) Direct reciprocity in structured populations. *P. Natl. Acad. Sci. USA* 109:9929–9934.

[29] Ohtsuki H, Hauert C, Lieberman E, Nowak MA (2006) A simple rule for the evolution of cooperation on graphs and social networks. *Nature* 441(7092):502–505.

[30] Szabó G, Fáth G (2007) Evolutionary games on graphs. *Phys. Rep.* 446:97–216.

[31] Yang HX, Wu ZX, Du WB (2012) Evolutionary games on scale-free networks with tunable degree distribution. *EPL (Europhysics Letters)* 99(1):10006.

[32] Chen YT (2013) Sharp benefit-to-cost rules for the evolution of cooperation on regular graphs. *Ann. Appl. Probab.* 23:637–664.

[33] Allen B, Nowak MA (2014) Games on graphs. *EMS Surv Math Sci* 1:113–151.

[34] Débarre F, Hauert C, Doebeli M (2014) Social evolution in structured populations. *Nat Commun* 5.

[35] Ohtsuki H, Pacheco JM, Nowak MA (2007) Evolutionary graph theory: breaking the symmetry between interaction and replacement. *J. Theor. Biol.* 246:681–694.

[36] Skyrms B, Pemantle R (2000) A dynamic model of social network formation. *P. Natl. Acad. Sci. USA* 97(16):9340–9346.

[37] Pacheco JM, Traulsen A, Nowak MA (2006) Coevolution of strategy and structure in complex networks with dynamical linking. *Phys. Rev. Lett.* 97:258103.

[38] Fu F, Hauert C, Nowak MA, Wang L (2008) Reputation-based partner choice promotes cooperation in social networks. *Phys. Rev. E* 78:026117.

[39] Antal T, Ohtsuki H, Wakeley J, Taylor PD, Nowak MA (2009) Evolution of cooperation by phenotypic similarity. *P. Natl. Acad. Sci. USA* 106(21):8597–8600.

[40] Tarnita CE, Antal T, Ohtsuki H, Nowak MA (2009) Evolutionary dynamics in set structured populations. *P. Natl. Acad. Sci. USA* 106(21):8601–8604.

[41] Szolnoki A, Perc M (2009) Resolving social dilemmas on evolving random networks. *EPL (Europhysics Letters)* 86(3):30007.

[42] Cavaliere M, Sedwards S, Tarnita CE, Nowak MA, Csikász-Nagy A (2012) Prosperity is associated with instability in dynamical networks. *Journal of Theoretical Biology* 299(0):126 – 138.

[43] Rand DG, Arbesman S, Christakis NA (2011) Dynamic social networks promote cooperation in experiments with humans. *P. Natl. Acad. Sci. USA* 108(48):19193–19198.

[44] Wu B et al. (2010) Evolution of cooperation on stochastic dynamical networks. *PLoS ONE* 5(6):e11187.

[45] Durrett R, Levin S (1994) The importance of being discrete (and spatial). *Theor. Popul. Biol* 46(3):363 – 394.

[46] Levin SA, Paine RT (1974) Disturbance, patch formation, and community structure. *P. Natl. Acad. Sci. USA* 71(7):2744–2747.

[47] Hassell MP, Comins HN, May RM (1994) Species coexistence and self-organizing spatial dynamics. *Nature* 370(6487):290–292.

[48] Tilman D, Kareiva PM (1997) *Spatial ecology. The role of space in population dynamics and interspecific interactions.* (Princeton University Press).

[49] Dieckmann U, Law R, Metz JAJ, eds. (2000) *The Geometry of Ecological Interactions.* (Cambridge University Press).

[50] Shakarian P, Roos P, Johnson A (2012) A review of evolutionary graph theory with applications to game theory. *Biosystems* 107(2):66 – 80.

[51] Nowak MA, Sasaki A, Taylor C, Fudenberg D (2004) Emergence of cooperation and evolutionary stability in finite populations. *Nature* 428(6983):646–650.

[52] Adlam B, Chatterjee K, Nowak MA (2015) Amplifiers of selection. *Proc. R. Soc. A.*

[53] Diaz J, Goldberg LA, Mertzios GB, Richbery D, Serna M, Spirakis PG (2013) On the fixation probability of superstars. *Proc. R. Soc. A.* 469(2156).

[54] Diaz J, Goldberg LA, Mertzios GB, Richbery D, Serna M, Spirakis PG (2014) Approximating fixation probabilities in the generalized Moran process. *Algorithmica* 69(1):78–91.

Supplementary Information: The computational complexity of ecological and evolutionary spatial dynamics

Rasmus Ibsen-Jensen[†] Krishnendu Chatterjee[†] Martin A. Nowak[‡]

[†] IST Austria

[‡] PED, Harvard University

1 Introduction and Organization

In this supplementary information we will present the detailed proofs of the results mentioned in the main text. To present a uniform treatment of the results we will consider the following notations:

1. We will always consider that there are two types of individuals that occupy the vertices of the graph, and call them as mutants and residents (they represent type A and type B individuals, respectively, as mentioned in the main article).

2. To model the ecological scenario, that the mutants has an advantage that once they occupy a position, then the residents cannot win it over, we will model it as the case that the residents do not reproduce (note that a residents reproducing to another vertex which is a resident does not change the scenario of the graph).

Organization of the results. The supplementary information is organized as follows:

1. We start with the formal definitions of the model and the computational questions in Section 2. We will consider two functions to change payoffs to fitness, namely, linear bounded fitness (where the fitness is linear function of the payoff, but at least 0), and the exponential fitness function. In the three following sections after Section 2 we present results about the linear bounded fitness model.

2. In Section 3 and Section 4, we consider linear bounded fitness and no resident reproduction (that models the ecological scenario with advantage for the mutants). We establish NP-completeness of the qualitative question in Section 3, and #P-completeness of the quantitative question in Section 4.

3. In Section 5 we consider linear bounded fitness with resident reproduction, and show that both the qualitative and quantitative questions are PSPACE-complete.

4. We consider the exponential fitness function in Section 6. We show that the quantitative question for no resident reproduction, and the qualitative question (even with resident reproduction) can be solved in polynomial time. We show that the quantitative problem with resident reproduction is PSPACE-complete.

5. Finally, in Section 7 we argue that evolutionary games on well-mixed population, and finding the rate of molecular clock problem can be solved in polynomial time.

2 Models of Evolution on Graphs

In this section we present the basic definitions related to the different models of evolution on graphs and the basic computational questions.

Evolutionary graphs. An *evolutionary graph* $G = (V, E_I, E_R)$ consists of a finite set V of vertices; a set $E_I \subseteq V \times V$ of *interaction* edges; and a set $E_R \subseteq V \times V$ of *replacement* (or reproduction) edges [12]. The sets E_I and E_R consist

of directed edges, and the graph $G_I = (V, E_I)$ is called the interaction graph, and $G_R = (V, E_R)$ is called the replacement graph. The graph G_I is responsible for determining the interaction of individuals in the graph (which affects the fitness or payoff), and the graph G_R captures the underlying structure for reproduction and replacement of individuals in the graph. Given an edge (v, u) we say u is a *successor* of v and v is a *predecessor* of u.

Payoff of individuals. Each vertex of the graph will be occupied by one of two types of individuals, namely, the *resident* type and the *mutant* type. In evolutionary games, along with the evolutionary graph there is a payoff matrix, which is defined as follows:

$$\begin{array}{cc} & \begin{array}{cc} R & M \end{array} \\ \begin{array}{c} R \\ M \end{array} & \begin{pmatrix} a & b \\ c & d \end{pmatrix} \end{array}$$

where the entries of the matrix are rational numbers and represent the payoff of an interaction, i.e., a (resp., b) is the payoff of a resident type interacting with another resident (resp., mutant) type, and c (resp., d) is the payoff of a mutant type interacting with a resident (resp., mutant) type. Given two vertices, x and y, we denote by $\mathsf{pay}(x, y)$ the payoff of the type of vertex x versus the type of vertex y.

Fitness of individuals. The fitness of an individual denotes the fecundity (or reproductive rate) and must be a non-negative number. Let $E_I(v) = \{u \mid (v, u) \in E_I\}$ denote the set of interaction successors of v. We define two natural (but not equivalent) ways of defining the fitness of v, denoted as $f(v)$, as follows:

1. *Linear bounded fitness.* The linear bounded fitness is the average payoff of the interactions but at least 0, i.e.,

$$f(v) = \max\left\{\frac{\sum_{u \in E_I(v)} \mathsf{pay}(v, u)}{|E_I(v)|}, 0\right\}.$$

   Note that since the fitness is non-negative it is bounded from below by 0.

2. *Exponential fitness.* The exponential fitness is an exponential function of the average payoff of the interactions, i.e.,

$$f(v) = \exp\left(\frac{\sum_{u \in E_I(v)} \mathsf{pay}(v, u)}{|E_I(v)|}\right).$$

   Note that the fitness function ensures that the fitness is always positive.

We will use LBF to refer to the linear bounded fitness function and ExF to refer to the exponential fitness function.

The evolutionary process. The evolutionary process we consider is the classical *birth-death* process on an evolutionary graph defined as follows:

1. Initially all vertices of the graph are of the resident type and a mutant type is introduced uniformly at random at one of the vertices of the graph.

2. Repeat the following step (referred to as a *generation*): In every generation, a vertex v is selected proportional to the fitness of the individual at the vertex to reproduce[1]. A new born individual replaces one of the replacement successors of v, i.e., it replaces a vertex chosen uniformly at random from the set $E_R(v) = \{u \mid (v, u) \in E_R\}$.

Step 2 (or generations) is repeated until nothing can change (in particular, if all vertices have fitness 0 or have the same type, then nothing can change).

Fixation probability. The most relevant question from an evolutionary perspective is the *fixation probability* which is the probability that the mutant takes over the population, i.e., eventually all vertices become the mutant type.

Computational questions. Given an evolutionary graph, a payoff matrix, and the payoff to fitness function (linear bounded, or exponential) we consider the following questions:

1. the *qualitative* decision question asks whether the fixation probability is positive; and

[1] If every vertex has fitness 0, then no vertex is selected for reproduction.

2. the *quantitative approximation* question, given $\epsilon > 0$, asks to compute an approximation of the fixation probability within an additive error of ϵ.

In this work we will establish several complexity bounds for the problem, and our most interesting results are the lower bounds. Lower bounds establish computational hardness of a problem, and if the lower bounds can be established even in restricted cases, then it shows that even special cases of the general problem is computationally hard, and thus the lower bounds become even more significant (e.g., a single lower bound for a special case can be applied to all generalizations of the special case).

Special cases. There are several special cases of interest that we will explore.

1. **Constant fitness with density constraints.** A special case of the payoff matrix is the *constant fitness* (aka constant selection) matrix defined as follows:

$$\begin{array}{cc} & \begin{array}{cc} R & M \end{array} \\ \begin{array}{c} R \\ M \end{array} & \begin{pmatrix} r & r \\ 1 & 1 \end{pmatrix} \end{array}$$

   i.e., the mutant types always have fitness 1 and the resident types fitness r, where $r \geq 0$[2]. Along with the evolutionary graph and the payoff matrix, we have two thresholds, namely, θ_R and θ_M, for the resident type and the mutant type, respectively. Intuitively, the thresholds represent a *density constraint*, and if an individual is surrounded by a lot of individuals of the same type, then its reproductive strength decreases. The density constraint is relevant in many applications of evolution (see books [2, page 470] [13, page 320], also see Remark 1). Let the selected vertex for reproduction be v. Let $\mathsf{Same}(v)$ denote the number of vertices in $E_I(v)$ that are of the same type as v. If v is a mutant type, and $\frac{\mathsf{Same}(v)}{|E_I(v)|} \leq \theta_M$ (resp., if v is a resident type, and $\frac{\mathsf{Same}(v)}{|E_I(v)|} \leq \theta_R$), then the individual gives birth to an individual of the same type. Note that the density constraint implies that if the constraint is violated, then the selected individual does not reproduce.

2. **The** I&R **and** IEQR **models.** One important special case is when the interaction and the replacement graphs coincide, i.e., $E_I = E_R$ [9, 11]. We refer to the general model as the I&R *model* (with possibly different interaction and replacement graphs) and the special case where the graphs coincide as the IEQR *model*.

3. **No resident reproduction.** Another special case is when the payoff matrix is the constant payoff matrix with $r = 0$. In this case, the resident types cannot reproduce. This represents the scenario that a mutant has an advantage over the residents such that if a mutant occupies a position, then the residents cannot win it back.

Remark 1 (Matrix encoding of density constraints in LBF.)**.** *For many of our lower bounds, we will use constant selection with density constraints, and we argue that the density constraints of our lower bounds, are special cases of the linear bounded fitness without any density constraints. In our results for lower bounds we consider two types of density constraints: (1) $\theta_M = \frac{1}{2} - \delta$, for $0 < \delta < 1/10$ (in Section 3 and Section 4), where there is no resident reproduction (hence θ_R is irrelevant); and and (2) $\theta_M = \theta_R = 0$ in Section 5. In all the lower bounds, the payoff matrix is constant. These two density constraints can be encoded as a payoff matrix (that is not constant) with linear bounded fitness function as follows:*

$$\begin{array}{cc} & \begin{array}{cc} R & M \end{array} \\ \begin{array}{c} R \\ M \end{array} & \begin{pmatrix} 0 & 0 \\ 1 & -1 \end{pmatrix} \end{array}; \qquad \begin{array}{cc} & \begin{array}{cc} R & M \end{array} \\ \begin{array}{c} R \\ M \end{array} & \begin{pmatrix} -N & 1 \\ 1 & -N \end{pmatrix} \end{array}.$$

The first payoff matrix encodes that a vertex that is a mutant can reproduce only if strictly less than half of the successors in E_I are mutants, and thus encode $\theta_M = \frac{1}{2} - \delta$, for $0 < \delta < 1/10$, in graphs where the outdegree is at most five. The second matrix (for a graph with N vertices) encodes that a vertex can reproduce only if all the successors in E_I are of the opposite type.

[2] In the literature, an alternative notion is to consider that the mutant have fitness r and the residents have fitness 1, we follow the notation that leads to uniform treatment

Organization. Our results are organized as follows.

1. In Section 3, Section 4, and Section 5, we consider the linear bounded fitness function. We will present upper bounds for the general case, and lower bounds for the special case of constant payoff matrix with density constraints (which is a restricted case as explained in Remark 1). In Section 3 we consider no resident reproduction, and present results for the qualitative case; and in Section 4 we again consider no resident reproduction and present results for the quantitative approximation. Finally, in Section 5 we present results for the qualitative and quantitative analysis in the general model with resident reproduction.

2. In Section 6 we present the results for the exponential fitness function.

	No Resident Reproduction		Resident Reproduction	
	IEQR model	I&R model	IEQR model	I&R model
Qual.	NP-c ((**LB**) Lem. 3)	NP-c ((**UB**) Lem. 2)	NP-h, PSPACE	PSPACE-c ((**LB**) Lem. 11, (**UB**) Lem. 9)
Appr.	#P-c ((**LB**) Thm. 8)	#P-c ((**UB**) Thm. 8)	#P-h, PSPACE	PSPACE-c ((**LB**) Lem. 11, (**UB**) Lem. 9)

Table 1: Complexity of evolution on graphs with linear bounded fitness. Qual is short-hand for qualitative and appr for approximation. Our main contributions of lower bounds (LB) and upper bounds (UB) are boldfaced. NP-c (resp., #P-c, PSPACE-c) means NP-complete (resp., #P-complete, PSPACE-complete). Similarly, NP-h (resp., #P-h) means NP-hard (resp., #P-hard).

3 Qualitative Analysis: No Resident Reproduction with LBF

In this section we establish two results for the no resident reproduction model with LBF: the qualitative analysis problem is (1) in NP for the general I&R model; and (2) is NP-hard in the special case of IEQR model, and even in a special case of LBF, where we have constant fitness with density constraints, (using density constraints mentioned in Remark 1).

3.1 Upper bound

The upper bound is relatively straightforward. We simply check if there exists an initial choice v_1 for the initial mutant and a sequence $(e_i)_{2 \leq i \leq n}$ of edges of length $n-1$ in the replacement graph for reproductions that ensures that all vertices are mutants. The initial vertex v_1 and the sequence of edges together define a unique sequence of vertices for reproduction; and at every stage we check that for the vertex chosen for reproduction can reproduce (i.e., has fitness strictly positive), and it is a mutant. We also need to check that in the end all vertices are mutants. The choice of the initial vertex and the sequence of reproductions then happen with positive probability and we are done. Observe that since there is no resident reproduction, if a vertex becomes a mutant, then it remains a mutant. Note that there always exists a sequence of length $n-1$, because if the fixation probability is positive, then we can WLOG assume (till all vertices are mutants) that in each step i there is a vertex v that is a mutant, with strictly positive fitness, and an edge (v, v') in the replacement graph such that v' is not a mutant (and becomes a mutant in step i), as otherwise nothing can change. This shows that if the answer to the qualitative decision question is yes in the no resident reproduction model, then there is a polynomial witness and polynomial-time verification procedure.

Lemma 2. *The qualitative decision question for no resident reproduction in the general* I&R *model with* LBF *is in NP.*

3.2 Lower bound

In this section we present an NP lower bound, and we will prove it for the IEQR model with no resident reproduction. We will also consider a special of LBF, which is constant fitness with density constraints. Moreover, since there is no

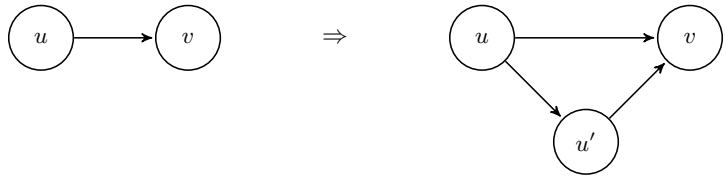

Figure 1: Illustration of a predecessor gadget (u, v).

resident reproduction, the threshold θ_R does not matter. We will present a reduction from the 3-SAT problem (which is NP-complete [3, 8, 5]) and use threshold θ_M as $\frac{1}{2} - \delta$, for any $0 < \delta \leq \frac{1}{10}$. However it would be easy to modify our construction for any threshold θ_M in $(0, 1)$. The "right" way to think of the threshold is that it is $\frac{1}{2}$ and that the density constraint uses a strict inequality. The upper bound is chosen because we will use vertices with degree five or less; recall Remark 1.

Notation. Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of n Boolean variables. Consider a 3-CNF formula $\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$, where each C_i is a *clause* of a list of (precisely) three *literals* (where a literal is a variable x or its negation $\overline{x}$, where $x \in X$). Each clause represents a disjunction of the literals that appear in it. An instance of the 3-SAT problem, given a 3-CNF formula φ, asks whether there exists a satisfying assignment. We will now construct an evolutionary graph $G(\varphi)$, given an instance of a 3-SAT problem, with (i) $E_I = E_R$, (ii) no resident reproduction, and (iii) threshold $\theta_M = \frac{1}{2} - \delta$, for $0 < \delta \leq \frac{1}{10}$ such that there is a satisfying assignment iff the answer to the qualitative decision problem is YES. We first present two gadget constructions that will be used in the reduction.

Predecessor gadget. We present a *predecessor* gadget for a vertex pair (u, v) such that v is the only successor of u. The gadget ensures the following property (namely, the *predecessor gadget* property): if all vertices become mutants, then the vertex u must have become a mutant before vertex v. The construction of the gadget is as follows: Add a new *dummy* vertex u'. Let the successors of u be v and u', and the successor of u' be only v. Then the only way for u' to become a mutant is if u is a mutant, since u is the only predecessor of u'. But u' can only become a mutant if u is a mutant and v is not (since otherwise the threshold condition with $\theta_M = \frac{1}{2} - \delta$ is not satisfied for u, for any $0 < \delta \leq \frac{1}{10}$). Hence, if all vertices become mutant, then u must become a mutant before v. There is an illustration of the predecessor gadget for (u, v) in Figure 1. We will denote by $\mathsf{PredEdges}(u, v, u')$ the set $\{(u, v), (u, u'), (u', v)\}$ of edges of the predecessor gadget.

(Extended) Binary tree gadget. Given a vertex rt, and a set L of vertices, we will denote by $\mathsf{BinTr}(\mathsf{rt}, L)$ a binary tree with rt as root and L as leaf vertices[3]. In a binary tree, every non-leaf vertex has out-degree 2. Note that the binary tree gadget adds additional vertices, and has $O(|L|)$ vertices. By an abuse of notation we will use $\mathsf{BinTr}(\mathsf{rt}, L)$ to denote both the set of vertices and the set of edges of the binary tree, and it would either be clear from the context or explicitly mentioned. Given a binary tree T and an *extension* vertex $z \notin T$, an *extended binary tree* (EBT) consists of T and an edge from every non-leaf vertex to z. Given a root vertex rt, a set L of leaf vertices, and an extension vertex z, we denote by $\mathsf{ExBinTr}(\mathsf{rt}, L, z)$ the edge set of the extended binary tree that extends the binary tree of rt and L. We will explicitly use the following property for an EBT (namely, *qualitative EBT (QEBT)* property):

- *(QEBT Property).* In an EBT, every non-leaf vertex has out-degree 3, and for density constraint with threshold $\frac{1}{2} - \delta$, for $0 < \delta \leq \frac{1}{10}$ (the construction works even if δ is up to $\frac{1}{6}$), if the root becomes a mutant and z is not a mutant, then the root can be responsible for making every vertex in the tree a mutant. However, note that if z is a mutant, then any vertex in the tree with out-degree 3 cannot make both its children in the underlying tree mutants due to the density constraint.

There is an illustration of a binary tree $\mathsf{BinTr}(x, \{v_1, v_2, v_3\})$ and the corresponding EBT $\mathsf{ExBinTr}(x, \{v_1, v_2, v_3\}, z)$ in Figure 2.

The evolutionary graph $G(\varphi)$. We now present the evolutionary graph $G(\varphi)$, see Figure 3 for an illustration, where we first describe the vertex set and then the edges. Recall that n is the number of variables and m the number of clauses of the 3-SAT instance ϕ.

[3]For a fixed L and rt there exists many possible binary trees $\mathsf{BinTr}(\mathsf{rt}, L)$, however every one of them will work for our purpose

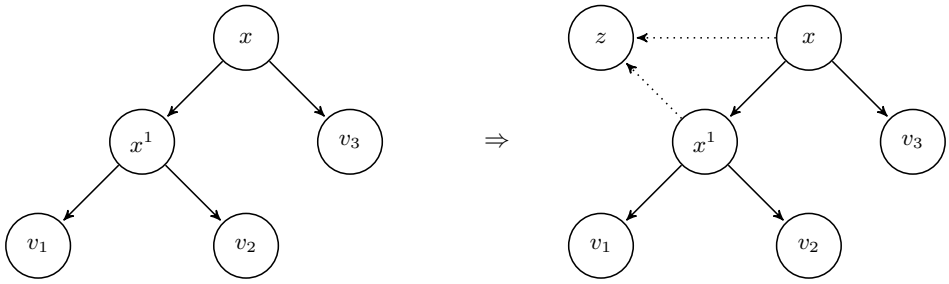

Figure 2: A binary tree $\mathsf{BinTr}(x, \{v_1, v_2, v_3\})$ and the corresponding EBT $\mathsf{ExBinTr}(x, \{v_1, v_2, v_3\}, z)$, where we extend with the vertex z. The edges to z are dotted to make the similarities easier to see.

The vertex set. The set V of vertices is as follows (intuitive descriptions follow):

$$\begin{array}{llll} \{v_\top, z_\perp, y_\perp, z'_\perp\} & \cup \ \{c_1, c_2, \ldots, c_m\} & \cup \ \{c_i^1, c_i^2, c_i^3 \mid 1 \le i \le m\} & \cup \ \{\widehat{c}_i^1, \widehat{c}_i^2, \widehat{c}_i^3 \mid 1 \le i \le m\} \\ & \cup \ \{x_i, x_i^t, x_i^f \mid x_i \in X\} & \cup \ \{v_0, v'_0\} & \\ & \cup \ \{u_i^t, u_i^f \mid 1 \le i \le n\} & \cup \ \bigcup_{1 \le i \le n} (\mathsf{BinTr}(x_i^t, L_i^t) \cup \mathsf{BinTr}(x_i^f, L_i^f)) & \end{array}$$

The vertex $v_\top$ will be the start vertex; and the vertices $z_\perp$, $y_\perp$, and $z'_\perp$ are end vertices (that will form a predecessor gadget for $(z_\perp, y_\perp)$ with dummy vertex $z'_\perp$). We have a vertex c_i for each clause C_i (named the clause vertices); and one for each literal c_i^1, c_i^2, and c_i^3 in the clause (named the clause-literal vertices). Similarly, we have a vertex x_i for each variable in X (named the variable vertices), and vertices x_i^t and x_i^f (named the variable-value vertices) to represent the truth values to be assigned to x_i. Corresponding to x_i^t and x_i^f we also have vertices u_i^t and u_i^f (named the duplicate vertices). The vertex v_0 forms a predecessor gadget (using the dummy vertex v'_0) to u_1^t. Let $L_i^t = \{\widehat{c}_k^j \mid 1 \le k \le m,\ 1 \le j \le 3,\ c_k^j = x_i\}$ denote a copy of the clause-literal vertices that correspond to x_i and $L_i^f = \{\widehat{c}_k^j \mid 1 \le k \le m,\ 1 \le j \le 3,\ c_k^j = \overline{x}_i\}$ denote a copy of the clause-literal vertices that correspond to negation of x_i. The set $\mathsf{BinTr}(x_i^t, L_i^t)$ (resp., $\mathsf{BinTr}(x_i^f, L_i^f)$) represents the vertices of a binary tree with the root vertex x_i^t (resp., x_i^f) and leaf vertices L_i^t (resp., L_i^f).

The edge set. We now describe the edge set:

- There is an edge from the initial vertex $v_\top$ to the first clause vertex c_1; and we have two predecessor gadgets; (i) $(z_\perp, y_\perp)$ with dummy vertex $z'_\perp$; and (ii) (v_0, u_1^t) with dummy vertex v'_0.
- For each clause vertex c_i, there are five edges, three to clause-literal vertices c_i^j (for $j = 1, 2, 3$) of the clause, one to the next clause vertex (for c_m this next vertex is x_1), and to the vertex u_1^t.
- For each variable vertex x_i, there are three edges: to x_i^t and x_i^f, and to the next variable vertex x_{i+1} (for x_n the next vertex is v_0).
- Each duplicate vertex u_i^t has three edges: to u_i^f, to x_i^t, and to $y_\perp$. Similarly, each vertex u_i^f has three edges: to u_{i+1}^t (u_n^f has edge to $z_\perp$ instead), to x_i^f, and to $y_\perp$.
- Finally, we have the EBT with x_i^α (for $\alpha \in \{t, f\}$) as root, L_i^α as leaf vertices and $y_\perp$ as the extension vertex. For each vertex in L_i^α, for $\alpha \in \{t, f\}$, we add edges to the corresponding clause-literal vertex and to u_1^t. This ensures that every internal vertex of the binary tree has degree three, and leaf vertices have degree two.

The formal description is as follows:

$$\{(v_\top, c_1)\} \ \cup \ \mathsf{PredEdges}(z_\bot, y_\bot, z'_\bot) \ \cup \ \mathsf{PredEdges}(v_0, u_1^t, v'_0)$$

$$\{(c_i, c_i^j) \mid 1 \le i \le m, 1 \le j \le 3\} \ \cup \ \{(c_i, u_1^t) \mid 1 \le i \le m\} \ \cup \ \{(c_i, c_{i+1}) \mid 1 \le i \le m-1\} \cup \{(c_m, x_1)\} \ \cup$$

$$\{(x_i, x_i^t), (x_i, x_i^f) \mid 1 \le i \le n\} \ \cup \ \{(x_i, x_{i+1}) \mid 1 \le i \le n-1\} \cup \{(x_n, v_0)\} \ \cup$$

$$\{(u_i^t, u_i^f) \mid 1 \le i \le n\} \ \cup \ \{(u_i^f, u_{i+1}^t) \mid 1 \le i \le n-1\} \ \cup \ \{(u_n^f, z_\bot)\} \ \cup \ \{(u_i^\alpha, x_i^\alpha), (u_i^\alpha, y_\bot) \mid 1 \le i \le n, \alpha \in \{t, f\}\} \cup$$

$$(\textstyle\bigcup_{1 \le i \le n}(\mathsf{ExBinTr}(x_i^t, L_i^t, y_\bot) \cup \mathsf{ExBinTr}(x_i^f, L_i^f, y_\bot)) \ \cup \ \{(\widetilde{c}_k^j, c_k^j), (\widetilde{c}_k^j, u_1^t) \mid \widetilde{c}_k^j \in L_i^\alpha, 1 \le i \le n, \alpha \in \{t, f\}\}$$

Example. We will now give an example of the graph $G(\varphi)$ for $\varphi = (\bar{x} \vee y \vee x) \wedge (z \vee x \vee \bar{x})$. See Figure 3. The edges to u_1^t are dashed and the edge from u_i^α for all $1 \le i \le 3$ and $\alpha \in \{t, f\}$ are dotted, for readability. Also, the vertex u_1^t is included twice to make it clearer that it is in a predecessor gadget.

Basic facts. We first mention some basic facts about the evolutionary graph obtained.

1. First, observe that the predecessor gadget property implies that for fixation the vertex v_0 must become a mutant before vertex u_1^t; and vertex $z_\bot$ before vertex $y_\bot$.

2. Second, for a vertex with degree ℓ, it can reproduce a mutant as long as at most $\ell \cdot (\frac{1}{2} - \delta)$ successors are mutants. In particular, for vertices with five (resp., three) successors, like the clause (resp., variable) vertices, it can reproduce a mutant until at most three (resp., two) successors are mutants, because of the bounds on θ_M. If a vertex has out-degree two (or one), then it can reproduce a mutant until at most one successor is a mutant, because of the bounds on θ_M. The conditions follow from the density constraint with threshold $\frac{1}{2} - \delta$.

Two phases for fixation. For mutants to attain fixation (i.e., all vertices become mutants), certain conditions must be fulfilled. The first basic fact above implies that for the evolutionary process to attain fixation, it must make vertex x_n a mutant (then vertex v_0 a mutant) before vertex u_1^t. We thus split the process of fixation in two phases: in the first phase u_1^t is not a mutant, and in the second phase u_1^t will be a mutant. We further split the first phase into two sub-phases, the first sub-phase is related to clause vertices becoming mutants, and the next sub-phase is related to the variable vertices becoming mutants. The description of the phases for fixation are as follows:

1. *(Phase 1:Part A).* The mutant must be initialized at the start vertex $v_\top$ (since $v_\top$ has no predecessor). After $v_\top$, the clause vertex c_1 becomes a mutant. Since at most half (three) successors can become mutant from c_1 (recall that c_1 has five successors), and one of them must be c_2 (as the only incoming edge for c_2 is from c_1), it follows that c_2 and at most two clause-literal vertices for clause C_1 becomes mutant from c_1. This process is then repeated for all the clause vertices c_i till x_1 becomes a mutant.

2. *(Phase 1:Part B).* Each of the vertices x_i has three successors, and hence can make two of them mutants. One of them must be x_{i+1} (as x_{i+1} has only x_i as the predecessor), and the other one is at most one of x_i^t or x_i^f. This continues till we reach v_0. Note that once x_i^t becomes a mutant, then the entire EBT under x_i^t, including the corresponding clause-literal vertices, but not $y_\bot$ and u_1^t, can become mutants, as long as $y_\bot$ and u_1^t are not mutants. The reasoning is as follows: the leaf vertices has two out-going edges, and since u_1^t is not a mutant, it can reproduce a mutant to the corresponding clause-literal vertices, and the rest follows from the QEBT property. The phase 1 ends with the predecessor gadget of (v_0, u_1^t) becoming mutants. Note that this phase corresponds to a partial assignment of truth values to the variables as follows: for a variable x_i, if x_i^t was chosen (made mutant), it corresponds to assigning true to x_i; if x_i^f was chosen, it corresponds to assigning false to x_i; otherwise, if neither was chosen, then it corresponds to no assignment to x_i (if fixation is reached without having made an assignment to some set U of variables, then any possible assignment of values to the variables of U will make the partial assignment a satisfying assignment).

3. *(Phase 2).* This phase starts after u_1^t is a mutant. We establish a key property of this phase that will be used in the proof. Consider the EBT under some variable-value vertex. Each leaf vertex of the tree has out-degree two:

one of the successors is u_1^t and the other is a clause-literal vertex. It follows that once u_1^t has become a mutant, then the leaf vertices cannot reproduce any more. Thus the key property of Phase 2 is as follows: leaf vertices of EBTs cannot reproduce mutants to clause-literal vertices after Phase 2 starts.

The graph $G(\varphi)$ has positive fixation probability iff φ is satisfiable. We present two directions of the proof.

1. *Satisfiablity implies positive fixation.* Consider a satisfying assignment to φ, and intuitively the assignment chooses at least one literal in each clause. The sequence of mutants reproduced in the two phases for fixation is as follows:

   - (Phase 1). The sequence in Phase 1 is the following: (1) initial vertex $v_\top$ becomes a mutant which then reproduces a mutant to c_1; (2) in vertex c_i, it reproduces upto three mutants, one to c_{i+1} (to x_1 for $i = m$) and upto two mutants for vertices c_i^j of the clauses which are not chosen by the satisfying assignment (this corresponds to Phase 1:Part A); (3) for a vertex x_i it reproduces two mutants, one to x_{i+1} (to v_0 for $i = n$), and the other to x_i^t (resp., x_i^f) if the assignment chooses x_i to be true (resp., false); and moreover, the entire EBT under x_i^t (resp., x_i^f) including the clause-literal vertices become mutants (other than u_t^1 and $y_\perp$); and (4) then v_0' becomes a mutant and then u_1^t becomes a mutant from v_0, and proceed to Phase 2.
   - (Phase 2). The sequence in Phase 2 is the following: (1) In every vertex u_i^α (for $\alpha = t$ or f) it makes x_i^α mutant (if it is not already a mutant) and then it makes the next vertex in line a mutant (if $i = n$ and $\alpha = f$, then the next vertex is $z_\perp$, otherwise, the next vertex is u_i^f if $\alpha = t$ and u_{i+1}^t if $\alpha = f$); moreover, once x_i^α becomes a mutant, so does the entire binary tree (other than $y_\perp$) under it (but not the clause-literal vertices since u_1^t is a mutant); and (2) finally the $(z_\perp, y_\perp)$ predecessor gadget becomes mutants.

   The claim follows.

2. *No satisfying assignment implies no fixation.* Note that for fixation we need the two phases. In every clause c_i at least one of the clause-literal vertices c_i^j was not made a mutant by c_i in Phase 1:Part A (or even after that). This implies that if Phase 2 has started and not all clause-literal vertices c_i^j of a clause c_i have become mutants, then at least one of these vertices cannot become a mutant, by the key property of Phase 2. For each (partial) assignment that is not satisfying, there exists at least one clause, in which no literals are chosen. Recall that the reproduction of mutants in Phase 1:Part B gives a partial assignment of truth values to variables. Hence, in the process of reproducing mutants in Phase 1:Part B, there must remain a clause where at most two clause-literal vertices are mutants. Therefore it implies that if there is no satisfying assignment, then fixation is not possible.

We obtain the following result. Lemma 2 and Lemma 3 give Theorem 4.

Lemma 3. *The qualitative decision question for no resident reproduction in the* IEQR *model with* LBF *is NP-hard.*

Theorem 4. *The qualitative decision question for no resident reproduction in both the general* I&R *model and the* IEQR *model with* LBF *is NP-complete.*

4 Quantitative Approximation: No Resident Reproduction with LBF

In this section we show that in the no resident reproduction model with LBF the following assertions hold: (i) the precise fixation probability can be computed in #P (for the general I&R model); and (ii) for $\epsilon > 0$, the problem of approximating the fixation probability within an additive error of ϵ is #P-hard (even in the IEQR model). Again in our lower bound we will consider a special case of LBF where we have constant fitness with density constraint.

4.1 Upper bound

Consider a sequence of reproductions $((v_i, u_i))_i$ that leads to fixation, where v_1 is the first vertex to be a mutant. In other words, for a given i, the i-th reproduction consists of v_i reproducing a mutant to u_i. The sequence $((v_i, u_i))_i$

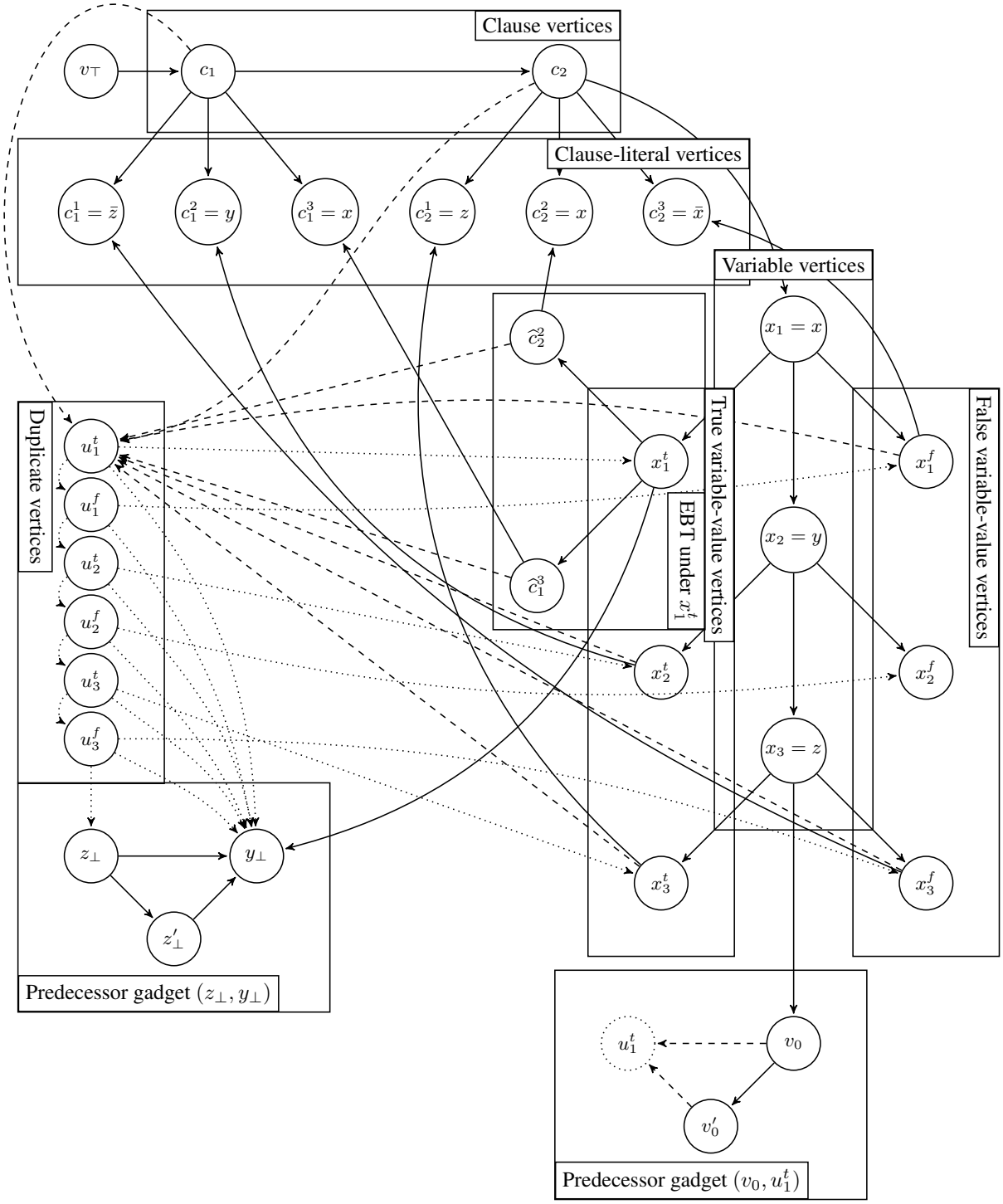

Figure 3: The graph $G(\varphi)$ for $\varphi = (\bar{z} \vee y \vee x) \wedge (z \vee x \vee \bar{x})$. Edges to u_1^t are dashed and edges from u_i^α are dotted for readability. The vertex u_1^t is included twice to make it clearer that it is in a predecessor gadget. The notation $c_1^2 = y$ indicates that the second variable of the first clause is variable y. The notation $x_1 = x$ indicates that the first variable is variable x.

defines the sequence of probabilities $(p_i)_i$ such that p_i is the probability for vertex v_i to reproduce to u_i in the graph where $\{v_1, u_1, \ldots, u_{i-1}\}$ is the set of mutants. Let d be the least common multiplier of the denominators of (i) $\frac{1}{N}$, where N is the number of vertices (this represents the probability that a given vertex is the start vertex), and (ii) p_i for each i in each sequence $(p_i)_i$ defined from a sequence of reproductions leading to fixation.

We will next argue that d is at most exponential. Consider some payoff matrix

$$\begin{array}{cc} & \begin{array}{cc} R & M \end{array} \\ \begin{array}{c} R \\ M \end{array} & \begin{pmatrix} 0 & 0 \\ a & b \end{pmatrix} \end{array} \tag{1}$$

where a and b are integers given in binary. The denominator for selecting a certain mutant to reproduce is the sum of the fitness of the mutants that can reproduce (*SFMR*[4]). The SFMR is a number on the form

$$a \cdot i + b \cdot j$$

where i (resp., j) is the number of edges $(u, v) \in E_I$, where u is a mutant that can reproduce and v is a resident (resp., mutant). Note that the description of i and j shows that they are non-negative numbers such that $i + j \leq N^2$. Hence, there are at most $k \leq N^4$ different SFMRs. A mutant can only reproduce if it has positive fitness and thus each SFMR is positive. Also, each SFMR have value at most $S \leq |a| \cdot N^2 + |b| \cdot N^2$. Therefore, $d \leq N \cdot S^k$, which is at most exponential (the N factor is to select the start mutant).

For a fixed sequence of reproductions $((v_i, u_i))_i$, let $p = \frac{\prod_i p_i}{N}$. Observe that p is the probability that the first mutant is v, and the reproductions happen exactly according to $((v_i, u_i))_i$. Also, note that $\frac{p}{d^n}$ is an integer.

We consider the following *probability counting problem*: Pick a sequence of reproductions $((v_i, u_i))_i$ leading to fixation, and an integer c between 1 and $\frac{p}{d^n}$. This is a candidate solution, and it is easy to check in polynomial time if it is indeed one. Thus counting the number of solutions is in #P. Observe that a fixed vertex v and sequence of reproductions $((v_i, u_i))_i$ is used in exactly $\frac{p}{d^n}$ solutions. Hence, by multiplying the number of solutions of the probability counting problem with d^n we get the probability that the original system fixate for the mutants. We get the following result:

Lemma 5. *The fixation probability computation for no resident reproduction in both the general* I&R *model and the* IEQR *model with* LBF *is in* #P.

Remark 6. *Note that the quantitative approximation problem is not defined as a decision problem. For* #P *upper bound above, for the approximate (as well as exact) fixation probability we mean that given the number of solutions to an* #P *problem we can compute in polynomial time the exact fixation probability.*

4.2 Lower bound

The remainder of this section will be on our lower bound, where we reduce the problem of counting perfect matchings in bipartite graphs to approximating the probability that mutants fixates. Like in the previous section, we will consider constant fitness with density constraints, and the threshold θ_M will be $\frac{1}{2} - \delta$, for any $0 < \delta \leq \frac{1}{10}$ (because the degree is again bounded by 5). Moreover our lower bound will be for the IEQR model.

Perfect matching in bipartite graphs. We present a reduction from the computation of the number of perfect matchings in a bipartite graph $G = (V, E)$. In a bipartite graph G, the vertex set V is partitioned into vertices V_ℓ (left vertices) and V_r (right vertices) and all edges go from a vertex in V_ℓ to a vertex in V_r (i.e., $E \subseteq V_\ell \times V_r$). We also have $|V_\ell| = |V_r| = n$. A *perfect matching* PM is a set $\{e_1, e_2, \ldots, e_n\}$ of n edges from E such that for every vertex $v_\ell \in V_\ell$ (resp., $v_r \in V_r$) there exists an edge $e_\ell = (v_\ell, v'_r)$ (resp., $e_r = (v'_\ell, v_r)$) in PM. Given a bipartite graph, the problem of computing the number of distinct perfect matchings was shown by Valiant [16] to be #P-complete.

Uniform degree property. First, we will show that we only need to consider bipartite graphs for which there exists an integer k such that all vertices in V_ℓ have either degree 2^k or 1. We refer to the property as the *uniform degree* property.

[4]S stands sum, F for fitness, M for mutant, R for reproduce

Reduction to uniform-degree graphs. We present a reduction from counting the number of perfect matchings in a general bipartite graph $G = (V, E)$ (with $|V_\ell| = |V_r| = n$) to counting the number of perfect matchings in a bipartite graph $G' = (V', E')$ with at most $6n$ vertices and which has the uniform degree property. Let $k = \lceil \log d_{\max} \rceil$, where $d_{\max}$ is the maximum degree of any vertex in G. The graph G' will have precisely as many perfect matchings as G. Observe that $2^k < 2n$. We construct G' by adding 2^k new pairs of vertices, one on each side, and for each new pair (v, v'), we add an edge from v to v'. Then, for vertex $v \in V_\ell$, we add edges from v to some newly added vertex in V'_r until v has degree 2^k. It is clear that any perfect matching in G corresponds to a perfect matching in G' using the same edges, and the edges between newly added pairs. Conversely, we also see that in each perfect matching in G', for each newly added pair (v, v'), the matching must use the edge between v and v', since the vertex in $(V'_\ell \setminus V_\ell)$ has degree 1. Thus every perfect matching in G' corresponds to one in G.

Perfect binary trees. We will consider perfect binary trees as gadgets.

- A *perfect binary tree* (PBT) is a balanced binary tree (every internal vertex has exactly two children) with all leaves at the same level (i.e. with 2^k leaf vertices, for some non-negative integer k). For a PBT we will use the following property, which we refer to as the *probabilistic PBT (PPBT)* property: if the root becomes a mutant, then eventually all vertices in a path from the root to some leaf will become mutants, where such a path is chosen uniformly at random. Since every non-leaf vertex has out-degree two, due to the density constraint, each internal vertex can make one of its children (chosen uniformly) a mutant and hence the PPBT property follows.

The graph Red(G)**.** Given a bipartite graph G with the uniform degree property, let the vertex sets be V_ℓ and V_r, respectively. Let $N(v) = \{u \mid (v, u) \in E\}$ denote the successors of a vertex $v \in V_\ell$. Let $V_\ell^k = \{v \in V_\ell \mid |N(v)| = 2^k\}$ be the set of vertices with degree 2^k; and $V_\ell^1 = V_\ell \setminus V_\ell^k$ be the set of vertices in V_ℓ with degree 1. Our reduction, denoted Red(G), will construct an evolutionary graph (with $E_I = E_R$ and hence we only specify one set of edges), which consists of three parts: part 1 sub-graph, then edges related to V_r, and a copy of part 1 with some additional edges. We first describe the part 1 sub-graph and then its copy.

- *(Part 1).* We have a start vertex v_s, a final vertex $y_\perp$, and we create an EBT B_s as follows: $\mathsf{ExBinTr}(v_s, V_\ell, y_\perp)$, i.e., the start vertex is the root, V_ℓ is the set of leaf vertices, and $y_\perp$ is the extension vertex. For every vertex $v \in V_\ell^k$, let $N(v) = \{u^1, u^2, \ldots, u^j\}$, and we consider the set $L_v^k = \{u_v^1, u_v^2, \ldots, u_v^j\}$ of $j = 2^k$ vertices and construct a PBT $P_v = \mathsf{BinTr}(v, L_v^k)$. Note that B_s is an EBT, but the underlying binary tree is not necessarily perfect.

- *(Edges related to V_r).* From every vertex $v \in V_\ell^k$, and every u_v^i in L_v^k, we add two edges: one to $u^i \in N(v)$ and one to $y_\perp$. From every vertex $v \in V_\ell^1$ (with degree 1), we add two edges: to the unique $u \in N(v)$ and to $y_\perp$. Every vertex in V_r has an edge to $y_\perp$.

- *(Copy 1 of Part 1 with additional edges).* First, we create a copy of the part of the graph described in part 1, along with one additional vertex $z_\perp$. For every vertex v of part 1, let the corresponding vertex in the copy be called $\overline{v}$, and the copy of the extension vertex is $\overline{y}_\perp$. We describe the difference in the copy as compared to the graph of part 1: (i) first there is an edge from $y_\perp$ to the copy $\overline{v}_s$ of the start vertex; (ii) for every vertex $\overline{z}$ which is a copy of a non-leaf vertex z in P_v, for some $v \in V_\ell^k$, (i.e., $z \notin L_v^k$), there are three additional edges from $\overline{z}$: (a) to z (i.e., from the copy to the original vertex), (b) to $\overline{y}_\perp$, and (c) to $z_\perp$; and (iii) for every vertex $\overline{z}$ which is a copy of a leaf vertex z in P_v, for some $v \in V_\ell^k$, (i.e., $z \in L_v^k$), there is only one edge which goes to z (i.e., there is no edge to V_r or $y_\perp$, but an edge from the copy to the original vertex). Hence in the copy of P_v, for any v, internal vertices have degree five, and leaf vertices have degree 1.

- Finally, we have the following edges: $\{(y_\perp, \overline{y}_\perp), (y_\perp, z_\perp), (\overline{y}_\perp, z_\perp)\}$.

We denote by $\widehat{n}$ the number of vertices in Red(G), and note that $\widehat{n} = O(m)$, where m is the number of edges in G.

Example. We consider the graph G with six vertices, where $V_\ell = \{v_1, v_2, v_3\}$ and $V_r = \{v_4, v_5, v_6\}$, such that v_1 and v_2 each have edges to v_4 and v_5 and v_3 has an edge to v_6. See Figure 4 for an illustration. Observe that G satisfies the uniform degree property. In Figure 5 we have part 1 of the graph Red(G) along with V_r. In Figure 6 we have the remainder of Red(G). Consider some fixed perfect matching PM in G, i.e. $v_1 \to v_4$ and $v_2 \to v_5$ and $v_3 \to v_6$. The

graph $\text{Red}(G)^{\mathsf{PM}}$ is then the same graph as in Figure 5 and Figure 6, except that in Figure 5 it does not contain the edges from v_2^1 or v_1^2.

The process of fixation in $\text{Red}(G)$**.** The process of fixation in $\text{Red}(G)$ can be decomposed in two phases. The first phase (Phase 1) is over when $y_\perp$ becomes a mutant; and the second phase (Phase 2) is over with the fixation. A key property of Phase 2 is as follows: vertices in V_r cannot become a mutant after $y_\perp$ has become a mutant: This is because for each vertex u in V_r, every predecessor v of u has exactly two successors, and one them is $y_\perp$ (and hence the density constraint with threshold $\frac{1}{2} - \delta$ ensures that if $y_\perp$ is a mutant, then vertices in V_r cannot become mutants after that).

- *Phase 1.* In Phase 1, the vertex v_s must be the first vertex to become a mutant (since it has no predecessor). After v_s, all vertices in B_s turn into mutants (by the QEBT property). Once a vertex $v \in V_\ell^k$ becomes a mutant, then a path in the PBT P_v under v is chosen uniformly at random to become mutants (by the PPBT property), and then the leaf of the path can make the corresponding vertex in V_r a mutant. Once a vertex v in V_ℓ^1 with degree 1 becomes a mutant, then it can reproduce a mutant to the unique neighbor in V_r. In the end, some vertex in V_r reproduces a mutant to $y_\perp$ and Phase 1 ends.

- In Phase 2, first the copy $\overline{v}_s$ becomes a mutant from $y_\perp$. After $\overline{v}_s$, all vertices which are copy of vertices in B_s become mutants (again by the QEBT property). Once copies of vertices in V_ℓ^k become mutant, then the tree underneath them in the copy become mutants. Consider a vertex $\overline{u}$ which is a copy of a vertex $u \in P_v$, for some $v \in V_\ell^k$, and there are two cases: (i) if u is a non-leaf vertex, then $\overline{u}$ has degree five, and can reproduce mutants until the two children in the tree and the original vertex u are mutants (note if $\overline{y}_\perp$ or $\overline{z}_\perp$ is a mutant, then both the children and the original copy cannot all become mutants due to the density constraint); (ii) if u is a leaf-vertex, then $\overline{u}$ has degree one, and can reproduce mutant for u. Finally, $y_\perp$ makes $\overline{y}_\perp$ a mutant, which then makes $z_\perp$ a mutant.

Fixation and a perfect matching. Observe that fixation implies that all vertices in V_r have become mutant, and no vertex in V_r can become a mutant in the second phase. Each vertex in V_ℓ is responsible for making at most one neighbor in V_r a mutant (for vertices with degree 1 it is the unique successor in V_r, and for vertices with degree 2^k, it corresponds to the leaf of the path in the perfect binary tree chosen uniformly at random by the PPBT property). This defines a perfect matching. Conversely, given a perfect matching, Phase 1 and Phase 2 of fixation can be described using the pairs of the matching (to chose paths uniformly at random in the perfect binary trees). Thus given fixation, it defines a perfect matching, and we say that fixation has *used* the perfect matching.

Exact fixation probability. Consider some perfect matching PM. Observe that if there are $s > 0$ perfect matchings, then the exact fixation probability is $s \cdot x_{\mathsf{PM}}$, where x_{PM} is the probability that we have fixation and used PM. This is because each perfect matching has the same probability to be the chosen matching in Phase 1 by the PPBT property. In Phase 2, any vertex v which is either a vertex in V_ℓ^1 or a leaf in P_v, for $v \in V_\ell^k$, cannot reproduce by the key property of Phase 2 (and thus can be viewed as having no out-going edges). Thus in Phase 2, by symmetry, the probability x_{PM} of fixation for a perfect matching PM is independent of PM.

Bounds on x **and** s**.** We show that the probability x for fixation of a fixed matching is at least $\eta = \widehat{n}^{-2\widehat{n}}$, where $\widehat{n}$ is the number of vertices in $\text{Red}(G)$. Each possible way that all vertices can become mutants happens with probability at least $\widehat{n}^{-2\widehat{n}}$, because there are at most $\widehat{n}$ reproductions (effective reproductions which produce a new mutant) and each specific reproduction chooses two vertices v and v' at random from some set of vertices and thus, a specific choice happens with probability at least $\widehat{n}^{-2}$. Thus the lower bound η on x follows. Finally, observe that the number s of perfect matchings can be at most $n!$ (i.e., upper bound on s is $n!$).

The graph $\text{Red}(G)^{\mathsf{PM}}$**.** Given a perfect matching PM, we can find x as the fixation probability for the graph $\text{Red}(G)^{\mathsf{PM}}$, which is similar to $\text{Red}(G)$, except that each leaf vertex u_v^i in P_v, for $v \in V_\ell^k$, if (v, u^i) is not in the matching, then we remove all out-edges from u_v^i, and otherwise u_v^i has the same edges as in $\text{Red}(G)$. It is clear that the fixation probability in $\text{Red}(G)^{\mathsf{PM}}$ is x.

Approximating the fixation probability is $\#\mathsf{P}$**-hard.** Our reduction is as follows: Given a graph G with the uniform degree property, we want to find the number of perfect matchings s in it. First, (i) we find an arbitrary perfect matching PM in polynomial time using the algorithm of [6] (if there exists none, we are done); (ii) construct $\text{Red}(G)$

and $\mathrm{Red}(G)^{\mathsf{PM}}$ in polynomial time; and (iii) compute the approximation y' of the fixation probability y^* in $\mathrm{Red}(G)$ for $\epsilon = \frac{\eta}{16}$, and the approximation x' of the fixation probability x in $\mathrm{Red}(G)^{\mathsf{PM}}$ for $\epsilon_{\mathsf{PM}} = \frac{\eta}{n! \cdot 16} = \frac{\epsilon}{n!}$. We now show how to obtain s from y' and x'. We have that y' is such that

$$y' \leq x \cdot s + \epsilon \leq (x' + \epsilon_{\mathsf{PM}}) \cdot s + \epsilon = x' \cdot s + \frac{\eta}{n! \cdot 16} \cdot s + \frac{\eta}{16} \leq x' \cdot s + \frac{\eta}{8} \ ,$$

and similarly $y' \geq x' \cdot s - \frac{\eta}{8}$. This shows that

$$s - \frac{\eta}{8x'} \leq \frac{y'}{x'} \leq s + \frac{\eta}{8x'}.$$

Since we also have $x' \geq x - \epsilon = \eta - \frac{\eta}{n! \cdot 16} \geq \frac{15 \cdot \eta}{16}$ we see that $\frac{\eta}{8x'} < 1/3$ and thus s is the integer closest to $\frac{y'}{x'}$.

Lemma 7. *The quantitative approximation problem for* $0 < \epsilon < 1$*, with* ϵ *given in binary, for no resident reproduction in both the general* I&R *model and the* IEQR *model with* LBF *is* #P*-hard.*

Lemma 5 and Lemma 7 gives the following result (also recall Remark 6).

Theorem 8. *The quantitative approximation problem, where the approximation number* $0 < \epsilon < 1$ *is given in binary, for no resident reproduction in both the general* I&R *model and the* IEQR *model with* LBF *is* #P*-complete (and even the exact fixation probability can be computed in* #P*).*

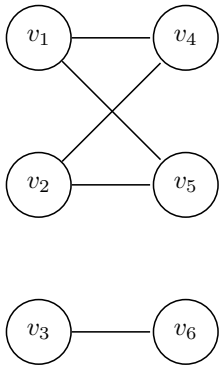

Figure 4: The graph G.

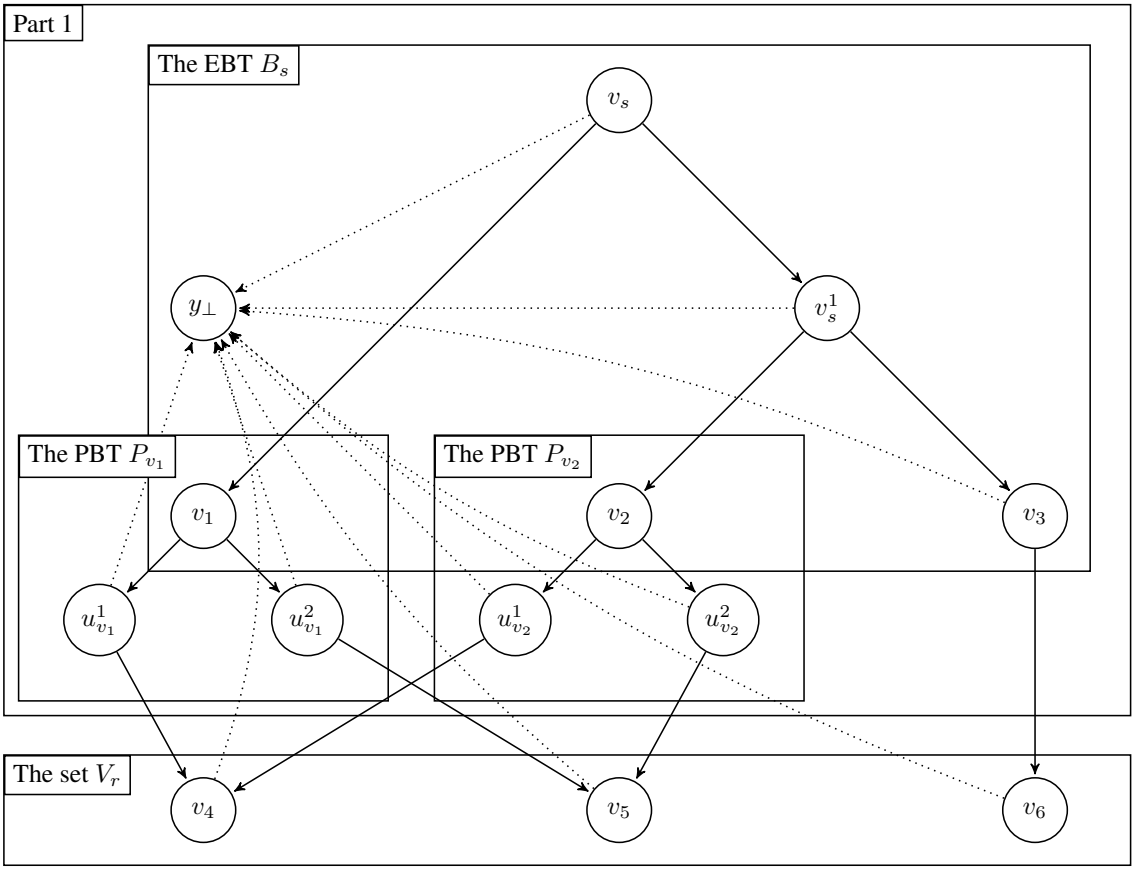

Figure 5: Part 1 and the edges related to V_r of the graph $\mathrm{Red}(G)$. The edges to $y_\perp$ are dotted for readability.

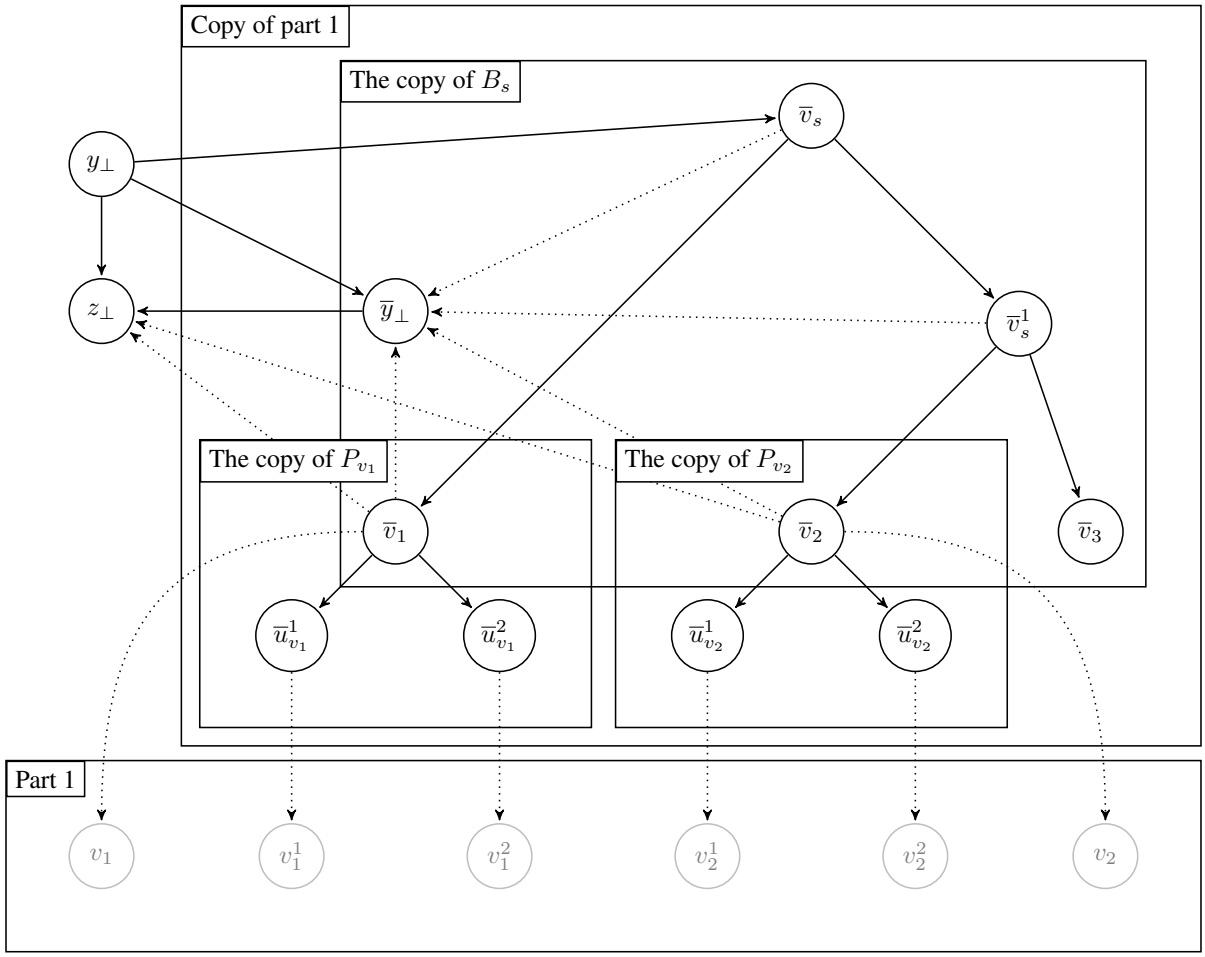

Figure 6: The copy of part 1 of the graph Red(G). Most edges to $\overline{y}_\perp$ and to $z_\perp$ are dotted for readability.

5 Qualitative Analysis and Quantitative Approximation: I&R Model with Resident Reproduction and LBF

In this section we will establish the polynomial space upper bound and lower bound in the I&R model with resident reproduction, when the fitness function is LBF.

5.1 Upper bound

Our algorithms is based on an exponential Markov chain construction. We first describe what is a Markov chain and Markov chains associated with an evolutionary graph.

Markov chain. A Markov chain $M = (S, \Delta)$ consists of a finite set S of states, and a probabilistic transition function Δ that assigns transition probabilities $\Delta(s, s')$ for all $s, s' \in S$, i.e., $0 \leq \Delta(s, s') \leq 1$ for all $s, s' \in S$ and for all $s \in S$ we have $\sum_{s' \in S} \Delta(s, s') = 1$. Given a Markov chain, its graph (S, E) consists of the set S as the set of vertices, and $E = \{(s, s') \mid \Delta(s, s') > 0\}$ positive transition probabilities as the set of edges.

Exponential Markov chain. Given an evolutionary graph $G = (V, E_I, E_R)$, with a payoff matrix, and the LBF function, an exponential Markov chain $M_E = (S, \Delta)$ is constructed as follows: (1) S consists of subsets of V which denotes the set of vertices of V which are currently mutants; (2) for $s \in S$ and $s' \in S$ there is positive transition probability if the cardinality of s and s' differ by 1 and the transition probability $\Delta(s, s')$ is computed depending on the payoff matrix, E_I, and E_R. Observe that for the Markov chain M_E, the transition probabilities of a state in the Markov chain can be constructed in polynomial space, and hence the Markov chain can be constructed in polynomial (working) space.

Qualitative analysis and approximation of Markov chains. We sketch the arguments for the upper bounds.

- The qualitative analysis is achieved by simply checking if in the graph of the Markov chain the state $s_f = V$ is reachable from some state $s = \{v\}$ for $v \in V$. Since the reachability problem can be solved in non-deterministic logspace [14], applying such a reachability algorithm on the Markov chain M_E (constructed on the fly) we get a non-deterministic polynomial space algorithm. Since by Savitch's theorem [15] non-deterministic polynomial space is equivalent to deterministic polynomial space, it follows that the qualitative question is in PSPACE.

- For the approximation problem we simulate the Markov chain as follows. We start at an initial state uniformly at random among those where there is exactly one mutant. Consider a *trial run* of the Markov chain as follows. Given the current state, we first check if (i) the current state is V; else we check (ii) if there is a path from the current state to $s_f = V$. If (i) is true we have a success; and if (ii) is false we have a failure. If we neither succeed or fail, we use the transition probability of the Markov chain to obtain the next state till we succeed or fail. Note that each trial run succeeds or fails eventually with probability 1. We can view the outcome of each trial run as the outcome of a Bernoulli distributed random variable with success probability equal to the fixation probability. Hence repeating the trial runs independently an exponential number of times, we can approximate the fixation probability using Chernoff bounds, within any given $\epsilon > 0$, with double-exponential small error probability. Then using the result of [10] we get a polynomial space upper bound for the quantitative approximation problem.

Lemma 9. *For the general* I&R *model with resident reproduction and* LBF*, the following assertions hold: (1) The qualitative decision problem is in PSPACE; and (2) the quantitative approximation problem can be solved in polynomial space.*

Remark 10. *Observe that since precise probabilities to reach a state in a Markov chain can be computed in polynomial time in the size of the Markov chain [7], it follows that the precise fixation probabilities can be computed in exponential time.*

5.2 Lower bound

In this section we present two lower bounds: (i) the qualitative decision question is PSPACE-hard; and (ii) the question that given an evolutionary graph with the promise that the fixation probability is close to either 0 or 1, deciding which is the case is PSPACE-hard (which implies PSPACE-hardness for the quantitative approximation problem). For simplicity, we present our lower bounds in two steps. We will first reduce the problem to a problem which we call *concurrent-if*, and then show that the concurrent-if problem is PSPACE-complete.

Concurrent-if problem. The intuitive description of the concurrent-if problem is as follows: it consists of a set of Boolean variables, and a set of if statements where each conditional is a conjunction of some of the Boolean variables or their negation, and if the conditional is true, then a Boolean variable is set to a truth value. At each step, any of the if-statements can be executed. The process ends either when the first Boolean variable is true or nothing can change (i.e., the conditional of all if-statements are false). Note that the execution can loop, and perhaps run forever. We first define an if-statement.

If-statement. Let $B = \{b_1, b_2, \ldots, b_n\}$ be a set of n Boolean variables. An if-statement s is as follows:

$$\bigwedge(\mathsf{cn}_1, \ldots, \mathsf{cn}_k) \Rightarrow b_i := \text{val},$$

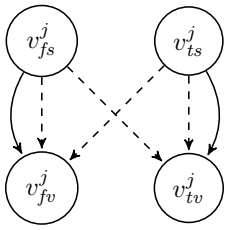

Figure 7: Boolean-value gadget: Dashed edges are in E_I and non-dashed are in E_R.

where $1 \leq i, k \leq n$, val is either true or false, each cn_j is either a Boolean variable b_ℓ or its negation $\overline{b}_\ell$. An if-statement is *satisfied* if each of the cn_j is true (i.e., cn_j is true if one of the following holds: if cn_j is b_ℓ and b_ℓ is true, or cn_j is $\overline{b}_\ell$ and b_ℓ is false).

Concurrent-if system. A concurrent-if system consists of a set $B = \{b_1, b_2, \ldots, b_n\}$ of n Boolean variables and a set $P = \{s_1, s_2, \ldots, s_m\}$ of m if-statements over the Boolean variables in B. The set of statements defines an execution from an initial setting of the Boolean variables as follows: repeatedly, a satisfied if-statement $\bigwedge(\mathsf{cn}_1, \ldots, \mathsf{cn}_k) \Rightarrow b_i :=$ val is selected and then b_i is set to val. If the first Boolean variable b_1 is eventually true, then the execution is *accepted*. If at each point of an execution there is at most one satisfied if-statement, then we say that the execution is *deterministic*.

The decision problem. Given a concurrent-if system, the associated decision problem is as follows: Given a set B of Boolean variables, an initial setting of the variables of B, and a set P of concurrent if-statements, such that the execution e from the initial setting is deterministic, whether e is accepting.

5.2.1 Reduction of the concurrent-if problem to evolutionary games on graphs

We first describe how we encode the Boolean variables and the if-statements of a concurrent-if system in evolutionary games on graphs. Later we show how to construct a graph such that if the fixation of the mutant happens, then the fixation happens according to three phases which are as follows.

1. First phase: The *setup* phase (to initialize the Boolean variables).
2. Second phase: The *execution of the concurrent-if system*.
3. Third phase: The *fixation* phase.

The fixation can only happen if the execution of the concurrent-if system accepts.

Density constraint. Again our lower bound result will be for a special case of LBF, where we have constant fitness with density constraints (recall Remark 1). Our construction will be for $\theta_R = \theta_M = 0$, but a similar construction will work for any choice of $\theta_R, \theta_M \in [0, 1)$. The thresholds $\theta_R = \theta_M = 0$ indicates that a vertex v can reproduce precisely as long as all its successors in E_I are of the opposite type of v, because of the density constraint.

Ideas and gadgets behind the reduction. We first introduce some key ideas and gadgets behind the reduction.

- *States which are nearly always a mutant/resident:* Similar to the previous lower bounds, we have a vertex v_s without any predecessor in E_R. Thus, if v_s is not made a mutant at the start, then it cannot become a mutant. Hence we will only consider the case when v_s is a mutant in the beginning and stays a mutant forever. We will also have a vertex $\widehat{v}_s$, and our construction will ensure that it stays a resident until all other vertices are mutants and then (after a few more steps) all vertices become mutants, and we get fixation. We will use the vertices v_s and $\widehat{v}_s$ to ensure that a given vertex has a desired type, and otherwise the vertex cannot reproduce. Our construction will ensure (using the density constraint) the following properties:
  - A vertex v with $\widehat{v}_s$ as a successor under E_I can only reproduce if it is a mutant (using the density constraint and $\widehat{v}_s$ is a resident). Similarly, a vertex v with v_s as a successor under E_I can only reproduce if it is a resident.

- *Boolean-value gadgets:* We describe how to implement *boolean-value* gadgets in evolutionary graphs for the Boolean variables of the concurrent-if system. Each boolean-value gadget j consist of four vertices v^j_{tv} (the *true-value-vertex*), v^j_{fv} (the *false-value-vertex*), v^j_{ts} (the *true-setter-vertex*) and v^j_{fs} (the *false-setter-vertex*). In the second phase (the execution of the concurrent-if system phase) each boolean-value gadget is such that the two setters, v^j_{ts} and v^j_{fs}, are mutants. Also, at most one of the value vertices v^j_{tv} and v^j_{fv}, can be a mutant at any given point. If v^j_{tv} is a mutant, then the value of j is *true*. If v^j_{fv} is a mutant, then the value of j is *false*. If neither is a mutant, then we say that j has *no value*. The edge set is as follows: (i) both v^j_{ts} and v^j_{fs} have $\widehat{v}_s, v^j_{tv}, v^j_{fv}$ as successors under E_I; (ii) v^j_{ts} (resp., v^j_{fs}) has only v^j_{tv} (resp., v^j_{fv}) as a successor under E_R (see Figure 7). The purpose of the edges in E_I are as follows: the edge to $\widehat{v}_s$ enforces that the setter vertex is a mutant before reproduction; and the other two edges enforce that only if the gadget has no value (i.e., both value vertices are resident), then the setter vertex can reproduce a mutant (by the density constraint and that $\theta_R = \theta_M = 0$). Observe that when the gadget has no value, then each of the setter vertices can set the value of the gadget to either true or false with positive probability in any such step.

- *If-statement gadgets:* Each if-statement gadget, for the if-statement $\bigwedge(\mathsf{cn}_1, \ldots, \mathsf{cn}_k) \Rightarrow b_i :=$ val, is implemented using a single vertex v (the if-statement-vertex). The if-statement gadget works under the requirement that v is a resident, and our construction will ensure that in the second phase (the execution of the concurrent-if system phase) each if-statement-vertex v is a resident. The edge set is as follows:

  1. The vertex v has the following edges in E_i: an edge to v_s; and for each Boolean variable j in $(\mathsf{cn}_1, \ldots, \mathsf{cn}_k)$ an edge to v^j_{tv}, and for each negation of a Boolean variable j' in $(\mathsf{cn}_1, \ldots, \mathsf{cn}_k)$ an edge to $v^{j'}_{fv}$.
  2. The vertex v has v^i_{fv} (resp., v^i_{tv}) as successor under E_R if val is true (resp., false).

  The purpose of the edges in E_I are as follows: the edge to v_s enforces that the if-statement-vertex is a resident before v reproduces; the other edges enforces that each literal in $(\mathsf{cn}_1, \ldots, \mathsf{cn}_k)$ has the correct value before reproduction. Consider the case where val is true (the case where it is false is similar). If v can reproduce at a given point in time, then $\bigwedge(\mathsf{cn}_1, \ldots, \mathsf{cn}_k)$ must be true. In that case, if the boolean-value-gadget for b_i has value false, then v reproduces to set b_i to no value. This then allows the setter-vertices of b_i to reproduce, and set b_i eventually to a value again. Observe that even though v tries to set b_i to true, the value of b_i might not be set to true immediately. The process is as follows: v tries to set b_i to true by ensuring that if it is false, then it sets it to no value, and ensures that the true-setter vertex has positive probability to set it to true. Hence eventually with probability 1 it is set to true. Note that given $\bigwedge(\mathsf{cn}_1, \ldots, \mathsf{cn}_k)$ is still true, v can simply reproduce until b_i becomes true. Since there is a fixed positive probability that the setter-vertices will set b_i to either value, eventually b_i becomes true with probability 1. We will only use the boolean-value gadgets for deterministic executions and thus, the condition $\bigwedge(\mathsf{cn}_1, \ldots, \mathsf{cn}_k)$ remains true until b_i becomes true. This is because the execution is deterministic and thus, no other if-statement is satisfied in the current situation as long as b_i is false or has no value. Especially, for the next if-statement to be satisfied it must depend on b_i being true.

We now describe how to construct the graph such that fixation implies the existence of an accepting execution of the concurrent-if system. First we will describe the vertices that reproduces in the setup phase, then how to use the boolean-value gadgets and the if-statement gadgets to encode some execution of the concurrent-if system, and finally how to ensure fixation if the execution is accepted.

The first phase: Setup. First, as mentioned, we consider the case where v_s becomes a mutant at the start (as otherwise fixation does not happen). The setup phase is split into two parts. The first part ensures that each boolean-value-gadget of the concurrent-if system has the right initial value, and the second part ensures that all setter-vertices of the boolean-value-gadgets are mutants. Each part corresponds to a boolean-value-gadget, s_1, s_2, respectively, and starts when the false-setter vertex, v_1, v_2, respectively, of the corresponding gadget becomes a mutant, and ends when the gadget has true value. The vertex v_s has only $\widehat{v}_s$ as successor in E_I (ensuring that it is a mutant when it reproduces) and v_1 as a successor in E_R. Hence, eventually v_1 becomes a mutant and this starts the first part of the setup phase.

- *The start of the first part of setup:* The vertex v_1 has only $\widehat{v}_s$ as successor in E_I (ensuring that it is a mutant when it reproduces), but has the following successors under E_R: The setter vertices of s_1 and the value vertices

corresponding to the initial value of each Boolean variable of the concurrent-if system. Hence, eventually all vertices which are successor of v_1 in E_R become mutants as well.

- *The end of the first part of setup:* There is an if-statement vertex $\widehat{v}_1$, (which, since it has not become a mutant yet, must be a resident), who has v_1 and all states which are successors of v_1 in E_R as successors under E_I; and v_1 as the lone successor in E_R. This vertex then can first reproduce once v_1 is done reproducing and then eventually sets s_1 to true. This completes the first part.

- *Between first and second part:* The true-value-vertex $v_{tv}^{s_1}$ of s_1 has only $\widehat{v}_s$ as successor in E_I; and v_1, v_2, and $\widehat{v}_1$ as successor in E_R. Hence, after s_1 has become true, each of the vertices v_1, v_2 and $\widehat{v}_1$ becomes mutants.

- *The start of the second part of setup:* The vertex v_2 has only $\widehat{v}_s$ as successor in E_I (ensuring that it is a mutant when it reproduces), but each setter vertex of s_2 and the boolean-value gadgets used in the concurrent-if system as successors in E_R. Hence, eventually, every successor of v_2 under E_R becomes a mutant.

- *The end of the second part of setup:* Similarly to the end of the first part of setup, there is an if-statement vertex $\widehat{v}_2$, (which, since it has not become a mutant yet, must be a resident), who has v_2 and all states which are successors of v_2 in E_R as successors under E_I; and v_2 as the lone successor in E_R. This vertex then can first reproduce once v_2 is done reproducing and then eventually sets s_2 to true. This completes the second part of setup.

- *The end of setup:* The true-value-vertex of s_2 has $\widehat{v}_s$ as successor in E_I (ensuring that it can only reproduce if it is a mutant) and v_2 and $\widehat{v}_2$ as successors in E_R. They then eventually become mutants.

The second phase: Execution of the concurrent-if system. We extend the construction of the graph for the concurrent-if system slightly as follows: Each if-statement-vertex v has some additional successors in E_I, besides the ones described in construction: The vertex v_s (ensuring that v is a resident before reproduction), the true-value-vertex of s_2 (ensuring that the setup phase is complete before v reproduces), and the false-value-vertex of the boolean-value-gadget for the first Boolean variable b_1 (ensuring that the second phase is not over). Hence, this ensures that the if-statement vertices are only active in the second phase. Clearly, if the execution is accepting, then the Boolean variable b_1 is eventually true.

The third phase: Fixation. The fixation part uses two special vertices v_e^1 and v_e^2 (that have not been used before). If the Boolean variable b_1 is true, then fixation will be achieved in the following steps as follows:

- (step 1): the true-value vertex of the boolean-value-gadget for b_1 will reproduce to set v_e^1 to a mutant;

- (step 2): then v_e^1 reproduces to turn all vertices (other than v_s and $\widehat{v}_s$), including v_e^2, into mutants;

- (step 3): after this step, v_e^1 again becomes a resident (but other than $\widehat{v}_s$ and v_e^1, all vertices are mutants);

- (step 4:) finally, v_e^2 makes $\widehat{v}_s$ a mutant, and at the end the true-value-vertex for b_1 again makes v_e^1 a mutant.

We now describe the above steps.

- The true-value-vertex $v' = v_{tv}^{b_1}$ of b_1 has no successor in E_I, and v_e^1 as a successor in E_R. Hence once v' is a mutant, it reproduces to turn v_e^1 into a mutant (step 1).

- The vertex v_e^1 has $\widehat{v}_s$ as successor in E_I (hence can only reproduce if it is a mutant); and each vertex (including v_e^2) *other than* vertex v_s (since no vertex should have v_s as a successor in the replacement graph) and vertex $\widehat{v}_s$ as successors in E_R. All the successors of v_e^1 in E_R becomes mutants (observe that the if-statement vertices might try to make the value-vertices of the boolean-value gadgets into residents, but sooner or later v_e^1 will have made all the if-statements vertices into mutants). This is step 2 above.

- After this only $\widehat{v}_s$ is a resident. The vertex $\widehat{v}_s$ has all other states as successors in E_I (ensuring that it can only reproduce at this point), and vertex v_e^1 as a successor in E_R. Thus now $\widehat{v}_s$ turns v_e^1 into a resident. Note that at this point other than v_e^1 and $\widehat{v}_s$ all vertices are mutants. This is step 3 above.

- The vertex v_e^2 has v_e^1 as successor in E_I and $\widehat{v}_s$ as successor in E_R. Notice that when v_e^1 has been made a resident by $\widehat{v}_s$ both v' and v_e^2 can reproduce. Whenever v' does so, we are back to the situation before $\widehat{v}_s$ reproduced, which then reproduces again. Hence, sooner or later v_e^2 reproduces making $\widehat{v}_s$ a mutant before v' and then v' reproduce after that making v_e^1 a mutant. At this point all vertices are mutants and fixation is achieved. This is step 4 above.

Hence it follows that if the first mutant is v_s, then (i) if the execution is accepting, then fixation happens with probability 1, and (ii) if the execution is not accepting, then the fixation probability is 0. Also note that if the first mutant is not v_s, then the fixation probability 0 because no vertex has v_s as successor in E_R.

Lemma 11. *Given a concurrent-if system that is deterministic we can construct in polynomial time an evolutionary game graph in the* I&R *model with resident reproduction and* LBF *such that (i) if the concurrent-if system accepts, then the fixation probability is positive; and (ii) if the concurrent-if system does not accept, then the fixation probability is 0.*

Fixation amplification. In the construction described above, the fixation happens only if v_s is the first mutant and the concurrent-if system executes an accepting run. However, the probability that the first mutant is v_s is $\frac{1}{n}$ as the first mutant is selected uniformly at random, where n is the number of vertices. We now present a construction that amplifies the fixation probability.

Modified construction. Consider a set S of new vertices. Each vertex in S has $\widehat{v}_s$ as the only successor in E_I and v_s as the only successor in E_R. The vertex v_e^1 has also the vertices of S as successors in E_R. (Also, the vertex $\widehat{v}_s$ still has all other vertices as successors.)

Correctness argument. Observe that if a vertex of S becomes the first mutant, then v_s becomes the next mutant, and then fixation happens if and only if the concurrent-if system accepts similar to before. Hence, we get the following statement.

- If the execution is accepting, then the fixation probability is at least $\frac{|S|}{|S|+n}$ (i.e., the probability that any of the vertices in S is the first mutant).
- If the execution is not accepting, then the fixation probability is at most $\frac{n}{|S|+n}$ (i.e., the probability that none of the vertices in S is the first mutant).

By making S much larger than the remaining graph (e.g., $|S| = n^2$ or $|S| = n^3$), the fixation probability is close to 1, if the execution is accepting, and close to 0 otherwise.

Lemma 12. *Given a concurrent-if system that is deterministic we can construct in polynomial time an evolutionary game graph in the* I&R *model with resident reproduction and* LBF *such that (i) if the concurrent-if system accepts, then the fixation probability is close to 1; and (ii) if the concurrent-if system does not accept, then the fixation probability is close to 0.*

5.2.2 Concurrent-if can simulate a deterministic space-bounded Turing machine

In this section we show that the concurrent-if problem is PSPACE-complete.

The PSPACE upper bound. The upper bound is straightforward and the argument is as follows. A PSPACE algorithm uses memory (or tape cells of the Turing machine) to store the Boolean variables, and then repeatedly execute in every round the following steps: (a) check if b_1 is true, and if so then accept; (b) check if more than 2^n rounds has been executed (by keeping a counter of $n + 1$ bits and incrementing at the end of every round), in which case the system must be cycling (by the pigeonhole principle), and we can reject; (c) find a satisfied if-statement (along with checking that there is exactly one, and otherwise reject), and update the Boolean variable according to the if-statement.

The PSPACE lower bound. We show that the concurrent-if problem is PSPACE-hard by showing that concurrent-if systems can simulate polynomial space-bounded deterministic Turing machines. Our simulation will be such that (a) if the Turing machine rejects the input or exceeds the space bound, then the execution stops; and (b) if the Turing machine accepts, then the special boolean b_1 becomes true; and (c) if the Turing machine loops, then the execution

loops. Also, each step of the Turing machine corresponds to between two and three iterations of the concurrent if-system (three in case when the space should be updated, two otherwise). For the remainder of this section, fix some deterministic Turing machine M, an input I for M, and a polynomial bound N on space[5]. We will next describe the concurrent-if system simulating M on I.

Notations. Every tape cell $i \in \{0, 1, \ldots, N\}$ of the Turing machine is a bit (i.e., either 0 or 1). A configuration of the Turing machine consists of the valuation of every tape cell, the current state of the Turing machine, and the current head position $0 \leq p \leq N$ of the Turing machine. A tape-configuration (v, p, c) consists of the current state v of the Turing machine, the current head position p, and the content c of the tape cell at the current head position. Note that the number of tape-configurations is polynomial given the input.

The Boolean variables. To describe the encoding of the Turing machine as the concurrent-if system, we first describe the Boolean variables and the encoding.

- *Product Turing machine:* We will use three copies M^3 of the given Turing machine M and modify it such that each move takes the current state to the next Turing machine (starting over when the third is reached). More precisely, if the current state of M forms the sequence $(v_i)_i$, then the current state of M^3 forms the sequence $((v_i, i \mod 3))_i$. This allows us to achieve the following: given two adjacent states of the sequence to distinguish which is the first.

- *Tape encoding:* We will use a *tape-boolean* t_i for the i-th bit of the tape. We will have such a Boolean variable for $i \in \{0, \ldots, N+1\}$ (note that we have additional Boolean variables 0 and $N+1$ so that we can check if the space usage has been exceeded in each direction).

- *Configuration:* We will use a *configuration-boolean* $b(v, p, c)$ for each possible current tape configuration (v, p, c) of the Turing machine defined as follows

  - The current state v of the Turing machine.
  - The current position p of the tape head (i.e., p has a value in $\{0, \ldots, N+1\}$, in other words, there are configuration-booleans also corresponding to being just outside the space bound).
  - The content of the tape c (either 1 or 0) under the tape head as the Turing machine entered the current state and position.

  In other words, we have a single boolean for simultaneously being in state v, position p, and the content of the tape head being c as the tape head was moved to the current position.

Initialization. Initially, the tape-boolean t_i is true iff the i-th bit of the input I is true. Also, the only true configuration-boolean is the one for being in the start state, at the start position, and having t_1 as the content of the tape.

The if-statements. Observe that the current tape-configuration (v_1, p_1, c_1), for $p_1 \in \{1, \ldots, N\}$ of the Turing machine, and the current content of the tape cells, uniquely defines the next tape-configuration (v_2, p_2, c_2). Simulating a move of the Turing machine is split into three iterations of the concurrent-if system, namely,

1. *update space*;
2. *tape-configuration super-position*; and
3. *resolve super-position*.

In the first step, the tape-boolean t_{p_1} is possibly updated. In the second step, either $b(v_2, p_2, \text{true})$ or $b(v_2, p_2, \text{false})$ is set to true[6]. In the third step, the configuration-boolean $b(v_1, p_1, c_1)$ is set to false.

[5]For readers not familiar with computer science, we point out that the problem we consider is similar to the halting problem for Turing machines which is undecidable, however, here we have the restriction that the Turing machine must operate with a polynomial space bound N, which makes the problem PSPACE-complete

[6]Observe that after doing so the configuration-boolean $b(v_1, p_1, c_1)$ is true and at least one of $b(v_2, p_2, \text{true})$ or $b(v_2, p_2, \text{false})$ is also true. This represents being in two tape-configurations at once, which we refer as *super-position*.

- **Update space.** If the current tape-configuration (v_1, p_1, c_1) updates the tape cell at position p_1 from true to false (resp., from false to true) before moving, then there is an if-statement that checks as the conditional that the configuration-boolean variable $b(v_1, p_1, c_1)$ is true, all other configuration-booleans are false, and the tape-boolean variable t_{p_1} is true (resp., false). If the conditional is true, then as its assignment it sets the tape-boolean variable t_{p_1} to false (resp., true).

- **Tape-configuration super-position.** For the second step there are two if-statements. One (resp., the other) if-statement checks as its conditional that the configuration-boolean variable $b(v_1, p_1, c_1)$ is true, all other configuration-boolean variables are false (i.e., the current configuration is $b(v_1, p_1, c_1)$), the tape-boolean variable t_{p_1} has been updated, and the tape-boolean variable t_{p_2} is true (resp., false). If this is the case, then it sets $b(v_2, p_2, \text{true})$ (resp., $b(v_2, p_2, \text{false})$) to true.

- **Resolve super-position.** For the third step there are again two if-statements. One (resp., the other) if-statement checks that the configuration-boolean variables $b(v_1, p_1, c_1)$ and $b(v_2, p_2, \text{true})$ (resp., $b(v_2, p_2, \text{false})$) are true, and all other configuration-boolean variables are false. That is, intuitively speaking, the current configuration is the super position of (v_1, p_1, c_1) with either (v_2, p_2, true) or (v_2, p_2, false). In that case it sets the configuration-boolean variable $b(v_1, p_1, c_1)$ to false.

Besides the above if-statements, we also have some additional if-statements. They are as follows: If the current state of the Turing machine is accepting, then set b_1 to true (formally: for each accepting state v and each $p \in \{1, \ldots, N\}$ and $c \in \{\text{true}, \text{false}\}$, there is an if-statement that checks whether $b(v, p, c)$ is true, and all other configuration-boolean variables are false, and if so, then sets b_1 to true). Furthermore there are no if-statements for configurations (v_1, p_1, c_1) where $p_1 \in \{0, 1, \ldots, N+1\}$ or where v_1 is rejecting, ensuring that the execution ends in that case (and it is especially not accepting). Note that the reduction is polynomial time since we use constantly many if-statements for every tape-configuration.

Deterministic. We now argue that for the concurrent-if system we obtain in the reduction, the execution from the initial setting is deterministic. The reasoning is as follows: (1) If there are at least three configuration-boolean variables that are true, then no if-statement can be satisfied. (2) We now consider the case that two configuration-boolean variables, (v_1, p_1, c_1) and (v_2, p_2, c_2), are true. Note that in this case the if-statements for update-space, and tape-configuration super-positions cannot be satisfied. Now we consider if-statements for resolve super-position case, which checks for the truth of two configuration-boolean variables, such that the tape-configurations can be consecutive. Since we consider M^3, at most one of the tape-configurations can immediately precede the other. Hence at most one if-statement can be satisfied. (3) Finally, we consider when one configuration-boolean variable is true. In this case, precisely one of an if-statement from update-space or tape-configuration super-position can be true, depending on the content of the current and next position of the tape head. From the above case analysis, it follows that the concurrent-if system is deterministic.

Lemma 13. *The problem of deciding whether a concurrent-if system, that is deterministic, accepts is PSPACE-complete.*

Lemma 13 along with Lemma 11 and Lemma 12 gives us the following result.

Lemma 14. *For the general* I&R *model with resident reproduction and* LBF*, the following assertions hold: (1) The qualitative decision problem is PSPACE-hard; and (2) given the promise that the fixation probability is close to either 0 or 1, deciding which is the case is PSPACE-hard.*

The previous lemma and Lemma 9 gives Theorem 15 which summarizes the result of this section.

Theorem 15. *For the general* I&R *model with resident reproduction and* LBF*, the following assertions hold: (1) The qualitative decision problem is PSPACE-complete; and (2) the quantitative approximation problem can be solved in polynomial space, and even given the promise that the fixation probability is close to either 0 or 1, deciding which is the case is PSPACE-hard (hence the quantitative approximation problem is PSPACE-hard).*

6 Complexity Results for the Exponential Fitness Function ExF

In this section we consider the case where the fitness of an individual at a vertex is an exponential function of the payoff, i.e., the fitness of an individual at vertex v is

$$f(v) = \exp\left(\frac{\sum_{u \in E_I(v)} \mathsf{pay}(v,u)}{|E_I(v)|}\right),$$

and we do not have density constraint. We first present the results, and describe how to obtain them.

1. *Result 1.* The qualitative problem can be solved in polynomial time.
2. *Result 2.* For the no resident reproduction case (i.e., when the fitness of a resident is set to 0), the quantitative approximation problem can be solved in polynomial time.
3. *Result 3.* For the resident reproduction case, we have the same complexity bounds as in the case where we have the LBF.

Result 1. Since the fitness is an exponential function and always positive, the fixation probability is positive if the replacement graph is connected, and otherwise the fixation probability is 0. Since whether a graph is connected or not can be checked in polynomial time, it follows that the qualitative problem can be solved in polynomial time.

Result 2. In case the resident does not reproduce, and the fitness of the mutant is always positive due to an exponential function of the payoff, then for every vertex a mutant at the vertex reproduces to turn all its successor in the replacement graph as mutants. Hence given a vertex v, if the mutant arises at v, then all vertices that are reachable from v in the replacement graph become mutants with probability 1. Given a vertex v, we say that $v \in \mathsf{All}$, iff all vertices in the graph are reachable from v in the replacement graph. Then the fixation probability is $|\mathsf{All}|/|V|$, i.e., the probability that the mutant arises initially in any one of the vertices in All. Since reachability can be computed in polynomial time, it follows that the exact fixation probability (and hence also the approximation) can be computed in polynomial time.

Theorem 16. *For the general* I&R *model with the fitness function is exponential function of the payoff, the following assertions hold:*

1. *The qualitative problem can be solved in polynomial time.*
2. *For no resident reproduction, the exact fixation probability (hence also the quantitative approximation problem) can be computed in polynomial time.*

Result 3. Given Theorem 16 the only remaining problem is to consider the problem of quantitative approximation for the model with resident reproduction. We prove that the complexity results of Section 5 also hold when the fitness is an exponential function of the fitness.

The PSPACE upper bound. The same argument for the PSPACE upper bound of Section 5.1 also holds when the entries of the payoff matrix are polynomial in the input, as an exponential size Markov chain can be constructed (the probabilities are obtained using the exponential fitness function), and hence we obtain the PSPACE upper bound.

The PSPACE lower bound. The rest of the section is devoted to show how to modify the PSPACE lower bound of Section 5.2 to show that the lower bound is also applicable to the case where the fitness is exponential of the payoff (and there is no density constraint). Note that Remark 1 shows that density constraints can be encoded in LBF, but it does not show how to encode density constraint in ExF. In the sequel, we use "with high probability" to refer to that the probability is at least $1 - \frac{1}{\mathrm{poly}(n)}$, where n is the size of input, and poly is a polynomial function.

The key idea. The key idea is as follows:

1. *First step:* First, we consider the problem with constant payoff along with density constraints and argue that the PSPACE hardness result holds even in the case where either mutants or residents fixate within an exponential number of steps with high probability.

2. *Second step:* In the hardness proof in the model with density constraints we require that a vertex can reproduce iff all its successors are of the opposite type. In the model with fitness exponential of payoff, there is always a positive probability to reproduce. Thus even if a vertex has all its successors of the opposite type, it can still reproduce, and we refer to such reproductions as "undesired reproductions" (for the hardness proof). We show that a payoff matrix (with exponential payoff and no density constraints) can encode that if a vertex does not have all its successes of the other type, then the probability to reproduce is exponentially small (i.e., the undesired reproduction probability is exponentially small). Since in the hardness result of the previous item, the fixation happens within exponentially many steps, using union bounds it is easy to argue that the probability that an undesired reproduction happens before fixation is negligible. Hence the PSPACE hardness result for the model with density constraints can be translated also to the model with fitness exponential of the payoff and no density constraints. Also in the hardness result we only need to consider that the entries of the payoff matrix are polynomial in the input.

Achieving the first step. Achieving the first step is done using the following: (a) first, modify the concurrent-if problem; and (b) then modify our reduction from concurrent-if problem to evolutionary games on graphs.

The modified concurrent-if problem. We consider a modification of the concurrent-if problem, where given a concurrent-system, we accept an execution if b_1 is set to true, and we reject an execution if b_2 is set to true. We consider concurrent-systems that are deterministic, along with the promise, that within an exponential number of steps either b_1 is set to true or b_2 is set to true. We refer to a system of the above form as modified concurrent-if system. The decision problem, given a modified concurrent-if system whether it accepts or rejects, is PSPACE-complete. The argument for the lower bound is as follows: in our original PSPACE-hardness reduction in Section 5.2, we consider a space-bounded Turing machine M, with input I, and space bound N which is polynomial. Instead of M we can consider another Turing machine M' that simulates M, and keeps in a counter the number of steps of M that have been executed. If the number of steps exceeds exponential, then M loops, and thus M' can reject (which is modeled in the concurrent-if system by setting b_2 to true). Also M' can reject if the space bound N is exceeded, again modeled by setting b_2 to true. The Turing machine M' can be reduced to a modified concurrent-if problem similar to the reduction of Section 5.2. Hence in our reduction we now consider the modified concurrent-if problem, which either accepts or rejects within exponentially many steps.

Properties to be ensured by the reduction. We will now consider a modified concurrent-if system, and then construct an evolutionary game on graph with the following properties: (a) if a vertex has all its successors in E_I of the opposite type, then it has a constant fitness (say, a constant $c > 0$); (b) if a vertex has at least one successor in E_I of the same type, then it cannot reproduce; (c) if the modified concurrent-if system accepts (resp., rejects), then in the evolutionary graph the mutants (resp., residents) fixate within an exponential number of steps with high probability. Note that the first two properties reiterate that we are considering the model with constant fitness and density constraints. This is the main idea to achieve the first item of the key idea. We now describe the key changes to the reduction of Section 5.2.

Changes to the setup phase. The main change in the setup phase is to construct a copy of v_s. Recall that since the size of S is much larger than the rest of the graph, we only need to consider that the first mutant starts in a vertex in S. In our reduction in Section 5.2, the vertex v_s played two different roles, which are as follows: (i) first, after a vertex in S, it becomes the second vertex to be a mutant, and is responsible for starting the process of reproducing mutants; and (ii) second, given v_s is a mutant, to ensure that a vertex v can reproduce only if it is a resident, we made v_s a successor of v in E_I. In this reduction we create a new vertex v'_s, such that v_s achieves the first role, whereas v'_s plays the second role. The modification is as follows: (i) for all vertices v such that v_s is a successor of v in E_I in the reduction of Section 5.2, then v has v'_s as successor in E_I instead of v_s; (ii) v'_s is a successor of v_s in E_R. Finally, $\widehat{v}_1$ has v'_s as a successor in E_I, ensuring that it becomes a mutant in the first part of setup.

Changes to the fixation phase. We first consider how to ensure fixation for residents if b_2 is set to true, and then describe the changes to ensure fixation for mutants if b_1 is set to true. We introduce additional nodes, v_e^3, v_e^4, v_e^5, v_e^6, v_M^3, v_M^4 and v_M^5 (note that there are no vertices v_M^1, v_M^2 or v_M^3, but the naming scheme is used since v_M^i is associated with v_e^i).

The subtle issue about the fixation. Our goal is to ensure that if b_2 (resp., b_1) is set to true, then the residents (resp., mutants) fixate. However, we must ensure in the evolutionary graph, that once b_2 (resp., b_1) is set to true, then the

fixation of mutants (resp., residents) does not happen. The fixation of mutants (resp., residents) is triggered by the boolean-value-gadget for b_1 (resp., b_2) being set to true by making $v_{tv}^{b_1}$ (resp., $v_{tv}^{b_2}$) a mutant.

Vertex v_e^3 for fixation of residents. We consider the case when the concurrent-if system has rejected by making $v_{tv}^{b_2}$ a mutant. Then vertex v_e^3 plays a crucial role in ensuring fixation for the residents. The vertex v_e^3 has the vertex v_s' and $v_M^3 = v_{tv}^{b_2}$ as successors in E_I; the edge to v_s' ensures that v_e^3 is a resident before v_e^3 reproduces, and the other edge ensures that the concurrent-if systems has rejected before v_e^3 reproduces. The successors of v_e^3 under E_R are as follows: (1) the vertices of S; (2) the vertex v_s; (3) all the vertices of the boolean-value-gadgets s_1 and s_2 together with $\widehat{v}_1$ and $\widehat{v}_2$; and (4) the setter vertices of all other boolean-value-gadgets. We now argue that once v_M^3 is a mutant, then within polynomially many steps, all successors of v_e^3 under E_R become residents with high probability. The vertices in (1) (i.e., S) are the only ones that can make vertices in (2) (i.e., v_s) mutants, and in general for $2 \leq i \leq 4$, vertices described above in (i) can be made mutants by the vertices in $(i-1)$, except for the vertices in (3) (which can be made mutants by the constant number of vertices of either (2) or (3)). Thus since there is a probability of one over a polynomial that v_e^3 makes a given successor a resident in one step (when it can reproduce), it follows that within a polynomial number of steps all vertices which are successor of v_e^3 under E_R are residents with high probability.

Checking that v_e^3 is done with reproduction. The other vertices and edges are as follows:

- The successors of vertex v_M^3 in E_I are the the successors of v_e^3 in E_R; this ensures that v_M^3 can only reproduce after v_e^3 has made all its successors residents. The successor of v_M^3 under E_R is v_M^4; this ensures that once v_e^3 has made all its successors under E_R residents, then v_M^4 becomes a mutant.

- The vertex v_e^4 has v_M^4 as successor in E_I; and all value vertices of the boolean-value-gadgets, the vertex v_s', and the vertex v_M^3 as successors in E_R. Thus when v_e^3 and v_e^4 has reproduced to make all their successors under E_R residents, then only the vertex v_M^4 can be a mutant.

- The vertex v_M^4 has all other vertices as successors in E_I and v_M^5 as successor in E_R. This ensures that v_M^4 can only reproduce when all other vertices are residents.

- The vertex v_e^5 has v_M^4 and v_M^5 as successors in E_I; and v_M^4 as successor in E_R. Thus the vertex v_e^5 can only reproduce when v_M^4 has turned v_M^5 to a mutant.

- The vertex v_e^6 has v_M^5 as the only successor in both E_I and E_R. This ensures that v_e^6 can only reproduce once v_M^5 has become a mutant.

The construction ensures that after v_M^4 is the only mutant, then it makes v_M^5 a mutant, and then both v_e^5 and v_e^6 can reproduce. If v_e^6 does so first, then we are back to the case when v_M^4 is the only mutant. Otherwise, we have that v_e^5 reproduces and then v_e^6 reproduces to turn all the remaining vertices to residents, and thus we obtain fixation for residents.

Changes for fixation of mutants. To ensure that fixation of the mutant phase cannot trigger the fixation of the resident phase, we divide the fixation of mutants into three parts, using a boolean-value-gadget to describe when each part is over. We present informally the construction (as the construction is similar to the setup phase of the original construction of Section 5.2). We consider the case when the concurrent-if system has accepted by making $v_{tv}^{b_1}$ a mutant.

- In the first part we make the vertices v_e^3 and v_e^6 mutants. Observe that v_e^3 being mutant does not matter, since all its successors in E_R are already mutants. The vertex v_e^6 will make v_M^5 into a mutant, but since only v_e^5 and v_e^6 have v_M^5 as a successor in E_I and v_e^5 will not reproduce since v_M^4 is a resident, this does not trigger fixation of residents. Note that no vertex (other than the boolean-value-gadget for this part) has v_e^3 or v_e^6 as a successor under E_I.

- The second phase makes v_e^4 a mutant, which will cause all the value vertices of the boolean-value-gadgets to become mutants, but cannot trigger fixation of the residents anymore, since v_e^3 is already a mutant.

- The last part is like the fixation of the mutant as in the original construction of Section 5.2, except that it also makes the vertices used in the boolean-value-gadgets for the parts of mutant fixation, the vertex v_e^5, the vertex

v_M^3, the vertex v_M^4, and the vertex v_M^5 into mutants. Observe that if v_e^5 or v_M^4 were mutants earlier than v_e^4, then v_e^4 could conceivably have triggered fixation of residents.

In the above reduction, it is easy to see that setup and fixation each takes a polynomial number of steps in expectation, and the execution of the concurrent-if system takes at most exponential steps in expectation. Let T' be the expected number of steps for fixation, and T' is exponential. Hence using Markov inequality, there exists $T \geq T'$ such that T is also exponential, and the fixation happens within T reproductions with high probability.

The hardness result. The above reduction achieves the following. Given evolutionary games on graphs with the following properties (a) if a vertex has all its successors in E_I of the opposite type, then it has a constant fitness (say, a constant $c > 0$); (b) if a vertex has at least one successor in E_I of the same type, then it cannot reproduce; (c) either the mutants or residents fixate within an exponential number T of steps with high probability; and the promise that the mutants fixate with probability close to either 0 or 1, decide which is the case is PSPACE-complete.

Reduction to fitness as exponential of payoff. We now show how the above reduction is modified to obtain the same PSPACE hardness result, when the fitness is exponential of the payoff and there is no density constraint. There is no modification in the graph, and we only define the payoff matrix

$$\begin{array}{cc} & \begin{array}{cc} R & M \end{array} \\ \begin{array}{c} R \\ M \end{array} & \begin{pmatrix} -x & 1 \\ 1 & -x \end{pmatrix} \end{array}$$

for some x, such that $p = T \cdot N \cdot \exp(1 - x/N)$ is small (to make p exponentially small the number x only needs to be polynomially small), where N is the number of vertices. Recall that T is such that in the previous model we have fixation within T steps with high probability and T is exponential. The idea is that if the payoff of a vertex is negative, then it is at most $1 - \frac{x}{N}$. Consider a given step in the evolutionary graph such that some vertex has non-negative payoff, and fix a vertex v that has negative payoff. The probability to pick v in this step to reproduce is at most $\exp(1 - x/N)$. Hence by union bound, the probability of an undesired reproduction in a given step is at most $N \cdot \exp(1 - x/N)$. On the other hand, by picking vertices with non-negative payoffs, the fixation happens with high probability (say p') in T steps (see property c above). Thus we have the following: (i) the conditional probability of fixation within the first T steps, given that there are no undesired reproduction in first T steps, is at least p'; and (ii) the probability that there are no undesired reproduction within the first T steps is at least $1 - p$. Hence the probability of fixation without any undesired reproductions within the first T steps is at least $p' \cdot (1 - p)$, which is close to 1, since p is small. Hence the fixation probability in the model with the above matrix where the fitness is exponential of the payoff is close to the fixation probability of the previous model. The hardness result follows. The following theorem summarizes the result.

Theorem 17. *For the general* I&R *model with the fitness function is exponential function of the payoff, where each payoff of the matrix is polynomial in the size of the graph, the following assertion holds: the quantitative approximation problem can be solved in polynomial space, and even given the promise that the fixation probability is close to either 0 or 1, deciding which is the case is PSPACE-hard (hence the quantitative approximation problem is PSPACE-hard).*

7 Solutions in Polynomial Time

The molecular clock problem. We consider the problem of molecular clock of neutral evolution. It was shown in [1] that the molecular clock can be accelerated or slowed due to spatial structure, which is represented as a graph. The problem is as follows: given a graph that represents a population structure, over time, the population acquies neutral genetic substituitions due random drift. Reproductions happen asexually, might depend on the precise vertex of the graph. For each vertex, there is a given probability that a neutral mutation arises. The molecular clock, denoted as K, is the average number of fixations of the mutations per generation. The basic computation question is to compute the value of K.

The molecular clock solution. The solution of the molecular clock problem is as follows: the fixation probabilities in the neutral case can be obtained as a unique solution to a set of linear equalities. A system of linear equalities can be solved in polynomial-time, for example, using Gaussian elimination (which is cubic time) [4]. The value of the

molecular clock K is characterized by a simple linear expression in the fixation probabilities [1]. Given the fixation probabilities, the linear expression can be evaluated in polynomial time.

Evolutionary games on well-mixed population. We consider the problem of evolutionary games on well-mixed population, i.e., both the interaction and replacement graphs are complete graphs. The basic computational problem is to compute the fixation probability. The problem we consider is actually a special case of the computational problems we have considered, and the special case is that the graphs are complete.

Well-mixed population solution. We show that the fixation probability can be computed in polynomial time for both LBF and ExF. Note that in a given generation, with a given number of mutants, the probability of the following events are defined, independent of the precise locations of the mutants and the generation number: the number of mutants (i) increase by 1; (ii) decrease by 1; (iii) remains the same; for the next generation. In other words, the distribution of the number of mutants in the next generation, depends only on the number of mutants of the present generation and the payoff matrix (in both the LBF and ExF model for fitness). Hence, we can construct a linear Markov chain, where each vertex is a single number i, which corresponds to having i mutants and $n-i$ residents, where there are n vertices. For each $0 \leq i \leq n$, the transition probabilities from i to i itself, $i-1$ and $i+1$ can be computed in polynomial time given the payoff matrix. The fixation probability, is the probability to eventually reach vertex n starting at vertex 1. Since the eventual reachability probability in Markov chains can be computed in polynomial time (again solving a set of linear equalities [4]), it follows that the fixation probabilities for well-mixed population can be computed in polynomial time.

8 Conclusion and Open Problems

In this work we studied the complexity of basic computation questions for related to ecology and evolution on graphs. We established many lower and upper bounds for complexity, and our most interesting and significant results are the lower bounds. We have established NP-hardness, #P-hardness, and PSPACE-hardness for several problems, and it implies that polynomial-time algorithms for any of the problems would imply P is equal to NP, solving a long-standing open problem. Moreover, under the widely believed conjecture that P is different from NP, it follows that for the problems for which we establish lower bounds, there exists no efficient algorithm. A simple equation based solution implies an efficient algorithm. The significance of our result is that it shows that for several fundamental problems in ecology and evolutionary games on graphs, in general there is no simple equation based solution for the fixation probability (in other words, a simple equation based solution would imply that P=NP).

There are several interesting open problems. First, is the problem of evolutionary games on graphs with constant fitness function, which is a special case of the general problem. In this case, the qualitative problem can be solved in polynomial time, and while the quantitative problem can be solved in PSPACE, a non-trivial lower bound for the problem is an open question. Second, another interesting direction would be to explore the computational question in the regime of weak selection.

Acknowledgments. We thank Erez Lieberman and Michal Koucky for insightful discussions and sharing their thoughts on the problem.

References

[1] B. Allen, C. Sample, Y. Dementieva, R. C. Medeiros, C. Paoletti, and M. A. Nowak. The molecular clock of neutral evolution can be accelerated or slowed by asymmetric spatial structure. *PLoS Comput Biol*, 11(2):e1004108, 02 2015.

[2] N. Barton, D. E. Briggs, J. A. Eisen, D. B. Goldstein, and N. H. Patel. *Evolution*. Cold Spring Harbor Laboratory Press, 2007.

[3] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 2009.

[5] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[6] J. E. Hopcroft and R. M. Karp. A $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. In *Proceedings of the 12th Annual Symposium on Switching and Automata Theory (Swat 1971)*, SWAT '71, pages 122–125, Washington, DC, USA, 1971. IEEE Computer Society.

[7] J. Kemeny, J. Snell, and A. Knapp. *Denumerable Markov Chains*. D. Van Nostrand Company, 1966.

[8] L. A. Levin. Universal sequential search problems. *Problems of Information Transmission*, 9(3):265–266, 1973.

[9] E. Lieberman, C. Hauert, and M. A. Nowak. Evolutionary dynamics on graphs. *Nature*, 433(7023):312–316, Jan. 2005.

[10] N. Nisan. RL $\subseteq$ SC. STOC '92, 1992.

[11] H. Ohtsuki, C. Hauert, E. Lieberman, and M. A. Nowak. A simple rule for the evolution of cooperation on graphs and social networks. *Nature*, 441:502–505, 2006.

[12] H. Ohtsuki, J. M. Pacheco, and M. A. Nowak. Evolutionary graph theory: Breaking the symmetry between interaction and replacement. *Journal of Theoretical Biology*, 246(4):681 – 694, 2007.

[13] S. Otto and T. Day. *A Biologist's Guide to Mathematical Modeling in Ecology and Evolution*. Princeton University Press, 2011.

[14] C. M. Papadimitriou. *Computational complexity*. Addison-Wesley, Reading, Massachusetts, 1994.

[15] W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177 – 192, 1970.

[16] L. G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979.